Burroughs

B 7700

INFORMATION PROCESSING SYSTEMS

REFERENCE MANUAL

PRICED ITEM

Printed in U.S. America Feb., 1976 1060233

Burroughs

B 7700
INFORMATION PROCESSING SYSTEMS

REFERENCE MANUAL

AA430046

PRICED ITEM

Printed in U.S. America Feb., 1976 1060233

Burroughs believes that the information described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, is accepted for any consequences arising out of the use of this material. The information contained herein is subject to change. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this document may be addressed directly to Systems Documentation, Technical Information Organization, Burroughs Corporation, P.O. Box 203, Paoli, Pa. 19301

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Cont)

LIST OF TABLES

INTRODUCTION

This system reference manual presents the technical details about the general architecture, the components, and the subsystems of the Burroughs B 7700 Information Processing System, which is the most advanced, the largest, and the most powerful member of the Burroughs family of 700 systems. For a more general coverage of the hardware and software characteristics of the system, refer to the system characteristics manual.

The chapters of this reference manual are as follows:

Chapter I, *Description of the B 7700 System,* introduces the idea of the interaction of independently operating computing, input/output, and memory modules through an exchange and a presentation of the range of configurations of the system.

Chapter II, *System Architecture,* discusses data representation, Polish notation and stack concepts, processor control words, and the concepts of the input/output subsystem map.

Chapter III, *Central Processor Module,* contains a functional description of the operation of the central processor module, an explanation of hardware interrupts, and a brief description of each program operator.

Chapter IV, *Input/Output Subsystem,* contains a general description of the operation of the input/output module, functional descriptions of the subsections of the input/output module, and detailed descriptions of the control words and descriptors associated with each type of peripheral device that may be included in the system.

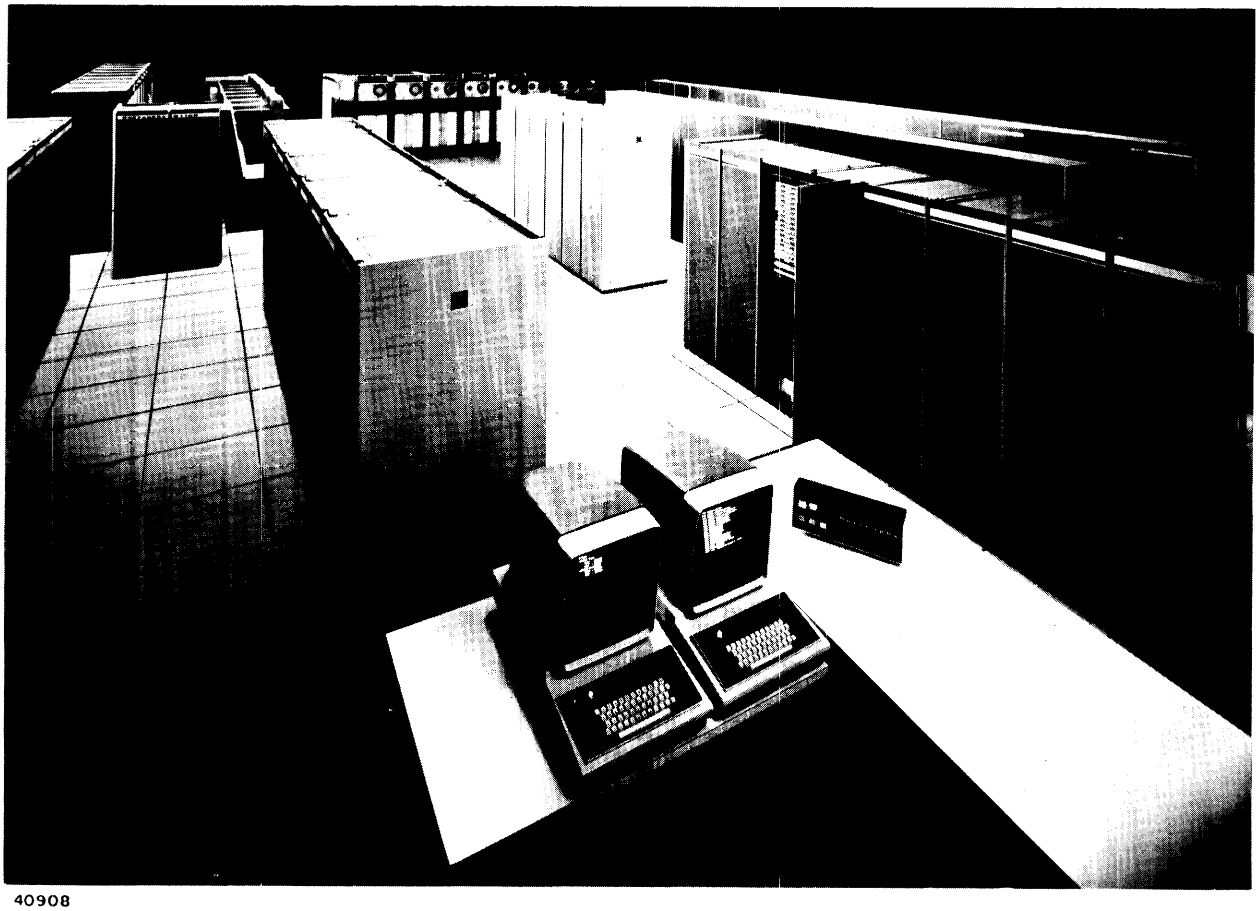
Chapter V, *Memory Subsystem,* a general description of the memory subsystem and details about both the memory control module and the memory storage unit.

Chapter VI, *Controls and Indicators,* contains detailed descriptions of the functions and uses of the controls and indicators of the central components of the system.

The term *"software"*, as used in this manual, applies to that category of Burroughs Program Products defined as *"Systems Software."*

Other categories of Burroughs Program Products are:

Application Program Products
Program Product Conversion Aids



40908

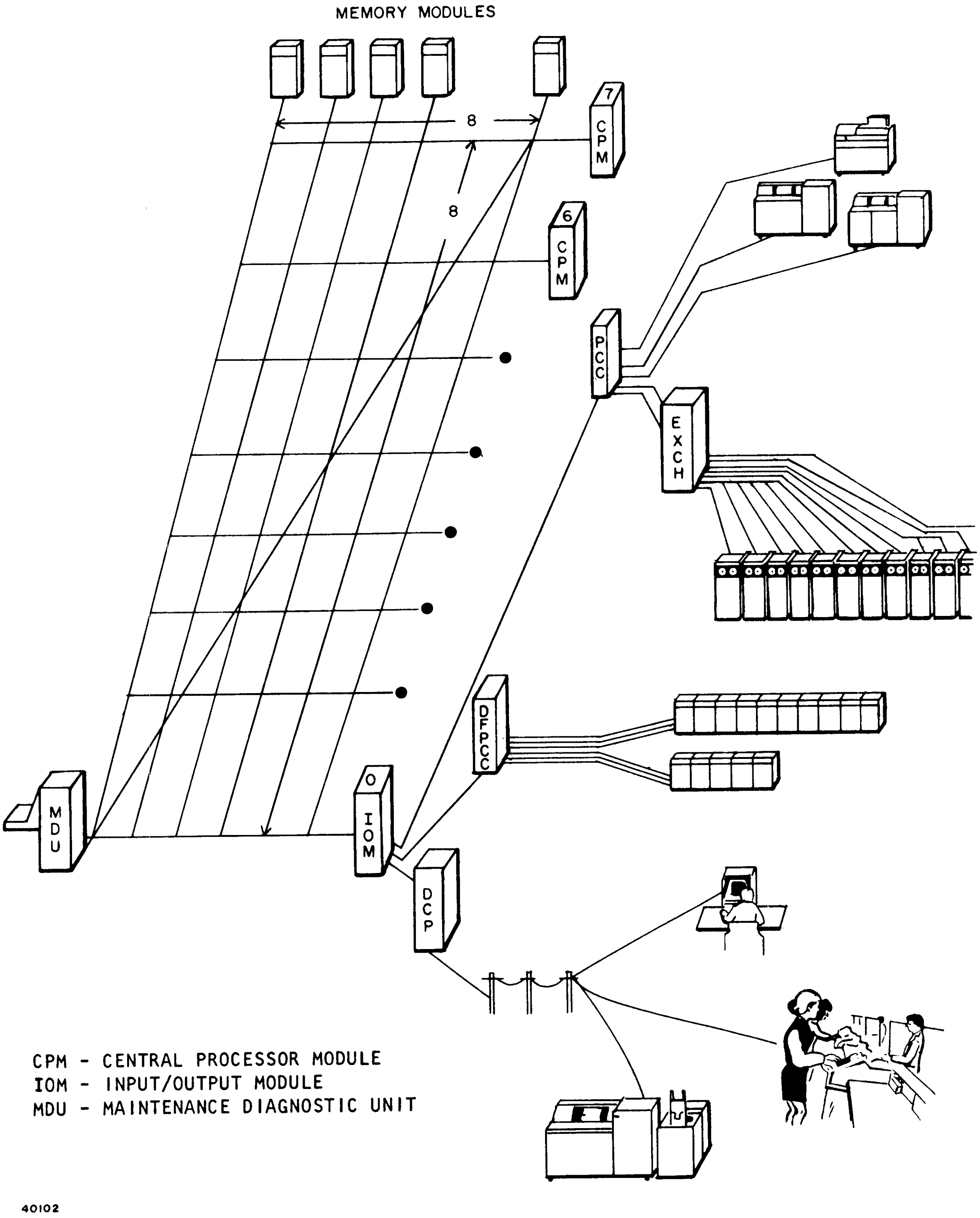

MEMORY MODULES
8
8
7 CPM
6 CPM
PCC
EXCH
DFPCC
O
IOM
MDU
DCP
CPM - CENTRAL PROCESSOR MODULE
IOM - INPUT/OUTPUT MODULE
MDU - MAINTENANCE DIAGNOSTIC UNIT
40102

Figure I-1. B 7700 Exchange

CHAPTER I

DESCRIPTION OF B 7700 SYSTEM

THE B 7700 SYSTEM

The Burroughs B 7700 Information Processing System is a large-scale, truly general-purpose, balanced, flexible, multiprogramming and multiprocessing computing system that is suitable for such diverse applications as time sharing, scientific problem solving, and business data processing. Carrying forward ideas proven successful in the Burroughs B 5700 and B 6700 information processing systems, the B 7700 is, in fact, completely code compatible with the B 6700 and affords Burroughs users the opportunity for growth without reprogramming or recompiling.

In other words, object code users' programs that can be executed successfully on the B 6700 can be executed without modification on the B 7700, and the object code that can be executed on the B 7700 can be executed without modification on the B 6700. Nevertheless, the B 7700 is designed to satisfy the increasingly complex data processing needs of the years to come. The system is able to handle complex data structures and sophisticated program structures dictated both by higher-level languages now in use and by the requirements of advanced problems, is able to manage efficiently the massive on-line and archival storage requirements of large data bases, and is able to accommodate vast networks of data communications devices.

A very fast, modular parallel processing system with exception versatility in configuration, the B 7700 can be tailored to the processing needs of a user by arranging central processor modules, input/output modules, and memory modules on a electronic grid, or exchange (figure I-1), in a variety of ways depending upon the exact needs of the user. If the high performance and adaptability of the B^{7}700 could be attributed to single factor, it would be to the balance attained by means of the controlled interaction of independently operating computing, input/output, and memory modules through the exchange. Thus, the throughput of the system as a whole is maximized, and the performance of no single element of the system is maximized to the neglect or detriment of others.

The key to the efficient balanced use of the system is the Burroughs master control program (MCP)a unique executive software operating system that automatically makes optimum use of all system resources. It is this operating system that makes multiprogramming and multiprocessing both functional and practical by dynamically controlling system resources and by scheduling jobs in the multiprogramming mix. In use, the master control program allocates system resources to meet the needs of the programs introduced into the computer. It continually and automatically reassigns resources, starts jobs, and monitors their performance.

Further implications of the modularity and flexibility of the system are its expandability (a capacity to add hardware modules without reprogramming) and its increased reliability (and thus increased availability to the user. This reliability is achieved by the use of fail-soft techniques that (in addition to providing for error detection and error correction, redundancy of data paths, and independence and redundancy of power supplies) exclude faulty modules from the system and permit processing to continue (again, without reprogramming) even with a temporarily reduced configuration.

Even though it is very large and immensely complicated, the B 7700 is, nevertheless, comprehensible to the persons who use it: programming is done only in higher-level, problem-oriented languages (COBOL, ALGOL, FORTRAN, PL/1, and ESPOL); the control language used in entering jobs into the system is a simple, free-form, English-like language; and the messages that pass between the system and the operator are brief, clear, and easy to learn.

DISTINGUISHING FEATURES

Although the balanced use of the principal components of the system as a whole under the control and coordination of the master control program is the key to the high throughput of the B 7700, the high performance of the system is in large part achieved by improving the speed of execution of instructions, by reducing or masking the overhead associated with references to memory, by freeing the central processor from concern with input/output operations, and by employing fail-soft measures that minimize system degradation.

Because system main-frame hardware has been designed and built strictly according to stringent circuit and wiring rules and proven design and packaging techniques and because its processing elements incorporate monolithic integrated circuits, the B 7700 system performs consistently at high operating frequencies: the central processor module at a clock rate of 16 megahertz and the remainder of the system at 8 megahertz.

By combining the following features with the high internal operating frequencies, the performance of the system is further enhanced.

1. The parallel and independent operation of the three main sections (program, execution, and storage) of the central processor module. This parallelism (coupled with the high clock rate) makes possible the speeding up of arithmetic computations and data manipulations and the overlapping of these computations and manipulations with memory references.

2. A special high-speed integrated circuit memory (program, stack, and associative data buffers). This high-speed local memory permits multiword transfers between the central processor and main memory and makes possible the anticipation of the need for program and data words. Hence, the time spent waiting for the completion of transfers to and from memory is reduced and at times virtually eliminated.

3. The four-way interleaving of addresses in main memory and the capability for phased multiword transfers of information to and from memory in groups of up to four words. Consequently, memory access times for each user of memory are reduced, and memory is thus made more accessible to all users.

4. The asynchronous performance of input/output operations by the input/output module independent of the central processor, which is therefore freed to do other useful work.

The three goals of the fail-soft features of the B 7700 are to keep the system running 100 percent of the time, to minimize system degradation, and to provide the user with tools for performing his own data recovery. These goals are achieved by the artful combination of hardware and software throughout the system.

The first goal, to keep running, is achieved as follows:

1. By the high reliability of system hardware.

2. By the incorporation of error detection circuits throughout the system.

3. By single-bit error correction of errors in memory.

4. By recording errors for software analysis.

5. By modular design, by use of separate power supplies and redundant regulators for each module, and by use of redundant buses.

6. By the ability of the master control program to reconfigure the modules of the system to exclude temporarily a faulty module.

7. By automatic instruction retry, if a hardware malfunction occurs during the performance of an instruction, the master control program analyzes the error and writes the appropriate entry in the on-line maintenance log. The processor is reset to its state prior to the error and the instruction is performed again.

In short, the detection and reporting of errors is done by hardware, analysis of errors is done by software, and the reconfiguration of the system is done dynamically by software. Because of the modularity of power supplies and the use of redundant regulated supplies for critical voltages, the impact of a malfunctioning dc supply is minimized and does not result in a catastrophic failure.

The second goal, to minimize system degradation, is achieved by providing diagnostic programs and equipment for rapidly identifying and repairing faults and for reestablishing confidence in a repaired module before it is returned to the user's system. The diagnostic programs of the B 7700 system identify a faulty module. By the use of the maintenance diagnostic unit, a fault in any main-frame module or in a disk file optimizer is narrowed to a single clock period and to a flip-flop and its associated the central processor module, the input/output the card tester on the maintenance diagnostic unit, the faulty integrated circuit chip is identified.

The third goal, to provide the user with tools for performing his own data recovery, is achieved by the use of such features as installation allocated disk, protected disk files, duplicated disk files, and fault statements in the higher-level programming languages used on the system.

Installation allocated disk allows the user to specify the physical allocation of his critical disk files to facilitate the maintenance and reconstruction of these files. Protected disk files allow a user to gain access to the last portion of valid data written in a file before an unexpected system halt. The use of duplicated disk files is to avoid the problem of fatal disk file errors. The master control program maintains more than one copy of each disk file row, and, if access cannot be gained to a record, an attempt is made to gain access to a copy of the record. By use of fault statements, the user can stipulate actions to be taken by his programs in the event errors occur.

SYSTEM CONFIGURATION

Physically, the components of the B 7700 system fall into three categories, as follows:

1. Central components of the B 7700 system: the central processor module, input/output

output module, the memory module, the maintenance diagnostic unit, and the operator's console (refer to table I-1).

2. Standard Burroughs cabinets that contain peripheral controls and exchanges, the disk file optimizer, the data communications processor, and ac power supplies.

3. Standard peripheral devices that are joined to the central system by means of standard Burroughs peripheral controls, adapters, and exchanges and standard remote devices that are joined to the central system by means of line adapters and data communications processor.

The arrangement of these components into a system and the size of the system depend on the application and workload of the user. In the following paragraphs, the range of configurations of the B 7700, the maximum configuration and the minimum configuration, is described.

Table I-1. Central Components of the B 7700 System

Style No.	Description
B 7750	System includes: one central processor (16 MHz with vectors), one input/output processor with 24 data switching channels, one maintenance diagnostic unit, one operator console with dual displays and control.
B 7760	System includes: two central processors (16 MHz with vectors, two input/output processors with 24 data switching channels each, one maintenance diagnostic unit, one operator console with dual displays and control.
B 7770	System includes: three central processors (16MHz with vectors), two input/output processors with 24 data switching channels each, one maintenance diagnostic unit, one operator console with dual displays and control.
B 7780	System includes; four central processors (16 MHz with vectors), two input/output processors with 24 data switching channels each, one maintenance diagnostic unit, one operator console with dual displays and control.
B 7001-4	Basic memory module – 1.5 megabytes of 88 ns/byte read access, error-correcting memory, four-way interleaving that permits four-word transfers to and from memory.
B 7702	Additional central processor.
B 7785	Additional input/output processor.

MAXIMUM CONFIGURATION

Figure I-2 illustrates the theoretical maximum configuration of the B 7700 system. As many as eight memory modules may be arranged on the exchange with a combined total of up to eight requestors of memory-central processor modules and input/output modules. Any single requestor of memory may address and gain access to the entire contents of high-speed main memory (1,048,576 words, or 6,291,456 eight-bit bytes). On the maintenance bus (which services the memory control modules, central processor modules, input/output modules, and disk file optimizers) one or two maintenance diagnostic units may be placed.

At rates of up to 6.75 million bytes per second, a single input/output module is capable of transferring data simultaneously between main memory and 28 peripheral controls (including eight high-speed controls) and between main memory and as many as four data communications processors. It is also capable of handling as many as four disk file optimizers (devices that are used in improving the rate of transfer of data between main memory and disk files). At present, the maximum number of high-speed, medium-speed, and low-speed peripheral devices that may be attached through controls and exchanges to a single input/output module or that may be included in the input/output subsystem of the B 7700 is 255. (Each card reader, pseudoreader, card punch, line printer, paper tape reader, paper tape punch, operator's display terminal, and free-standing magnetic tape unit; each station on a magnetic tape cluster; and each electronics unit in a disk file subsystem is considered a device.) By suitable cross-connection through exchanges, it is possible to establish pathways between disk files, disk packs, or magnetic tape units and more than one input/output module; hence, these peripheral devices can be shared by all of the input/output modules in the system.

Among the peripheral devices available are disk file and disk pack memory modules that constitute a virtual memory that in effect greatly expands the storage capacity of the main memory of the system; these modules, which are interfaced with the input/output module through controls are as follows:

1. Head-per-track disk file modules that are combined under the control of disk file optimizers to form optimized-access memory banks capable of storing from 450 million to 8 billion eight-bit bytes of information per input/output module and whose access time is as low as 2-to-6 milliseconds.

2. Head-per-track disk file modules that are combined (without the control of the optimizer) into random-access memory banks of from 15 million to 16 billion eight-bit bytes per input/output module and whose average access

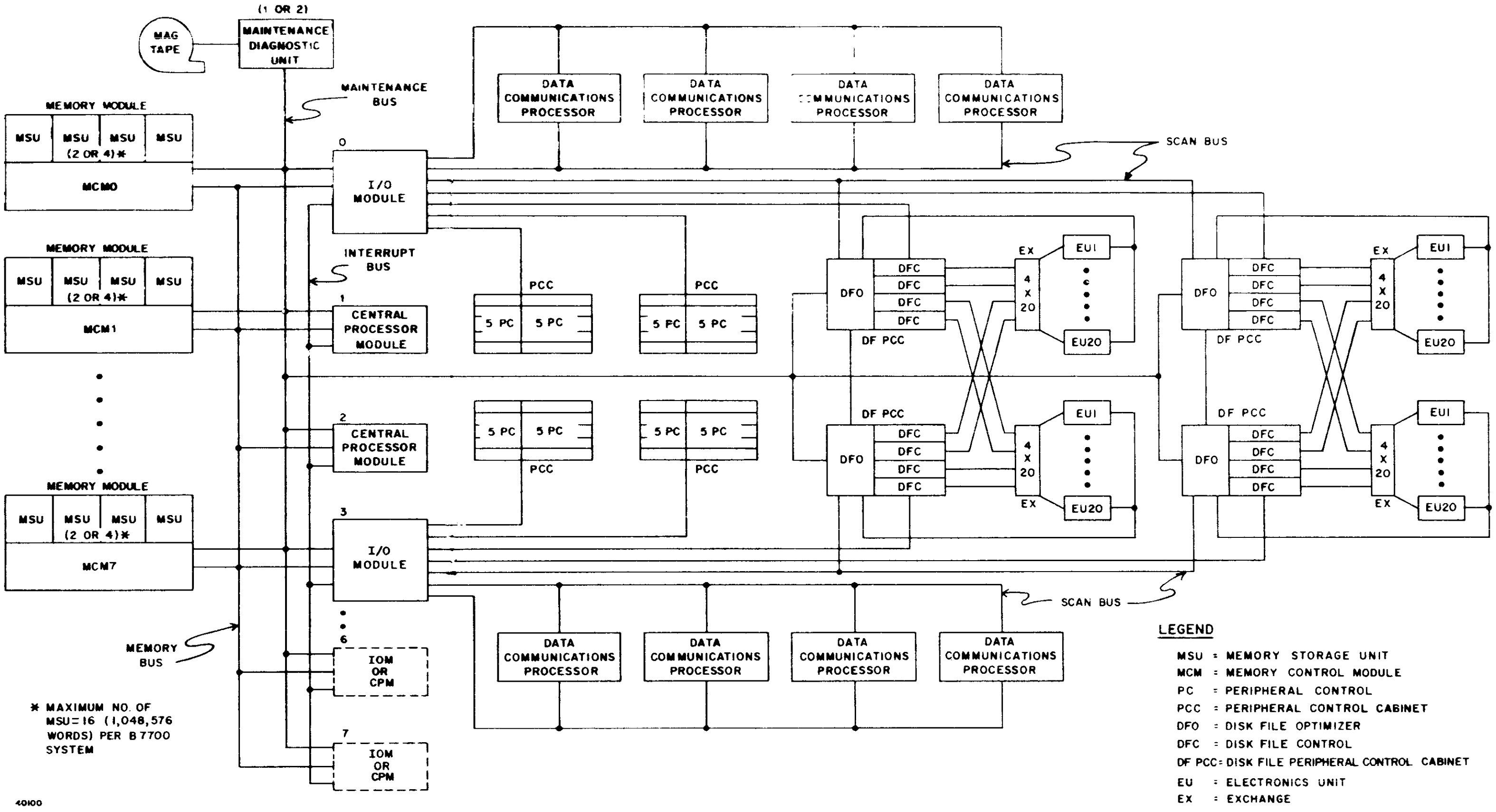


Figure I-2. Maximum Configuration of the B 7700 System

time is 23 or 40 milliseconds.

3. Disk pack memory modules that are combined into random-access memory banks with a capacity for from 121 million to many billions of eight-bit bytes of storage per input/output module and whose average access time is 30 milliseconds.

In addition to the 255 peripheral devices that may be included in the input/output subsystem, there is a vast network of remote terminals, remote controller, and remote computers that can be accommodated by as many as 1024 remote lines served by the four programmable data communications processors that can be controlled by a single input/output module. Normally, each line handles a number of remote devices, and, naturally, systems that have more than one input/output module can have more than one data communications network. Theoretically, the maximum number of data communications processors that could be included in a B 7700 system is 28. (However, currently, the software can only handle a maximum of eight.)

MINIMUM CONFIGURATION

The smallest possible B 7700 system is composed of the central components listed below.

Central Components	Qty
Central processor module (CPM)	1
Input/output module (IOM)	1
Memory module	1
Memory control module (MCM)	1
Memory storage cabinet (MSC)	2
Memory storage unit (MSU)	4
Maintenance diagnostic unit (MDU) and its associated magnetic tape unit	1
Operator's console	1

In addition to these central components, the minimum configuration must contain a disk file memory subsystem at least large enough to hold the master control program, a card reader, a line printer, a magnetic tape unit, peripheral controls, and ac power cabinets. In practice, other peripheral devices and their controls are used with this minimum configuration.

Naturally, this minimum system lacks the redundancy and power of larger configurations. First (lacking redundancy of main-frame modules), this configuration does not take complete advantage of the fail-soft features possible with the B 7700 and second (because each memory control module controls but two storage units), only two-ward transfers, not four-word transfers, to and from memory are possible.

CHAPTER II

SYSTEM ARCHITECTURE

SECTION 1

DATA REPRESENTATION

GENERAL

The basic information structure used in the B 7700 Information Processing System is the word. Each word contains 48 information bits, three tag bits, and one parity bit. (See figure 11-1-1.) The information bits may be used to store character values, logical values, or numeric values. The tag bits are control bits which identify the type of information contained in the information field. The tag bits are inaccessible to normal state (user) programs. The parity bit is used to check for correct information transfer between the CPM or IOM and main memory.

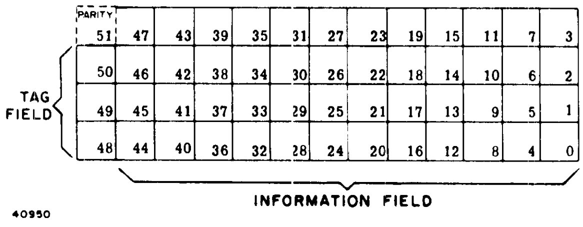


Figure II-1-1. Word Structure

INTERNAL CHARACTER CODES AND COLLATING SEQUENCES

Extended Binary Coded Decimal Interchange Code (EBCDIC) is the primary internal character code of the B 7700. EBCDIC is an eight-bit alphanumeric code containing four zone bits and four numeric bits. Other internal codes which may be used include the American Standard Code for Information Interchange (ASCII), and the Burroughs Common Language Code (BCL). ASCII is the primary data communication code; BCL is used to interface with peripheral units. Numeric EBCDIC and BCL codes may be packed into four-bit digits by internal commands which delete the zones and compress the numeric portion of the characters. In general, characters are collated according to their internal binary value. Character codes and collating sequences are provided in the appendices.

NUMBERS AND NUMBERING SYSTEMS

The B 7700 is a digital computer; that is, values are stored internally in binary digits (bits). Data displayed in registers and printed forms may be in octal or hexadecimal format. Generally, we think in terms of, and manually perform arithmetic with, decimal numbers. Thus, an understanding of all of these numbering systems is desirable.

The decimal system is based on the ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, and upon the powers of ten. The binary system is based upon the two digits 0 and 1, and the powers of two. Two raised to the third power (2^3) is 8, the base of the octal system. Two raised to the fourth power (2^4) is 16, the base of the hexadecimal system. The set of digits for each number system is shown in figure II-1-2.

The digits 0 through 9 and the alphabetic characters A through F comprise the 16-character requirement for the hexadecimal numbering system. The letter A is assigned a value of 10. B equals 11, etc., to F, which equals 15.

DECIMAL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BINARY	0	1														
OCTAL	0	1	2	3	4	5	6	7								
DECIMAL	0	1	2	3	4	5	6	7	8	9						
HEXADECIMAL	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

40951

Figure II-1-2. Number Base Graphic Characters

BINARY NOTATION

Because a binary digit may have only one of two values, it can be represented by a flip-flop or a bit. A number in internal binary representation is then a series of bits which are either on or off. When a bit is on (1), its position determines the value. Consider an example of five bits.

The least significant bit, if on (1), has a value of 2^0, or 1; the next most significant bit to

$2^0 = 1$

0 = off bit

1 = on bit

value of position = 2^4 2^3 2^2 2^1 2^0

...0 0 0 0 1 = 0 +0 +0 +0 + 1 = decimal 1

...0 0 0 1 0 = 0 +0 +0 +2 + 0 = decimal 2

...0 0 0 1 1 = 0 +0 +0 +2 + 1 = decimal 3

.

.

.

...1 1 1 1 1 = $2^4+2^3+2^2+2^1+$ 1 = 1 = 16 + 8 + 4 + 2 + 1 = decimal 31

40952

Figure II-1-3. Binary Integers

the left of the binary point has the value of 2^1, or 2; the third bit (count from right to left) has the value of 2^2, or 4; etc. In this way, any integer can be represented in binary form. Figure II-1-3 illustrates some integers. Fractions in binary are much the same as integers. Here, though, the powers are negative powers with the first power to the right of the binary point having the value of 2^{-1}, or 1/2; the second bit has the value of 2^{-2}, or 1/4; the third bit 2^{-3}, or 1/8; the fourth bit, 2^{-4}, or 1/16; etc. It is apparent that while some fractions are represented correctly, others can only be approximated. However, the degree of error is very small when a sufficient number of bits are used.

HEXADECIMAL AND OCTAL NOTATION

Since binary words are cumbersome to display, the more efficient methods of hexadecimal and octal notation are used. The hexadecimal representation of a binary word is obtained by dividing the bits into groups of four with each group assigned a successive power of 16. A binary-to-octal conversion is obtained by dividing the bits into groups of three and assigning successive powers of 8 to each group (figure II-1-4).

The relationship between octal, decimal and hexadecimal is shown in figure II-1-5 using the decimal number 1013_{10} (equivalent to 1765_8 and $3F5_{16}$ where the subscript 8, 10, of 16 indicates the base).

$1765_8 = 1 \times 8^3 + 7 \times 8^2 + 6 \times 8^1 + 5 \times 8^0 =$

$1 \times 512 + 7 \times 64 + 6 \times 8 + 5 \times 1 =$

$512 + 448 + 48 + 5 = 1013_{10}$

$1013_{10} = 1 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 3 \times 1^0 =$

$1 \times 1000 + 0 \times 100 + 1 \times 10 + 3 \times 1 =$

$1000 + 0 + 10 + 3 = 1013_{10}$

$3F5_{16} = 0 \times 16^3 + 3 \times 16^2 + F \times 16^1 + 5 \times 16^0 =$

$0 \times 4096 + 3 \times 256 + F \times 16 + 5 \times 1 =$

$0 + 768 + 240 + 5 = 1013_{10}$

40954

Figure II-1-5. Relationship of Octal, Decimal, and Hexadecimal Numbers

NUMBER CONVERSION

BINARY TO DECIMAL CONVERSION

INTEGRAL

This conversion is effected by adding together the value of each bit that is on. In this way, the binary number 11010011 would be equal to:

$1x2^7+1x2^6+0x2^5+1x2^4+0x2^3+0x2^2+1x2^1+1x2^0=$

$1x2^7+1x2^6+$ 0 $+1x2^4+$ $0^{+0}+1x2^1+1x2^0=$

128 + 64 + 16 + 2 + 1 211_{10}

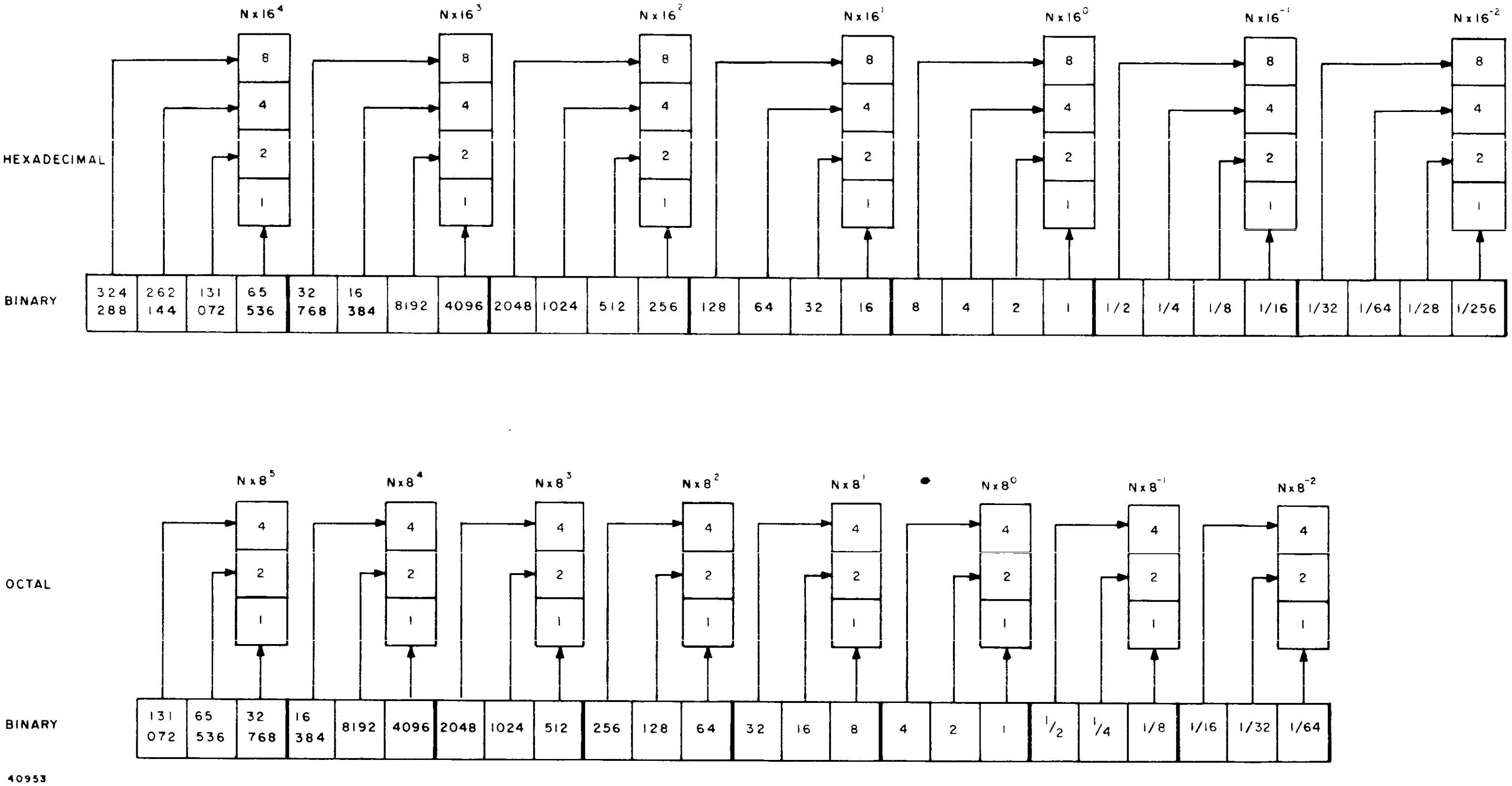


Figure II-1-4. Binary to Hexadecimal and Octal Conversion

A second method of effecting a binary-to-decimal conversion is the "double dabble" method. In this procedure, the high-order bit is doubled (multiplied by 2) and then added to the next lower-order bit. This sum is then doubled and again added to the next lower bit. This process is continued until the entire binary number has been expended (figure II-1-6A). The correct result is obtained after the low-order bit (units) has been added.

FRACTIONAL

The above process will work for integral numbers and for the integral part of fractional numbers, but it will not work for the fractional part of fractional numbers. To convert binary fractions to decimal fractions, division is used. As was previously stated, the bits to the right of the binary point have the decreasing values of 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, etc., or, as fractions 1/2, 1/4, 1/8, 1/16, etc., respectively.

To find the decimal equivalent of a binary fraction, the lowest order significant bit is taken as the integer 1 and divided by 2. The next higher-order bit is then added into the units position of the resulting quotient, and the division is repeated. This is repeated until the binary point is reached. The result is complete when the bit to the immediate right of the binary point has been added into the units position and the result divided by 2. This process is shown in figure II-1-6B.

DECIMAL TO BINARY CONVERSION

INTEGRAL

Decimal to binary conversion may be effected in several ways. If the powers of 2 are known, then the binary equivalent can be found by subtracting from the number the largest power of 2, which is smaller than the decimal number, and then recording a bit for that power of two. The largest power of 2, which is smaller than the result of the preceding subtraction, is then found, subtracted, and the corresponding binary bit recorded. In effect, this is the reverse of the first method of converting from binary to decimal.

A second method of conversion is done by successive division. The decimal number to be converted is divided by 2 and the quotient and remainder are noted. The remainder will always be either 0 or 1. Then the quotient is divided by 2, resulting in another quotient and remainder. This is repeated until the quotient is 0. The remainder, resulting from the first division, is the low order bit; the last remainder is the high order bit. This process is valid for the integral part of a number (figure II-1-7A).

FRACTIONAL

The fractional part of a number may be converted in a method similar to the preceding method of division. The fraction is multiplied

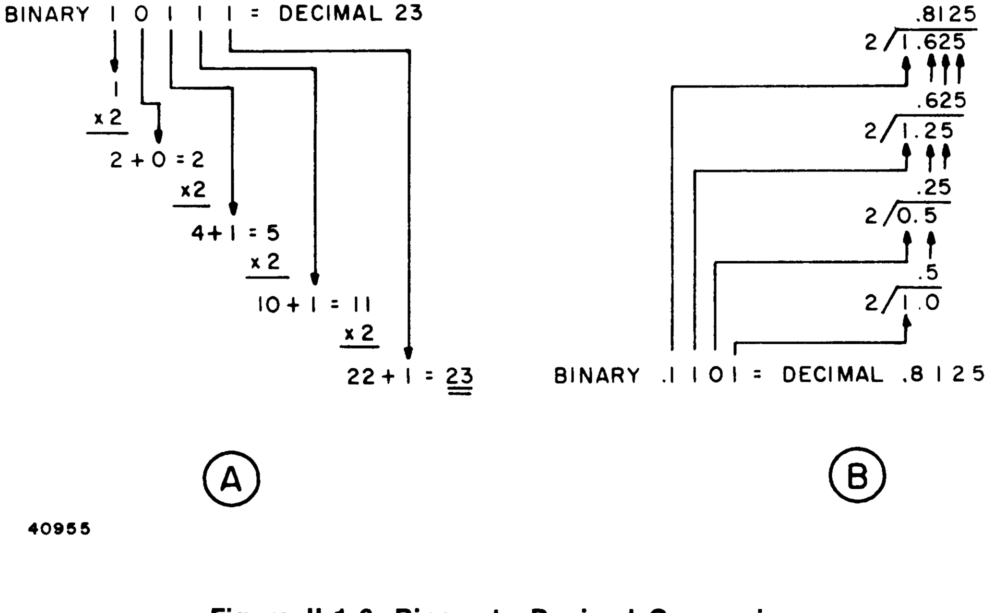
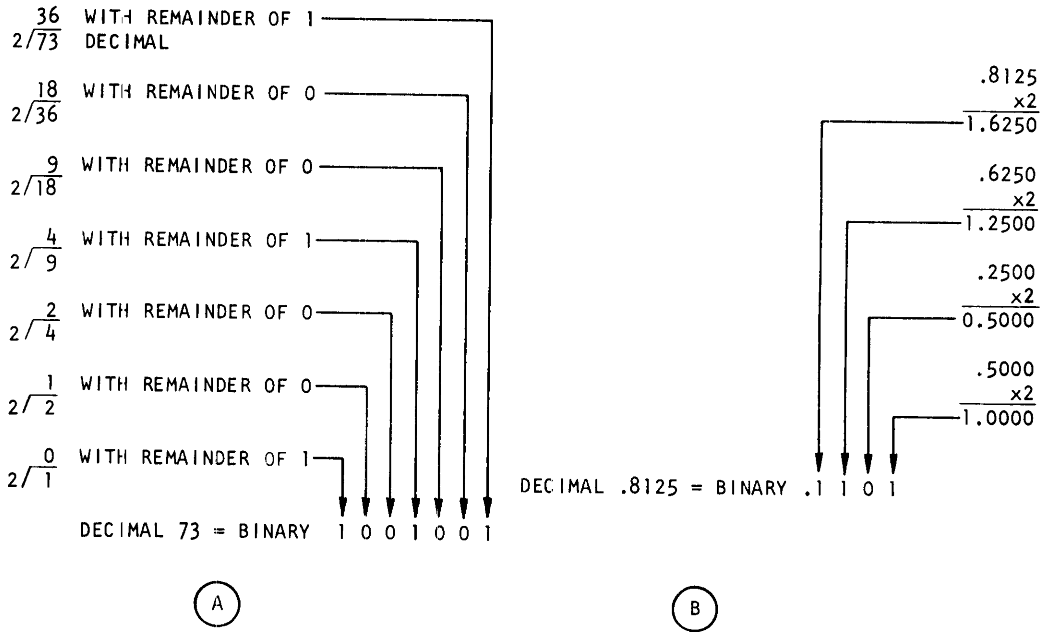


Figure II-1-6. Binary to Decimal Conversion



40956

Figure II-1-7. Decimal to Binary Conversion

by 2 and, if the result is greater than 1, the 1 is recorded in the binary string as a 1 bit. If the product remains less than 1, the binary bit is 0. The fractional part of the product is carried down and again multiplied by 2. This is repeated until the fractional part is equal to 0, or the required degree of accuracy is attained. This process is shown in figure II-1-7B.

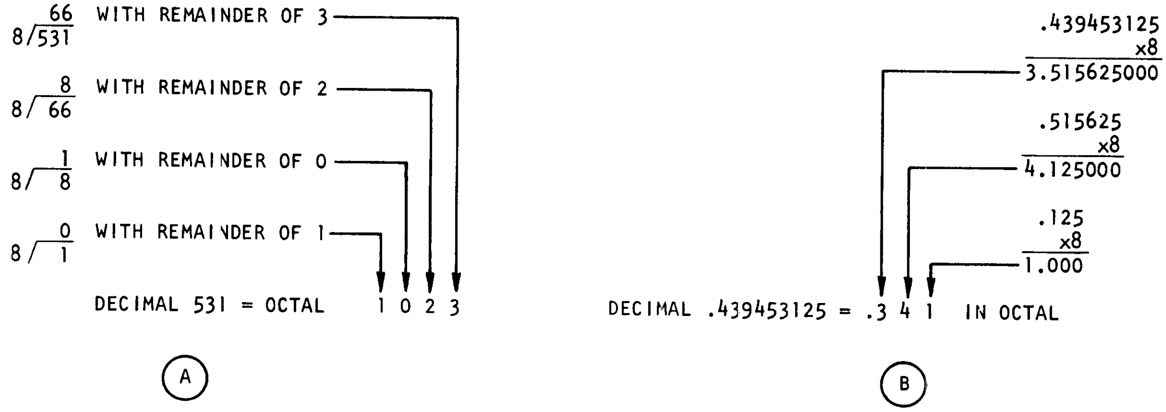
DECIMAL TO OCTAL CONVERSION

INTEGRAL

To convert a decimal number to its octal form, the powers of eight may be used. Another method is to divide the number by eight. The remainder is the low-order octal digit. The quotient is then again divided by eight, and the remainder resulting is the next higher-order octal digit. This process is repeated until the quotient is zero. This method is used for the integral part of numbers (figure II-1-8A).

FRACTIONAL

When a fractional part of the number is to be converted, multiplication is used. Here, the fraction is multiplied by eight and the integral portion formed is the first octal digit to the right of the octal point. This process is repeated until either the fraction is zero, or the desired degree of accuracy is attained. This conversion is shown in figure II-1-8B.



40957

Figure II-1-8. Decimal to Octal Conversion

8^n	n	8^{-n}
1	0	1.0
8	1	0.125
64	2	0.015625
512	3	0.001953125
4096	4	0.000244140625
32768	5	0.000030517578125
262144	6	0.000003814697265625
2097152	7	0.000000476837158203125
16777216	8	0.000000059604644775390625
134217728	9	0.000000007450580596923828125
1073741824	10	0.0000000009313225746154785515625
8589934592	11	0.000000000116415321826934814453125
68719476736	12	0.000000000014551915228366851806640625
549755813888	13	0.000000000001818989493545856475830078125

40958

Figure II-1-9. Powers of 8

OCTAL TO DECIMAL CONVERSION

OCTADE

In octal to decimal or decimal to octal conversions, if the powers of 8 are known, then the procedure is much the same as the corresponding subtraction method of binary. The difference is the digital multiplier which will have a value of from 0 through 7 in octal. Each octal digit will be referred to as an octade. The values of the octades are shown in figure II-1-9.

INTEGRAL

On the conversion from octal to decimal, a method very similar to "double dabble" may be used. Here, the higher-order octade is multiplied by 8 and then added to the next lower octade. This sum is then multiplied by 8 and again added to the next lower octade. This is continued until the first octade to the left of the octal point is reached. After the units octade has been added, the result should be complete (figure II-1-10A).

FRACTIONAL

The above method is valid for the integral part of a number, but for the fractional part of a number, the following must be used. The lowest order octade is considered to be an integer. As such, it is divided by 8. The next higher octade is then added to this quotient in the

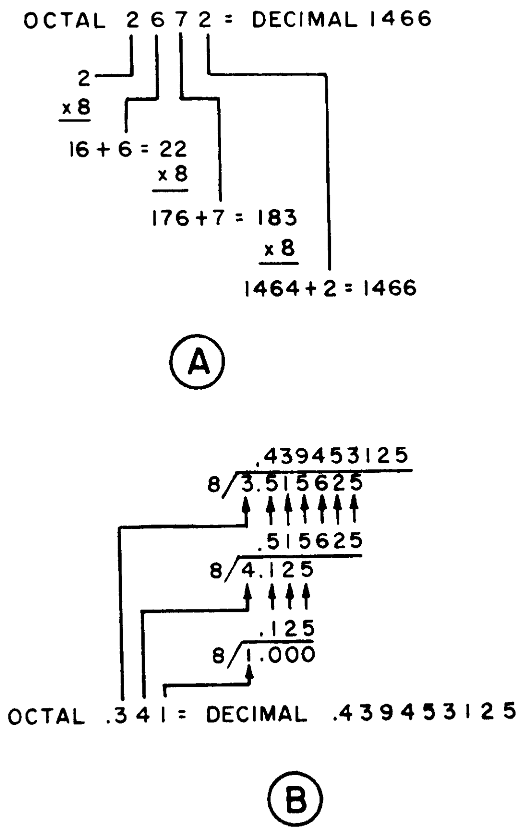


Figure II-1-10. Octal to Decimal Conversion

units position and the sum is again divided by 8. This continues until the first octade to the right of the octal point has been added and the result divided by 8. (See figure II-1-10B.)

DECIMAL TO HEXADECIMAL CONVERSION

To convert an integral or a fractional decimal number to its hexadecimal form, the powers of 16 may be used. Methods similar to those used for conversion to octal representation may also be used, with the multiplication or division being by 16 rather than eight; however, such methods are very cumbersome. The simplest method is to convert the decimal number to a binary number as described earlier, and then convert the binary number to its hexadecimal representation (each four binary digits are used to form one hexadecimal digit).

HEXADECIMAL TO DECIMAL CONVERSION

The simplest method for converting integral or fractional hexadecimal numbers to their decimal equivalent is to first convert the hexadecimal number to its binary equivalent (each hexadecimal digit is used to from four binary digits) and then convert the resulting binary number to its decimal representation as described earlier.

OPERAND FORMATS

Operands are the words of information that are worked with when processing. An operand may be used to store numeric values (a numeric operand), logical values (a logical operand), or character values (a string operand). Most operands are one word in length, and are identified by a tag field of zero. Double precision operands, which are used to store numbers in which many significant digits of accuracy are needed, are two words in length and are identified by a tag field of two. Thus, the tag field of an operand indicates the size of the operand (one or two words).

NUMERIC OPERANDS

Numeric operands are used to store numeric values (numbers) in floating point format. A numeric operand may be single or double precision.

When the tag bits of a memory word (bits 50, 49, 48) are 0 (000), they denote a single-precision operand. When the tag bits are 2 (010), i.e., bit 49 set, they denote a double precision operand.

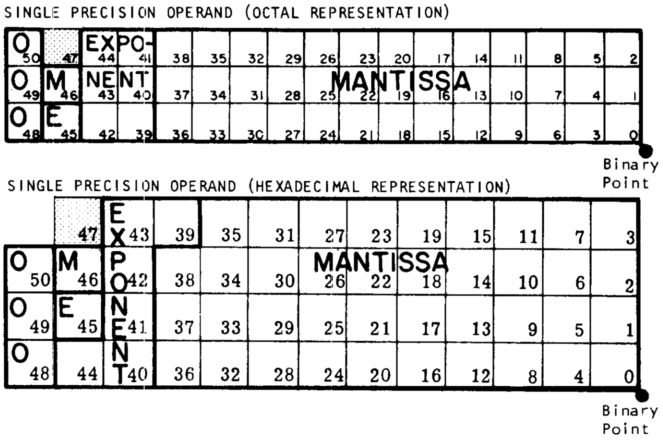
SINGLE PRECISION OPERANDS

All numeric operands are expressed in floating point form, where each numeric operand has both a mantissa and an exponent. This form may be related to power of ten notation where 13297. is the mantissa and -3, the exponent in a representation of the number 13.297 (13297. x 10^{-3}). The mantissa of a single precision operand is comprised of 39 bits which make up 13 octades. The mantissa of a single precision numeric operand is considered to be an integer and is treated as such; i.e., the binary point is considered to be to the right of the least significant octade. The exponent of the number is represented by 6 bits (bits 44 through 39) which form two octades. Bit number 45 is the sign of the exponent. When 45 is off, the exponent is positive; when on, negative. Bit 46 is the sign of the mantissa, which is the overall sign of the operand.

The structure of a single precision operand is shown in figure II-1-11. Because the exponent is an octal scale factor, the single precision operand is shown in both hexadecimal and octal representation.

EXPONENT FIELD

The exponent is a binary number which, with its sign, is an octal scale factor for the mantissa. That is, the binary point in the mantissa must be shifted left three binary places (the mantissa must be shifted right three binary places) for each increase by one in the value of the exponent. The exponent is used for automatic scaling of operands when arithmetic, comparison and integer operations are being performed. The range of the exponent is from +63 to -63 for single-precision operands.


SINGLE PRECISION OPERAND (OCTAL REPRESENTATION)

SINGLE PRECISION OPERAND (HEXADECIMAL REPRESENTATION)

Q 50 47 EXPO- 44 41 38 35 32 29 26 23 20 17 14 11 8 5 2
O 49 M 46 NENT 43 40 37 34 31 28 25 MANTISSA 22 19 16 13 10 7 4 1
O 48 E 45 42 39 36 33 30 27 24 21 18 15 12 9 6 3 0
Binary Point
47 E X 43 39 35 31 27 23 19 15 11 7 3
O 50 M 46 P O 42 38 34 30 26 MANTISSA 22 18 14 10 6 2
O 49 E 45 N E 41 37 33 29 25 21 17 13 9 5 1
O 48 44 N T 40 36 32 28 24 20 16 12 8 4 0
Binary Point

TAG	50:3	000
	47:1	Not used
M	46:1	Sign of Mantissa. 1 = Negative, 0 = Positive.
E	45:1	Sign of exponent. 1 = Negative, 0 = Positive.
EXPONENT	44:6	Exponent.
MANTISSA	38:39	Mantissa.

Figure II-1-11. Single Precision Operand

MANTISSA FIELD

The mantissa is the significant part of the operand. The magnitude of the operand is obtained by multiplying the value contained in the mantissa by eight raised to the value of the exponent sign and exponent as follows:

$V = \pm M \times 8^{\pm E}$

where:

V = Value of number

± M = Mantissa with sign

± E = Exponent with sign

The order of number magnitude in the 39 bit mantissa, as decimal numbers and powers of base 16, 8, and 2 is shown in figure II-1-12.

DOUBLE PRECISION OPERANDS

Double precision operands are identified by a tag field of two, indicating that the operand is one of a pair of two words (figure II-1-13).

The first word of the double precision operand is identical to the single precision operand.

The integral part of the mantissa is contained in the mantissa field of the first word. The fractional part of the mantissa is contained in the mantissa extension field of the second word.

The 15-bit exponent of a double precision operand is formed by the concatenation of the

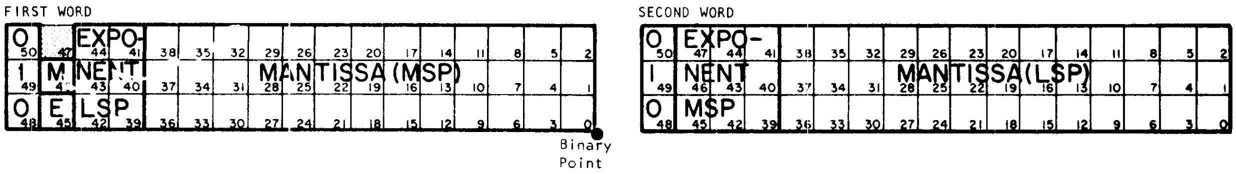
REGISTER BIT SET	DECIMAL	DECIMAL RECIPROCAL	HEX.	OCTAL	BINARY
0	1	1.0	16^0	8^0	2^0
1	2	0.5			
2	4	0.25			
3	8	0.125		8^1	2^3
4	16	0.0625	16^1		
5	32	0.03125			
6	64	0.015625		8^2	2^6
7	128	0.0078125			
8	256	0.00390625	16^2		
9	512	0.001953125		8^3	2^9
10	1024	0.0009765625			
11	2048	0.00048828125			
12	4096	0.000244140625	16^3	8^4	2^{12}
13	8192	0.0001220703125			
14	16384	0.00006103515625			
15	32768	0.000030517578125		8^5	2^{15}
16	65536	0.0000152587890625	16^4		
17	131072	0.00000762939453125			
18	262144	0.000003814697265625		8^6	2^{18}
19	524288	0.0000019073486328125			
20	1048576	0.00000095367431640625	16^5		
21	2097152	0.000000476837158203125		8^7	2^{21}
22	4194304	0.0000002384185791015625			
23	8388608	0.00000011920928955078125			
24	16777216	0.000000059604644775390625	16^6	8^8	2^{24}
25	33554432	0.0000000298023223876953125			
26	67108864	0.00000001490116119384765625			
27	134217728	0.000000007450580596923828125		8^9	2^{27}
28	268435456	0.0000000037252902984619140625	16^7		
29	536870912	0.00000000186264514923095703125			
30	1073741824	0.000000000931322574615478515625		8^{10}	2^{30}
31	2147483648	0.0000000004656612873077392578125			
32	4294967296	0.00000000023283064365386962890625	16^8		
33	8589934592	0.000000000116415321826934814453125		8^{11}	2^{33}
34	17179869184	0.0000000000582076609134674072265625			
35	34359738368	0.00000000002910383045673370361328125			
36	68719476736	0.000000000014551915228366851806640625	16^9	8^{12}	2^{36}
37	137438953472	0.0000000000072759576141834259033203125			
38	274877906944	0.00000000000363797880709171295166015625			
*	549755813887				
39	549755813888	0.000000000001818989403545856475830078125		8^{13}	2^{39}

* FIRST 39 BITS SET. (MAXIMUM INTEGER VALUE ALLOWED).

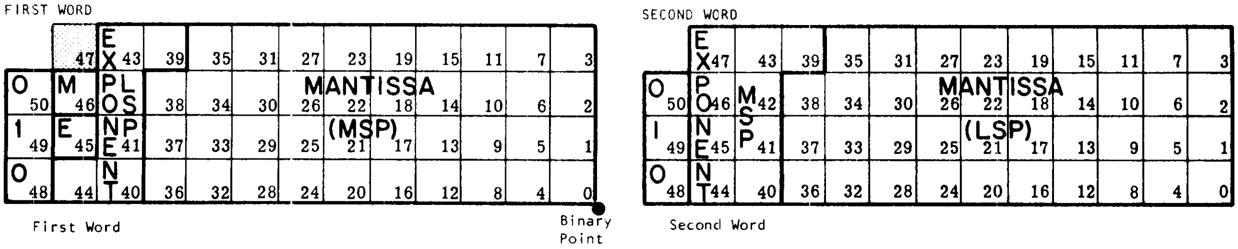
40961

Figure II-1-12. Order of Magnitude Chart

DOUBLE PRECISION OPERAND (OCTAL REPRESENTATION)



DOUBLE PRECISION OPERAND (HEXADECIMAL REPRESENTATION)



First Word

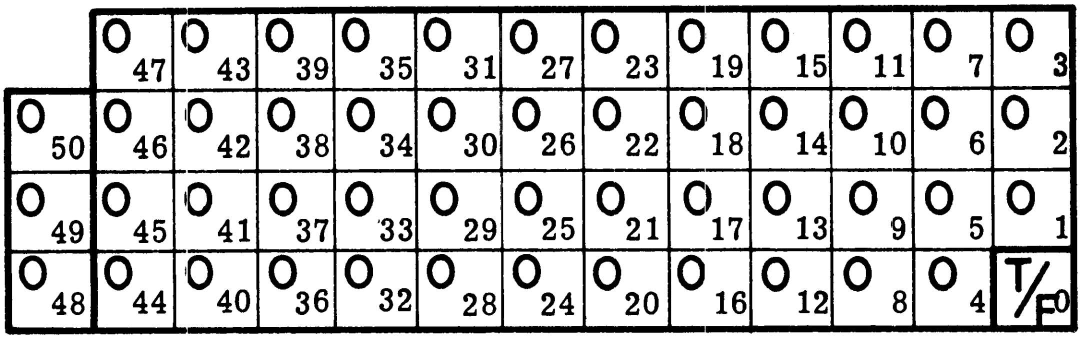
Field	Bits	Description
TAG	50:3	010
	47:1	Not used
M	46:1	1 = negative, 0 = positive.
E	45:1	Sign of exponent.
		1 = negative, 0 = positive.
EXPONENT LSP	44:6	Least significant portion of exponent.
MANTISSA MSP	38:39	Most significant portion of mantissa.

Second Word

Field	Bits	Description
TAG	50: 3	010
EXPONENT MSP	47:9	Most significant portion of exponent.
MANTISSA LSP	38:39	Least significant portion of mantissa.

40962

Figure II-1-13. Double-Precision Operand



Field	Bits	Description
TAG	50:3	000
	47:47	All zeroes.
T/F	0:1	True/false bit.
		1 = True, 0 = False

40963

Figure II-1-14. Logical Operand

exponent extension with the exponent. The exponent extension is more significant than the exponent.

NUMBER RANGES AND NORMALIZATION

To add and subtract two numeric operands on the B 7700, the exponents of the two operands must be equal. The B 7700 equalizes the exponents of the two operands automatically; this equalization may require that one of the operands be "normalized." Normalization occurs if the exponent difference of the two operands is greater than the number of leading zero (octal) digits in the mantissa of the operand with the larger exponent. In such cases, the larger operand is normalized, and the mantissa of the smaller operand is then shifted right until the exponents are equal.

A normalized number is a number which has the smallest exponent with which the number can be expressed without losing the most significant digit of the number. A number is normalized by shifting the mantissa to the left, (moving the binary point right) in three-bit increments until the number of leading zeroes in the mantissa is less than three. For each three-bit shift to the left (of the mantissa), the exponent is decreased by one.

Because of automatic normalization by the CPM, the range of numbers which are useable on the B 7700 includes both normalized and unnormalized numbers. In general, normalized numbers are those which the system may use for arithmetic, and unnormalized numbers are those which the system may store.

The largest and smallest numbers representable as normalized and unnormalized operands are:

The largest single precision integer		549755813887_{10}	decimal
	or	$8^{13}-1$	
		0007777777777777	octal
The largest single precision number		$4.31359146673 \text{x} 10^{68}$	decimal
	or	$(8^{13}-1) \text{x} 8^{63}$	
		0777777777777777	octal
The largest double precision integer		302231454903657293676543	decimal
	or	$8^{26}-1$	
	(first word)	0157777777777777	octal
	(second word)	0007777777777777	
The largest double precision number		$1.948828382050280791124469 \text{x} 10^{29603}$	decimal
	or	$(1-8^{-26}) \text{ x } 8^{32780}$	
	(first word)	0777777777777777	octal
	(second word)	7777777777777777	
The smallest positive unnormalized single precision number		$1.27447352891 \text{x} 10^{-57}$	decimal
	or	8^{-63}	
		1770000000000001	octal
The smallest positive normalized single precision number		$8.7581154020 \text{x} 10^{-47}$	decimal
	or	8^{-51}	
		1771000000000000	octal
The smallest positive normalized double precision number		$1.93854585713758583355640 \text{x} 10^{-29581}$	decimal
	or		
	(first word)	1771000000000000	octal
	(second word)	7770000000000000	

The number sets are symmetrical with respect to zero. The negative number corresponding to any valid positive number may also be expressed. From the ranges above, one can see that a single precision integer must always have an exponent of zero.

LOGICAL OPERANDS

Logical operands (figure II-1-14) have one of two values: true (on) or false (off). Logical values are the result of Boolean operations or relational operations. Relational operators generate a logical value as the result of an algebraic comparison of two arithmetic expressions. Bit 0 contains the logical value. Relational operators set bit 0, where conditional operators use bit 0 for the decision.

> NOTE
>
> Logical operators (LAND, LOR, LNOT, and LEQV) cause a logical operation to be performed on each bit of the two operands and the results of these operations (48 single precision values or 96 double precision values) are left in the top-of-stack operand. Logical operators may operate on logical, string, or numeric operands.

STRING OPERANDS

A string operand is a single word operand (identified by a tag of zero) which is used to store characters. Character representation may be 8-bit (EBCDIC), 7-bit (USASCII), 6-bit (BCL), or 4-bit (packed BCD) characters. Generally, a string of characters is stored in one or more string operands in memory as an array or table. Such arrays or tables are addressed by means of string descriptors. The format of string operands for storage of 8-bit, 7-bit, 6-bit, and 4-bit characters is shown in figure II-1-15.

String operands may also be used to store signed numeric characters in 8-bit, 6-bit, and 4-bit formats. Each string operand can store one signed numeric number consisting of six 8-bit characters, eight 6-bit characters, or 11 4-bit characters. Eight-bit and 6-bit characters are divided into a zone portion and a number portion. The number portion consists of the four least significant bits of each character; the remaining bits form the zone. When 8-bit or 6-bit signed numeric characters are stored in a string operand, the sign of the characters is stored in the zone bits of the least significant character. When 4-bit signed numeric characters are stored in a string operand, the sign of the characters is stored as the most significant character of the operand. Table II-1-1 shows the bit configurations for negative and positive signs in 8-bit, 6-bit, and 4-bit formats. Figure II-1-16 illustrates the manner in which a signed number (–4259) is stored in 8-bit bit, 6-bit, and 4-bit code.

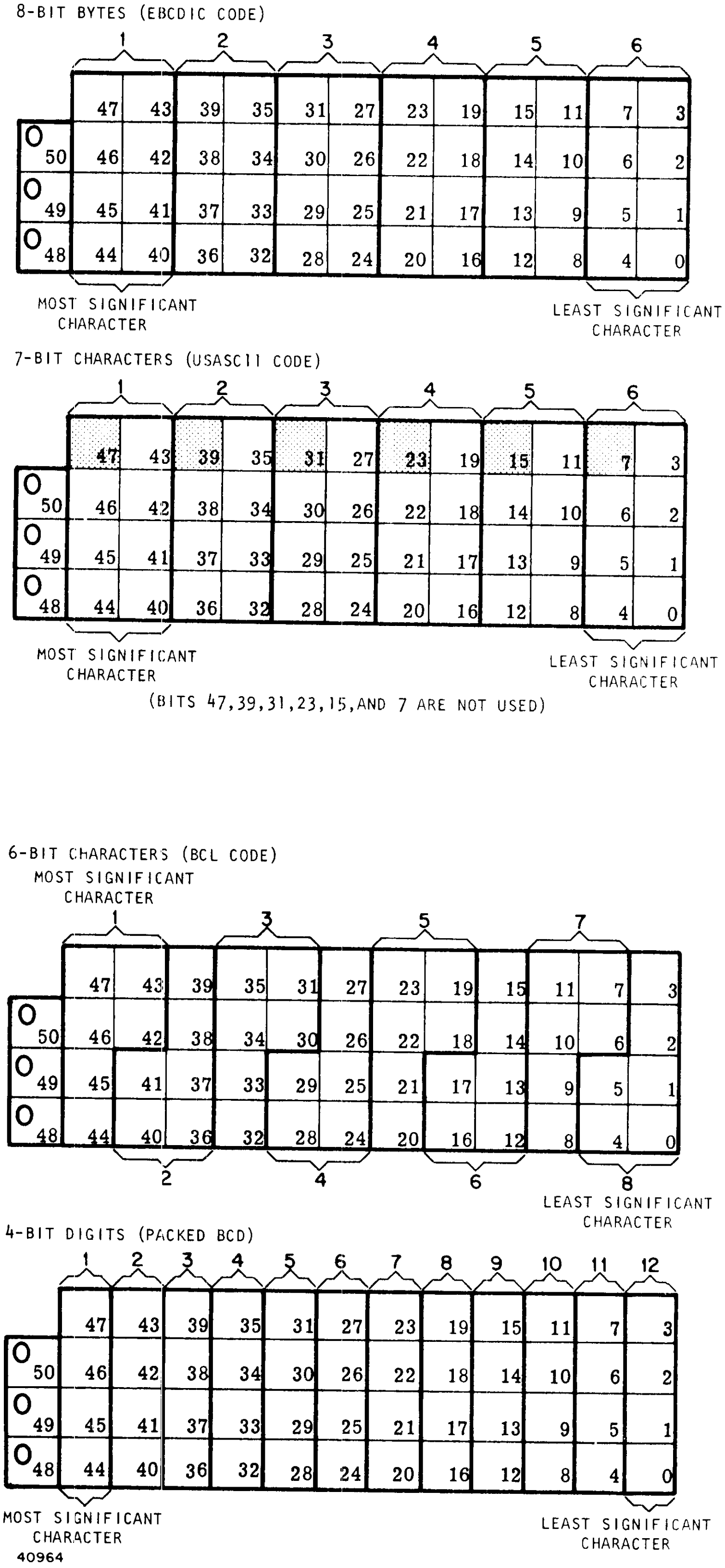


Figure II-1-15. String Operands

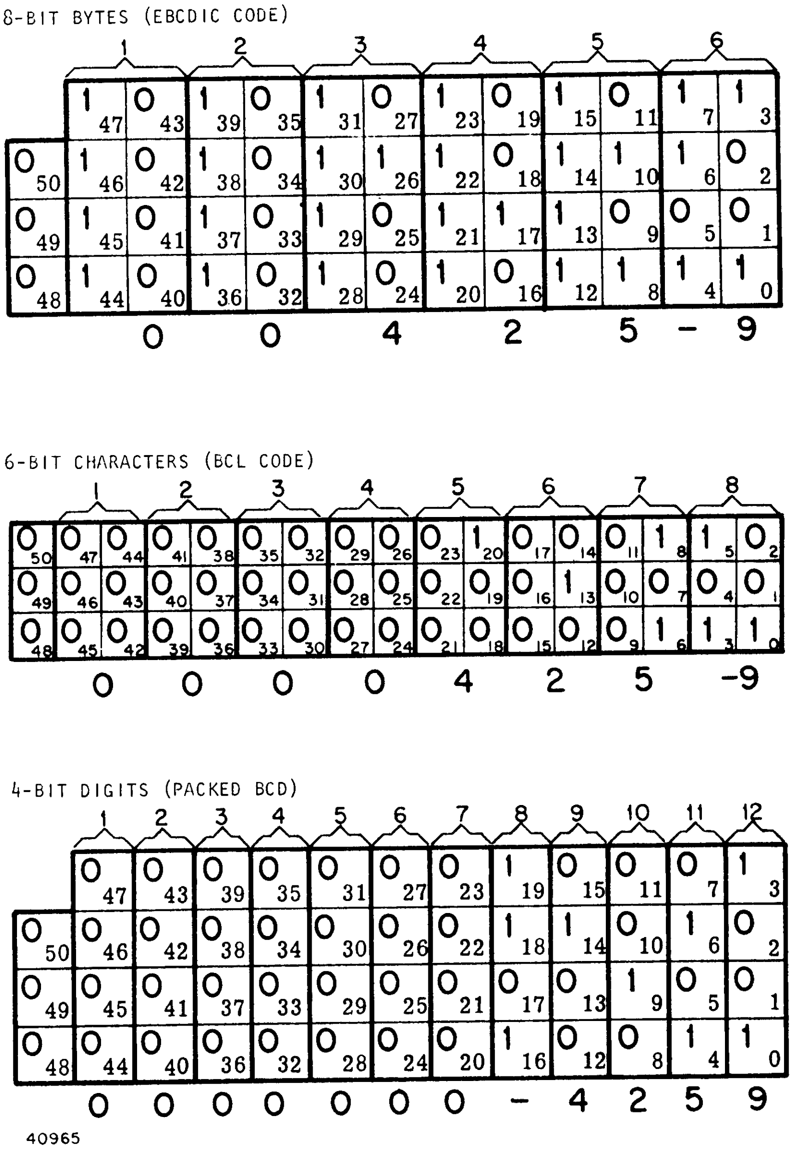


Figure II-1-16. Use of String Operand to Store Signed Number (-4259)

Table II-1-1. Sign Configurations Of String Operands

Size	Sign Location	Negative	Positive
8-bit	Zone, least significant character	1101	Any bit configuration other than the negative bit configurations
6-bit	Zone, least significant character	10	Any bit configuration other than the negative bit configurations
4-bit	Most significant digit	1101	Any bit configuration other than the negative bit configurations

SECTION 2

POLISH NOTATION AND STACK

GENERAL

To facilitate the understanding of the B 7700 stack concept, a method of mathematical notation known as Polish notation must be understood. A problem that exists with most forms of mathematical notation is clarifying the boundaries of specific terms. This has been eliminated with the use of parentheses, brackets, and braces. However, with a complex equation, it becomes necessary to duplicate the use of the few types of delimiters that exist. It might be noted that it is common to encounter mathematical equations such as Y = 5Z + 7/2Z and Y = (5Z + 7)2Z. Two equations express different functions of Z, but one could easily be used when the other was intended. From this it can be seen that an error in notation can change the whole problem, because the parentheses have definite meaning.

Polish notation is an arithmetical or logical notational system using only operands and operators arranged in a sequence or string which eliminates the necessity of factor boundaries. The B 7700 compilers translate source statements to Polish strings, and convert these Polish strings to a series of machine instructions (program operators).

POLISH NOTATION

The essential difference between Polish notation and conventional notation is that operators are written to the right of operands instead of between them. For example, the conventional B + C would be written B C + in Polish notation. Looking at the example, A = 7 (B + C), it would be written in Polish notation as follows:

A7BC+*=

Any expression written in Polish notation is called a Polish string. In order to fully understand this concept, the rule for evaluating a Polish string should be known.

GENERAL RULES FOR GENERATION OF POLISH STRING

Figure II-2-1 is a flow chart for generation of a Polish string. In general, the rules for generation of a Polish string may be stated as follows. If the source of expression is:

Name	Action
A Variable	Place variable in string being built and examine next symbol.
An Operator -Separator	Place in delimiter list and examine next symbol.
-Arithmetic or Boolean operator and last entered delimiter list symbol was: a. an operator of lower priority. b. a left bracket " [" or paren "(". c. a separator. d. nothing (delimiter list empty).	Place operator in the delimiter list and examine next source symbol.
-An Arithmetic or Boolean operator and last entered delimiter list symbol was: an operator of priority equal to or greater than the symbol in the source.	Remove the operator from the delimiter list and place in the string being built. Then compare the next symbol in the delimiter list against the source expression symbol.
-A right bracket "] " or parenthesis ")"	Pull out from delimiter list or until corresponding left bracket or parenthesis.

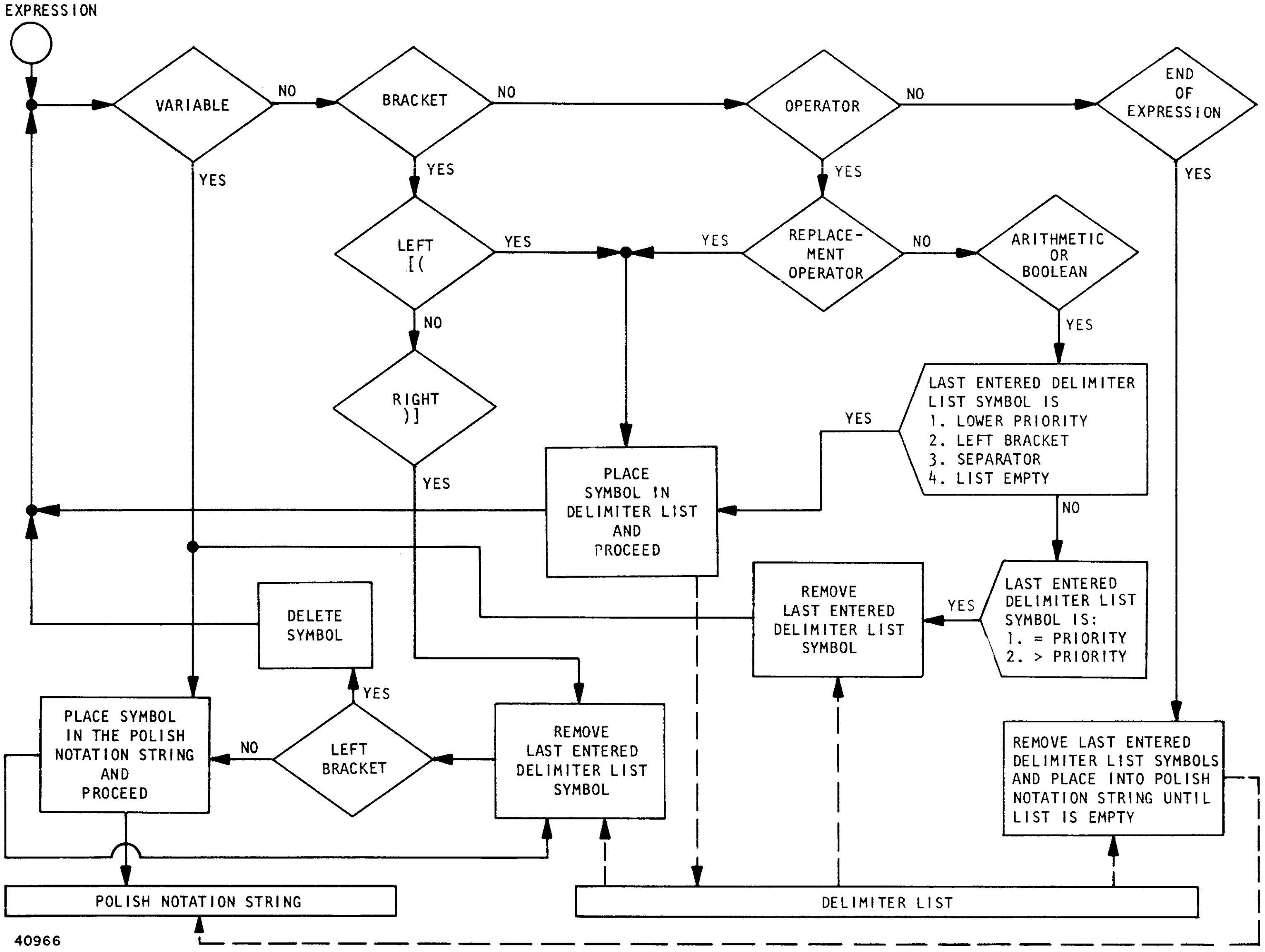

EXPRESSION
VARIABLE
NO
BRACKET
NO
OPERATOR
NO
END OF EXPRESSION
YES
YES
YES
YES
LEFT [(
YES
YES
REPLACE-MENT OPERATOR
NO
ARITHMETIC OR BOOLEAN
YES
NO
RIGHT)]
YES
LAST ENTERED DELIMITER LIST SYMBOL IS
1. LOWER PRIORITY
2. LEFT BRACKET
3. SEPARATOR
4. LIST EMPTY
YES
PLACE SYMBOL IN DELIMITER LIST AND PROCEED
NO
REMOVE LAST ENTERED DELIMITER LIST SYMBOL
YES
LAST ENTERED DELIMITER LIST SYMBOL IS:
1. = PRIORITY
2. > PRIORITY
DELETE SYMBOL
YES
PLACE SYMBOL IN THE POLISH NOTATION STRING AND PROCEED
NO
LEFT BRACKET
REMOVE LAST ENTERED DELIMITER LIST SYMBOL
REMOVE LAST ENTERED DELIMITER LIST SYMBOLS AND PLACE INTO POLISH NOTATION STRING UNTIL LIST IS EMPTY
POLISH NOTATION STRING
DELIMITER LIST
40966

Figure II-2-1. Polish Notation Flow Chart

EVALUATING POLISH STRING

The following procedure may be used to evaluate a Polish string.

a. Scan the string from left to right.

b. Remember the operands and the order in which they occur.

c. When an operator is encountered do the following:

1) Take the two operands which were last encountered.
2) Operate upon them according to the type of operator encountered.
3) Eliminate these two operands from further consideration.
4) Remember the result of (2) and consider it as the last operand encountered.

Following this procedure through the Polish string A7BC+* = would evaluate to A assuming the value 7 (B + C) (figure II-2-2).

NOTE

> Because replacement operators vary depending upon the language used, ←, =, and := may be used interchangeably in discussing Polish strings.

PROGRAM CODE STRING

When a program is compiled, the source language statements are converted into a string of machine language operators. These operators are assembled into a Polish notation string and are referred to as the program code string. Each machine instruction in the string normally consists of one to three 8-bit syllables. The instructions are packed consecutively into program words. (See figure II-2-3.) An array of program words, which can be any length, is called a program code segment. The compiler usually divides the generated code string into two or more program segments. The number of segments depend on the structure of the source program. Program segments are normally stored on disk files. When a program is executed, program segments are made present in memory as needed. Because program segments are not modified during execution, a single copy of a program segment in memory may be used for several concurrent executions of the same program; thus, the program code string is often described as "re-entrant".

Step	Symbol Being Examined	Symbol Type	Operands Being Remembered and Their Order of Occurrence (1 or 2) Before Operation	Operation Taking Place	Results Operation
a	B	Operand			
b	C	Operand	1 B		
c	+	Add Operator	2 C 1 B	B + C	(B + C)
d	7	Operand	1 (B + C)		
e	x	Multiply Operator	2 7 1 (B + C)	7 x (B + C)	7 x (B + C)
f	A	Operand	1 7(B + C)		
g	=	Replace Operator	2A 1 7(B + C)	A ← 7(B + C)	A = 7(B + C)

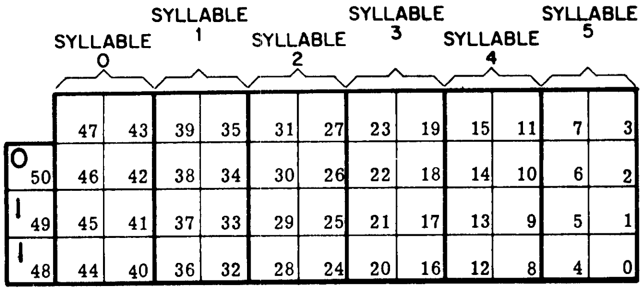
Figure II-2-2. Evaluation of Polish String A7BC +* =

COMPILATION USING POLISH NOTATION

Polish notation is used as the base for the B 7700 ALGOL compilation algorithm. An ALGOL arithmetic or Boolean expression or assignment statement may be translated to Polish notation in much the same way as the arithmetic (or algebraic) expression that already has been considered. In compiler translation, the source expression is examined one symbol at a time with a left to right scan and is combined into logical entities. As each logical entity is examined, a specific procedure is followed so that the Polish notation expression is constructed in its finalized form with one scan of the source expression.

For each program segment, there is a single segment descriptor, which defines the length and location of the program segment. The segment descriptors are stored in a special stack known as the segment dictionary.

Each job is associated with an unique job stack and with a segmented dictionary stack which may be shared by several jobs. (In addition, the MCP has its own stack and segment dictionary.) Within the job stack, a Program Control Word is provided for each point of entry into a segment of code. The PCW provides an index, not only into the segment dictionary to locate the proper segment descriptor, but also into the program segment itself to locate



	47	43	39	35	31	27	23	19	15	11	7	3
0 50	46	42	38	34	30	26	22	18	14	10	6	2
1 49	45	41	37	33	29	25	21	17	13	9	5	1
1 48	44	40	36	32	28	24	20	16	12	8	4	0

Field	Bits	Description
Tag	50:3	Tag field. Value of three indicates that this word is non-modifiable (except by Overwrite operators).
	47:8	Syllable 0
	39:8	Syllable 1
	31:8	Syllable 2
	23:8	Syllable 3
	15:8	Syllable 4
	7:8	Syllable 5

Figure II-2-3. Program Word

the proper program word and syllable. The formats of the segment descriptor and the PCW are described in detail in section 3 of this chapter.

STACK CONCEPTS

The constants and variables of a program are assigned locations within the "stack" of the program when it is compiled. The stack can be thought of as analogous to a physical stack where the last item placed on the stack is the top of the stack. When items are removed (one at a time) from the stack, the item on the top of the stack is the first item to be removed. The item at the bottom of the stack remains at the bottom of the stack until all other items have been removed from the stack. The stack not only provides an easily manageable means for keeping a dynamic history of the program as it is being processed, but also lends itself to the use of program code strings based on Polish notation.

GENERAL

A job is activated by having a processor assign to the job stack. Two top-of-stack locations (A and B) are linked to the job's stack (figure II-2-4). This linkage is established by the stack-pointer register (S), which contains the memory address of the last word placed in the stack. The two top-of-stack locations (A and B) extend the stack to provide quick access for data manipulation.

Data are brought into the stack through the top-of-stack locations in such a manner that the last operand placed into the stack is the first to be extracted. Total capacity of the top-of-stack locations (A and B) is two operands. Loading a third operand into the top-of-stack locations causes the first operand to be pushed from the top-of-stack locations into the stack. The stack-pointer register (S) is incremented by 1 before a word is placed into the stack and is decremented by 1 after a word is withdrawn from the stack and placed in the Top-of-Stack locations. As a result, the S register continually points to the last word placed into the job's stack.

BASE AND LIMIT OF STACK

A job's stack is bounded, for memory protection, by two registers: the Base-of-Stack register (BOSR) and the Limit-of-Stack register (LOSR). The contents of BOSR define the base of the stack, and the contents of LOSR define the upper limit of the stack. The job is interrupted if the S register is set to the value, contained in either LOSR or BOSR.

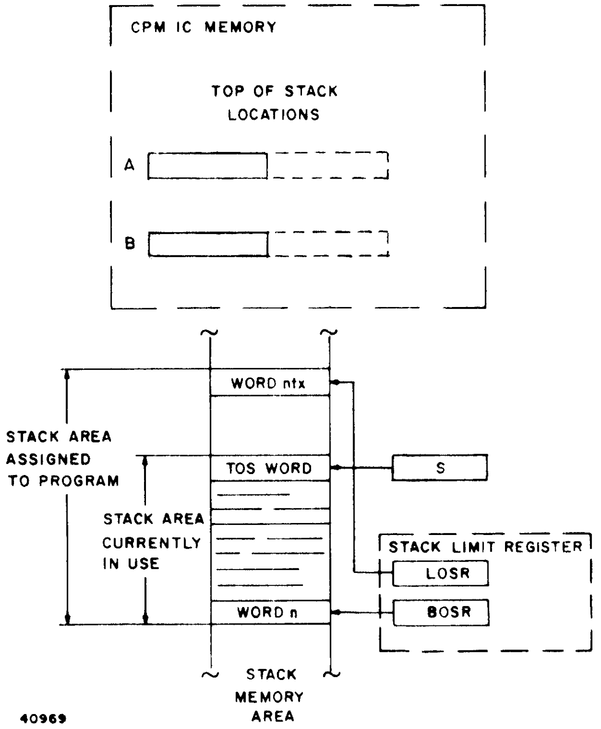


Figure II-2-4. Top of Stack and Stack Bounds Register

BI-DIRECTIONAL DATA FLOW IN THE STACK

The contents of the top-of-stack locations are maintained automatically by the processor to meet the requirements of the current operator. If the current operator requires data transfer into the stack, the top-of-stack locations receive the incoming data, and the surplus contents, if any, of the top-of-stack locations, are pushed into the stack. Words are brought out of the stack into the top-of-stack locations. These words are used by operators which require the presence of data in the top-of-stack locations. These operators, however, do not explicitly move data into the stack.

DOUBLE PRECISION STACK OPERATION

Each top-of-stack location (A and B) can accommodate two memory words. For single precision operations, location A will contain one single precision operand and location B will contain the other single precision operand. However, calling a double precision operand into either top-of-stack location (A or B) will cause both halves of the double precision operand to be loaded into the A or B location. The first word is loaded into the top-of-stack location and its tag bits are checked. If the value of the tag bits indicates double precision, the second half of the operand is loaded into the second half of the top-of-stack location.

Double precision operands revert to single words when they are pushed down into the stack (the most significant half of the operand is pushed down first). The process is reversed when a double precision operand is returned from the stack to the top-of-stack locations. That is, the least significant half of the double precision operand is popped up first and the tag is discovered to have a value of two, causing the most significant half of the operand to also be popped into the top-of-stack.

HARDWARE IMPLEMENTATION

The B 7700 stack implementation includes a 32-word stack buffer, which permits a portion of an active stack to be contained in IC memory locations within the CPM. This stack buffer (see figure II-2-5) may contain information which has not yet been written to core memory, as well as copies of words which are resident in core memory. The stack buffer permits a portion of the stack to be held local within the CPM, to provide quick access for stack manipulation by the execution unit of the CPM.

In addition to the portion of the stack held local in the stack buffer, certain other data from the stack may be contained in a local memory within the CPM. This local memory, the associative memory, is used to capture data fetched by program unit look ahead which is not resident in the stack buffer.

Although an active stack may be contained partly in the stack buffer within the CPM and partly in core memory, the stack buffer is purged whenever the stack becomes inactive (when a move-to-stack operation takes place). This purging of the stack buffer causes the unique data within the stack buffer to be copied to core memory. Thus, for practical purposes, this section discusses the stack as if it exists solely within core memory. A detailed description of the stack buffer and the associative memory may be found in chapter III.

DYNAMIC PROGRAM HISTORY

One very important aspect of the B 7700 is the retention of the dynamic history for the program being processed. Two lists of program history are maintained in the B 7700 stack, the addressing environment list and the stack history list.

Both of these lists are dynamic, varying as the job proceeds along different program paths with varying sets of data. The two lists grow and contract in accordance with the procedural depth of the program. Both of these lists are generated automatically by the B 7700 hardware. Before further stack discussion can be considered, addressing history and stack history must be discussed.

ADDRESSING HISTORY

The B 7700 CPM provides two methods for addressing data. Direct addressing is provided by descriptors, which contain the address (core or disk) of the data. Descriptors are used to address data which are located outside of the stack area of the job. Relative addressing is provided by the Indirect Reference Word (IRW) and the Stuffed Indirect Reference Word (SIRW). The IRW and SIRW address components are both relative address components. The IRW addresses within the immediate environment of the job relative to one of 32 CPM display registers. The SIRW addresses beyond the immediate environment of the current procedure, the addressing being relative to the base of some job stack. Addressing across stacks is accomplished with an SIRW.

DIRECT ADDRESSING

In general, the descriptor describes and locates data associated with a given job. String descriptors and data descriptors are used to fetch data to the stack or to store data from the stack into an array located outside the

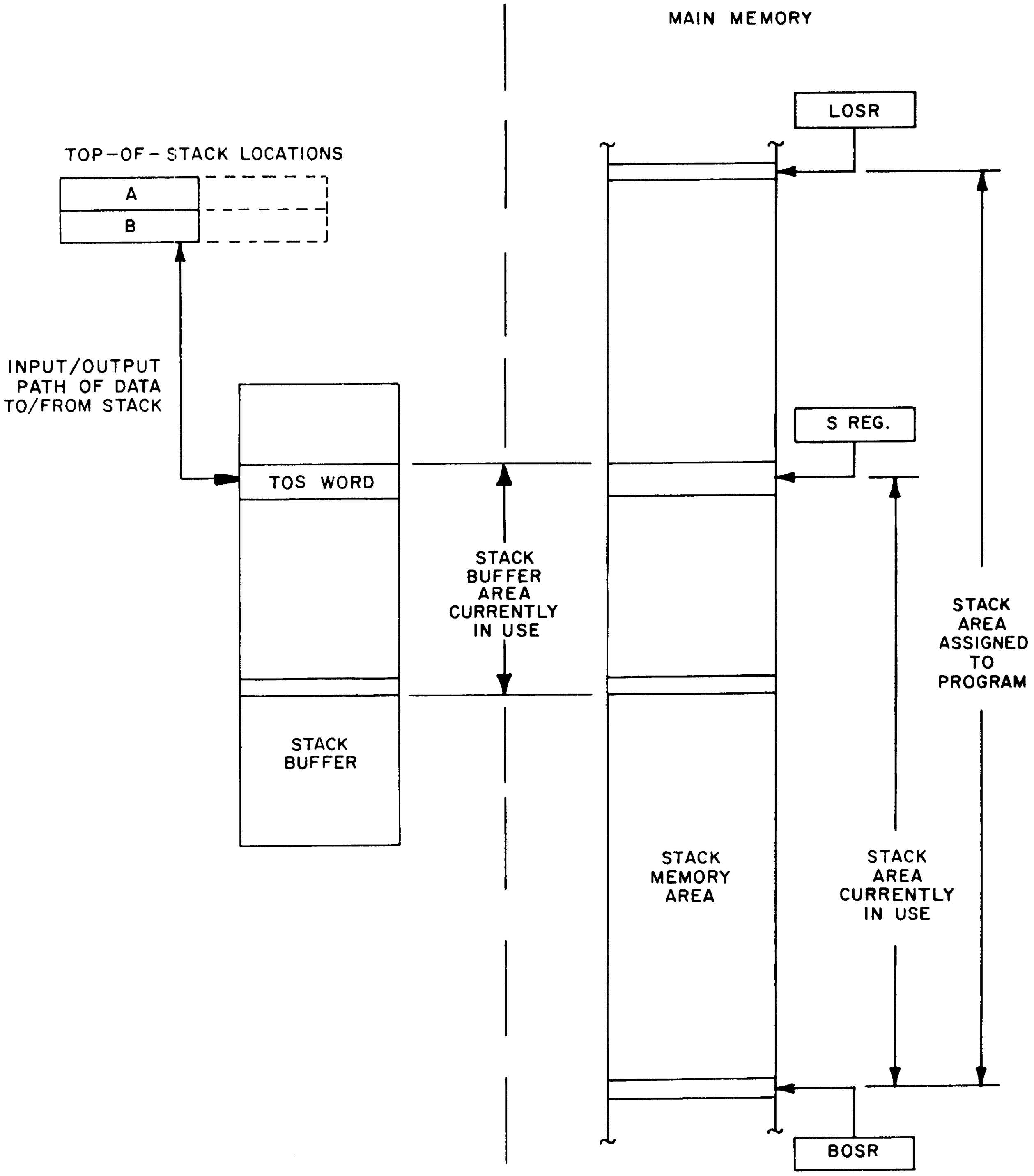

MAIN MEMORY
LOSR
TOP-OF-STACK LOCATIONS
A
B
INPUT/OUTPUT PATH OF DATA TO/FROM STACK
S REG.
TOS WORD
STACK BUFFER AREA CURRENTLY IN USE
STACK AREA ASSIGNED TO PROGRAM
STACK BUFFER
STACK MEMORY AREA
STACK AREA CURRENTLY IN USE
BOSR

Figure II-2-5. Stack Buffer and Stack Memory Area

stack area of the job. The address contained in one of these descriptors is the absolute address of an array in either system main memory or in the backup disk file, as indicated by the setting of a single bit called the presence bit. Another bit, called the presence bit. Another bit, called the double-precision bit, is used to identify the referenced data as single precision or double precision. The formats of string and data descriptors, and detailed discussions of each, are presented in section 3 of this chapter.

RELATIVE-ADDRESSING

Analyzing the structure of an ALGOL program results in a better understanding of the relative-addressing procedures used in the B 7700 stack. The addressing environment of an ALGOL procedure is established automatically as the program is structured by the programmer and is referred to as the lexicographical ordering of the procedural blocks. At compile time, the lexicographical ordering is used to form address couples. An address couple consists of two items:

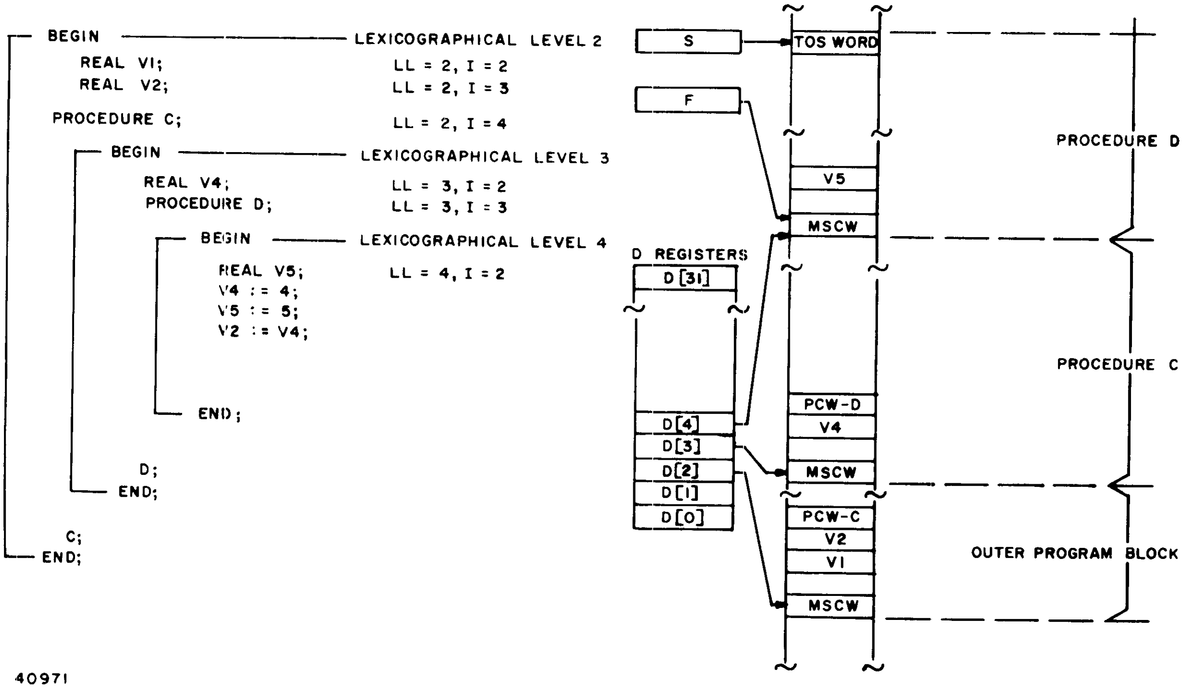
1. The lexicographical addressing level (LL) of the variable,
2. An index value (I) used to locate the specific variable within its addressing level.

The lexicographical ordering of the program remains static as the program is executed, thereby allowing variables to be referenced via address couples as the program is executed.

The lexicographical structure of a very simple ALGOL program is illustrated in figure II-2-6. When executed, this program would call procedure C (LL=3) from the outer block of the program (LL=2), and, in turn, procedure C would call procedure D (LL=4). The stack structure is illustrated as it would exist as procedure D was being executed. It can be seen that, as the outer block of the program was entered, and again as each procedure was entered, a Mark Stack Control Word (MSCW) was placed in the stack. The MSCW (described in detail in section 3 of this chapter) denotes the base of each lexicographical addressing level.

DISPLAY REGISTERS

Each MSCW provides a point in the stack relative to which the variables for the associated addressing level may be referenced. The B 7700 CPM unit contains 32 display registers (D [0] through D for [31]).As shown, the base of each addressing level is addressed by



```
BEGIN ---------------- LEXICOGRAPHICAL LEVEL 2
   REAL V1;             LL = 2, I = 2
   REAL V2;             LL = 2, I = 3
PROCEDURE C;            LL = 2, I = 4
  BEGIN -------------- LEXICOGRAPHICAL LEVEL 3
     REAL V4;           LL = 3, I = 2
     PROCEDURE D;       LL = 3, I = 3
       BEGIN --------- LEXICOGRAPHICAL LEVEL 4
         REAL V5;       LL = 4, I = 2
         V4 := 4;
         V5 := 5;
         V2 := V4;

       END;
     D;
  END;
C;
END;
```

Figure II-2-6. ALGOL Program With Lexicographical Structure and Related Stack Structure

one of these registers. The local variables of the outer block or of the procedures are addressed relative to the D registers. The D registers are updated at each procedure entry or exit.

ABSOLUTE ADDRESS CONVERSION

Each variable is indirectly addressed by an address couple containing a lexicographical level and an index value. The address couple is converted into an absolute memory address when the variable is referenced. The lexicographical level portion of the address couple selects the D register which contains the absolute memory address of the MSCW for the environment (lexicographical level) in which the variable is located. The index value of the address couple is added to the contents of the D register to generate the absolute memory address of the desired variable.

ADDRESSING ENVIRONMENT

Thus far we have considered a very simple program in which each procedure has a different lexicographical addressing level. Generally, however, many procedures of a program may have the same lexicographical addressing level; however, no two procedures of a program may have the same addressing environment. Consider the more advanced exemplary program shown in figure II-2-7.

This program consists of an outer block (LL=2), two procedures which have a lexicographical addressing level of three (procedures A and C), and two procedures which have a lexicographical level of four (procedures B and D). The addressing environment of the program is maintained automatically by linking the MSCWs together in accordance with the lexicographical structure of the program. This linkage is composed of the stack number (STACK NO.) and displacement (DISP) fields of the MSCW, and is inserted into the MSCW when the procedure is entered. A tree-structured addressing environment list is formed by linking the MSCW to the MSCW at the preceding lexicographical level to the procedure being entered. This tree-structured list indicates the addressing environment of the procedures.

```
BEGIN ---------------------------- LEXICOGRAPHICAL LEVEL 2
   REAL V1;                          LL = 2, I = 2
   REAL V2;                          LL = 2, I = 3
   PROCEDURE A;                      LL = 2, I = 4

     BEGIN ----------------------- LEXICOGRAPHICAL LEVEL 3
        REAL V3;                     LL = 3, I = 2
        PROCEDURE B;                 LL = 3, I = 3

           BEGIN ----------------- LEXICOGRAPHICAL LEVEL 4
              V3 := 3;
              V1 := V3;

           END;


        B
     END;



  PROCEDURE C;                       LL = 2, I = 5

     BEGIN ----------------------- LEXICOGRAPHICAL LEVEL 3
        REAL V4;                     LL = 3, I = 2
        PROCEDURE D;                 LL = 3, I = 3

           BEGIN ----------------- LEXICOGRAPHICAL LEVEL 4
              REAL V5;               LL = 4, I = 2
              V4 := 4;
              V5 := 5;
              A;
              V2 := V4;


           END;


        D;
     END;

  C;
END;
```

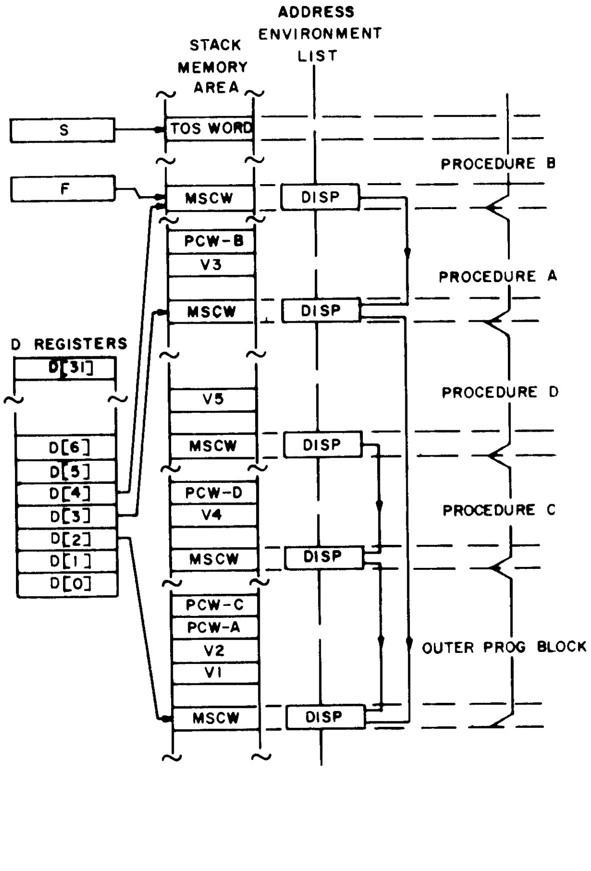


Figure II-2-7. More Advanced ALGOL Program

Comparing the addressing tree in figure II-2-8 with the exemplary program, one can see that when procedure B is being executed, the addressing environment includes only the variables in procedures B and A and the outer block; variables declared in procedure C and D are not addressable by procedure B. Thus, one can see that the address couples assigned to the variables in a program need not be unique. This is true because if there is no procedure which can address both of any two variables, then the two variables may have identical address couples. This addressing scheme is practical because two variables which have the same address couples will be contained within two different addressing environments.

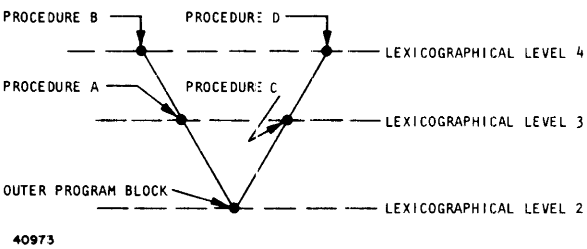


Figure II-2-8. Addressing Environment Tree of ALGOL Program

ADDRESSING ENVIRONMENT LIST

There is a unique set of MSCWs which the D registers must address during the execution of any particular procedure. The D registers must be changed, upon procedure entry or exit, to address the correct MSCWs. The process of changing the D registers is referred to as display update. The list of MSCWs which the D registers address is is the addressing environment list, and the areas of the stack which can be addressed relative to the settings of the D registers are the addressing environment.

STACK HISTORY

The B 7700 stack provides an easily manageable means for keeping line control information (program history) necessary for procedure entry and exit. The stack history list is a list of Mark Stack Control Words, linked together by their DF fields (figure II-2-9).

An MSCW is inserted into the stack as a procedure is entered and is removed as that procedure is exited. Therefore, the stack history list grows and contracts with the procedural depth of the program. Mark Stack Control Words identify the portion of the stack related to each procedure. When the procedure is entered, its parameters and local variables are entered in the stack following the MSCW. When the procedure is executed its parameters and local variables are referenced by addressing relative to the MSCW.

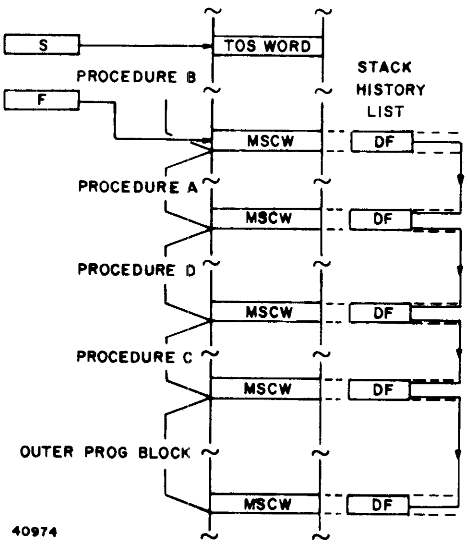


Figure II-2-9. Stack History List

Each MSCW is linked to the prior MSCW through the contents of its DF field in order to identify the point in the stack where the prior procedure began. When a procedure is exited, its portion of the stack is discarded. This action is achieved by setting the stack-pointer register (S) to address the memory location preceding the most recent MSCW (figure II-2-10). This topmost MSCW, addressed by another register (F), is deleted from the stack-history list by changing F to address the prior MSCW, placing this MSCW at the head of the stack history.

SIMPLE STACK OPERATION

All program information must be in the system before it can be used. Input areas are allocated for information entering the system and output areas are set aside for information exiting the system; array and table areas are also allocated to store certain types of data. Thus data is stored in several different areas: the input/output areas, data tables (arrays), and the stack. Since all word is done in the top-of-stack locations, all information or data is transferred to the top-of-stack locations and the stack itself.

At this point, an ALGOL assignment statement and the Polish notation equivalent will be related to the stack concept of operation. The example is Z:=Y + 2x(W+V), where :=

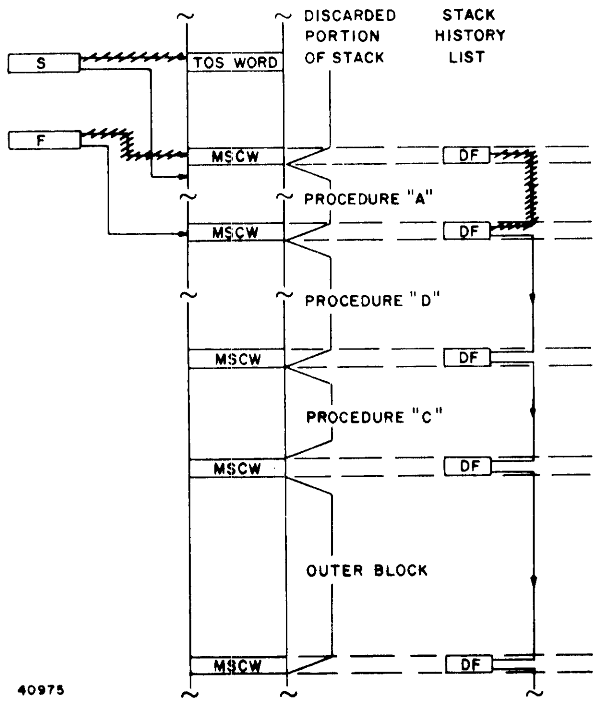


Figure II-2-10. Stack Cut Back on Procedure Exit

means "is replaced by." In terms of a computer program, this assignment statement indicates that the value resulting from the evaluation of the arithmetic expression is to be stored in the location representing the variable Z.

When Z:=Y + 2x(W+V) is translated to Polish notation, the result is ZY2WV+ x +:=. Each element of the example expression causes a certain type of syllable to be included in the machine language program when the source problem is compiled. The following is a detailed description of each element of the example, the type of syllable compiled, and the resulting operation (see figure II-2-11 and table II-2-1).

In the example statement, Z is to be the recipient of a value, so the address of Z must be placed in the stack. This is accomplished by a Name Call (NAMC) syllable which places an Indirect Reference Word (IRW) in the stack. The IRW contains the address of Z in the form of an "address couple" that references the memory location reserved in the stack for the variable Z.

Since Y is to be added to a quantity, Y is brought into the top of the stack as an operand. This is accomplished with a Value Call (VALC) syllable that references Y. The value 2 is then brought to the stack, with an eight-bit literal syllable (LT8). Since W and V are to be added, the respective variables are brought to the stack with Value Call syllables. The ADD operator adds the two top operands and places the sum in the top of stack. This example assumes, for simplicity, single-precision operands not requiring use of additional top-of-stack locations which are used in double-precision operations.

The multiply operator (MULT) is the next symbol encountered in the Polish string; when executed, it places the product "2x(W+V)" in the top of the stack. The next symbol, ADD, when executed, leaves the final result "Y+2x(W+V)" in the top of the stack.

The store syllable (STOD) completes the execution of the statement Z:=Y + 2x(W+V). The store operation examines the two top-of-stack operands looking for an IRW or Data Descriptor. In this example, the IRW addresses the location where the computed value of Z is to be stored. The stack is empty at the completion of this statement.

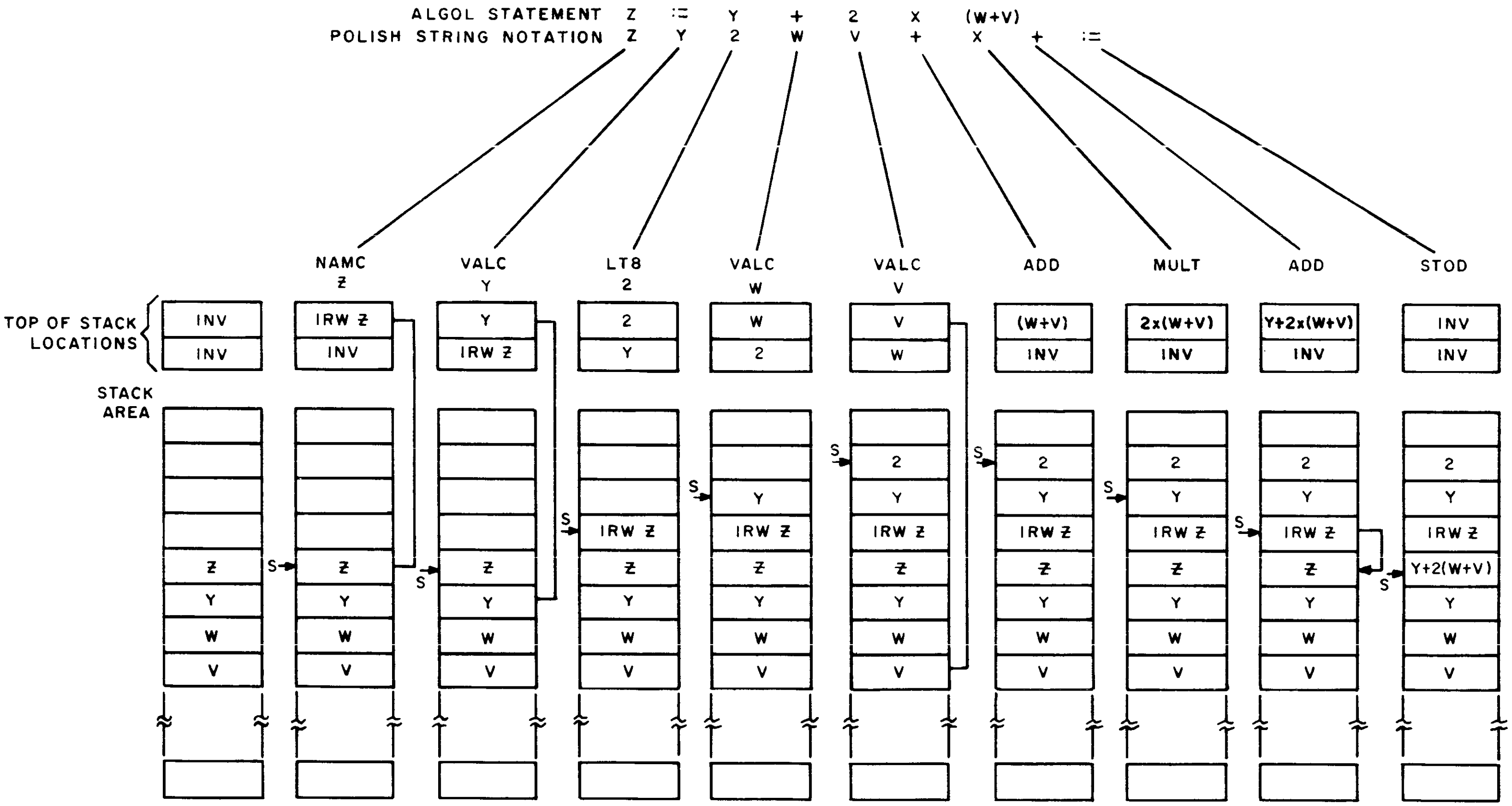
Thus, the Polish string ZY2WV + x+:= is used to produce the following code string:

```
NAMC (Z)
VALC (Y)
LT8 2
VALC (W)
VALC (V)
ADD
MULT
ADD
STOD
```

When this code string is executed on the B 7700, the value of the expression Y+2x(W+V) is stored in the stack location reserved for the variable Z.

INTERRUPT HANDLING

In the B 7700, hardware interrupts are treated as hardware-originated procedure calls. When the hardware detects an interrupt condition, the CPM causes a MSCW to be placed in the stack, then places in the stack an IRW addressing the interrupt handling procedure, places two parameters in the stack to identify and describe the interrupt condition, and then causes the interrupt handling procedure to be entered. When the interrupt handling procedure is entered, the D registers are updated to make all legitimate variables addressable. Similarly, upon return from the interrupt handling procedure, the D registers are again updated to make all of the variables of the former procedure addressable again. A detailed description of interrupt handling is provided in chapter III.



SYLLABLE TYPES

VALC	VALUE CALL	STOD	STORE DESTRUCTIVE
NAMC	NAME CALL	ADD	ADD
LT8	LITERAL (8 BIT)	MULT	MULTIPLY

40976

Figure II-2-11. Stack Operation

Table II-2-1. Description of Stack Operation

Execution Sequence	Polish Notation Element	Syllable Type Compiled	Function of Syllable During Running of the Program
0	–	–	Stack location of program variables illustrated.
1	Z	Name call for Z.	Build an indirect reference word that contains the address of Z and place it in the top of the stack.
2	Y	Value call for Y.	Place the value of Y in the top of the stack.
3	2	Literal 2.	Place a 2 in the top of the stack.
4	W	Value call for W.	Place the value of W in the top of the stack.
5	V	Value call for V.	Place the value of V in the top of the stack.
6	+	Operator add.	Add the two top words in the stack and place the result in the A location as the top of the stack operand.
7	x	Operator multiply.	Multiply the two top-of-stack operands. The product is left in the A location as the top of the stack operand.
8	+	Operator add.	Add the two top words in the stack and leave the result in the A location as the top of the stack operand.
9	:=	Operator store destructive.	Store an item into memory. The address in which to store is indicated by an indirect reference word or a data descriptor. The address can be above or below the item stored.

MULTIPLE STACKS AND RE-ENTRANT CODE

The B 7700 stack mechanism provides a facility for handling several active stacks, which are organized in a tree structure. The trunk of this tree structure is a stack containing MCP global quantities.

LEVEL DEFINITION

As the MCP is requested to run an execution of a program, a level-1 branch of the stack is created. This level-1 branch is a separate stack which contains only the descriptors pointing to the executable code and the read-only data segments for the program. Emerging from this level-1 branch is a level-2 branch, containing the variables and data for this job. Starting from the job's stack and tracing downward through the tree structure, one finds first the stack containing the variables and data for the job (at level 2), the segment descriptor to be executed (at level 1), and the MCP's stack at the trunk (level 0).

RE-ENTRANCE

A subsequent request to run another execution of an already-running program. Thus two jobs which are different executions of the same program have a common node, at level-1, describing the executable code. It is in this way that program code is re-entrant and shared. This results simply from the proper tree-structured organization of the various stacks within the machine. All programs within the system are re-entrant, including all user programs as well as the compilers and the MCP.

JOB-SPLITTING

The B 7700 stack mechanism also provides the facility for a single job to split itself into two independent jobs. A common use of this facility occurs when there is a point in a job where two relatively large independent processes must be performed. This splitting can be used to make full use of a multiprocessor configuration, or to reduce elapsed time by multiprograming the independent processes.

A split of this type establishes a new limb of the tree-structured stack, with the two independent jobs sharing that part of the stack which was created before the split was requested. The process is recursively defined and can happen repeatedly at any level.

STACK DESCRIPTOR

Stack branches are located by an array of descriptors, the stack vector array (figure II-2-12). There is a data descriptor in this array for every stack branch. This data descriptor, the stack descriptor, describes the length of the memory area assigned to a stack branch and its location in either main memory or disk.

A stack number is assigned to each stack branch. The stack number is the index value of the stack descriptor in the stack vector array.

STACK VECTOR DESCRIPTOR

The array size of the stack vector and its location in memory is described by the stack vector descriptor, located in a reserved position of the trunk of the stack (figure II-2-12). All references to stack branches are made through the stack vector descriptor, indexed by the stack number.

PRESENCE BIT INTERRUPT

A Presence Bit Interrupt results when an addressed stack is not present in memory. This Presence Bit Interrupt facility permits stack overlays and recalls under dynamic conditions. Idle or inactive stacks may be moved from main memory to disk as the need arises and, when a stack is subsequently referenced, a Presence Bit Interrupt is generated to cause the MCP to recall the nonpresent stack from disk.

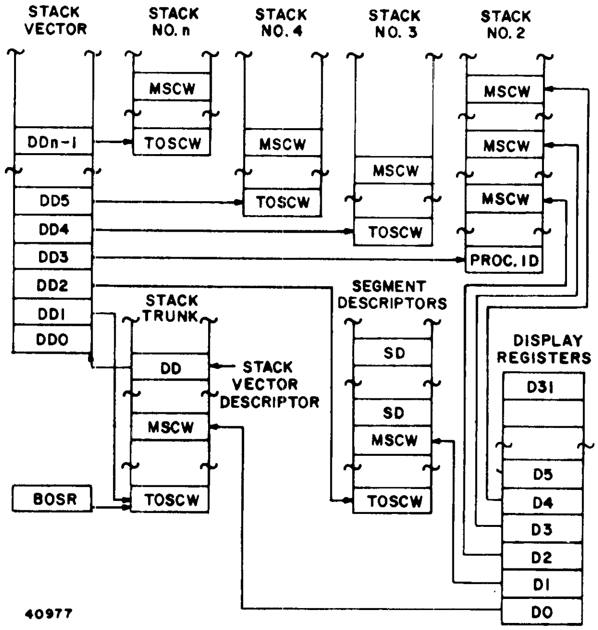


Figure II-2-12. Multiple Linked Stacks

SECTION 3

PROCESSOR WORD FORMATS

GENERAL

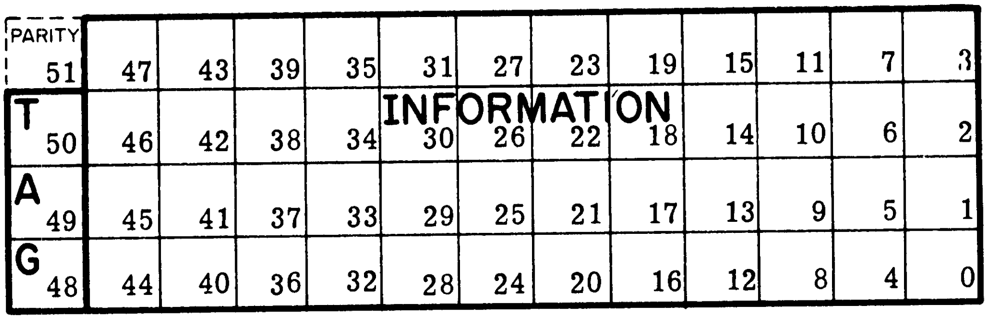
The basic information structure of the B 7700 is the word. As transferred between CPMs or IOMs and core memory, a word consists of 52 bits (see figure II-3-1), and is considered in three parts: a parity bit, which is used to maintain overall parity for the word being transferred; a 3-bit tag field, which indicates the type of information contained within the word, and a 48-bit information field, which contains the actual information.

The tag field not only serves to identify the type of information contained in the word but also can be thought of as an extension of the operator being executed against the word. For example, because the tag field indicates to the arithmetic unit whether the operation involves single precision or double precision operands, a single instruction (ADD) serves both types of operations. In similar fashion, if the sum obtained was a double precision number (requiring two memory words of storage), and the receiving memory word indicates that a single precision operand was resident there, the CPM will round the sum to single precision and then store it.

The tag field also prevents the user from writing over program code or read-only data areas, and prevents him from reading (as data) program code, processor control words, and uninitialized operands.

Consider the bit assignments for the tag field, as illustrated. One can see that words which have bit 48 set, such as IRW's, SIRW's, Segment Descriptors, MSCW's, RCW's, Data Descriptors, and Program Control Words, should not be alterable by the user. The CPM will not allow such words to be modified except by use of the overwrite operators. Words that are used for stack control, MSCW's, RCW's, and PCW's, have bits 49 and 48 of the tag field set. The CPM will not allow such words to be interpreted as operands.

The information field may be used to store data (logical operands, string operands, numeric operands), to store program code (program word), to address data or code outside of the stack (data descriptor, string descriptor, segment descriptor), to address within stacks (indirect reference word, stuffed indirect reference word, stuffed indirect reference word, program control word), to store information re-



PARITY												
51	47	43	39	35	31	27	23	19	15	11	7	3
T 50	46	42	38	34	INFORMATION 30	26	22	18	14	10	6	2
A 49	45	41	37	33	29	25	21	17	13	9	5	1
G 48	44	40	36	32	28	24	20	16	12	8	4	0

Field	Bits	Description
Parity	51:1	Parity bit. Odd parity for the 52 bit word.
Tag	50:3	Value of this field indicates the usage of the information field, as described below.

Tag Value	Information Field Usage
0	Single Precision Operand, Logical Operand, String Operand, Occurs Index Word, Time of Day Function Word
1	Indirect Reference Word, Stuffed Indirect Reference Word
2	Double Precision Operand
3	Mark Stack Control Word, Return Control Word, Top of Stack Control Word, Program Word, Segment Descriptor
4	Step Index Word
5	Data Descriptor, String Descriptor
6	Uninitialized Operand
7	Program Control Word

Field	Bits	Description
INFORMATION	47:48	Use of this field depends on the value of the tag field.

40978

Figure II-3-1. Basic Word Format

garding stack history (mark stack control word, return control word, top of stack control word), or to provide a parameter for use with certain operators (step index word, and occurs index word. Data words (operands) and program words were described in the previous sections of this chapter. The other various processor words are described in this section.

WORDS FOR ADDRESSING OUTSIDE OF THE STACK

There are three types of descriptors which are used for addressing data or code which is not resident in the stack. The type of descriptor is directly related to the data or code being referenced. Thus, a segment descriptor will always address a segment of program code (contained in program words), a string descriptor will always address a string operands), and a data descriptor will address an array of word operands.

The ADDRESS field in each of these descriptors is 20 bits in length; this field contains the absolute address of an array in either system main memory or in the backup disk file, as indicated by setting of the Presence bit (P). The referenced data is in main memory when the presence bit is set.

PRESENCE BIT

A Presence Bit Interrupt occurs when the job references data by means of a descriptor in which the P-bit is equal to 0; i.e., the data is located in a disk file, rather than in main memory. The Master Control Program (MCP) recognizes the Presence Bit Interrupt and transfers data from disk file storage to main memory. After the data transfer to main memory is completed, the MCP marks the descriptor present by setting the P-bit to 1, and places the new main memory address into the address field of the descriptor. The interrupted job is then reactivated.

INDEX BIT

A Data Descriptor describes either an entire array of data words, or a particular element within an array of data words. If the descriptor describes the entire array, the Index bit (I-bit) in the descriptor is 0, indicating that the descriptor has not yet been indexed. The length field of the descriptor defines the length of the data array.

INVALID INDEX

A particular element of an array is described by indexing an array descriptor. Memory protection is ensured during indexing operations by performing a comparison between the length field of the descriptor and the index value. An Invalid Index Interrupt results if the index value exceeds the length of the memory area defined by the descriptor, or if the index is less than 0.

VALID INDEX

If the index value is valid, the length field of the descriptor is replaced by the index value, and the I-bit in the descriptor is set to 1 to indicate that indexing has taken place. The address and index fields are added together to generate the absolute machine address whenever an indexed Data Descriptor in which the P-bit is set is used to fetch or store data.

The Double-Precision bit (D) is used to identify the referenced data as single-or double-precision and directly affects the indexing operation. The D-bit equal to 1 signifies double-precision and causes the index value to be doubled before indexing.

READ-ONLY BIT

The Read-Only bit (R) specifies that the memory area described by the Data Descriptor is read-only area. If the R-bit of a descriptor is set to 1, and the area referenced by that descriptor is used for storage purposes, an interrupt results.

COPY BIT

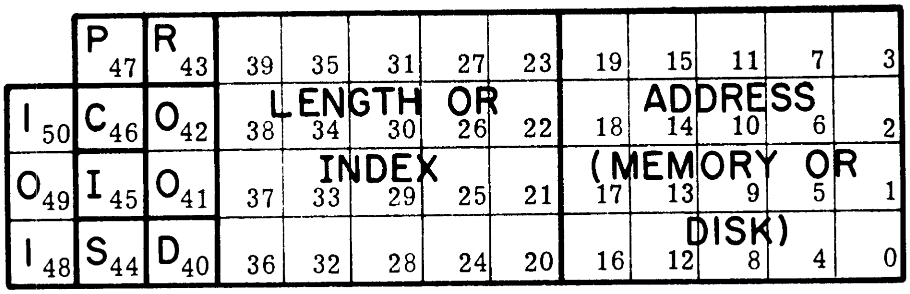
The Copy bit (C) identifies a descriptor as a copy of a master descriptor and is related to the presence-bit action. The copy bit links multiple copies of an absent descriptor (i.e., the presence bit is off) to the one master descriptor. The copy bit mechanism is invoked when a copy is made in the stack. If it is a copy of the original, absent descriptor, the processor sets the copy bit to 1 and inserts the address of the master descriptor into the address field. Thus, multiple copies of absent data descriptors all point back to the master descriptor.

DATA DESCRIPTOR

Data descriptors refer to data areas, including input/output buffer areas. The data descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in operands is contained in the length field of the descriptor. Data descriptors may directly reference any memory word address from 0 through 1,048,576. The structure of the data descriptor is illustrated in figure II-3-2.

STRING DESCRIPTOR

String descriptors refer to strings of 4-bit digits, 6-bit or 7-bit characters, or 8-bit bytes.



Field	Bits	Description
Tag	50:3	Tag field. Value of five.
P	47:1	Presence bit. Indicates the presence or absence of data in main memory. A 0 causes a presence bit interrupt whenever the descriptor is used by a processor to obtain non-present data. A 1 indicates that the data described is in main memory.
C	46:1	Copy bit. A 0 indicates that this is the original descriptor for the particular data area. A 1 indicates that this descriptor is a copy of the original descriptor.
I	45:1	Indexed bit. A 0 indicates that an indexing operation is required before the descriptor may be used to obtain data. A 1 indicates that indexing has already taken place and the index value is stored in bit positions 39:20 (Length or Index).
S	44:1	Segmented bit. A 0 indicates that the data is not segmented. A 1 indicates that the data is divided into segments.
R	43:1	Read-only bit. A 0 indicates that the data may be referenced for reading or writing. A 1 indicates that the data can only be referenced for reading.
	42:2	Size field, must be 0 to indicate a data descriptor.
D	40:1	Double-precision bit. A 0 indicates single-precision operands, a 1 indicates double-precision operands.
Length or Index	39:20	This field contains either the length (in operands) of the memory area (if bit 45 = 0) or an index value (if bit 45 = 1). If bit 45 equals 0, the descriptor has not been indexed. This field is used for size checking during the indexing operation. If bit 45 equals 1, the descriptor has been indexed. For a double-precision operation, the index is doubled after index size checking, and the result is stored in the index field.
Address (Memory or Disk)	19:20	This field contains either a main memory or disk address. If the presence bit (bit 47) equals 1, this field contains the memory address of data. If the presence bit equals 0 and the copy bit (bit 46) equals 0, this field contains the disk address of the data. If the presence bit equals 0 and the copy bit equal 1, this field contains the memory address of the original descriptor.

Figure II-3-2. Data Descriptor

The string descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in characters is contained in the length field of the descriptor. The structure of the String Descriptor is illustrated in figure II-3-3.

SEGMENT DESCRIPTORS

Segment descriptors refer to areas of program code. The descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in program words is contained in the length field of the descriptor. The structure of the segment descriptor is illustrated in figure II-3-4.

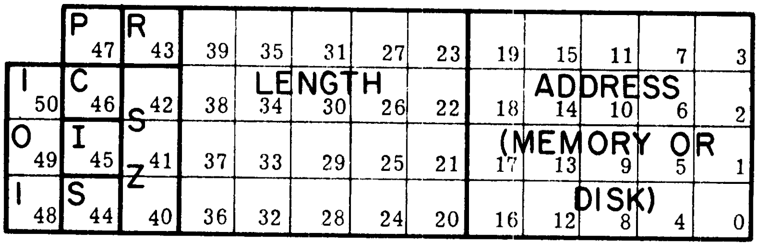
WORDS FOR ADDRESSING WITHIN STACKS

There are three types of words which are used for addressing data or descriptors which are resident within a stack. A Program Control Word is used, at the time of procedure entry, to locate a segment descriptor (and the proper word and syllable of code) for the procedure. An Indirect Reference Word is used to address within the current addressing environment of a procedure. A Stuffed Indirect Reference Word is used to address outside the current addressing environment of a procedure.

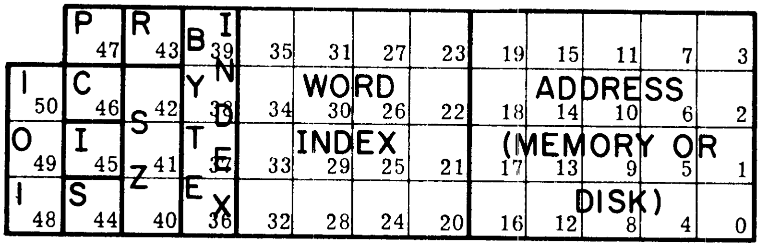
PROGRAM CONTROL WORD

The Program Control Word (PCW), and the MSCW are used during entry into a procedure. The organization of the PCW is illustrated in figure II-3-5 and contains the following:

STRING DESCRIPTOR (NON-INDEXED)



STRING DESCRIPTOR (INDEXED)



Field	Bits	Description
Tag	50:3	Tag field. Value of five.
P	47:1	Presence bit. A 0 causes a presence bit interrupt if the descriptor is used to access data. A 1 indicates the data is present in main memory.
C	46:1	Copy bit. A 0 indicates that this is the original descriptor for the particular data area. A 1 indicates that this descriptor is a copy of the original descriptor.
I	45:1	Indexed bit. A 0 indicates indexing is required. A 1 indicates that indexing has taken place and the word and character index are in the WORD INDEX and BYTE INDEX fields.
S	44:1	Segmented bit. A 0 indicates that the data area is not segmented. A 1 indicates that the data is segmented.
R	43:1	Read only bit. A 0 indicates that the data may be referenced for reading or writing. A 1 indicates that the data can be read only.
SZ	42:3	Size field. 100 indicates character size of 8-bit bytes, 101 indicates 7-bit ASCII characters, 011 indicates 6-bit characters, and 010 indicates 4-bit digits.
Length	39:20	Bits 39:20, contain either the length of the memory area (bit 45=0) or an index value (bit 45=1). When bit 45 equals 0, this field contains the length of the area in digits, characters or bytes.
Byte Index	39:4	Byte index (Bit 45=1).
Word Index	35:16	Word Index (Bit 45=1).
Address (Memory or Disk)	19:20	This field contains either a main memory or a disk address. If the presence bit (bit 47) is 1, the field contains a memory address of the data. If both the presence bit and the copy bit (bit 46) are equal to 0, the field contains the disk address of the non-present data. If the presence bit is 0 and the copy bit is 1, the field contains the memory address of the original descriptor.

Figure II-3-3. String Descriptor

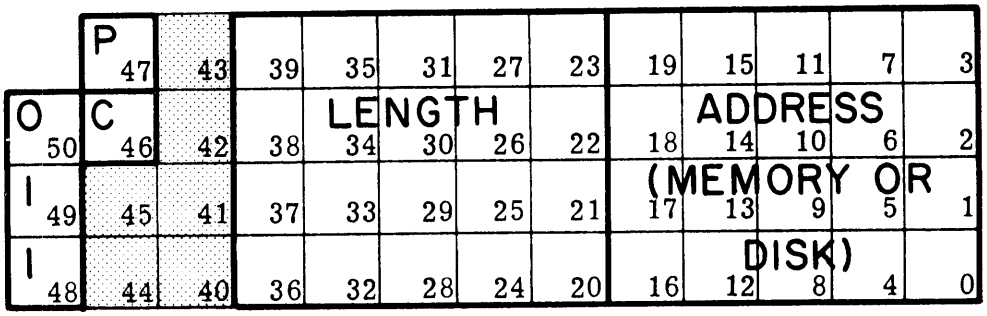
INDIRECT REFERENCE WORD

Referencing a variable within the current addressing environment of a procedure is accomplished through the address couple in the Indirect Reference Word (IRW). References are relative to the D register specified by the address couple. The format of the IRW is shown in figure II-3-6.

STUFFED INDIRECT REFERENCE WORD

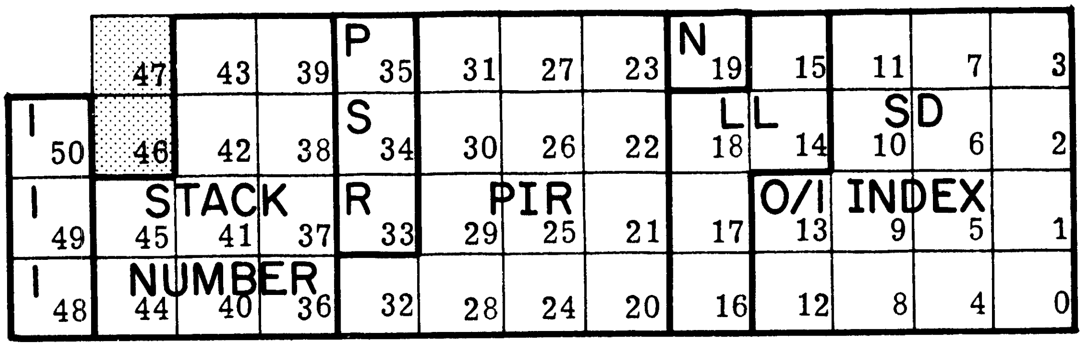
Reference to variables outside the current environment is accomplished by a Stuffed Indirect Reference Word. This addressing is relative to the base of the stack in which the variable is located.

The SIRW contains the stack number, the location (DISP) of the MSCW, and the index to



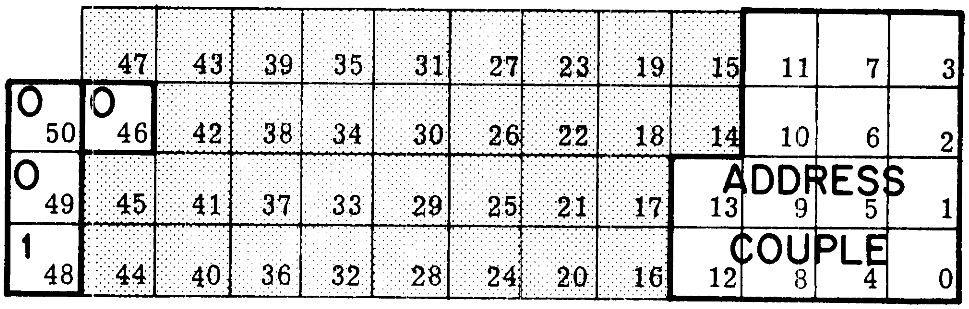
Field	Bits	Description
Tag	50:3	Tag field. Value equals three.
P	47:1	Presence bit. A 0 indicates that the segment is absent from main memory.
C	46:1	Copy bit. A 0 indicates that this is the original segment descriptor. A 1 indicates that this is a copy of the original segment descriptor.
	45:4	Not used. Unused bits may be either 0 or 1.
Length	39:20	The length of the program segment in words.
Address (Memory or Disk)	19:20	This field contains either the main memory address or the disk file address. If the presence bit (bit 47 equals 1, the field contains the main memory address of the program segment. If both the presence bit and the copy bit (bit 46) equal 0, the field contains the disk address of the non-present program segment. If the presence bit equals 0 and the copy bit equals 1, the field contains the absolute memory address of the original program segment descriptor.

Figure II-3-4. Segment Descriptor

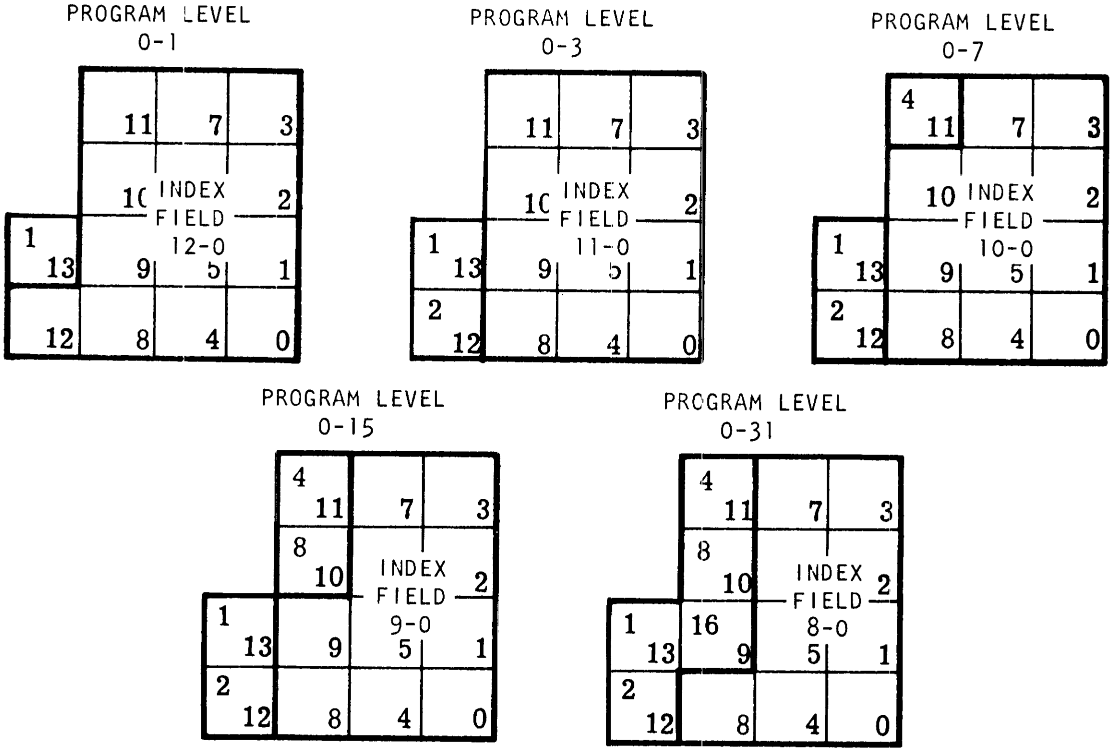


Field	Bits	Description
Tag	50:3	Tag field. Value equals seven.
	47:2	Not used.
Stack Number	45:10	The number of the stack which contains the PCW.
PSR	35:3	The program syllable (0-5) within the word located by PIR.
PIR	32:13	Index to the Program Base Register. Locates a word within the code segment.
N	19:1	Normal state (0) or control state (1).
LL	18:5	The level of the procedure being entered.
SD Index	13:14	The segment descriptor index. Bits 12 through 0 specify the value to be added to either D-register 0 or 1. When bit 13 equals 0, D-register 0 is selected; when bit 13 equals 1, D-register 1 is selected. The sum of the contents of the display register and the index locates a segment descriptor.

Figure II-3-5. Program Control Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals one.
	47:1	Not used
	46:1	Environment bit. Must equal zero for an IRW. (1 = SIRW).
	45:32	Not used.
Address Couple	13:14	Selects D Register (According to current program level as indicated by rLL) and provides index value (see below).



NOTE: THE BIT ORDER OF THE LL FIELD IS INVERTED.

40983

Figure II-3-6. Indirect Reference Word

the variable relative to the MSCW. The absolute memory location of the variable is formed by adding the contents of DISP and index to the base address of the referenced stack from the stack descriptor. The contents of the SIRW (with the exception of index) are dynamic and are accumulated as the program is executed. The stack number and DISP fields are entered into the SIRW by the Stuff Environment (STFF) operator. The bit format of the SIRW is shown in figure II-3-7.

WORDS FOR STORING STACK HISTORY

Certain words can be thought of as words used for storing stack history. These words, used for procedure entry and exit, as well as for storing the stack state for inactive stacks, include the Mark Stack Control Word, the Return Control Word, and the Top Of Stack Control Word.

MARK STACK CONTROL WORD

The Mark Stack Control Word (MSCW), together with the Return Control Word (RCW), provides a linking mechanism for the history of previous control-register settings through the stack.

The MSCW is placed in the stack by the Mark Stack operator. The MSCW is organized as illustrated in figure II-3-8.

RETURN CONTROL WORD

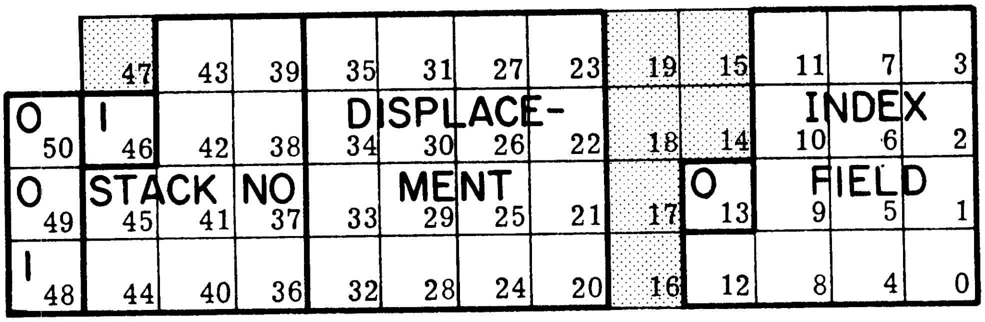
The Return Control Word (RCW) and the MSCW are used for subroutine handling. The Return Control Word stores the environment to which the subroutine will return. The organization of the RCW is illustrated in figure II-3-9.

TOP OF STACK CONTROL WORD

The Top Of Stack Control Word (TOSCW) contains all information needed to restore the operating environment when a stack (or process) is activated. When a stack is active, the first word of the stack is a single precision operand containing the processor ID (a number, 0 through 7). When the stack is made inactive, the processor ID is changed to a TOSCW, containing the status of various processor flip-flops necessary to restore the stack's environment when it is again activated. The TOSCW is created by the Move Stack (MVST) operator. The TOSCW is illustrated in figure II-3-10.

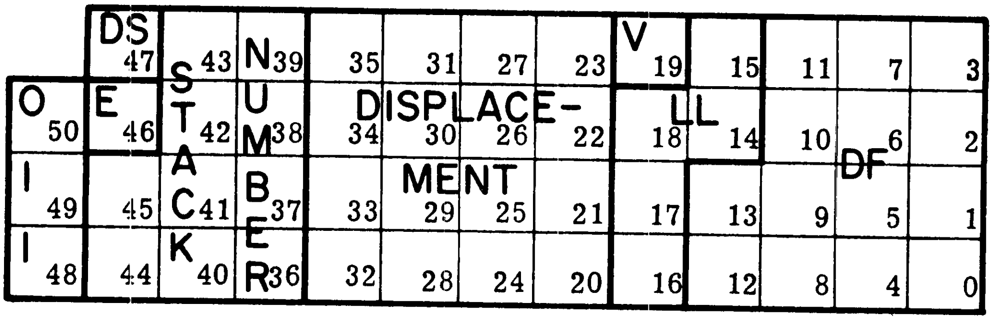
WORDS USED AS SPECIAL PARAMETERS

Certain control words are used only as a parameter to a single operator. Among these are the Step Index Word, used with the Step and Branch operator; the Occurs Index Word, used with the Occurs Index operator; and the Read Time Of Day Function Word, used with the Scan In operator.



Field	Bits	Description
Tag	50:3	Tag field. Value equals one.
	47:1	Not used.
	46:1	Environment bit. Must be a one (0=1RW).
Stack No.	45:10	The number of the stack containing the referenced word.
Displacement	35:16	This number, added to the stack base address, addresses an MSCW.
	19:6	Not used.
	14:1	Must be 0.
Index Field	12:13	This number, added to the address of the MSCW, addresses the referenced word.

Figure II-3-7. Stuffed Indirect Reference Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals three.
DS	47:1	Different-stack bit. A 0 indicates that the stack-number field refers to the current stack. A 1 indicates that the stack-number field refers to a different stack.
E	46:1	Environment bit. A 0 indicates an inactive MSCW, generated directly by the Mark Stack operator. The procedure entry has not been performed. A 1 denotes an active MSCW generated upon entry into a procedure, at which time the environment fields (stack number, displacement, value, and LL fields) are stored into the MSCW.
Stack Number	45:10	Stack-number field. Contains the number of the stack from which the PCW was obtained at procedure-entry.
Displacement	35:16	Displacement field. When added to the stack base address, locates the MSCW of the prior lexicographic level.
V	19:1	Value bit. A 0 indicates that the MSCW was generated during any operation that will be restarted from the beginning. A 1 indicates that the operator must continue after the Exit or Return which refers to this MSCW (e.g., an accidental entry by a Value Call).
LL	18:5	LL field. Denotes the lexicographical level at which the program will run when the procedure is entered.
DF	13:14	Denotes the stack history. This field is used to locate, in the stack, the preceding MSCW (i.e., the previous "F" register setting).

Figure II-3-8. Mark Stack Control Word

STEP INDEX WORD

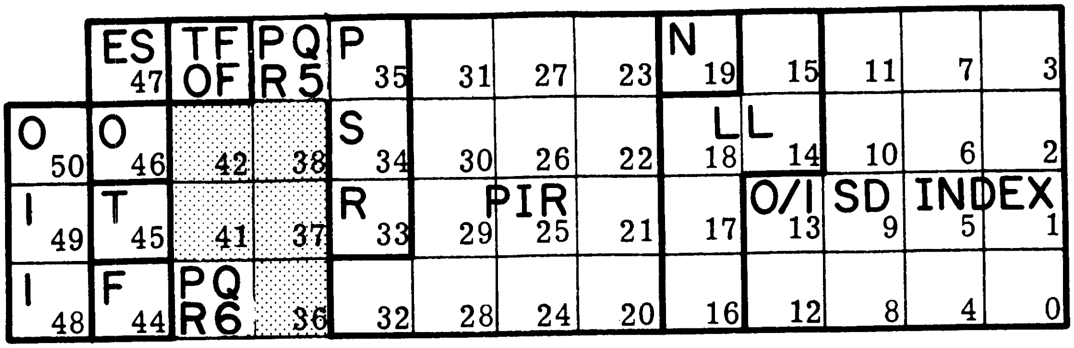
The Step Index Word (SIW) is used as a parameter to the Step and Branch operator, to increase the efficiency of this operator in iteration loops. When the Step and Branch operator is invoked, the SIW addressed by the IRW in the top of stack location is located. The increment field is added to the current value field. If the current value field is then greater than the final value field, PIR and PSR are set from the next two syllables in the program code string and the branch is made. If the current value field is not greater than the final value field, PIR and PSR are advanced three syllables, the SIW is replaced in memory, and the iteration loop continues. The format of the SIW is illustrated in figure II-3-11.

OCCURS INDEX WORD

The Occurs Index Word (OIW) is used to index a field within an array. COBOL permits arrays to be constructed of a series of fields of a specified character size (through use of the OCCURS clause). This series of fields may not necessarily begin at a word boundary, because the array may be one of several items subordinated under a group item. The OCRX operator, together with an OIW in the A location and an index value in the B location, is used to calculate a new index value which is left in the top of the stack. The original index value is an integer which indicates the relative position of the desired field within the array. The new index value is the displacement (in characters) of the desired field from the first character of the array. The character size (specified in a descriptor) and the index value (left in the top of stack) can then be used to address the desired field. The format of the OIW is shown in figure II-3-12.

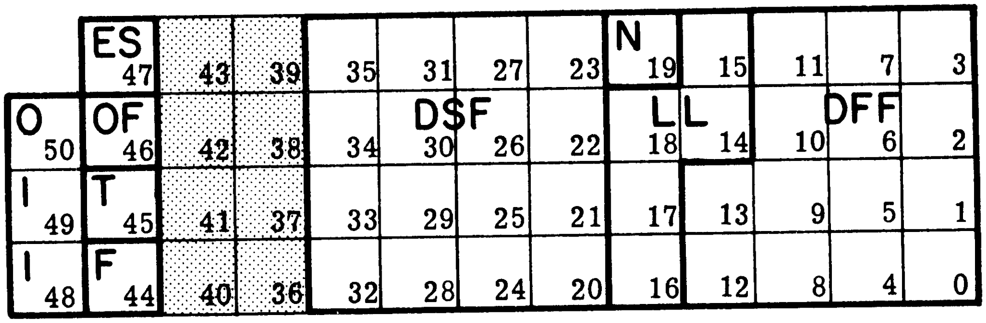
TIME OF DAY FUNCTION WORD

This word is used as a parameter to the Scan In operator, to specify that the time of day is to be interrogated by the MCP. The format of the Time of Day Function Word is shown in figure II-3-13.



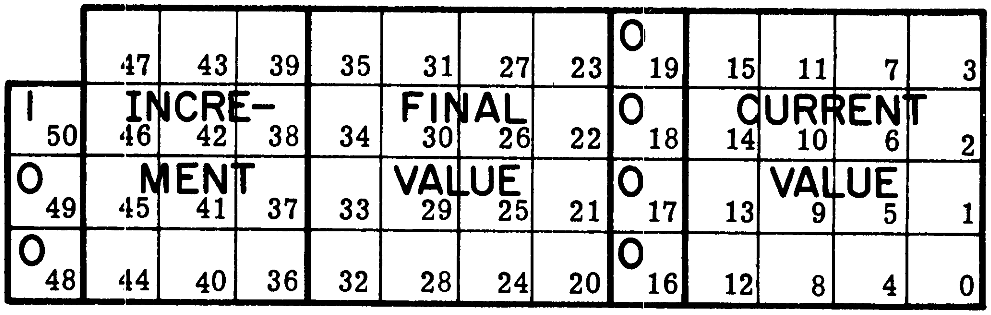
Field	Bits	Description
Tag	50:3	Tag field. Value of three.
ES	47:1	External Sign flip-flop.
O	46:1	Overflow flip-flop.
T	45:1	True/False flip-flop.
F	44:1	Float flip-flop.
TFOF	43:1	True/False flip-flop occupied flip-flop.
PQR6, PQR5	40:1 39:1	Special hardware bits. These bits are used for controlling display update operations in the processor.
PSR	35:3	Program syllable of the operator to be executed after return from the subroutine.
PIR	32:13	PIR setting of the operator to be executed next in the calling routine.
N	19:1	Normal state (0) or control state (1) procedure.
LL	18:5	Level of the calling procedure when the RCW was generated (at procedure entry).
SD Index	13:14	Segment descriptor index. Bits 12 through 0 specify the value to be added to either D-register 0 or 1. When bit 13 = 0, D-register is selected; when bit 13 = 1, D register 1 is selected. The sum of the contents of the selected display register and the index locates a segment descriptor.

Figure II-3-9. Return Control Word



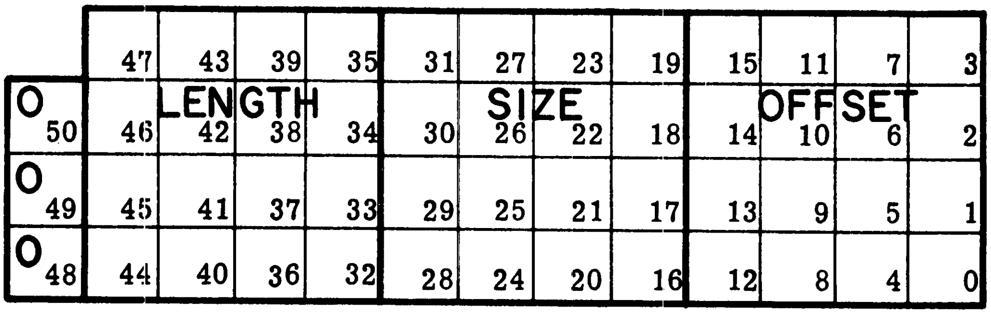
Field	Bits	Description
Tag	50:3	Tag field. Value equals three.
ES	47:1	External sign flip-flop.
OF	46:1	Overflow flip-flop.
T	45:1	True/False flip-flop.
F	44:1	Float flip-flop.
	43:8	Not used.
DSF	35:16	Delta S-register field. The value of S-register displacement above BOSR.
N	19:1	Normal-control state flip-flop. 0 = normal; 1 = control state.
LL	18:5	Lexicographic level.
DFF	13:14	Delta F-register field. The value of F-register displacement below the S-register.

Figure II-3-10. Top of Stack Control Word



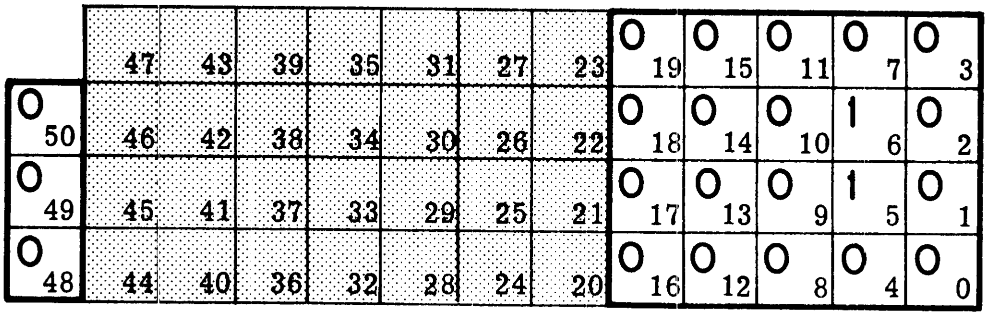
Field	Bits	Description
Tag	50:3	Tag field. Value equals four.
Increment	47:12	Increment: value to be added to current value field.
Final Value	35:16	Final value: value used to terminate the iteration loop.
	19:4	Must be 0 for S1W.
Current Value	15:16	Current value or count. The branch is made if this field is greater than the final value field.

Figure II-3-11. Step Index Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals zero.
Length	47:16	The length, in characters, of each field in the array.
Size	31:16	The size, in fields, of the array.
Offset	15:16	The number of characters preceding the first field of the array.

Figure II-3-12. Occurs Index Word



Field	Bits	Description
Tag	50:3	Tag field. Value equal zero.
	47:28	Not used.
	19:13	Must equal zero.
	6:2	Must equal three.
	4:5	Must equal zero.

Figure II-3-13. Time-of-Day Function Word

SECTION 4

INPUT/OUTPUT SUBSYSTEM MAP STRUCTURE

INTRODUCTION

The B 7700 Input/Output Modules (IOM) operate in parallel with the Central Processor Modules (CPM). The purpose of the IOM is to control all data transfers between main memory and peripheral devices, or between two peripheral devices, so that the CPM is released from I/O operations at the earliest possible moment. In brief, the IOM controls not only the selection of I/O requests from lists of such requests in main memory, but also path selection to the desired devices, the initiation of requests on the appropriate device, the transfer of data as specified by the requests, and the construction of a list of completed requests in main memory. The CPM, on the other hand, builds the I/O request, places it in the appropriate list in main memory, notifies the IOM of the presence of the request (if this is the only request for the device), and then is free to continue with other processing. Routinely, the CPM checks memory for the presence of completed I/O requests and processes the completed requests.

Each IOM is, in effect, a separate computer with its own local memory, logic, arithmetic, and communication capabilities. This independent processing capability permits the IOM to perform routine input-output tasks without interrupting the CPM. Thus the IOM can control transfers of data between peripheral storage devices and main memory or other storage devices without direct supervision of the CPM. In fact, parallelism within the IOM permits it to initiate, service, and terminate data transfers for several users while the CPM is processing data for yet another user.

QUEUE-DRIVEN I/O

To allow the IOMs to properly select paths to the devices and to service I/O requests, certain structures are created by software when the system is initialized. These structures, which provide a mechanism to allow the CPMs to queue I/O requests, allow each IOM to be aware of the requests, of the devices it can service, and of the order of priority of devices served by an exchange. These structures are referred to as the I/O Subsystem Map, and hence this type of I/O is often reffered to as "map" I/O or "queue-driven" I/O. Because the use of the map allows the IOM to process many I/O operations in parallel, independent of CPM, I/O performed using the map is also known as asynchronous I/O. The IOM may also operate synchronously to process one I/O request at a time; however, such synchronous operation is used only for special applications such as system initialization and is not further discussed in this section.

The operation of asynchronous I/O is illustrated in simplified form in figure II-4-1. When the I/O subsystem map is initialized the CPM places information about each peripheral device and the paths to it into a table in memory. During operation, the I/O subsystem map is accessed by both the CPM and IOM as I/O requests are built (by the CPM) and processed (by the IOM). In essence, the CPM builds I/O requests and places them in queues of such requests in main memory. Each request specifies the desired I/O operation and the device on which the operation is to be performed. The IOM extracts requests from these queues on a first-in first-out basis, processes each request, and places the completed requests into a queue in main memory.

Periodically, the CPM extracts the completed requests from the queue in main memory and takes the necessary action to check them.

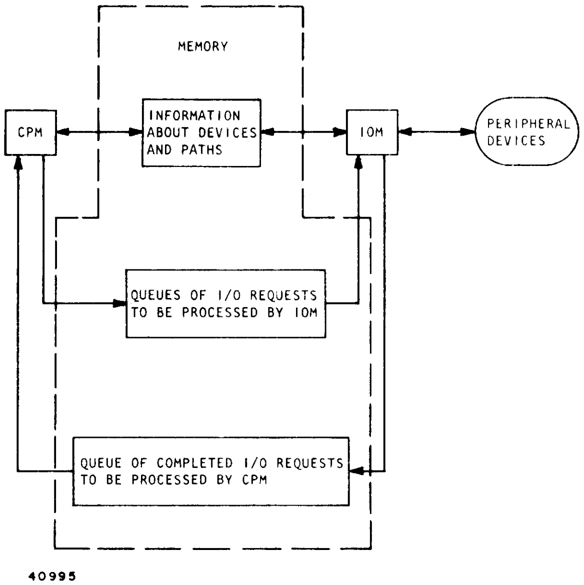

MEMORY
CPM
INFORMATION ABOUT DEVICES AND PATHS
IOM
PERIPHERAL DEVICES
QUEUES OF I/O REQUESTS TO BE PROCESSED BY IOM
QUEUE OF COMPLETED I/O REQUESTS TO BE PROCESSED BY CPM
40995

Figure II-4-1. Asynchronous I/O Operation, Simplified Block Diagram

Once the IOM is notified (by the CPM) of the presence of an I/O request in one of the input queues, all requests in that queue will be processed by the IOM independent of CPM actions until the queue becomes empty. The CPM may place additional requests into a queue while the IOM is processing a request from the queue. Thus, once the IOM starts processing a queue, the CPM may process other programs, queue new I/O requests, and perform computations; effectively masking out the IOM transfer times.

ERROR HANDLING

From time to time conditions may arise which prevent I/O operations from being accomplished successfully. A printer may run out of paper, a card punch may be out of cards, or a device may for some reason not be ready. The design of the I/O subsystem map allows the IOMs to continue to process requests for other devices even though an error is detected on a particular device. When the error is recognized by an IOM, processing of further requests for the particular device is suspenended, the I/O request is marked as containing an error, and that marked request is linked into the queue of completed requests. The CPM is not interrupted to handle the error; however, when the CPM does process the queue of completed requests it will recognize and process the error. When the error has been processed, the CPM can again cause the IOM to process requests for the device on which the error was detected.

If such a strategy were to be applied to the handling of all input/output errors a catastrophic situation might arise. If, say, the IOM itself were the source of the error, it is conceivable that it could then process all (or many) I/O requests erroneously. However, in such cases the B 7700 IOM stops all processing of I/O requests (for all devices) and immediately interrupts the CPM. In short, I/O errors associated with a particular device cause processing of further requests for the device to be halted but allow the processing of requests for other devices by the IOM to continue. I/O errors which can be associated only with an IOM and not with a particular device cause the IOM involved to stop all processing of requests (other IOMs are not affected) and causes the system to be interrupted so that the IOM error may be processed. Provision is also made to allow the software to request that the system be interrupted when a particular I/O request is completed.

DEFERMENT OF PATH BINDING

The I/O subsystem map allows the IOM to select the transfer path for a device as the path becomes available. This dynamic path selection is logically similar to the call routing of a long distance telephone network; that is, the route of the call is selected based on the locations of the correspondents and the available paths. The user need only be concerned about the type of device to be used (card reader, magnetic tape, disk file, etc.); the MCP will associate the logical file with a physical device when the program is executed, and the IOM, when it initiates each transfer of data, will select an available transfer path to the device.

Maximum I/O throughput can be realized only if the binding of the data path between an IOM and a device is delayed until the device is ready to initiate the job. As shown in figure II-4-2, if device 4 is to be initiated, the path required to connect CPM 1 with device 4 involves selecting between two IOM's and between two channels within each IOM. (The peripheral controls have been excluded from this figure because they do not affect the concepts

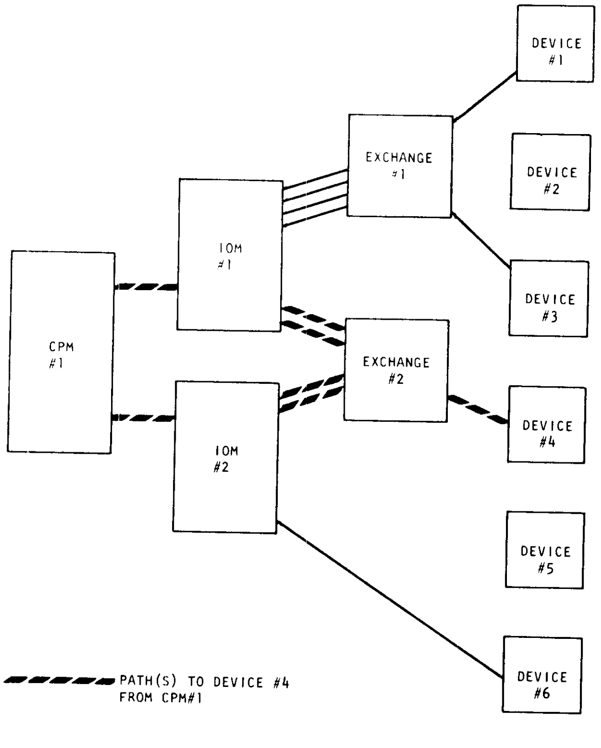


Figure II-4-2. Data Transfer Path Selection

being described. For purposes of this discussion the peripheral controls may be thought of simply as extensions of the IOMs.) If the path to device 4 were to be preselected programmatically, a situation could develop in which the device is free but the preselected path is not. Thus, execution of the request would be unnecessarily delayed if in fact an alternate path to the device was available.

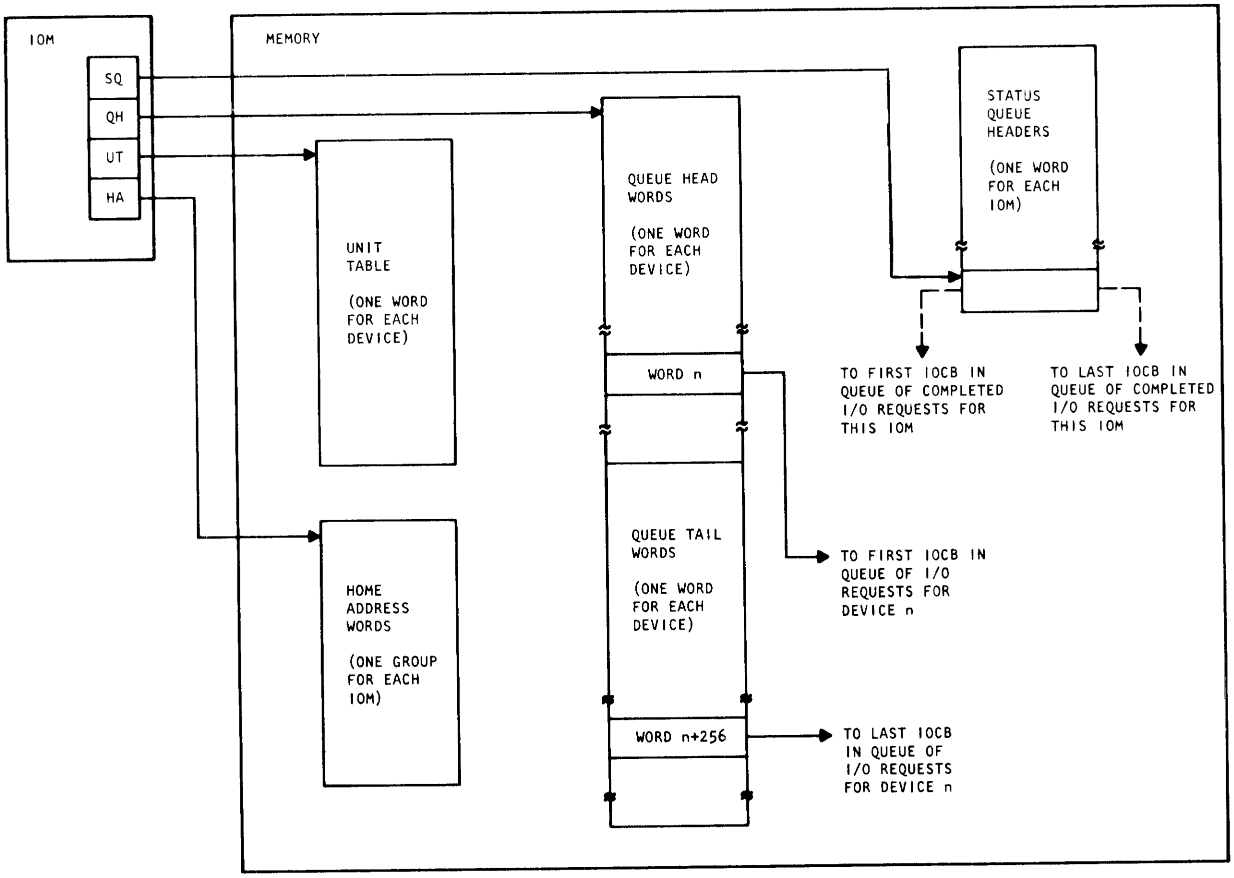
To delay binding the path programmatically generally would require that the CPM which initiated the job be involved in the operation until the request is actually initiated on the specified device. The I/O subsystem map, however, allows the IOM's to manage selection and binding of paths, allowing the CPM's to be free to do other processing. Thus, because the IOM processes I/O requests without CPM intervention, and because the IOM selects data paths at the time of execution, the total system time required to accomplish an I/O operation is limited to the amount of time required for a CPM to build an I/O request and place it in memory.

I/O SUBSYSTEM MAP

As shown in figure II-4-3, the I/O subsystem map is made up of four major software structures in main memory. These four software structures are addressed by registers within the IOM: the Home Address words are addressed by the HA register; the Unit Table is addressed by the UT register; the Queue Head and Queue Tail words table is addressed by the QH register; and the Status Queue Header is addressed by the SQ register. The IOM uses the Queue Head word for the appropriate device to locate the I/O request. Thus, the IOM can locate any element of the map as necessary. Of course, since the map is constructed by the MCP, it too is aware of the location of each element of the map.

COMMANDS AND REQUESTS

Before further discussion of the I/O subsystem map can take place, the difference between an I/O command and an I/O request must be made clear. An I/O command is an or-


IOM
SQ
QH
UT
HA
MEMORY
UNIT TABLE
(ONE WORD FOR EACH DEVICE)
HOME ADDRESS WORDS
(ONE GROUP FOR EACH IOM)
QUEUE HEAD WORDS
(ONE WORD FOR EACH DEVICE)
WORD n
QUEUE TAIL WORDS
(ONE WORD FOR EACH DEVICE)
WORD n+256
STATUS QUEUE HEADERS
(ONE WORD FOR EACH IOM)
TO FIRST IOCB IN QUEUE OF COMPLETED I/O REQUESTS FOR THIS IOM
TO LAST IOCB IN QUEUE OF COMPLETED I/O REQUESTS FOR THIS IOM
TO FIRST IOCB IN QUEUE OF I/O REQUESTS FOR DEVICE n
TO LAST IOCB IN QUEUE OF I/O REQUESTS FOR DEVICE n

40997

Figure II-4-3. I/O Subsystem Map, Simplified Block Diagram

der to an IOM which can cause one operation or many operations for one device to be initiated by the IOM. Although there are special I/O commands which control but a single I/O operation, the I/O command most often associated with asynchronous I/O is the Start IO command, which causes the IOM to process I/O requests from a queue of such requests until the queue is empty. Each I/O request contains information describing a single input or output operation that is used not only by the IOM but also by the peripheral control and even the device itself. Each I/O request is made up of several words and is known as an I/O Control Block (IOCB). The IOCB is discussed in detail later in this section; I/O commands are described in Chapter IV.

MAP INTEGRITY

Because the I/O subsystem may be accessed and modified by all CPM's and IOM's in the I/O subsystem, the integrity of the map is protected by special lock bits and lock words. This system of locks prevents conflicts between the IOM's and CPM's which use and modify the map. As shown in figure II-4-4, the system consists of three types of locks; a lock bit and a lock word for each group of Home Address words, and a lock bit for each Unit Table word.

The software lock word prevents two or more CPM's from attempting to build I/O commands in the Home Address words simultaneously. This word must be unlocked before a CPM can access the Home Address words; the CPM will immediately lock this word when it gains access.

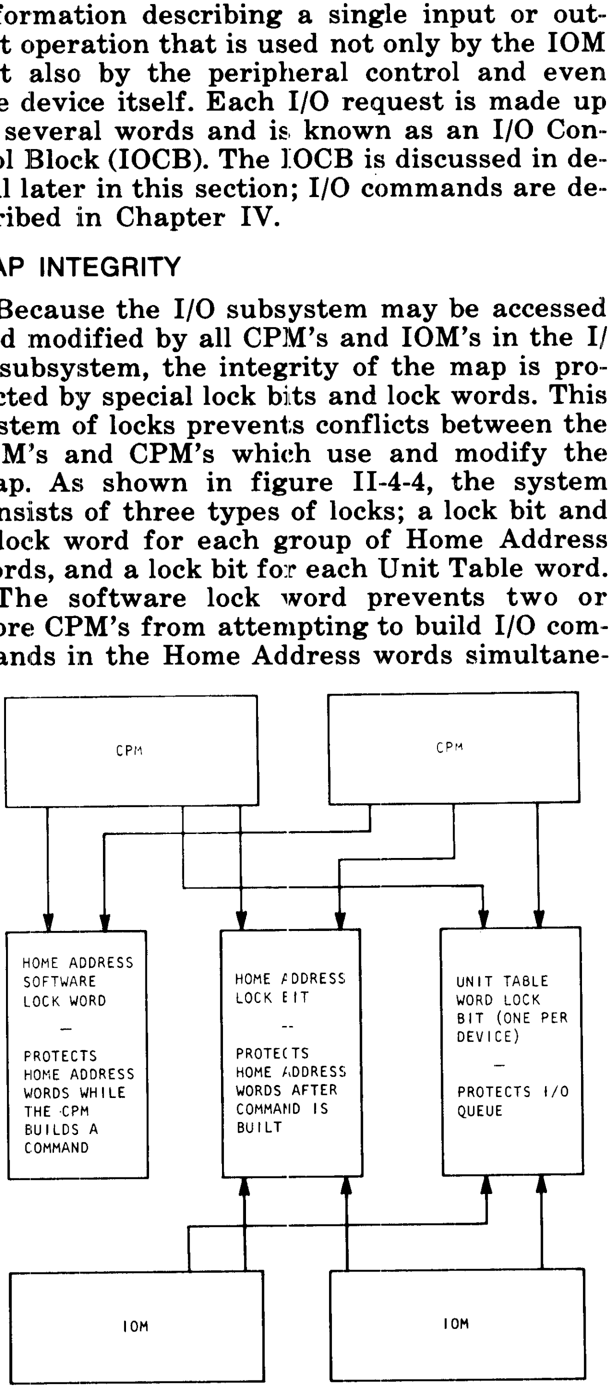

CPM
CPM
HOME ADDRESS SOFTWARE LOCK WORD
–
PROTECTS HOME ADDRESS WORDS WHILE THE CPM BUILDS A COMMAND
HOME ADDRESS LOCK BIT
--
PROTECTS HOME ADDRESS WORDS AFTER COMMAND IS BUILT
UNIT TABLE WORD LOCK BIT (ONE PER DEVICE)
–
PROTECTS I/O QUEUE
IOM
IOM

Figure II-4-4. I/O Subsystem Map Protection

The Home Address lock bit prevents a command from being altered once it has been placed in the Home Address words for execution. The CPM locks this bit when a command is placed in the Home Address words.

In response to a channel interrupt, the IOM exchanges the contents of HA word with zero, decodes the home command, and executes the operation. When a CPM gains access to an HA block via the software lock-word, the CPM does not insert a HA command into the Home Address word until the HA lock bit is unlocked.

The lock bit in each Unit Table word protects the IO queues so that access to an I/O queue is not granted to more than one IOM or CPM at a time. The I/O queue can only be accessed when the lock bit is unlocked. Each IOM or CPM locks the bit when it is using the I/O queue and unlocks the bit when it is finished.

HOME ADDRESS WORDS

For each IOM there exists a unique set of Home Address words in memory. The basic purpose of the Home Address words is to provide a location into which CPM's can store an I/O command until an IOM is ready to execute the command. The most generally used command is Start IO, which is used to initiate the processing of a queue of I/O requests for a device by an IOM. Other commands allow the IOM to perform special functions, such as loading into the IOM the addresses of the structures in the I/O map or performing synchronous I/O operations. Other words in the Home Address words are used as software lock words and, in certain cases, to store result descriptors for completed I/O operations.

UNIT TABLE

For each device in the I/O subsystem there is one word in the Unit Table. This word is used both by the MCP and the IOM, and contains a lock bit which prevents conflicts of interest. This word indicates the path or paths to the unit, and provides other information needed by the IOM.

I/O QUEUE HEAD AND TAIL WORDS

For each device in the I/O subsystem there is one Queue Head word and one Queue Tail word. These words contain the address of the first IOCB and the last IOCB, respectively, in the queue of I/O requests for the unit. If there are no IOCB's to be processed for the unit, these words will be empty.

STATUS QUEUE HEADERS

For each IOM there is a Status Queue Header. Fields in the Status Queue Header contain the addresses of the first and last IOCB in a queue of completed IOCB's. Thus, the Status Queue Header allows each IOM to maintain a single queue of completed I/O requests. Periodically, the MCP checks these completed requests.

INPUT/OUTPUT CONTROL BLOCK

An Input/Output Control Block (IOCB) contains the information needed by the IOM to perform one I/O operation on a device. I/O Control Blocks (see figure II-4-5) contain information needed to link queues of IOCB's together, to describe the I/O operation to be performed, to locate the data buffer to be used for the operation, and in the case of completed IOCB's, to store the result descriptor describing the completed operation. A generalized illustration of an IOCB is shown in figure II-4-5.

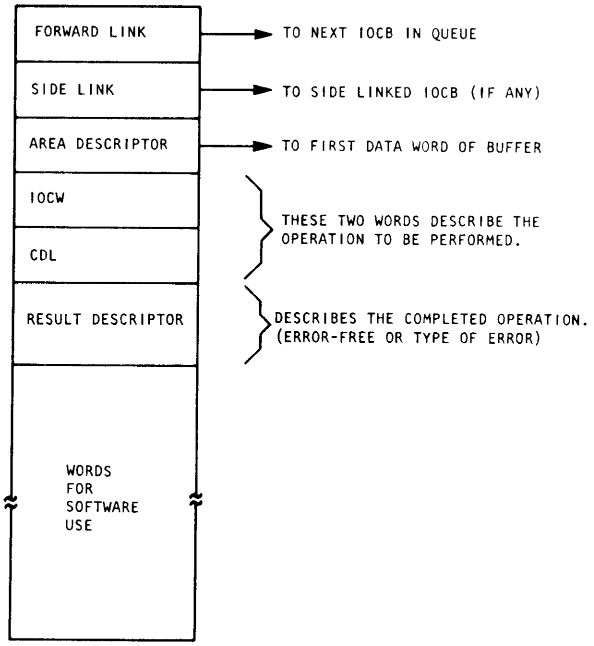

FORWARD LINK
TO NEXT IOCB IN QUEUE
SIDE LINK
TO SIDE LINKED IOCB (IF ANY)
AREA DESCRIPTOR
TO FIRST DATA WORD OF BUFFER
IOCW
CDL
THESE TWO WORDS DESCRIBE THE OPERATION TO BE PERFORMED.
RESULT DESCRIPTOR
DESCRIBES THE COMPLETED OPERATION. (ERROR-FREE OR TYPE OF ERROR)
WORDS FOR SOFTWARE USE

Figure II-4-5. IOCB Format, Simplified

CHAPTER III

CENTRAL PROCESSOR MODULE

SECTION 1

FUNCTIONAL DESCRIPTION OF THE CENTRAL PROCESSOR MODULE

The B 7700 Central Processor Module, which is a highly parallel machine and completely program-compatible with the B 6700 Processing System, consists of three major functional sections that are operationally independent:

a. The program section, which performs instruction decoding operations of object code strings and address calculations of absolute addresses.

b. The execution section, which performs all arithmetic and logical data manipulation operations.

c. The storage section, which performs all storage related functions.

Figure III-1-1 is a simplified block diagram showing the general interconnections and data flow between the three sections. Communications between the sections is established by operations queues.

The program section consists of the program buffer and barrel, program control unit, the fault control logic and the address unit. The program section is responsible for extracting each instruction from the program code string and initiating processing of the instructions, and for the update of the program index and program syllable registers. The program section also controls and responds to the fault interrupt system. The primary responsibility of the program section is to separate the object code string into operations which are then placed in the appropriate queues for execution section. A few instructions are executed entirely by the program section, such as an unconditional branch, and others are executed in part, such as the address calculation portion of Value Call.

The execution section consists of the execution unit and the execution unit input queues. The execution section is responsible for all data and control manipulations. The execution section performs all arithmetic and logical operations as well as stack related control functions. The execution section is driven in an orderly manner from a first in first out list of operations placed in its operator queue by the program section.

The storage section consists of the storage unit, the stack buffer unit, the associative memory and the communications unit. The storage section is responsible for all storage related functions. Some of the storage section's duties are implied such as maintaining the stack buffer, but most operations are explicit in that they result directly from the processing of program code. Implicit operations for the storage section are placed in the input queue of the storage unit by the program section or in the storage output register by the execution section. It is the responsibility of the storage section to determine if an address reference points to local storage or to main memory, in which case, a main memory cycle is necessary.

These major sections are subdivided into units which operate relatively independently. The operation of each of the units of the CPM is described separately. First some of the basic operational concepts of the CPM are presented to aid in understanding the physical and conceptual design of the Central Processor Module.

In general, the program operators in the program code string are fetched from memory in multi-word segments and placed in the program buffer. The operators are extracted one at a time by the program control unit and each is separated into one or more micro operators, which are queued for processing by the execution unit. When possible the program control unit determines what data will be required for execution of the micro operators and requests this data from the storage unit. For literal values, which are contained in the code string, the program control unit extracts the data and forwards it directly to the execution unit. Therefore, as the execution unit processes the micro operators, the required data is usually available, allowing the execution unit to perform the required proc-

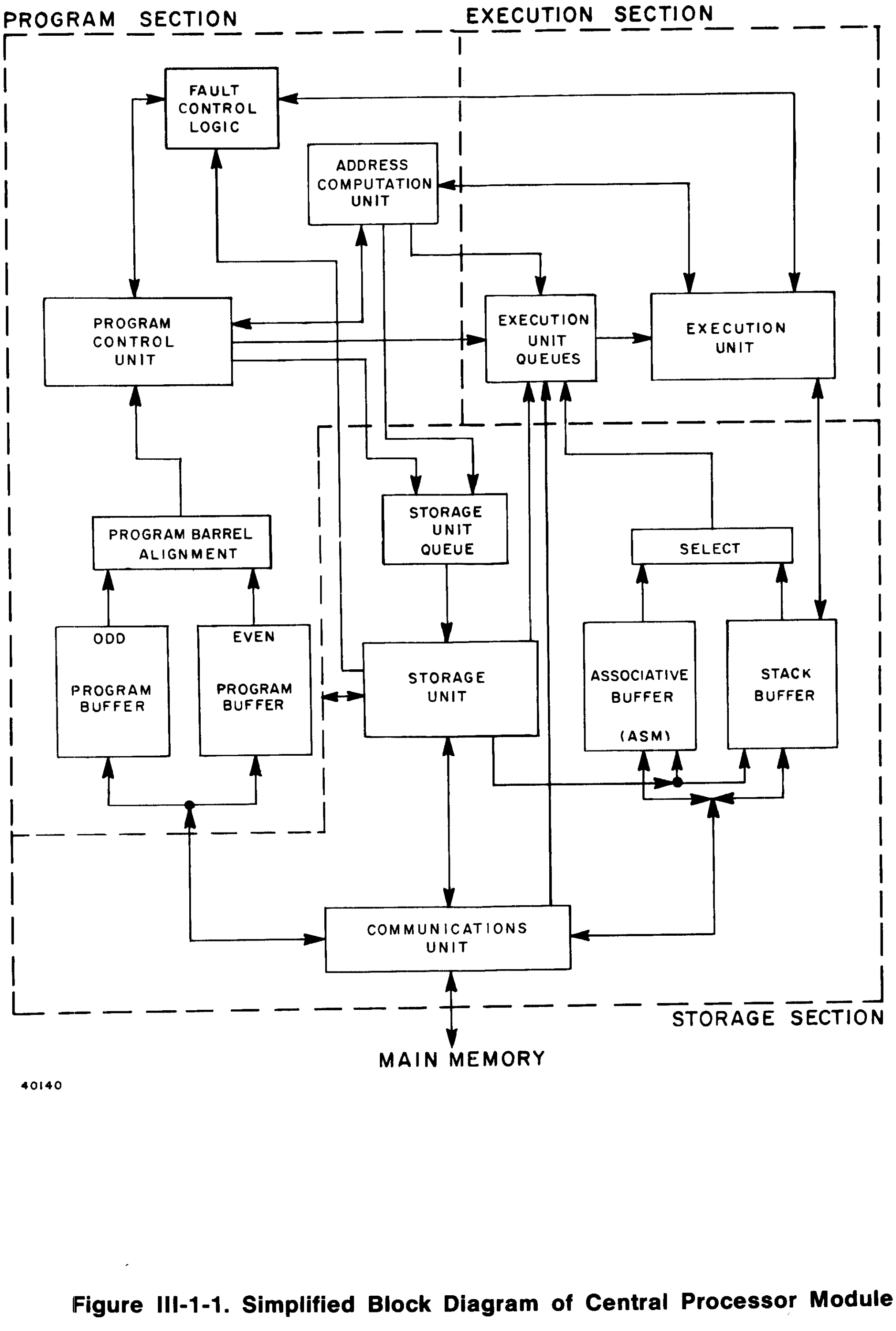


Figure III-1-1. Simplified Block Diagram of Central Processor Module

essing without delay. Results derived by the execution unit may either be stored in one of the local memory areas or may be sent through the storage unit and the communications unit to main memory. By using this pipeline technique, relatively high-speed processsing has been achieved without compromising equipment reliability.

To further increase processing speed, extensive use has been made of buffer memory areas contained within the processor. These local memory areas are used to store program code, a portion of the active program stack, and referenced variables. The following example shows how these local memory areas increase processing speed by eliminating many memory references.

OPERATIONAL CONCEPTS OF THE CENTRAL PROCESSOR MODULE

The B 7700 Central Processor Module (CPM) is designed as a pipeline processing unit; therefore each processing station may be operating simultaneously on a different task. As any instruction is passed through the processing pipeline, successive operations are performed by the various processing stations until the instruction is fully executed.

USE OF DIVISION OVERLAP AND LOCAL BUFFERING

Figure III-1-2 shows a simple statement along with its compiler generated code and traces each operator as it is encountered in each of the major processing units.

The sequence begins as the first Value Call arrives at the program control unit. This unit selects the appropriate display register and calculates the absolute address.

The address and operator are placed in the input queue of the storage unit. At the same time, the operator is placed in the input queue of the execution unit. The execution unit then begins to wait for the return of the value. Next, the program control unit processes the Name Call. Detection of the Name Call alerts the program control unit to look at the next instruction to set the context of the Name Call.

The appearance of the Index allows the Name Call to be concatenated with the Index operator. The address calculation is performed and the address and operator are placed in the storage unit's queue. A micro operator indicating a concatenated Name Call is placed in the execution unit queue. The Index operator is now placed in the execution queue.

The second Value Call is then processed by the program unit and the address and operator are placed in the queues. During this time, the storage unit has been busy with the first Value Call. The program control unit now has the second Name Call and has finished its concatenation investigation and subsequence address calculation. It then places the address and operator in the storage unit queue and passes another pseudo operator to the execution unit. The storage unit just prior to this has completed the first Value Call, which was found in the stack-buffer area of the processor, and has passed the operand to the waiting execution unit. The storage unit now goes to the next item in its queue, the first Name Call reference.

The execution unit investigates the control information of the first Value Call and places it in the top-of-stack location. The execution unit is now waiting for the Name Call reference which will be found local in the stack buffer and thus will be transferred by the storage unit. At the same time, the program control unit has placed the Index Load Value in the execution unit queue. The storage unit finds the first Name Call reference to be local in the stack buffer and places it in the execution unit queue. The execution unit now begins the Index function and the storage unit goes on to the second Value Call and then on to the second Name Call.

The execution unit subsequently accepts both of these calls through its queue and begins the computation involved with the Index Load Value instruction. When the execution unit supplies to the storage unit the address for the fetch, the memory reference is initiated, if the data to be fetched is not in local memory, and the execution unit holds until the return of the value. Upon return of the value, the execution unit places the data in the storage unit. These units remain in sync until the store is completed.

The program control unit has, at this point, proceeded to the Branch instruction, but prior to this, a temporary hold was placed on the pipeline, because the Index Load Value operator has caused the execution unit queue to go full. The hold was released as soon as a slot became available and the program control unit went on to the branch.

The branch point is calculated and presented to the program buffer control for local test. If the branch point is within the portion of program code held in the program buffer, the local pointers are readjusted and processing continues. If the code at the branch address is not available in the program buffer, then a main memory reference is initiated by the storage control unit.

STATEMENT

A [I] : = B [I]

COMPILER CODE

VALC; NAMC; INDX; VALC; NAMC; NXLV; STOD;

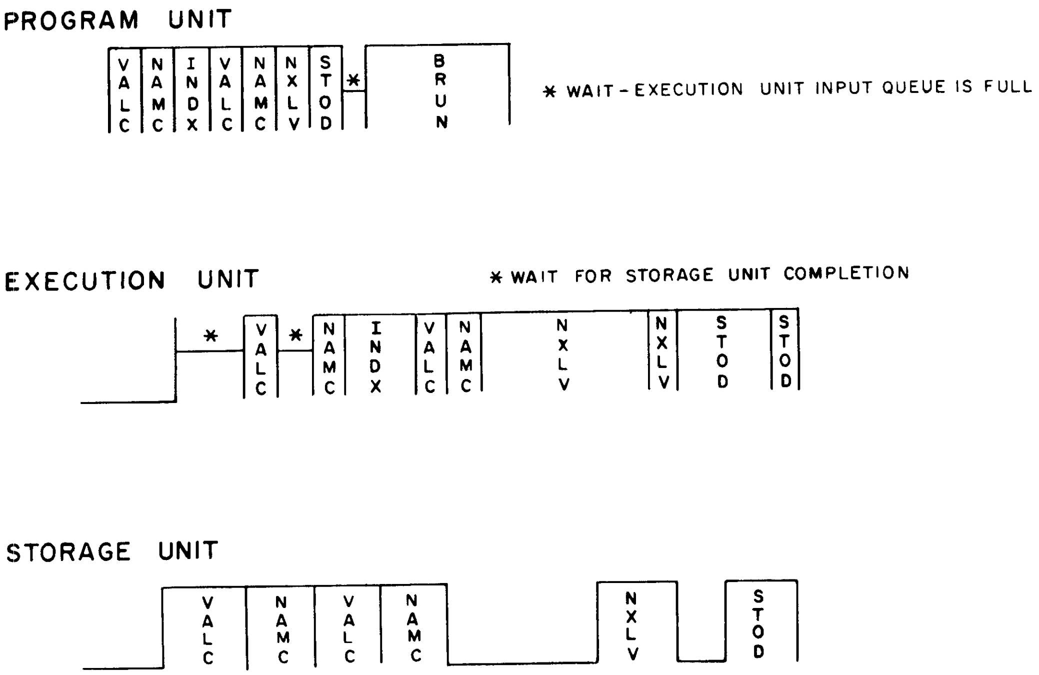


Figure III-1-2. Division Overlap

MEMORY OVERLAP

In the preceding example, all reference data were found in the local buffers. However, the pipeline processing technique is efficient even when none of the required data is found local. This efficiency is illustrated by the following example.

As shown in figure III-1-3, the program control unit progresses without interruption through the entire sequence of code. All three Value Calls are found to require main memory fetches. The execution unit expends much of its time waiting for the first two Value Calls to be transferred into its queue by the storage unit. The third Value Call, however, completely masks the multiply time. Although average time was used for multiply in the diagram, a maximum multiply would still conclude before the third Value Call arrived at the execution unit queue. The program control unit again took advantage of concatenating the Name Call and thus reduced the time necessary for the execution unit's portion of the store. The Branch instruction was again completed long before the execution unit reached this point in the program.

PROGRAM BUFFER

The program buffer provides local storage for up to 32 words of the executing program's object code. The algorithm for loading the buffer is based on anticipation rather than waiting until all code in the buffer has been processed, so that full advantage is taken of the natural idle time on the main memory bus. Because an average of 3.5 instructions (operators) are contained in each program word, program loops are often entirely contained within the buffer. Therefore, in many cases, branching may take place without a main memory reference for the new program word. (A branch which may be made with no main memory reference is referred to as a "local branch."

In addition, by fetching the code for the program buffer in multi-word blocks, the number of memory accesses required is significantly reduced.

As shown in figure III-1-4, the IC-memory storage area of the program buffer is arranged in four blocks, with eight 60-bit words in each block. Each eight-word block is further divided into odd and even segments. For example in block 0, words 0, 2, 4, and 6 constitute the even segment, and words 1, 3, 5, and 7 constitute the odd segment.

BUFFER WORD FORMAT

When fetched into the CPM, each word of object code consists of six eight-bit syllables, a tag of 011 (which identifies the word as containing object code), and an odd parity bit. Because operators vary in length and because they are packed in main memory without regard for memory word boundaries, to ensure operator integrity, separate parity bits are generated on each syllable of the program code prior to entry of the word into the buffer. The parity is checked as the syllables are used.

The 60-bit program buffer word consists of six eight-bit syllables of code, six syllable parity bits, three tag bits, an overall parity bit, and two error-check bits. (The error bits are always set to zero unless the program buffer word is overwritten by an error word from the communications unit.)

READING FROM THE PROGRAM BUFFER

To read a location in any of the local memory areas in the processor, it is necessary only to enter the address into the appropriate read pointer register. (Refer to figure III-1-5.) The decoding logic then selects the addressed word and gates the contents of the location to the storage area output as long as the read pointer contents remain unchanged. Thus the contents of the addressed location are always available on the output, and the output can be used as needed.

In the program buffer, two words, one from an odd segment and the other from an even segment, are addressed simultaneously. This is accomplished by using two read pointers. The program odd buffer (POB) is the read pointer for the odd segments, and the program even buffer (PEB) is the read pointer for the even segments.

The two words read simultaneously from the buffer storage come from consecutive addresses in main memory. One of the words contains the beginning of or all of the next operator to be preprocessed, and the other word is the word which was fetched from the next-higher memory address. The second word may or may not contain syllables of the desired operator as the operators are of variable length and are not restricted to memory word boundaries.

WRITING INTO THE PROGRAM BUFFER

When a word of code is ready for entry into the buffer storage, it is written into the buffer word location (word 0 thru 31) pointed at by

STATEMENT

L: = I⊗J + K

COMPILER CODE

VALC; VALC; MULT; VALC; ADD; NAMC; STOD;

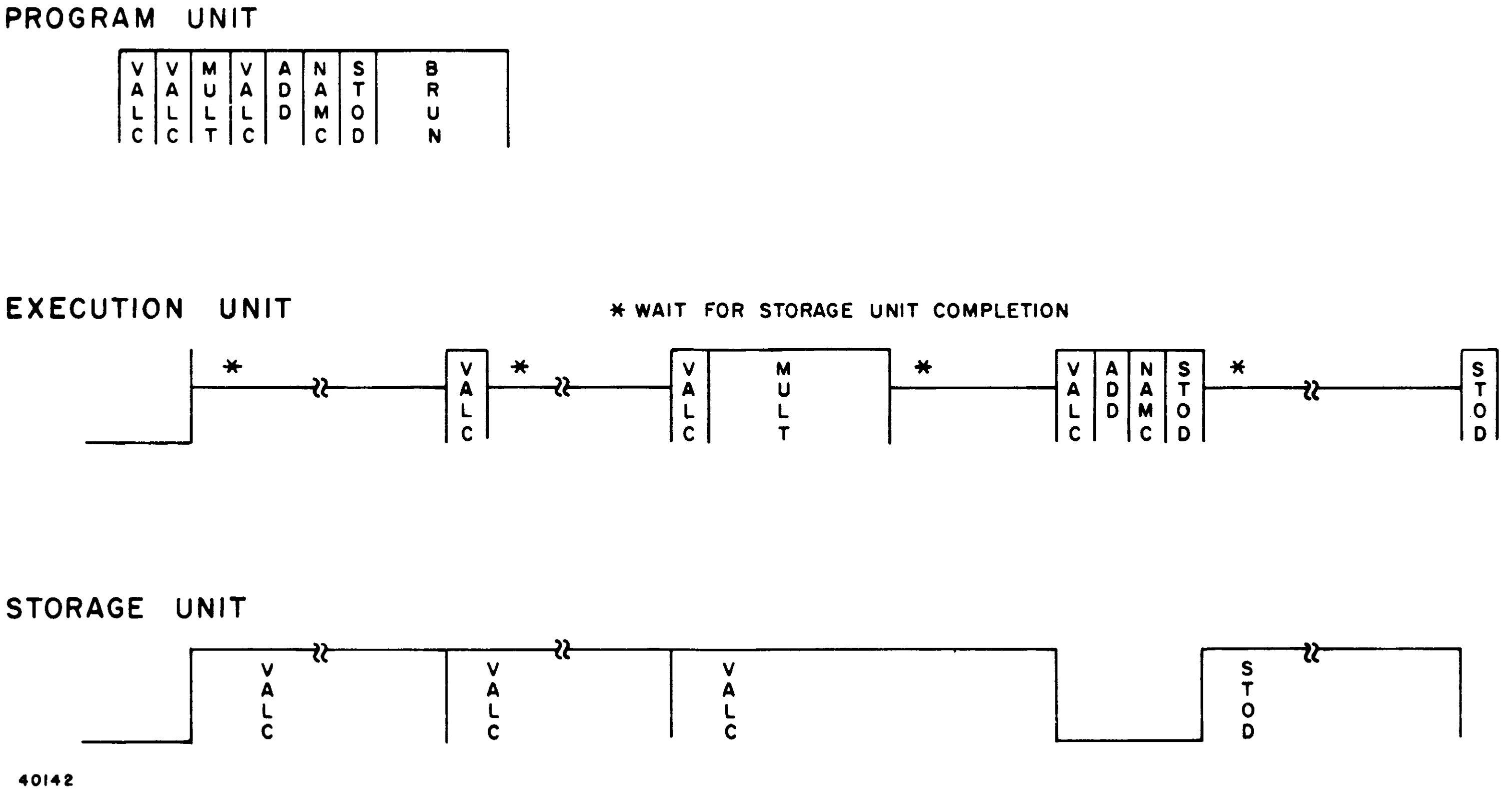
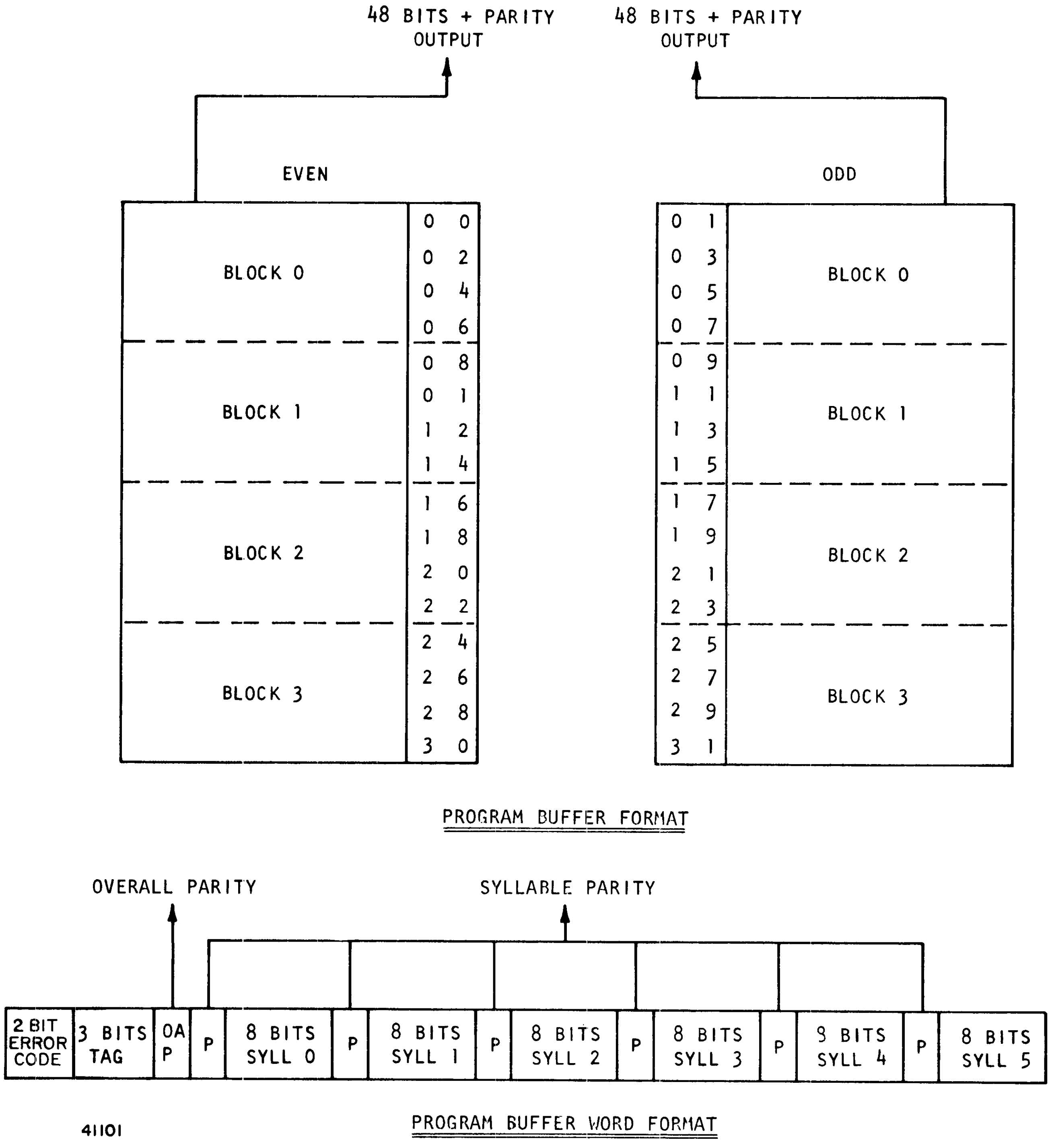


Figure III-1-3. Memory Overlap



PROGRAM BUFFER FORMAT

PROGRAM BUFFER WORD FORMAT

Figure III-1-4. Program Buffer Arrangement

the five-bit program write pointer (PWP). (See figure III-1-5.)

The two high-order bits are decoded to select an eight-word block, and the three low-order bits are decoded to select a word location within the block. The actual write cycle is initiated by triggering a single-shot circuit. Each time a valid data word is written into the buffer storage, the count in the program write pointer is increased by 1.

The absolute memory address of the next word to be placed in the program buffer is held in the program upper register (PUR).

The program lower register (PLR) contains the absolute memory address of the first word of the block that has been in the program buffer for the longest time. When the CPM is started, the buffer is filled from word 0 thru 31. Thereafter, each word fetched normally overwrites the oldest word resident in the buffer.

When the processing of the last two words of program code in the buffer begins, the program-buffer logic requests the communications unit to fetch the next eight words of code from main memory. The PLR contents are then increased by eight.

The branch pointer (BR) identifies with two bits which block of the buffer contains the word whose address is in the PLR. The BR is counted up each time the PLR address is changed.

BRANCHING

Whenever a branch is executed, the branch address is entered into the program address register (PAR). Then the PAR contents are compared with the contents of the PU and PL to determine if the branch is local. If the branch address is between the PU and PL addresses, the code is in the program buffer.

For local branches, the read pointers are updated by determining the offset of the branch address from the PLR setting. This offset, together with the BR contents, can then be used to provide the block and word address of the code in the buffer.

If the branch is not local, excluding branch on true (BRTR) and dynamic branch true (DBTR), the program buffer is declared empty and the address in PAR is transferred to the PUR and PLR, then to the communications unit as the address for the eight-word fetch.

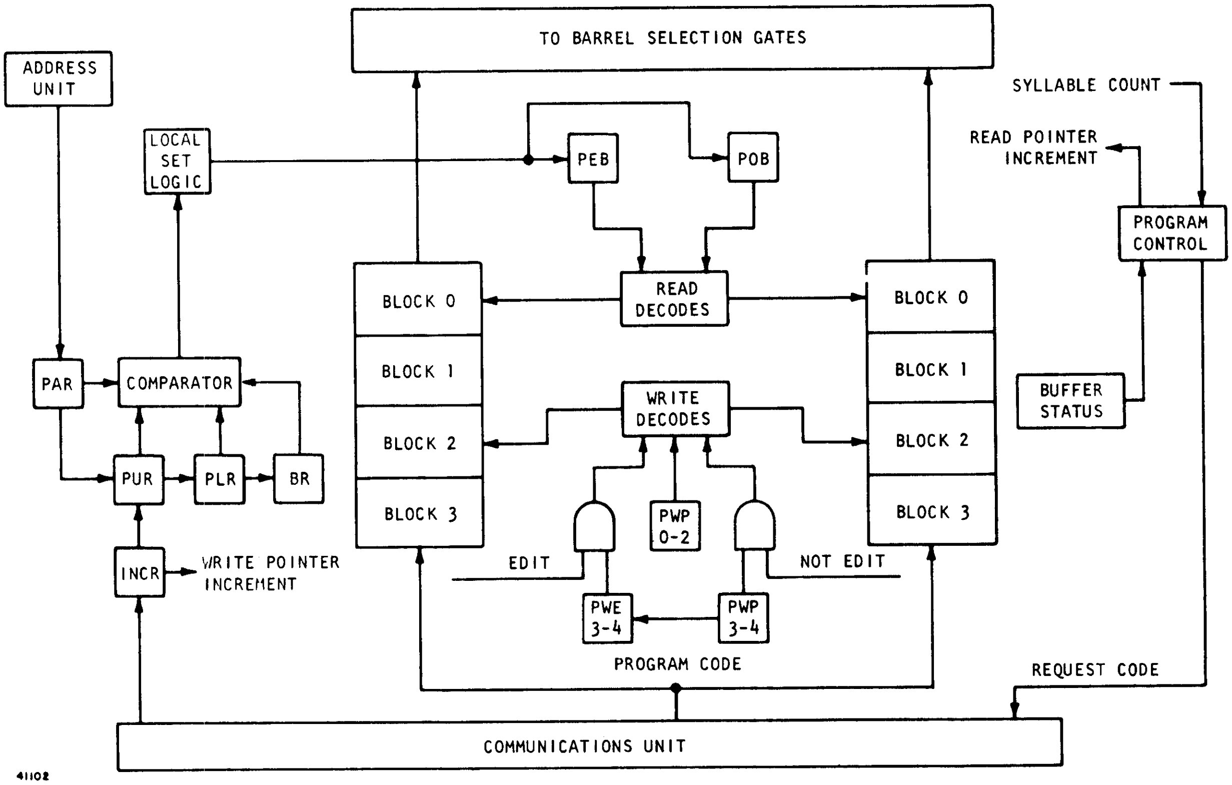


Figure III-1-5. Program Buffer Unit

After the eight-word fetch is received from memory, the PU contents are increased by a count of eight. The next branch address is then compared with the new PU and PL addresses.

For BRTR and DBTR operators, a check for loop (CFL) is made to determine if the branch is local in the next 4-word code block prior to the actual fetch from memory. To do this, the hardware increases the PU contents by 4 and compares the PAR contents with the contents of the PU and PL.

If the branch address is between the PU and PL addresses, the program buffer is declared empty and the address in PU is returned to the previous setting, then transferred to the communications unit as the address for the eight-word fetch. Thus, the last block of code is saved so that branching back can be performed locally in the program buffer.

If the branch is not local in the four-word code block, the hardware performs the branch operation as explained in the previous paragraph. The CFL condition is released when the next change in direction occurs, or when an enter, exit, or a return is processed.

EDIT MODE OPERATION

The operation of the program buffer is altered during table edit mode. When the processor executes a Table Enter Edit operator, it in effect branches to a block of code referred to as an edit table. An edit table consists of a series of special operators used to edit data. One pass is made thru the table each time the table is used. The last operator in the table is an End Edit operator. At the completion of the table pass, the processor returns to the operator following the Table Enter Edit operator in the program code string.

Some of the program code string is maintained in the buffer during the table pass to facilitate an orderly return after the edit mode operators have been processed. Therefore, during edit mode, two blocks of the buffer are used for edit operators and two blocks are assigned for keeping a portion of the program code. The program write edit pointer (PWE) flip-flops replace the two high-order bits of the PWP during edit mode operation. These PWE bits are then counted in a manner that allows only the two blocks assigned to the edit operators to be loaded during edit mode. The two high-order bits of PWP remain unchanged and are used to reestablish the write pointer address on completion of edit mode.

On entry into edit mode, the PL address is adjusted to account for the blocks of program code which will be overwritten by edit operators, then the updated PL address is saved. On exit from edit mode, the PL address is reentered into PLR. Then the PL address, together with the block (BK) count, is used to set up the PU address. (The block counter shows how many blocks of program code were in the buffer prior to entry into edit mode.)

VECTOR MODE OPERATION

Another change in operation occurs when the processor encounters vector mode operators. Vector mode operators facilitate the repetitive execution of an operator or a group of operators on all items in an array or a group of arrays. When the vector mode operators appear in a single word of code, the logic simply forces an automatic one-word branch backwards until all items have been processed. When the vector mode operators extend beyond the boundaries of a program word, advantage is taken of the automatic local-branch-point detection.

PROGRAM BARREL

The program barrel, shown in figure III-1-6, is a shifting mechanism used for aligning and extracting the program operators from the two words of code read from the program buffer. The program barrel consists of one selection stage and one shift stage. The selection state receives as an input the two program words which are being read from the program buffer.

SELECTION GATING STAGE

The selection gates align the two words read from the program buffer so that the word in which the beginning (most-significant portion) of the next operator appears is placed in the more-significant word position, and the other word in the less-significant word position. When the odd-even flip-flop (OEF) is set, the odd word is placed as more significant. When OEF is reset, the even word is placed as more significant. The OEF flip-flop is complemented each time the last syllable of a word is processed.

Only the eight most-significant syllables of the two words are provided as an output from the selection gates. The four least-significant syllables of the word in the less-significant position are not required for decoding the operator and are stripped off at this point. The eight output syllables include at least the most-significant syllable of the next operator to be processed and the two syllables which follow the most-significant syllable. All operators can be decoded from this information.

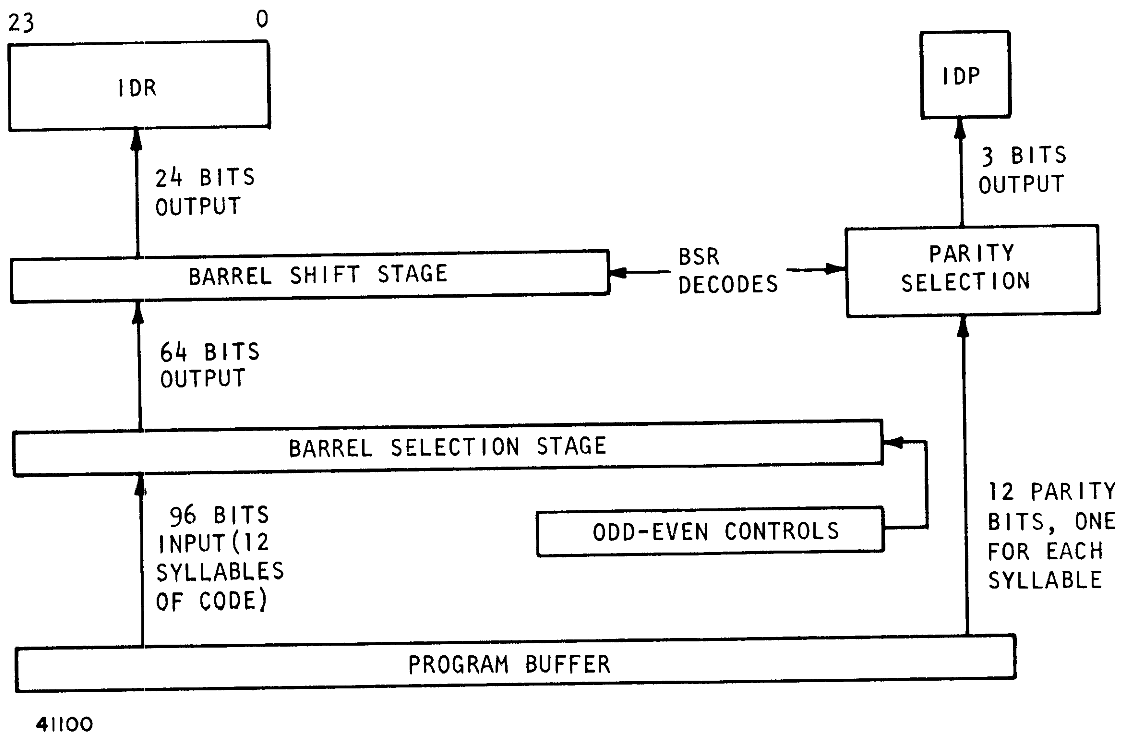


Figure III-1-6. Program Barrel

BARREL SHIFT STAGE

The shift stage is used to extract the most-significant syllable of the next operator and the two following syllables. These three syllables are then forwarded as the input to the instruction decode register (IDR). In effect, the three syllables are left justified and then extracted. The three syllables required are selected by decoding the contents of the barrel select register (BSR). The contents of BSR provide the syllable position of the beginning of the operator being extracted. The BSR count is increased by one when the 24 bits are extracted.

As soon as the operator has been decoded, the BSR count is advanced further if the operator contains more than one syllable. Each time a BSR count cycle (from 0 thru 5) is completed, the end of a program word has been reached. Then the OEF flip-flop is complemented. When a branch occurs, both the BSR and OEF are force-loaded to properly identify the location of the new code in the program buffer. The BSR is located in the program control unit and is updated under control of that unit.

SYLLABLE PARITY CHECKING

Separate gating is provided to extract from the program barrel the syllable parity bits associated with the three selected syllables of program code (see figure III-1-6). The 12 parity bits from the two words read from the program buffer are applied to the parity gating, which uses the OEF and BSR contents to select the three desired bits. These parity bits are loaded into the IDR parity register (IDP), at the same time as the operator syllables are loaded into the IDR.

PROCESSING OF LT48 OPERATOR

Most of the operators consist of three or less syllables. However the LT48 operator has a one-syllable operator code in one program code word and a 48-bit literal value contained in the following program word. The execution of this operator consists of placing the 48-bit literal value on the top of the stack. Because only 24 bits are output from the barrel, the 48-bit literal value is not available at the barrel output. However, the LT48 operator code is passed thru the barrel, and the 48-bit literal value is available at the program buffer output. Therefore the value is passed to the execution unit directly from the program buffer output. This is accomplished by applying both the odd-word and even-word outputs from the program buffer to the execution unit's data input register (the EWR), then complementing OEF and using the result to select the literal value for loading into the EWR. Then, so that the next two words of program code are selected during the following operation, the BSR is reset to 0 and OEF is complemented once more.

PROGRAM CONTROL UNIT

The primary tasks of the program control unit (PCU) are to decode the program operators and to partition the object code into a series of micro operators which are placed in the appropriate queues for execution. Figure III-1-7 is a simplified block diagram of the PCU and shows the important operational flow and control interconnections.

Two major registers in the PCU are directly in the operator pipeline and constitute two of the processing stations in the pipeline; these

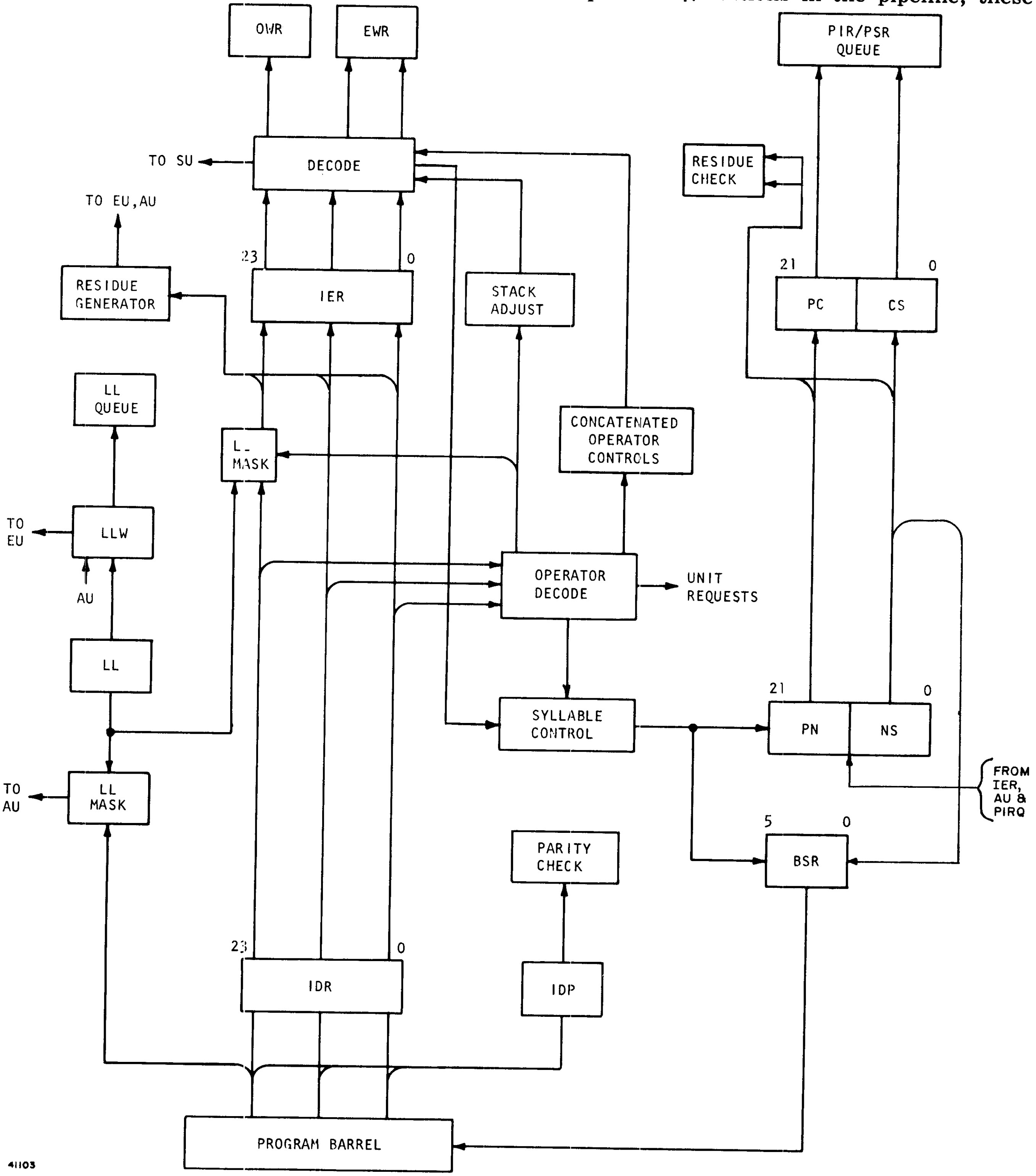


Figure III-1-7. Program Control Unit

are the instruction decode register (IDR) and the instruction execute register (IER). The IDR is the preprocessing or "look-ahead" station of the PCU. The IER holds the operator when it is being divided into micro operators and is the major execute register of the PCU.

INSTRUCTION DECODE REGISTER

As a "look-ahead" station, the IDR is used to decode the program operator and to set up conditions for PCU execution of the operator in the IER. The operator remains in the IDR until the PCU processing of the preceding operator is complete, then the operator in the IDR is passed on to the IER and the following operator is loaded into the IDR. The 24-bit IDR is loaded with the output of the program barrel and, by its decodes, in conjunction with the IER decodes, controls the output of the program barrel.

The processing functions of the IDR include the initiation of stack adjustments to provide the proper configuration of operands in the execution unit at the start of each operator, requesting access to other units within the CPM when the operator decode indicates that communication with other units is required, examination of the operator following each Name Call operator to determine if the Name Call and the following operator can be concatenated, and initiation of the appropriate timing sequence for IER execution of the operator.

REGISTERS ASSOCIATED WITH THE IDR

The control registers associated with the IDR are the instruction decode parity (IDP) register, the program index next (PN) register, the next syllable (NS) register, and the barrel select (BS) register. The IDP register contains the three syllable parity bits associated with the syllables contained in the IDR and is used in checking IDR parity.

The PN and NS registers contain the program index and syllable counts for the operator contained in the IDR. The program index value, when added to the contents of the program base register (PBR), provides the absolute main memory address of the program operator. The syllable count identifies the starting syllable position in the memory word of that operator.

The barrel select register, which identifies the syllable position of the first syllable of the next operator to be placed in the IDR, is updated in accordance with the decodes from the IDR and the IER. The contents of the barrel select register controls the output of the program barrel as previously described.

IDR DECODES

Decoding of the operator in the IDR provides two types of decode signals: control decodes, which are used for updating the contents of the registers associated with the IDR, and operational decodes, which are used in the preprocessing of the operator in the IDR.

CONTROL DECODES

When processing of a new segment of program code string begins, the contents of the program base register, which is maintained in the address memory area of the address unit, are updated to provide the base address for the code segment being executed. The memory address of each operator in the segment is maintained as an index to this base. This program index and the associated syllable count are passed along with the operator in the CPM pipeline until the execution of the operator is complete.

If some interrupt is encountered in the pipeline, the memory address of the operator is thus available for re-execution or error reporting purposes. The program index and syllable counts are established in the PN and NS registers. The contents of these registers are updated by IDR and IER decodes as part of the preprocessing of each operator. The syllable count in the NS register is updated along with the BS register contents. Each time an NS register cycle, which is a count from 0 thru 5, is completed, the PN count is increased by 1.

The control decode signals from the IDR are also used to update the contents of the BS register. The BS register contents are then used to control the alignment of the next operator coming out of the program barrel. The contents of the BS register are upcounted by 1 each time the IDR is loaded. For monosyllabic operators, no additional update is required.

For multi-syllable operators which will be held in the IER for only one machine cycle, additional updating occurs at the beginning of the next machine cycle. However, for those operators which will be held in the IER for several cycles, additional updating occurs at various times, but always quickly enough so that the next operator may be in the IDR for at least one machine cycle before transfer to the IER.

OPERATIONAL DECODES

The operational decodes of the IDR contents are used primarily for stack adjustments, for concatenation investigation, and for gaining access to other units as required.

To facilitate the issuing of micro-operators which provide for the proper operand configuration in the execution unit at the beginning of each program operator, the PCU must predict what top-of-stack operands will be left in the execution unit at the completion of each operator. This prediction is maintained in the stack-A-operand (SKA) and stack-B-operand (SKB) flip-flops in the PCU. When an operator is in the IDR, the control logic determines what the initial operand configuration for that operator must be. If the contents of SKA and SKB indicate that the top-of-stack operands in the EU will not be in the proper configuration, then IDR Operational decodes are passed to the IER which inserts stack-adjustment micro operators as required.

The purpose of the NAMC operator is to place an IRW on the top of the stack. However, if the operator following the NAMC requires that the address couple in the IRW be evaluated to derive the memory address, the NAMC operator is concatenated with the following operator, so that the address couple in the code string can be converted directly to an address. The concatenation occurs whenever NAMC is followed by an Enter operator, any index operator, DBUN, LOAD, LODT, or store operator. The IDR operational decodes are used to detect when concatenation may occur.

Because the NAMC operator contains two syllables, the third syllable in the IDR with the NAMC operator is the operator code of the next operator in the code string. When concatenation is possible, the operational-decode signals set a control flip-flop to denote the action.

The IDR decodes are also used for access requests. If the decoding of the operator in the IDR shows that a fetch or store operation is required for execution of the operator, the IDR requests use of the storage unit, so that when the operator is in the IER, the PCU may queue a request for the required operation. In a similar manner, if data is to be provided directly by the PCU to the execution unit along with the micro operators, or if variant information is to be loaded into the K and L queues, the IDR decode signals request use of the execution write register (EWR) for the PCU.

INSTRUCTION EXECUTE REGISTER

The 24-bit instruction execute register is the main PCU processing register. The IER is loaded from the IDR each time the IER completes the PCU processing sequence of the current operator. As the next operator is loaded into the IER and decoded, the IER issues any required stack-adjustment micro operators to the execution unit. Then the IER issues the required micro operator sequence to the EU.

All micro operators issued are placed in the OW register in the PCU. The contents of the OWR are then written into the execution unit operator queue or, if the execution unit is waiting for work, are passed directly to the execution unit. The micro-operator codes issued to the execution unit are eight bits in length. Simple program operators require a series of micro-operators to complete the operator functions. In many cases, the operator code of the program operator is issued directly to the execution unit as a micro operator.

The primary timing control signals for PCU processing are developed by a down-counter, which is preset to the proper configuration from IDR decodes. During processing, the IER decodes issue a micro operator on each machine cycle. Issued along with each micro operator are variant codes and, in the case of literals, data. Variant information and data provided by the PCU are loaded into the EWR and then queued for EU use. Often the variant information supplied to the execution unit is taken directly from the second and third syllables of multisyllable operators. When a micro operator requires a fetch or store of data for execution, the IER decodes cause the address to be queued for storage unit action.

When the last micro operator of a program operator sequence is issued by the IER, the PCU sets a special bit in the OW register which informs the execution unit that this micro operator completes a program operator sequence.

Many of the micro operators issued by the PCU can be executed in only one machine cycle, others require multiple cycles for execution. When micro operators requiring several machine cycles are executed, the execution unit operator queue may become full. In such cases, PCU operation is suspended until space is available.

As previously stated, the program index and syllable information remains with the operator throughout the execution of the operator. To accomplish this, the contents of the PN and NS registers are transferred into the PC (program current) and CS (current syllable) registers at the same time as the associated program operator is transferred from the IDR to the IER. Then, each time a micro operator is issued, the contents of the PC and CS registers are queued along with the micro operator for execution unit use.

PREPROCESSING OF VALUE CALL AND NAME CALL OPERATORS

The Value Call and Name Call operators are the most frequently used operators in the

B 7700 operator set. Therefore, special processing features are provided to facilitate one-cycle execution of these operators.

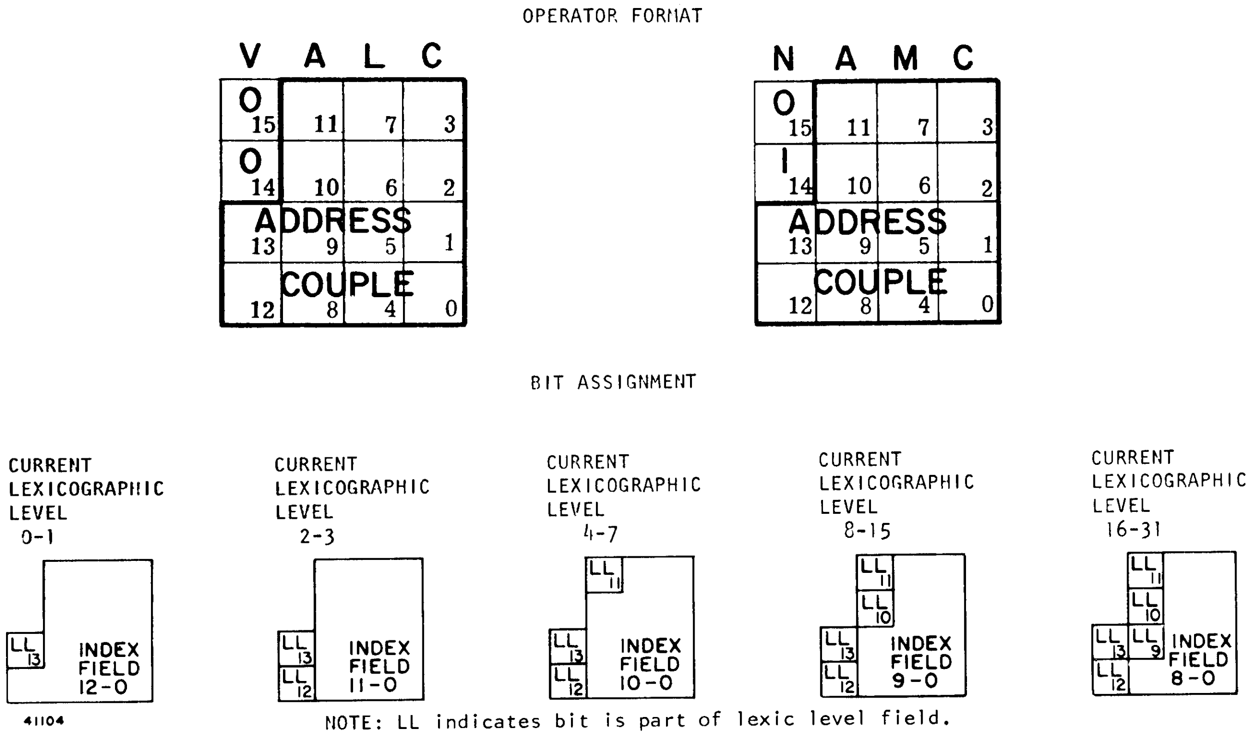
The two-syllable Value Call instruction (VALC) requires that the 14-bit address couple in the instruction be evaluated to provide an absolute address from which data are fetched and placed on the top of the stack for EU use.

The two-syllable Name Call instruction (NAMC) indicates that the address couple in the instruction may be used to form an IRW, which is then placed on top of the stack. However, if the NAMC operator is followed by an operator which would require evaluation of the address couple to derive an absolute address, then the NAMC is concatenated with the following operator and the address couple is evaluated immediately. NAMC is concatenated when the next operator in the program code string is any of the following: ENTR, INDX, NXLN, NXLV, STOD, STON, OVRD, OVRN, DBUN, LOAD, and LODT. If a NAMC cannot be concatenated, an IRW containing the address couple is placed on the top of the stack for EU use.

The 14-bit address couple in the NAMC and VALC instructions consists of a lexicographic-level field (LL) and an index field (I). As shown in figure III-1-8, the length of each of these fields varies with the current lexic level of the active program. The LL field ranges from one to five bits in length and contains only as many bits as are required to define the current lexic level. The remaining bits are the index field. (The bits of the LL field are in inverse order so that the least-significant bit of the field is located in the most-significant bit position of the address couple.)

To facilitate preprocessing of VALC and NAMC, the PCU presupposes that every operator in the code string is either a NAMC which can be concatenated or a VALC. Therefore, as each operator is loaded into the IDR, it is assumed to contain an address couple which must be converted into an absolute address. If subsequent decoding reveals that no address-couple conversion is required, or that there is no address couple in the operator, the conversion is terminated with no loss in processing speed.

When any operator is transferred from the program barrel to the IDR, bits 21:5 of the barrel output, which contain all possible bits of the LL field of the address couple (if the operator is a VALC or a NAMC), are gated with a lexic-level mask in addition to being entered intact into the IDR. The lexic-level mask gates the LL field bits of the address couple and inhibits any bits which are part of the index field. The lexic-level mask is set up by decoding the current program lexicographic level,



NOTE: LL indicates bit is part of lexic level field.

Figure III-1-8. Address Couple Bit Assignment

which is maintained in the lexic level (LL) register. The output of the LL mask, then, is the bits of the LL field if the operator is a NAMC or a VALC. In any event, these bits are loaded into the five LS bits of the program read pointer in the address unit.

If no request for use of the AU is pending, the contents of the display register addressed by PRP are read out of the address memory area and entered into the display read register. The LL field, now in PRP, is also written into the lexic level write register (LLW) for possible later use.

By this time, the decoding of the operator, now in the IDR, is complete. If the operator is either a VALC or NAMC, it contains an address couple and the LL field is not transferred to the IER with the rest of the operator, but is stripped out of the operator code by use of the lexic-level mask. Therefore when a NAMC or VALC operator is transferred into the IER, bits 20:13 of the IER (the largest possible index field) contain only index bits from the address couple. These bits are now applied directly to the address-adder selection gates in the address unit for use as one of the inputs to the adder. The other input is the base address now contained in the display read register. Therefore, the adder output is the absolute memory address described by the address couple.

For a Value Call, the output of the address adder is gated into the input register of the storage unit, along with the PCU request for a fetch and the EU-data-queue address reserved for the requested data. This information is then queued for storage-unit processing. The same operation is performed for a concatenated Name Call which requires a fetch operation. (NAMC concatenated with an index-type or Enter operator requires a fetch operation.) For a name call-store combination, the address and a request for a store are queued for storage-unit action, but the store is not performed until the EU supplies the data to be stored.

For an unconcatenated Name Call, the output of the address adder is ignored, and an IRW is built in the EWR. The address couple for the IRW is formed by combining the index value in the IER with the LL field, which was saved for this purpose in the LLW register.

ADDRESS UNIT

The address unit (AU) contains the logic necessary for the calculation of absolute addresses, both directly as in Value Call and indirectly as in pointer update for string operators. As shown in figure III-1-9, the functional parts of the address unit are the display write (DW) register, the display read (DR) register, the address adder, the address-storage area, and the read and write pointers (PRP, ERP, and DWP) for the AU local storage. The address unit is not directly in the processing pipeline and is therefore not queue driven. It is an autonomous unit only to the extent that a write cycle into the address-storage area need only be initiated and not completely controlled by the initiating unit.

The local address-storage area comprises 48 locations, each location having 20 bits for storage of an absolute address and two bits for storage of address residue. The local storage contains the 32 display registers used in relative addressing within the active program stack. Each display register in use contains the absolute address of the MSCW for a different lexicographical level. The display registers are numbered in order from D0 thru D31; the D0 register contains the MSCW address for lexicographic level 0, the D1 register contains the MSCW address for lexicographic level 1, etc. To address within any level, the appropriate D register contents are read, and then the displacement of the desired item from the MSCW is added to the MSCW address to provide an absolute address for the required item.

The remaining 16 locations of the storage area provide storage for certain index, base and miscellaneous registers. The registers maintained in these locations are as follows:

Register Mnemonic	Register Name
SIR	Source Index Register
DIR	Destination Index Register
TIR	Table Index Register
BOSR	Base of Stack Register
S1LS	Scratch (Space Local Storage)
PBR	Program Base Register
SBR	Source Base Register
DBR	Destination Base Register
TBR	Table Base Register
SNR	Stack Number Register
PDR	Program Dictionary Register
S2LS	Scratch (Spare Local Storage)
ADZ	Alternate D0 register
APIR	Alternate Program Index Register
ALL1	All 1's Register
LD1	Last D [1] used as SD1 base

All the contents of the address storage are addressable by the SPRR and RPRR operators. (A total of 64 registers are addressable by these operators, 48 of which are in the AU address storage area. Those registers which are addressable but which are not contained in the AU address storage are also addressed by use of the read and write decode

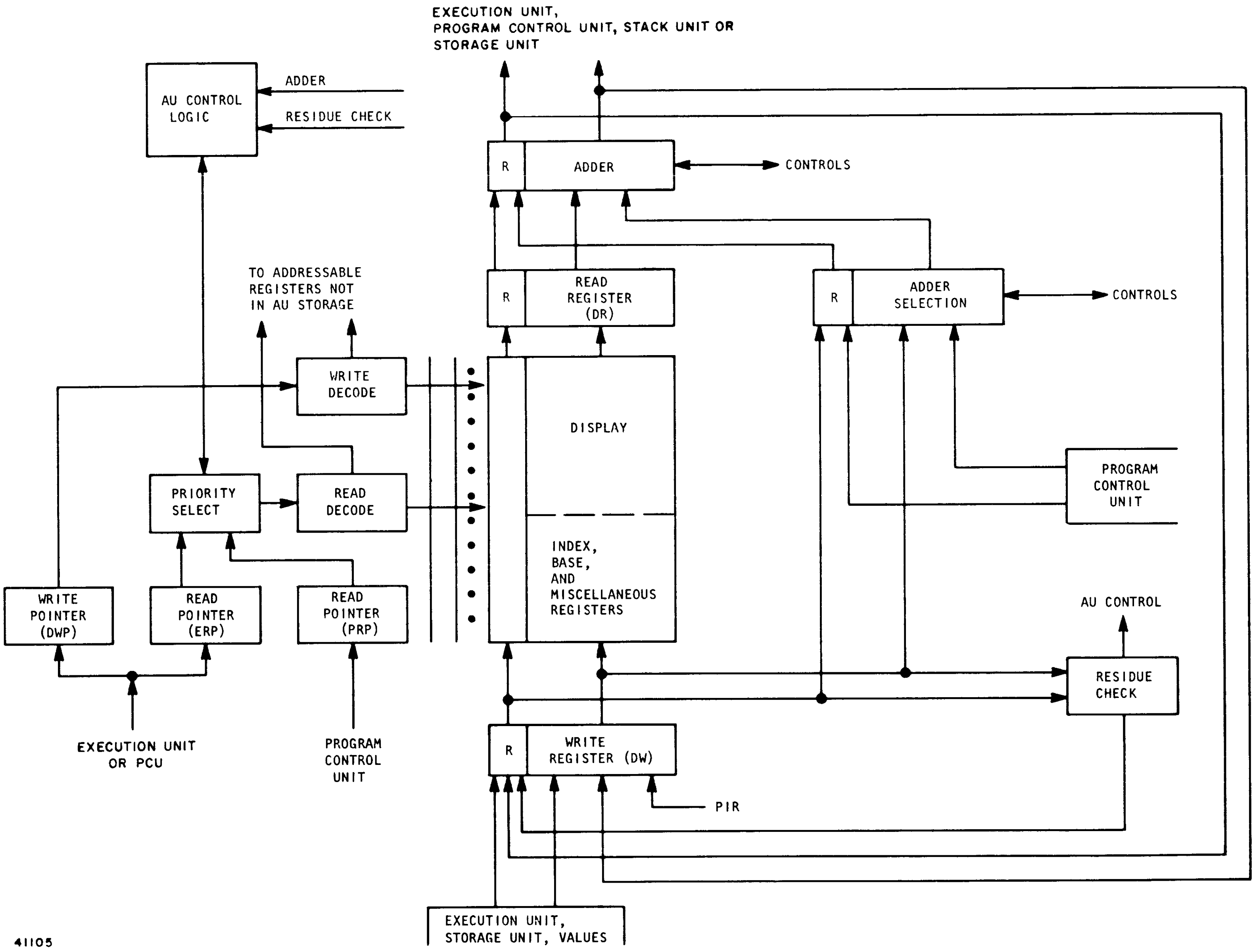


Figure III-1-9. Address Unit

circuits of the address unit.)

The display write (DW) register, buffers information being written into the address-storage area so that the controlling logic can release immediately instead of waiting for the storage cycle to complete. Residue of the address being written is checked when the address is contained in the DW Register. The DW register is also used as one of the two adder inputs. In addition, it serves as an accumulator when more than one adder cycle is required. For example, in string processing, two adder cycles are required when adding the index, a constant, and a base to derive an address.

All writing into the storage area is controlled by either the EU or PCU, but the DW register may be loaded by the EU, or PCU when its contents are to be used as an adder input. The DW register contents are written into the storage only when a write decode selects a storage location and a write strobe is generated by the AU control logic.

The DW register can be loaded in true or complement form so the adder may be used for addition or subtraction. Subtraction is used when performing limit comparisons.

The display read (DR) register is the output register of the address-storage area. The DR register contents are always an input to the address adder. The DR register is always loaded with any information read from the address-storage area. When a limit comparison is required prior to a write cycle, the DR can be used to buffer the contents of the limit register so that the write cycle may be initiated while the add cycle for the limit comparison is in progress.

For write operations, the hexadecimal address of the addressable register is entered directly into the six-bit display write pointer. The write-decode circuitry then selects the appropriate storage location and the write cycle commences. For addressable registers not contained in the address-storage area, the decode logic selects the appropriate data paths to facilitate updating the addressable register.

There are two read pointers, the PRP which is used exclusively by the PCU, and the ERP which is used exclusively by the EU. Because either the EU or the PCU may request a read operation by loading the appropriate read pointer, priority-selection logic is required to resolve access conflicts. In general the PCU has priority, but must release the AU after one cycle. The EU has priority when the PCU is in a hold condition and can maintain control of the AU until it has finished any required operations.

Access to the AU is granted by the control logic. Like the write pointer, the read pointers are loaded with any address of an addressable register. If the address is in AU local storage, a read cycle occurs and the contents of the addressed register is subsequently loaded into the DR register. When an addressable register not in local storage is read, the contents of the register are gated to the execution unit by the read-decode logic.

The two-input aaaress adder adds the contents of the DR register to the contents of either the DW register or a value (such as a displacement for Value Call) inserted directly through the selection logic from the PCU.

The adder output may be routed to the EWR in the EU; to the F, S, or LOSR in the stack buffer; to the MAR or SIR in the storage unit; to the PA or PIR in the PCU, or to the DW register for use in a subsequent adder cycle.

The AU control logic monitors the output of the adder to detect adder overflow, all 0 bit results, or the results of adder comparison.

The AU control logic provides the timing and control signals necessary for operation of the AU. The basic operations performed by the AU include a read cycle, a quick-write cycle, and a read-add-write cycle.

The read operation is performed in two machine cycles. During the first cycle, the contents of the addressed location are read, and during the second cycle the information is routed thru the adder, combined with any other input selected to the adder, and the result is routed to the appropriate destination. For example, in address calculation for Value Call, the appropriate D register is read, the displacement is added to the contents of the D register, and the result is sent to the SIR to be queued for SU action.

The quick-write cycle is used by the EU to update information in an addressable register. The two-cycle operation consists of a load cycle and a write cycle.

The read-add-write cycle is used to modify the contents of an address-storage location when the modification is based on the current contents of the location. The operation is performed in three cycles.

EXECUTION UNIT

GENERAL

The execution unit (EU) is the final stop in the processing pipeline. The great majority of instructions are not completed until the execution unit has processed the instruction. The execution unit is the only unit in the processor which operates on value data. It also has some control word formation and address calculation responsibilities. This unit includes storage for the two top of stack operands, A and B, and may temporarily store parts of character strings on which it is operating. The pipeline processing technique is implemented further in the EU. There exists within the EU, three distinct processing stations: an operator level, a command level, and a store level.

INPUT QUEUES

The execution unit like the program control unit is queue driven. All operations and operator associated data are placed into the queues of the execution unit by the program control unit. The value data inputs are supplied by the storage unit or the communications unit. Since the queues are implemented by memory chips, simultaneous read and write can take place. The purpose of each of the various EU input queues is described in the following paragraphs.

OPERATOR QUEUE

The operator queue (QQ) is the storage area for micro operators pending EU processing. The use of the queue allows the PCU and the EU to operate independently. Information to be written into the operator queue is placed in the order-code-write register (OWR). The PCU loads the OWR, then initiates the write cycle. The micro operator and control information in the OWR are then written into the address selected by the contents of the operator-queue write pointer (OQW). Following the write, the OQW count is advanced. The operator queue contains eight 12-bit locations; each location is loaded with the following information:

a. Bits 0 thru 7 contain the micro operator code.

b. Bit 8 (END) is set if this micro operator is the last in a program operator sequence.

c. Bit 9 (RPT) is the report bit, which when set, directs the EU to notify the PCU when execution of the micro operator is complete.

D. Bit 10 (ESB) when set, indicates that the B operand location will contain data at the start of the micro operator execution.

e. Bit 11 (ESA) when set, indicates that the A operand location will contain data at the start of the micro-operator execution.

The operator queue is read only by the EU. The contents of the operator-queue read pointer (OQR) provide the queue address for the read operation. The read address is updated when the EU is executing the last routine of the previous micro operator if the queue is not empty, no hold condition exists, and the EU is not generating a micro operator. When the address is updated, the new information becomes immediately available at the output of operator queue. When the output of the operator queue is to be used, the micro-operator information is entered into the EU operator registers.

EU DATA QUEUE

The EU data queue (DQ) is an eight-word IC local storage area organized as four two-word groups. The purpose of the data queue is to buffer data generated by the PCU preprocessing (look-ahead) logic. Literals, Value Calls, and concatenated operators cause space to be reserved, and, in the case of literals, data to be entered. Each of the four group locations has a validity bit associated with it. When the data from a look-ahead fetch becomes available, the storage unit places it in the reserved data-queue location, and sets the associated group validity bit. The second word of each group is loaded with the second-half (least-significant portion) of double-precision operands, or, in the case of a non-present data descriptor, with the absolute address from which the descriptor was fetched. For single-precision words other than non-present data descriptors, the second word location in the group is left empty.

Writing into the data queue is performed by the PCU, the storage unit, or the communications unit. Reading from the data queue is performed by the EU only.

DATA QUEUE WRITE OPERATION

The input register for the EU data queue is the EU write register (EWR). When a write cycle is initiated, the contents of EWR are entered into the address selected by the write pointer (DQW) and flip-flop WLSQ. The data group to be loaded is selected by decoding the contents of DQW, and WLSQ selects which of the two word locations in the group is to be loaded. (The first word of the two word group is loaded when WLSQ is reset.)

The PCU keeps track of the next group to be filled by use of the data queue assignment (DA) counter. When the PCU encounters a literal value, it loads the value into EWR, transfers the DA count into DQW, initiates the write cycle, and then advances the count in DA.

When an operator requires data not contained in the program code string, the PCU posts in the storage unit operations queue a request for the data together with the main memory address of the data and the contents of the DA counter. Then the PCU advances the DA count, thus reserving space for the requested data. When the storage unit or the communications unit obtains the requested data, the word is entered through the EWR into the reserved data-queue location.

In order that the EU may know when a group location has been loaded and whether a single precision operand has been written into the group location, there are separate data-queue-valid (DQV) and queue-single-precision (QSP) flip-flops associated with each data-queue group location.

When a look-ahead fetch is performed by either the SU or Comm Unit, the QRL (queue read lookhead) flip-flop is set. Excluding certain conditions, the QRL, when set, allows the EU to perform an early evaluation of the tag information in the EWR while writing the EWR contents into the data queue. (The evaluation of tag information determines what action should be taken to process the fetched item.) An early tag evaluation can not be performed when an item in the EWR is identified as a non-present descriptor, DP operand, or has a tag of 2, 4, 6, or 7.

DATA QUEUE READ OPERATION

The EU reads the information from the data queue in the same order in which the PCU reserved the group locations. Read addressing for the data queue is accomplished by decoding the contents of the data-queue-read (DQR) pointer, which selects the group to be read, and of flip-flop RLSQ, which identifies which of the two words in the group is to be read. The contents of the word location selected by DQR and RLSQ are available on the queue output lines. To use the information, the EU simply gates the information onto the T bus. The EU will not attempt to use the information until the associated DQV bit is set.

EU LOOK-AHEAD DATA QUEUE

The control information provided with each group written into the EU data queue is also entered into an EU look-ahead data queue (EL queue). This control information includes a copy of bits 45 thru 50 of the data being entered into the data queue and a bit which indicates if the associated data-queue word is in integer form.

The look-ahead information is addressed by the data-queue read pointer, but the address decoding is such that the look-ahead information is available at the EL queue output when the preceding data-queue group is being addressed. The look-ahead information is also used to determine what commands should be called to process the associated operands.

K AND L QUEUES

The K queue (KQ) and the L queue (LQ), which are used in conjunction with the operator queue, contain coded variant information for use in execution of the related micro operator. The K and L queues each contain eight 10-bit locations. The operator variant codes are eight bits in length, and in the other two bits of each location residue is maintained on the variant code. Usually the variant information supplied with a micro operator is taken directly from the second or third syllable of the program operator in the IER. Such variables could be used to identify the length of a selected field or a bit location in a word. In other cases, the variant information is generated by the PCU to alter or further define the related micro operator.

Addressing of the K and L queues for both read and write operations is accomplished by use of the operator queue read and write pointers. Therefore, each time an entry is made into the operator queue, an entry is also made into the K and L queues although this information may be all 0 bits. Likewise, when an operator-queue location is addressed by the read pointer, the associated locations in K and L are also addressed.

The first 16 bits of EWR are used as the input register for the K and L variant information. The variant-residue (VNR) register in the PCU supplies the residue for K, but the residue for L is written directly into the queue from the PCU residue generator.

The outputs of the K and L queues are entered into the K and L registers in the EU. Like the operator register, the K and L registers may be loaded directly from the PCU if the queue is empty.

PIR AND PSR QUEUE

The PIR and PSR queue (PQ) contains the PIR and PSR values associated with the micro operators in the operator queue. (The PIR and PSR, together with the contents of PBR, identify the absolute address and starting syllable position of the associated program operator.) The PIR queue is loaded with the contents of the program current and current syllable registers. The PIR queue is written into at the same time as is the operator queue and uses the same write pointer (OQW) for addressing.

A separate read pointer PQR is used for the PIR queue. The PQR is counted in a manner which causes the PIR and PSR information associated with each micro operator to be available at the output of the PIR queue when the micro operator is at the store level of execution in the EU.

Usually the output of the PIR queue is not used. However, when the EU starts interrupt processing, the output of the PIR queue may be gated into the PN and NS registers to provide the PSR and PIR values associated with the program operator being re-executed.

LEXIC LEVEL QUEUE

This queue (LLQ) contains either the lexicographical addressing level of a particular operator in the operator queue or the value which was in the PCU display read pointer (PRP) when the associated micro operator was in the IDR.

The current lexicographical level is written into the queue any time the PCU processing sequence for an operator includes timing count Phase 9 (PH9). The PRP read pointer value is entered into the queue when either a Name Call (NAMC) or Value Call (VALC) operator is processed by the PCU. For all other operators, no entry is made in the queue.

The information in the LL queue is then available at the output of the queue for use by the EU during the execution of the related micro operators.

Writing into the LL queue is accomplished by use of the operator queue write pointer (OQW), and reading from the queue is accomplished by use of the PIR queue read pointer loaded into the IDR. (PQR).

MAJOR EU DATA CIRCUITS

As shown in figure III-1-10, the EU contains five full-word data registers, three data buses, local storage areas, and adder and shifting mechanisms. These data circuits are described in the following paragraphs.

ADDER

The most obvious major item required by the execution unit is a fast adder. The adder used comprises a single-carry-save adder which drives a carry-propagate adder. The adder fully propagates all carries each time it is used, employing a high-speed carry look-ahead technique.

To accommodate all single-precision mantissas, the mantissa portion of the adder is 39 bits wide with an extension for multiply and divide. This portion of the adder has three inputs. During multiply, two of the inputs are used to insert selected multiples of the multiplicand, and the third input inserts the accumulated product. By this method, six bits of the multiplier are calculated on each adder pass. The separate, seven-bit exponent adder is included to facilitate single-precision, floating-point arithmetic.

The adder is also used to perform double-precision arithmetic. This is accomplished by buffering the double-precision operands and intermediate results in the local EU memory area.

The adder provides an efficient means for executing logical operations. The single-sum and single-carry outputs, which are developed for each bit position by the first stage of the adder, are selectively gated to derive the logical AND, OR, and EXCLUSIVE OR functions.

The adder output is checked for residue or continuity errors. If either input to the adder is invalid, a continuity error occurs. If the sum is incorrect, a residue error is detected. Separate residue is maintained on the exponent and mantissa during floating point arithmetic operations.

SHIFT MECHANISM

The barrel switch is a 48 bit wide, end-around shifting mechanism. It is capable of shifting by any number of bits from 0 to 47, to the left. The barrel output is gated with enabling logic, which allows any number of bits from 0 thru 48, out of the barrel, from the left or the right.

The barrel has three shift stages. The first stage shifts the input data by multiples of 12, the second stage shifts the input by multiples of 3, and the third stage shifts the input by multiples of 1. The multiples are selected by the contents of the shift register in which two bits are used to define the selected multiple for each shift stage. The shift amount, coded in successive bit values of 1, 2, 3, 6, 12, and 24, may be entered into the shift register in either true or complement form. When this amount is loaded in complement form, a right shift effectively occurs; when loaded in true form, a left shift is executed.

The barrel output gating enables a field of any selected length or the entire word to be extracted from the barrel. The allow register contents determine the number of bits to be extracted at the barrel output. Additional gating allows this field to be selected either right or left justified.

If either the allow register or the shift register was not loaded when the barrel is used, the output of the barrel is declared invalid and a continuity error is subsequently detected. Residue is maintained on the contents of both the shift and allow registers. A residue error in the contents of these registers results in the detection of an EU residue error. Residue is also maintained on the data transferred through the barrel. If the barrel output has a residue error, an EU residue error condition is detected.

LOCAL MEMORY

The execution unit also includes a local memory for storage of the top of stack operands, character strings and intermediate results. Memory chips are again used for this purpose. Parity is maintained on each word in the operand storage, except for DP arithmetic, and the parity is checked each time a word is read from local storage. The local memory is split into two parts in order to be able to read two

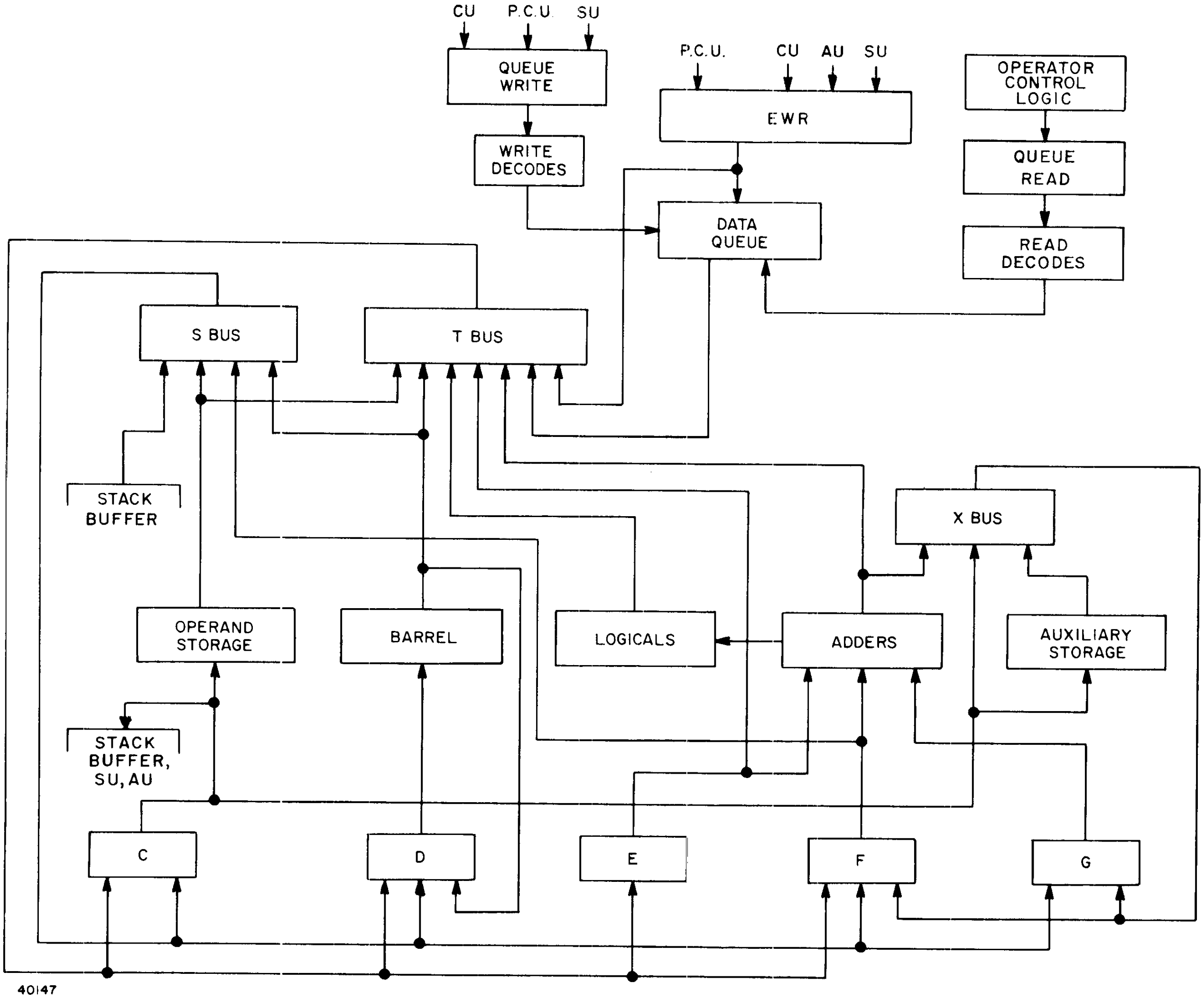


Figure III-1-10. Execution Unit, Major Data Circuits

words simultaneously. The larger part is called the operand storage; the smaller part is called the auxiliary storage. The memory space is allocated as follows:

Operand Storage (52 bits wide)

1. A/B space 0: Two alternate double-precision locations.
2. A/B space 1: Two alternate double-precision locations.
3. R space: Four words.
4. W space: Four words, which is sometimes used as two alternate locations of two words each.

Auxiliary Storage (49 bits wide)

1. H: one word.
2. J: one word.
3. Bad C: one word.

The time required to cycle the memory chips is comparable to the time required to propagate the worst case carry through the adder, so the memory fits neatly into the overall timing scheme. To maintain the fastest possible single precision execution times, single-precision operands are buffered in the adder and barrel input registers rather than in operand storage.

The purpose for having alternate locations for A and B is to be able to form a new A or B without destroying the original input operands. This is desirable for diagnostics and error recovery. Two values for the A and B operands are maintained in storage during the execution of each program operator. The contents of the A and B operands at the start of the operator are held unchanged in the initial allocation area, while the current allocation area is successively loaded with the current values of the operands. When the final result of the program operator is obtained, the allocation is interchanged, so that the final copies of the current operands are designated as the initial configuration for the next program operator.

The adjustment of the memory allocation is controlled by use of the EU allocation flip-flops. These allocation flip-flops also facilitate rapid execution of stack related operators.

When three top-of-stack operands must be maintained in the EU, as during execution of the Rotate Stack operator, the W storage area is used to maintain the third top-of-stack item.

DATA REGISTERS

The execution unit includes five full-word data registers (C, D, E, F, and G registers). The A and B words of the B 7700 are implemented in the EU operand storage, as previously described. Intercommunication between the data registers is carried out by the use of three data transfer buses (T bus, S bus, and X bus).

There is one register which loads the operand storage, and which is the EU output register for communication with the rest of the processor. This register is the C register. All data error checking, as well as parity and residue generation is accomplished when a data word is in the C register.

The D register is the only barrel input register. Any data entered in the D register is immediately shifted by the barrel and allowed out of the barrel, in accordance with the contents of the shift and allow registers. Because the D register is the only barrel input no selection gating is necessary.

Because the adder is used often, it is especially important that the delay between the registers and the adder be minimized. This is accomplished by transferring the contents of the adder-input registers into the adder with no selection gating. Therefore certain registers, namely the E, F and G registers, are dedicated as adder input registers, and actual adder selection is accomplished at the input to these registers. The E register has special significance in that it is the accumulator, while the F and G registers supply new inputs. The E and F registers also have six-bit exponent fields which are inputs to the exponent adder.

Unless the contents of these data registers are specifically "held", the contents must be transferred during every machine cycle or the word will be marked invalid. Therefore to maintain validity of the A and B operands when they are not being used for processing, the E register and T bus are often assigned to keeping the A operand valid, while the F register and the S bus are assigned to keeping B valid.

DATA TRANSFER BUSES

Placing the outputs of the adder, the barrel, the registers, and the operand storage onto buses which go to the registers (which in turn are the inputs to the adder, barrel and operand storage) is the fastest way to transfer data without using excessive amounts of hardware. The data transfer buses, which consist of multiple input OR gates, are designed to match the stack type features of the simple operators and the micro operators. This approach led to the implementation of two major data transfer buses, one serving as the top element of the "bus stack" and the other serving as the second element. In simple operations on single precision data, the A operand is on the T (top) bus, and the B operand is on the S (second) bus. During complex operations, data segments will be moved onto the T and S buses as inputs for micro operators commands.

The processing requirements of simple and micro operators suggest particular data paths for the buses. In particular, each bus must go to those registers which give the bus access to the adder, the barrel, the operand storage, and areas external to the execution unit. Inputs to the stack go to the top position so the operand input (for Value Calls, Literals, etc.) from the data queue or queue input goes to the T bus only.

Arithmetic results are left in the top position, so the adder output goes only to the T bus. The extension of the stack (stack buffer) goes to the S only to handle stepping up ("popping") the stack. When executing a monadic operator with both A and B full initially, it is desirable to load B back onto the S bus at the end of the operator, so the operand storage output is routed to the S bus. However, it is also desirable that the inputs to the "bus stack" for micro operators go to the top position so the operand storage also goes to the T bus. Field results are left in the top position, so the barrel output is applied to the T bus. Certain other paths to the buses are available in order to hold operands on the buses or to facilitate particular algorithms.

Various special paths from the buses to the registers exist to speed up multiply and divide and these special paths are also useful for scale left and scale right. There is one other special path, called the X (auxiliary) bus, which is not part of the "bus stack". The X bus is used during multiply and other extended arithmetic calculations to speed up these computations. During multiply, the X bus transfers to the adder registers a quantity equal to three times the multiplicand.

MAJOR EU CONTROL CIRCUITS

Figure III-1-11 shows the major EU control circuits. These circuits include the operator and command registers; the K, L, and R registers; A, B, and W control registers, and the A, B, R, and W pointer registers.

The A and B control registers are used to maintain ready access to the control information provided with the A and B operands. Such information includes the tag bits, the sign bits, and a bit which indicates if the related data is in integer form. When three top-of-stack operands are maintained in the EU, the W control register is used for the control bits of the third operand. If during execution of an operator, the control information may be altered, the contents of these control registers are declared invalid until the updated information can be loaded from the C register. The registers are initially loaded as the data becomes available to the EU.

The pointer registers (AP, BP, RP, and WP) are used for addressing a particular segment of a word contained in the EU local storage. To extract part of a word from EU local storage, the pointer contents are decoded to provide a word address and a digit location within the word. The word portion is used for reading the proper word from operand storage and the digit information is used in setting up the shift and allow registers so that the desired segment can be extracted by use of the barrel. For repeated operations, the contents of the pointer register is updated by use of a special adder, and the updated contents are then loaded back into the pointer for use on the next cycle.

The K and L registers receive the variant information from the K and L queue, or, for dynamic field operators, from the C register. The information in K and L may be decoded or, for field operators, may be routed thru the K-L adder, to provide information for extraction of a desired field. The output of the K-L adder is K-L+1, therefore in field and bit operators when K contains the starting bit, and L contains the length, the K-L adder output gives the shift amount required to right justify the desired field in the barrel, and the L register contents provide the input for the allow register.

When the K register contains the scale value for scale operators, the amount in K is transferred through the K-L adder to the R register. The R register is a repetition counter. For scale operators, after each cycle, the count in R is decreased by use of the subtract logic and the difference is loaded back into R until the operation is complete. The R register also serves to check the binary value of the K-L adder output. (Effectively, this conversion consists of dividing the binary input by three, then placing the remainder as the two least significant bits of the quotient.) The conversion of K-L and L values are loaded into SH and AL.

The remaining major control registers, the operator registers and the command registers are the primary control registers of the EU. The operator registers are used to decode the micro operators and the command registers are used to generate the data transfer signals

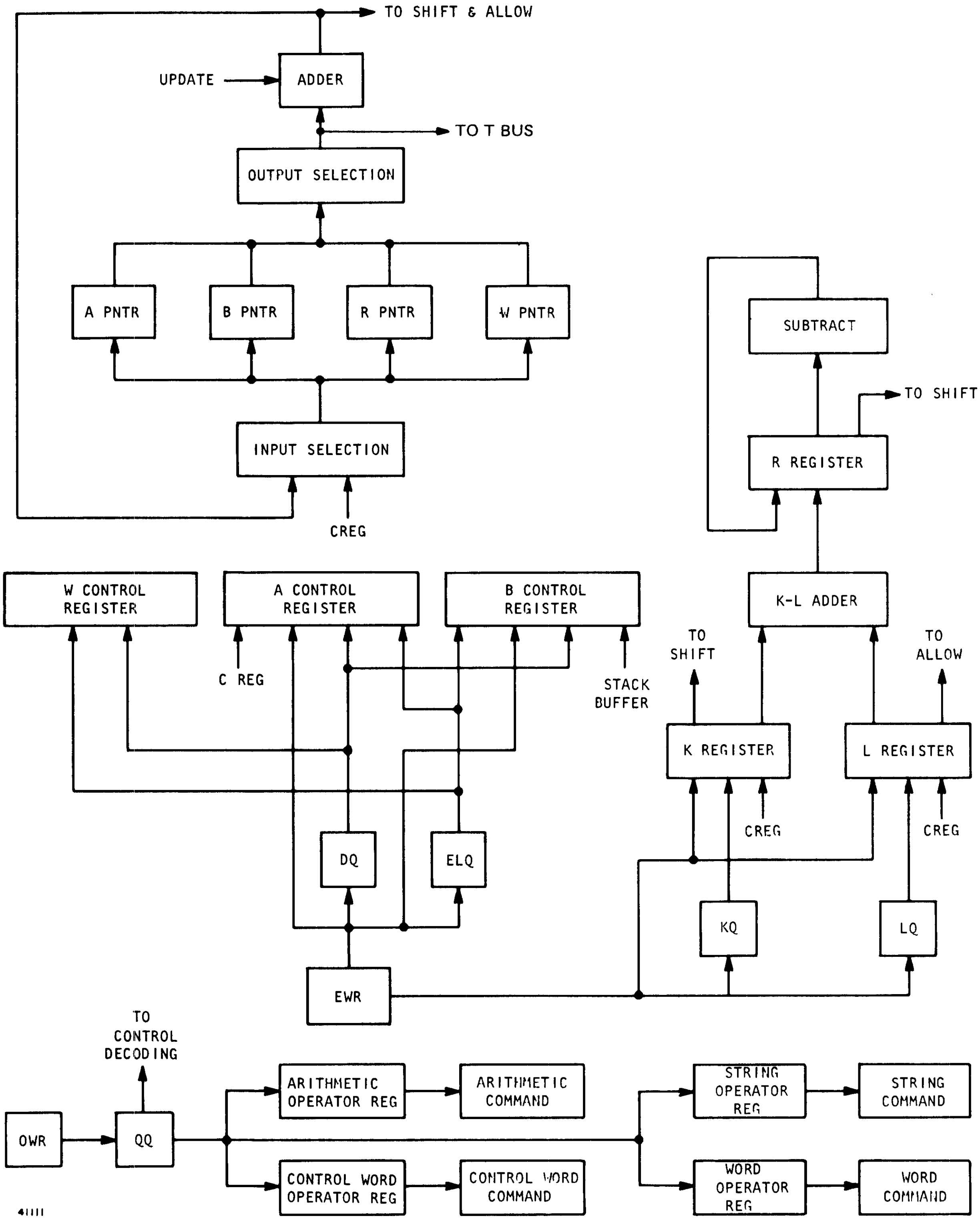


Figure III-1-11. Execution Unit, Major Control Circuits

for execution of the micro operators. The use of these registers is described in the following paragraphs.

BASIC EU OPERATION

As previously described, the EU does not operate on basic program code (program operators), but rather it executes micro operators forwarded to it by the PCU. One or more micro operators are executed in completing each program operator. The major advantage in this method is that many program operators are quite similar in execution, so that the same micro operator may be issued for a group of program operators.

The micro operators are grouped by function into four families: the arithmetic family, the string family, the control word family, and the word family. To simplify micro operator decoding, separate operator and command registers and separate timing counters are provided for each micro-operator family. The fifth and sixth bits of each eight-bit micro-operator code identify the family to which the micro operator is assigned.

When a micro operator is read from the operator queue, it is entered simultaneously into each of the four operator registers. However, the fifth and sixth bits of the eight-bit micro-operator code are not loaded into the operator registers, but are decoded to enable the use of only the appropriate operator register. At this point, the timing counter for the selected family is also enabled. The micro operator is decoded, and the first of a sequence of operator commands is generated. One coded command is entered into the appropriate command register at each count of the timing counter. Each command is held in the command register for only one machine cycle, during which the command is decoded.

The use of command registers allows groups of sub-commands to be generated as efficiently as possible. The sub-commands are data transfer and control signals. The command register generation of these data transfer signals determines the use of the data registers and data bus networks in the EU. The signal which loads a command into the command register is also used to route the information from the data buses to the data registers.

On the next effective clock, the command decode causes the necessary transfers of the data units thru the selected logic (such as the adder or the barrel shifting mechanism) to the data buses. Because the command registers are loaded with a new command on each effective clock, on a given clock, one command is loading the buses while the next command is selecting the buses to the registers. The only requirement for compatibility between the two commands is that the resulting bus configuration of the first command is compatible to the input bus requirements of the next command. (The "bus configuration" refers only to the presence or absence of data on each bus.)

For example, if data in the E and F registers were to be added by one command and the next command required that the sum be routed thru the barrel shifting mechanism, then the command decode of the first would generate a data transfer signal which would route the adder output to the T bus, while the signal which loads the second command would simultaneously cause the contents of the T bus to be routed to the barrel input register (D register).

The next effective clock would then load the second command, while the sum on the T bus would go to both the D register for use in the second command and, automatically, to the C register for error checking and for storage of a copy of the result of the add operation. This automatic routing of the results of each command to the C register is the "store-level" action required for each command.

When the contents of the C register are to be transferred (unloaded) to a location other than EU local storage (such as transfer to the SU, the AU, or some EU register), the transfer is controlled by a coded store-level command entered into the store (ST) register.

The execution of each micro operation entails the processing of a series of one or more commands. However, although the enabled command register is loaded with a new command on each effective clock, not all of these commands are issued by the decode of the micro operator in the operator register.

Each command register is allowed to reload itself with successive commands when a natural unit of actions requires a certain sequence of commands. Such a sequence of commands is called a routine. At the end of the routine, the operator register is allowed to load the command register and the timing count is advanced. The command load for the last command of a routine generates a final-command signal, which enables the operator-register output and the timing count.

For each micro operator, when the last command generated by the operator-register decode logic is loaded to the command register, the operator-level signal, final routine, is produced. When the final-routine signal has been generated, the operator register is loaded with the next micro operator from the operator queue.

The last micro operator of each program operator is specially tagged by the PCU. When this micro operator has been fully executed, the EU prepares for the next program operator. This preparation consists of checking for interrupts and of switching the allocation of the EU local storage areas so that the final stored results of this program operator are designated as the initial A and B configuration of the next program operator. During execution of this next program operator, this initial operand configuration will be saved, while the current A and B operands are repeatedly entered into the alternate A and B storage locations. This initial configuration must be maintained for possible use in error recovery and diagnostics.

EU ERROR DETECTION METHODS

Three error-detection systems are used in the EU, namely, parity checks, residue checks, and continuity checks. Parity is used to detect errors in EU local storage and in data received from other units. Mod 3 residue is used to detect errors anywhere in the EU data paths and data registers, but not in either the EU local storage or most control registers. Also, the residue logic checks addresses sent to the EU from the AU or SU. In addition, residue is the primary means for detecting errors caused by an extra data transfer signal. Continuity checking, the use of a validity bit which indicates if the current contents of a register are valid, is used to detect missing, and sometimes extra, data transfer signals for the most commonly used EU data paths.

The implementation of these methods produced the following results:

a. The EU continuity logic provides a considerable check on the sequence of operators and control bits sent to the EU by the PCU.

b. The EU residue system detects errors in arithmetic operations, field and bit extractions and insertions, and barrel shift operations.

c. The EU parity logic detects data storage and transfer errors involving the EWR, the EU local storage, the associative memory, and the stack buffer.

d. Continuity and residue checks detect most errors involving the generation of subcommands and data transfer signals.

STACK BUFFER

The stack is an area of memory assigned to a job to provide storage for basic program and data references. The stack also provides temporary storage of data and job history. When a job is activated, a linkage between its stack and the top-of-stack operands (A and B) is established by the stack pointer register (S), which contains the memory address of the last word pushed into the top of the stack. The stack buffer serves to extend the stack memory area into processor local IC memory and to provide quick access for stack manipulation by the execution unit. (See figure III-1-12.)

STACK BUFFER FUNCTION

The primary purpose of the stack buffer is to hold, locally, a portion of the stack environment. New stack items are entered into the stack buffer from the EU in such a manner that the last item placed in the stack is the first to be extracted. After the two top-of-stack positions in the EU are filled, loading a third operand onto the top of the stack causes the first to be pushed into the stack buffer. As entries are pushed into the stack buffer, and saturation is attained, a segment of buffer entries is autonomously moved into main memory so that the stack buffer maintains the top area of the stack memory area. Any stack adjustment to main memory is always accomplished in multi-word segments in order to take full advantage of the phased memory system. Thus, the stack buffer tends to capture the current addressing environment of the executing program stack.

In the B 7700 processing system the stack buffer can be directly addressed, within limits, as if it were actually an area of main memory. The direct addressing of the stack buffer is transparent to the programmer. Therefore, knowledge of this action is not necessary for the programmer.

STACK BUFFER OPERATION

As shown in figure III-1-13, the major elements of the Stack Unit include the stack buffer, stack link (KL) register, stack vulnerability (KV) register, limit, local, and vulnerability comparators, stack length (KLN) register, subtractor, and associated stack controls.

The stack buffer is a 32-word local storage area consisting of four 8-word segments. Each storage word in the stack buffer consists of 54 bits. These bits comprise 48 bits of information, three tag bits, a parity bit, and two memory error bits. The memory-error bits are defined as follows:

SB53	SB52	
0	1	No error.
1	1	1 bit error during a fill operation.
1	0	Error other than a 1 bit error during a fill operation.

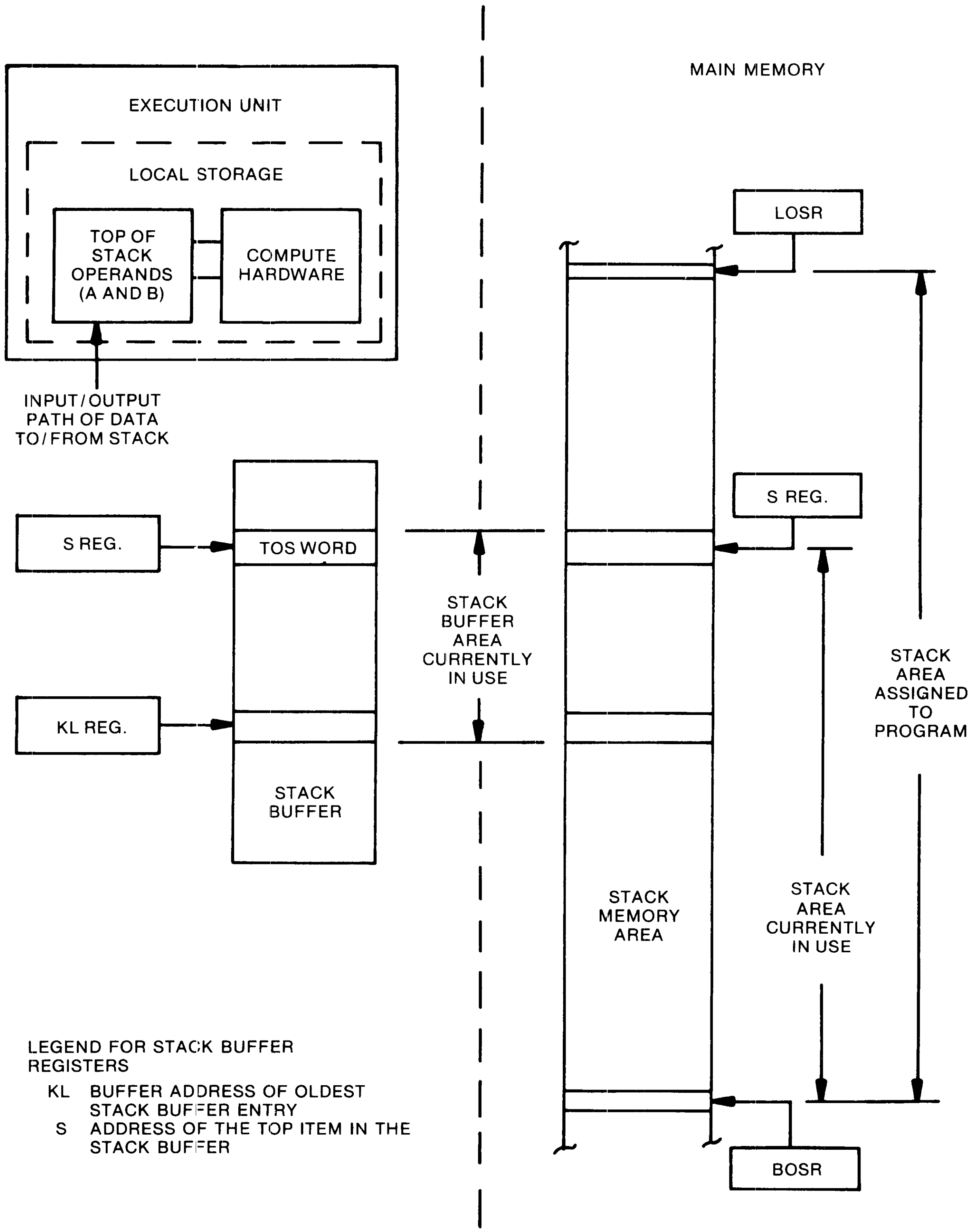

EXECUTION UNIT
LOCAL STORAGE
TOP OF STACK OPERANDS (A AND B)
COMPUTE HARDWARE
INPUT/OUTPUT PATH OF DATA TO/FROM STACK
S REG.
TOS WORD
KL REG.
STACK BUFFER
STACK BUFFER AREA CURRENTLY IN USE
MAIN MEMORY
LOSR
S REG.
STACK MEMORY AREA
STACK AREA CURRENTLY IN USE
STACK AREA ASSIGNED TO PROGRAM
BOSR
LEGEND FOR STACK BUFFER REGISTERS
KL BUFFER ADDRESS OF OLDEST STACK BUFFER ENTRY
S ADDRESS OF THE TOP ITEM IN THE STACK BUFFER

Figure III-1-12. Stack Buffer and Stack Memory Area

The buffer may contain from 0 to 32 words of the active stack. All valid contents of the buffer have contiguous main-memory addresses. The main-memory addresses for the contents of the stack buffer are indicated by three 20-bit registers: the S register, the stack link register, and the stack vulnerability register.

The stack register (S register) holds the address of the top item in the stack buffer. Also functions as the stack buffer read/write pointer for POP/PUSH operations.

The stack link register (KL) contains the address for the bottom (or oldest) item in the stack buffer. In addition, the KL register provides memory address information for Comm Unit functions as the stack buffer read pointer during empty operations, and as the stack buffer write pointer during fill operations.

The third register, the stack vulnerability register (KV) along with four top of stack vulnerability flip-flops (TKV0 thru TKV3) are used to identify fetched buffer locations between S and KV (S $\geq$ MAR > KV). The purpose of this logic is to detect when the EU pops stack buffer contents that were previously fetched and placed in the EU data queue by the Storage Unit. This condition will occur during certain program applications.

Any time the Storage Unit fetches an address between S and KV, the TKV flip-flop corresponding to the relative address (0 thru 3) above the address in the KV register is set. If the EU executes a pop out of an address identified by a set TKV, the stack buffer sends signal KRSN.1 to the EU. As a result, the EU notifies the PCU of change of direction, a clear queue is issued by the PCU, the Storage Unit fetch is again executed, and the instruction, which follows the instruction calling the fetch, is issued to the EU by the PCU.

Initially KV is set to S-4 during the return or Set Processor Register "S" operation. In all following operations, any change of S contents results in a corresponding change of KV contents. Consequently, the contents of KV always remain at S-4.

Any set TKV flip-flop is cleared if four consecutive pushes into the stack occur before a pop is executed. All set TKV flip-flops are reset if a return/SPRR, store, or push condition occurs.

In local return or local SPRR operation, the subtractor subtracts the contents of KLR from the least significant bits in the MAR, then adds to the difference the count of one. The result is placed in KLN to identify the new stack length.

The memory copies present (MCP) flip-flop, when set, informs the Storage Unit to store to main memory even though a store or overwrite is found local in the stack buffer. MCP is set anytime a fill operation is required before executing an EU pop operation. MCP is reset when a word is pushed into the top of the stack buffer.

The 5-least significant bits of the S register, MAR register, and KL register are decoded to select stack-buffer word locations, which range from 0 thru 31. When storing to or fetching from the top of the stack buffer, the S register is selected as the pointer into the stack buffer. MAR register is selected as the pointer source for the Storage Unit, KL register is selected as the pointer source for the Comm related operations; fill, empty and purge.

FETCHING THE TOP ITEM FROM THE STACK BUFFER

When the top item in the active stack is required in the EU for use as the A or B operand (an action referred to as a "pop" of data from the stack), a main-memory fetch is not required unless the stack buffer is empty.

If the stack buffer is not empty, the top item in the buffer is read from the buffer and transferred to the EU, and the contents of both the S register and the KV register are decreased by 1. At the same time, the stack buffer checks to ensure that the address in S is not less than the address in F. If S is less than F, then stack underflow is detected. (F contains the address of the top MSCW in the active stack.)

If the stack buffer is empty and a "pop" is required, first the address in KL is decreased by 4. The KL address is passed to COMM and used to fetch the top four words of the active stack from main memory. The contents of KL are used as a buffer address then counted up as each word is received. After the four words have been entered into the stack buffer, the contents of KL are decreased by four so that it again points to the bottom word in the buffer. (This action is called a "fill" operation.) With the required information now in the stack buffer, the "pop" described in the preceding paragraph occurs.

ADDING A NEW ITEM TO THE CONTENTS OF THE STACK BUFFER

When the A or B operand in the EU must be placed in the active stack to establish the proper operand configuration for execution of the next operator, the operand is transferred into the stack buffer. Because this operand is added to the top of the active stack. the

operation is referred to as a "push" of data into the stack. A "push" of data from the EU into the buffer will not cause a store to main memory unless the stack buffer is full.

If a "push" is required when the stack buffer is not full, the S register contents are increased by 1, then the operand to be added to the stack is transferred from the EU to the stack buffer and written into the stack-buffer location identified by the contents of the five least-significant bits of the S register.

If the MCP flip-flop is set, the KLN register is preset to 1 so that the first push is effectively into an empty stack. Also, the contents of the S register (S + 1) are loaded into KL to indicate that only one location is valid in the stack buffer.

If the stack buffer is full and a push is to be performed, four buffer words are stored to main memory, and then the data in the EU may be placed in the stack buffer. (This operation is referred to as an "empty" operation.)

This is accomplished by requesting COMM to perform an overwrite of the four oldest words into memory. The KL address is passed to COMM and used as the start address for the overwrite. Because KL register functions as the read pointer for the stack buffer, COMM increases KL contents by 1 as each word is transferred to memory. At the same time, the contents of KLN are decreased by 1. After the "empty" operation, the push of new entries into the stack buffer from the EU may proceed as described in the preceding paragraph.

Each time a push is to be performed, the stack buffer compares the contents of the S register to the contents of the limit of stack register (LOSR). The LOSR contains the address of the highest usable location in the active program stack. If S equals LOSR, a stack overflow interrupt is generated.

FETCHING AND STORING OF LOCAL INFORMATION

The stack buffer maintains up to 32 words of the top items in the active stack. The storage unit can fetch from or store to any point in the buffer that is within the local addressing environment. Therefore fetch requests made by the EU for any of the items "captured" locally in the stack buffer can be fulfilled without a main memory access. When updated information is stored to an address which is local, the store operation is performed to the stack buffer.

However, if the contents of MA are equal to KL, or the MCP is set, the information is also stored to memory because it may be the least significant half of a double precision word. The local addressing environment includes all items between S and KL.

For local stores or overwrites, the selected SU data queue information is simply written into the stack buffer location pointed to by the least significant 5 bits of the MAR.

For local fetches, the selected buffer information is simple transferred to the EWR register. At the same time, a comparator determines if MAR is greater than KV and if S register is greater or equal to MAR. If both of these conditions are met, the operation is not a vector mode fetch, and the two least significant bits of MAR are set to some binary count of 0 thru 3, one of four TKV flip-flops is set. This identifies that the fetch was obtained from one of the top four locations in the stack buffer.

STACK BUFFER PURGE OPERATION

A store to main memory is initiated when the stack buffer is purged. A purge causes all buffer words with main memory addresses between S and KL to be stored to memory. After a purge operation, the stack buffer is marked empty. A stack-buffer purge occurs each time an SPRR63, SPRR52, RDLK, MVST or SLMT operator is executed.

If RDLK or SLMT is preceded by FMFR, stack purging will not occur.

ASSOCIATIVE MEMORY

The associative memory (ASM) is a general data buffer, implemented to provide fast access to frequently used variables and descriptors which are outside the area contained in the stack buffer.

The ASM is loaded with any item fetched from main memory except those items fetched during execution of the masked search for equal, linked list lookup, or fetch memory fail register or any operators preceded by a fetch main memory reference operator. As shown in figure III-1-14, the major elements of the ASM include the following:

a. Data Array - Consists of 1024 words, each word of which contains 48 data bits, three tag bits, a data valid bit, and an error bit. The 1024 words are divided into 64 blocks (0 thru 63), each block of which contains four four-word groups.

b. Address Array - Consists of 256 address words, each address word of which contains 12 most significant bits of main memory address plus 2 bits of residue. The 256 address words are divided into 64 blocks (0 thru 63), each block of which contains four address words.

c. Priority List Array - Consists of 64 (0 thru 63) priority words, one word for each one of the 64 address array blocks. Each priority word contains three two-bit priority codes. These priority codes identify the oldest ad-

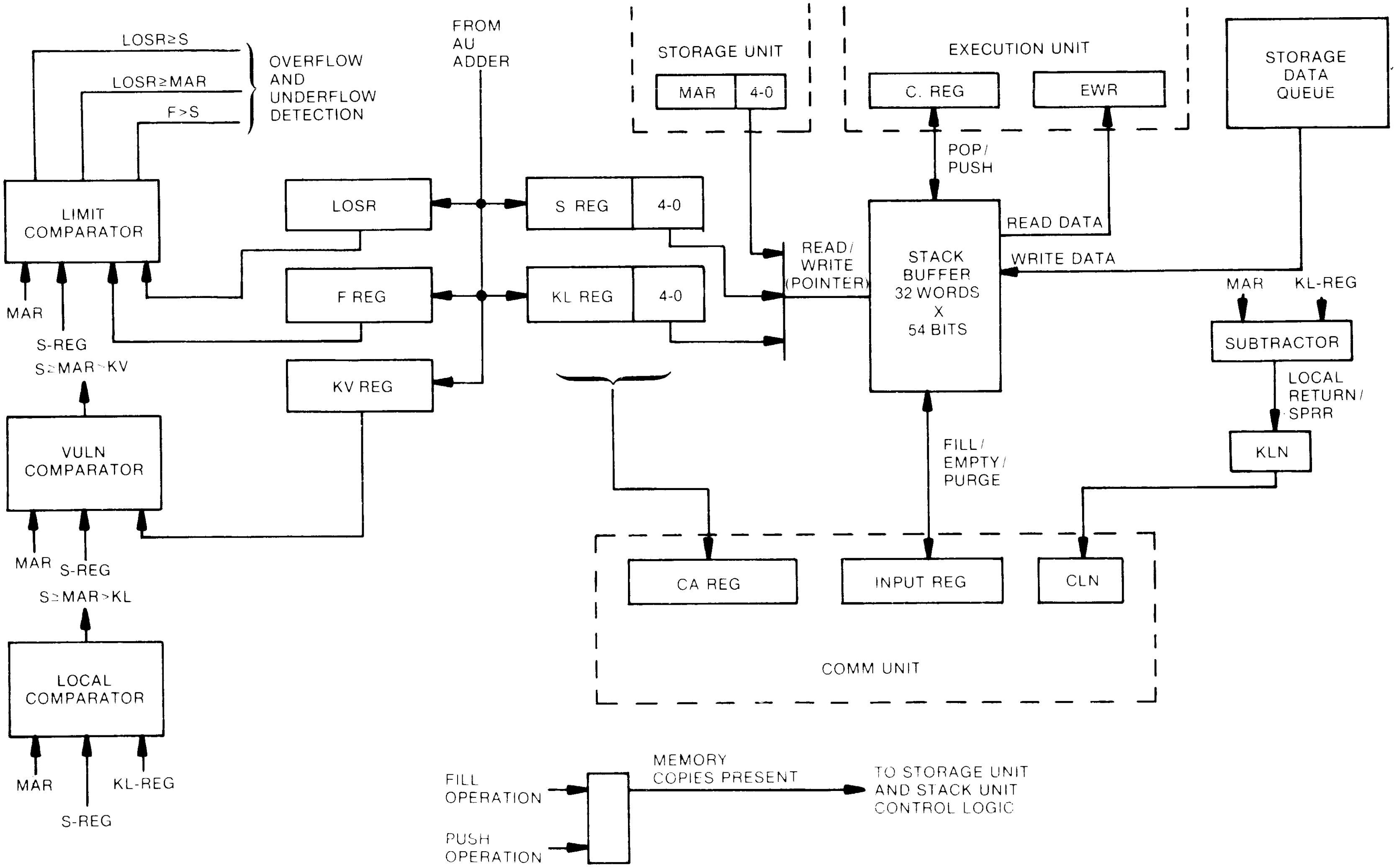


Figure III-1-13. Stack Unit, Block Diagram

dresses within an address array block as follows:

Priority Code Bits	Definition
5 and 4	Third oldest (or second newest) address word.
3 and 2	Second oldest address word.
1 and 0	Oldest address word.

NOTE

The newest address is derived from decoding Priority List register output into a two-bit code (7 and 6).

d. Priority List Register - Displays the word in the Priority List Array selected by memory address currently available in the ASM.

e. Address Residue Decoder - Subtracts the residue of bits 0 and 1 from residue bits 20 and 21 of the current memory address in the ASM. The resultant residue, along with the new address, is stored in the proper address array location and is also sent to the address comparator.

f. Address Comparator - Compares the 12 most significant bits of the current memory address and the associated residue with each one of the four address words (plus their residue) stored in the selected address array block. If any address word compares with current memory address, the comparator develops a two-bit code to identify that address word and a signal which informs the SU or CU that information was found locally in the ASM.

g. Priority List Shifter - Shifts the three two-bit codes of the PL register right or left by multiples of two bits. The output of the shifter is then written back into the addressed word location of the priority list array. The conditions for shifting the three two-bit priority codes are as follows:

1. When an address word in the address array block is designated as the newest address by a priority code, all codes to the left of the code identifying the newest address are shifted right by two bits.

For example:

NEW 0,1	Decode of PL Reg. PL7,6	PL5,4	PL3,2	PL1,0	
00	11	10	01	00	(orig. state)
	00	11	10	01	(next state)

2. When an address word in the address array block is designated as the oldest address by a priority code, all codes to the right of the code identifying the oldest address are shifted left by two bits.

For example:

NEW 0,1	Decode of PL Reg. PL7,6	PL5,4	PL3,2	PL1,0	
11	01	11	10	00	(orig. state)
	01	10	00	11	(next state)

h. Selection Logic - Selects two-bit code from address comparator when storage unit has control of ASM or when CU is performing stack empty operation or entering an error word into the ASM. Selects bits 0 and 1 of priority register when CU is performing a fill operation in the ASM.

ASM LOCAL OPERATION

On all local fetch, store, and overwrite operations, a check is made under control of the storage unit to determine if the requested information is currently contained in the associative memory. As shown in figure III-1-14, the contents of the MAR are presented to the data array, address array, and address comparator through the address selection and address decoder logic. The control circuitry of the ASM reads the four addresses in the address array block which is selected by address bits 2 thru 7, then the comparator circuit compares these addresses with address bits 8 thru 19, and the 2 bit residue decode.

If a compare exists within the four addresses, the address groups is declared local and a two-bit code, identifying this group, is passed onto the data array logic. In effect, the two-bit code provides the enabling controls for reading from or writing into the corresponding four-word group in the data array.

Other enabling controls are provided by the two least significant bits of address. These bits select the word location (0, 1, 2, or 3) within the four-word group. For local fetches, the data word is simply read out to the EWR in the execution unit. For local stores and overwrites, the data from the storage data queue is strobed into the proper word location of the ASM. Note that protected stores are detected on the store to ASM operations. In the event that a protected write is found in the ASM, the store to ASM operation is terminated.

The decode of memory address bits 2 thru 7 selects one of the 64 priority array locations.

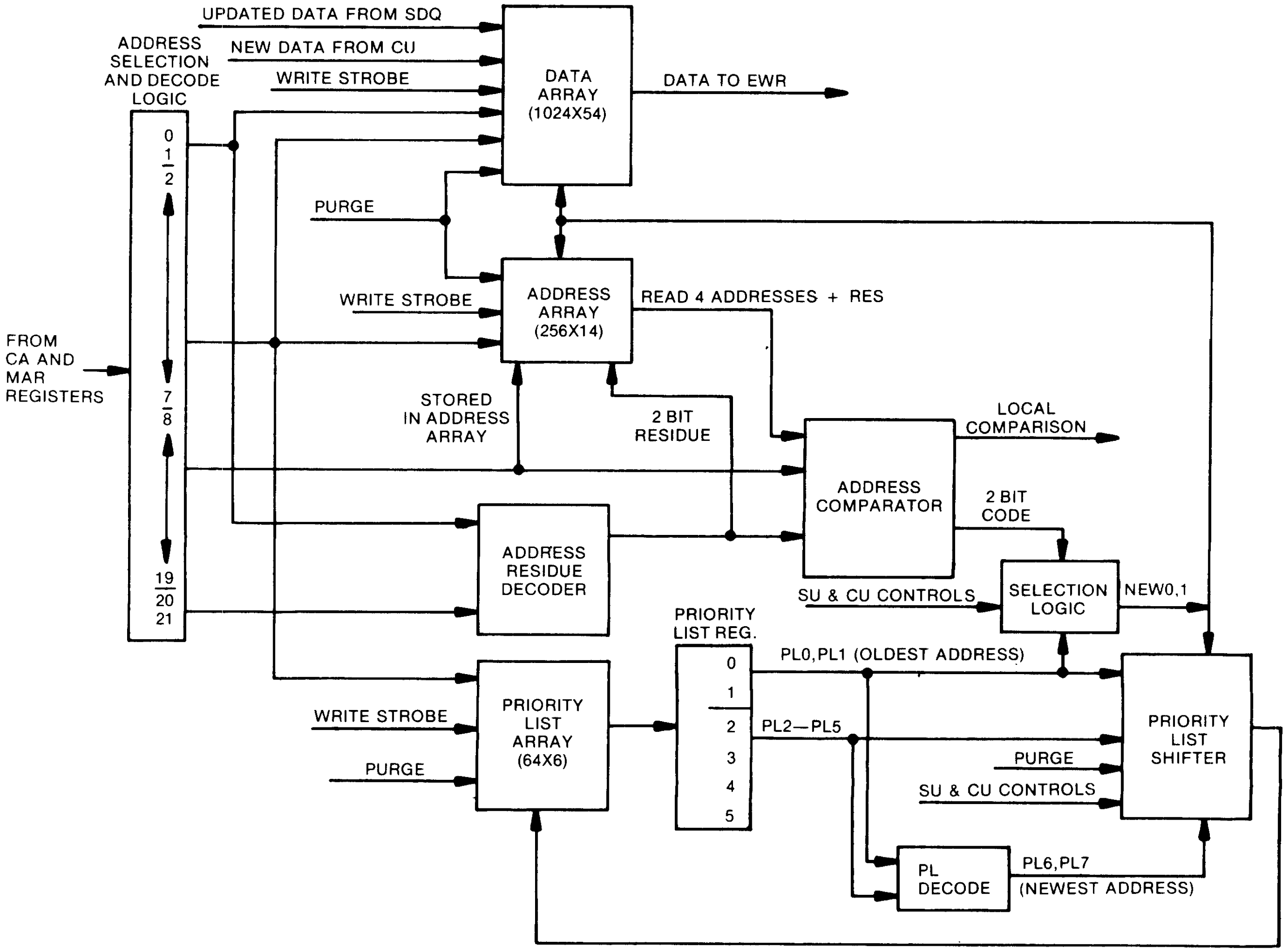


Figure III-1-14. Associative Memory

The priority word in that location is then read out to the priority list register and passed onto the priority list shifter. Here, after the memory address is designated the newest address, the order of the three two-bit priority codes is updated to designate the three oldest addresses with the selected address array block. The new order of address codes is then written back into the addressed location of the array.

ASM FILL OPERATION

The ASM fill operation begins by decoding address bits 2 thru 7 from the CA register. As in the ASM local operations, the decode of these bits selects one of the 64 block locations in the data array, one of the 64 block locations in the address array, and one of the 64 priority list array locations. The priority word in that location is read out to the priority list register. Then on command from the CU, the oldest address location, as defined by bits 0 and 1 in the priority list register, is invalidated by writing zeros into the associated address residue bit locations. As the new address is written in, the data word received by CU is subsequently written into the data array location referenced by the oldest address word and the two LSB's of address in the CA register. The two LSB's of address in the CA register are counted up so that next data word can be placed into proper ASM location. Upon completion of the fill operation, correct residue is generated and written into the associated residue bit locations of the address word.

ASM PURGE OPERATION

The hardware can invalidate all information in the ASM and initialize all locations in the priority list array when necessary such as when entering the MCP for reallocation. To purge the ASM, 0's are written into all address residue bits. To initialize the priority list array, binary codes of 00, 10, and 01 are written into bit locations 0, 1; 2, 3; and 4, 5, respectively, of each word in the array.

STORAGE UNIT

The storage unit (SU) controls the initiation of memory fetches other than program code fetches, stack fills and empties, and of all store operations. The actual operations with main memory are performed by the communications unit, but the storage unit assembles the memory address and control information and for stores supplies the data to the communications unit. As part of its functions, the SU controls the checking of the stack buffer and associative memory local addresses to determine if the required operation is to be performed locally.

The storage unit is capable of performing overlapping operations. Such overlap occurs whenever a main-memory fetch is initiated. While the communications unit is fetching the requested information, the storage unit can go on to the next entry in its operations queue. If

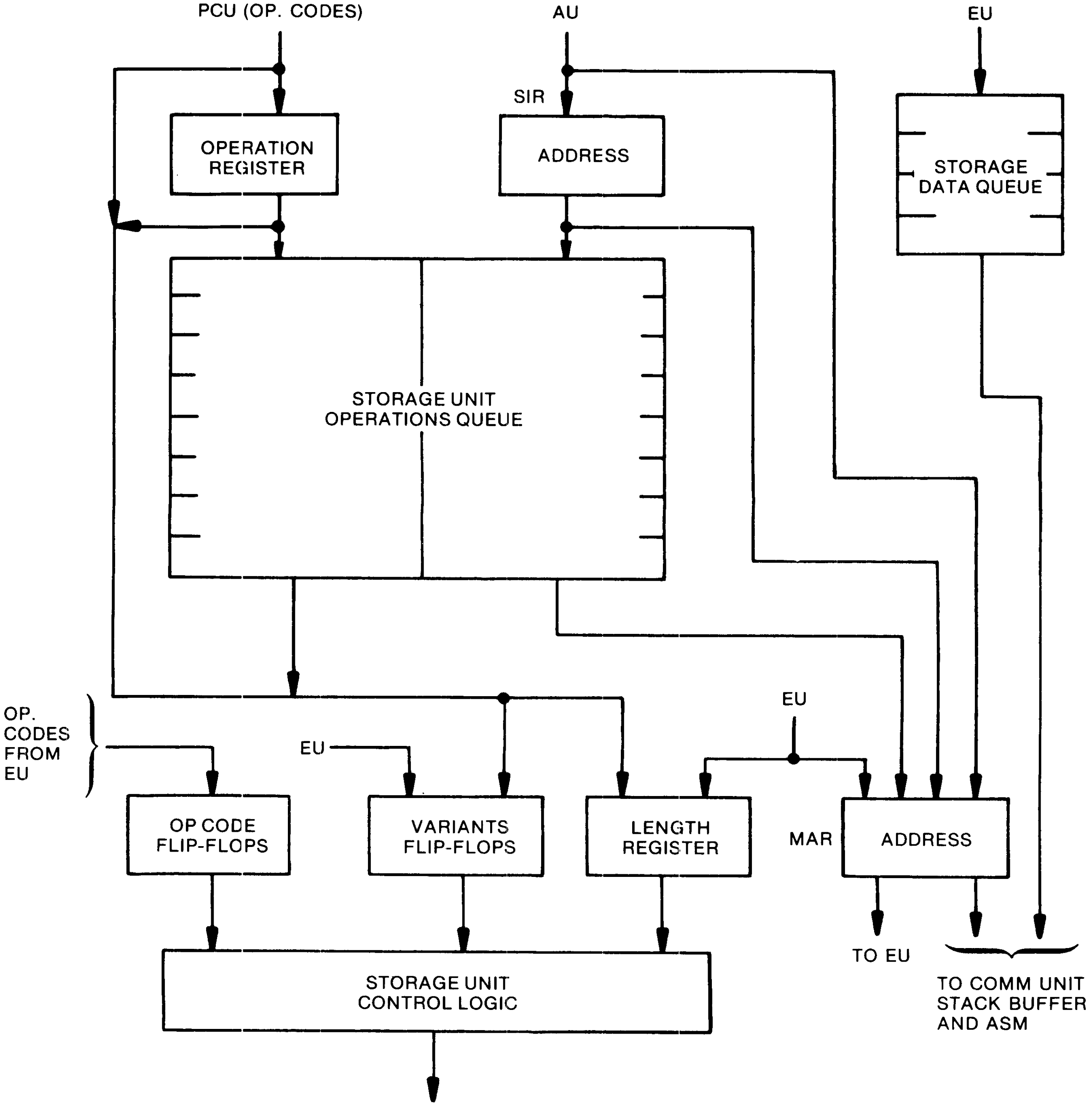


Figure III-1-15. Storage Unit

that operation references an item that is local in the stack buffer or the associative memory, then the local reference is completed in parallel with the main memory reference. The overlap is not restricted to one operation. The storage unit is free to process operations from its queue until another main-memory reference is required.

The block diagram of the storage unit, figure III-1-15, shows the general flow of information through the unit. In preprocessing the operators in the program code string, the PCU determines if a fetch or write-type operation is required for the execution of the operator.

Only fetch-type operations, such as Value Calls or Name Calls concatenated with an index-type or enter operator, are initiated by the PCU. If such an operation is required, the PCU operation codes are passed directly to the SU variant flip-flops and length register. At the same time, the memory address for this operation is calculated by the AU and passed directly to the MA register. For write-type operations (stores, overwrites, and certain string-type operations which execute write operations) the PCU enters a wait bit into the operations queue.

If the store or overwrite is concatenated with a Name Call, the memory address is loaded in SIR and then queued with the wait bit. If the address information is not available, the wait bit is queued without an address, and the address is supplied during execution by the EU. If the operation queue is empty when a PCU job is available and the EU is not requesting the SU, in addition to being written into the queue, the job is passed directly to the variants flip-flops and length register, and the address, if available, is passed to the MA register. The storage unit data queue is used to hold the information for store-type operations. The information to be stored is always supplied by the EU.

INPUT TO STORAGE UNIT OPERATIONS QUEUE

The 6-bit input operation register and storage input register (SI) serve as the write registers for the storage unit operations queue. The PCU loads the 6-bit input operation register, then initiates the write cycle. The types of information which can be loaded into this register include the following:

1. 1WT - Wait bit is set whenever a store or overwrite operation is to be performed.
2. 1MR - Memory reference bit is set whenever a fetch is preceded by a FMMR (force main memory reference) operation.
3. 1QL0 & 1QL1 - Two bits which identify the EU-data queue address for fetch operations.
4. 1LD - Look for double precision bit is set whenever a single word fetch or certain vector mode single word fetches are performed.
5. 1L2 - Length two bit is set whenever a fetch of the MSCW and RCW is required as the result of the PCU preprocessing a return or exit operator. This bit is also set when a two word vector mode fetch is required.

The contents of the SI register include a 20-bit memory address field and two bits of residue on the address. The memory address and residue are loaded by the AU when available.

STORAGE UNIT OPERATIONS QUEUE

The storage unit operations queue is a local storage area containing eight locations. The operations in the SI register are written into successive locations in the queue, and subsequently read from the queue and processed by the SU in the order in which they were queued. The queue address for writing into the queue is provided by the storage operations write pointer (SQW). The pointer is counted up after each write operation. A read pointer (SQR) provides the location in the queue of the next word to be read from the queue.

LOADING THE STORAGE UNIT OP CODE FLIP-FLOPS

The OP code flip-flops are the primary control flip-flops for EU-initiated jobs. The contents of these flip-flops identify the SU operation currently requested by the EU. The operations are defined as follows:

OPERATION	SRW	STY	SPT	SFB	SIL	SML
Fetch	0	0	0	0	0	0
Fetch Memory Fail Register	0	1	0	0	0	0
Overwrite	1	0	0	0	0	0
Single Word Store	1	0	1	1	0	0
Multiple Word Store	1	1	1	0	0	0
Read with Lock*	1	0	0	1	0	0
Requestor Inhibits Load*	1	0	0	0	1	0
Memory Limits Load*	1	0	0	0	0	1

*This operation is performed as an overwrite-type operation by the SU.

LOADING THE STORAGE UNIT VARIANTS FLIP-FLOPS

There are seven variants flip-flops which contain additional control information for the operation currently being performed by the SU. These flip-flops are loaded from either the queue, input operation flip-flops, PCU, or the EU. The type of information loaded into the variants flip-flops includes the following:

1. SBY – Bypass operations queue. SBY flip-flop is set by all EU-initiated jobs, except those cases involving certain store and overwrite operations. This flip-flop is only set by stores arising from tracing through a chain of IRW's or linked descriptors. This flip-flop is not set when an overwrite is issued to the SU as a result of the EU processing the step and branch operator.
2. SEW – Request use of the EWR in the EU. SEW flip-flop is set by all EU-initiated jobs, except overwrite operations not within a string operator.
3. SQL0 and SQL1 – Two bits which identify the EU-data queue address for PCU fetch operations.
4. S4A – 4 word ASM. S4A flip-flop is set by all operations, except Masked Search For Equal, Linked List Lookup, fetch of MSCW, RCW for Exit and Return, and operators preceded by FMMR.
5. SMR – Memory reference. SMR flip-flop is set whenever the SU receives a PCU or EU-initiated job as the result of executing a program operator preceded by FMMR.
6. SLD – Look for double precision. SLD flip-flop is set whenever EU-initiated IRW fetches (excluding those issued by the enter or evaluate operator) and load transparent fetches are performed. This flip-flop is also set whenever PCU-initiated single word fetches and certain vector mode single word fetches are performed.

LOADING THE STORAGE UNIT LENGTH REGISTER

The length register is a 3-bit register which contains the number of words to be stored or fetched for PCU or EU-initiated operations. All operations have a length of 1 or 2, except certain jobs issued by string and edit operators. String and edit fetches can have a length of 4, string stores have a length of 4, and string overwrites have a length of 4, except for the last overwrite which can have any length (up to 4 words).

STORAGE UNIT DATA QUEUE

The storage unit data queue contains four 52-bit word locations. A separate write pointer (SDW) and read pointer (SDR) are used for addressing in the SU data queue. Writing into the queue is performed only by the EU. Information is read from the queue under control of the SU. The information taken from the queue is usually passed to the communications unit for storage in main memory, but is also sent to the associative memory and stack buffer when the store is to an address which is local.

Because the EU must determine that the store was completed successfully before additional operators can be executed, the SU data queue contains the data for only one store operation at a time. The queue is four words long so that it may hold the data for any multi-word store operation.

MEMORY ADDRESS REGISTER

The MA register, which contains the memory address for the current operation, is loaded from the queue, from the EU, from the AU, or directly from the SI register. The address in the MA register is sent to the stack buffer and associative memory to allow these units to determine if the address is local, and is forwarded to the communications unit for use in main memory operations.

For multi-word operations, the starting address is forwarded to the CU, then the address in the MAR is counted up so that a check for local addresses may be performed on each address involved in the multi-word operation.

STORAGE UNIT CONTROL LOGIC

The timing and controls for SU operations are provided by this portion of the SU. The operations include special fetches, read with lock functions, single or multi-word fetches, protected stores, or overwrites. At the start of any operation, the control logic obtains access to the ASM, and, if necessary the SB and the EWR. Then the CU is started if an EU-intialed fetch is to be performed, and a check is made to determine if the address is local in the SB or ASM. The CU is not required during overwrite and stores if the address is local in the SB, the stack does not contain memory copies, and the MA is not equal to KL (oldest entry in stack buffer). If MA equals KL, the information in location referenced by KL is stored to main memory because it may be the least-significant half of a double precision word. If, during a local store operation, a protected write is discovered in that location, the CU and SU operations are terminated. Also, the CU operation is terminated if a protected write is found in main memory.

For single-word fetch operations, if the information is locally available in the SB or ASM, the information is transferred to the EWR in the EU and the CU operation is terminated. However, the CU performs a residue check on the address even though the address is local. If both SB and ASM contain the information, only SB information is transferred to the EWR. A check is performed to determine if information in EWR is a double precision operand (tag 2) or is a nonpresent descriptor (bit 47 set). If a tag 2 is found, a local check of next address is performed for second half of the double precision operand. If non-present descriptor is found, the memory address from which the descriptor was fetched is sent to the EU-data queue.

For multi-word fetch operations, if all of the information is available only in the stack buffer (above KL) or ASM, the CU operation is terminated. If some of the information is available only in the stack buffer or ASM, the portion in main memory is fetched, then the portion in the stack buffer or ASM, if not copied in stack buffer, is fetched.

For store and overwrite operations queued by the PCU, a wait bit is set in the operation code. When the operation is performed, this wait bit causes the control logic to hold the operation until the required data has been entered into the SU data queue. If the address is not provided from the queue, the SU also waits until the MA is loaded by the EU or the AU. When the information is available, the control logic enables the read lines of the SU data queue and disables the write lines of the queue until the operation is complete. For single word and multi-word store operations, if address is local, the SU changes the store to an overwrite operation and sends the information found in local storage to the EWR. There a check is made to determine if information being replaced in local storage has protect bit on (bit 48) before the local store is completed. If the protect bit is on, the write into local storage is not performed, and the CU operation is terminated. If a store is preceded by a FMMR instruction, the local storage words are not examined for protect bit because the words in the equivalent memory addresses are more current than those in local storage. Thus, it becomes necessary to use memory flashback to determine if any of the memory words are protected. If the memory addresses do not contain protected words, the SU changes the store to an overwrite operation and then proceeds to complete the local storage operation. If a protected word is sensed, the memory returns the failsoft command, which, in turn terminates the local store and CU operations.

On completion of any store or overwrite operation, the control logic resets the SU data queue pointers to zero and enables the write lines when CU has completed its portion of the operation.

COMMUNICATIONS UNIT

The communications unit (CU) provides the interface between the CPM and main memory. All main memory accesses are performed by this unit. Requests for memory operations are made to the CU by the program buffer, the storage unit, and the stack buffer. Information fetched by the CU from memory is forwarded to the execution unit, the stack buffer, the associative memory, or, for program code, to the program buffer.

Access to the CU is granted to the requesting CPM units on a priority basis. First priority is given to the stack buffer, because the EU is waiting for the results of any request made by the stack buffer. The stack buffer requests are made when performing a stack-buffer fill, empty, or purge operation. The storage unit has second priority as the EU may be waiting for the results of an SU request. The program buffer requests have third priority as these requests are made in anticipation of the actual need for additional program code.

The major logic elements of the CU, as shown in figure III-1-16, include input (IN) and output (OP) registers, the communications address (CA) register, the communications length (CLN) register, the cancel length register, the start address flip-flops the remember-suspend (RS) register, the fail (FR) register, and the control logic.

For all memory operations, the absolute memory address is contained in the CA register and the number of words to be fetched or stored is in the CLN register. During the operation, both the address and the word count are adjusted for each word fetched or stored. In addition to holding the absolute memory address and word count of an operation, the CU is also supplied with a start address and cancel length count whenever the CU is servicing a storage unit fetch operation. This combined information informs the CU on what portion of a multi-word fetch from memory is sent to the EU. This condition arises as the result of the storage unit locating one or more words in local storage and the remaining words in main memory. Thus only those non-local words remaining in memory are sent to the EU even though the entire fetch from memory is loaded into the ASM. The other portion of the fetch was previously supplied to the EU during the local fetch operation. In most cases, the storage unit fetch is accompanied by an ASM load operation and thus a four-word fetch is required from memory.

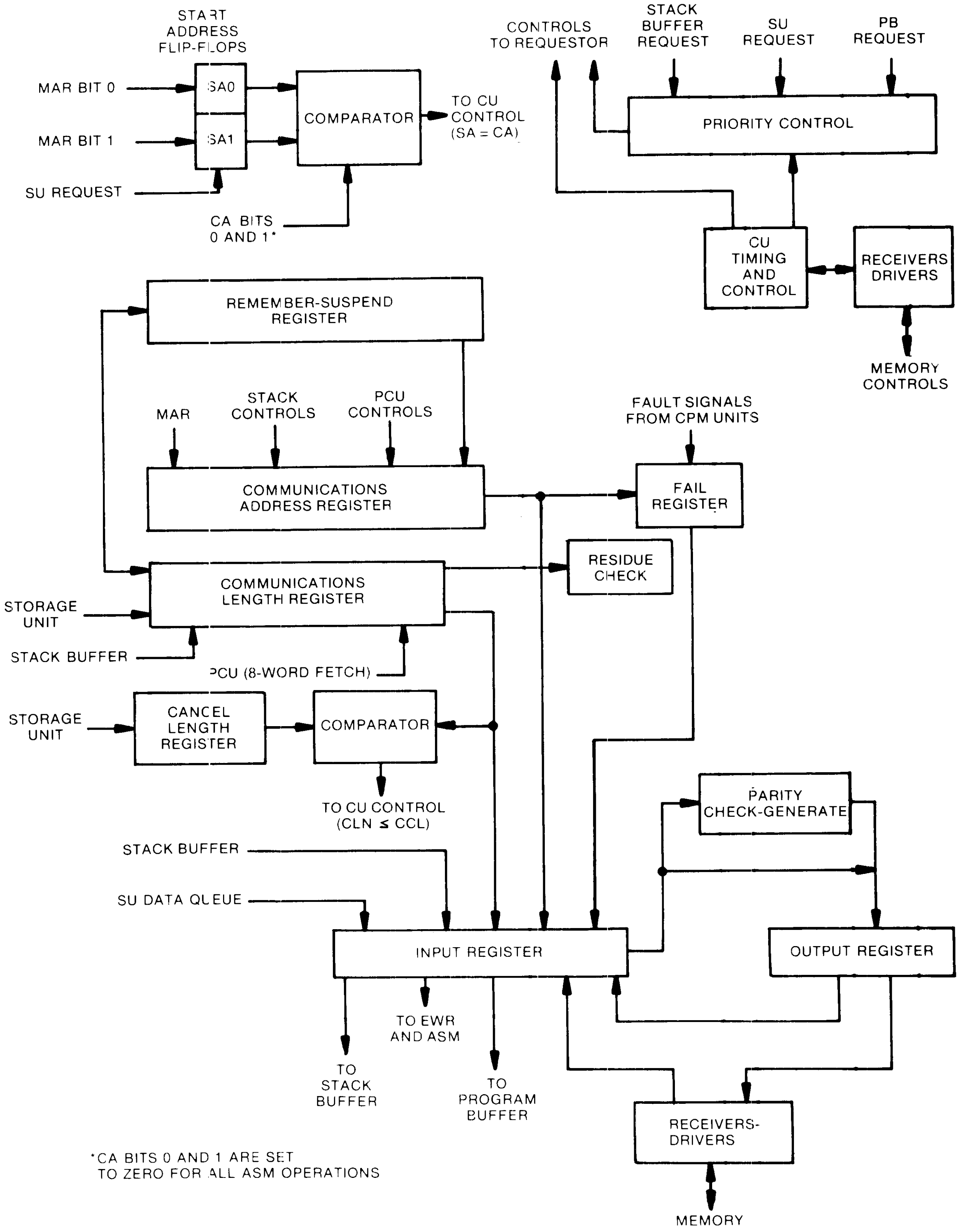


Figure III-1-16. Communications Unit

Program code is fetched in eight-word blocks, which requires two four-word fetches. If, at the end of the first four-word fetch of program code, assuming an MSU configuration, a higher priority request has been made for CU use, the current memory address and word length are transferred to the remember-suspend register for temporary storage. Then the second four-word fetch is delayed until after the higher-priority request has been serviced. When no other requests are pending, the RS register contents are loaded back into the CA and CLN registers and the fetching of code is resumed.

When access to main memory is required, the CU control logic compares the six most-significant bits of the address in the CA register with the limits established for each MCM, and selects the appropriate module. Then, the starting address and other control information for the operation are sent to the selected module in a memory control word. The control word is assembled in the input register, then transferred to the output register and is sent to the addressed memory control module. The receipt of the control word is acknowledged by the MCM.

For fetch operations, the MCM notifies the CU that access has been granted by sending an ACK signal. A Data Present (DAP) signal along with the data is then sent to the CU. The data is received by the IN register and is subsequently forwarded to the program buffer, the stack buffer, the associative memory, or the EWR, as appropriate.

If a memory related error is detected by the CU during a fetch or flashback operation, the word, which is involved at the time of the failure is replaced by an error word. This error word, which is a copy of the control word, is contained in the OP register, then transferred with other related information into IN, and IN sends the error word to the location that is to receive the expected memory word. (See figure III-1-17 for format of the error word.)

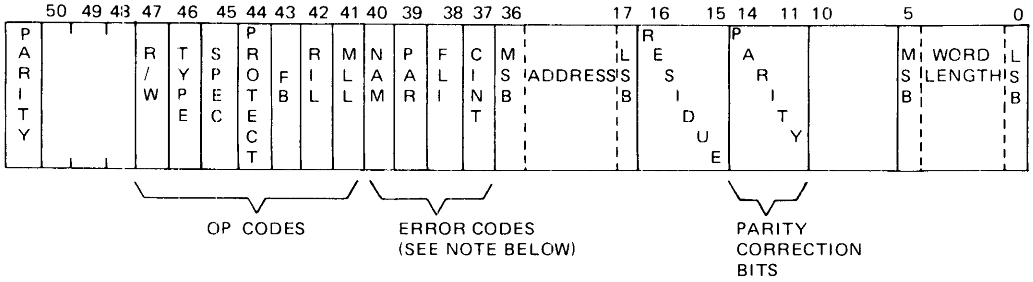
Data for store operations is received by the CU from either the SU data queue or the stack buffer. Data for store operations is buffered in the IN register until the CU gains memory access. Following the transfer of the control word and the acknowledgement of the receipt of this word, the selected MCM will inform the CU of access by sending a send-data signal to the CU. On obtaining access, the CU transfers the data into the output register and the word is then sent to the selected MCM.

The fail register (FR) is used to provide information concerning processor internal and memory related error conditions. When an error condition is detected, a bit assigned to designate that condition is set in the FR register. Detailed discussion of the FR register is provided in Section 2 Interrupts of this manual.

FAULT CONTROL LOGIC

The fault control logic is mechanized in such a way as to aid in general maintenance and error recovery under the guidance of the MCP. Error recovery is aided by a system of multiple levels of control mode coupled with alternate stack and display zero capabilities. The fault condition register records system interrupts and conditions the processor to take the necessary action in order to handle these interrupts (see figure III-1-18). This register records both operator dependent and operator independent interrupts. The program index and program syllable counts on operator dependent interrupts are adjusted to allow re-execution of the interrupted operator. The operator independent interrupts are processed at the conclusion of the operator being executed at the time of the interrupt.

The interrupt system allows for special "complete" communication and for recording single-bit errors and two-bit errors from main memory. It also allows for the sensing and recording of special errors internal to the main memory and Central Processor.



A. OP CODES

47	46	45	44	43	42	41	OPERATION
0	0	1	0	0	0	0	SINGLE DATA WORD FETCH
0	0	0	0	0	0	0	N–LENGTH DATA WORD FETCH
0	1	1	0	0	0	0	FAIL REGISTER FETCH
1	0	1	0	1	0	0	SINGLE–WORD OVERWRITE WITH FLASHBACK
1	0	1	1	0	0	0	SINGLE–WORD PROTECTED WRITE
1	0	1	1	1	0	0	SINGLE–WORD PROTECTED WRITE WITH FLASHBACK
1	0	0	0	0	0	0	N–LENGTH OVERWRITE
1	0	1	0	0	0	0	SINGLE–WORD OVERWRITE
1	1	0	1	0	0	0	N–WORD PROTECTED WRITE
1	0	1	0	0	1	0	LOAD REQUESTOR INHIBITS
1	0	1	0	0	0	1	LOAD MEMORY LIMITS

B. ERROR CODES

40	39	38	37	
1	0	0	0	NO-ACCESS-TO-MEMORY OR INVALID CHANNEL
0	1	0	0	WRONG PARITY ON INCOMING MEMORY TRANSFERS
0	0	1	0	FAIL 1 INTERRUPT
0	0	0	1	COMM INTERNAL (RESIDUE ERROR IN CA)

Figure III-1-17. Error Word Format

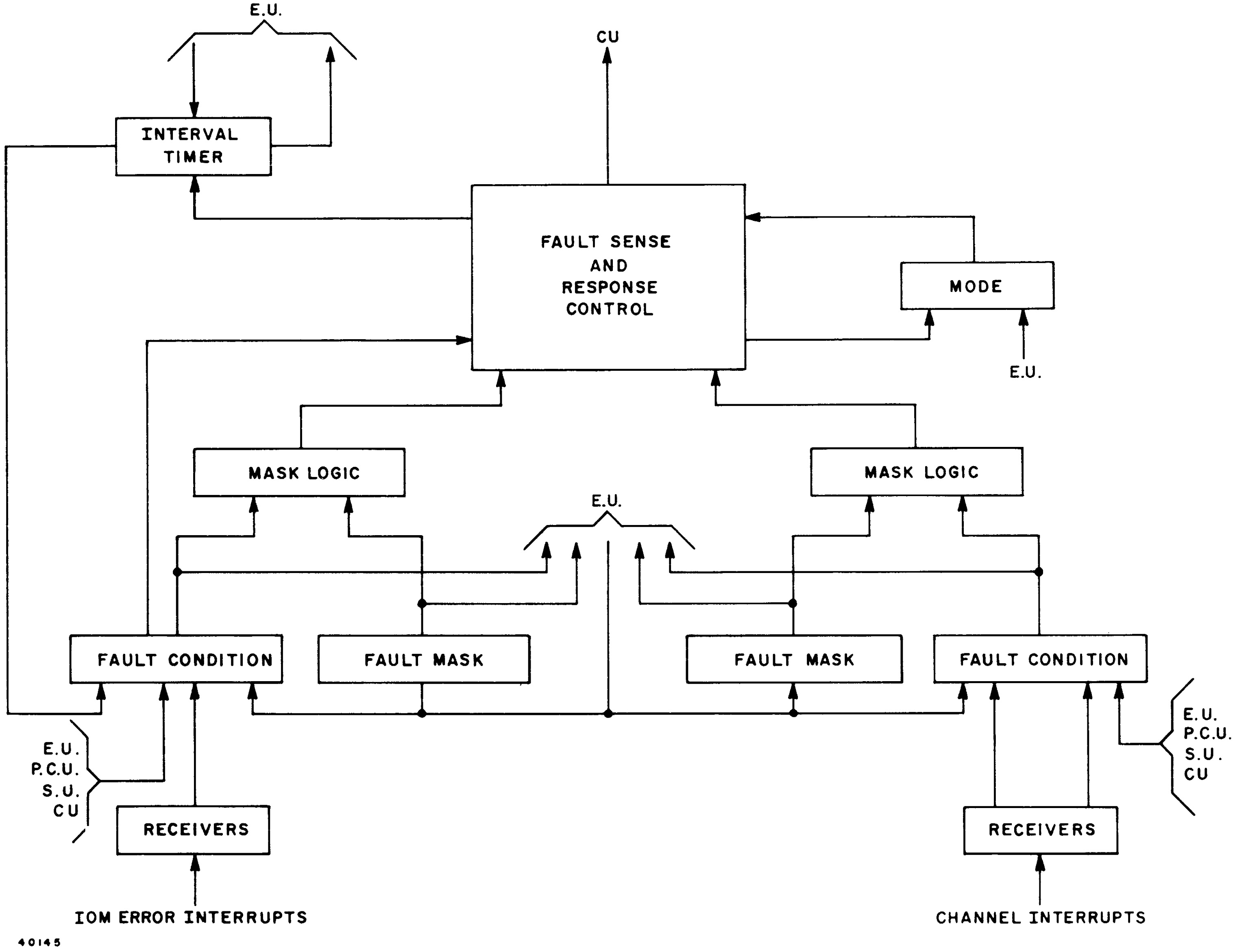


Figure III-1-18. Fault Control Logic

SECTION 2

INTERRUPTS

INTRODUCTION

An interrupt is a means of diverting a processor temporarily from the job which it is doing if certain predetermined conditions occur, so that some higher priority job may be done immediately. Interrupts are processed by the interrupt handling mechanism of the MCP. When the interrupt has been processed, the MCP will (if conditions permit) reactivate the interrupted process.

The interrupt handling mechanism of the MCP deals with two classes of interrupts: hardware interrupts and software interrupts. Hardware interrupts are generated automatically by the B 7700 system (when interrupt conditions occur) and are processed by the MCP interrupt procedure. Software interrupts are programmatically defined, and are used both by the MCP and by object programs for communication between processes. This discussion deals only with hardware interrupts.

HARDWARE INTERRUPT SYSTEM

The B 7700 hardware interrupt system is a primary interface between the MCP and the hardware. Interrupt conditions may be detected by the Central Processor Module (CPM), the Input/Output Module (IOM); or the Memory Control Module (MCM). When detected, interrupt conditions are processed by the Fault Control Logic of the CPM. Normally, the CPM prepares the stack for procedure entry, places the necessary parameters in the stack, and causes an entry into the MCP interrupt procedure.

CPM STATES AND MODES

The CPM operates in either of two states: control state, used only by the MCP; or normal state, used both by user programs and by the MCP. Normal state is always used when executing user programs. Control state is used when executing certain portions of the MCP, including the MCP interrupt handling procedure. In the control state, external interrupts and interval timer interrupts are inhibited (except during an IDLE or PAUS instruction) and the CPM may execute privileged instructions which it may not execute in normal state.

In addition to the two states, the CPM can be in any one of five interrupt handling modes: Normal Mode (Control Mode 0), Control Mode 1 (CM1), Control Mode 2 (CM2), Control Mode 3 (CM3), and Control Mode 4. The CPM operates in normal mode until an interrupt condition is detected. The first three control modes allow for recursive attempts to enter MCP interrupt handling procedures by the fault control logic of the CPM. The CPM halts in Control Mode 4 if these attempts are not successful. The CPM will return to Normal Mode if an interrupt condition is handled successfully in CM1, CM2, or CM3.

There is no direct connection between the states of operation and the modes of operation of the CPM. The CPM may be in control state or in normal state while in any control mode. In a system which contains more than one CPM, any or all of the CPM s may operate in control state or normal state, as well as in any of the interrupt modes. The CPM states are described below; the interrupt modes are further described in the discussion of interrupt processing.

CONTROL STATE

Entry into control state (from normal state) occurs when the MCP enters or returns to a control state procedure (an MCP SAVE procedure), or when the CPM executes a Disable External Interrupts operator (DEXI). (Control state procedures have bit 19, the N bit, of the PCW set.) While the CPM is operating in control state the reporting of external interrupts to the MCP interrupt handling routine is disabled. Additionally, the CPM may execute certain privileged operators while in control state which may not be executed in normal state. When the CPM is operating in control state, the normal control state flip-flop (PST) and the inhibit interrupt flip-flop (IIH) are both set (except during an IDLE instruction).

The interrupts which are inhibited while in control state include the Channel interrupts, the IOM Error interrupts, the Interval Timer interrupt, and the Memory Fail 2 interrupts. Although the processing of these interrupts is inhibited, the appropriate bit in the CPM Interrupt (Fault) register will be set if one of these interrupts is detected, and the interrupt will be processed when the CPM enables External interrupts either by returning to normal state or by executing an IDLE or PAUS operator.

The Egg Timer interrupt, although an external interrupt in priority, is not inhibited in control state.

The operators which are enabled only when the CPM is in control state include Set Interval Timer (SINT), Inhibit Parity (IGPR), Set Memory Inhibits (SINH), and Set Memory Limits (SMLT).

NORMAL STATE

Return to normal state (from control state) occurs whenever the MCP initiates or returns to a normal state procedure (non-SAVE procedure), or when the CPM executes an Enable External Interrupt operator (EEXI). (normal state procedures have bit 19, the N bit, of the PCW reset). When the CPM is operating in normal state, the processor state flip-flop (PST) and the inhibit interrupt flip-flop (IIH) are both reset. When a CPM returns to normal state after servicing an interrupt, it does not necessarily return to the program which was executing when the interrupt was detected. The selection of the job to be run is a function of the MCP.

FAULT CONTROL LOGIC

The fault control logic of the CPM contains four registers which are used to record and process hardware interrupts: the Fault Condition, Fault Mask, Fail, and Control Mode registers. The Fault Condition register is used to indicate the detection of one or more interrupt conditions (one bit for each condition). The Fault Mask register is used to inhibit (mask out) the processing of one or more interrupt conditions. (The Fault Condition register may be read in such a way as to obtain only interrupt conditions which are not masked out; thus indicating an interrupt condition which must be processed by the MCP.) The Fail register further identifies errors which are internal to the CPM and CPM-MCM interface errors. The Control Mode register is used to identify the interrupt mode (Normal, Control Mode 1, Control Mode 2, and Control Mode 3) in which the CPM is operating.

In addition to the CPM registers, a Memory Fail Register in each Memory Control Module (MCM) is used to give detailed information about memory-related failures concerning that MCM. (Discussion of the MCM fail register is given in Chapter V of this manual.) For IOM error interrupts, detailed information about the IOM failure is given in an IOM Fail Word.

FAULT CONDITION REGISTER

The Fault Condition register contains one bit for each of the possible interrupt conditions (see table III-2-1). The low order bits of the register are associated with interrupts which have the highest priority for being processed by the CPM; the high order bits are as-

Table III-2-1. B 7700 Interrupt Bit Assignments

Interrupt	Fault Condition Register (Bit)	Fault Mask Register (Bit)	Interrupt Ident. (P1) (Bit)	
Alarm (First Priority)				
Loop	0	NONE	0	
Memory Parity	1		1	
Memory Fail 1	2		2	
Invalid Address (No Access)	3		3	Plus Bit 25
Stack Underflow	4		4	
Invalid Program Word	5		5	
Processor Internal	6		6	
Syllable (Second Priority)				
Memory Protect	9		0	
Invalid Operand	10		1	
Divide By Zero	11		2	
Exponent Overflow	12	N0NE	3	
Exponent Underflow	13		4	Plus Bit 24 or Bit 23 (See Note 1.)
Invalid Index	14		5	
Integer Overflow	15		6	
Bottom Of Stack	16		7	
Presence Bit	17		8	
Sequence Error	18		9	
Segmented Array	19		10	
Programmed Operator	20		None	
Privileged Instruction	21		11	

Table III-2-1. B 7700 Interrupt Bit Assignments (Cont)

Interrupt	Fault Condition Register (Bit)	Fault Mask Register (Bit)	Interrupt Ident. (P1) (Bit)
Special			
Stack Overflow (Third Priority)	24	24	1
Interval Timer (Fifth Priority)	23	23	0 Plus Bit 22
External (Fourth Priority)			
Channel 0	26	26	0
Channel 1	27	27	1
Channel 2	28	28	2
Channel 3	29	29	3
Channel 4	30	30	4
Channel 5	31	31	5
Channel 6	32	32	6
Channel 7	33	33	7 Plus Bit 21
IOM Error 0	34	34	8
IOM Error 1	35	35	9
IOM Error 2	36	36	10
IOM Error 3	37	37	11
IOM Error 4	38	38	12
IOM Error 5	39	39	13
IOM Error 6	40	40	14
IOM Error 7	41	41	15
Memory Fail 2	42	42	16
Egg Timer	None	None	None

NOTE 1:

On syllable interrupts ID Bit 24 indicates class 1 interrupt (PIR, PSR, PDR, PBR have not been modified, ID Bit 23 indicates class 2 (PIR, PSR, PDR, and PBR are undefined).

sociated with interrupts which have the lowest priority. When interrupt conditions are detected, the bits associated with those conditions are set in the Fault Condition register.

Normally, the Fault Condition register is set by the interrupt condition. As each interrupt condition is processed, the bits in the register are selectively reset. Programmatic control of the Fault Condition register is accomplished by use of the Set Processor Register (SPRR) and Read Processor Register (RRRR) operators. The RPRR operator causes the contents of the register to be placed in the stack, and the register itself to be reset. The SPRR operator causes an inclusive OR setting of the register; that is, bits are set, but bits which already are set are not reset.

FAULT MASK REGISTER

The Fault Mask register allows the processing of certain interrupts to be inhibited or deferred. Alarm interrupts, Syllable interrupts, and the Egg Timer interrupt may not be masked. The Special interrupts and the other External interrupts have a corresponding bit in the Fault Condition and Fault Mask registers (see table III-2-1). An interrupt condition will only be recognized by the CPM if the Fault Mask register bit for that condition is set (logical one). If the Fault Mask bit is reset for an interrupt condition, that interrupt bit will still be recorded in the Fault Condition register but will go unnoticed by the fault control logic. If the mask configuration is later changed, then interrupts (including those resident in the Fault Condition register when the mask is changed) which are now unmasked will be recognized and processed. In this way, processing of selected interrupts can be deferred.

The Fault Mask register may only be set programmatically. The Read Processor Register operator causes a simple read of the register (without reset); the Set Processor Register operator causes a simple set of the register (each bit is set either to logical one or to logical zero).

INTERRUPT IDENTIFICATION

Each interrupt condition reported to the MCP is identified by a unique literal value, known as interrupt parameter P1 (see table III-2-1). this parameter is passed to the MCP interrupt procedure by the fault control logic to identify the condition which is to be processed. The P1 parameter is derived from the contents of the Fault Condition and Fault Mask registers through a series of gates. Interrupt conditions reported in the Fault Condition register which are not masked out by the Fault Mask register are used to make up the P1 parameter.

Normally, this parameter is read and placed into the stack by the fault control logic, although it may be read into the stack programmatically. In either case, the resultant action is as follows. The value of P1 is read into the stack and the bits which were set in P1 are reset in the Fault Condition register. In a particular P1 parameter, all interrupts of a particular priority level which are not masked out are reported, but only one priority level is reported on each read. The priority level reported will be the highest priority level for which there is at least one bit set in the Fault Condition register which is not masked out. If the value of the P1 parameter is read programmatically (using the Read Processor Register operator), and if there are no unmasked interrupts to report, a word of all zeros is read into the stack. (The fault control logic will read P1 only when there is an unmasked interrupt to report).

CPM FAIL REGISTER

The CPM fail register (see figure III-2-1) provides specific information about processor internal interrupts. The type of processor internal interrupts is identified by one of seven bits in the CPM fail register. These bits are EU Continuity Error (EUC), EU Residue Error (EUR), EU Parity Error (EUP), PCU Error (PER), Adder Residue Error (ADD), Input Register Parity (INP) and CU Residue Error (CRS). Usually only one of these bits is set for a given internal error. However, it is conceivable to have a CRS error occur during a memory fetch operation and then to have it replaced in the CPM fail register by another failure in a subsequent processor operation. It can be determined that a CRS error occurred during a fetch and was the cause of the processor interrupt by observing that P2 interrupt parameter contains a CU error word in which bit 37 is on. (Refer to figure III-1-17 for error word format.)

The CPM fail register contains several bits which are useful in analyzing CU detected errors. Whenever the CU detects an error, the CU loads the Memory Control Module number, the most significant 6 bits of memory address, and the most basic type of operation information into the CPM fail register. The no-access-to-memory errors are detected by the CU, and include Wrong Channel Number (WCN) and Memory Time Out (MTO). The Single-bit error bit (F2) is set if one bit core error occurs during any memory operation. When the CPM fail register is read by using the Read Processor Register operator, the CPM fail register is cleared.

When an alarm interrupt occurs during a store or overwrite type operation in the Communications Unit, the CPM interrupts immediately. Thus, the information in the CU and memory portions of the CPM fail register is always current provided OP=1. However, the CU may detect more than one error on fetch operations before the CPM fail register is read because the CU errors do not necessarily result in interrupts until an attempt is made to process the memory word involved at the time of

26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9 – 4	3 – 0
INR	MES	EUC	EUR	EUP	PER	▨	ADD	WCN	INP	CRS	CSE	MTO	F2	SU	SK	OP	MADS	MCM NO.

	Field	Bit	Error
MEMORY	BN	3:4	Box Number (MCM No.)
	MADS	9:6	Memory Address (most significant 6 bits)
	OP	10	Operation (1=Store/0=Fetch)
	SK	11	1=Stack Operation/0=Not Stack Oper.
	SU	12	1=SU Oper./0=Not SU Oper.
	F2	13	Single-Bit Error
COMM. UNIT	MTO	14	Memory Time Out
	CSE	15	Comm Single Error
	CRS	16	CU Residue Error
	INP	17	Input Register Parity on data from CPM
	WCN	18	Wrong Channel No
	ADD	19	Adder Residue Error
	PER	21	PCU Error
EXECUTION UNIT	EUP	22	EU Parity Error
	EUR	23	EU Residue Error
	EUC	24	EU Continuity Error
	MES	25	Probable Inconsistent State
	INR	26	Interrupt probably not caused by operator indicated by RCW

Figure III-2-1. CPM Fail Register

failure. In some cases, the memory word is never used because a conditional branch is taken or because it is an unused portion of a two- or four-word fetch.

When information regarding a CU-detected error is loaded with an empty CPM fail register, the Communications Single Error (CSE) bit is set. If a subsequent CU error occurs before the fail register is cleared, the CU will overwrite the previous information in the fail register and reset CSE. Thus, CSE reset in the fail register indicates that the CU and memory information may not correspond to

The INR and MES bits in the CPM fail register indicate whether all information necessary to retry an operation has been preserved by the EU when an alarm interrupt occurs. If an alarm interrupt occurs, such as Processor Internal, and the MCP finds both INR and MES bits reset, instruction retry will be attempted.

CONTROL MODE REGISTER

The Control Mode register indicates the interrupt mode in which the CPM is operating. The use of interrupt modes provides for recursive entries into the fault control logic. The progression to higher interrupt modes is controlled automatically by the hardware. In addition, programmatic control of the Control Mode register may be accomplished by use of the Read Processor Register and Set Processor Register operators.

The Control Mode register contains three bits which display the interrupt modes of the CPM: the CM1 bit, the CM2 bit, and the CM3 bit. In Normal Mode, none of these bits are set. In Control Mode 1, only the CM1 bit is set. In Control Mode 2, only the CM2 bit is set. In Control Mode 3, the CM1 and CM2 bits are both set. In Control Mode 4, only the CM3 bit is set. The CPM will be halted with the last interrupt displayed in the Fault Condition register and the HSI flip-flop set if an interrupt is detected in CM3.

The CPM operates in Normal Mode while not attempting to process an interrupt. When an interrupt condition is detected, the CPM advances to CM1 and attempts to call the procedure pointed to by D [0] +3 (the MCP interrupt procedure) from the stack of the user program. If an interrupt is detected while in CM1, the CPM advances to CM2, changes the stack environment by moving to an alternate stack (determined by indexing the stack vector by the CPM number), and attempts to call the MCP interrupt procedure again. If an interrupt condition is detected in CM2, the CPM advances to CM3, changes the entire environment by setting D [0] to the value in the ADZ register, moves to the proper alternate stack in the new environment, and attempts to enter the interrupt handler at the new D [0] +3.

If still another interrupt is detected while in CM3, it is obvious that a recursive interrupt processing situation exists, and the CPM advances to CM4 and halts. If the CPM succeeds in entering the MCP interrupt procedure, the Control Mode register is reset to Normal Mode programmatically.

IOM FAIL WORD

The IOM Fail word (figure III-2-2) is a 48-bit word which contains information regarding errors which cannot be associated with a particular channel or device. (Such errors cause an IOM Error interrupt). When an IOM Error interrupt occurs, an IOM Fail Word is built by the fail mode logic within the IOM translator and placed in the result descriptor word of the "Fail IOCB". The Fail IOCB is associated with Unit Designate Number 0. The Fail IOCB is delinked from the queue of Fail IOCB's and linked into the queue of completed IOCB's (defined by the Status Queue Header) in the same manner as a normal I/O termination.

INTERRUPT PROCESSING

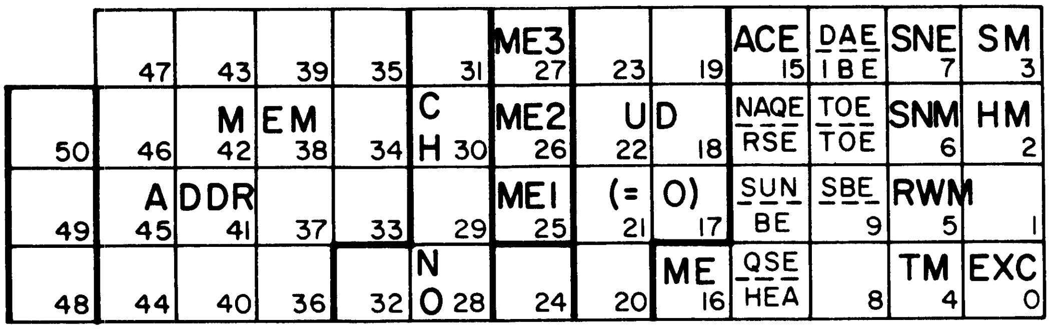
All interrupt conditions which are reported in the Fault Condition Register and which are not masked out by the Fault Mask register are accumulated into a general signal to alert the fault control logic of the CPM to the fact that one or more interrupts require processing. When an interrupt requires processing the CPM will advance the Control Mode register (in most cases from Normal Mode to CM1) and will attempt to enter the MCP interrupt procedure.

INTERRUPT PROCESSING IN NORMAL MODE

After advancing the Control Mode register from Normal Mode to CM1, the CPM will attempt to perform the following sequence of operations:

a. Read and save the P1 parameter.

b. Place a Mark Stack Control Word (MSCW) into the stack.



Field	Bits	Error
EXC	0	Exception Bit
MODE WHEN ERROR OCCURRED		(See bits 2 to 6)
HM	2	Home Address Mode
SM	3	Start Mode
TM	4	Terminate Mode
RWM	5	Ring Walk Mode
SNM	6	Scan Mode
SNE	7	(See bits 9 to 14 or bits 10 to 14)
SCAN ERRORS		(If SNE bit 7=1)
SBE	9	Scan Bus Parity Error
TOE	10	Time Out Error (Scan Bus)
DAE	11	Disk Address Error
QSE	12	DFO Stack Parity Error
SUN	13	Storage Unit Not Available
NAQE	14	No Access to DFO Exchange
NON-SCAN ERRORS		(If SNE bit 7=0)
TOE	10	Time Out Error (Data Service)
IBE	11	Initiate Busy Channel
HAE	12	Home Address Illegal Command
BE	13	Buffer Register Parity Error
RSE	14	Residue Error (Memory Address)
ACE	15	Active Channel Stack Parity Error
CH NO	21:5	Channel No. (If ACE bit 15=1)
MEM ADDR	47:20	Memory Address of 10CB (If ACE bit 15=0)
ME	16	Memory Error (see bits 25 to 27)
ME1, ME2, ME3	27:3	Memory Error Code (If ME bit 16=1)
UD	24:8	Unit Designate (=0)

Figure III-2-2. IOM Fail Word

c. Place an Indirect Reference Word (IRW) into the stack. The IRW references a reserved location (D [0] + 3) in the MCP stack. (When in Control Mode 3, the IRW references a reserved location (D [0] + 3) in the Alternate D [0] stack.)

d. Place the P1 parameter into the stack.

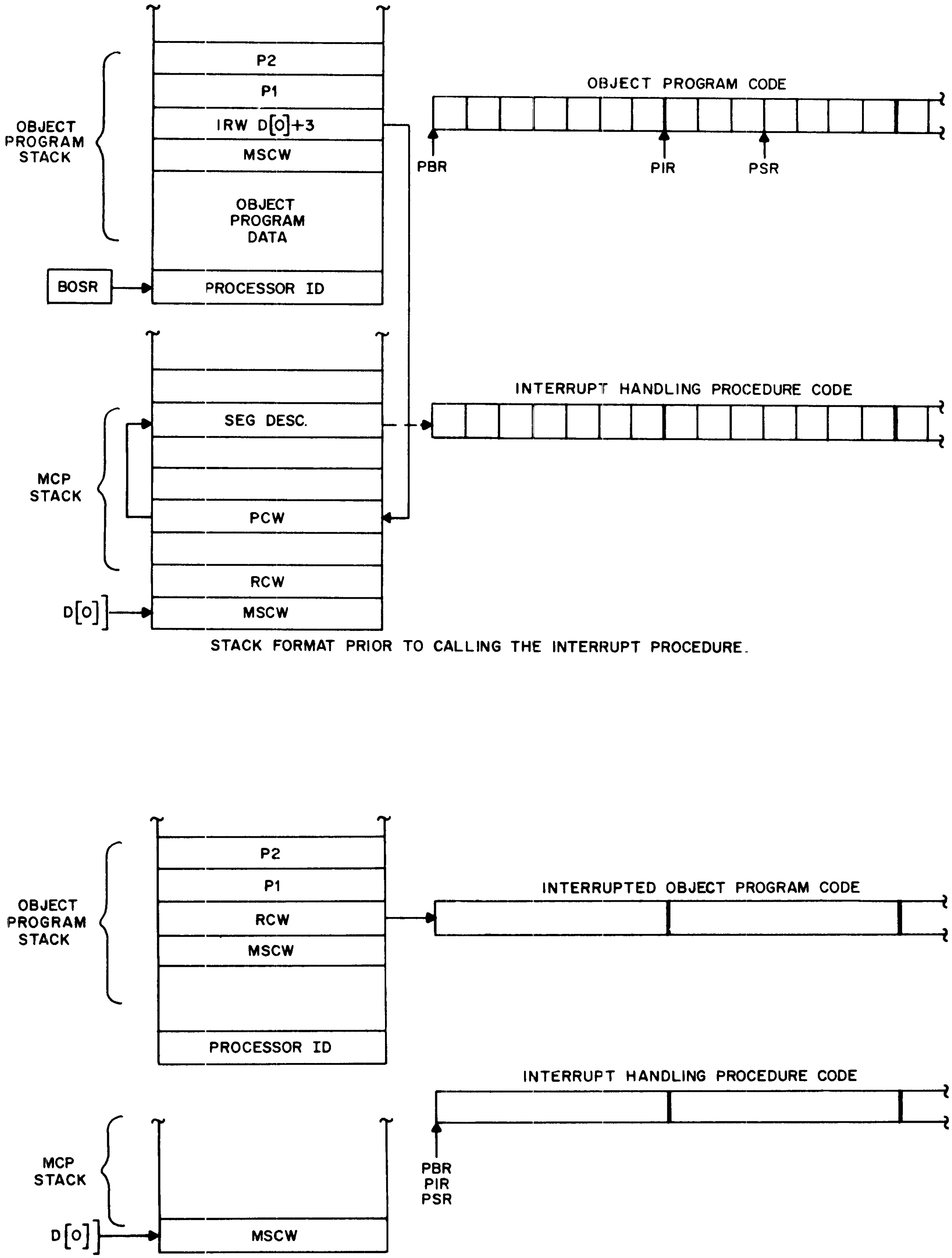
e. Place a second parameter into the stack (the P2 parameter), giving further information about the interrupt.

f. Execute an Enter operator. The fault control logic expects to find a Program Control Word (PCW) at D [0] + 3; however, an SIRW, an IRW or an IRW chain which points to a PCW are possible conditions.

The two interrupt parameters (P1 and P2) that are inserted into the stack supply information describing the interrupt condition. The P1 parameter provides information concerning the type of interrupt, the interrupt priority level, and the interrupt class. The P2 parameter supplies supplementary information about the interrupt condition, such as a memory address (memory related interrupts) or a copy of the non-present descriptor (presence bit interrupts), etc. If P2 is not used by the interrupt condition being reported, P2 will be set to zero.

When the interrupt procedure of the MCP is entered, the IRW in the stack (step c above) is overwritten with a Return Control Word (RCW) by the ENTER operator. As with any procedure entry, this RCW points to the point in the code string to which control is to be returned following execution of the procedure.

Figure III-2-3 depicts the stack format just



STACK FORMAT PRIOR TO CALLING THE INTERRUPT PROCEDURE.

STACK FORMAT AFTER ENTERING THE INTERRUPT PROCEDURE

Figure III-2-3. Stack Format

prior to and just after entering the interrupt procedure.

INTERRUPT PROCESSING IN CMI

When an interrupt is detected while in CM1, the CPM advances to CM2 and attempts to enter the MCP interrupt procedure from its alternate stack. The new stack is found by using the processor number as an index into the Stack Vector Array. (This array is pointed to by the Stack Vector Descriptor, located at D [0] + 2.) The index into the Stack Vector Array results in a data descriptor, which points to the base of the stack for the new stack. Alternate stacks are established by the MCP at the time of system initialization.

The Bottom Of Stack Register (BOSR) is set to the base address of the new stack, which contains the Top Of Stack Control Word (TOSCW) for the new stack. A modified move-to-stack operation then causes the TOSCW for the old stack, the old BOSR setting, and the old SNR register (stack number) setting to be placed in the top of the new stack. After these parameters have been placed, the stack is marked, the IRW and the P1 and P2 parameters are placed in the stack, and the MCP interrupt procedure is entered. The stack structure just prior to entering the MCP interrupt procedure is shown in figure III-2-4.

INTERRUPT PROCESSING IN CM2

At system initialization time, the MCP establishes a special CM3 operating environment at the top of memory. This environment includes an abbreviated D [0] stack with its own stack vector and an interrupt handler. The main memory address of this alternate D [0] stack is loaded into the ADZ register of all CPM's. When a CPM detects an interrupt while in CM2, the CPM advances to CM3, and changes to the CM3 environment by setting the D [0] register to the value in ADZ. The CPM then attempts to move to its alternate stack in the new stack vector (at ADZ+2) and enter the new interrupt handler at ADZ+3, as described in the previous paragraphs.

INTERRUPT PROCESSING IN CM3

If an interrupt is detected while in CM3, it is obvious that a recursive interrupt condition exists. In such cases the CPM is halted in CM4, the most recent interrupt is identified in the Fault Condition Register, the HSI flip-flop is set, and the CM3 bit is set.

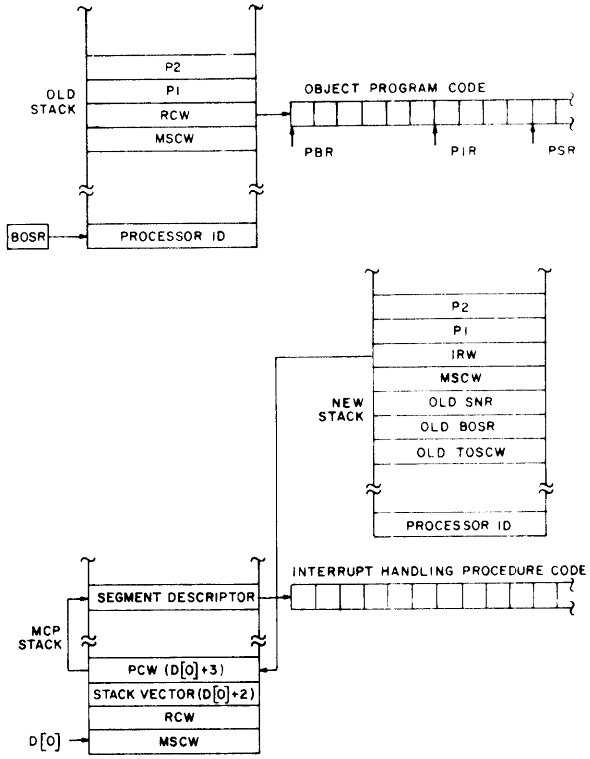


Figure III-2-4. Stack Format Prior to Calling Interrupt Procedure While in CM1 (Move Stack Operation)

CONTROL MODE ADVANCEMENT

Figure III-2-5 illustrates the priority scheme for reporting interrupts, the conditions for advancing the Control Mode register, and the interrupt conditions which may be left in the Fault Condition register for later servicing. In case 1, the Fault Condition register contains an Alarm interrupt (first priority) a Stack Overflow interrupt (third priority), and may also contain Syllable interrupts (second priority), Interval Timer interrupts (fifth priority), and External interrupts (fourth priority). The Alarm interrupt causes the Control Mode register to be advanced (from Normal to CM1, CM1 to CM2, CM2 to CM3, or from CM3 to CM4), the P1 parameter reports the Alarm interrupt, and the External interrupts are still contained in the Fault Condition register (all other interrupts are cleared from the register).

Case 2 shows all priorities of interrupts except Alarm interrupts present in the Fault Condition Register. The resultant action is similar to case one, in that the highest priority interrupt (Syllable) is serviced first. P1 reports the Syllable interrupt, the Control Mode register is advanced, and the Stack Overflow and External interrupts are still contained in the Fault Condition register (in this case the Interval Timer interrupt is also left in the Fault Condition register).

Following entry into the software interrupt procedure, the Stack Overflow interrupt is reported by another P1, the Control Mode register is advanced, the Interval Timer interrupt is cleared from the Fault Condition register, and the External interrupts are left for later servicing. The stack structure for either case one or case two is shown in figures III-2-6 and III-2-7.

Case 3 of figure III-2-5 shows a Syllable interrupt (second priority), an Interval Timer interrupt (fifth priority), and an External interrupt (fourth priority) all present in the Fault Condition register. In this case, the highest priority interrupt present (Syllable) is reported in P1, the Control Mode register is advanced, and the Interval Timer and External Interrupts are left for later servicing.(The external interrupt is serviced first.)

Case 4 shows a Stack Overflow interrupt, an Interval Timer interrupt, and an External interrupt present in the Fault Condition register. The Stack Overflow interrupt is reported in P1, the Interval Timer interrupt is cleared from the register, and the External interrupt is left for later servicing.

Case 5 shows servicing of an External interrupt, leaving an Interval Timer interrupt for later servicing. Case six shows servicing of an Interval Timer interrupt. Notice that these two cases can only occur when the CPM is in Normal State. (When the CPM advances to CM1 and the MCP interrupt procedure is entered, the CPM operates in Control State and the recognition of Interval Timer and External interrupts is inhibited.)

ALARM INTERRUPTS (FIRST PRIORITY)

Detection of an Alarm interrupt causes an immediate entry (or reentry) into the fault control logic. The Control Mode register is advanced and a P1 parameter is formed which identifies all Alarm interrupts which are present in the Fault Condition register. Syllable Dependent interrupts, Stack Overflow Interrupts, and Interval Timer interrupts (if present) are cleared from the Fault Condition register and the interval timer is disarmed. The MCP interrupt procedure is entered.

SYLLABLE DEPENDENT INTERRUPTS (SECOND PRIORITY)

Detection of a Syllable Dependent interrupt (if no Alarm interrupts are present) causes an immediate entry (or reentry) into the fault control logic. The Control Mode register is advanced and a P1 parameter is formed which identifies all Syllable Dependent interrupts which are present. The MCP interrupt procedure is entered.

SPECIAL INTERRUPTS

STACK OVERFLOW (THIRD PRIORITY). All Stack Overflow interrupts are processed by the fault control logic and causes advance of the Control Mode register. All Stack Overflow interrupts do cause a P1 parameter reporting the interrupt to be formed. Interval Timer interrupts (if unmasked) are cleared from the Fault Condition register and the Interval Timer is disarmed. The MCP interrupt procedure is entered.

INTERVAL Timer (fifth priority). Interval Timer interrupts are cleared (and the interval timer is disarmed) when an Alarm or Stack Overflow, interrupts is reported. All uncleared Interval Timer interrupts cause entry into the fault control logic if the mask is set and either is in normal state or if executing an IDLE in Control state. The Control Mode register is advanced to CM1 (from Normal). (Interval Timer interrupts are inhibited when the CPM is in Control State.) The MCP interrupt procedure is entered.

EXTERNAL INTERRUPTS (FOURTH PRIORITY)

Although External interrupts can occur at any time, these interrupts, with the exception of the Egg Timer interrupts, are inhibited when the CPM is in Control State. If an External interrupt occurs when the CPM is in Normal State, the Control Mode register is advanced to CM1, a P1 parameter describing the external interrupts is formed, the Interval Timer is disarmed, and the MCP interrupt procedure is entered.

MEMORY RELATED INTERRUPTS

Memory related interrupts include Memory Parity errors (MPAR) which are discovered by the Communications Unit), Memory Fail 1 (MF1) errors (which are discovered by the MCM and reported to the requestor), invalid address errors (which are detected by the Communications Unit in its interface with the MCM's as an NAM) and Processor Internal errors (PINT) (which are discovered by the Communications Unit in its interface with the other units of the CPM). It should be noted that PINT interrupt can also be set by other units in the CPM. These four types of errors are differentiated in the P1 interrupt parameter. Memory Fail 1 interrupts from all MCM's are combined into a single alarm interrupt, represented by bit 2 in the Fault Control register and the Interrupt ID; the identifica-

FAULT CONDITION REGISTER (BEFORE REPORTING INTERRUPT)					CONTROL MODES REGISTER				REPORTED IN	FAULT CONDITION REGISTER (AFTER REPORTING INTERRUPT)					
INTERVAL TIMER	EXTERNAL	STACK OVERFLOW	SYLLABLE	ALARM	NORMAL	CM 1	CM 2	CM 3	PARAMETER P1	EXTERNAL	INTERVAL TIMER	STACK OVERFLOW	SYLLABLE	ALARM	
ϕ	ϕ	1	ϕ	1	CM1	CM2	CM3	* —	ALARM	ϕ	0	1	0	0	CASE ONE
ϕ	ϕ	1	1	0	CM1 CM2	CM2 CM3	CM3 *	* –	SYLLABLE STACK OVERFLOW	ϕ ϕ	ϕ 0	1 0	0 0	0 0	CASE TWO
ϕ	ϕ	0	1	0	CM1	CM2	CM3	*	SYLLABLE	ϕ	ϕ	0	0	0	CASE THREE
ϕ	ϕ	1	0	0	CM1	CM2	CM3	*	STACK OVERFLOW	ϕ	0	0	0	0	CASE FOUR
ϕ	1	0	0	0	CM1	–	–	–	EXTERNAL	ϕ	0	0	0	0	CASE FIVE
1	0	0	0	0	CM1	–	–	–	INTERVAL TIMER	0	0	0	0	0	CASE SIX

* PROCESSOR HALTS
ϕ MAY BE A ONE OR A ZERO

40354

Figure III-2-5. Interrupt Reporting

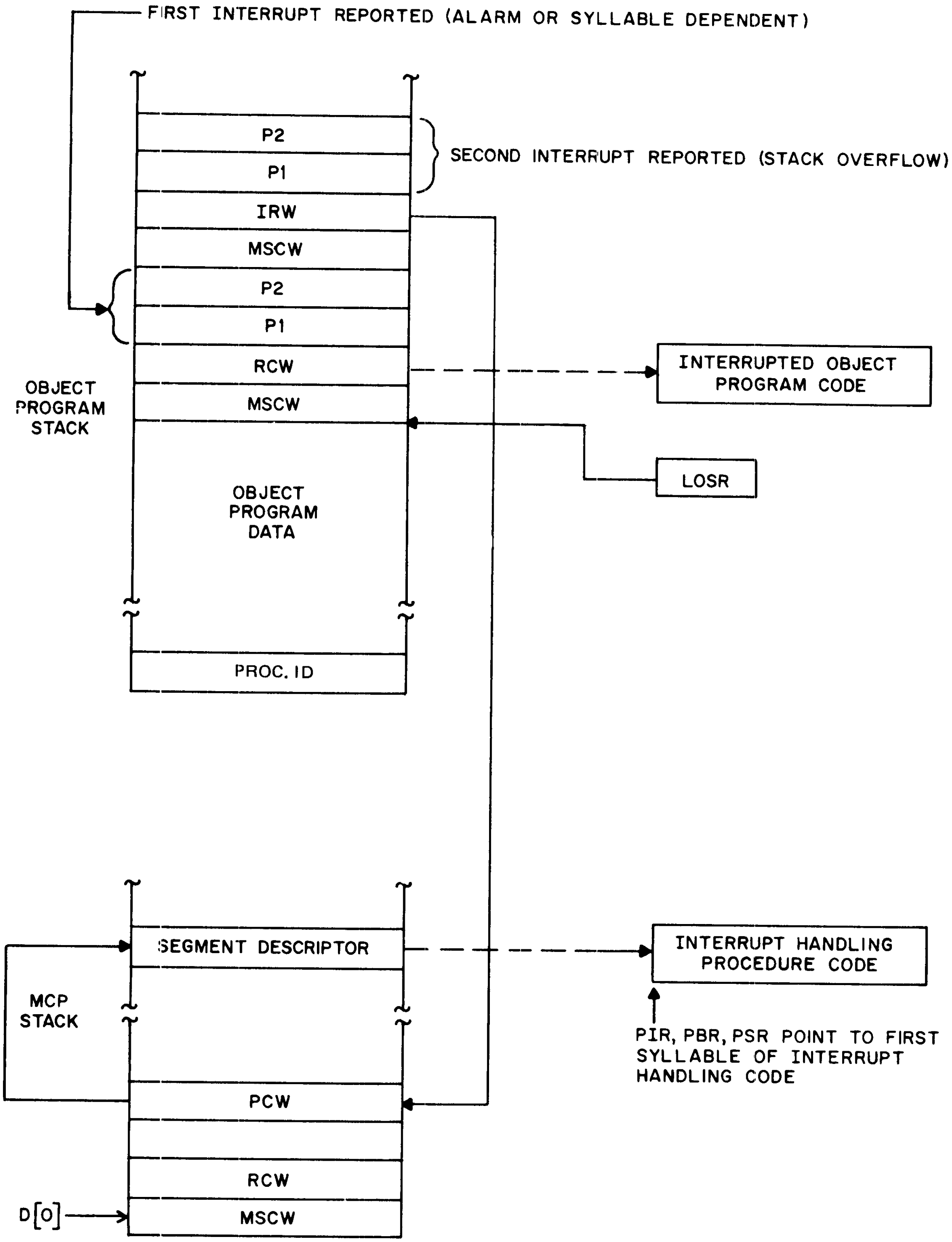

FIRST INTERRUPT REPORTED (ALARM OR SYLLABLE DEPENDENT)
P2
P1
SECOND INTERRUPT REPORTED (STACK OVERFLOW)
IRW
MSCW
P2
P1
RCW
INTERRUPTED OBJECT PROGRAM CODE
OBJECT PROGRAM STACK
MSCW
LOSR
OBJECT PROGRAM DATA
PROC. ID
SEGMENT DESCRIPTOR
INTERRUPT HANDLING PROCEDURE CODE
MCP STACK
PIR, PBR, PSR POINT TO FIRST SYLLABLE OF INTERRUPT HANDLING CODE
PCW
RCW
D[0]
MSCW

Figure III-2-6. Stack Format Before Reentering Interrupt Procedure to Report Stack Overflow

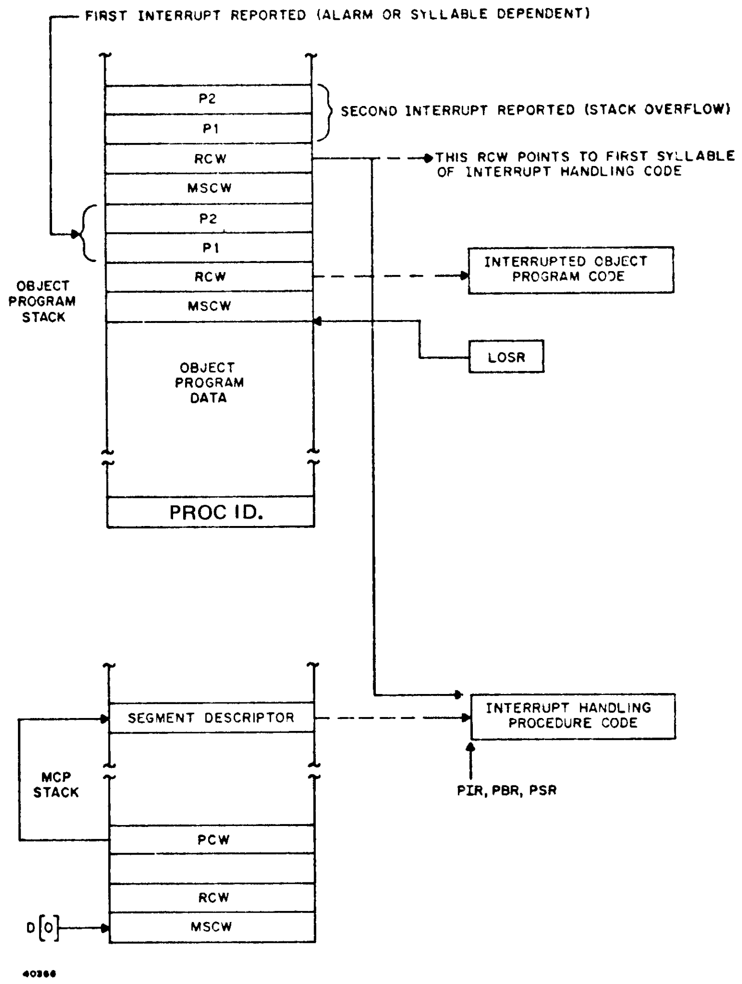

FIRST INTERRUPT REPORTED (ALARM OR SYLLABLE DEPENDENT)
P2
P1
SECOND INTERRUPT REPORTED (STACK OVERFLOW)
RCW
THIS RCW POINTS TO FIRST SYLLABLE OF INTERRUPT HANDLING CODE
MSCW
P2
P1
RCW
INTERRUPTED OBJECT PROGRAM CODE
OBJECT PROGRAM STACK
MSCW
LOSR
OBJECT PROGRAM DATA
PROC ID.
SEGMENT DESCRIPTOR
INTERRUPT HANDLING PROCEDURE CODE
MCP STACK
PIR, PBR, PSR
PCW
RCW
D[0]
MSCW
40344

Figure III-2-7. Stack Format After Reentering Interrupt Procedure and Reporting Stack Overflow

tion of which MCM was involved is given in the P2 parameter or in the CPM fail register, depending on the type of CU operation being performed. Wrong Channel Number and Memory Time Out error indications in the CPM fail register are combined into a single alarm interrupt, represented by bit 3 (Invalid Address) in the Fault Control register and the Interrupt ID. Likewise, CU residue (address residue error in CA register) and Bad Parity to CU error indications in the CPM Fail register are combined into a single alarm interrupt, represented by bit 6 (Processor Internal) in the Fault Control register and the Interrupt ID.

Explanatory information about these errors may be found either in P2 parameter or in the CPM Fail register. If P2 is not used, it will be set to zero. Details regarding the handling of these interrupts are provided in table III-2-2.

Memory Fail 2 interrupts (single bit errors, corrected) are also combined into a single interrupt in the CPM Fault Condition and Fault Mask registers. The identification of the MCM involved is given in the CPM Fail register. Since the error is corrected, the corrected data can be and is used.

INTERRUPT DESCRIPTIONS

Interrupts which can occur in the CPM are described in the following paragraphs. The interrupts are described in order of their priority. Alarm interrupts are described first, Syllable Dependent interrupts second, Special interrupts third, and External interrupts last.

ALARM INTERRUPTS

Alarm interrupts are caused by conditions which were not expected by the CPM. They inform the system of some detrimental change in environment. In most cases, Alarm inter-

Table III-2-2. CPM Handling of Memory Related Errors

Source of Request for Memory Access	Alarm Interrupts (MPE, MF1, IAE, PINT)
Storage Unit fetch (includes flashback)	The indication of the error (or error word) is queued in place of the result of the access (whether in the EWR, EU data queue, or the associative memory). When the EU attempts to use the error word, the alarm interrupt occurs. The error word is placed in R1 location of the EU local storage as the P2 parameter.
Storage Unit write (no flashback)	The alarm interrupt occurs immediately. There is no P2 parameter; rather the explanatory information (including the identity of the involved MCM) is contained in the CPM Fail register.
Stack Buffer fill	The error word is stored in the stack buffer and the alarm interrupt does not occur until the execution unit attempts to use this word. The error word is placed in R1 location of the EU local storage as the P2 parameter.
Stack Buffer empty or purge	The alarm interrupt occurs immediately. The reason for this is that stack-to-memory transfers are initiated only when the Execution Unit purges the stack buffer or when the Execution Unit attempts to push new information into the stack and the stack is full; hence, a stack empty operation is required. In either case, the Execution Unit is waiting for the memory access and cannot complete the current operator until the memory access is completed or, as in the error cases, aborted.
Program Buffer fetch	The error word is queued in the buffer and not reported until the Program Control Unit attempts to process the code string involved. The error word is then passed onto the Execution Unit as the P2 parameter.

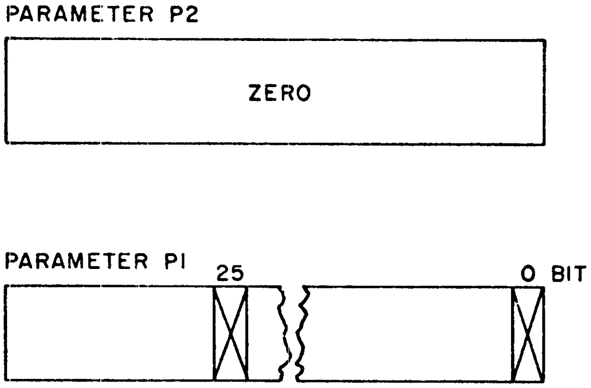
rupts result from hardware failures. The Alarm interrupts cannot be inhibited, and always cause entry into the fault control logic. The fault control logic terminates the current operator, clears the top of stack registers, prepares the stack (MSCW, IRW, P1, P2), and causes the MCP interrupt procedure to be entered. When an Alarm interrupt is cleared from the Fault Condition register, all Syllable Dependent interrupts present in the register are cleared. The Alarm interrupts are:

Loop
Memory Parity
Memory Fail 1
Invalid Address (no access)
Stack Underflow
Invalid Program Word
Processor Internal

Alarm interrupts generally result in termination of the process involved. Exceptions are (1) during a halt load when the MCP uses an alarm interrupt (invalid address) to determine the amount of memory available, and (2) when instruction retry by the MCP is successful after a Processor Internal interrupt.

LOOP

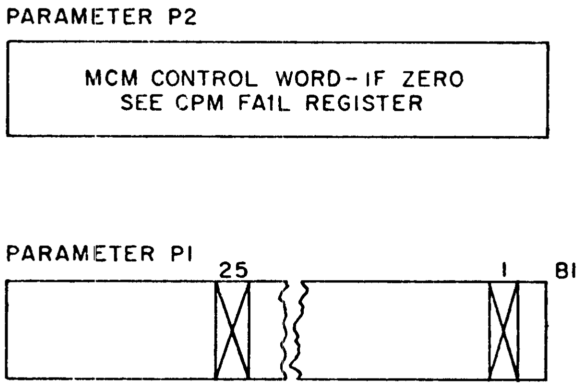
This interrupt occurs when the CPM has expended two or in some cases up to five seconds in the execution of one operator. This interrupt can be caused either by a hardware failure or by bad data. Should this interrupt occur, PIR may not be accurate.



Loop Interrupt Parameters

MEMORY PARITY

This interrupt occurs if the CPM receives a memory word with an even number of 1's. Should this interrupt occur, PIR points to the word containing the operator which initiated the interrupt. Supplementary information describing the error will be contained in the CPM Fail register (see "Memory Related Interrupts").



Memory Parity Interrupt Parameters

MEMORY FAIL 1

This interrupt occurs if any of the following errors occur:

a. Data Word parity error

b. Illegal operation code

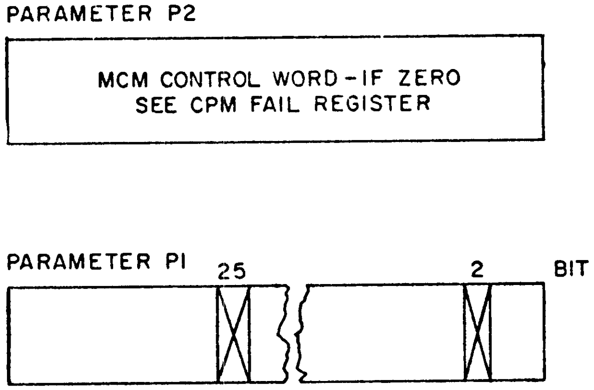
c. Address is for a different Memory Module than requested

d. Data strobe error

e. Internal control error

f. Multiple bit data error

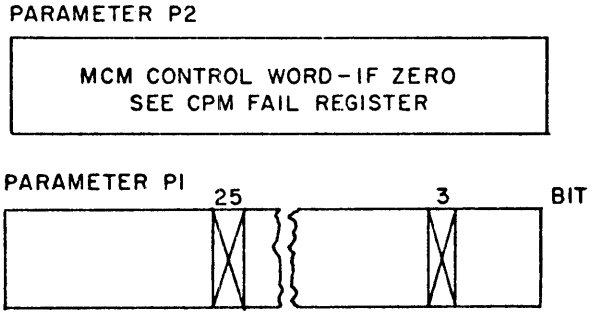
In all of the above cases, supplementary information describing the error will be contained in the MCM Fail register (see "Memory Related Interrupts").



Memory Fail 1 Interrupt Parameters

INVALID ADDRESS

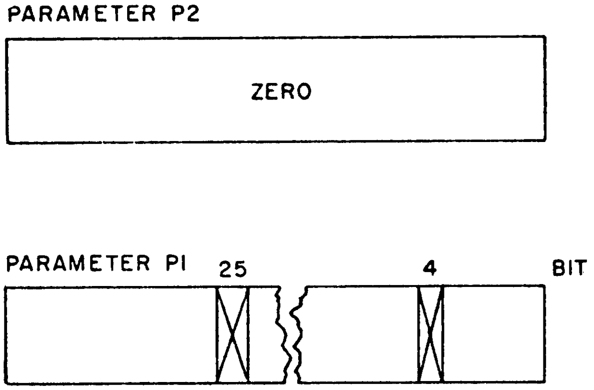
This interrupt occurs when the CPM attempts to access a memory address which is not available to the system. The Memory Module may not exist or it may be inoperative. Supplementary information is placed in the CPM Fail register (see "Memory Related Interrupts").



Invalid Address Interrupt Parameters

STACK UNDERFLOW

This interrupt occurs if the CPM attempts to move the top of stack (S register setting) to an address less than the address of the most recent MSCW (F register setting) during a stack adjustment. This could occur as a result either of a compiler error or if a hardware control failure in executing MKST, EXIT, or MVST (all of which change F setting and could calculate an incorrect address).



Stack Underflow Interrupt Parameters

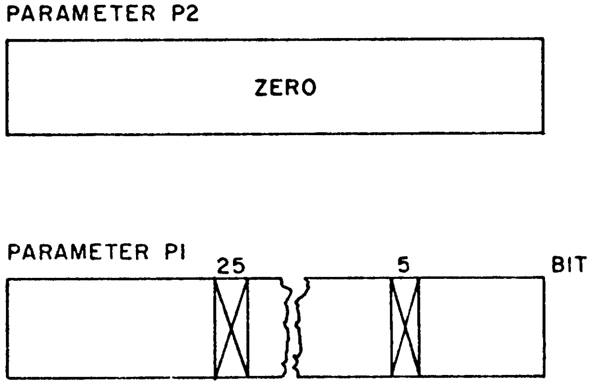
INVALID PROGRAM WORD

This interrupt occurs under any of the following conditions:

a. An attempt is made to execute a program word which does not have a tag of three (or tag of 0 if in Edit mode

b. The Variant code (Escape to 16-bit Instruction, VARI) is detected as the second syllable of a Variant operator.

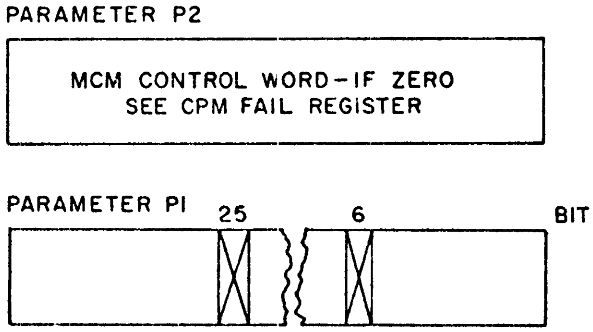
c. An attempt is made to execute an operator which is considered illegal in Edit mode or Vector mode.



Invalid Program Word Interrupt Parameters

PROCESSOR INTERNAL

This interrupt occurs whenever an internal logic failure is detected within the CPM. The CPM Fail register will provide additional information regarding the failure (see table III-2-3). For further information regarding memory related Processor Internal interrupts, see "Memory Related Interrupts".



Processor Internal Interrupt Parameters

Table III-2-3. Processor Internal Interrupts

Fail Register Indication	Possible Error Source
Execution Unit Continuity (EUC)	EU data transfers Inconsistent execution of EU algorithm PCU provided incorrect sequence of micro-operators or wrong stack count (AFUL, BFUL) to the Execution Unit
Execution Unit Residue (EUR)	C, D, E, F, or G register Main or exponent adder, barrel, auxiliary storage, T or S or X bus, or residue generator Extra transfer signal in the use of one of the above elements Address or PIR from AU storage, display buffer, or EWR to the Execution Unit Residue control logic (use of ABC, BOC, BLU, and EWF registers) Residue error in SH, AL, or R register Leading zeros residue encoder K or L registers, or K/L Queue Binary divide quotient look-head logic
Execution Unit Parity (EUP)	Associative memory EU local storage Stack buffer Program buffer EU data queue EWR, T bus, S bus, C register, or EU parity generator Parity error in program buffer
Program Control Unit Error (PER)	Instruction execute register residue error in PIR Q SSR or AU storage (APIR or TIR location) AU adder PA, PL, PU registers Display read register
Adder Residue (ADD)	S register (for MKST, MVST, and RPRR S operations or when select address couple (SAC) flip-flop (1-8-42) is set.) F register (for MKST, ENTR, MVST, and RPRR F operations.) LOSR (for RPRR, LOSR, and MOV operations) Display buffer Display read register Display write register AU adder Display write residue checker Execution Unit 20 bit residue generator
Input Register Parity on data from CPM (INP)	Parity error in storage data queue Input register Output register CU parity generator Parity error in stack buffer (for empty or purge operations)

Table III-2-3. Processor Internal Interrupts (Cont.)

Fail Register Indication	Possible Error Source
CU Address Residue (CRS)	Storage Unit related failure:
	AU storage
	Display read register
	AU adder
	Storage input register
	Storage operator queue
	Execution Unit 20 bit residue generator
	Memory address register
	CU address register or residue checker
	Stack Buffer related failure:
	Stack link register
	CU address register or residue checker
	Program buffer related failure:
	AU storage location (PBR or TBR)
	Display read register
	AU adder
	PA register
	PU register
	CU address register or residue generator
	Remember suspend register

SYLLABLE DEPENDENT INTERRUPTS

Syllable Dependent interrupts generally result from programming errors. These interrupts cannot be inhibited, and always cause entry into the fault control logic. The fault control logic terminates the current operator, prepares the stack (MSCW, IRW, P1, P2), and causes the MCP interrupt procedure to be entered. The contents of the top of stack registers may or may not be saved, depending upon the type of interrupt.

Syllable Dependent interrupts are divided into two classes. Class 1 interrupts (identified by the setting of bit 24 of parameter P1) are those interrupts in which the values of PIR, PSR, PBR, and PDR have not been modified by the operator. Class 2 interrupts (identified by the setting of bit 23 of parameter P1) are those interrupts in which the value of PIR, PSR, PBR, and PDR have been changed. Thus, class 1 interrupts permit the operator to be reexecuted; class 2 interrupts prohibit the operator from being reexecuted.

Most Syllable Dependent interrupts occur as class 1 interrupts. The only Syllable Dependent interrupts which can occur as class 2 interrupts are the Invalid Index, Bottom Of Stack, and Sequence Error interrupts. The Syllable Dependent interrupts are:

Memory Protect
Invalid Operand
Divide By Zero
Exponent Overflow
Exponent Underflow
Invalid Index
Integer Overflow
Bottom Of Stack
Presence Bit
Sequence Error
Segmented Array
Programmed Operator
Privileged Instruction

MEMORY PROTECT

This interrupt occurs when:

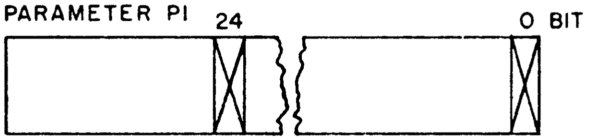
a. a store, overwrite, read/lock, or string transfer operation is attempted using a data descriptor that has the read only bit (bit 43) set. The operation is terminated before the memory access. The data descriptor is used as the P2 parameter, except for string transfer.

b. a store operation is attempted into a word in memory that has a tag field representing a PCW, RCW, MSCW; or segment descriptor (tag = 3, 7). The memory write is discontinued when bit 48 is detected in the code word being referenced. The flashback is used as the P2 parameter.

PARAMETER P2

DATA DESCRIPTOR WITH BIT 43 SET, OR MEMORY WORD WITH THREE TAGS, OR NUMBER OF ITEMS BELOW THE MSCW NEEDED TO GET THE DATA DESCRIPTOR

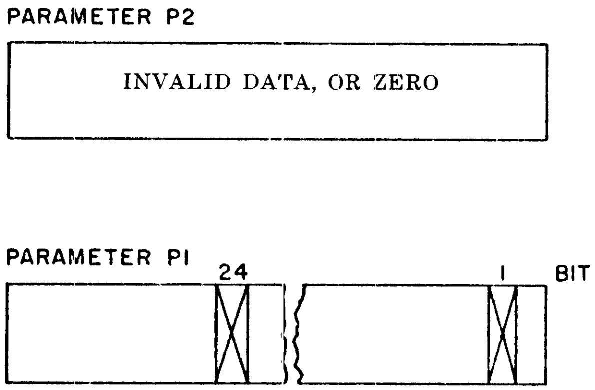
PARAMETER PI



Memory Protect Interrupt Parameters

INVALID OPERAND

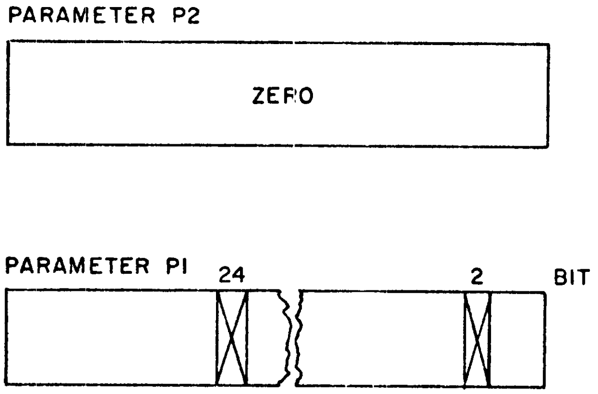
This interrupt occurs when the CPM attempts to execute a valid operator on data which is invalid for that operator or attempts to execute the invalid operator NVLD. Each operator executes checks to insure that control words and data meet the necessary requirements of the operator. Should this interrupt occur, PIR and PSR are left pointing to the current syllable.



Invalid Operand Interrupt Parameters

DIVIDE BY ZERO

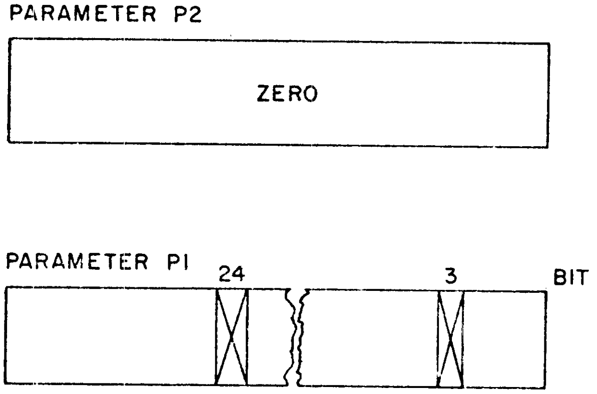
This interrupt occurs when a division operation is attempted with the divisor (contained in the A register) equal to zero. Should this interrupt occur, PIR and PSR point to the initiating operator, and the divisor and dividend will be left on the top of the stack (below the MSCW, RCW, P1, and P2).



Divide By Zero Interrupt Parameters

EXPONENT OVERFLOW

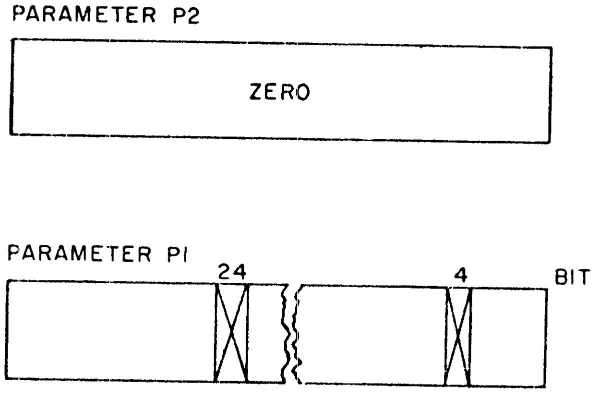
This interrupt occurs when the capacity of a positive sign exponent field is exceeded for either single or double precision arithmetic results. Should this interrupt occur, PIR and PSR point to the initiating operator.



Exponent Overflow Interrupt Parameters

EXPONENT UNDERFLOW

This interrupt occurs when the capacity of a negative sign exponent field is exceeded for either single or double precision arithmetic results. Should this interrupt occur, PIR and PSR point to the initiating operator.



Exponent Underflow Interrupt Parameters

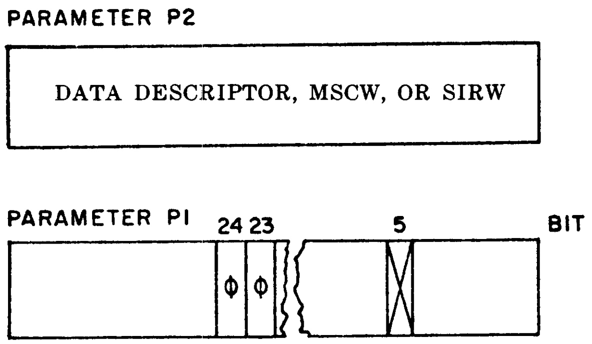
INVALID INDEX

This interrupt occurs if an attempt is made to index a descriptor by an amount which is less than zero or which is greater than or equal to the upper bound (length) in any of the following operations:

a. Occurs Index
b. Linked List Lookup
c. Index
d. Move Stack
e. Display Update
f. VALC
g. Stuffed IRW (pseudo operator)
h. Index and Load Name
i. Index and Load Value

If this interrupt occurs, the operation is terminated prematurely. The input operands will be left on the top of the stack (below the MSCW, RCW, P1, and P2). Except for Display Update, all operations in the list above will cause PIR and PSR to point to the initiating operator. The interrupt occurs as a class 2 interrupt (bit 23 = 1) if an attempt was made to index the Stack Vector Array descriptor (D [0] + 2) during a display update operation using a stack number which is greater than or equal to the length field of the Stack Vector Array descriptor.

Note+ Bit 23 and bit 24 may not both be set simultaneously.



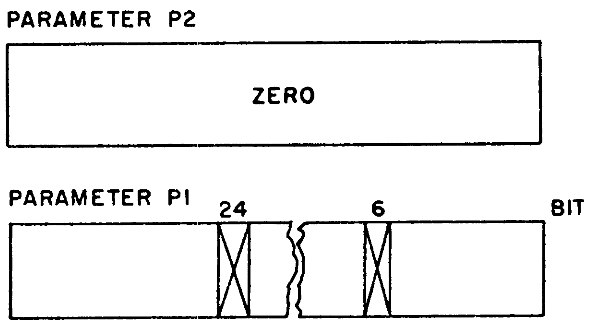
Note: Bit 23 and Bit 24 may not be set simultaneously.

Invalid Index Interrupt Parameters

INTEGER OVERFLOW

This interrupt occurs upon detection of the attempted use of an operand which exceeds the maximum integer value (2^{39}-1) by an operator which requires an integer. The following is a partial list of operators which may cause this interrupt to occur:

a. Integer Divide
b. Integerize Truncate
c. Integerize Rounded
d. Occurs Index

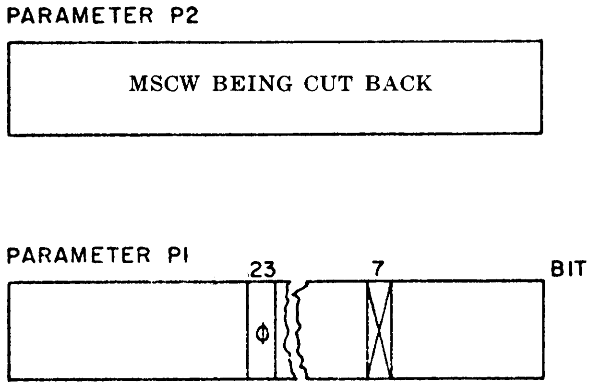


Integer Overflow Interrupt Parameters

BOTTOM OF STACK

This interrupt occurs if a Return operator or an Exit operator causes the program stack to be cut back to its base. (The F register points to the MSCW located at the BOSR setting plus 1.) The P2 parameter will be a copy of the MSCW being cut back.

This interrupt occurs as a class 2 interrupt (bit 23 = 1).



Bottom Of Stack Interrupt Parameters

PRESENCE BIT

This interrupt occurs when an attempt is made to access a word or group of words which are not present in main memory. All operators that access memory with descriptors may be interrupted with this interrupt. The interrupt occurs if an attempt is made to reference memory through a descriptor which has the presence bit (bit 47) reset, indicating that the descriptor points to words which are not present in main memory. There are two classes of presence bit interrupt conditions; data dependent and procedure dependent.

DATA DEPENDENT. Data dependent presence bit interrupt conditions occur when the CPM is seeking data from within its current addressing environment. In all cases except Value Call, recovery is achieved by re-executing the operator upon return from the MCP interrupt procedure. The MCP interrupt procedure makes the absent words present before return is made to the interrupted program. To permit this reexecution, the PIR and PSR settings for the current operator are saved in the RCW. Value Call always sets this RT bit for data dependent interrupts; however, Value Call never sets this RT bit for procedure dependent interrupts. Value Call or pseudo value call will always turn on the VS bit (bit 39) and cause the V bit in the MSCW to be turned on. Figure III-2-8 illustrates the PIR, PSR, Exit/Return, RT, VS, and RE bit relationships in the various presence bit interrupt conditions.

ACCIDENTAL ENTRY. Procedures which have been entered accidentally during the VALC operator also require special consideration for the manipulation of PIR and PSR settings for the RCW. The VALC operator is completed after the return operator mechanism when returning from an accidentally entered procedure. A pseudo value call operator provides the facility to continue searching an IRW or data descriptor chain until an operand is located. The pseudo value call operator is activated at the end of a normal return operator if the V bit of the MSCW had been set. The V-bit is set when either a VALC or pseudo value call operator enters a procedure accidentally. If a not present segment descriptor causes an interrupt during a return from an accidental entry of value call, a pseudo RT bit (Bit 45) is turned on in P1 so the presence bit procedure will finish with a return instead of an exit if the VS (Bit 39) is also on. The RT bit and pseudo RT bit are used by the software to execute the proper code. The V bit is used by the hardware to change the return into a pseudo value call so the IRW or data descriptor chain may be chased.

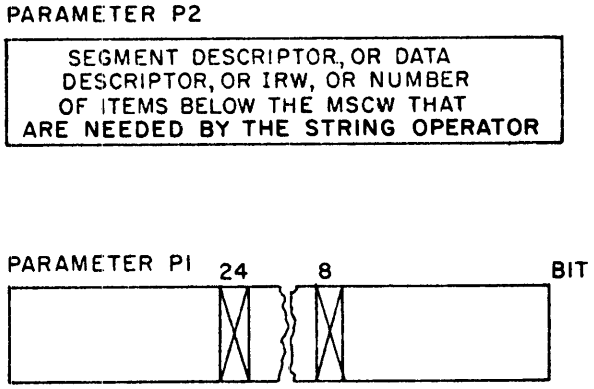
PIR and PSR values, pointing to the next operator syllable, are inserted into the RCW for VALC while the PIR and PSR values from the old RCW are inserted into the RCW for a value call pseudo operator.

All other operators which may incur accidental entries are restarted; therefore, the PIR and PSR settings which point to the current operator syllable are saved in the RCW. The V-bit is set to zero.

PROCEDURE DEPENDENT. Procedure dependent interrupts occur when the CPM is attempting to enter a new addressing environment, or attempting to return to an old addressing environment. These interrupts occur during display update, and also when trying to process a non-present segment descriptor. Recovery is achieved by the Exit operator or the Return operator after the MCP interrupt procedure has made the referenced environment present. Because the CPM has not yet fetched the first operator of the new procedure when this interrupt occurs, the PIR and PSR settings from the PCW (for entry) or the RCW (for return) are stored in the RCW which is made when the MCP interrupt procedure is entered. Thus, when the reference environment is made present, the entry or return is to the referenced environment.

PROGRAM RESTART. Following a Presence Bit interrupt, a program may be restarted either by executing a Return operator or an Exit operator. The Return operator must return either an IRW or a Data Descriptor. The RT bit of the P1 parameter (bit 46) indicates to the MCP interrupt procedure whether to perform an Exit operator (bit 46 is reset) or a Return operator (bit 46 is set) when returning to the interrupted procedure.

PARAMETER P2. During the execution of certain string operators, if a Presence Bit interrupt occurs the P2 parameter may contain a number which indicates the number of items below the MSCW which are needed by the string operator.


PARAMETER P2
SEGMENT DESCRIPTOR, OR DATA DESCRIPTOR, OR IRW, OR NUMBER OF ITEMS BELOW THE MSCW THAT ARE NEEDED BY THE STRING OPERATOR
PARAMETER P1
24
8
BIT

Presence Bit Interrupt Parameters

SEQUENCE ERROR

This interrupt occurs if an indirect reference encounters an invalid condition or reference sequence. Generally, this interrupt is caused either by a hardware error or a systems software error, and the MCP will terminate the program which generated the interrupt. The interrupt can occur as a class 2 interrupt (bit 23 = 1) only under the following conditions:

a. When a word other than a Segment Descriptor is fetched relative to the PDR during the final algorithm for the Enter, Exit, or Return operators.

b. When the F register points to a word which is not an MSCW at the beginning of execution of the Exit or Return operators.

c. When tracing back through the DF links of an MSCW chain (DF locates the preceding MSCW in the stack) during an Exit, Return, or Move Stack operation and a word which is not an MSCW is fetched.

PRESENCE BIT INTERRUPT CONDITIONS		P_2	P_1 PRESENCE BIT ID			RETURNING OPERATOR	PIR, PSR NEW RCW	SOFTWARE FUNCTION
			RT (46)	VS (39)	RE (45)			
Data Dependent	Stack Vector DD or Stack D.D. During Reference Through Stuffed IRW	DESC (4)*	0	0	0	EXIT	S_n (8)	Locate Not Present D.D. By the IRW. If necessary, make the D.D. present and return an IRW where noted
		IRW (1)	0	0	0	EXIT	S_n (8)	
		IRW (2)	1	1	0	RETURN	S_n+ 2 (8)	
		IRW (3)	1	0	0	EXIT	S_n (8)	
	Data Descriptor	DESC (2) (copy)	1	1	0	RETURN	S_n+ 2 (8)	Search Stack for copies of Not Present D.D. Make MOM and copies present, return present D.D. where noted
		DESC (7) (copy)	1	0	0	RETURN	S_n (8)	
Procedure Dependent	Stack Vector DD or Stack D.D. During Display Update	DESC (6) (copy)	0	0	0	EXIT	From RCW or PCW	Search Stack for copies of Not Present D.D. Make MOM and copies present, Return D.D. where noted
		DESC (5) (copy)	0	0	1	EXIT		
		DESC (2) (copy)	0	1	0	EXIT		
	Segment Desc	DESC (2) (copy)	0	1	0	EXIT	From RCW or PCW	Locate S.D. (MOM) via copy in P_2 AD Field Of Copy Points to MOM
		DESC (6) (copy	0	0	0	EXIT		
		DESC (5) (copy)	0	1**	1	RETURN		

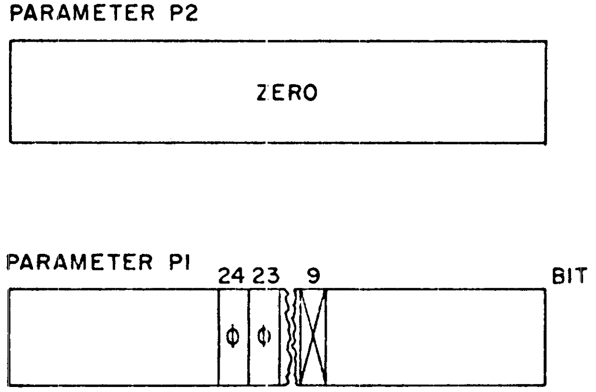
1. Enter or IRWL
2. VALC
3. All Operators Except VALC, ENTR, MVST, RETN, IRWL
4. MVST
5. RETN
6. All Operators Except RETN and VALC
7. All Operators Except ENTR, VALC, or IRWL
8. S_n indicates that PIR and PSR point to current operator syllable.

*Fetch new stack desc thru IRW only.
**If RVLC (V-bit in the MSCW is on).

40994

Figure III-2-8. Presence Bit Interrupt Chart

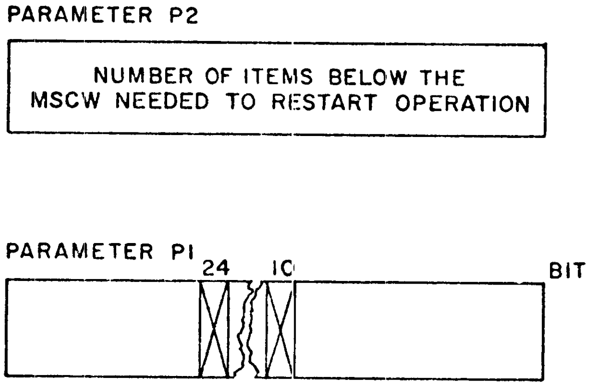
d. When a word which is not a Segment Descriptor is fetched relative to the PDR during a Dynamic Branch operator execution.



Sequence Error Interrupt Parameters

SEGMENTED ARRAY

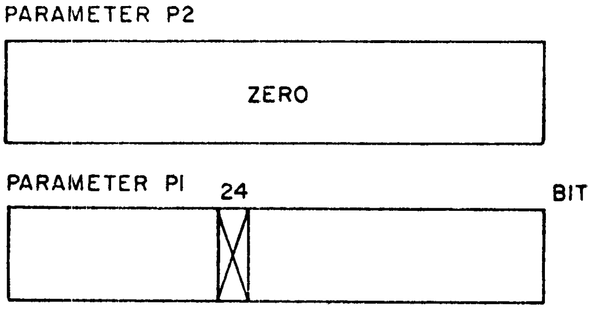
This interrupt occurs when a string operator attempts to index beyond the end of the current segment of a segmented array. Arrays in main memory may be segmented into groups of 256 words each, bounded on both ends by memory links. The memory link words are created by the MCP with the memory protect bit (bit 48) set. During string operations, each word read from memory is checked to see if bit 48 is set. If such a word is referenced, the Segmented Array interrupt will occur. The P2 parameter will indicate how many words (in the stack below the MSCW, RCW, P1, and P2) are needed to restart the operation after the new segment of the array has been made available in main memory.



Segmented Array Interrupt Parameters

PROGRAMMED OPERATOR

This interrupt occurs if the CPM attempts to execute an operator code which is not currently assigned. The Programmed Operator interrupt acts as a communicate operator to the MCP, and allows the MCP to simulate the action of the operator programmatically, if desired. All unassigned operator codes cause this interrupt. (None of the unassigned operator codes cause Loop, Invalid Program, or Invalid Operand interrupts. Scan In Time Of Day Clock is an assigned operator: any other variation of Scan In causes the Invalid Operand interrupt.)

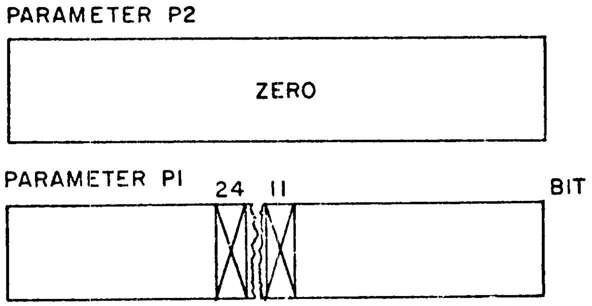


Programmed Operator Interrupt Parameters

PRIVILEGED INSTRUCTION

This interrupt occurs if an attempt is made to execute a Control State operator while the CPM is in Normal State. The Control State operators are:

a. Set Interval Timer (SINT)
b. Inhibit Parity (IGPR)
c. Set Memory Inhibits (SINH)
d. Set Memory Limits (SMLT)



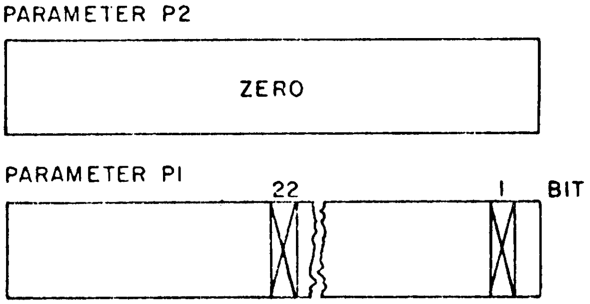
Privileged Instruction Interrupt Parameters

SPECIAL INTERRUPTS

Special interrupts take third priority for processing. There are just two Special interrupts: Stack Overflow and Interval Timer.

STACK OVERFLOW

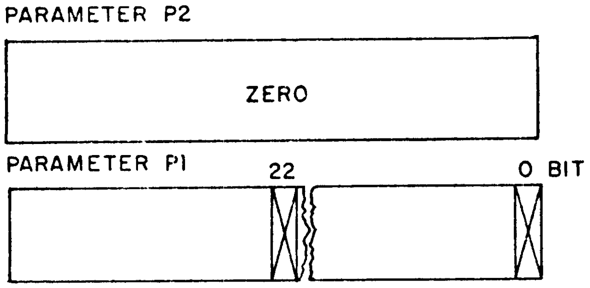
This interrupt occurs when the Stack Controller senses the use of the highest address allotted for the stack of the program (the S register and the Limit of Stack register (LOSR) point to the same address). The MCP interrupt procedure may either allocate a larger stack area, or it may terminate the program. If the current operator has not been completely executed, PIR and PSR are changed to point to the operator.



Stack Overflow Interrupt Parameters

INTERVAL TIMER

This interrupt occurs if the value in the hardware interval timer is zero and the interval timer is armed. The timer is armed and an initial value is stored by the Set Interval Timer operator (SINT). The count in the timer is decreased every 512 microseconds until the count reaches zero or until the timer is reset. If the timer is still armed when the count reaches zero, the interrupt occurs. The maximum interval to which the timer can be set is one second. This interrupt is used by the MCP to insure that no process can control a CPM for more than one second without giving the MCP a chance to regain control of the CPM.



Interval Timer Interrupt Parameters

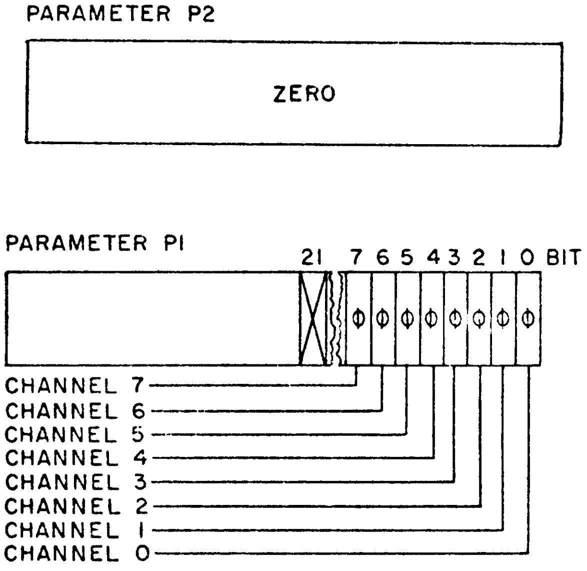
EXTERNAL INTERRUPTS

External interrupts are used to inform the MCP of changes in external environment, and also to permit communications between requestor modules (CPM or IOM). Normally, these interrupts result in the momentary interruption of a program while the interrupt is handled or recorded by the MCP. Following the handling of the interrupt, the program is continued. The External interrupts are:

Channel (0 thru 7)
IOM Error (0 thru 7)
Memory Fail 2
Egg Timer

CHANNEL INTERRUPT

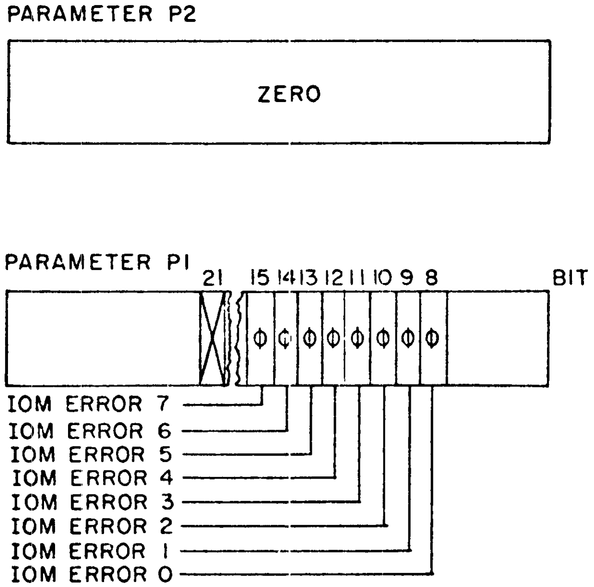
This interrupt may be generated by any of the eight possible requestor modules (CPM or IOM). The interrupt identification (parameter P1) indicates the source of the interrupt. This interrupt may be generated to indicate an expected event (such as IO Complete) or it may be generated by the Interrupt Channel N operator (which allows any CPM to interrupt any requestor module).



Channel Interrupt (0-7) Parameters

IOM ERROR INTERRUPT

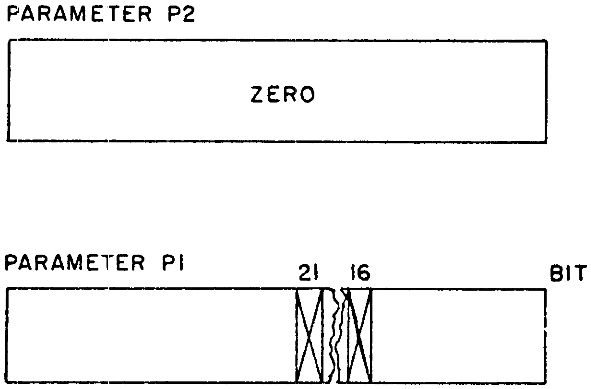
This interrupt may be generated by any of the IO modules in the system. The interrupt identification (parameter P1) indicates the channel (0 thru 7) to which the IOM is connected. This interrupt is used to report errors detected by an IOM which are not device related. If possible, the IOM will link a dummy IOCB into the status queue (RESULTQ). The dummy IOCB will contain a Result Descriptor which will further describe the error. Otherwise, the Fail Result Descriptor will be placed at Home Address (HA) + 5.



IOM ERROR (0-7) Interrupt Parameters

MEMORY FAIL 2

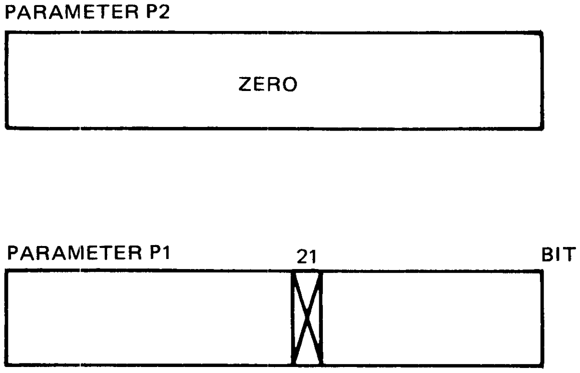
This interrupt occurs if a Memory Control Module detects and corrects a single-bit error. It is transmitted from the MCM (with the corrected data) to the requestor module (CPM or IOM). The MCM Fail register contains the absolute address and the bit number of the word in error. The identification of the MCM involved is contained in the CPM Fail register.



Memory Fail 2 Interrupt Parameters

EGG TIMER INTERRUPT

This interrupt occurs if the Egg flip-flop in the CPM is not reset every 8 to 16 seconds by the MCP. This interrupt is used by the MCP to prevent the CPM from looping while in the control state.



Egg Timer Interrupt Parameters

(THIS PAGE INTENTIONALLY LEFT BLANK)

SECTION 3

OPERATORS

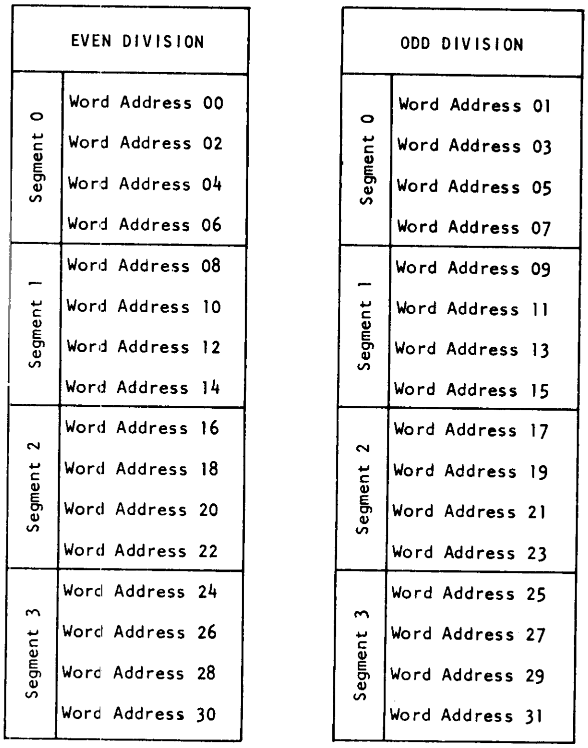
INTRODUCTION

Operators are machine language code generated by the compiler and stored by the master control program in memory in the area allocated to program segments. (Program segments contain no data and are not modified by the processor as the program is executed.) Program segments are sequences of instructions which are moved by the program control unit as 52-bit words from memory into the program buffer. Parity is checked on all 52-bits of each program word as it is brought to the program buffer.

The program buffer, a 32-word, 60-bit IC memory within the processor, locally maintains enough code to keep the processor busy at all times. The buffer may contain 8, 16, 24, or 32 program words. A request is generated to replenish the buffer by fetching 8 words at a time whenever the read pointer is within two words of the writer pointer.

The buffer is interleaved so that it alternately stores all odd-address words from memory in one division of the buffer and stores all even-address words in the other division.

Each division consists of four segments each of which contains four words as shown in Figure III-3-1.



	EVEN DIVISION		ODD DIVISION
Segment 0	Word Address 00	Segment 0	Word Address 01
	Word Address 02		Word Address 03
	Word Address 04		Word Address 05
	Word Address 06		Word Address 07
Segment 1	Word Address 08	Segment 1	Word Address 09
	Word Address 10		Word Address 11
	Word Address 12		Word Address 13
	Word Address 14		Word Address 15
Segment 2	Word Address 16	Segment 2	Word Address 17
	Word Address 18		Word Address 19
	Word Address 20		Word Address 21
	Word Address 22		Word Address 23
Segment 3	Word Address 24	Segment 3	Word Address 25
	Word Address 26		Word Address 27
	Word Address 28		Word Address 29
	Word Address 30		Word Address 31

Figure III-3-1. Format of Program Buffer Word Storage

Each program word consists of 48 bits, 3 tag bits, and an overall parity bit. Since information will be extracted from the program buffer in syllable form there is no way to check overall parity. Therefore, as the program word is parsed into six 8-bit syllables while being loaded into the program buffer, parity is also generated on each syllable of the word and stored in the buffer with each word. The parity of each syllable can thus be checked and the integrity of the program word maintained. Figure III-3-2 illustrates the format of the program buffer word.

Since the B 7700 allows operators to overlap word boundaries, the program buffer is read out serially. Two words are read out of the buffer at the same time, one even address (indicated by the PEB pointer) and one odd (indicated by the POB pointer). These pointers together with the odd/even flip-flops (OEA, OEB, and OEC) then select which of the two words will be left-justified in the barrel switch of the program unit. (These pointers may also be adjusted to facilitate a local branch or entry into an edit table.) The syllables will then be processed left to right in sequence.

An instruction may be either a Value Call, a Name Call, or an operator. The two high-order bits (e.g., bits 7 and 6 in Figure III-3-2) of each instruction determine the type of instruction to be executed.

Instruction Decode Table

Instruction Type	Identification (Bits 7 & 6)	No. of Syllables	Function
Value Call	00	2	Brings an operand into the stack
Name Call	01	2	Brings an IRW into the stack
Operator	1x	1 to 12	Performs the specified operation

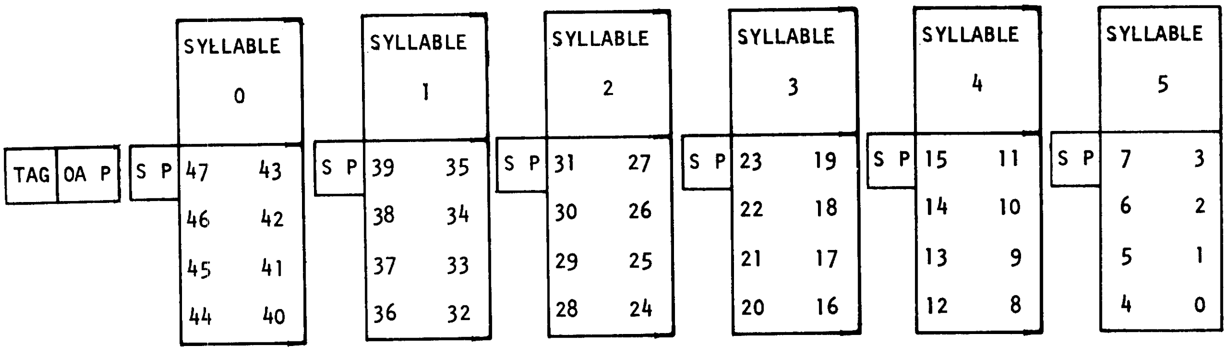


Figure III-3-2. Program Buffer Word Format

Value Call is a two-syllable instruction that brings an operand from memory into the top-of-stack. A concatenation of the two Value Call syllables gives a 14-bit address couple. If the referenced memory location is an indirect reference word or a data descriptor, additional memory accesses are made until the operand is located. The operand is then placed in the top-of-stack register. The operand may be either single-precision or double-precision, causing either one or two words to be loaded into the stack.

Name Call builds an indirect reference word in the stack. Stack adjustment takes place so that the top-of-stack is empty. The six low-order bits of the first syllable for this operator are concatenated with the eight bits of the following syllable to form a 14-bit address couple. The address couple is placed, right-justified into the top-of-stack; the remainder of the top-of-stack register is set to zero. The tag field is set to 001 and the register is marked full.

Operators vary from 1 to 12 syllables in length. The first syllable of each operator indicates the number of additional syllables forming the operator.

Operators work on data as either full words (48 data bits plus 3 tag bits) or as strings of data characters. Word operators work with operands (single-or double-precision) in the top of the stack.

String operators are used for transferring, comparing, scanning, and translating strings of digits, characters, or bytes. In addition, a set of micro-operators (EDIT Mode operators) provides a means of formatting data for input/output. String operators and edit mode operators use source and destination pointers located in the stack to set hardware registers.

In some of the string operators the source pointer may not be used. In this case, an operand may be in the stack; its characters are circulated as it is being used. String operators have an optional update function, producing updated source and destination pointers and counts.

If both the source and destination descriptors have size fields equal to zero, the size registers indicate 8-bit character size. When both a source and destination are required and the size field of one is equal to zero and the other is not, then the size field of the non-zero descriptor is used.

If neither size field is equal to zero and the size fields are not equal and the operator is not Translate or Transfer Words, the invalid-operand interrupt is set and the operator is terminated. The size field is considered equal to zero when the source is an operand.

In the B 7700 Systems operands may be used to represent either numeric or logical information. An operand may be a single-precision (SP) operand or double-precision (DP) operand. Memory word tag bits (bits 50, 49, and 48), when 000, designate an SP operand and, when 010, designate a DP operand.

Logical operands may be either true (ON) or false (OFF). Logical values are the result of Boolean operations or relational operations. Relation operators generate a logical value as the result of an algebraic comparison of two arithmetic expressions. Bit 0 contains the logical value. Relational operators set bit 0, and conditional operators use bit 0 for the decision. Logical (Boolean) operators consider each bit from 47 to 0 as an individual logical value and operate on the whole operand.

GROUPING OF OPERATORS

Operators may be identified by name, mnemonic, or hexadecimal code. In this document to facilitate reference to the description of the operators, the operators are listed in the appendix in two ways: alphabetically by mnemonic, and sequentially by hexadecimal code.

In each case the page number of the operator description is given.

When describing operators, considerable redundancy is eliminated by grouping operators with similar functions and only describing their differences. Also, for convenience of the user, operators used for related manipulations (e.g., arithmetic operators i.e. ADD, SUBT, MULT, DIVD, etc.) are described sequentially.

As shown in Figure III-3-3 all central processor program operators are grouped into one of four modes: primary (P), variant (V), edit (E), or vector (Z). Several operators are classed as universal (U) because they can operate in any mode. (The letters in the above parentheses are used in this document as a mode-identifier prefix before the hexadecimal code associated with each operator; e.g., (P) 80 indicates a primary mode operator and 80 is the hexadecimal code for the ADD operator.) In this document, the operator descriptions are grouped by mode and preceding each group of descriptions for each mode there is a listing giving the order of specific operator descriptions.

PRIMARY MODE (P) xx

1st ↓ 2nd →	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0-3	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	0-3
4-7	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	4-7
8	ADD	SUBT	MULT	DIVD	IDIV	RDIV	NTIA	NTGR	LESS	GREQ	GRTR	LSEQ	EQUL	NEQL	CHSN	MULX	8
9	LAND	LOR	LNOT	LEQV	SAME	VARI	BSET	DBST	FLTR	DFTR	ISOL	DISO	INSR	DINS	BRST	DBRS	9
A	BRFL	BRTR	BRUN	EXIT	STBR	NXLN	INDX	RETN	DBFL	DBTR	DBUN	ENTR	EVAL	NXLV	MKST	STFF	A
B	ZERO	ONE	LT8	LT16	PUSH	DLET	EXCH	DUPL	STOD	STON	OVRD	OVRN	LODT	LOAD	LT48	MPCW	B
C	SCLF	DSLF	SCRT	DSRT	SCRS	DSRS	SCRF	DSRF	SCRR	DSRR	ICVD	ICVU	SNGT	SNGL	XTND	IMKS	C
D	TEED	PACD	EXSD	TWSD	TWOD	SISO	SXSN	ROFF	TEEU	PACU	EXSU	TWSU	TWOU	EXPU	RTFF	HALT	D
E	TLSD	TGED	TGTD	TLED	TEQD	TNED	TUND	VMOM	TLSU	TGEU	TGTU	TLEU	TEQU	TNEU	TUNU	VMOS	E
F	CLSD	CGED	CGTD	CLED	CEQD	CNED		FMMR	CLSU	CGEU	CGTU	CLEU	CEQU	CNEU	NOOP	NVLD	F

VARIANT MODE (V)xx

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
4			JOIN	SPLT	IDLE	SINT	EEXI	DEXI	IGPR		SCNI			PTPA	WHOI		4
8					PAUS	OCRX		NTGD	MIN		MAX	LOG2		DADD		INCN	8
A		UPKL				PAKL	RODI	PAKR	SINH	UPKR	SLMT	SZTN	FMFR		ROD2	MVST	A
B					STAG	RTAG	RSUP	RSDN	RPRR	SPRR	RDLK	CBON	LODT	LLLU	SRCH	STOP	B
D	USND	UABD	TWFD	TWTD	SWFD	SWTD		TRNS	USNU	UABU	TWFU	TWTU	SWFU	SWTU	RDEF	HALT	D
E																	E
F	SLSD	SGED	SGTD	SLED	SEQD	SNED			SLSU	SGEU	SGTU	SLEU	SEQU	SNEU	NOOP	NLVD	F

EDIT MODE (E) xx

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
D	MINS	MFLT	SFSC	SRSC	RSTF	ENDF	MVNU	MCHR	INOP	INSG	SFDC	SRDC	INSU	INSC	ENDE	HALT	D
F							SSSZ								NOOP	NLVD	F

VECTOR MODE (Z) xx

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0-3	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	0-3
4-7	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	4-7
E	LDA	LDAI	LDB	LDBI	LDC	LDCI	VXIT	VMOM	DLA	DLAI	DLB	DLBI	DLC	DLCI	VEBR	VMOS	E
F	STA	STAI	STB	STBI	STC	STCI			DSA	DSAI	DSB	DSBI	DSC	DSCI	NOOP	NVLD	F

Figure III-3-3. B 7700 CPM Program Operator Hexadecimal Code Assignments

The most frequently used operators are called primary mode operators. Each of the other modes is entered by first executing certain operators in primary mode. The "operator" portion of the primary mode operators begins with the first syllable and may extend for several syllables.

Primary mode operators are described in this document in the following groups: arithmetic, bit, branch, compare, enter edit mode, enter vector mode, index and load, input convert, literal call, logical, pack, relational, scale, stack, store, string, string transfer, subroutine, transfer, type-transfer, miscellaneous, and universal. (In several cases a variant mode operator is conveniently described with a group of primary mode operators.)

Variant mode operators are less frequently used than primary mode operators and extend the number of hexadecimal codes available to identify the operators. Variant mode operators require two syllables. The first syllable of a variant mode operator has the hexadecimal code 95 which is the primary mode operator called Escape to 16-Bit Instruction (the mnemonic for this operator is VARI). The second syllable then gives the actual variant mode operation to be performed. The variant mode operators are described in this document in the following groups: scan, scan while, tag field, unpack, miscellaneous, operators exclusive to the B 7700, and universal operators.

Edit mode operators perform edit functions (such as insert, move, and skip) on strings of data being prepared for output. The Edit mode is entered from the primary mode via one of the enter edit operators (EXSD, EXSU, EXPU, TEED, or TEEU). Subsequent edit operators follow as either single micro operators in the program string or as edit operators in a separate table which is executed as a program string. In edit mode the program buffer memory is reduced to 16 words (total available area) for processing the edit operators; the other 16 words contain the primary program syllables.

The basic B 7700 architecture avoids index registers in order to facilitate block or procedure entries. Although this improves the machine's overall performance, it does impede processes of an iterative nature such as the ordinary handling of arrays. The B 7700 overcomes this difficulty with vector mode operation (a variation of string operator edit mode) which permits successive accesses to the elements of an array by using the source, destination, and table pointer areas of the IC memory as index registers and by improved loop control. Vector mode hardware provides additional register capabilities and permits operators to be generated by the compilers to effect improved handling of vectors.

Vector mode is entered from primary mode by using either of two operators: a Single Word Vector Mode (VMOS) entry or Multiple-Word Vector Mode (VMOM) entry. The two enter vector operators assign addresses to the index registers and perform either a VMOS or a VMOM operation. Single-Word Vector Mode forces an automatic one-word branch backward while the processor is in vector mode. The Multiple-Word Vector Mode uses the automatic local branch point detection. The operators Vector Branch and Vector Exit are used only in the Multiple-Word Vector Mode and provide control of program iterations and exiting. Twenty-four vector stack operators link the top of stack with the word addressed by a specific IC register, thus enabling direct, indexable transfers between memory and the top of stack. Forty operators are permitted for vector and matrix manipulations.

In vector mode certain limitations must be considered; for example, the processor is in control state and cannot be interrupted to service other needs. The arrays manipulated by vector mode cannot be segmented and must be present in their entirety while in vector mode. Therefore, the use of vector mode in a general multiprocessing environment must necessarily be restricted; nevertheless, it provides a powerful tool for a particular class of problems.

Detection of an invalid operator condition terminates the operator, and an invalid operator interrupt is set in the processor interrupt register. The processor will proceed to process the interrupt whether it is in normal state or control state.

Invalid instructions are detected by the following methods:

1. Testing for unassigned operator codes. In the B 7700 all unassigned operators cause a programmed operator interrupt.

2. Testing for any value other than 011 in bit positions 50, 49, and 48 of any program word (an attempt to execute something which is not code). This results in an invalid program word interrupt except when in table mode which allows a tag 0 or a tag 3.

3. Testing for an invalid operator function; for example, an attempt to dial to a non-existent bit. This results in an invalid operand interrupt.

Bit 48 of each word in main memory is a memory protect bit. This bit is ON in all program words, indirect reference words, data descriptors, program descriptors, main memory storage links, and processor-generated control words.

Except for stack pushdowns and the overwrite operators an attempt by a processor to write into a location when the contents of that location has the memory protect bit set will cause a memory-protect interrupt to be set in the processor interrupt register. The overwrite operators will overwrite whatever is in the addressed area. When the string or edit operators attempt to access the source or destination areas they will get a segment array interrupt but when they attempt to access a table they will get a memory-protect interrupt.

PRIMARY MODE OPERATORS

Primary mode operators may consist of as many as seven syllables but the first syllable defines the operation.

ARITHMETIC OPERATORS

Dyadic arithmetic operators require two operands in the top-of-stack storage. These operands are combined by the arithmetic process specified and are replaced with the resulting operand. Both operands may be either single-precision, double-precision, or intermixed types. The specified arithmetic process adapts automatically to the environment: a single-precision process is invoked if both operands are of the single-precision type and a double-precision process is invoked if either operand is of the double-precision type. Each double-precision operand occupies two words. The second word of the operand is an extension of the first word of the operand, i.e., the mantissa of the first word of the operand may be an integer but the mantissa of the second word is always a fraction.

Add, subtract, multiply, and integer divide operations with two integer operands yield an integer result if no overflow occurs. If one or both operands are noninteger or if the result overflows, the result is noninteger.

ADD (ADD) (P)80

The Add operator causes the two top-of-stack operands to be added algebraically and the sum to be left in the top-of-stack.

SUBTRACT (SUBT) (P)81

The Subtract operator causes the top-of-stack operand to be algebraically subtracted from the second operand in the stack and the result to be left in the top-of-stack.

MULTIPLY (MULT) (P)82

The Multiply operator causes the two top-of-stack operands to be algebraically multiplied and the product to be left in the top-of-stack.

EXTENDED MULTIPLY (MULX) (P)8F

The Extended Multiply operator causes the two top-of-stack operands to be algebraically multiplied and a double-precision product to be left in the top-of-stack.

DIVIDE (DIVD) (P)83

The Divide operator causes the second operand in the stack to be algebraically divided by the top-of-stack operand the quotient to be left in the top-of-stack. If the mantissa of the second operand in the stack is zero, the exponent and quotient are set to zero. If the top-of-stack mantissa is zero, the divide-by-zero interrupt is set. In either case the operation is terminated.

INTEGER DIVIDE (IDIV) (P)84

The Integer Divide operator causes the second operand in the stack to be algebraically divided by the top-of-stack operand and the integer part of the quotient to be left in the top-of-stack in integer form. If the mantissa of the second operand in the stack is zero, the exponent and quotient are set to zero. If the top-of-stack mantissa is zero, the divide-by-zero interrupt is set. In either case the operation is terminated.

REMAINDER DIVIDE (RDIV) (P)85

The Remainder Divide operator causes the second operand in the stack to be algebraically divided by the top-of-stack operand to develop an integer quotient. The remainder of this division is left in the top-of-stack. If this remainder is an integral value, it is in the form of an integer. If the mantissa of the second operand in the stack is zero, the exponent and quotient are set to zero. If the top-of-stack mantissa is zero, the divide-by-zero interrupt is set. In either case the operation is terminated.

INTEGERIZE, TRUNCATED (NTIA) (P)86

The Integerize (Truncated) operator converts the top-of-stack operand to an integer without rounding. If the operand cannot be integerized, i.e., the exponent is greater than the number of leading zeros in the operand. The integer-overflow interrupt is set and the operation is terminated.

INTEGERIZE, ROUNDED (NTGR) (P)87

The Integerize (Rounded) operator converts the top-of-stack operand to an integer with rounding. Rounding takes place if the absolute value of the fraction is greater than 4. If the operand cannot be integerized, i.e., the exponent is greater than the number of leading zeros in the operand or a non-integer results from the rounding operation, the integer-overflow interrupt is set and the operation is terminated.

INTEGERIZE ROUNDED, DOUBLE PRECISION (NTGD) (V)87

The Integerize (Rounded, Double Precision) operator converts the top-of-stack operand to a double-precision integer (exponent +13) with rounding.

BIT OPERATORS

Bit operators set or reset bits in the top-of-stack or in the second item in the stack.

BIT SET (BSET) (P)96

The Bit Set operator sets a bit in the top-of-stack. The bit set corresponds to the value of the bit specified by the second syllable of the operator. If the program syllable defining the bit to be set has a value greater than 47, the invalid-operand interrupt is set and the operation is terminated.

DYNAMIC BIT SET (DBST) (P)97

The Dynamic Bit set operator sets a bit in the second item in the stack. The bit set corresponds to the value of the bit specified by the top-of-stack operand. If the word in the top-of-stack is not an operand an invalid-operand interrupt is set and the operation is terminated. The word is integerized before it is used as a bit number. If after being integerized the operand is less than zero or greater than 47, an invalid-operand interrupt is set and the operation is terminated.

BIT RESET (BRST) (P)9E

The Bit Reset operator resets a bit in the top-of-stack. The bit reset corresponds to the bit specified by the second syllable of the program operator. If the program syllable defining the bit to be reset has a value greater than 47, an invalid-operand interrupt is set and the operation is terminated.

DYNAMIC BIT RESET (DBRS) (P)9F

The Dynamic Bit Reset operator resets a bit in the second item in the stack. The reset bit corresponds to the value of the bit specified by the top-of-stack operand. If the word in the top-of-stack is not an operand an invalid-operand interrupt is set and the operation is terminated. The word is integerized before it is used as a bit number. If, after being integerized, the operand is less than zero or greater than 47, an invalid-operand interrupt is set and the operation is terminated.

CHANGE SIGN BIT (CHSN) (P)8E

The Change Sign Bit operator complements (changes from 1 to 0 or from 0 to 1) the sign bit (bit 46) of the top-of-stack operand.

COUNT BINARY ONE'S (CBON) (V)BB

The Count Binary One's operator counts the number of binary ones in the information part of the word in the top-of-stack and places this count in the top-of-stack.

LEADING ONE TEST (LOG2) (V)8B

The Leading One Test operator locates the most significant information bit of the word in the top-of-stack. The number of that bit plus one is placed in the top-of-stack. If a one bit is not located, a zero is placed in the top-of-stack.

BRANCH OPERATORS

Branch instructions function to break the normal sequence of serial instruction fetches. Branching may be either relative to the base address of the current program segment or to a location in some other program segment. Branch operators may be conditional or unconditional. Branch addresses are always checked for possible residency in the local program buffer.

BRANCH UNCONDITIONAL (BRUN) (P)A2

The Branch Unconditional operator replaces the contents of the program index register (PIR) and the program syllable register (PSR) with the next two syllables from the program string. The two syllables following the actual operator syllable provide the new PIR and PSR settings: the three high-order bits are placed in the PSR and the next 13 low-order bits are placed in the PIR.

BRANCH ON TRUE (BRTR) (P)A1

If the low-order bit of the top-of-stack word is a one, the Branch on True operator replaces the contents of the program index register and the program syllable register with the next two syllable positions and the program continues in sequence. Otherwise, the PIR and PSR are advanced three syllable positions and the program string continues in sequence.

BRANCH ON FALSE (BRFL) (P)A0

If the low-order bit of the top-of-stack word is a zero, the Branch on False operator replaces the contents of the program index register and the program syllable register with the next two syllables from the program string. Otherwise, PIR and PSR are advanced three syllable positions and the program string continues in sequence.

DYNAMIC BRANCH UNCONDITIONAL (DBUN) (P)AA

If the top-of-stack word is either a program control word or an indirect reference to a PCW, the Dynamic Branch Unconditional operator branches to the specified syllable of the program segment.

If the top-of-stack word is an operand, the program index register and program syllable register are set according to the contents of this operand as follows: The operand is made into an integer. If it is negative or if it is greater than 16384 the invalid-index interrupt is set and the operation is terminated. If bit zero of the operand is zero, PSR is set to zero; otherwise, if bit zero of the operand is one, PSR is set to three. The next higher-order 13 bits are placed in the PIR.

DYNAMIC BRANCH TRUE (DBTR) (P)A9

If the low-order bit of the second word in the stack is a one and the top-of-stack word is a program control word (PCW) or an indirect reference to a PCW, the Dynamic Branch True operator will cause a branch to the specified syllable in the program segment. Otherwise, a one is added to the PIR and PSR and the program continues in sequence.

If the low-order bit of the second word in the stack is a one and the top-of-stack word is an operand, PIR/PSR are replaced from this operand as in the DBUN operator. Otherwise, PIR and PSR are advanced and the program string continues in sequence.

DYNAMIC BRANCH FALSE (DBFL) (P)A8

If the low-order bit of the second word in the stack is a zero, and the top-of-stack word is a program control word or an indirect reference to a PCW, the Dynamic Branch False operator causes a branch to the specified syllable of the program segment. Otherwise, the PIR/PSR are continued in sequence.

If the low-order bit of the second word in the stack is a zero and the top-of-stack word is an operand, PIR/PSR are replaced from this operand as in the DBUN operator. Otherwise, PIR and PSR are advanced and the program string is continued in sequence.

STEP AND BRANCH (STBR) (P)A4

The Step and Branch operator is initiated with a reference to either a step index word (SIW) or an operand in the top-of-stack. The target item may be reached through a chain of indirect reference words, indexed data descriptors, and/or accidental entries. The format of the SIW is shown in figure II-3-4.

If the target item is an SIW, the increment field of the SIW is added to the current-value field of the SIW, and the SIW is replaced in memory. If the current-value field after adding the increment is less than or equal to the final-value field, then PIR and PSR are advanced three syllable positions, the program string is continued in sequence, and an operand representing a Boolean value "true" (bit 0 ON) is left in the top-of-stack. If the current-value field is greater than the final-value field, the program takes the branch by replacing PIR and PSR with the next two syllables from the program string. If the branch is taken, no Boolean is left in the top-of-stack.

If the target item is an operand, the operand will be left in the second stack position, the top-of-stack will be set to zero, representing a Boolean value "false," and PIR and PSR will be advanced to the next operator.

If the target item is other than an operand or SIW, the invalid-operand interrupt is set.

COMPARE OPERATORS

The compare operators perform the specified compare of two strings of data. The tru/false flip-flop is conditioned by the results of the compare.

COMPARE CHARACTERS GREATER, DESTRUCTIVE (CGTD) (P)F2

The Compare Characters Greater Destructive operator makes a character-by-character comparison of two strings of data until it finds an unequal pair. (All comparisons are by the binary character position in the collating se-

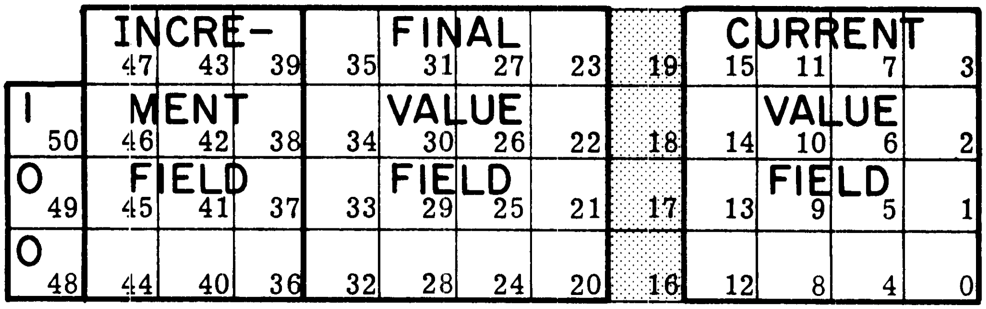


Figure III-3-4. Step Index Word Format

quence.) If the characters in the B string (destination) are greater than the characters in the A string (source), then the true/false flip-flop is set to one; otherwise, the true/false flip-flop is set to zero. If the repeat count is less than or equal to zero, the true/false flip-flop is reset.

The top-of-stack is an operand which specifies the number of characters to be compared. The second item in the stack is an operand or descriptor pointing at the source character string against which comparisons are to be made. The third item in the stack is a descriptor pointing to the character string to be compared. If either of the data strings has the memory protect bit ON (bit 48=1), the segmented array interrupt is set, and the operation is terminated.

COMPARE CHARACTERS GREATER, UPDATE (CGTU) (P)FA

The Compare Characters Greater, Update operator performs a Compare Characters Greater, Destruction operation except that the accesses to memory continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE (CGED) (P)F1

The Compare Characters Greater or Equal, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the destination is greater than or equal to the source.

COMPARE CHARACTERS GREATER OR EQUAL, UPDATE (CGEU) (P)F9

The Compare Characters Greater or Equal, Update operator performs a Compare Characters Greater or Equal, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS EQUAL, DESTRUCTIVE (CEQD) (P)F4

The Compare Characters Equal, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the source is equal to the destination.

COMPARE CHARACTERS EQUAL, UPDATE (CEQU) (P)FC

The Compare Characters Equal, Update operator performs a Compare Characters Equal, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE (CLED) (P)F3

The Compare Characters Less or Equal, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the destination is less than or equal to the source.

COMPARE CHARACTERS LESS OR EQUAL, UPDATE (CLEU) (P)FB

The Compare Characters Less or Equal, Update operator performs a Compare Less or Equal, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS LESS, DESTRUCTIVE (CLSD) (P)F0

The Compare Characters Less, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the destination is less than the source.

COMPARE CHARACTERS LESS, UPDATE (CLSU) (P)F8

The Compare Characters Less, Update operator performs a Compare Characters Less, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE (CNED) (P)F5

The Compare Characters Not Equal, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the source is not equal to the destination.

COMPARE CHARACTERS NOT EQUAL, UPDATE (CNEU) (P)FD

The Compare Characters Not Equal, Update operator performs a Compare Characters Not Equal, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

ENTER EDIT MODE OPERATORS

Enter edit mode operators provide the means for transition from primary mode operation to edit mode operation. The edit mode operators in a program string are entered via the Execute Single Micro or Single Pointer.

The edit mode operators may also be in a table and in which case they are entered by the Table Enter Edit operator. (See also the descriptions under "Edit Mode Operators.")

TABLE ENTER EDIT, DESTRUCTIVE (TEED) (P)D0

The Table Enter Edit, Destructive operator is used to control edit micro instructions which are contained in memory as a table rather than as part of the normal program string. This operator causes characters to be transferred from the source string to the destina-

The transfer is under control of the string of edit micro-operators which are located by the table pointer.

The top-of-stack word (a descriptor) is the table pointer, the second word (a single-precision operand or descriptor) in the stack is the source pointer, and the third word in the stack (a descriptor) is the destination pointer. If the first word in the stack is not a descriptor, the invalid-operand interrupt is set and the operation is terminated. If the second item in the stack is a single-precision operand, it is a source string. If the third item in the stack is not a descriptor, the invalid-operand interrupt is set and the operation is terminated. In table mode, the micro-operator words can be tagged as single-word operands (tag-0).

TABLE ENTER EDIT, UPDATE (TEEU) (P)D8

The Table Enter Edit, Update operator performs a Table Enter Edit Destructive operation. At the completion of the operation, the source pointer and destination pointer are updated.

EXECUTE SINGLE MICRO, DESTRUCTIVE (EXSD) (P)D2

The Execute Single Micro, Destructive operator transfers characters from the source string to the destination string under the control of the single micro-operator which follows this operator syllable. The first item in the stack is a single-precision operand that defines the field length and is used as a micro-operator repeat field. The second item in the stack is the source pointer, the third item in the stack is the destination pointer.

EXECUTE SINGLE MICRO, UPDATE (EXSU) (P)DA

The Execute Single Micro, Update operator performs an Execute Single Micro, Destructive operation. At the completion of the operation, the source pointer and destination pointer are updated.

EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE (EXPU) (P)DD

The Execute Single Micro, Single Pointer Update operator performs an Execute Single Micro, Destructive operation. At the completion of the operation, the pointer is updated.

The top-of-stack operand is used as a micro-operator repeat field. The second item in the stack is used to set both the source and destination pointers. Only the destination pointer is updated.

ENTER VECTOR MODE OPERATORS

The enter vector mode operators provide the means of transition from primary mode to vector mode. Either one of two operators are available to enter vector mode: for operations using instructions in only one program word, operator VMOS is used; for operations involving instructions in more than one program word, operator VMOM is used. (See also the descriptions under "Vector Mode Operators.")

Vector mode hardware provides increased efficiency in the ordinary handling of arrays that frequently dominates a FORTRAN or an ALGOL program. For example, when processing the following FORTRAN DO loop:

```
DO 10 I = 1, 100
A(I) = B(I) + C(I)
10 CONTINUE
```

Each trip through the loop (for each value of I) requires a descriptor for each of the three arrays and the value of I with which to index each descriptor. This means six memory accesses, in addition to code fetching and execution.

To example above (the FORTRAN D0 loop) can be expressed in ESPOL as follows:

```
D0 VECTORMODE ( [ 1,1,1 ]
 A [ * ] ,B [ * ] ,C [ * ] , for 100)

 BEGIN

 A=B+C;

 INCREMENT A,B,C;

 END;
```

The example has specified three increment values, three beginning addresses, a length (or number of iterations), and the operations to be performed on the array elements in each iteration. From this information the compiler generates the following:

1. Primary mode code to place the seven parameters in the stack.
2. The enter vector mode operator.
3. Vector mode code to perform the operations on the array elements.

Before entering vector mode, the values to be stored in the IC memory registers must be placed in the stack. They are arranged in the stack in the following order (from the top-of-stack down):

Pointer C	(descriptor)
Length (optional)	(operand)
Pointer A	(descriptor)
Pointer B	(descriptor)
Pointer C increment	(operand)
Pointer A increment	(operand)
Pointer B increment	(operand)

The seven parameters are inserted in the local buffer as follows:

TBR(33)←——Pointer C [19:20] (+ Pointer C [35:16] if I = 1)
S2LS(37)←——LENGTH [19:20] (or 2^{20}-1)
SBR(31)←——Pointer A [19:20] (+ Pointer A [39:20] if I = 1)
DBR(32)←——Pointer B [19:20] (+ Pointer B [39:20] if I = 1)
TIR(28)←——Pointer C increment [19:20]
SIR(21)←——Pointer A increment [19:20]
DIR(22)←——Pointer B increment [19:20]

I is the indexed bit, bit 45, in the descriptor. (See figure II-3-5.)

The enter vector mode operator may be terminated by one of the following interrupts:

a. INVALID OP: Pointer A, B or C not tagged as a data descriptor or Pointer A or B has bit 44=1.

b. MEMORY PROTECT: Pointer A is read only (bit 43=1).

c. PRESENCE BIT: Pointer A, B or C has bit 47=0.

Length specifies the number of iterations through the code to be executed while in vector mode, usually the number of elements in the arrays being manipulated. The presence of a length value in the stack is indicated by bit 44=1 in Pointer C. If bit 44=0, a default length of 2^{20}–1 is stored in the length register. Bit 44 (segmented bit) must be OFF in Pointer A and Pointer B. (The software ascertains that bit 44 is OFF in Pointer C before using it to indicate the presence of a length value.)

If the entry to vector mode is the single-word vector mode (VMOS) operator, the single word of code following that entry is executed a number of times equal to the length parameter. Each time the word is executed length is decremented by one until it becomes zero. Then vector mode is exited and normal operation continues with the next word of code in sequence. It should be noted that in the vector mode hardware, the VMOM operator subtracts 1 from the length and then compares, whereas the multiple-word (VOMS) operator first compares the length and then subtracts.

When the entry to vector mode is the (VMOM) operator, any code that follows it is executed under vector mode rules. The vector mode operators explained below are used only in conjunction with the VMOM operator.

a. Vector Branch (VEBR) is a three-syllable operator. The two syllables following the operator name contain the branch address. The Vector Branch operator examines length. If it is greater than zero, length is decremented by one, the next two program syllables containing the branch address are skipped, and the program is resumed at the following syllable. If the examined length is zero, vector mode is exited, and normal operation commences with the program word located by the branch address.

b. Vector Exit (VXIT) operator causes the program to return to normal operation.

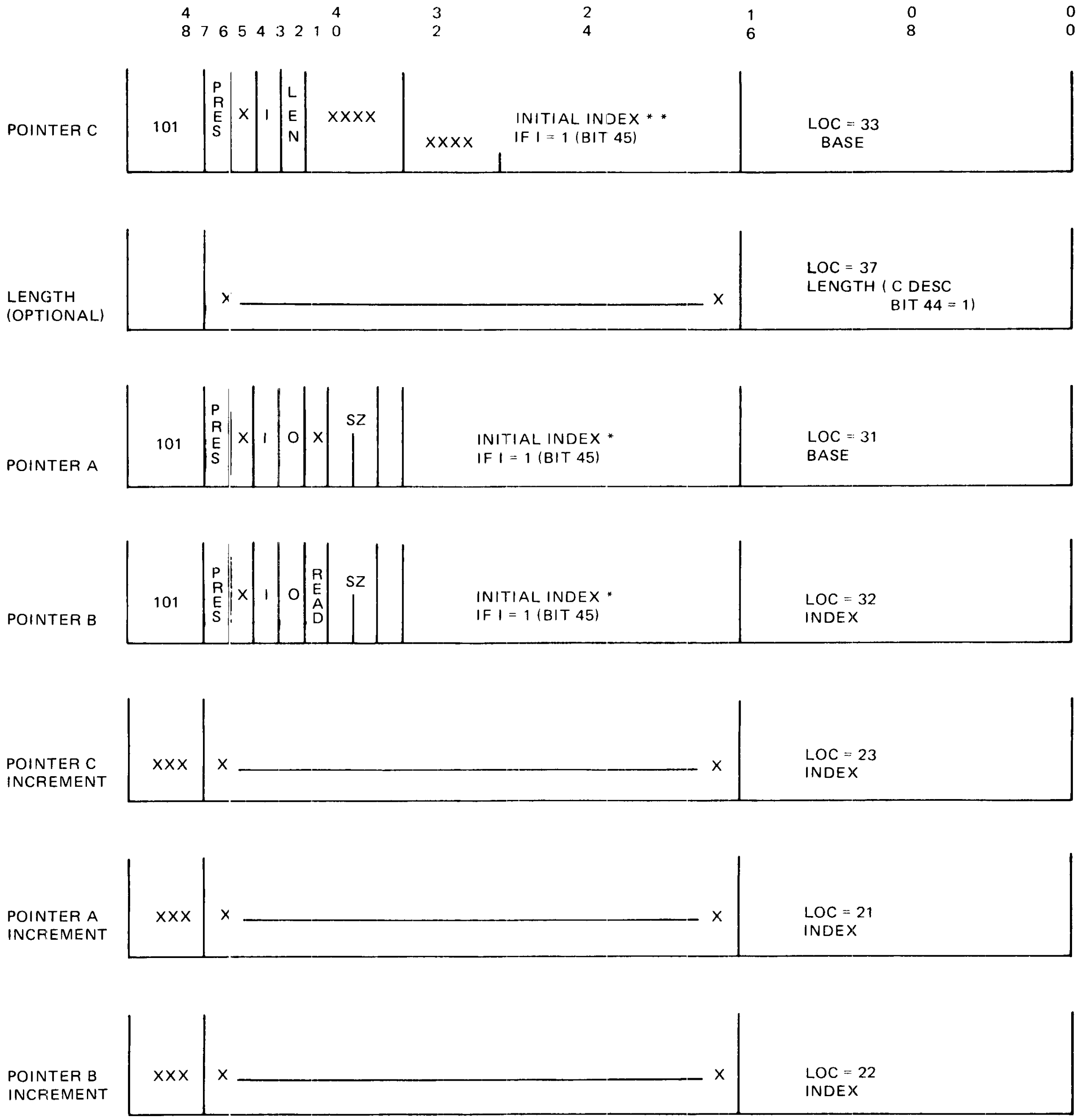
There are 24 Vector Stack operators (with a common syllable format) which are used to move operands between the top-of-stack and absolute memory addresses pointed to by descriptors. Variations of this syllable provide the capabilities of storing or loading the top-of-stack with a single – or double-precision operand and choosing whether or not to increment the pointer. If the memory address is protected, the following recovery procedure is followed:

a. If a store operation, vector mode is terminated with a Memory Protect Interrupt.

b. If a load operation, then:

(1) If the length parameter was passed to the vector mode, vector mode is terminated with a Memory Protect Interrupt.

(2) If no length parameter was passed to the vector mode, vector mode is terminated but no interrupt is set. The stack



** IF $\overline{45}$ THEN BASE → 33
IF $\overline{45}$ THEN BASE + C [35:16] → 33

* IF $\overline{45}$ THEN BASE → 31 or 32
I = 45 AND
SZ = 0 THEN BASE + [39:20] → 31 or 32
or SZ ≠ 0 THEN BASE + [35:16] → 31 or 32

REAↃ (POINTER B ONLY) MUST BE 0 ELSE MEMORY PROTECT

Figure III-3-5. Vector Table

is then cut back as defined by the operator that sensed the Memory Protect condition (refer to RA and RB below).

The word for which a protect is sensed is not marked *present* in the stack.

The format of the vector operator syllable is defined as follows:

a. If a length is not passed when vector mode is entered, the format is as follows:

0	LS	RA	RB	D	A1	A0	I

where 0, the high-order bit, must be OFF (0).

b. When a length is passed, the format is as follows;

1	1	1	LS	D	A1	A0	I

where the three high-order bits must be ON (1).

In either format:

Bit **Description**

D Double-precision bit. If D=0, load or store a single-precision operand. If D=1, load or store a double-precision operand.

RA If a memory protect interrupt is sensed and no length is passed to the vector mode and RA=0, the top-of-stack word is deleted. If RA=1, the top-of-stack word is not deleted.

RB Same as the RA bit except that it governs the action taken on the second word of the stack.

LS Bit is OFF (0) for a top-of-stack load operator and ON (1) for a top-of-stack store operator.

A1, A0 Selects the IC memory address register.

A1	A0	
0	0	Load from Pointer A
0	1	Load from Pointer B
1	0	Load from Pointer C

I When I equals 1, the pointer used for the memory address is increased by its corresponding pointer increment following the load or store operator. When I equals 0, the pointer increment is inhibited.

The Vector Stack operators are described under "Vector Mode Operators."

Two other operators (FTCH and STOR) are used to load/store the top-of-stack from/to relative memory addresses designated by an address couple. They are enabled only when a length is passed by the vector mode entry. (The operators FTCH and STOR are described under "Vector Mode Operators.")

A memory protect interrupt occurs during Vector Mode if a protected word (bit 48 set) is sensed as a result of processing VALC, NAME, or stack vector operator.

External interrupts are disabled during Vector Mode. Exponent underflow interrupts are inhibited for arthmetic operators, and in lieu of the interrupt, an answer of zero is returned and the TFFF is turned ON.

No facilities are provided for recovery from interrupts that occur while in vector mode. P1 is returned with bit 19=1.

SINGLE-WORD VECTOR MODE (VMOS) (P)EF

The Single-Word Vector Mode operator is the primary mode operator used to access vector mode to perform on a vector those operations defined by one program word. VMOS extracts the seven parameters (described above) from the stack, inserts them in their IC address, and, after skipping up to five syllables, repeatedly executes the next complete word of program code from the program register. VMOS is inhibited from accessing additional program code, thus causing vector mode exit.

If the descriptor in the top-of-stack has bit 44 ON, then the second item in the stack is the length operand and it gives the iteration repeat count; otherwise, the default repeat count of 1,048,575 is used.

Each of the three descriptors (pointers) represents a full or partial array of operands which will be operated on repeatedly by the same word of code.

MULTIPLE-WORD VECTOR MODE (VMOM) (P)E7

The Multiple-Word Vector Mode operator provides access to the vector mode operators for multiple program words. The VMOM operator performs a VMOS operation supplemented by operator Vecor Branch which leads to additional program word loops. The special vector exit operator VXIT returns the program to normal operation.

INDEX AND LOAD OPERATORS

The index and load operators provide the means to index the top-of-stack word and the means to load an operand or descriptor into the top-of-stack.

INDEX (INDX) (P)A6

The two top-of-stack items are a descriptor (or indirect reference to a descriptor) and an operand. The operand is used to index the descriptor. The Index operator places the integerized value of the second item in the stack into the 20-bit length/index field of the descriptor in the top-of-stack. The descriptor is marked indexed (i.e., bit 45 is set to "one").

If the word in the top-of-stack is an operand, the top-of-stack operand is exchanged with the second-item operand. If the word in the top-of-stack is neither a descriptor nor an indirect reference word pointing to a descriptor, the invalid-operand interrupt is set and the operation is terminated.

If the indexing value is negative or greater than or equal to the length field of the descriptor, the invalid-index interrupt is set and the operation is terminated.

If the descriptor represents an array which is segmented, the index is partitioned into two portions by dividing it by the proper divisor determined by the type of data referenced by the descriptor, (D.P. word-128, S.P. word-256, 4-bit digit-3072, 6-bit character-2048, or 8-bit byte-1536). The quotient is used as an index to the given descriptor to fetch the array-row descriptor. The remainder is used to index the row descriptor.

If the double-precision bit (bit 40) in the descriptor is "one", the index value in the second item is doubled. The balance of the operation is as described in the first paragraph of this operator.

INDEX AND LOAD NAME (NXLN) (P)A5

The Index and Load Name operator performs an Index operation. After the word in the top-of-stack is indexed, the data descriptor pointed to by this word is brought to the top-of-stack, the copy bit (bit 46) of the data descriptor is set to "one", and the top-of-stack is marked full.

If the presence bit (bit 47) is OFF, the address of the original descriptor is placed in the address field of the stack copy. If the word accessed by the indexed word in the top-of-stack is not a data descriptor, the invalid-operand interrupt is set and the operation is terminated. If the data descriptor accessed by the indexed word in the top-of-stack has the index bit (bit 45) set to "one", the invalid-operand interrupt is set and the operation is terminated.

INDEX AND LOAD VALUE (NXLV) (P)AD

The Index and Load Value operator performs an Index operation. After the word in the top-of-stack is indexed, the operand pointed to by this descriptor is brought to the top-of-stack. The top-of-stack is marked full.

If the word accessed is other than an operand the invalid-operand interrupt is set and the operator is terminated.

LOAD (LOAD) (P)BD

The Load operator places the word addressed by the indirect reference word or by the indexed data descriptor in the top-of-stack.

If at the start of this operator the top-of-stack contains other than a data descriptor or an indirect reference word pointing at a data descriptor, the invalid-operand interrupt is set and the operation is terminated.

If the word pointed at by the data descriptor is another data descriptor, that descriptor is marked as a copy (copy bit [**bit 46**[is set to "one") and if the presence bit (bit 47) is OFF, the address of the original data descriptor is placed in the field defined by bits 19:20 of the copy in the stack.

LOAD TRANSPARENT (LODT) (V)BC

If the top-of-stack word is a data descriptor or an indirect reference word, the Load Transparent operator performs a Load operation; otherwise, the word addressed by the 20 least-significant bits of the top-of-stack word is loaded to the top-of-stack. Copy-bit action does not occur.

INPUT CONVERT OPERATORS

The input convert operators convert the various character sets (digit, BLC, EBCDIC, or ASCII) to operands for arithmetic operations.

INPUT CONVERT, DESTRUCTIVE (ICVD) (P)CA

The Input Convert, Destructive operator converts 4-bit digit, or 6-bit BCL, or 8-bit EBCDIC (or ASCII) to an operand for internal arithmetic operations.

The first item in the stack is an operand that is integerized to form the repeat field. The second item in the stack is a descriptor used as a source pointer.

The specified number of characters are transferred from the source string to the top-of-stack. Only the numeric portion of the character is transferred. The transferred string is converted to a double-precision operand if the length is greater than 12. If a double-precision operand is produced, the true/false flip-flop is set to false; otherwise, it is set to true. The sign bit of the operand is set negative if the zone of the last character transferred is 10_2 (for six-bit characters) or 1101_2 (for eight-bi characters). At the completion of the operator the second item in the stack is marked full. The tag field is set to indicate a single – or double-precision operand.

INPUT, CONVERT, UPDATE (ICVU) (P)CB

The Input Convert, Update operator performs an Input Convert, Destructive operation. At the completion of the operation, the source pointer is updated.

LITERAL CALL OPERATORS

The literal call operators place defined-value operands in the top-of-stack.

LIT CALL ZERO (ZERO) (P)B0

The Lit Call Zero operator places in the top-of-stack a single-precision operand with a value of zero.

LIT CALL ONE (ONE) (P)B1

The Lit Call One operator places in the top-of-stack a single-precision operand with a value of one.

LIT CALL 8 BITS (LT8) (P)B2

The Lit Call 8 Bits operator places in the top-of-stack a single-precision operand equal in value to the second syllable of this operator.

LIT CALL 16 BITS (LT16) (P)B3

The Lit Call 16 Bits operator places in the top-of-stack a single-precision operand equal in value to the second and third syllables of this operator.

LIT CALL 48 BITS (LT48) (P)BE

The Lit Call 48 Bits operator places in the top-of-stack a single-precision operand equal in value to the next program word.

NOTE

Since the literal is synchronized by word, this operator can be 7-12 syllables long. Any unused syllables are filled in with the invalid operator code.

MAKE PROGRAM CONTROL WORD (MPCW) (P)BF

The Make Program Control Word operator performs a Lit Call 48 Bits operation except that the tag field is set to 111 to indicate a program control word and the stack number field of the PCW is inserted from the stack number register.

LOGICAL OPERATORS

Logical operators operate on the two top-of-stack operands bit for bit from bit 47 thru bit 0 to obtain logical values (48 logical values for single-precision operands and 96 for double-precision operands) which are left as the top-of-stack operand. If only one of the operands associated with LAND, LOR, LNOT, or LEQV is a double-precision operand, then the other operand will be extended with zeros. Logical operators may be used to operate on logical, string, or numeric operands.

LOGICAL AND (LAND) (P)90

The Logical And operator logically AND's each bit (except tag bits) of the two top-of-stack operands leaving the result in the top-of-stack. Each bit of the top-of-stack operand is set to one where a one appears in the corresponding bit positions of the two top-of-stack operands; the other information bits in the top-of-stack operand are set to zero. The tag of the second operand is undisturbed except for a double-precision operand in the top-of-stack, in which case the second operand is made double precision and the tag field is changed accordingly. AND is defined as follows:

Operand A	Operand B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

NOTE

The tag field is set equal to the second item in the stack.

LOGICAL OR (LOR) (P)91

The Logical Or operator logically OR's each bit (except tag bits) of the two top-of-stack operands leaving the result in the top-of-stack. OR is defined as follows:

Operand A	Operand B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NOTE

The tag field is set equal to the second item in the stack.

LOGICAL NEGATE (LNOT) (P)92

The Logical Negate operator complements each bit position (except tag bits) of the top-of-stack operand.

LOGICAL EQUIVALENCE (LEQV) (P)93

The Logical Equivalence operator compares the corresponding bits of the two items in the top-of-stack (except the tag bits). The two items are replaced by a single item with a tag field equal to the tag field of the second item in the stack and by a one in each bit position where the corresponding bits of the two top-of-stack items were equal.

PACK OPERATORS

PACK, DESTRUCTIVE (PACD) (P)D1

The Pack, Destructive operator packs data (as addressed by the source pointer) right-justified into the top-of-stack in 4-bit (digit) format.

The top-of-stack operand defines the length/repeat field (in digits) to be packed. The source pointer is the second item in the stack. The specified number of digits are transferred from the source to the top-of-stack (dropping the zones when required). If the digit length transferred is less than 13 the tag field in the top-of-stack is set to a single-precision operand; otherwise, the tag field is set to a double-precision operand.

If the length is not less than 25 an invalid-operand interrupt is set and the operation is terminated. If the source data has the memory protect bit (bit 48) set to "one," the segmented-array interrupt is set and the operation is terminated.

If the sign of the source data is negative, the true/false flip-flop is set to "one;" otherwise, the flip-flop is reset. Sign conventions are as follows:

Data Bit Format	Sign Location	Neg. Sign Zone Bit Config.
4-bit	Most significant digit	1101
6-bit	Least significant character	10
8-bit	Least significant byte	1101 (EBCDIC)
8-bit	Least significant byte	1111 (ASCII)

PACK, UPDATE (PACU) (P)D9

The Pack, Update operator performs a Pack, Destructive operation. At the completion of the operation, the source pointer is updated.

RELATIONAL OPERATORS

The relational operators perform algebraic comparisons on the two top-of-stack operands. The operands are removed from the stack and the result of the comparison is a logical operand which is placed in the top-of-stack. The result is a single-precision operand with the least significant bit set to one if the relation is true or a single-precision operand with all information bits set to zero if the relation is false.

GREATER THAN (GRTR) (P)8A

If the second operand in the stack is greater-than the top-of-stack operand, the Greater Than operator replaces the two operands with a single-precision operand which has the least-significant bit set to one.

If the second operand in the stack is not greater than the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

GREATER THAN OR EQUAL (GREQ) (P)89

If the second operand in the stack is greater than or equal to the top-of-stack operand, the Greater Than or Equal operator replaces the two operands with a single-precision operand which has the least-significant bit set to one. If the second operand in the stack is not greater than or equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

EQUAL (EQUL) (P)8C

If the second operand in the stack is algebraically equal to the top-of-stack operand, the Equal operator replaces the two operands with a single-precision operand which has the least-significant bit set to "one". If the second operand in the stack is not equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

LESS THAN OR EQUAL (LSEQ) (P)8B

If the second operand in the stack is less than or equal to the top-of-stack operand, the Less Than or Equal operator replaces the two operands with a single-precision operand which has the least significant bit set the "one". If the second operand in the stack is not less than or equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

LESS THAN (LESS) (P)88

If the second operand in the stack is less than the top-of-stack operand, the Less Than operator replaces the two operands with a single-precision operand which has the least-significant bit set to "one". If the second operand in the stack is not less than the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

NOT EQUAL (NEQL) (P)8D

If the second operand in the stack is not equal to the top-of-stack operand, the Not Equal operator replaces the two operands with a single-precision operand with the least significant bit set to "one". If the second operand in the stack is equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

LOGICAL EQUAL (SAME) (P)94

The Logical Equal operator compares all bits (including tag bits) of the two items (operands, control words, descriptors, etc.) in the top-of-stack. If all bits are equal, a single-precision operand (with the least significant bit set to one and all other information bits set to zero) is stored in the top-of-stack; otherwise, a single-precision operand with all information bits set to zero is stored in the top-of-stack.

SCALE OPERATORS

Some higher level languages such as COBOL and PL-I require integer arithmetic. The Scale-Left operators provide a means of aligning the decimal points prior to performing arithmetic operations. The Scale-Right operators provide a means of converting binary arithmetic to decimal arithmetic.

SCALE LEFT (SCLF) (P)C0

The Scale Left operator shifts the operand in the top-of-stack for decimal point alignment. The operand in the top-of-stack is first converted to an integer and then multiplied by 10 raised to the power specified by the scale factor. The scale factor is obtained from the second syllable (i.e., the program syllable following the operator syllable).

If scaling of a single-precision operand would result in overflow, the single-precision operand is converted to a double-precision integer. For the Scale operators, a double-precision integer is defined as a double-precision operand with an exponent equal to 13 (octal). If scaling of the operand results in an exponent greater than 13 (double-precision operand), the overflow flip-flop is set to "one".

DYNAMIC SCALE LEFT (DSLF) (P)C1

The Dynamic Scale Left operator performs a Scale Left operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second operand in the stack. The operand in the top-of-stack is converted to an integer before scaling takes place.

SCALE RIGHT SAVE (SCRS) (P)C4

The Scale Right Save operator shifts the top-of-stack operand to the right for conversion from a binary to a decimal numbering system. The operand in the top-of-stack is converted to an integer and divided by 10 raised to the power specified by the scale factor. The scale factor is obtained from the second syllable. If the scale factor is greater than 12, the invalid-operand interrupt is set and the operation is terminated.

The quotient resulting from the division is left in the top-of-stack. The second operand in the stack is the remainder which is converted to decimal (4-bit digits) and left justified.

DYNAMIC SCALE RIGHT SAVE (DSRS) (P)C5

The Dynamic Scale Right Save operator performs a Scale Right Save operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second item in the stack. The top-of-stack operand is converted to an integer before scaling takes place.

SCALE RIGHT TRUNCATE (SCRT) (P)C2

The Scale Right Truncate operator performs a Scale Right Save operation except that the remainder resulting from the division is deleted from the stack.

DYNAMIC SCALE RIGHT TRUNCATE (DSRT) (P)C3

The Dynamic Scale Right Truncate operator performs a Scale Right Truncate operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second operand in the stack.

SCALE RIGHT ROUNDED (SCRR) (P)C8

The Scale Right Rounded operator performs a Scale Right Save operation except that the remainder resulting from the division is deleted from the stack. If the most significant digit of the remainder is greater than or equal to five the quotient from the division is rounded by adding "one" to it.

DYNAMIC SCALE RIGHT ROUNDED (DSRR) (P)C9

The Dynamic Scale Right Rounded operator performs a Scale Right Rounded operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second operand in the stack.

SCALE RIGHT FINAL (SCRF) (P)C6

The Scale Right Final operator performs a Scale Right Save operation except that the quotient is deleted from the stack and the sign of the quotient is copied into the external sign flip-flop. If the quotient was not equal to zero at the conclusion of the operation, the overflow flip-flop is set.

DYNAMIC SCALE RIGHT FINAL (DSRF) (P)C7

The Dynamic Scale Right Final operator performs a Scale Right Final operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second item in the stack.

STACK OPERATORS

The stack operators are used to adjust the relative positions of the top items in the stack and to copy or delete the top of stack item.

EXCHANGE (EXCH) (P)B6

The Exchange operator causes the two top-of-stack items to be exchanged.

ROTATE STACK DOWN (RSDN) (V)B7

The Rotate Stack Down operator rotates the three top-of-stack words as follows:

Before Rotation	After Rotation
Word 1	Word 2
Word 2	Word 3
Word 3	Word 1

ROTATE STACK UP (RSUP) (V)B6

The Rotate Stack Up operator rotates the three top-of-stack words as follows:

Before Rotation	After Rotation
Word 1	Word 3
Word 2	Word 1
Word 3	Word 2

DUPLICATE TOP-OF-STACK (DUPL) (P)B7

The Duplicate Top-of-Stack operator duplicates the item in the top-of-stack.

DELETE TOP-OF-STACK (DLET) (P)B5

The Delete Top-of-Stack operator deletes the top-of-stack item.

PUSH DOWN STACK REGISTERS (PUSH) (P)B4

The Push Down Stack Registers operator pushes down the top-of-stack items and stack buffer contents into memory.

STORE DESTRUCTIVE (STOD) (P)B8

The Store Destructive operator stores the second item in the stack into memory. The address into which the item is to be stored is indicated by an indirect reference word or indexed data descriptor in the top-of-stack. If the top-of-stack item is an operand, the two top-of-stack items are exchanged so that the address item is in the top-of-stack and the item to be stored is in the second position. After the item is stored, both the item and its address are deleted from the stack.

If the word addressed by the indirect reference word is another indirect reference word or indexed data descriptor, or the word addressed by the data descriptor is another indexed data descriptor, the store operation will not occur to that location, but will be retried using the address indicated by that word. This chaining of address items will continue until a "target" location is reached; however, once a data descriptor has been encountered, an indirect reference word or PCW is not allowed, and once a stuffed indirect reference word has been encountered, a normal IRW is not allowed. Either of these conditions will cause an invalid-operand interrupt.

If the word addressed by the indirect reference word is a program control word, accidental procedure entry occurs. The spontaneously generated RCW causes STOD to be reexecuted upon return from the procedure.

If a data descriptor used as an address item has the read-only bit (bit 43) ON, or if the addressed word has the memory protect bit (bit 48) ON and is not a data descriptor, IRW, or PCW, the memory-protect interrupt is set and the operation is terminated.

If the presence bit in the data descriptor is zero, the presence-bit interrupt is set. After the data has been made present, the operation is restarted.

STORE NON-DESTRUCTIVE (STON) (P)B9

The Store Non-Destructive operator performs a Store Destructive operation, except that only the address item is deleted from the stack. The item which was stored is left in the top-of-stack.

OVERWRITE DESTRUCTIVE (OVRD) (P)BA

The Overwrite Destructive operator performs a Store Destructive operation, except that the addressed location will be overwritten regardless of its contents. Chaining of address items, memory protection checks, or accidental procedure entry do not occur.

OVERWRITE NON-DESTRUCTIVE (OVRN) (P)BB

The Overwrite Non-Destructive operator performs a Store Non-Destructive operation, except that the addressed location will be overwritten regardless of its contents. Chaining of address items, memory protection checks, or accidental procedure entry do not occur.

READ WITH LOCK (RDLK) (V)BA

The Read With Lock operator is a variant of the Overwrite Non-Destructive operator. The word in the top-of-stack and the specified word in memory are interchanged after all local

STRING OPERATORS

The string operators are used for transferring, comparing, scanning, and translating strings of data. In addition, a set of micro-operators provide a means of formatting data for input/output.

The string operators use a repeat value and source and destination pointers which are located in the stack. For most string operators, the repeat value range is from 0 to 2^{20}–1. If the repeat value is ≤ 0, the string operator checks for valid inputs and terminates. If the string operator is an update type operator, the normal updated descriptors are produced.

The source for the string operator can either be a pointer into an array or a single or double precision operand. If the source is an operand, the source character size is determined by either the string operator or the destination character size. The first source character to be used by the string operator is the left most character in the most significant word of the operand.

As the string operator acts upon each character in the operand, the operand is rotated left by one character so that the next character to be used is always the left most character in the rotated source operand. For update type string operators, the operand is placed back into the stack in its rotated form. The source and destination pointers can be:

a. An unindexed data descriptor.
b. An indexed data descriptor.
c. An unindexed string descriptor.
d. An indexed string descriptor.

When one descriptor (source or destination) is a data descriptor and the other is a string descriptor, the data descriptor is converted to a string descriptor of the same type.

If both descriptors are data descriptors or there is only one descriptor and it is a data descriptor, then the conversion is made to 8-bit character string descriptors. Note that the index field used by the string operators is the same as that found in the original descriptors.

If string descriptors, except for the translate and transfer word operators, contain different character sizes, the invalid-operand interrupt is caused.

If string operators contain an update variant, the indexed string descriptors pointing to the next character in the array to be used are left in the stack.

STRING ISOLATE (SISO) (P)D5

The String Isolate operator transfers from the source string to the top-of-stack the number of bytes specified by the repeat field. This string is right-justified and filled with leading zeros.

At the start of the operation, the top-of-stack operand specifies the length of the byte string and the second item in the stack is an operand or a descriptor used as the source pointer. If the number of bytes exceeds one word (6 bytes or 48 bits), the tag of the result is set to double precision. If the number of bits is greater than 96, an invalid operand interrupt is set and the operation is terminated. If the source data has the memory-protect bit (bit 48) set to "one", the segmented-array interrupt is set and the operation is terminated.

STRING TRANSFER OPERATORS

String transfer operators give the system the ability to transfer characters or words from one location in memory to another location in memory. The source and destination pointers are set from string descriptors in the stack.

TRANSFER WORDS, DESTRUCTIVE (TWSD) (P)D3

The Transfer Words, Destructive operator transfers the number of words specified by the top-of-stack operand and from the source string to the destination string. The first operand is integerized and is used as the count or repeat field. The second item in the stack (a string descriptor or operand) is the source pointer; i.e., it points at the source string. The third item in the stack (a string descriptor) is the destination pointer which is used to provide the address of the destination string. The number of words specified by the repeat field are transferred from the source to the destination. If the memory protect bit is ON during execution of the Transfer Words operator, then the segmented-array interrupt is set and the operation is terminated.

TRANSFER WORDS, UPDATE (TWSU) (P)DB

The Transfer Words, Update operator performs a Transfer Words, Destructive operation. At the completion of the operation the source and destination pointers are updated to point to the memory location where the transfer ended. If either pointer was a data descriptor, then an indexed data descriptor is updated.

TRANSFER WORDS, OVERWRITE DESTRUCTIVE (TWOD) (P)D4

The Transfer Words, Overwrite Destructive operator performs a Transfer Words, Destructive operation bypassing the memory-protection checks.

TRANSFER WORDS, OVERWRITE UPDATE (TWOU) (P)DC

The Transfer Words, Overwrite Update operator performs a Transfer Words, Update operation bypassing the memory-protection checks.

TRANSFER WHILE GREATER, DESTRUCTIVE (TGTD) (P)E2

The Transfer While Greater, Destructive operator transfers the number of characters specified by the second operand (bits 19:20) in the stack or while the source character is greater than a delimiter. The top-of-stack operand is the delimiter. The third item in the stack is the source pointer, and the fourth item is the destination pointer.

If the second item in the stack is a descriptor, it is used as a source pointer. This means that no repeat field was given and the default field length is 1,048,575.

If either the source or destination word has the memory protect bit ON (bit 48 = 1), the segmented-array interrupt is set and the operation is terminated.

All comparisons are binary (EBCDIC collating sequence). When the source pointer is an operand, it must be a single-precision operand or an invalid-operand interrupt is set and the operation is terminated. The source character is compared with the delimiter. If the comparison is true, the true/false flip-flop is set to "one"; if the comparison fails, the true/false flip-flop is set to zero.

TRANSFER WHILE GREATER, UPDATE (TGTU) (P)EA

The Transfer While Greater, Update operator performs a Transfer While Greater, Destructive operation. At the completion of the operation, the source and destination pointers are updated to point at the next character in the source and destination strings, respectively. At the completion of the operation, a count of the number of characters not transferred is placed on the top-of-stack. If all the characters specified by the length field are transferred, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false.

If the operation is terminated because the relationship is not met, the source pointer points to the character which stopped the transfer.

TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE (TGED) (P)E1

The Transfer While Greater or Equal operator performs a Transfer While Greater, Destructive operation while the source character is greater than or equal to the delimiter.

TRANSFER WHILE GREATER OR EQUAL, UPDATE (TGEU) (P)E9

The Transfer While Greater or Equal, Update operator performs a Transfer While Greater Than or Equal operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE EQUAL, DESTRUCTIVE (TEQD) (P)E4

The Transfer While Equal, Destructive operator performs a Transfer While Greater or Equal, Destructive operation while the source character is equal to the delimiter.

TRANSFER WHILE EQUAL, UPDATE (TEQU) (P)EC

The Transfer While Equal, Update operator performs a Transfer While Equal, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE (TLED) (P)E3

The Transfer While Less or Equal, Destructive operator performs a Transfer While Greater or Equal, Destructive operation while the source character is less than or equal to the delimiter.

TRANSFER WHILE LESS OR EQUAL, UPDATE (TLEU) (P)EB

The Transfer While Less or Equal, Update operator performs a Transfer While Less or Equal, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE LESS, DESTRUCTIVE (TLSD) (P)E0

The Transfer While Less, Destructive operator performs a Transfer While Less or Equal, Destructive operation while the source character is less than the delimiter.

TRANSFER WHILE LESS, UPDATE (TLSU) (P)E8

The Transfer While Less, Update operator performs a Transfer While Less, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE NOT EQUAL, DESTRUCTIVE (TNED) (P)E5

The Transfer While Not Equal, Destructive operator performs a Transfer While Greater or Equal, Destructive operation while the source character is not equal to the delimiter.

TRANSFER WHILE NOT EQUAL, UPDATE (TNEU) (P)ED

The Transfer While Not Equal, Update operator performs a Transfer While Not Equal Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE TRUE, DESTRUCTIVE (TWTD) (V)D3

The Transfer While True, Destructive operator transfers characters from the source string to the destination string for the number of characters specified by the length operand while the stated relationship is met. If the relationship is not met the transfer is terminated at that point. The relationship is determined by using the source character to index a bit in the table. If the bit indexed is a "one" the relationship is true. An all zero's character indexes to the most significant bit of the table.

The operator uses the top four words in the stack as follows: The top-of-stack word is a table pointer to specific addresses in the table; the second word in the stack provides the length of the string to be transferred or, if it is a descriptor, it is used as a source pointer since no repeat field was given and the default field length is set at 1,048,575; the third word in the stack is an operand or a descriptor which gives the address of the source string or is a single-precision operand which is the source string; the fourth word in the stack is a descriptor pointing at the destination string.

The table is indexed as follows to obtain the decision bit: The source character is expanded to eight bits, if necessary, by appending two or four leading-zero bits. The three high-order bits of the source character select a word from the table, indexing the table pointer. The remaining five bits of the expanded source character select a word from the table, indexing the table pointer. The remaining five bits of the expanded source character select (by their value) a bit from this word.

At the completion of the operation, a count of the number of characters not transferred is placed on the top of stack.

If all the characters specified by the length field are transferred, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false.

The table format is as follows:

Source Size	Table Length	Bits/Word
4	1 word	(31:16)
6	2 words	(31:32)
8	8 words	(31:32)

TRANSFER WHILE TRUE, UPDATE (TWTU) (V)DB

The Transfer While True, Update operator performs a Transfer While True, Destructive operation. At the completion of the operation, the source destination pointers and the count are updated. If all the characters specified by the length field are transferred, the true/false flip-flop is set to "one" (true); otherwise it is set to zero.

TRANSFER WHILE FALSE, DESTRUCTIVE (TWFD) (V)D2

The Transfer While False, Destructive operator performs the Transfer While True operation except that the relationship is true if the bit found by indexing into the table is a zero.

TRANSFER WHILE FALSE, UPDATE (TWFU) (V)DA

The Transfer While False, Update operator performs a Transfer While False, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER UNCONDITIONAL, DESTRUCTIVE (TUND) (P)E6

The Transfer Unconditional, Destructive operator transfers from the source to the destination the number of characters specified by the top-of-stack operand. If the top-of-stack item is a descriptor, it is used as a source pointer. Since no repeat field was given, the field length is set by default at 1,048,575. The second item in the stack is the destination pointer. If all characters specified by the length field are transferred, the true/false flip-flop is set to "one" (true) by this operand; otherwise, the flip-flop is set to zero (false).

TRANSFER UNCONDITIONAL, UPDATE (TUNU) (P)EE

The Transfer Unconditional, Update operator performs a Transfer Unconditional, Destructive operation. At the completion of the operation, the source and destination pointers are updated.

SUBROUTINE OPERATORS

Subroutine operators are those operators which can move the program operation across machine architecture such as from stack to stack, or from subroutine to subroutine, etc.

Any subroutine operator which can "chain" indirect reference words (IRW's) or stuffed indirect reference words (SIRW's) can obtain accidental procedure entry if a program control word (PCW) is pointed to by the IRW or SIRW last in the chain.

MARK STACK (MKST) (P)AE

The Mark Stack operator inserts a mark into the stack which is to be subsequently used by an Enter operator. The mark is placed in the top-of-stack in the form of a mark stack control word. The F register is set to the location of the MSCW.

The Mark Stack operator is normally used when an entry to a procedure is anticipated. The normal sequence of events to enter a procedure is (1) mark the stack; (2) insert an indirect reference to a program control word; (3) insert parameters, if any are to be passed to the procedure; and then (4) execute an Enter operator, which will in turn, cause an entry into the program segment located by the program control word.

INSERT MARK STACK (IMKS) (P)CF

The Insert Mark Stack operator inserts a mark stack control word in the current stack below the two top-of-stack items.

NAME CALL (NAMC) (P)40 THRU (P)7F

Name Call builds an indirect reference word in the top-of-stack. The six low-order bits of the first syllable and the eight bits of the second syllable form a 14-bit address couple. This address couple is placed in the top-of-stack with the tag field set to 001.

In the B 7700, if the Name Call is followed by a ENTR, INDX, NXLN, NXLV, STOD, STON, OVRD, OVRN, DBUN, LOAD, or LODT operator, the IRW is not placed in the stack. Instead, the referenced memory address is calculated and, if appropriate, the memory access is initiated by the program control unit. The following operator is sent to the execution unit along with an indication that the operator has been started. Since the address computation and, in some cases, the memory fetch, is complete by the time the operator reaches the execution unit, a considerable time savings is realized.

VALUE CALL (VALC) (P)00 THRU (P)3F

Value Call is a two-syllable instruction that brings an operand from memory into the top-of-stack. A concatenation of the two Value Call syllable gives a 14-bit address couple. If the referenced memory location contains an indirect reference word or a data descriptor, additional memory accesses are made until the "target" operand is located. The operand is then placed in the top-of-stack. The operand may be either single-precision or double-precision, causing either one or two words to be loaded into the top-of-stack. (Figures III-3-6 is simplified flow charts of the Value Call operator.)

If the word accessed is an indexed data descriptor, the word addressed by the data descriptor is brought to the top-of-stack. If the word accessed is a non-indexed word data descriptor, the descriptor is indexed using the second word in the stack as the index value, and the word addressed by the non-indexed data descriptor is brought to the top-of-stack. If the double-precision bit (bit 40) in the data descriptor is set, the second half of the double-precision operand is placed in the second half of the top-of-stack location.

If the presence bit in the data descriptor is zero, the presence-bit interrupt is set. After the data has been made present, the operation is restarted.

If a data descriptor does not address an operand, step index word, or a word descriptor or indexed string descriptor, an invalid-operand interrupt is set and the operation is terminated.

If the word accessed by the Value Call is an indirect reference word (IRW), the word addressed by the IRW is brought to the top-of-stack.

If the word accessed is a program control word (PCW), an accidental entry into the subroutine addressed by the PCW is initiated. A mark stack control word and return control word are placed in the stack and an entry is made into the subprogram. Upon completion of the subprogram, a return operation will re-enter the Value Call operator flow.

If the target operand is a step index word (tag = 4) instead of an operand, the current-value field (bits 15:16) of the SIW will be placed in the top-of-stack with the tag set to zero.

The "chaining" of memory accesses continues until a target operand is reached; however, once a data descriptor has been encountered, an indirect reference word or PCW is not allowed, and once a stuffed indirect reference word has been encountered, a normal IRW is not allowed. Either of these conditions will cause an invalid-operand interrupt.

EVALUATE DESCRIPTOR (EVAL) (P)AC

The Evaluate Descriptor operator loads into the top of stack a data descriptor or an indirect reference word which points to the referenced operand. This operand may be referenced through a chain of indirect reference words or descriptors. (Figure III-3-7 is a sim-

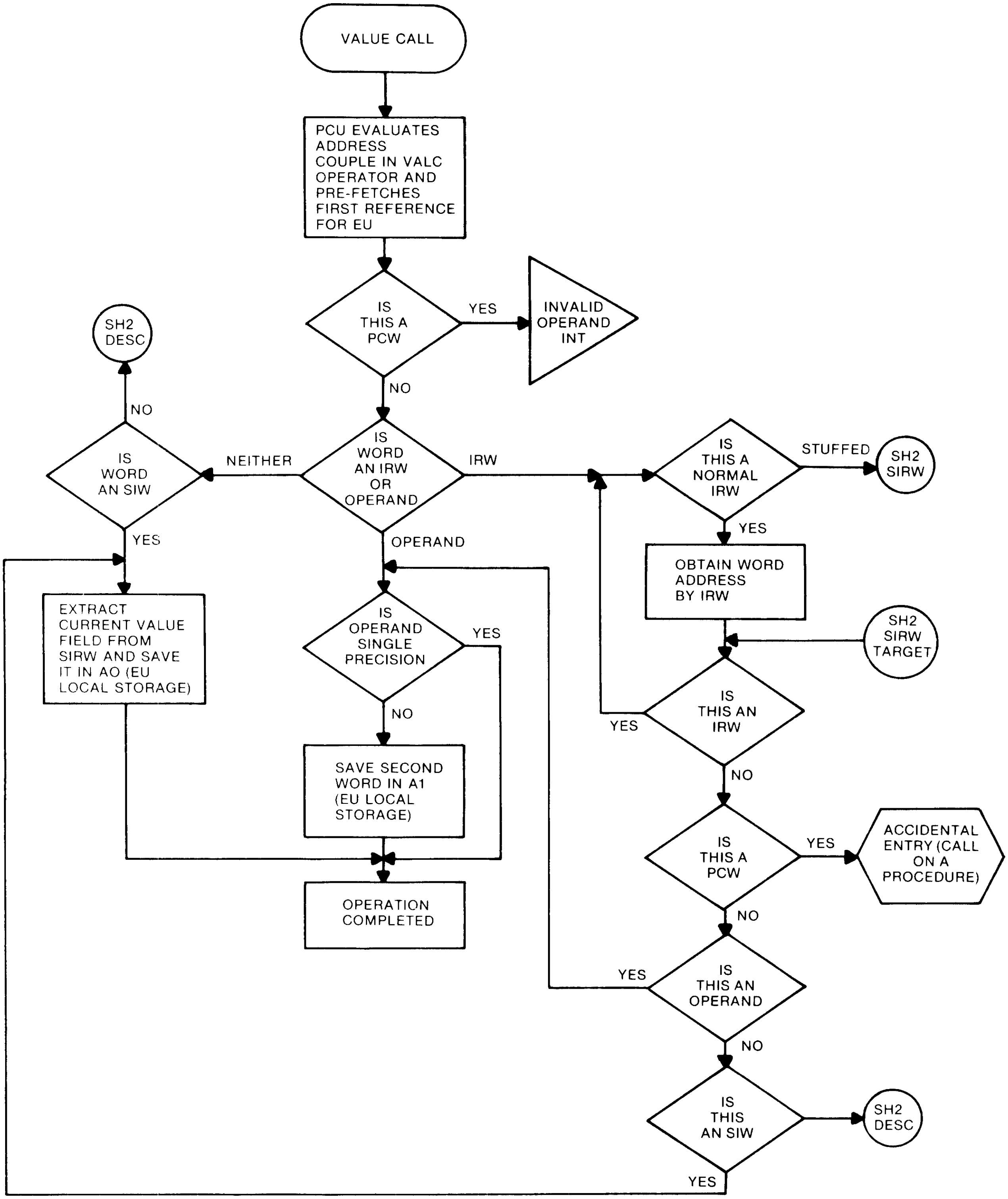

VALUE CALL
PCU EVALUATES ADDRESS COUPLE IN VALC OPERATOR AND PRE-FETCHES FIRST REFERENCE FOR EU
IS THIS A PCW
YES
INVALID OPERAND INT
NO
SH2 DESC
NO
IS WORD AN SIW
NEITHER
IS WORD AN IRW OR OPERAND
IRW
IS THIS A NORMAL IRW
STUFFED
SH2 SIRW
YES
OBTAIN WORD ADDRESS BY IRW
SH2 SIRW TARGET
YES
OPERAND
EXTRACT CURRENT VALUE FIELD FROM SIRW AND SAVE IT IN AO (EU LOCAL STORAGE)
IS OPERAND SINGLE PRECISION
YES
NO
SAVE SECOND WORD IN A1 (EU LOCAL STORAGE)
OPERATION COMPLETED
IS THIS AN IRW
YES
NO
IS THIS A PCW
YES
ACCIDENTAL ENTRY (CALL ON A PROCEDURE)
NO
YES
IS THIS AN OPERAND
NO
IS THIS AN SIW
SH2 DESC
YES

Figure III-3-6. Flow Chart of Value Call Operator (Sheet 1 of 2)

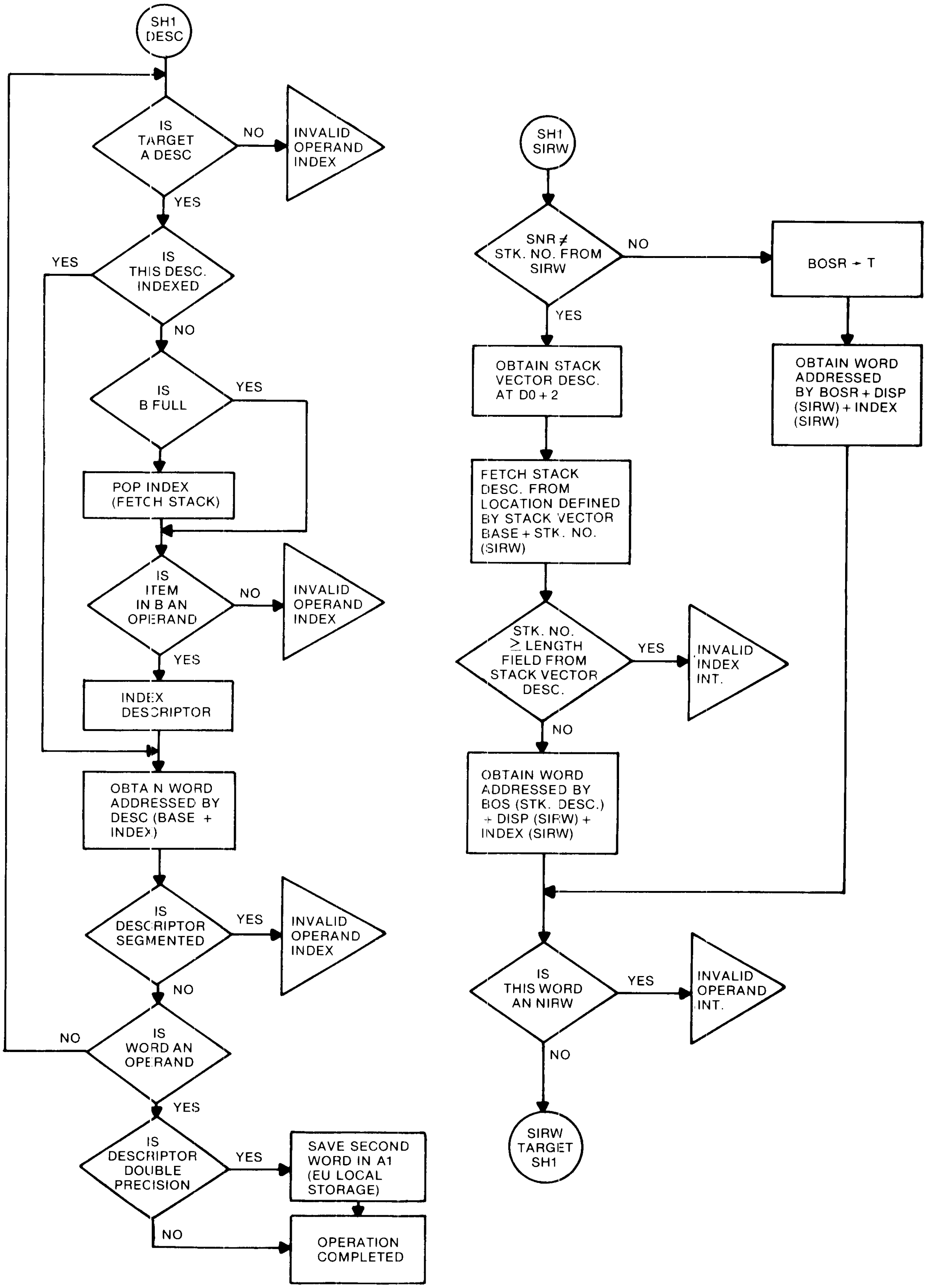

SH1 DESC
IS TARGET A DESC
NO
INVALID OPERAND INDEX
YES
IS THIS DESC. INDEXED
YES
NO
IS B FULL
YES
POP INDEX (FETCH STACK)
IS ITEM IN B AN OPERAND
NO
INVALID OPERAND INDEX
YES
INDEX DESCRIPTOR
OBTAIN WORD ADDRESSED BY DESC (BASE + INDEX)
IS DESCRIPTOR SEGMENTED
YES
INVALID OPERAND INDEX
NO
IS WORD AN OPERAND
NO
YES
IS DESCRIPTOR DOUBLE PRECISION
YES
SAVE SECOND WORD IN A1 (EU LOCAL STORAGE)
NO
OPERATION COMPLETED
SH1 SIRW
SNR ≠ STK. NO. FROM SIRW
NO
BOSR → T
YES
OBTAIN STACK VECTOR DESC. AT D0 + 2
OBTAIN WORD ADDRESSED BY BOSR + DISP (SIRW) + INDEX (SIRW)
FETCH STACK DESC. FROM LOCATION DEFINED BY STACK VECTOR BASE + STK. NO. (SIRW)
STK. NO. ≥ LENGTH FIELD FROM STACK VECTOR DESC.
YES
INVALID INDEX INT.
NO
OBTAIN WORD ADDRESSED BY BOS (STK. DESC.) + DISP (SIRW) + INDEX (SIRW)
IS THIS WORD AN NIRW
YES
INVALID OPERAND INT.
NO
SIRW TARGET SH1

Figure III-3-6. Flow Chart of Value Call Operator (Sheet 2 of 2)

plified flow chart of the Evaluate Descriptor operator.)

A descriptor is left in the stack if the operand was referenced by a descriptor. If only indirect reference words are used, multiple memory accesses are made until the operand is located. A stuffed indirect reference word pointing to the operand is left in the stack.

An invalid-operator interrupt is set and the operation is terminated if the top-of-stack word is not a descriptor or an indirect reference word at the start of the Evaluate operator.

ENTER (ENTR) (P)AB

The Enter operator causes an entry into a procedure from a calling procedure. (The sequence of events to enter a procedure is: (1) mark the stack; (2) insert an indirect reference to a program control word; (3) insert parameter(s), if any are to be passed to the procedure; and, (4) execute an Enter operator.) The Enter operator causes entry into the program segment located by the program control word. A return control word is stored at stack location F+1. (Figure III-3-8 is a flow chart of the Enter operator.)

EXIT (EXIT) (P)A3

The EXIT operator causes a called procedure to return to a calling procedure and is used when the called procedure is not required to return a result. The Exit operator returns all control registers to the position they were in prior to the calling procedure, saves the bottom of stack register (BOSR), and cuts back the stack. (Figure III-3-9 is a flow chart of the Exit operator.)

RETURN (RETN) (P)A7

The Return operator causes a called procedure to return to a calling procedure (as in EXIT) but is used when the called procedure is required to return a result. An operand or name in the top-of-stack is returned to the calling procedure. If a name is returned and the V bit (bit 19) in the MSCW is ON, the name is evaluated to yield an operand as in VALC (since the V-bit indicates that the RETN is to VALC which caused accidental entry). (See figure III-3-9.)

TRANSFER OPERATORS

The transfer operators transfer any field of bits from one word in the stack to any field of another word in the stack.

NOTE

For all transfer operators the values specified in the stack must be non-negative.

FIELD TRANSFER (FLTR) (P)98

The Field Transfer operator uses the three syllables following it to establish the pointers used in the field transfer. Stack adjustment takes place so that the two top-of-stack locations are full. The contents of the field in the top-of-stack, starting at the bit position addressed by the third syllable of FLTR, is transferred into a field of corresponding length in the second location in the stack. The field in the second location in the stack starts at the bit position indicated by the second syllable of FLTR and proceeds toward the low-order-bit positions. When the number of bits specified by the fourth syllable of FLTR has been transferred the top-of-stack word and the operation is complete.

If the second or third syllables of the operator are found to be greater than 47 or the fourth syllable is greater than 48, the invalid operand interrupt is set and the operation is terminated.

DYNAMIC FIELD TRANSFER (DFTR) (P)99

The Dynamic Field Transfer operator causes a Field Transfer operation using the top-of-stack operand to specify the field length, using the second operand in the stack to specify the starting-bit position of the field from which the transfer will be made, and using the third operand in the stack to indicate the starting bit of the field to which the transfer will be made.

As each of these operands is used to establish a pointer for the transfer, it is first integerized and checked for being greater than 47 or 48, as above, then is deleted from the stack. The fourth and fifth stack operands become the two top-of-stack operands, and the transfer takes place as in the FLTR operator.

FIELD ISOLATE (ISOL) (P)9A

The Field Isolate operator isolates a field in the top-of-stack word. The second syllable of the operator specifies the starting bit. The third syllable specifies the length of the field in bits. The isolated field is right-justified with all other information bits set to zero. The tag bits are not changed.

DYNAMIC FIELD ISOLATE (DISO) (P)9B

The Dynamic Field Isolate operator performs a Field Isolate operation using the top-of-stack operand to specify the length of the field to be isolated and using the second operand in the stack to specify the starting bit. These operands are then deleted from the stack and the Field Isolate operation is performed on the next operand.

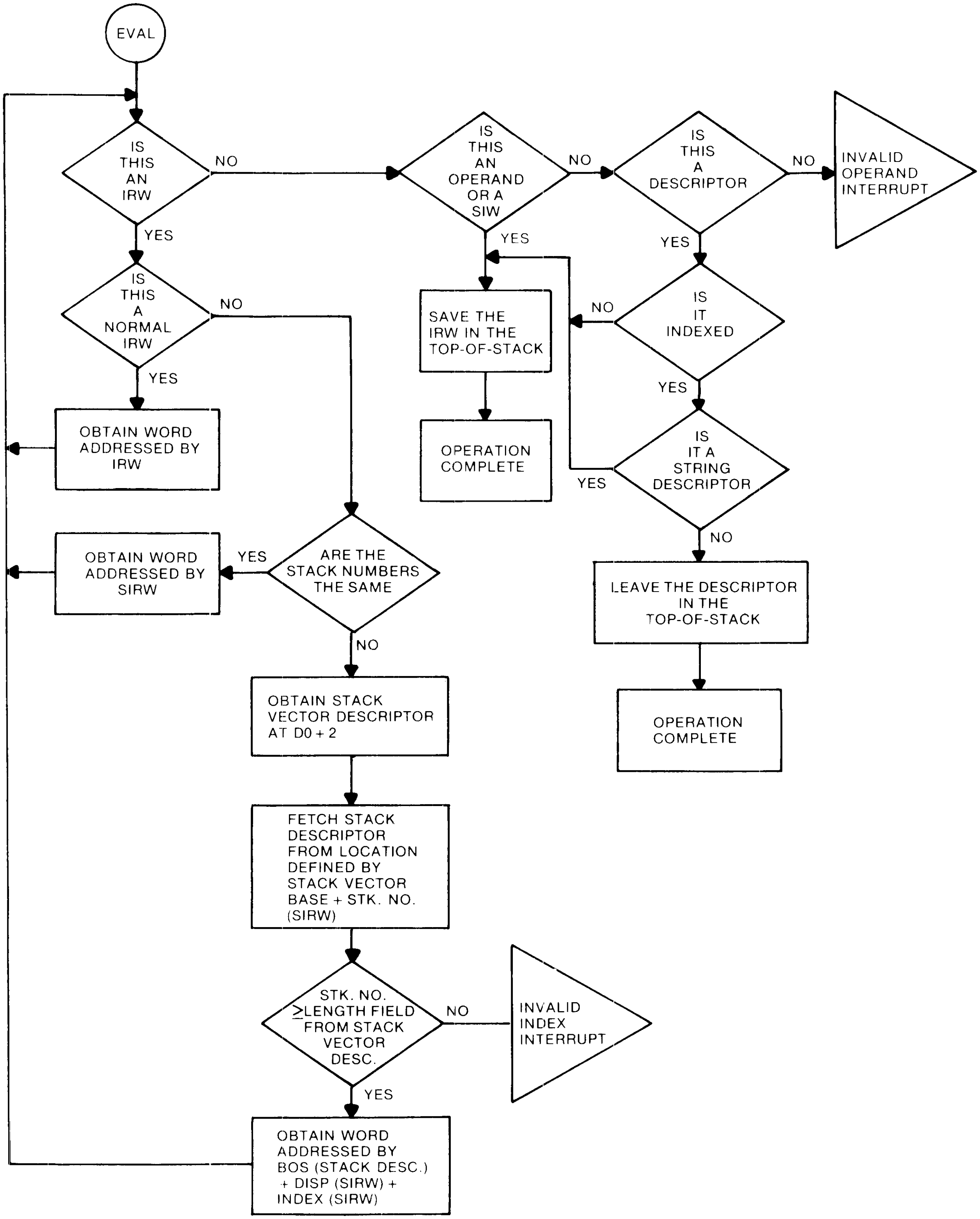
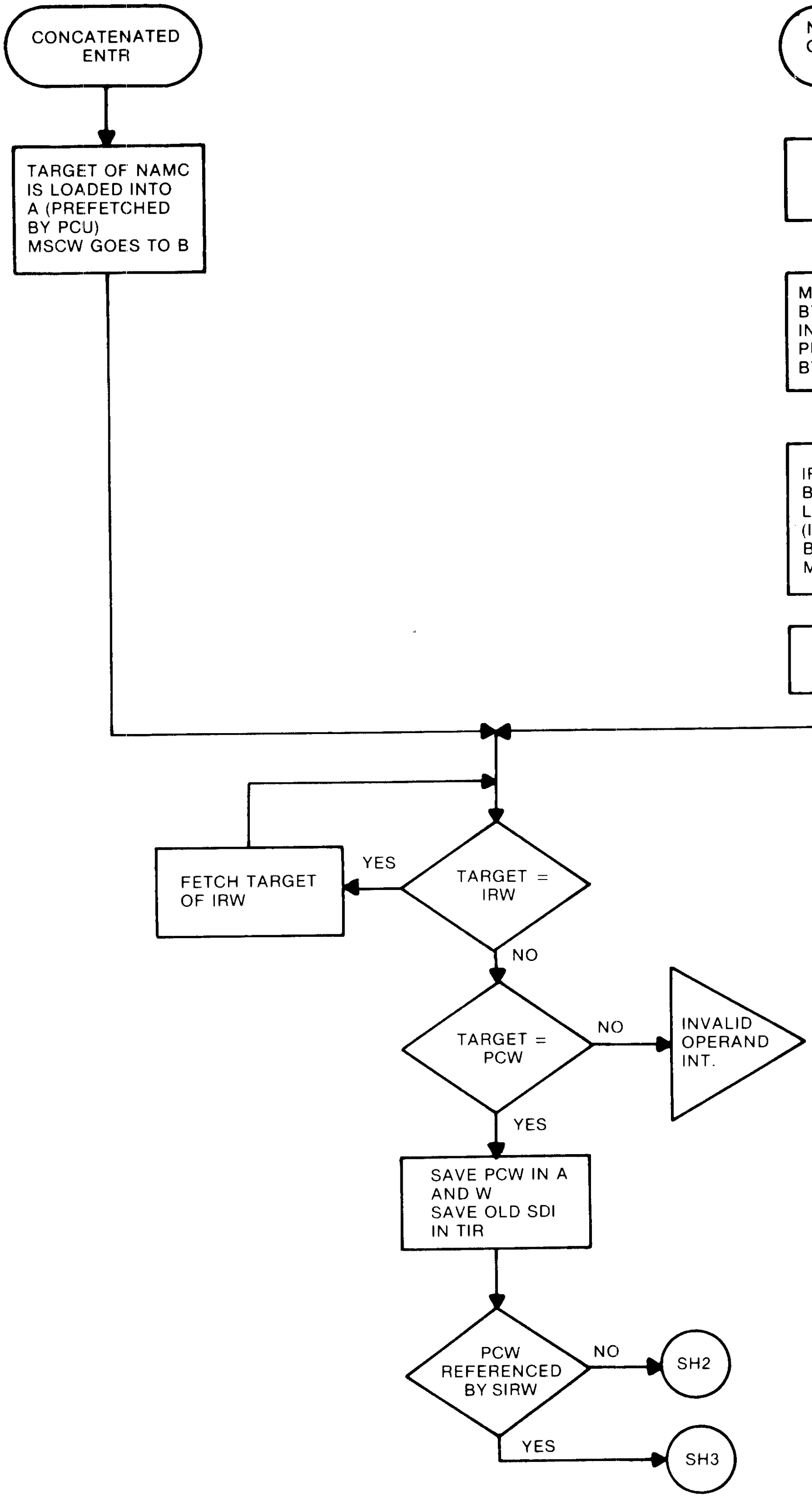

EVAL
IS THIS AN IRW
NO
YES
IS THIS A NORMAL IRW
NO
YES
OBTAIN WORD ADDRESSED BY IRW
IS THIS AN OPERAND OR A SIW
NO
YES
SAVE THE IRW IN THE TOP-OF-STACK
OPERATION COMPLETE
IS THIS A DESCRIPTOR
NO
YES
INVALID OPERAND INTERRUPT
IS IT INDEXED
NO
YES
IS IT A STRING DESCRIPTOR
YES
NO
LEAVE THE DESCRIPTOR IN THE TOP-OF-STACK
OPERATION COMPLETE
OBTAIN WORD ADDRESSED BY SIRW
YES
ARE THE STACK NUMBERS THE SAME
NO
OBTAIN STACK VECTOR DESCRIPTOR AT D0 + 2
FETCH STACK DESCRIPTOR FROM LOCATION DEFINED BY STACK VECTOR BASE + STK. NO. (SIRW)
STK. NO. ≥LENGTH FIELD FROM STACK VECTOR DESC.
NO
INVALID INDEX INTERRUPT
YES
OBTAIN WORD ADDRESSED BY BOS (STACK DESC.) + DISP (SIRW) + INDEX (SIRW)

Figure III-3-7. Flow Chart of Evaluate Operator



CONCATENATED ENTR
TARGET OF NAMC IS LOADED INTO A (PREFETCHED BY PCU) MSCW GOES TO B
NON-CONCATENATED ENTR
PUSH A AND B (IF NECESSARY)
MSCW ADDRESSED BY F IS LOADED INTO A (WAS PRE-FETCHED BY PCU)
IRW ADDRESSED BY F+1 IS LOADED INTO A (IRW PRE-FETCHED BY PCU) MSCW GOES TO B
FETCH TARGET OF IRW
FETCH TARGET OF IRW
YES
TARGET = IRW
NO
TARGET = PCW
NO
INVALID OPERAND INT.
YES
SAVE PCW IN A AND W SAVE OLD SDI IN TIR
PCW REFERENCED BY SIRW
NO
SH2
YES
SH3

Figure III-3-8. Flow Chart of Enter Operator (Sheet 1 of 3)

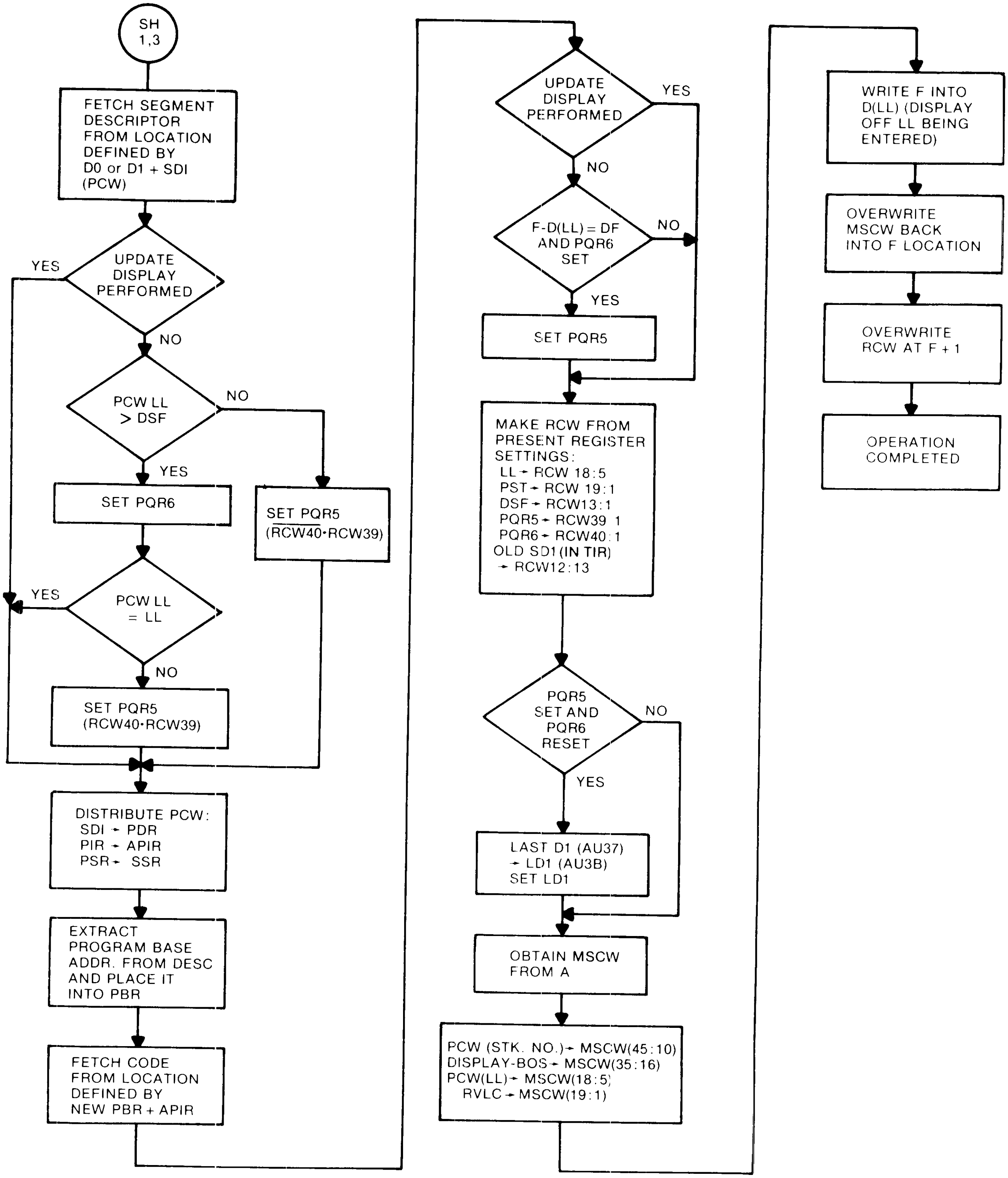

SH 1,3
FETCH SEGMENT DESCRIPTOR FROM LOCATION DEFINED BY D0 or D1 + SDI (PCW)
UPDATE DISPLAY PERFORMED
YES
NO
PCW LL > DSF
NO
YES
SET PQR6
SET PQR5 ($\overline{\text{RCW40}}$•RCW39)
PCW LL = LL
YES
NO
SET PQR5 (RCW40•RCW39)
DISTRIBUTE PCW:
SDI → PDR
PIR → APIR
PSR → SSR
EXTRACT PROGRAM BASE ADDR. FROM DESC AND PLACE IT INTO PBR
FETCH CODE FROM LOCATION DEFINED BY NEW PBR + APIR
UPDATE DISPLAY PERFORMED
YES
NO
F-D(LL) = DF AND PQR6 SET
NO
YES
SET PQR5
MAKE RCW FROM PRESENT REGISTER SETTINGS:
LL → RCW 18:5
PST → RCW 19:1
DSF → RCW13:1
PQR5 → RCW39 1
PQR6 → RCW40:1
OLD SD1 (IN TIR) → RCW12:13
PQR5 SET AND PQR6 RESET
NO
YES
LAST D1 (AU37) → LD1 (AU3B) SET LD1
OBTAIN MSCW FROM A
PCW (STK. NO.) → MSCW(45:10)
DISPLAY-BOS → MSCW(35:16)
PCW(LL) → MSCW(18:5)
RVLC → MSCW(19:1)
WRITE F INTO D(LL) (DISPLAY OFF LL BEING ENTERED)
OVERWRITE MSCW BACK INTO F LOCATION
OVERWRITE RCW AT F + 1
OPERATION COMPLETED

Figure III-3-8. Flow Chart of Enter Operator (Sheet 2 of 3)

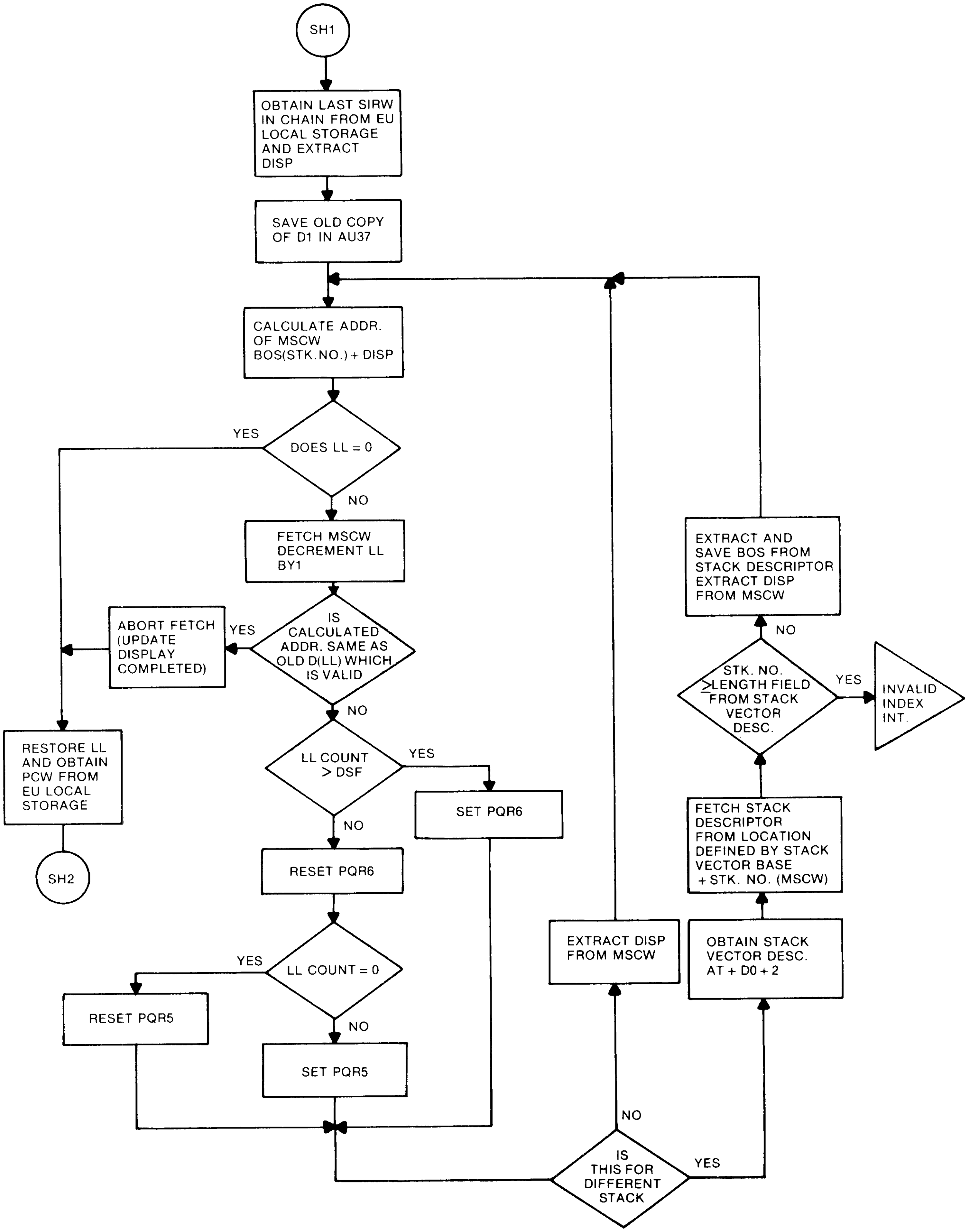

SH1
OBTAIN LAST SIRW IN CHAIN FROM EU LOCAL STORAGE AND EXTRACT DISP
SAVE OLD COPY OF D1 IN AU37
CALCULATE ADDR. OF MSCW BOS(STK.NO.) + DISP
DOES LL = 0
YES
NO
FETCH MSCW DECREMENT LL BY1
IS CALCULATED ADDR. SAME AS OLD D(LL) WHICH IS VALID
YES
ABORT FETCH (UPDATE DISPLAY COMPLETED)
NO
RESTORE LL AND OBTAIN PCW FROM EU LOCAL STORAGE
SH2
LL COUNT > DSF
YES
SET PQR6
NO
RESET PQR6
LL COUNT = 0
YES
RESET PQR5
NO
SET PQR5
IS THIS FOR DIFFERENT STACK
NO
EXTRACT DISP FROM MSCW
YES
OBTAIN STACK VECTOR DESC. AT + D0 + 2
FETCH STACK DESCRIPTOR FROM LOCATION DEFINED BY STACK VECTOR BASE + STK. NO. (MSCW)
STK. NO. ≥LENGTH FIELD FROM STACK VECTOR DESC.
YES
INVALID INDEX INT.
NO
EXTRACT AND SAVE BOS FROM STACK DESCRIPTOR EXTRACT DISP FROM MSCW

Figure III-3-8. Flow Chart of Enter Operator (Sheet 3 of 3)

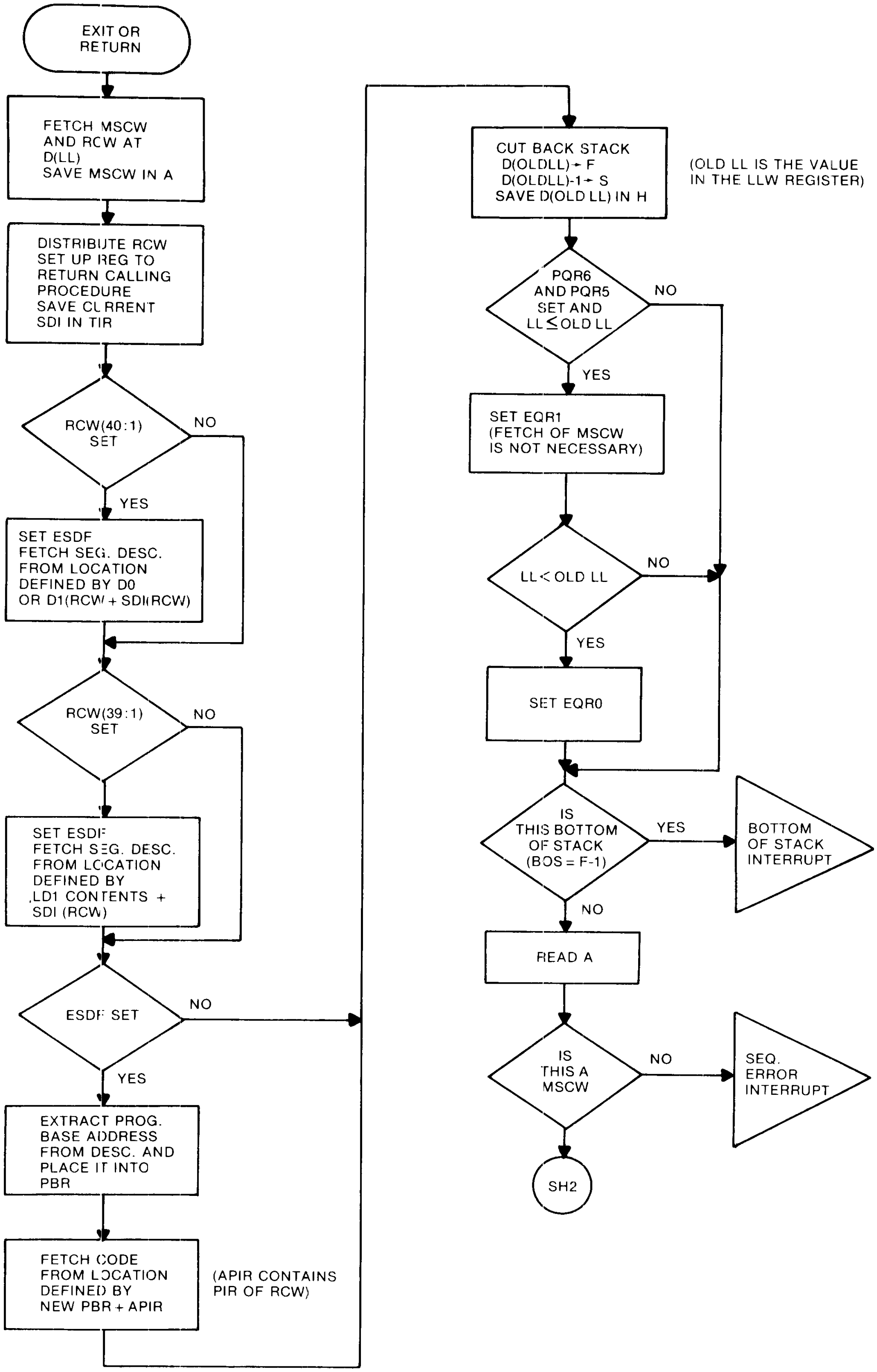

EXIT OR RETURN
FETCH MSCW AND RCW AT D(LL) SAVE MSCW IN A
DISTRIBUTE RCW SET UP REG TO RETURN CALLING PROCEDURE SAVE CURRENT SDI IN TIR
RCW(40:1) SET
NO
YES
SET ESDF FETCH SEG. DESC. FROM LOCATION DEFINED BY D0 OR D1(RCW + SDI(RCW)
RCW(39:1) SET
NO
SET ESDF FETCH SEG. DESC. FROM LOCATION DEFINED BY LD1 CONTENTS + SDI (RCW)
ESDF SET
NO
YES
EXTRACT PROG. BASE ADDRESS FROM DESC. AND PLACE IT INTO PBR
FETCH CODE FROM LOCATION DEFINED BY NEW PBR + APIR
(APIR CONTAINS PIR OF RCW)
CUT BACK STACK D(OLDLL) ← F D(OLDLL)-1 ← S SAVE D(OLD LL) IN H
(OLD LL IS THE VALUE IN THE LLW REGISTER)
PQR6 AND PQR5 SET AND LL ≤ OLD LL
NO
YES
SET EQR1 (FETCH OF MSCW IS NOT NECESSARY)
LL < OLD LL
NO
YES
SET EQR0
IS THIS BOTTOM OF STACK (BOS = F-1)
YES
BOTTOM OF STACK INTERRUPT
NO
READ A
IS THIS A MSCW
NO
SEQ. ERROR INTERRUPT
SH2

Figure III-3-9. Flow Chart of Exit and Return Operator (Sheet 1 of 3)

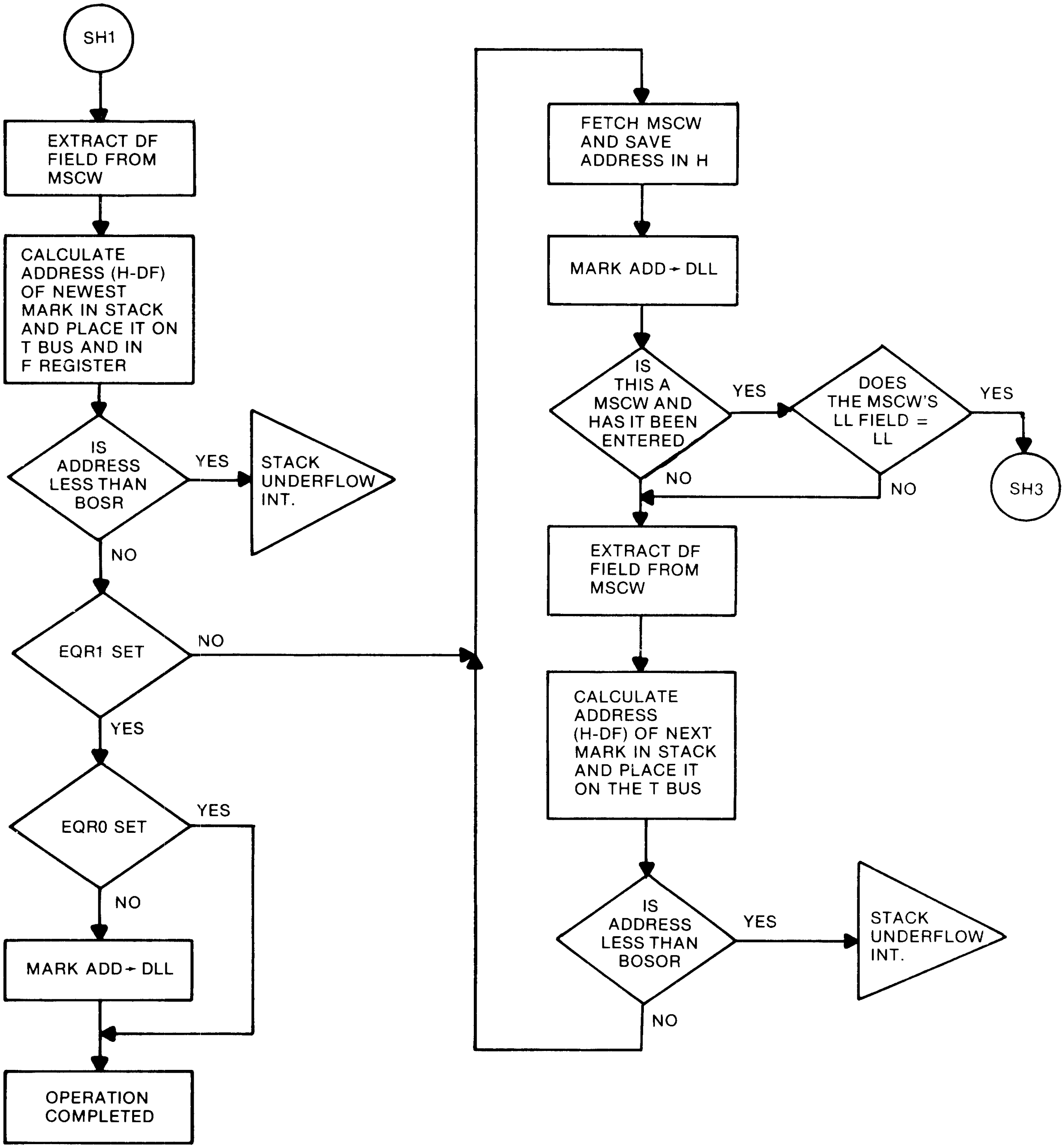

SH1
EXTRACT DF FIELD FROM MSCW
CALCULATE ADDRESS (H-DF) OF NEWEST MARK IN STACK AND PLACE IT ON T BUS AND IN F REGISTER
IS ADDRESS LESS THAN BOSR
YES
STACK UNDERFLOW INT.
NO
EQR1 SET
NO
YES
EQR0 SET
YES
NO
MARK ADD→DLL
OPERATION COMPLETED
FETCH MSCW AND SAVE ADDRESS IN H
MARK ADD→DLL
IS THIS A MSCW AND HAS IT BEEN ENTERED
YES
DOES THE MSCW'S LL FIELD = LL
YES
SH3
NO
NO
EXTRACT DF FIELD FROM MSCW
CALCULATE ADDRESS (H-DF) OF NEXT MARK IN STACK AND PLACE IT ON THE T BUS
IS ADDRESS LESS THAN BOSOR
YES
STACK UNDERFLOW INT.
NO

Figure III-3-9. Flow Chart of Exit and Return Operator (Sheet 2 of 3)

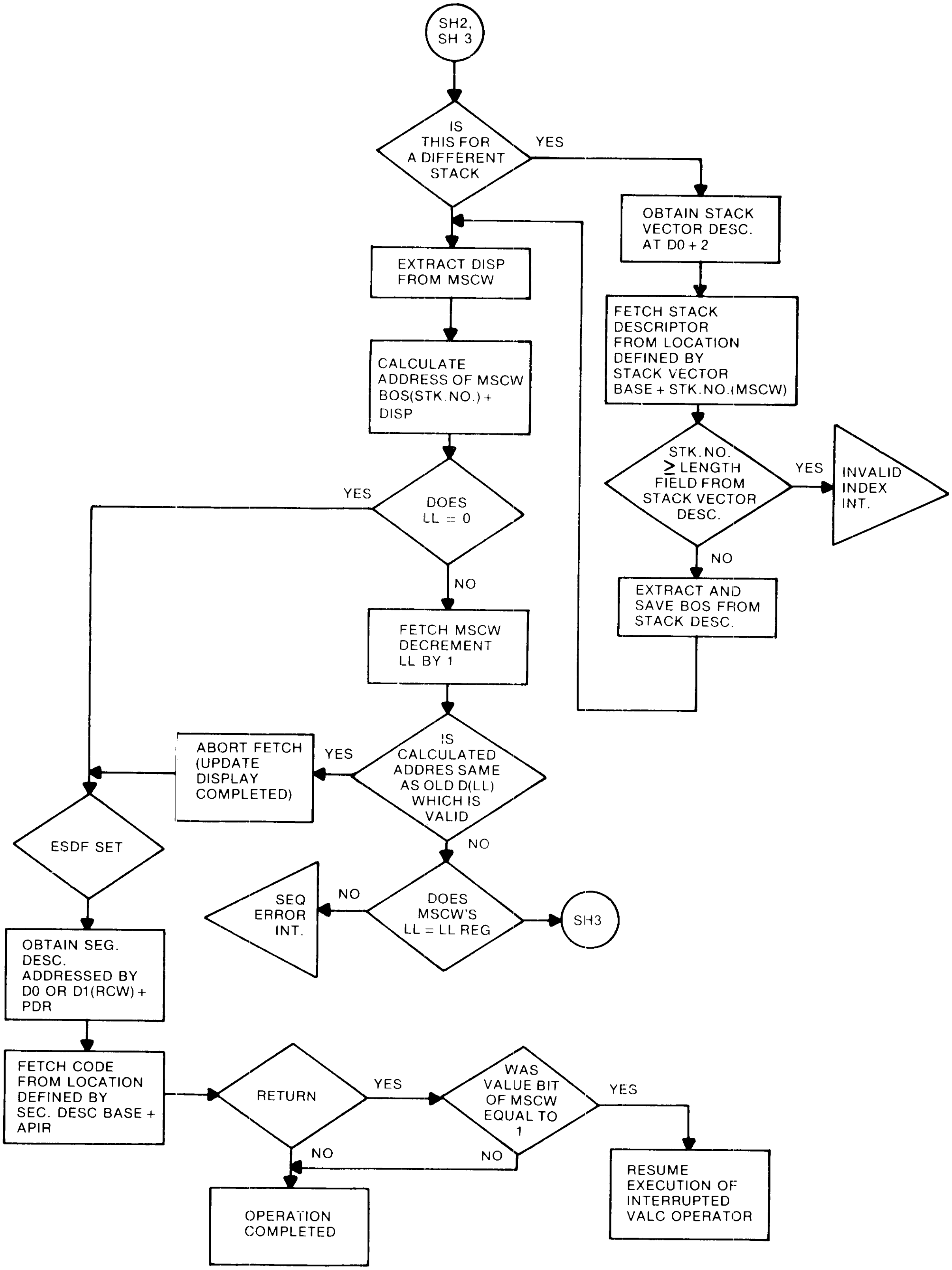

SH2, SH 3
IS THIS FOR A DIFFERENT STACK
YES
OBTAIN STACK VECTOR DESC. AT D0 + 2
FETCH STACK DESCRIPTOR FROM LOCATION DEFINED BY STACK VECTOR BASE + STK.NO.(MSCW)
STK.NO. ≥ LENGTH FIELD FROM STACK VECTOR DESC.
YES
INVALID INDEX INT.
NO
EXTRACT AND SAVE BOS FROM STACK DESC.
EXTRACT DISP FROM MSCW
CALCULATE ADDRESS OF MSCW BOS(STK.NO.) + DISP
DOES LL = 0
YES
NO
FETCH MSCW DECREMENT LL BY 1
IS CALCULATED ADDRES SAME AS OLD D(LL) WHICH IS VALID
YES
ABORT FETCH (UPDATE DISPLAY COMPLETED)
NO
ESDF SET
DOES MSCW'S LL = LL REG
NO
SEQ ERROR INT.
SH3
OBTAIN SEG. DESC. ADDRESSED BY D0 OR D1(RCW) + PDR
FETCH CODE FROM LOCATION DEFINED BY SEC. DESC BASE + APIR
RETURN
YES
WAS VALUE BIT OF MSCW EQUAL TO 1
YES
NO
NO
RESUME EXECUTION OF INTERRUPTED VALC OPERATOR
OPERATION COMPLETED

Figure III-3-9. Flow Chart of Exit and Return Operator (Sheet 3 of 3)

FIELD INSERT (INSR) (P)9C

The Field Insert operator inserts a field from the top-of-stack into the second word. Stack adjustment assures that the top two positions are occupied. The right-justified field in the top-of-stack is inserted into the second word starting at the position specified by the second syllable of the Field Insert operator. The third syllable specifies the length of the field to be inserted. The top-of-stack word is deleted after the field is inserted in the second word.

DYNAMIC FIELD INSERT (DINS) (P)9D

The Dynamic Field Insert operator performs a Field Insert operation, transferring a field from the top operand in the stack into the fourth operand in the stack. The second operand in the stack specifies the length of the field to be inserted, and the third operand in the stack specifies the starting bit of the field.

TYPE-TRANSFER OPERATORS

Type-Transfer operators are used to manipulate operand relative to single-precision or double-precision operands.

SET TO SINGLE-PRECISION, TRUNCATED (SNGT) (P)CC

The Set to Single-Precision, Truncated operator sets the top-of-stack operand to a single-precision operand without rounding.

SET TO SINGLE-PRECISION ROUNDED (SNGL) (P)CD

The Set to Single-Precision, Rounded operator sets the top-of-stack operand to a single-precision operand with rounding.

SET TO DOUBLE-PRECISION (XTND) (P)CE

The Set Double-Precision operator sets the top-of-stack operand to a double-precision operand.

SET DOUBLE TO TWO SINGLES (SPLT) (V)43

The Set Double to Two Singles operator splits a double-precision operand into two single-precision operands.

SET TWO SINGLES TO A DOUBLE (JOIN) (V)42

The Set Two Singles to a Double operator joins two single-precision operands to form one double-precision operand.

MISCELLANEOUS PRIMARY MODE OPERATORS

Miscellaneous primary mode operators are those operators which cannot be readily described or grouped with other operators.

ESCAPE TO 16-BIT INSTRUCTION (VARI) (P)95

The Escape to 16-Bit Instruction operator provides transition from the primary mode operators to the variant mode operators, i.e., the first syllable (VARI) indicates that the actual operator is in the second syllable. (Interrupts are not allowed between the VARI syllable and the following syllable.)

READ AND CLEAR OVERFLOW FLIP-FLOP (ROFF) (P)D7

The Read and Clear Overflow Flip-Flop operator places a single-precision operand in the top-of-stack with the least-significant bit set equal to the overflow flip-flop. The overflow flip-flop is reset.

READ TRUE FALSE FLIP-FLOP (RTFF) (P)DE

The Read True False Flip-Flop operator places a single-precision operand in the top-of-stack with the least significant bit set equal to the true/false flip-flop.

SET EXTERNAL SIGN (SXSN) (P)D6

The Set External Sign operator places the operand sign bit of the top-of-stack word into the external sign flip-flop.

STUFF ENVIRONMENT (STFF) (P)AF

The Stuff Environment operator places the current stack number and displacement into the stack number field and displacement field of the top-of-stack IRW. Bit 46 is set to indicate that it is now a stuffed indirect reference word.

UNIVERSAL OPERATORS

The operators NOOP, HALT, and NVLD are universal except that they cannot follow operators EXSU, EXSD, EXPU, and EXPD; in these cases a Loop Timeout will occur.

CONDITIONAL HALT (HALT) (U)DF

The Conditional Halt operator halts the processor if the conditional halt switch is in the ON position; if the conditional halt switch is OFF, the operator is treated as a NOOP.

INVALID OPERATOR (NVLD) (U)FF

The Invalid Operator sets the invalid-operator interrupt.

NO OPERATION (NOOP) (U)FE

No operation occurs when the No Operation operator is encountered except that the PSR and PIR are advanced to point at the next operator.

VARIANT MODE OPERATORS

Variant mode operators is the name used to describe those primary mode operators which are less frequently used. There is no functional significance to the category "variant mode." Variant mode operation extends the number of operation codes. Variant mode operators require two syllables: the first syllable is the Escape to 16 Bit Instruction (VAR1) operator: The syllable following VARI is the actual operation and the syllable pointer is positioned beyond the two syllables.

Variant mode codes VEO thru VEF are detected and cause a programmed operator interrupt. All other unassigned variant mode codes cause no action and result in a loop timer interrupt.

Variant mode operations are both word-and string-oriented operators.

SCAN OPERATORS

SCAN IN (SCNI) (V)4A

The Scan-In operator uses the 20 low-order bits of the top-of-stack word as the address of the Time of Day (TOD) register and reads the TOD contents to the top-of-stack. Other variants, which are valid in the B 6700, produce an invalid operand interrupt in the B 7700. The B 7700 MCP causes the correct B 7700 code to be executed to handle the interrupt.

SCAN WHILE OPERATORS

SCAN WHILE GREATER, DESTRUCTIVE (SGTD) (V)F2

The Scan While Greater, Destructive operator scans the number of characters specified by the second operand in the stack or while the source character is greater than a delimiter. The top-of-stack operand is the delimiter. The third item in the stack is the source pointer. If the second item in the stack is a descriptor, it is used as a source pointer and the length of the character string is set to 1,048,575. All comparisons are binary. When the source is an operand, it must be a single-precision operand.

At the completion of this operator if all the characters have been scanned the true/false flip-flop is set to one. If the scan was stopped by the delimiter test before the end of the string the true/false flip-flop is set to zero.

SCAN WHILE GREATER, UPDATE (SGTU) (V)FA

The Scan While Greater, Update operator performs a Scan While Greater, Destructive operation. At the completion of the operation, the source pointer and count are updated. At the completion of the operation, a count of the number of characters not scanned is placed in the top-of-stack. If all the characters specified by the length field are scanned, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false. The source pointer locates the character which stopped the scan.

SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE (SGED) (V)FL

The Scan While Greater or Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is greater than or equal to the delimiter.

SCAN WHILE GREATER OR EQUAL, UPDATE (SGEU) (V)F9

The Scan While Greater or Equal, Update operator performs a Scan While Greater Than or Equal, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE EQUAL, DESTRUCTIVE (SEQD) (V)F4

The Scan While Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is equal to the delimiter.

SCAN WHILE EQUAL, UPDATE (SEQU) (V)FC

The Scan While Equal, Update operator performs a Scan While Equal, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE LESS OR EQUAL, DESTRUCTIVE (SLED) (V)F3

The Scan While Less or Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is less than or equal to the delimiter.

SCAN WHILE LESS OR EQUAL, UPDATE (SLEU) (V)FB

The Scan While Less or Equal, Update operator performs a Scan While Less or Equal, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE LESS, DESTRUCTIVE (SLSD) (V)F0

The Scan While Less, Destructive operator performs a Scan While Greater, Destructive operation while the source character is less than the delimiter.

SCAN WHILE LESS, UPDATE (SLSU) (V)F8

The Scan While Less, Update operator performs a Scan While Less, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE NOT EQUAL, DESTRUCTIVE (SNED) (V)F5

The Scan While Not Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is not equal to the delimiter.

SCAN WHILE NOT EQUAL, UPDATE (SNEU) (V)FD

The Scan While Not Equal, Update operator performs a Scan While Not Equal, Destructive operation. At the completion, the source pointer and count are updated.

SCAN WHILE TRUE, DESTRUCTIVE (SWTD) (V)D5

The Scan While True, Destructive operator uses each source character as an index into a table to locate a bit in the table. In order to index the table the source character is expanded to eight bits (if necessary) by appending two or four leading-zero bits. The three high-order bits of these eight select a word from the table, indexing the table pointer. The remaining five bits of the expanded source character select a bit from this word by their value. If the bit located is a one, the relationship is true and the scan continues. An all zero's character indexes to the most significant bit of the table.

The top-of-stack word is a table pointer. The second item in the stack specifies the number of characters to be scanned or, if it is a descriptor, it is used as a source pointer and the length of the character string is set at 1,048,575. The third item in the stack is the source pointer. If all the characters specified by the length field are scanned, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false. At the completion of the operation, a count of the number of characters not scanned is placed on the top of stack. The table format is as follows:

Source Size	Table Length	Bits/Word
4	1 word	(31:16)
6	2 words	(31:32)
8	8 words	(31:32)

SCAN WHILE TRUE, UPDATE (SWTU) (V)DD

The Scan While True, Update operator performs a Scan While True, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE FALSE, DESTRUCTIVE (SWFD) (V)D4

The Scan While False, Destructive operator performs a Scan While True, Destructive operation except that the relationship is true if the bit found by indexing into the table is a zero.

SCAN WHILE FALSE, UPDATE (SWFU) (V)DC

The Scan While False, Update operator performs a Scan While False, Destructive operation. At the completion of the operation, the source pointer and count are updated.

TAG FIELD OPERATORS

SET TAG FIELD (STAG) (V)B4

The Set Tag Field operator sets the tag field (bits 50:3) of the second word in the stack to the contents of bits 2:3 of the top-of-stack word.

READ TAG FIELD (RTAG) (V)B5

The Read Tag Field operator replaces the top-of-stack word with a single-precision operand with bits 2:3 equal to the tag field of the original top-of-stack word.

SET INTERVAL TIMER (SINT) (V)45 (CONTROL STATE OPERATOR)

The Set Interval Timer operator integerizes the top-of-stack operand. If the operand cannot be integerized, an integer-overflow interrupt is set and the operation is terminated. The value of the 11 low-order bits of the top-of-stack operand is used to set the interval timer associated with the processor which is executing this operator. Once set, the interval timer will start to decrement once each 512 microseconds. The associated processor is interrupted when the value has been counted to zero if the timer is still armed.

The interval timer is disarmed whenever the associated processor is interrupted by an external interrupt.

READ PROCESSOR IDENTIFICATION (WHOI) (V)4E

The Read Processor Identification operator places a single-precision operand with a value equal to the processor's number on the top-of-stack.

ENABLE EXTERNAL INTERRUPTS (EEXI) (V)46

The Enable External Interrupts operator allows this processor to respond to external interrupts.

DISABLE EXTERNAL INTERRUPTS (DEXI) (V)47

The Disable External Interrupts operator prohibits this processor from responding to external interrupts.

IDLE UNTIL INTERRUPT (IDLE) (V)44

The Idle Until Interrupt operator suspends program execution by this processor. External interrupts are allowed, and the processor will enter its interrupt-handling routine upon receipt of an interrupt.

READ PROCESSOR REGISTER (RPRR) (V)B8

The Read Processor Register operator reads into the top-of-stack the contents of one of the eight base registers, or one of the eight index registers, or one of the 32 D registers. Register address assignments are as follows:

Add. (dec.)	Add. (hex)	Reg. Name	Register Usage
0-31	0-1F		Display Registers
32	20	PIR	Program Index
33	21	SIR*	Source Index
34	22	DIR*	Destination Index
35	23	TIR*	Table Index
36	24	LOSR	Limit of Stack
37	25	BOSR*	Base of Stack
38	26	F	Most Recent MSCW Address
39	27	S1LS*	Scratch (Spare Local Storage)
40	28	ID	Interrupt Identifier
41	29	SCAN*	MDP Control Register
42	2A	IMR	Interrupt Mask Register
43	2B		Spare
44	2C	IFR	Interrupt Fault Register
45	2D		Spare
46	2E	INT	Interval Timer
47	2F	TOD	Time of Day
48	30	PBR*	Program Base Register
49	31	SBR*	Source Base Register
50	32	DBR*	Destination Base Register
51	33	TBR*	Table Base Register
52	34	S	Top of Stack Address
53	35	SNR*	Current Stack Vector Index
54	36	PDR*	Current Segment Descriptor Index
55	37	S2LS*	Scratch (Spare Local Storage)
56	38	ADZ*	Alternate [D0] Register
57	39	APIR*	Alternate Program Index
58	3A	ALL1*	All ones
59	3B	LD1*	Last D [1] used as SD1 base.
60	3C	PFR	Processor Fail Register
61	3D	PMR	Processor Mode Register
62	3E	PGAM	Purge Associative Memory
63	3F	PGKA	Purge Stack and Associative Memory

*Local Memory Register

An invalid-operator interrupt is set and the operation is terminated if the top-of-stack word is not a descriptor or an indirect reference word at the start of the Evaluate operator.

SET PROCESSOR REGISTER (SPRR) (V)B9

The Set Processor Register operator sets the processor register addressed by the second word in the stack to the value contained in the top-of-stack word. On every SPRR the contents of the stack buffer are purged and stored in main memory.

UNPACK OPERATORS

UNPACK ABSOLUTE, DESTRUCTIVE (UABD) (V)D1

The Unpack Absolute, Destructive operator unpacks a string of left-justified digits from the second operand in the stack. The top-of-stack operand defines the string length (in 4-bit digits) of the second operand in the stack. The specified number of digits are transferred from the second operand to the destination. The third item in the stack is a string descriptor destination pointer. Zone fill in the destination is as follows:

1. If the destination bit format is 8-bit ASCII, the digits are transferred to the destination string with the leading-zone bits set to 0011.

2. If the destination bit format is 6-bit BCL, the digits are transferred to the destination with the two leading-zone bits set to zero.

3. If the destination bit format is 8-bit EBCDIC, the digits are transferred to the destination string with the four leading-zone bits set to one.

4. If the destination character size is 0, it is set to 6 and the digits are transferred to the destination string with the two leading-zone bits set to zero (BCL).

UNPACK ABSOLUTE, UPDATE (UABU) (V)D9

The Unpack Absolute, Update operator performs an Unpack Absolute, Destructive operation. At the completion of the operation the destination pointer is updated.

UNPACK SIGNED, DESTRUCTIVE (USND) (V)D0

The Unpack Signed, Destructive operator performs an Unpack Absolute, Destruction operation except that the external sign is considered.

If the external sign flip-flop is ON (indicating negative data) then a zone of 10 is inserted in the last 6-bit character or a zone of 1101 is inserted in the last 8-bit byte. For 8-bit ASCII formatted data the negative sign is indicated in the least-significant byte by a zone of 1111. If the data format of the destination is 4 bits, the first digit position of the destination string is set to 1101 if the external sign flip-flop is ON; if the external sign flip-flop is OFF the first digit of the destination string is set to 1100.

UNPACK SIGNED, UPDATE (USNU) (V)D8

The Unpack Signed, Update operator performs an Unpack Signed, Destructive operation. At the completion of the operation the destination pointer is updated.

LINKED LIST LOOKUP (LLLU) (V)BD

The Linked List Lookup operator searches a linked list of words.

This operator expects the third stack entry (bits 27:28) to contain an argument, the second stack entry to contain a non-indexed data descriptor, and the top-of-stack to contain an operand index value pointing into a linked-list of words. The argument is not required to be an integer, but only the right-most 28 bits are significant after the argument has been integerized as required. The base address, size field, and argument are saved throughout the operator.

The word addressed by the base plus the index value is read into local storage. Bits 47:28 are compared to the argument value. If the argument of the linked-list word is less than the argument value, this process is repeated using the link as the new index. If the linked-list argument is greater than or equal to the argument value, the operation is complete. At completion the top-of-stack register contains an index which points to the link that points to the satisfying argument.

If the value of the link portion of the linked-list word is equal to zero, the top-of-stack register is set to minus one (–1) and marked full as the operation is completed.

If the index value in the linked-list word is greater than the length value from the descriptor, an invalid-index interrupt is set and the operation is terminated.

When the first word in the stack at the start of this operator is not an operand an invalid-operand interrupt is set and the operation is terminated.

If the data descriptor has been indexed, the invalid-operand interrupt is set and the operation is terminated.

MASKED SEARCH FOR EQUAL (SRCH) (V)BE

The Masked Search for Equal operator searches a data word list for a word identical to the third word in the stack. At the beginning of this operator, the top word in the stack contains a data descriptor, the second word in the stack contains a 51-bit mask, and the third word in the stack contains a 51-bit argument value. If the descriptor is not present, the presence-bit interrupt is set and the operator is exited. Otherwise, if the descriptor is indexable (i.e., bit 45 equals zero), the indexed bit (bit 45) is turned ON and the length/index field value is decreased by 1.

The descriptor points to a word which is then fetched into the processor. This word is ANDed with the mask and a test is made to determine whether the result is identical to the argument.

When an equal compare is made, the second stack register is marked empty; the top-of-stack contains an index which gives the address of the last word inspected.

When a not-equal compare is made, the index value is decreased by one and the operation is repeated (except when the index value is zero). When the index value is zero, the top-of-stack register is set to –1 and marked full, the second stack register is marked empty, and the operator is exited.

MOVE TO STACK (MVST) (V)AF

The Move to Stack operator causes the processor's environment (or addressing space) to terminate and to be moved from the current stack to the program stack specified by the operand in the top of stack.

The operator builds a top-of-stack control word and places it at the base of the current stack as addressed by the base-of-stack register.

The operand in the B register is integerized and checked for invalid index against the stack vector. The value in the B register is added to the address field of the stack vector descriptor (at **D[0[+2)**, to address the descriptor for the new stack.

The data descriptor for the requested stack is accessed. If its presence bit is ON, the address field is placed into the base-of-stack register. The top-of-stack control word is brought up and the stack is marked "active" by storing the processor ID at the base of the stack. The TSCW is distributed and the D registers are updated.

If during the integerization the operand in the B register is too large, the integer-overflow interrupt is set and the operation is terminated. The stack buffer is purged on every execution of MVST.

If the index value is less than zero or greater than the length field of the data descriptor for the stack vector array, an invalid index interrupt is set and the operation is terminated.

OCCURS INDEX (OCRX) (V)85

The Occurs Index operator is used to index a field in an array. This operator requires an Occurs Index Word (OIW) in the top-of-stack and an index value (operand) in the second stack position. The format of the IOW is shown below.

	47	43	39	35	31	27	23	19	15	11	7	3
O 50	LENGTH 46	42	38	34	SIZE 30	26	22	18	OFFSET 14	10	6	2
O 49	45	41	37	33	29	25	21	17	13	9	5	1
O 48	44	40	36	32	28	24	20	16	12	8	4	0

41062

The operator creates a new index value from the OIW and the operand in the following manner:

The operand is integerized. If the resulting index is greater than the maximum integer value (549,755,813,887), the integer overflow interrupt is set and the operation is terminated. If either the OIW or the index has a value of zero, or if the index is less than zero or greater than the SIZE field of the OIW, the invalid index interrupt is set and the operation is terminated.

The LENGTH field of the OIW is multiplied by the index value ±15:161 minus 1, and that value is added to the OFFSET field of the IOW, resulting in the new index value. The two original top-of-stack items are deleted and the new index value is left in the top-of-stack.

In the IOW the "length" field gives the number of characters in a field; the "size" field gives the number of fields in the array; the "offset" field indicates the beginning of the first character position in the first field of the first word.

TRANSLATE (TRNS) (V)D7

The Translate operator transfers from the source to the destination the number of characters specified by the second item in the stack while performing the following translation.

The translation uses a table containing the translated characters. The word in the top-of-stack is a descriptor that addresses the translation table. The second operand in the stack specifies the length of the string. The third word in the stack is a descriptor addressing the source string (or an operand which is the source string). The fourth word in the stack is a descriptor addressing the destination string. Source and destination are updated at the end of the operation.

Translation occurs as follows: The specified string character is used as index into the table to locate a character. An all zeroes character locates the most significant character in the table. The located character is transferred to the destination string.

The least significant 32 bits of each table word provide four 8-bit characters. Table sizes are as follows:

a. 4-bit digits provide a 4-word table length.

b. 6-bit characters provide a 16-word table length.

c. 8-bit bytes provide a 64-word table length.

OPERATORS EXCLUSIVE TO THE B 7700

SET MEMORY INHIBITS (SINH) (V)A8 (CONTROL STATE OP)

The Set Memory Inhibits operator transfers the inhibit settings in the second stack register to the memory module specified in the top stack register. The two top-of-stack registers are marked empty. (All tags are legal.) The memory module number is given in the top-of-stack (bits 3:4). The inhibit field setting is given in the second item in the stack (bits 7:8).

SET MEMORY LIMITS (SLMT) (V)AA (CONTROL STATE OP)

The Set Memory Limits operator transfers the limits and availability settings in the second stack register to the memory module specified in the top-of-stack register. The two top-of-stack registers are then marked empty. (All tags are legal.) The limits specify the range of addresses (in 16K increments) behind the module and the availability setting specifies which stack(s) (of a possible four) are to be used. (All tags are legal.) The top-of-stack gives the memory module number (bits 3:4). The second item in the stack gives module availability (bits 3:4) and memory addressing limits: upper limit (bits 15:6) and lower limit (bits 9:6).

FETCH MEMORY FAIL REGISTER (FMFR) (V)AC (CONTROL STATE OP)

The Fetch Memory Fail Register operator fetches the contents of the fail register from the memory module specified in the top-of-stack (bits 3:4). The contents of the fail register are placed in the top-of-stack.

IGNORE PARITY (IGPR) (V)48 (CONTROL STATE OP)

The Ignore Parity operator is used for confidence checking and requires the processor to be in the control state. In control mode 0, words entering the CPM are checked for correct parity but the IGPR operator sets the IGP flip-flop which inhibits transmission of parity error messages for those words with incorrect parity. Likewise, IGPR inhibits correct parity generation before storage for those words detected in the CPM with incorrect parity.

Parity error interrupts and new parity generation will be inhibited with the CPM in control mode 0 by IGPR until any one of the following occurs:

1. Some other interrupt causes the CPM to move to control mode 1.
2. Another IGPR is decoded while the CPM is in a control mode greater than zero.
3. Or the CPM returns to normal state.

Any one of the above conditions cause the IGP flip-flop to be reset and the CPM to resume parity error interrupts and generation of new parity.

PAUSE UNTIL INTERRUPT (PAUS) (V)84

The Pause Until Interrupt operator suspends program execution until an external interrupt or an interval timer interrupt occurs. If the processor is operating in control state, the operation continues in sequence; to clear the interrupt the INT. I.D. must be read. If the processor is operating in normal state, the interrupt is handled as in IDLE.

INTERRUPT CHANNEL N (INCN) (V)8F

The Interrupt Channel N operator sends signals to the channel or channels specified by the top-of-stack. The top-of-stack is then marked empty. Bit 0 interrupts channel 0; bit 1 interrupts channel 1; etc.

STOP (STOP) (V)BF

The STOP operator causes an unconditional halt of the central processor. The STOP operator is primarily used for diagnostic purposes. The processor may be restarted by pressing and releasing the START button on the processor control panel.

EDIT MODE OPERATORS

Edit Mode operators perform editing functions on strings of data. Edit functions are normally involved in preparing information for output. These operators include Insert, Move, and Skip, in the form of micro-operators in either the program string or in a separate table. In the program string, they are single micro-operators and are entered by use of the Execute Single Micro or Single Pointer operators. (See the "Enter Edit Mode Operator" descriptions.) If the micro-operators are in a table, the table becomes the program string that is to be executed. This table is entered by means of the Table Enter Edit operators, and is exited through the End Edit micro-operator.

If the source or destination data has the memory protect bit (bit 48) equal to one, the segmented-array interrupt is set and the current micro-operator is terminated.

INSERT OPERATORS

INSERT UNCONDITIONAL (INSU) (E)DC

The Insert Unconditional micro-operator places an insert character into the destination string for the number of times specified by the repeat value. When this operator is entered by a Table Enter Edit operator, the repeat is in the syllable following the micro-operator syllable, and the insert character is in the next syllable (the third syllable).

When this operator is entered via an Execute Single Micro Instruction operator, the repeat field is in the top-of-stack operand and the insert character is the second syllable. The operator length is then two syllables.

INSERT CONDITIONAL (INSC) (E)DD

The Insert Conditional operator inserts the character defined by the third syllable into the destination string if the float toggle is OFF. If the float toggle is ON, the character defined by the fourth syllable is inserted into the destination string. The insertion is repeated the number of times specified by the second syllable when this operator is entered by the Table Enter Edit operation.

When this operator is entered via an Execute Single Micro Instruction operator, the repeat field is the top-of-stack operand. The operator length is then three syllables.

INSERT DISPLAY SIGN (INSG) (E)D9

The Insert Display Sign operator inserts the character defined by the second syllable into the destination string if the external sign flip-flop is set; otherwise, the character defined by the third syllable is inserted.

INSERT OVERPUNCH (INOP) (E)D8

The Insert Overpunch micro-operator places a sign overpunch in the destination string character. If the external sign flip-flop is reset, the operator skips one destination string character. If the external sign flip-flop is set, the zone bits of the destination character are set to 10 for 6-bit data and to 1101 for 8-bit EBCDIC data; the destination pointer is then advanced one character. The zone bits for 8-bit ASCII data are set to 1111.

MOVE OPERATORS

MOVE WITH INSERT (MINS) (E)D0

The Move With Insert micro-operator performs a Move Numeric Unconditional or an Insert operation under control of the float flip-flop.

If the float flip-flop is set, a Move Numeric Unconditional operation is performed. If the float flip-flop is reset and the source character numeric is zero, the character defined by the third syllable is transferred to the destination string. If the float flip-flop is reset and the source character numeric is not zero, then the float flip-flop is set and a Move Numeric Unconditional is performed.

The number of characters transferred from the source string to the destination string is defined by the repeat value. In Table Edit mode the second syllable is the repeat value and the third syllable is the character to be inserted under control of the float flip-flop. In Execute Single Micro mode the repeat field value is in the word in the top-of-stack and the insert character is in the syllable following the micro-operator syllable.

MOVE WITH FLOAT (MFLT) (E)D1

If the float flip-flop is set, the Move with Float operator causes a Move Numeric Unconditional operation to be performed.

If the float flip-flop is reset and the source character numeric is zero, then the character defined by the third syllable is transferred to the destination string.

If the float flip-flop is reset and the source character numeric is not zero, then the float flip-flop is set. If the external sign flip-flop is set, the character defined by the fourth syllable (the second insert character) is transferred to the destination string; otherwise, the character defined by the fifth syllable (the third character) is transferred. Then a Move Numeric Unconditional operator is performed.

In Table Edit mode, the above operation is repeated for the number of characters specified by the second syllable; the third, fourth, and fifth syllables are the insert characters.

When this operand is entered via an Execute Single Micro instruction, the repeat field is the top-of-stack operand. The operand length is then four syllables, three of which contain insert characters.

MOVE CHARACTERS (MCHR) (E)D7

The Move Characters operator transfers the number of characters specified by the second syllable from the source string to the destination string, if the operator is entered this by the Table Enter Edit, Destructive operator.

When this operator is entered via an Execute Single Micro Destructive instruction, the number of characters transferred is specified by the top-of-stack operand. The operator length is then one syllable.

MOVE NUMERIC UNCONDITIONAL (MVNU) (E)D6

The Move Numeric Unconditional operator transfers from the source string to the destination string the number of characters specified by the second syllable. The zones are not transferred but are set to 00 for 6-bit data, to 1111 for 8-bit EBCDIC data, and to 0011 for 8-bit ASCII data.

When this operator is entered via an Execute Single Micro instruction, the number of characters transferred is specified by the top-of-stack operand. The operator length is then one syllable.

SKIP OPERATORS

SKIP FORWARD SOURCE CHARACTERS (SFSC) (E)D2

The Skip Forward Source Characters operator causes a skip forward for the number of source characters specified by the syllable following the micro-operator's syllable, if the entry to this operator is by the execution of the Table Enter Edit operator. When this operator is entered via an Execute Single Micro, Destructive instruction, the number of characters skipped is specified by the top-of-stack operand. The operator length is then one syllable.

SKIP REVERSE SOURCE CHARACTERS (SRSC) (E)D3

The Skip Reverse Source Characters operator decrements the source pointer register for a skip in reverse for the number of source characters specified by the second syllable.

SKIP FORWARD DESTINATION CHARACTERS (SFDC) (E)DA

The Skip Forward Destination Characters operator causes a skip forward for the number of destination characters specified by the second syllable.

SKIP REVERSE DESTINATION CHARACTERS (SRDC) (E)DB

The Skip Reverse Destination Character operator causes a skip in reverse for the number of destination characters specified by the second syllable.

RESET FLOAT (RSTF) (E)D4

The Reset Float micro-operator sets the float flip-flop to zero.

END FLOAT (ENDF) (E)D5

The End Float operator transfers to the destination string the character defined by the second syllable if the float flip-flop is reset and the external sign flip-flop is set.

If the float flip-flop is reset and the external sign flip-flop is reset, the character defined by the third syllable is transferred to the destination string.

If the float flip-flop is set, the End Float operator is treated as a NO-OP.

END EDIT (ENDE) (E)DE

The End Edit operator terminates the execution of this string of edit micro-operators in Table Enter Edit mode. The micro program string must end with the End Edit operator.

UNIVERSAL OPERATORS

NO OPERATION (NOOP) (U)FE

No operation takes place when the NOOP operator is encountered. The program index register (PIR) and the program syllable register (PSR) are advanced to the next syllable. This operator is valid in Variant Mode and Edit Mode but is not valid in Single Micro mode.

CONDITIONAL HALT (HALT) (U)DF

This operator halts the processor if the conditional halt switch is in the ON position. If the conditional halt switch is OFF, the operator is treated as a NOOP. This operator is valid in Variant Mode and Edit Mode but is not valid in Single Micro mode.

INVALID OPERATOR (NVLD) (U)FF

This operator sets the invalid operand interrupt.

VECTOR MODE OPERATORS

Certain scientific and mathematical analysis involves manipulation of vectors and matrices. Programming wise a vector is a string of numbers which may be a row, a column, or a diagonal in an array of numbers. Operations may be performed on each item in the vector, on multiple vectors, between vectors, on a matrix of numbers within the array, between matrices, etc. Seventy-seven operators are available for vector (arithmetic, logical, relational, etc.) manipulations. (The two operators (VMOS and VMOM) used to enter vector mode operation are described in the primary mode section of this document.)

In vector mode, the source, destination, and table pointer areas are used as index registers. Absolute addresses and their corresponding increments of items in the vector are extracted from the stack by VMOS or VMOM and placed in the index registers. The stack initially contains three descriptors and three or four operands: If bit 44 of the descriptor in the top of erand and is stored as the repeat count; otherwise, a default repeat of 1,048,575 is used. Each of the three descriptors represents a full or partial array of operands to be operated upon by the repetition of the word of code.

Vector mode operation is entered via the VMOS operator for a single-word vector mode operation or via the VMOM operator for multiple-word vector mode operations. The enter vector operators extract three absolute addresses, their corresponding increment, and an optional length from the stack and place them as index registers in the pointer areas of the IC memory. Vector stack operators load, store, or increment the top-of-stack with a single-precision or a double-precision operand for single program word loops or for multiple program word loops.

The number of program iterations is indicated by "length." Following the enter vector mode instruction, up to five syllables are ignored and the next full program word is fetched into the P register. If a repeat (length) is specified, this word will be executed that number of times. (If the repeat is not given, the execution is controlled by the operator itself.) In VMOM the number of program operators accessed is controlled by the operators Vector Branch and Vector Exit. Only those operators listed below (by mnemonic) may be used to manipulate vectors or matrices in vector mode.

Operator Type	Mnemonics of Operators Permitted in Vector Mode
Arithmetic	ADD DIVD IDIV MULT MULX NTGD NTGR NTIA RDIV SUBT
Bit	BRST BSET CBON CHSN DBRS DBST LOG2
Literal Call	ONE LT8 LT16 LT48 MPCW ZERO
Logical	LAND LEQV LNOT LOR
Relational	EQUL GREQ GRTR LESS LSEQ NEQL SAME
Scale	DSLF DSRF DSRR DSRS DSRT SCLF SCRF SCRS SCRT
Stack	DLET DUPL EXCH PUSH RSDN RSUP
Type Transfer	JOIN SNGL SNGT SPLT XTND
Transfer	DFTR DISO DINS FLTR INSR ISOL
Branch	BRFL BRTR BRUN
Unpack	LLLU
Tag Field	OCRX RPRR RTAG SINT SPRR STAG WHOI
Index Load	LODT
Universal	HALT
Miscellaneous	ROFF RTFF SXSN VARI
Primary Mode Operators Exclusive to the B 7700	DADD FMFR IGPR INCN MAX MIN PAKL PAKR RDEF ROD2 ROD1 UPKL UPKR

VECTOR BRANCH (VEBR) (Z)EE

Vector Branch is a three-syllable operator. The two syllables following the operator name contain the branch address. Vector Branch examines length. If it is greater than zero, length is decremented by one, the next two program syllables containing the branch address are skipped, and the program is resumed at the following syllable. If the examined length is zero, vector mode is exited, and normal operation commences with the program word located by the branch address.

VECTOR EXIT (VXIT) (Z)E6

The Vector Exit operator causes the program to return to normal operation.

VECTOR STACK OPERATORS

Vector stack operators store or load the top of stack from absolute memory addresses with a single-precision or double-precision operand, and, if specified, increment the loading or storing address.

LOAD A (LDA) (Z)E0

The stack is adjusted (0,2) and the single-precision word selected by Pointer A is loaded into the top of stack.

LOAD B (LDB) (Z)E2

The stack is adjusted (0,2) and the single-precision word selected by Pointer B is loaded into the top of stack.

LOAD C (LDC) (Z)E4

The stack is adjusted (0,2) and the single-precision word selected by Pointer C is loaded into the top of stack.

LOAD A, INCREMENT (LDAI) (Z)E1

The stack is adjusted (0,2) and the single-precision word selected by Pointer A is loaded into the top of stack. Pointer A is increased by its increment following the transfer.

LOAD B, INCREMENT (LDBI) (Z)E3

The stack is adjusted (0,2) and the single-precision word selected by Pointer B is loaded into the top of stack. Pointer B is increased by its increment following the transfer.

LOAD C, INCREMENT (LDCI) (Z)E5

The stack is adjusted (0,2) and the single-precision word selected by Pointer C is loaded into the top of stack. Pointer C is increased by its increment following the transfer.

STORE A (STA) (Z)F0

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer A.

STORE B (STB) (Z)F2

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer B.

STORE C (STC) (Z)F4

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer C.

STORE A, INCREMENT (STAI) (Z)F1

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer A. Pointer A is increased by its increment following the transfer.

STORE B, INCREMENT (STBI) (Z)F3

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer B. Pointer B is increased by its increment following the transfer.

STORE C, INCREMENT (STCI) (Z)F5

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer C. Pointer C is increased by its increment following the transfer.

DOUBLE LOAD A (DLA) (Z)E8

The stack is adjusted (0,2) and the double-precision word selected by Pointer A is loaded into the top of stack.

DOUBLE LOAD B (DLB) (Z)EA

The stack is adjusted (0,2) and the double-precision word selected by Pointer B is loaded into the top of stack.

DOUBLE LOAD C (DLC) (Z)EC

The stack is adjusted (0,2) and the double-precision word selected by Pointer C is loaded into the top of stack.

DOUBLE LOAD A, INCREMENT (DLAI) (Z)E9

The stack is adjusted (0,2) and the double-precision word selected by Pointer A is loaded into the top of stack. Pointer A is increased by its increment following the transfer.

DOUBLE LOAD B, INCREMENT (DLBI) (Z)EB

The stack is adjusted (0,2) and the double-precision word selected by Pointer B is loaded into the top of stack. Pointer B is increased by its increment following the transfer.

DOUBLE LOAD C, INCREMENT (DLCI) (Z)ED

The stack is adjusted (0,2) and the double-precision word selected by Pointer C is loaded into the top of stack. Pointer C is increased by its increment following the transfer.

DOUBLE STORE A (DSA) (Z)F8

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer A.

DOUBLE STORE B (DSB) (Z)FA

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer B.

DOUBLE STORE C (DSC) (Z)FC

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer C.

DOUBLE STORE A, INCREMENT (DSAI) (Z)F9

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer A. Pointer A is increased by its increment following the transfer.

DOUBLE STORE B, INCREMENT (DSBI) (Z)FB

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer B. Pointer B is increased by its increment following the transfer.

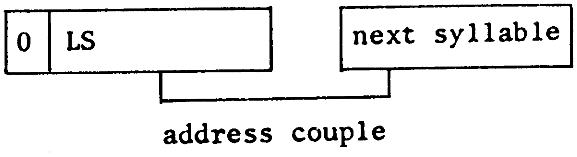
DOUBLE STORE C, INCREMENT (DSCI) (Z)FD

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer C. Pointer C is increased by its increment following the transfer.

VECTOR FETCH AND STORE OPERATORS

The Vector Fetch and Vector Store operators are used in conjunction with addresses relative to a Mark Stack Control Word (only when a "length" is passed) to load or store operands to or from the top-of-stack. The operands are moved to or from the memory location indicated by the normal address-couple decoding convention (as in Value Call) unless the memory location is protected. (An attempt to access a protected memory location causes a vector mode exit and a memory protect interrupt.) A single-precision operand is placed in the top-of-stack position designated A0. A double-precision operand uses both the ^{9}s and the ^{9}q osition of the top-of-stack; the least significant half of the mantissa is placed in the^{9}q position.

The format of the Vector Fetch and Vector operators is:



where, if

LS=0 then load (FTCH operator) or when

LS=1 then store (STOR operator).

Certain precautions must be exercised in using the Fetch and Store operatcrs. It must be assured that:

a. Words being loaded or stored are operands.

b. The precision of the operand to be stored agrees with that of the receiving location.

c. The address couple does not reference a lexicographical level which is higher than the level in which the address couple is now.

VECTOR FETCH (FTCH) (Z)00 THRU (Z)3F

The Vector Fetch operator loads into the top-of-stack the operand referenced by this address couple of this operator. The address couple is formed by the concatenation of the siz low-order bits of the first syllable of this operator with the eight bits of the second syllable of this operator. The address couple (in the stack) references a memory storage location relative to an MSCW. The operand may be single precision or double precision, thus, one or two words will be loaded, respectively.

After the operand is brought to the top-of-stack, the top-of-stack is marked full.

VECTOR STORE (STOR) (Z)40 THRU (Z)7F

The Vector Store operator stores single-precision or double-precision operands from the top-of-stack into a memory area relative to an MSCW. The least significant six bits of the first syllable of the STOR operator are concatenated with the eight bits of the second syllable of the STOR operator to form an address couple which links the storage area with the MSCW.

After the operand is stored, the operand and its address are deleted from the stack.

CHAPTER IV

INPUT/OUTPUT SUBSYSTEM

SECTION 1

GENERAL DESCRIPTION AND OPERATION OF THE INPUT/OUTPUT MODULE

PRELIMINARY

The B 7700 Input/Output Module, is designed to serve as a buffer and control unit for all B 7700-system input and output data transfers. The IOM services requestors from a queue of requests constructed by the Central Processing Module (CPM) and stored in the Memory Storage Unit (MSU).

The IOM is informed, via an interrupt from the CPM, of the presence of a service request in the MSU. Once informed, the IOM controls the desired input/output operation in its entirety; thus, the CPM time required to initiate an I/O operation is only that needed to construct a request, queue it in the MSU, and interrupt the IOM.

BASIC IOM CONFIGURATION

As illustrated in figure IV-1-1, the IOM consists of six major subsections. Each subsection is totally independent of the other subsections, and operates asynchronously with them.

CONTROL WORD FLOW

All control word flow between main memory and up to 255 system peripherals is via (1) an IOM subsection called the Memory Interface Unit (MIU), (2) an IOM control subsection called the Translator (XLATOR), and (3) one of four IOM subsections, each of which is uniquely buffered to match the class of data transfer assigned to it. The XLATOR subsection routes control of a given job request to one of these subsections dependent upon data class (batch, high speed, data communications, or real-time interactive).

DATA FLOW

All data flow between main memory and the peripherals is via the appropriate data-transfer subsection and/or the MIU; the XLATOR is not involved and is free for control of additional job requests. When a data transfer is complete, however, the XLATOR is given control over job termination, and control flow to main memory is via the appropriate data-transfer subsection, the XLATOR, and the MIU.

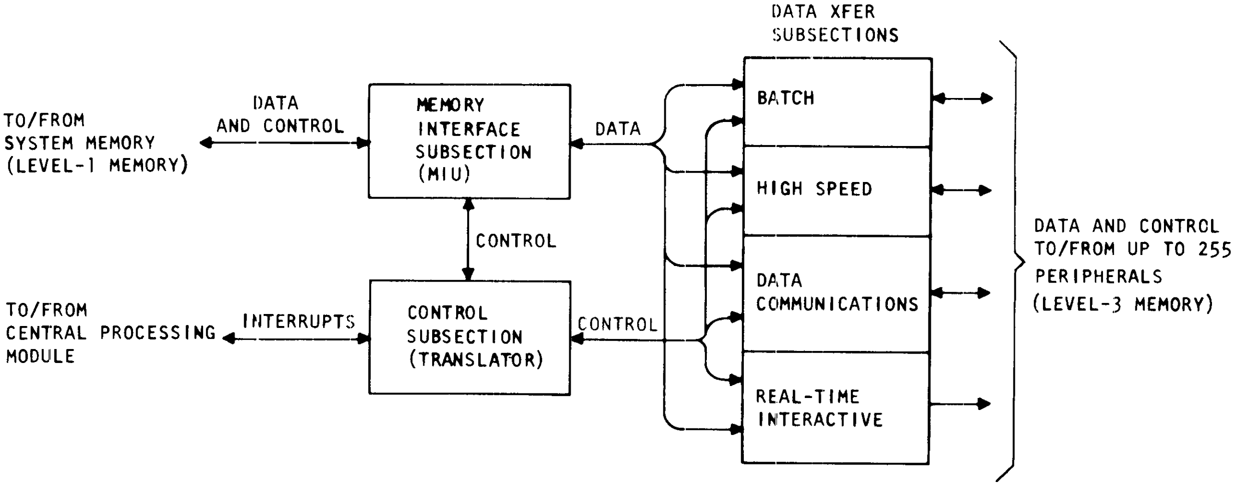

DATA XFER SUBSECTIONS
TO/FROM SYSTEM MEMORY (LEVEL-1 MEMORY)
DATA AND CONTROL
MEMORY INTERFACE SUBSECTION (MIU)
DATA
BATCH
HIGH SPEED
CONTROL
DATA COMMUNICATIONS
TO/FROM CENTRAL PROCESSING MODULE
INTERRUPTS
CONTROL SUBSECTION (TRANSLATOR)
CONTROL
REAL-TIME INTERACTIVE
DATA AND CONTROL TO/FROM UP TO 255 PERIPHERALS (LEVEL-3 MEMORY)

Figure IV-1-1. IOM Basic Block Diagram

IOM/PERIPHERAL INTERFACE CONFIGURATION

Figure IV-1-2 illustrates typical peripheral devices which may be assigned to each data-transfer class; also illustrated are the data-transfer subsection names which are henceforth referred to. The following is a brief description of the interface capability of each subsection, and its physical relationship to typical peripheral equipment. The descriptions are presented in reference to figure IV-1-3, which illustrates the interface capability provided when two maximum-configuration Input/Output Modules and appropriate exchanges are used. It should be noted that a maximum of 28 peripheral controllers (excluding DFO's and DCP's) may be connected to a single IOM.

PERIPHERAL CONTROL INTERFACE (PCI)

The PCI of a single IOM consists of either one or two interface sections, dependent upon user requirements. Each section has 10-channel interface capability, for a total maximum capacity of 20 channels per IOM.

The controls serviced by a PCI are housed in one or two peripheral control cabinets (PCC's). Each PCI/PCC cabinet services one IOM and has a 10-channel interface capability. Up to five of these channels can be assigned to large controllers, but the remaining channels must be assigned to small controllers. Table IV-1-1 lists the various controllers which may be housed in PCI/PCC cabinets.

Any combination of small controls may be housed in the PCI/PCC cabinet. The large controls (SLC and MTC) may be connected to the peripheral units directly, or, in the case of the MTC only, via exchanges. Any unused channels in the PCC cabinet are left empty.

The PCI multiplexes all 20 channels by generating overlapping 1-micro-second data-service cycles and by use of "windows" in a self-contained local memory. In the typical configuration illustrated in figure IV-1-3, the use of two IOMs and appropriate exchanges (4X16) allows access by either IOM of 64 magnetic tape units. IOM number 1 is illustrated as having access to an additional non-exchange magnetic tape unit, and both IOM's are illustrated as having access to SPO units via Single Line Controls (SLC).

DISK FILE INTERFACE (DFI)

The DFI of a single IOM consists of either one or two interface sections, dependent upon user requirements. Each section has an interface capability of four channels, for a total disk-file-channel capability of eight channels per IOM.

Each DFI four-channel section services a single DFI/PCC cabinet. This cabinet contains only large channels (four maximum), which are dedicated to either disk files or disk packs. The channels may be connected to the peripherals either directly or via exchanges. In the typical configuration illustrated in figure IV-1-3, the use of two maximum DFI-configuration IOMs (eight channels per IOM, four each disk file and disk pack) and appropriate exchanges (2X20 for disk file, 2X16 for disk pack) allows access by either IOM of 80 disk file electronics units (400 disk file storage units) and 64 disk packs. Table IV-1-2 lists the controllers which may be housed in the DFI/PCC cabinet.

SCAN INTERFACE (SCI)

The SCI consists of two sections: a DFO scan interface and a DCP scan interface. The DFO scan interface provides scan-in and scan-out control for up to four DFO's via a scan bus. If a second IOM is used, the DFO scan bus is shared by the two IOM's (see figure IV-1-3).

The DCP scan interface provides scan-out control only, and may communicate with up to four DCP's via a scan bus. The SCI is not used for DCP scan-in functions, which are initiated by the DCP. For these functions, the DCP communicates with main memory directly via the MIU. The DCP scan bus is not shared by a second IOM.

DATA COMMUNICATIONS PROCESSOR INTERFACE (DCI)

The DCI provides the data and control interface for IOM-initiated scan-out operations, and the data interface only for DCP-initiated scan-in operations. Interface is provided in each IOM for up to four DCP's. As illustrated in figure IV-1-3, the use of two IOM's in a system allows interface with eight DCP's.

IOM/MAIN AND IOM/CPM INTERFACE CONFIGURATIONS

Figure IV-1-4 illustrates a typical interface configuration between the IOM's, MCM's, and CPM's of a typical system. The following is a brief description of the interface capability of the IOM/MIU subsection with main memory and the IOM/XLATOR subsection with system CPM's.

IOM/MCM INTERFACE

As illustrated in figure IV-1-4, the MIU contains eight interface areas. Each interface area is dedicated to a distinct Memory Control Module (MCM), and is connected to it via a unique memory bus. The bussed IOM/MCM interface is referred to as a memory/user pair.

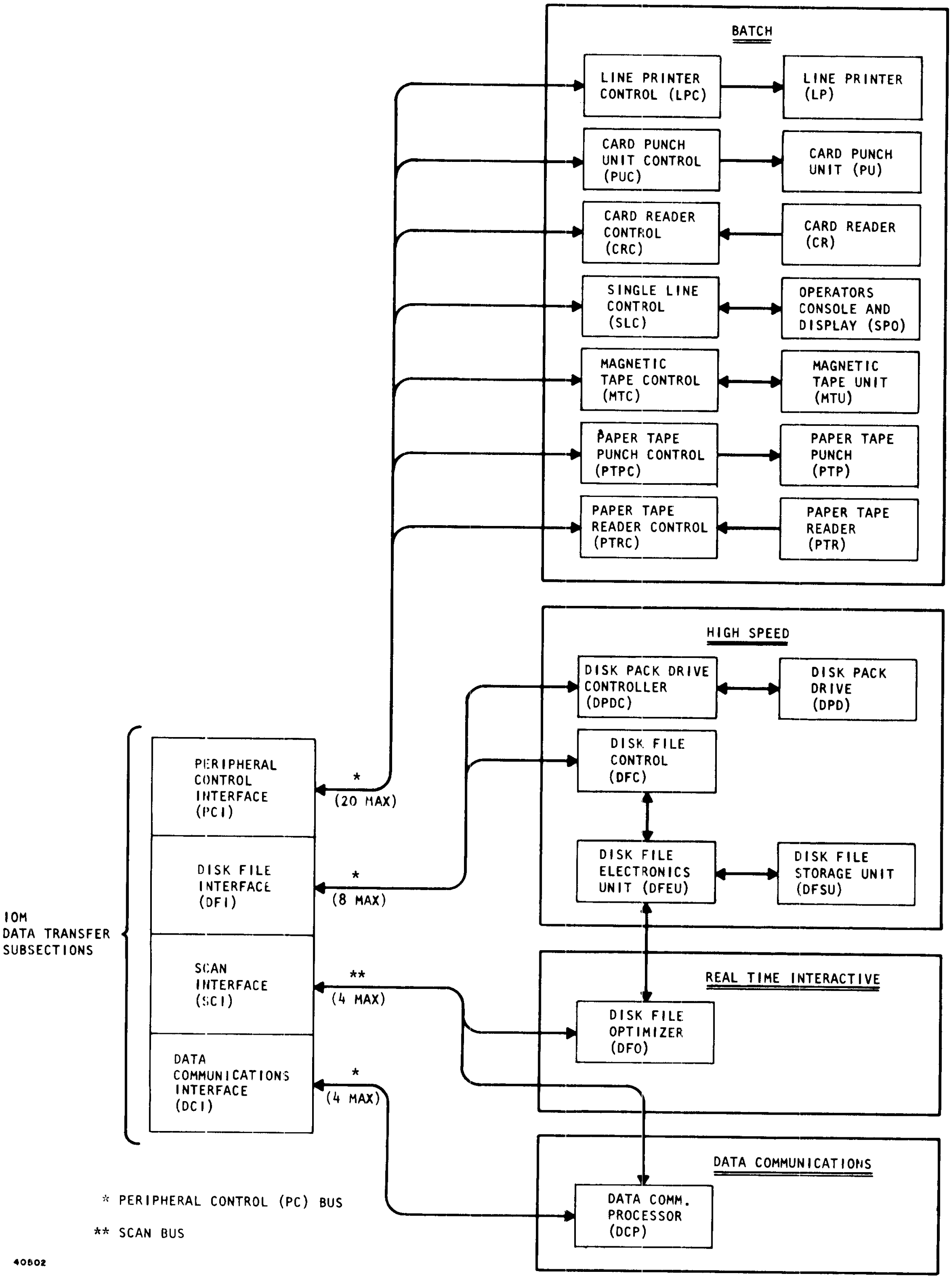

BATCH
LINE PRINTER CONTROL (LPC)
LINE PRINTER (LP)
CARD PUNCH UNIT CONTROL (PUC)
CARD PUNCH UNIT (PU)
CARD READER CONTROL (CRC)
CARD READER (CR)
SINGLE LINE CONTROL (SLC)
OPERATORS CONSOLE AND DISPLAY (SPO)
MAGNETIC TAPE CONTROL (MTC)
MAGNETIC TAPE UNIT (MTU)
PAPER TAPE PUNCH CONTROL (PTPC)
PAPER TAPE PUNCH (PTP)
PAPER TAPE READER CONTROL (PTRC)
PAPER TAPE READER (PTR)
HIGH SPEED
DISK PACK DRIVE CONTROLLER (DPDC)
DISK PACK DRIVE (DPD)
DISK FILE CONTROL (DFC)
DISK FILE ELECTRONICS UNIT (DFEU)
DISK FILE STORAGE UNIT (DFSU)
REAL TIME INTERACTIVE
DISK FILE OPTIMIZER (DFO)
DATA COMMUNICATIONS
DATA COMM. PROCESSOR (DCP)
IOM DATA TRANSFER SUBSECTIONS
PERIPHERAL CONTROL INTERFACE (PCI)
DISK FILE INTERFACE (DFI)
SCAN INTERFACE (SCI)
DATA COMMUNICATIONS INTERFACE (DCI)
* (20 MAX)
* (8 MAX)
** (4 MAX)
* (4 MAX)
* PERIPHERAL CONTROL (PC) BUS
** SCAN BUS
40602

Figure IV-1-2. Typical Data-Transfer Classifications and Related IOM Subsections

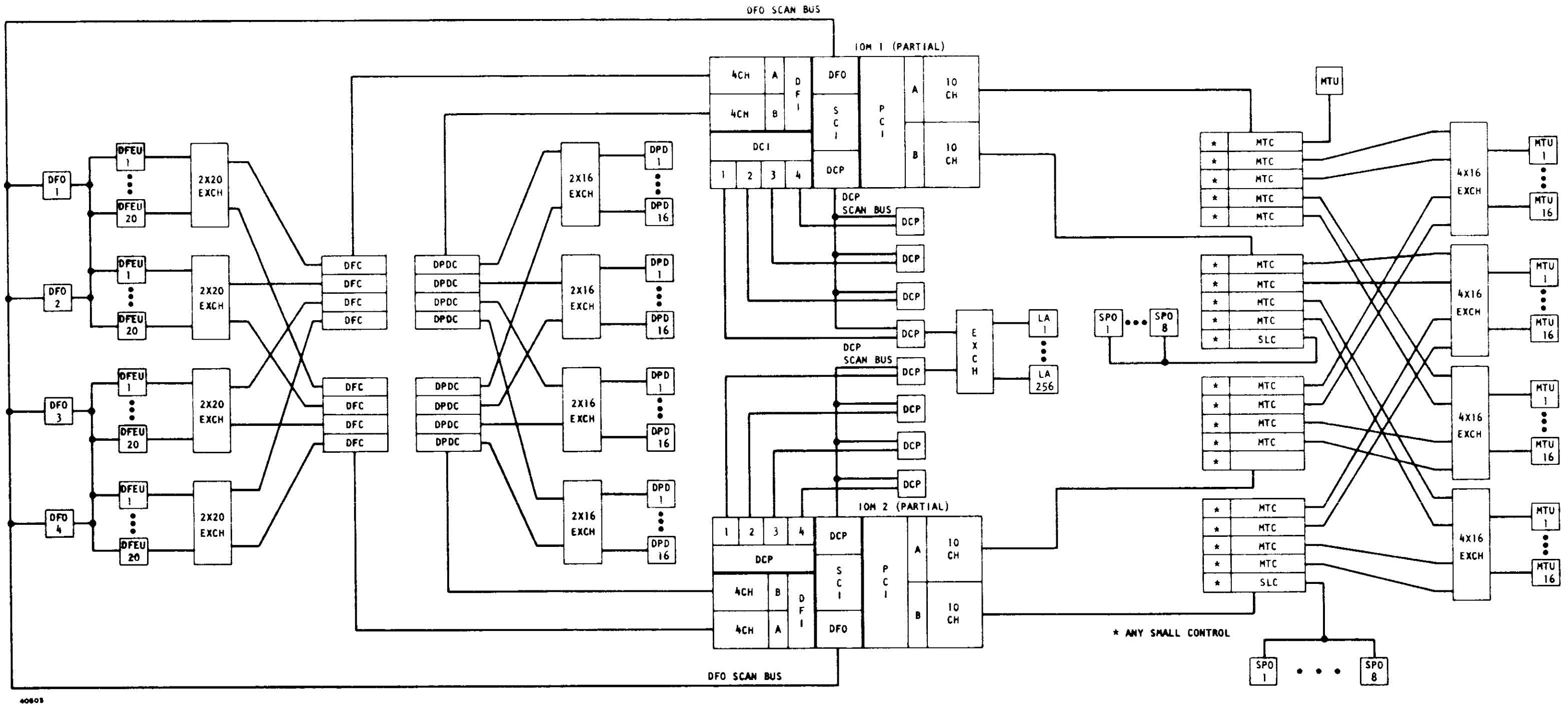


Figure IV-1-3. Typical IOM/Peripheral Configuration

Table IV-1-1. PCI/PCC Channels

	Peripheral Controls		Peripheral Units
Size	**Model**	**Type**	
L	B 7381-11	18/36 KB, NRZ, 9 Ch. MTC	18/36 KB, 9 Ch. NRZ Clusters (800 BPI, 2, 3, or 4 Station)
L	B 7381-12	36/72 KB, PE, 9 Ch. MTC	36/72 KB, 9 Cu. PE Clusters (1600 BPI, 2, 3, or 4 Station)
L	B 7381-14	18/36 KB, NRZ, 9 Ch., Dual MTC with 2x8 Exchange	18/36 KB, 9 Ch., NRZ Clusters (800 BPI, 2, 3, or 4 Station)
S	B 7240 with B 9943	Printer Control Printer Memory	300/400 LPM, 120/132 Print Position Printers
			860 LPM, 120 Print Pos; 725 LPM, OCR "A" and "B" Numeric & Std; 1100 LPM, 120 Print Position; 900 LPM, OCR "A" and "B" Numeric and Std Printers
S	B 7120 and B 9926	Paper Tape Reader Control and Input Code Translator	Paper Tape Reader, 500-1000 CPS
S	B 7220 and B 9928	Paper Tape Punch and Output Code Translator	Paper Tape Punch, 100 CPS
L	B 7381-15	36/72 KB, PE, 9 Ch., Dual MTC with 2x8 Exch.	36/72 KB, 9 Ch. PE Clusters (1600 BPI, 2, 3, or 4 Station)
L	B 7381-16	18/36 KB, 36/72 KB, NRZ/PE, 9 HC., Dual MTC with 2x8 Excm.	18/36 KB, 9 Ch. NRZ/PE Clusters (800/1600 BPI, 2, 3, or 4 Station; 36/72 KB, 9 Ch. NRZ/PE Clusters (800/1600 BPI, 2, 3, or 4 Station)
L	B 7391-3	72 KC, 200/556/800 BPI, 7 Ch MTC	18/50/72 KC, 7 Ch. (200/556/800 BPI) MTU
L	B 7391-4	96 KC, 200/556/800 BPI, 7 Ch MTC	24/66/96 KC, 7 Ch. (200/556/700 BPI) MTU
L	B 7393-1	72 KB, 800 BPI, 9 Ch. MTC	72 KB, 9 Ch., 800 BPI MTU
L	B 7393-2	144/240 KB, 1600 BPI, 9 Ch. MTC	144 KB, 9 Ch., 1600 BPI MTU
L	B 7393-3	96 KB, 800 BPI, 9 Ch. MTC	96 KB, 9 Ch., 800 BPI MTU
L	B 7393-5	320/400 KB, 1600 BPI, 9 Ch. MTC	240 KB, 9 Ch., 1600 BPI, MTU
L		SINGLE LINE CONTROL-1 (SLC-1)	Burroughs Terminal Computer TC500; Burroughs Input and Display Terminal B 9352
L		Single Line Control-2 (SLC-2)	Burroughs Terminal Computer TC500; Input and Display Terminal B 9352; Input and Display System B 9351
S	B 7110	Card Reader Control	800 and 1400 CPM Card Readers
S	B 7212 and B 7610	Card Punch Control and BCL-BCL Code Translator	300 CPM Card Punch

Table IV-1-2. DFI/PCC Channels

Model	Peripheral Controls	Peripheral Units
	Type	
B 7877	Disk File Control IV	IC-3 and IC-4 Disk Files
B 7380-1	Single Dual-Drive Control, Disk Pack	Single Data Access, Dual Drive Disk Packs, 121 Megabyte
B 7380-2	Dual Dual-Drive Control, Disk Pack	Simultaneous Data Access, Dual Drive Disk Packs, 121 Megabyte
B 7383-1	Single Dual-Drive Control, Disk Pack	Single Data Access, Dual Drive Disk Packs, 242 Megabyte
B 7383-2	Dual Dual-Drive Control, Disk Pack	Simultaneous Data Access, Dual Drive Disk Packs, 242 Megabyte

Similarly the Central Processing Module (CPM) also contains eight MCM interface areas. Each CPM interface area is dedicated to a distinct MCM and is connected to it via a unique memory bus. The bussed CPM/MCM interface is also referred to as a memory/user pair.

The interface capability of an MCM is eight memory busses, each of which is connected to one and only one IOM or CPM. Therefore, the maximum combined number of CPM's and IOM's which may be bussed to an MCM is limited to eight.

The maximum number of MCMs which may be contained in a B 7700 system is also limited to eight. This limitation is imposed by the eight MCM-dedicated interface areas of each IOM and CPM in the system.

The typical memory-bus configuration illustrated in figure IV-1-4 indicates the use of two IOM's, two CPM's, and two MCM's. This configuration provides a total of eight memory/user pairs (MCM 0 to users 0 thru 3 and MCM1 to users 0 thru 3). The maximum number of Memory Storage Units (MSU) with which an MCM can communicate (four) is also illustrated. Each of these MCM's can access 262,144 words of memory (4 MSU's of 65,536 words each). Each IOM or CPM, when connected as illustrated on figure IV-1-4, can therefore access 524,288 words of memory.

IOM/CPM INTERFACE

The interface between the IOM's and CPM's of a B 7700 system consists only of an interrupt bus. The XLATOR section of an IOM is informed by the CPM of job requests via the bus, and the XLATOR informs the CPM of non-channel-related IOM errors via the bus. In addition, the XLATOR uses the bus to inform the CPM of (1) I/O job completion when so requested by software, and (2) status change by a single-line control device.

The interrupt bus is common to all IOM's and CPM's in a system, as is illustrated in figure IV-1-4.

IOM OPERATIONAL CHARACTERISTICS

The IOM is designed to operate asynchronously with the CPM in the initiation, service, and termination of input/output transfers by use of a "job map" which is stored in level-1 memory. The "job map" consists basically of five software-constructed elements which define the job request, the peripheral device, and the IOM channel.

In general the map elements inform the CPM of its IOM/Peripheral resources and their status. When necessary, the CPM alters the queued job requests of the "job map" to the extent of its interest and interrupts the IOM to request service. The IOM then accesses the "job map" to determine the input/output job and initiate it. Since the "job map" is a shared resource of the IOM and CPM, the IOM transfer times are masked by the continual processing and queueing of new requests by the CPM; thus maximum system throughput is attained with a minimum of CPM time.

The IOM also manages path selection to the requested device (instead of the programmatic preselection of the path which is generally used). This path management eliminates the occurrence of situations whereby (1) the requested device is free, (2) the preselected path is not, and (3) an alternate path exists but cannot be used due to the programmatic preselection. These situations generally require involvement of the CPM until the preselected path is free and the job is initiated, which effectively reduces the parallelism of the CPM and IOM. Since the IOM manages the path selection in the B 7700 system, CPM involvement regarding job initiation ends when an interrupt is sent to the IOM. The IOM then initiates the job request when the requested device and any path to that device is available.

The design of the IOM incorporates extensive error-detection logic which monitors the flow of control words and data between the IOM and other main-frame modules, within the IOM module itself, and between the IOM module and peripheral devices. Particular emphasis is placed upon preserving the integrity of all memory operations. In general, the error-detection hardware consists of: parity check and generate circuitry; residue check circuitry; circuitry to detect illegal commands, conditions, and control states; and timeout circuitry for memory transfers, scan bus operations, and internal IOM transfers.

IOM JOB MAP

The job map which an IOM accesses from main memory consists of the following five software-constructed elements:

a. Home Address Words (HA)
b. Unit Table Word (UT)
c. I/O Queue (IOQ)
d. I/O Control Block (IOCB)
e. Status Queue (SQ)

The following four level-1 addresses, which are loaded into the IOM XLATOR at initialize time, enable the IOM to service the job map:

a. Home Address
b. UT Base Address
c. IOQ Header (IOQH) Address
d. SQ Header (SQH) Address

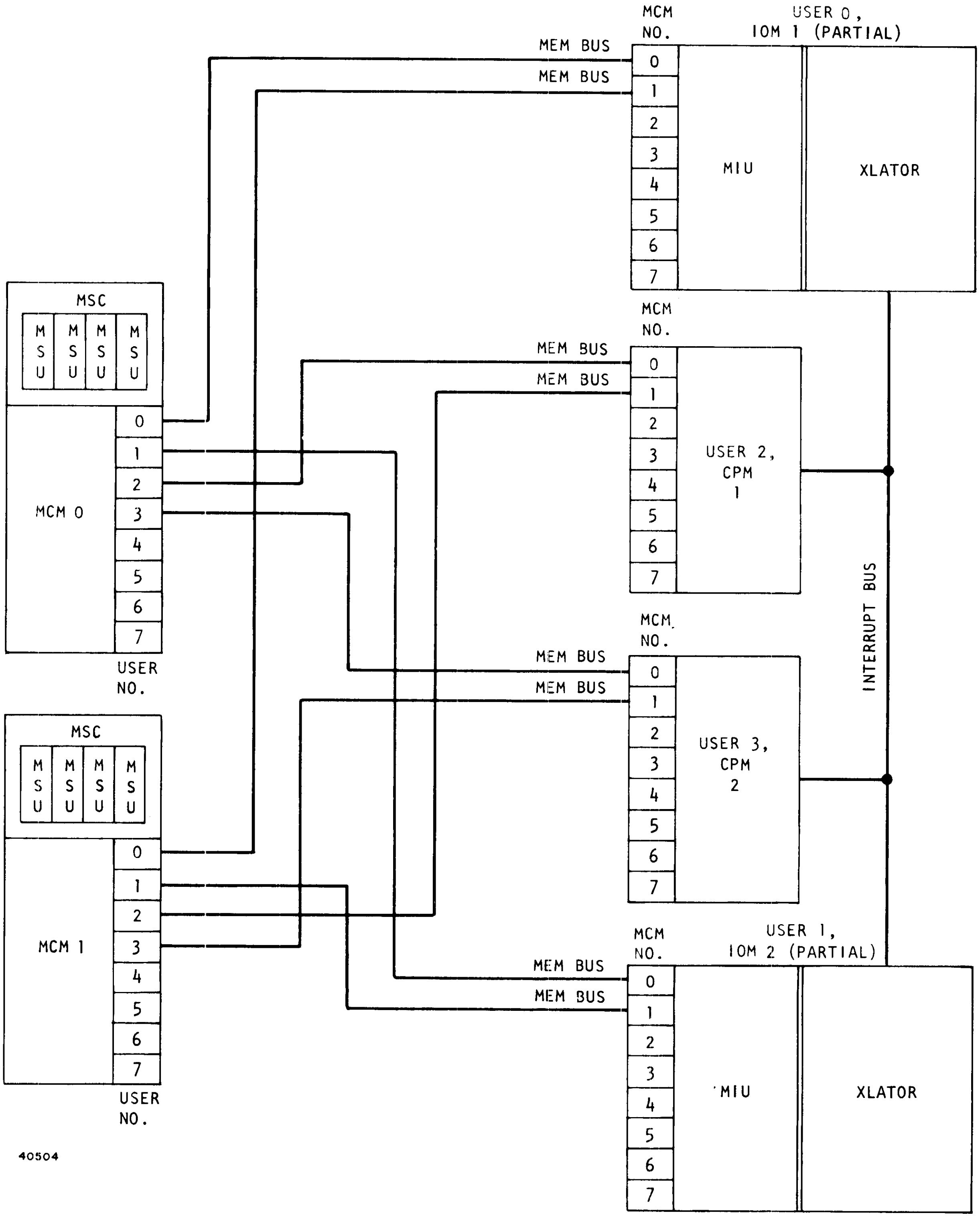


Figure IV-1-4. Typical IOM/Main Memory and IOM/CPM Interface Configurations

By use of these stored addresses and the contents of previously-fetched map elements, job requests originally constructed by the CPM are reconstructed in the IOM and are serviced.

The following basic description of each map element and the sequence in which the job map is serviced is presented in reference to figure IV-1-5. For detailed formats of all words discussed, refer to the appendix of IOM word

HOME ADDRESS WORD

The 51-bit home address word (HA word) is the first map element fetched by the IOM when interrupted by the CPM. It is fetched by use of the pre-loaded home address stored in the IOM XLATOR, and contains information which describes the basic command and, as applicable, information which describes the device or channel to be used.

The command to be performed is defined by a code within the HA word, called the home code. In some instances further definition of the command is provided by additional bits of the HA word. Based on the command decoded, the logic of the IOM is conditioned to perform one of 20 possible control operations. The commands are described under the heading IOM COMMANDS later in this section.

Only one of the 20 commands, the Start I/O command, requires immediate further access of other map elements; however, some scan-out commands require access of a second 51-bit HA word. The Start I/O command is the basic command used to initiate service of new job requests, whereas the remaining commands are provided for either cold start/halt load, scan out control, configuration determination, or diagnostic purposes.

A home address (HA) word which contains a Start I/O home code also contains a unit designate (UD) number. This number specifies the device to be used for the operation, and is part of the information needed to access the remaining map elements.

UNIT TABLE WORD

The unit table (UT) word is the next map element fetched by the IOM in response of a Start I/O command. The fetch is performed by use of the UT address preloaded in the IOM XLATOR and the UD number derived from the HA word. The preloaded address serves as a locator for the unit table, and the UD number serves as an index to a particular word of the unit table.

The unit table consists of 256 words, which are numbered 0 thru 255. Word 0 is reserved for use as a fail UT word, and is accessed when an error occurs which cannot be associated with a specific job request. In this instance, a special UD number (000), called a fail UD number, serves as an index to UT word 0. Each of the remaining 255 UT words is assigned to a unique device, and contains information which defines the device and its assignment within the system.

The device-type and assignment information specifically indicates (1) whether the device is a disk-pack or a magnetic tape unit (2) if the device is a disk file, whether it is under DFO control, (3) whether the device is connected to an exchange, (4) the lowest IOM channel to which the device is connected.

For a device connected to an exchange, the UT word contains additional information for use in IOM device/path management. The devices connected to an exchange are described by a linked list of UD numbers in the next unit on exchange (NUD) fields of their UT words. The number (modulo 4) of the last (highest) IOM channel on the exchange (LCEX) is also indicated. The description of the exchange is complete because (1) all IOM's on an exchange must use the same channels, (2) channels on an exchange must be consecutive, and (3) the largest exchanges serve a maximum of four channels. A bit (called the "job bit" or JB) is set if a job request for an exchange device must be delayed because a path is not currently available.

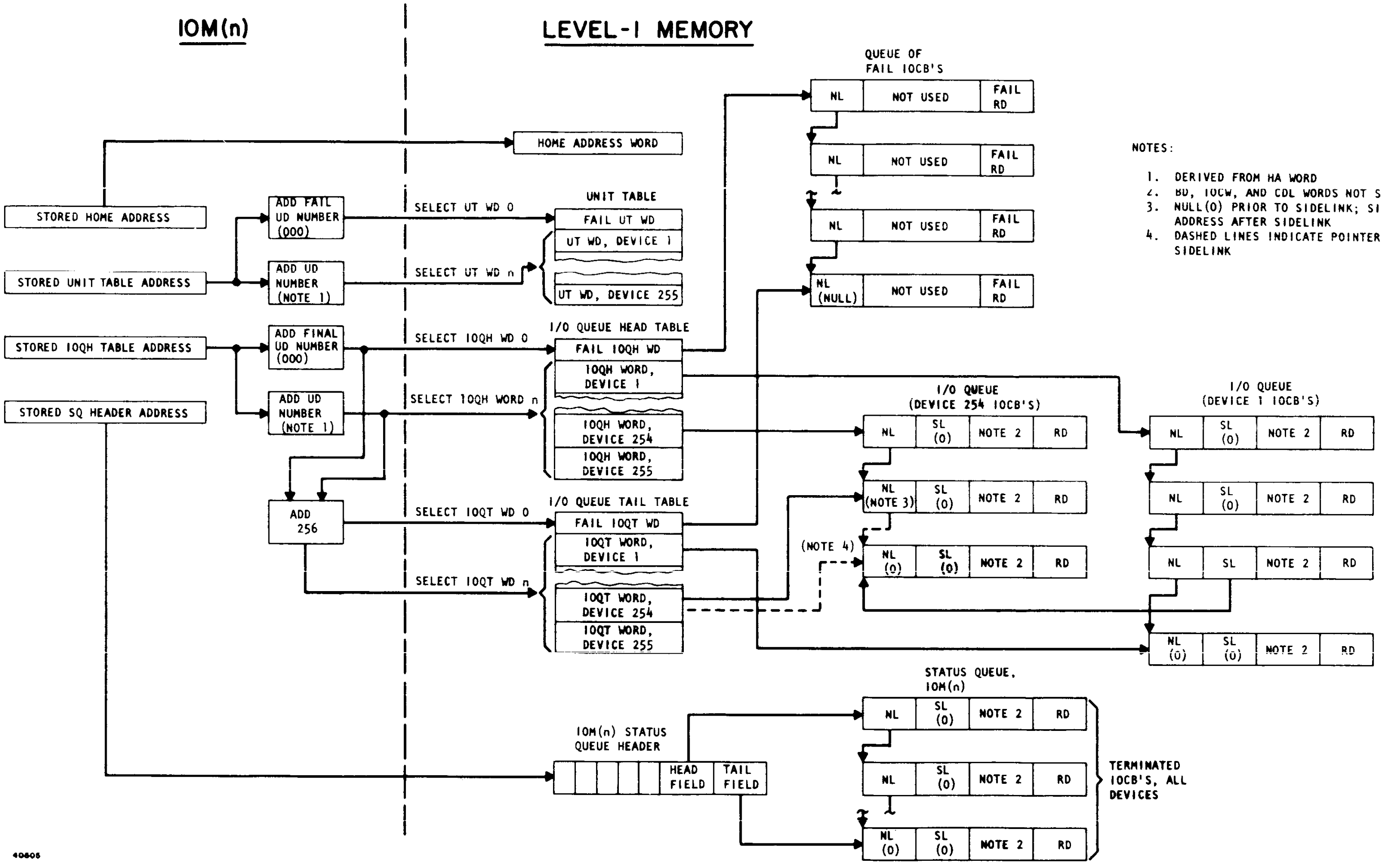
IOQ HEAD (IOQH) AND IOQ TAIL (IOQT) TABLES AND WORDS

The I/O Queue (IOQ), which is constructed by the CPM in main memory, contains linked job requests (I/O Control Blocks) for each device of the system. The extent of the linked job requests for each device is defined by words which indicate the main memory addresses of the first and last of the requests. These words are called the I/O Queue Head (IOQH) word and the I/O Queue Tail (IOQT) word, respectively.

The IOQH words for all devices (255 words) are stored in a table called the I/O Queue Head table; similarly, the IOQT words for the devices (also of 256 words) are stored in a table called the IOQT table, which immediately follows lows the IOQH table in memory.

The IOQH and IOQT tables contain one special word each (word 0) which is reserved for use by the IOM to report errors that cannot be associated with a specific job request. These words are pointers to a list of the fail I/O Control Blocks (fail IOCB's) reserved for failure-reporting by the IOM's of the system (later described).

The IOQT table is the element accessed by the CPM to queue additional requests for a device. The IOM also accesses this element when a sidelink operation to another device is



NOTES:

1. DERIVED FROM HA WORD
2. BD, IOCW, AND CDL WORDS NOT SHOWN
3. NULL(0) PRIOR TO SIDELINK; SIDELINK ADDRESS AFTER SIDELINK
4. DASHED LINES INDICATE POINTERS AFTER SIDELINK

Figure IV-1-5. IOM Job Map

specified. This access is required so that the sidelink operation indicated in the job-request queue of one device may be linked to the queue of job requests for the device designated for the sidelink operation. The IOQT word for the sidelink device is altered to reflect the main memory address of the sidelink job, which becomes the last job queued.

The IOQH table is the element accessed by the IOM in order to service job requests. The IOQH word for a device indicates the main memory address of the first job request for that device. Memory addresses of additional jobs for the device are indicated by the next link (NL) word in each job request, thus linking all job requests for a given device.

As is indicated in figure IV-1-4, the last job request for a device is recognized by the IOM when the next link field of a request is found to contain zeroes (null).

The IOQH word is fetched by use of (1) the IOQH table base address (stored in the XLATOR), and (2) the UD number (derived from the previously-fetched HA word). The UD number indicates which device is to be initiated, and therefore which IOQH word of the IOQH table should be fetched. The UD number is thus an index to the IOQH table.

When a non-request-related error is detected by the IOM and access to the fail IOQH word (word 0) is required, the word is fetched by use of the fail UD number (000) and the IOQH base address. The memory address of the first available fail IOCB, which is contained in the fail IOQH word, is used to fetch the fail IOCB. The NL field contained in the fetched fail IOCB is then used to update the memory address of the fail IOQH word, so that if a second failure is detected, the next fail IOCB of the queue of fail IOCB's can be accessed. The fail IOQT word, which defines the last IOCB in the queue of ten fail IOCB's, is used only by software; it is not accessed by the IOM.

When a sidelink operation requires a fetch of the IOQT word for a device, 256 is added to the IOQH word address (the IOQT-word address for a device designated for a sidelink operation equals the IOQH table base address plus the UD number of the device plus 256).

I/O CONTROL BLOCKS

The job requests for each device are stored in map elements called I/O Control Blocks Each I/O Control Block (IOCB) contains words which are fetched sequentially starting with the memory address obtained from either (1) the IOQH word if the job request is the first for the device, (2) the next link (NL) field of the job request (IOCB) in process if the job is other than the first for that device, or (3) the side link field of the job request (IOCB) in process, if a sidelink (SL) to another device is indicated. The six IOCB words fetched by the IOM are as follows:

a. Next Link (NL) Word
b. Side Link (SL) Word
c. Buffer Descriptor (BD) Word
d. I/O Control Word (IOCW)
e. Channel Designate Level (CDL) Word
f. Result Descriptor (RD) Word

As previously indicated, the NL word contains the address of the next IOCB for a device, and is the means whereby job requests for a device are linked within the IOQ. When this word contains all zeroes (null), it indicates the request being serviced is the last currently enqueued for the device.

The SL word is used to indicate that a sidelink operation (the service of a job request by a device other than that presently being serviced, without intervention by the CPM) is required. The SL word contains the memory address of the sidelink job (IOCB), which is started immediately if no other jobs are queued for the designated sidelink device. If other jobs are queued for the device, the sidelink job is linked to the queue of job requests by insertion of the sidelink memory address in both the IOQT word for the sidelink device and the NL field of the last IOCB previously queued for that device.

The BD word contains the address of the first data location in memory, and the length of the memory area in words.

The IOCW contains the control information necessary to perform the input/output operation, such as read or write, whether code translation is necessary, backward/forward (tape), frame length (6 or 8 bit), etc. The contents of the IOCW and the BD word are used to format the first job word sent to the selected IOM channel.

The CDL word is used to format the second job word sent to the selected IOM channel. This word generally contains information such as the OP code, the device number, the device variant, and for disk, the segment address.

The RD word is used for storage of the termination status of each request. The RD word is built by the IOM, which then links the terminated request (terminated IOCB) into the status queue.

FAIL I/O CONTROL BLOCKS

A queue of special IOCB's, which are not related to job requests, is also built in memory. These IOCB's, which are called fail IOCB's, are used by the IOM's of the system to reporting errors which cannot be associated with a specific request. The fail IOCB's contain the same six words as job-request IOCB's; however, only the next link word and the result descriptor word have significance.

The result descriptor word is used for storage of a fail result descriptor. The IOM builds the fail result descriptor, inserts it in the fail IOCB RD word, and links the fail IOCB into the status queue.

STATUS QUEUE

The Status Queue (SQ) is a queue of (1) all job-request related IOCB's which have been serviced and terminated, and (2) any fail IOCB's which have been generated by the IOM. When job-request IOCB's are terminated (or fail IOCB's are generated) and the necessary result descriptor information has been stored in the RD word of the IOCB, the IOCB is unlinked from the job IOQ (or fail IOQ) and is linked into the status queue. The linked IOCB's in the status queue represent a mix of terminated IOCB's for all devices and any fail IOCB's.

The SQ for the system consists of queues of linked IOCB's, one queue for each IOM on the system. The number of queues is dependent on the number of IOM's in the system.

The mechanism by means of which the status queue is accessed is the SQH address stored in the IOM XLATOR at initialize time. This address is unique for each IOM used, and serves as a pointer to a word in memory which defines the queue of linked IOCB's associated with a particular IOM. This word is called the status queue header (SQH) word.

When a request is terminated, the SQ address of an IOM is used to fetch the SQH word, which contains the following basic information:

a. Null (empty) state of SQ
b. Head field
c. Tail field
d. Status-Change-Vector bit
e. CPM-Interrupt bit
f. CPM Number

The null state of the SQ is checked to determine whether it contains any terminated IOCB's. If the SQ is null (empty), no linkage of the current terminated IOCB to previously-terminated IOCB's in the SQ is required. Conversely, if the SQ is not null (contains one or more IOCB's or fail IOCB's), the current terminated IOCB must be linked to the queue of terminated IOCB's in the SQ.

The head field of the SQH word contains the base address of the first terminated IOCB of the SQ. The tail field of the SQH contains the base address of the last terminated IOCB of the SQ, except when the SQ is null or contains only one terminated IOCB. If the SQ is null the tail field is not used; if the SQ contains only one IOCB, the tail field contains the same address as the head field.

A terminated IOCB is linked to previously terminated IOCB's stored in the SQ by inserting its base address in the next link (NL) word of the terminated IOCB indicated by the SQH tail field. The address in the tail field of the SQH is then replaced with the base address of the currently-terminated IOCB, so that link capability is present when another request is terminated.

If the CPM interrupt bit is on in the SQH word, or the interrupt bit is on in the NL word of a terminated IOCB, a channel interrupt is sent to the CPM specified in SQH when the terminated IOCB is linked into SQ. An IOM error interrupt is always sent to the designated CPM when a fail IOCB is linked into the SQ.

When an SPO or a DCP requests an input operation, the status-change vector bit in SQH is set; and a channel interrupt is always sent to designated CPM.

IOM HOME (HA) COMMANDS

There are 20 home commands which the IOM can be directed to perform. When the IOM receives an interrupt from the CPM, it indicates that a home command has been constructed by the CPM and placed in memory. The home address stored in the IOM is then used to fetch the HA word. A code within HA word 1, which is called the Home Code, is then decoded to determine which command or command group is to be performed.

Table IV-1-3 lists the valid home codes, and the commands and/or command groups defined

Table IV-1-3. IOM HA Operations and Corresponding Home Codes

Home Code	IOM Operation
0000	Illegal
0001	Start I/O
0010	Set Channel Busy/Set Channel Reserved
0011	Reset Channel Busy/Reset Channel Reserved
0100	Load Home Address
0101	Load Unit Table Address
0110	Load IOQ Head Table Address
0111	Load SQ Header Address
1000	DFO/DCP Scan-out Commands: DFO: Clear the Stack Store Control Word Request DCP: Initialize Halt Set Attention
1001	DFO Scan-In Operations: Queued Control Word Top of Stack Report
1010	Synchronous I/O
1011	Interrogate Peripheral Status
1100	Inhibit IOM
1101	Activate IOM
1110	Load DFO Flags
1111	Illegal

by them. As indicated, scan commands are defined by the home code as only scan-in or scan-out groups; determination of type of scan-in or scan-out, and whether DFO or DCP, is defined by other portions of HA word 1. Similarly, channel busy/channel reserved commands are resolved by other portions of HA word 1. HA word 2 is not used for all commands; when used, it contains information to further define the command.

The following brief command descriptions are presented in reference to figure IV-1-6, which depicts the basic contents of the HA words for each command. Detailed formats of the HA word for each command are given in the appendix of IOM word formats.

START I/O (HOME CODE 0001)

The Start I/O command is the basic command used to initiate input/output servicing of a new job request for a device. The device is defined by a unit designate number contained in bits 28 thru 35 of HA word 1, HA word 2 is not used. This command need only be given once in order to service all queued requests for the designated device.

SET CHANNEL BUSY/SET CHANNEL RESERVED (HOME CODE 0010)

Home code 0010 may represent one of two commands, dependent upon the state of bit 39 of HA word 1. If bit 39 is a "0", the Set Channel Busy command has been received; if bit 39 is a "1", the Set Channel Reserved command has been received. Both commands are for exchange channels; the channel number is defined by bits 23 thru 27 of HA word 1. HA word 2 is not used.

The Set Channel Busy and Set Channel Reserved command are used primarily for diagnostic purposes. A start I/O command for an UD, which has the reserved channel bit (RC) set in its UT word, must use a channel that has been set to reserved; otherwise, a reserved channel will not be used. An I/O operation can not use a channel that has been set to busy. Once either command has been received, the specified channel remains busy (or reserved) until a counter command is received.

RESET CHANNEL BUSY/RESET CHANNEL RESERVED (HOME CODE 0011)

Home code 0011 may also represent one of two commands, dependent upon the state of bit 39 of the HA word ("0" defines the Reset Channel Busy command; "1" defines the Reset Channel Reserved command). These commands are the counter commands to the Set Channel Busy/Set Channel Reserved command.

LOAD ADDRESS COMMANDS

Load home address (home code 0100); load unit table address (home code 0101); load IOQ head table address (home code 0110); load SQ header address (0111).

The commands are normally used to load fixed addresses into the IOM XLATOR at initialize time; however, they may also be used to establish new base addresses at any time after initialization. The address to be loaded by each command is contained in bits 0 thru 19 of HA word 1; HA word 2 is not used.

DFO/DCP SCAN-OUT COMMANDS (HOME CODE 1000)

Home Code 1000 specifies a scan-out command for either a DFO or a DCP; the specific device for which the scan-out command is intended, as well as the specific type of scan-out command, is defined by other bits of HA word 1.

Whether the scan-out command is for a DFO or a DCP is dependent upon bits 19 thru 16 of HA word 1. A bit configuration of 1001 indicates the command is for a DFO, whereas a bit configuration of 1100 indicates the command is for a DCP.

DFO SCAN-OUT COMMANDS

There are two specific scan-out commands for DFO's. The command type is determined by the configuration of bits 4 and 5 of HA word 1 as follows:

a. Bit 4 = 0, bit 5 = 1: Clear the Stack

b. Bit 4 = 1, bit 5 = 0: Store Control Word Request

The Clear the Stack Command is used to erase the DFO queuer stack and to reset any DFO error flip-flop previously set. The unit number of a Disk File Electronics Unit (DFEU) is contained in bits 8 thru 15 of HA word 1. This unit number is used with bit 7 (the exchange select bit) to define either the DFO with which the DFEU is directly connected or the DFO with which the DFEU is indirectly connected. HA word 2 is not used for the Clear the Stack command.

The Store Control Word Request command is used to request storage of a control word in the DFO queuer stack. The DFO is defined by bit 7 and bits 8 thru 15 of HA word 1, as described for the Clear the Stack command. A fetch of HA word 2 is required to implement this command. Bits 28 thru 47 of HA word 2

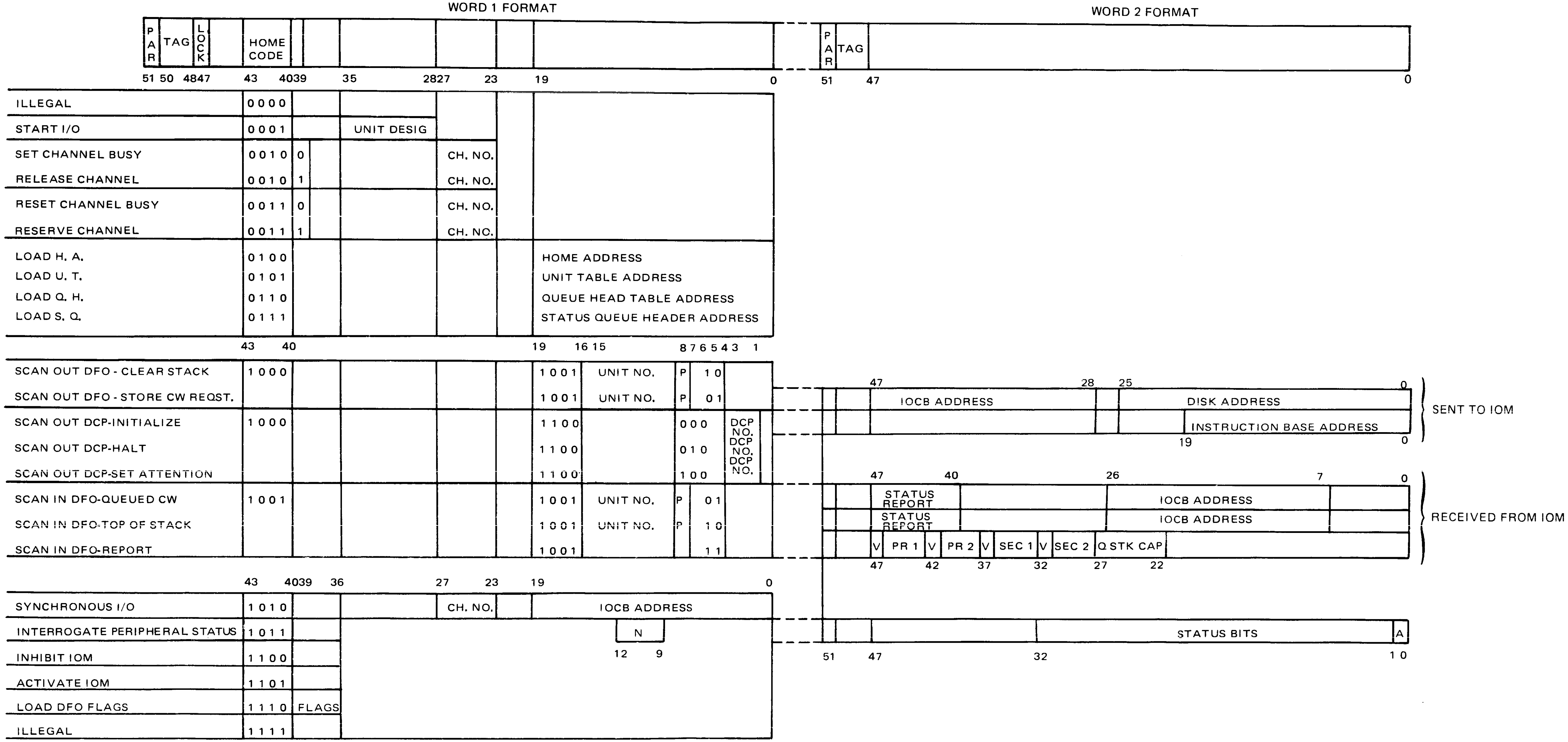

WORD 1 FORMAT
PAR
TAG
LOCK
HOME CODE
51 50 4847 43 4039 35 2827 23 19 0
WORD 2 FORMAT
PAR
TAG
51 47 0
ILLEGAL 0000
START I/O 0001 UNIT DESIG
SET CHANNEL BUSY 0010 0 CH. NO.
RELEASE CHANNEL 0010 1 CH. NO.
RESET CHANNEL BUSY 0011 0 CH. NO.
RESERVE CHANNEL 0011 1 CH. NO.
LOAD H. A. 0100 HOME ADDRESS
LOAD U. T. 0101 UNIT TABLE ADDRESS
LOAD Q. H. 0110 QUEUE HEAD TABLE ADDRESS
LOAD S. Q. 0111 STATUS QUEUE HEADER ADDRESS
43 40 19 16 15 8 7 6 5 4 3 1
SCAN OUT DFO - CLEAR STACK 1000 1001 UNIT NO. P 10
SCAN OUT DFO - STORE CW REQST. 1001 UNIT NO. P 01
SCAN OUT DCP-INITIALIZE 1000 1100 000 DCP NO.
SCAN OUT DCP-HALT 1100 010 DCP NO.
SCAN OUT DCP-SET ATTENTION 1100 100 DCP NO.
SCAN IN DFO-QUEUED CW 1001 1001 UNIT NO. P 01
SCAN IN DFO-TOP OF STACK 1001 UNIT NO. P 10
SCAN IN DFO-REPORT 1001 11
47 28 25 0
IOCB ADDRESS
DISK ADDRESS
INSTRUCTION BASE ADDRESS
19 0
SENT TO IOM
47 40 26 7 0
STATUS REPORT
IOCB ADDRESS
STATUS REPORT
IOCB ADDRESS
V PR 1 V PR 2 V SEC 1 V SEC 2 Q STK CAP
47 42 37 32 27 22
RECEIVED FROM IOM
43 4039 36 27 23 19 0
SYNCHRONOUS I/O 1010 CH. NO. IOCB ADDRESS
INTERROGATE PERIPHERAL STATUS 1011 N
12 9
STATUS BITS A
51 47 32 1 0
INHIBIT IOM 1100
ACTIVATE IOM 1101
LOAD DFO FLAGS 1110 FLAGS
ILLEGAL 1111

Figure IV-1-6. Home Address Commands

memory, and bits 0 thru 25 contain the disk address to be used.

DCP SCAN-OUT COMMANDS

There are three specific scan-out commands for the DCP; the command type is determined by bits 5, 6, and 7 of HA word 1 as follows:

a. Bits 5, 6, and 7 = 0: Initialize
b. Bits 5 = 0, bit 6 = 1, bit 7 = 0: Halt
c. Bits 5 and 6 = 0, bit 7 = 1: Set Attention

The DCP for which the command is intended is indicated by a DCP number contained in bits 1 thru 3 of HA word 1.

The Initialize command requires access by the IOM of HA word 2, which contains an instruction base address (bits 0 thru 19). HA word 2 is not used for the Halt and Set Attention commands.

The Initialize and Halt commands cause psuedo fault interrupts to occur within the DCP. In the case of the Initialize command, the interrupt causes the 20-bit instruction base address to be loaded into the DCP scratch-pad memory. The interrupt generated by the Halt command stops DCP operations. In either case, stop actions which would normally occur within the DCP due to fault interrupt occurrence are inhibited.

The Set Attention command is used to notify the DCP that attention to the B 7700 system is required.

DFO SCAN-IN COMMANDS (HOME CODE 1001)

Home code 1001 specifies that the command to be performed is one of three DFO scan-in commands. The specific command is defined by the configuration of bits 4 and 5 of HA word 1 as follows:

a. Bit 4 = 1, bit 5 = 0: Queued Control Word
b. Bit 4 = 0, bit 5 = 1: Top of Stack
c. Bit 4 = 1, bit 5 = 1: Report

The desired DFO is addressed by use of a DFEU number and exchange-select bit as in DFO scan-out commands. This information however, is not used for the Report commands.

HA word 2 is not accessed to further define command parameters. However, it is later used for storage of the scan-in word received via the scan information lines, and is the mechanism by means of which the CPM is notified of the location and status of the scan data.

The Queued Control Word command is used to request an optimized control word from the queuer stack of the DFO. The Top of Stack command is used to request DFO transmission of the control word located at the top of the DFO queuer stack.

The report command is used to request a report from the DFO, which defines the DFEU's connected to all DFO primary and secondary parts. The report is stored in HA word 2. Four fields in the report stored in HA word 2 are used to define the DFEU's which are connected to the four DFO ports (two primary and two secondary). Each field consists of (1) a bit (valid) which indicates whether an EU/DFO bus is connected to the port, and therefore whether the field is valid, and (2) four bits which represent the most significant bits of the unit number of the lowest-numbered DFEU connected to the port. A maximum of 40 DFEU's may be connected to a DFO (20 direct and 20 indirect). Since the DFO provides four ports, a maximum of 10 DFEU's are connected to a port, and the four bits are sufficient to define them.

The report returned and stored in HA word 2 also contains information which defines the capacity of the DFO queuer stack. The information is reported in bits 22 thru 27 of HA word 2.

SYNCHRONOUS I/O COMMAND (HOME CODE 1010)

The Synchronous I/O command provides a means of servicing a single job request during initialization. Only HA word 1 of the job map, which contains the IOCB base address (bits 0 thru 19), is accessed; no queue mechanisms are used. When the single job request is terminated, the result descriptor information is stored in HA word 5, and a channel interrupt is sent to the CPM.

INTERROGATE PERIPHERAL STATUS COMMAND (HOME CODE 1011)

The Interrogate Peripheral Status command is used to determine the ready status of all devices assigned to a particular status vector. The status vector to be interrogated is indicated by bits 9 thru 12 of HA word 1.

HA word 2 is not accessed for command determination, but is later used for storage of the returned status information. The status information, which is returned in bits 1 thru 32 of HA word 2, provides indication of the ready status of up to 32 devices on a vector. Bit 0 of HA word 2 (ATTN) notifies the CPM that the status word has been returned.

INHIBIT IOM COMMAND (HOME CODE 1100)

The Inhibit IOM command is used to inhibit all automatic IOM functions, such as data-path management, DFO scan-in and scan-out functions, chaining of linked and side-linked job requests and ringwalk. If linked job requests for ringwalk devices are being serviced when the command is received, chaining stops after the IOCBs in progress on each channel are completed.

The content of HA word 1 consists only of the home code; HA word 2 is not accessed.

ACTIVATE IOM COMMAND (HOME CODE 1101)

The Activate IOM command is used to restore automatic functions of the IOM after the Inhibit IOM command has been given. The command consists only of the home code in HA word 1; HA word 2 is not used.

LOAD DFO FLAGS COMMAND (HOME CODE 1110)

The Load DFO Flags command is used to mask out one or more DFO's connected to an IOM. The HA word in which the Load DFO Flags command is received contains four flag bits, one for each DFO which may be connected to an IOM. These bits (36 thru 39) are referred to as the DFO ON/OFF flags for DFO numbers 0 thru 3, respectively. When an OFF flag (0) is detected, scan-in and scan-out operations with the associated DFO are inhibited.

AUTOMATIC SERVICE OF DISK JOBS FOR UNITS UNDER DFO CONTROL

In B 7700 disk file subsystems where the disk jobs are not optimized, the service of multiple job requests for disk units on a common exchange involves an inherent delay between service of each job request. This delay is partially due to the manner in which the jobs must be linked under the queue of IOCB's for each Disk File Electronics Unit (EU); that is, without regard to the relationship of the disk starting address specified by each job request and the current disk position (relative to the head), since the current disk position is unknown.

In B 7700 disk file subsystems where Disk File Optimizers are used, the inherent delay between the service of multiple job requests is reduced. The job requests are linked under the queues of IOCB's for the DFO's, rather than under the queues of IOCB's for the EU's as in a non-optimized system. Upon receipt of a Start I/O HA command, the UT word is fetched. If the DFO bit is set, the job requests are automatically scanned out to the DFO job stack when possible. The DFO's constantly monitor the disk addresses specified by the job requests in the stack and compare them with the current disk position relative to the head. This information is used to maintain a job-stack pointer which indicates the current optimum job request relative to disk/head position. This current optimum job request is referred to as a queued control word.

The DFO's communicate with the IOM via a common scan bus and individual status lines. The status lines transfer information regarding the capability of the individual DFO's to receive job requests from the IOM over the scan bus. In addition, the status lines transfer levels which indicate the availability of queued control words which require service. The SCI section of the IOM scans these status lines to determine whether queued control words are available from any DFO. If the status lines of any DFO indicate the availability of queued control words, the SCI section of the IOM determines whether a disk channel is available on the exchange to which the DFO is connected. If so, a scan address word is formatted by the Xlator and SCI sections of the IOM, and is then sent over the scan bus to all DFO's. The contents of the scan address word sent over the scan bus identify the exchange and the DFO on that exchange which has indicated availability of queued control words. The scan address word on the scan bus is recognized only by the identified DFO, and therefore is, in essence, an acknowledge to that DFO.

In response to the scan address word received, the identified DFO transfers a scan information word over the scan bus to the IOM. This word contains a complete memory link address, which is used by the IOM to further access the IOM job map for the identified job. The map access performed provides information which identifies the EU which is to control the disk job, whether that EU is available, and whether the previously-available disk channel is still available. If all conditions are met, the job is initiated and data are transferred between the DFI section of the IOM and the specified EU. Upon completion of the data transfer, the disk job is terminated in the normal manner. If the identified EU is not available or if the disk channel is not available, the disk job is relinked under the queue of IOCB's for the DFO. It is then later transferred again to that DFO for reoptimizing and another attempt at job initialization.

AUTOMATIC DISK-PACK OPERATION

The IOM has provisions to automatically reinitiate a type 225 Disk Pack Unit after the unit has completed a seek operation. This type of unit, when issued a conditional I/O command requiring head positioning (seek), must be issued the same command after the seek has been completed in order to accomplish data transfer. The IOM performs this function by examining all device result descriptors received from Disk Pack units.

DATA TRANSLATION

The PCI section of the IOM has the capability of translating data from one representation code to another during an I/O operation. The data types actually encountered are device dependent, and the translation to be performed (if any) is determined for each individual job request by standard control bits in the IOCW. The IOCW bits used to specify code translations are:

Bit 46 - ASCII on for any translation having ASCII input or output.
Bit 44 - READ - (READ = 1, WRITE = 0)
Bit 42 - TRANSLATE
Bit 41 - FRAME LENGTH (8 bit characters = 1, 6 bit characters = 0)

All possible combinations of these code bits are listed in detail in table IV-1-4 and the specific translation codes used by software for each peripheral devices are given in table IV-1-5.

EBCDIC-BCL EXCEPTIONS

Bi-directional translation of corresponding EBCDIC graphics to/from corresponding BCL graphics are provided with the following exceptions:

a. EBCDIC to BCL (output translator)

EBCDIC	BCL
PZ	+
MZ	x (times)
Corresponding graphics	Corresponding graphic
Non-corresponding graphics	(See Note)

NOTE:

The following graphics are printed dependent upon whether the printer is equipped for EBCDIC or BCL:

	0111 11C1 0101 0110 1101 0100 1111			
		¬	_	|
EBCDIC Printer	(Apostrophe)	(Logical not)	(Underscore)	(Vertical bar)
BCL Printer	≥	≥	≠	→

b. BCL to EBCDIC translator (input translator)

BCL	EBCDIC
X (times)	MZ
Corresponding graphics	Corresponding graphics

IOM-GENERATED INTERRUPTS

The IOM generates the following two interrupts and sends them over individual lines to each central processor:

1. Channel Interrupt
2. IOM Error Interrupt

An IOM interrupts exactly one processor; the CPM to be interrupted is determined by the CPM field (bits 44-42) of SQH.

There are three conditions under which the IOM generates an interrupt to a CPM:

a. IO Complete - On job request termination a channel interrupt is generated when bit 40 of the IOCB NL word is set or when bit 40 of the Status Queue Header is set. These bits are set by software; bit 40 in SQH is reset by the IOM after an interrupt is generated. Unless requested by software, an interrupt is not set for "exception conditions" (peripheral parity error, end of tape, etc.). The only action taken for an "exception condition" is that the next request job, if there is more than one request job queued, is not started and bit 0 of the UT word is set.

Table IV-1-4. General Translation Specification Codes

R	T	FL	A	Translation
44	42	41	46	
0	0	0	0	Write 6-bit bytes with no translation
0	0	0	1	(Illegal Code)
0	0	1	0	Write 8-bit bytes with no translation
0	0	1	1	Write EBCDIC from ASCII
0	1	0	0	Write BCL External from BCL Internal
0	1	0	1	Write BCL External from ASCII
0	1	1	0	Write BCL External from EBCDIC
0	1	1	1	Write ASCII from EBCDIC
1	0	0	0	Read 6-bit bytes with no translation
1	0	0	1	(Illegal Code)
1	0	1	0	Read 8-bit bytes with no translation
1	0	1	1	Read EBCDIC into ASCII
1	1	0	0	Read BCL External into BCL Internal
1	1	0	1	Read BCL External into ASCII (See Note 1)
1	1	1	0	Read BCL External into EBCDIC
1	1	1	1	Read ASCII into EBCDIC

Note 1:
In these combinations the frame length bit should be "1". However, due to encoding considerations, it is necessary to use this code and alter the FL decode to cover these cases.

b. Status Changes - When the "inquiry request" line for a Single Line Control device changes from "off" to "on" state, or when a DCP requests CPM attention, the IOM sets the proper bit in its status change vector (status vector 8). If this is the first such change since the vector was last interrogated by software, the IOM sets bit 45 in SQH to request software to read the status change vector and generates a channel interrupt. (Bit 45 in SQH is reset by software.)

c. IOM Errors - An IOM error interrupt is generated for any error not related to a specific job request (for example, a memory parity error on the Home Address word). The generated fail result descriptor is placed in a dummy IOCB from unit 0.

Table IV-1-5. Translation Codes by Device

CODE SPECIFIER

Device	R	T	FL	A	Description
	44	42	41	46	
Card Reader	1	0	0	0	Read binary data (6-bit to 6-bit)
	1	0	1	0	Read EBCDIC data (8-bit to 8-bit)
	1	0	1	1	Read EBCDIC into ASCII
	1	1	0	0	Read BCL External into BCL Internal
	1	1	0	1	Read BCL External into ASCII
	1	1	1	0	Read BCL External into EBCDIC
Card Punch	0	0	0	0	Punch binary (6-bit to 6-bit)
	0	0	1	0	Punch EBCDIC (8-bit to 8-bit)
	0	1	0	0	Punch BCL External from BCL Internal
	0	1	0	1	Punch BCL External from ASCII
	0	1	1	0	Punch BCL External from EBCDIC
	0	0	1	1	Punch EBCDIC from ASCII
Line Printer	0	0	0	0	Write BCL External (6-bit to 6-bit)
	0	1	0	0	Write BCL External from BCL Internal
	0	1	0	1	Write BCL External from ASCII
	0	1	1	0	Write BCL External from EBCDIC
Train Printer	0	0	0	0	Write with no translation (6-bit to 6-bit)
	0	0	1	0	Write with no translation (8-bit to 8-bit)
	0	0	1	1	Write EBCDIC from ASCII
	0	1	0	0	Write BCL External from BCL Internal
	0	1	0	1	Write BCL External from ASCII
	0	1	1	0	Write BCL External from EBCDIC
	0	1	1	1	Write ASCII from EBCDIC
P.T. Reader	1	0	0	0	Read binary (6-bit to 6-bit)
	1	0	1	0	Read EBCDIC or Read ASCM (8-bit to 8-bit)
	1	0	1	1	Read EBCDIC into ASCII
	1	1	0	0	Read BCL External into BCL Internal
	1	1	0	1	Read BCL External into ASCII
	1	1	1	0	Read BCL External into EBCDIC
	1	1	1	1	Read ASCII into EBCDIC
P.T. Punch	0	0	0	0	Punch binary (6-bit to 6-bit)
	0	0	1	0	Punch EBCDIC or Punch ASCII (8-bit to 8-bit)
	0	1	0	0	Punch BCL External from BCL Internal
	0	1	0	1	Punch BCL External from ASCII
	0	1	1	0	Punch BCL External from EBCDIC
	0	1	1	1	Punch ASCII from EBCDIC
	0	0	1	1	Punch EBCDIC from ASCII
7 TR. TAPE	0	0	0	0	Write with no translation (6-bit to 6-bit)
	0	1	0	0	Write BCL External from BCL Internal
	0	1	0	1	Write BCL External from ASCII
	0	1	1	0	Write BCL External from EBCDIC
	1	0	0	0	Read with no translation (6-bit to 6-bit)
	1	1	0	0	Read BCL External into BCL Internal
9 Tr. Tape	0	0	1	0	Write with no translation (8-bit to 8-bit)
	0	1	1	1	Write ASCII from EBCDIC
	0	0	1	1	Write EBCDIC from ASCII
	1	0	1	0	Read with no translation (8-bit to 8-bit)
	1	1	1	1	Read EBCDIC into ASCII
	1	0	1	1	Read ASCII into EBCDIC
SLC	0	0	1	0	Write EBCDIC or Write ASCII (8-bit to 8-bit)
	1	0	1	0	Read EBCDIC or Read ASCII (8-bit to 8-bit)
Disk Pack and Disk (See Note 1)	0	0	1	0	Write with no translation (8-bit to 8-bit)
	1	0	1	0	Read with no translation (8-bit to 8-bit)

Note 1:
The DFI section of the IOM has no hardware translation capabilities.

SECTION 2

FUNCTIONAL OPERATION OF INPUT/OUTPUT MODULE SUBSECTIONS

GENERAL

This section contains a brief description of the operation of each of the IOM subsections described in section 1 of this chapter. For the formats of the words discussed, refer to the appendix of IOM word formats in this manual.

FUNCTIONAL OPERATION OF TRANSLATOR

The translator (figure IV-2-1) is a special-purpose processor capable of performing specific hardwired microsequences. It is the mechanism of the IOM that services I/O requests, generates the request descriptors required to initiate peripheral devices, and reports job termination and failure status conditions to the Central Processor. The operation of the translator is keyed to respond to certain declared flag conditions.

JOB SERVICE INITIATION

In response to an interrupt from the Central Processor, the IOM unlock-fetches the word in memory referenced by the 20-bit Home Address stored in the HA location of the lower stack. The HA-word control fields define the control codes and function details for the request as described in Section 1 of this chapter. When the Start I/O command is decoded, the Unit Designate (UD) field of the Home Address word is loaded into the UD register (see figure IV-2-1).

The UD field is added to the 20-bit Unit Table (UT) base address (stored in the UT location of the lower stack) to address and lock fetch from memory (write with flashback) the Unit Table word for the device to be started. For devices other than a DFO, the contents of the channel number identification field of the UT word are used to access the Active Channel Stack (ACS). If the device is not connected to an exchange, and if the ACS and UT word busy bits are reset, the I/O Queue Head word (IOQ base address + UD) is fetched from memory to obtain the base address of the IO Control Block (IOCB).

Successive fetches are made of (1) the IOCB base address plus 2, to obtain the buffer descriptor, (2) the IOCB base address plus 3 to obtain the IO control word (IOCW), and (3) the IOCB base address plus 4 to obtain the Channel Designate Level (CDL) field. The buffer descriptor is comprised of two fields: base address information and buffer-length information. The 20-bit base address field is used to locate the buffer in memory.

The IOCW Standard Control Field (SCF) contains information useful to the data service sections such as read/write, translate, and format bits. Information contained in the buffer descriptor, SCF from the IOCW, and CDL word of the IOCB is sent to the data service section for starting up the selected device. The Unit Designate number is stored in the ACS and the ACS busy bit is set. The Unit Table word busy bit is set, and the UT word stored in memory is unlocked. The HA word is unlocked and set to all zeros. Control is transferred to the initial state.

If the device to be started is connected to an exchange and the busy bit of the base channel location in the ACS is set, the translator logic selects the next channel of the exchange and checks its busy bit. If a channel is available, information is fetched from memory and is sent to the data service section, to start the selected device. If all channels of the exchange are busy, the job bit (JB) in the Unit Table word is set, and the UT word is stored unlocked in memory. Control is transferred to the initial state. These conditions are summarized in table IV-2-1.

JOB-SERVICE TERMINATION

When a device either completes a service or is terminated as a result of an error condition, the data service unit causes the terminate bit to be set for that channel. The terminate bits are located in the Active Channel Stack (ACS) of the translator; one bit for each of the possible 28 channels available. In response to a terminate bit being set, the translator reads the corresponding Unit Designate (UD) information from the ACS. This information is used along with the unit table base address to index and lock-fetch from level-1 memory the UT word for the terminating device. The I/O Queue Head is then fetched to obtain the base address of the I/O Control Block (IOCB). The Result Descriptor (RD) information received

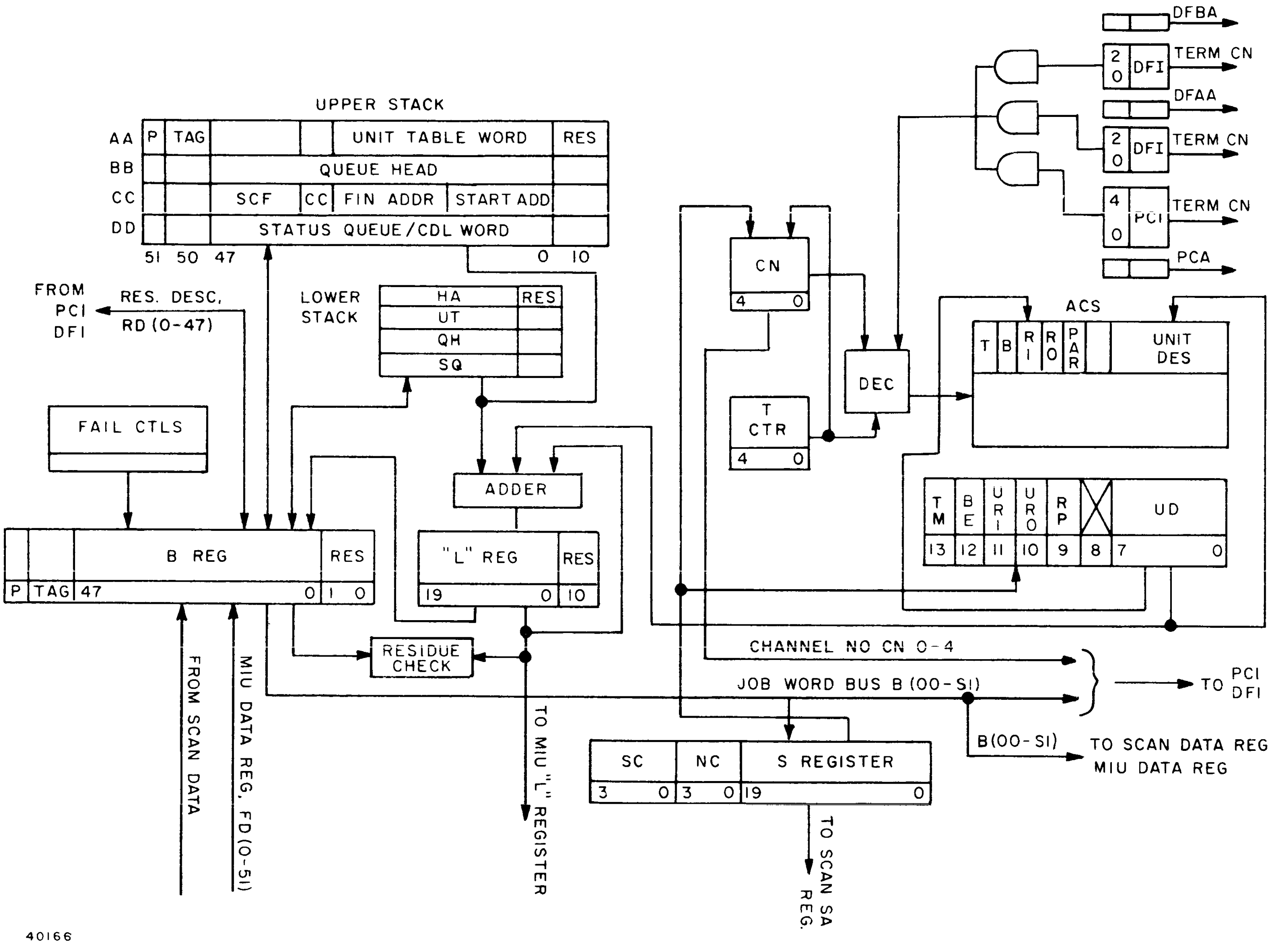


Figure IV-2-1. Translator Component Interface

Table IV-2-1. Unit Table and Active Channel Coded Decisions

Unit Table Word					A.C. Stack		
B20 RC	B38 EX	B37 JB	B36 BZ	CBF	CTF	CRF	Decisions
M	X	0	0	0	0	M	Start job; unlock UT; Set BZ (UT)
X	0	1	X	X	X	X	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
X	0	X	1	X	X	X	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
X	0	X	X	1	X	X	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
X	0	X	X	X	1	X	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
U	0	X	X	X	X	U	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
X	*1	1	0	X	1	X	Unlock UT; Go to initial state
X	*1	1	0	1	X	X	Unlock UT; Go to initial state
X	1	0	0	X	1	X	Unlock UT; Set JB; Go to initial state
X	1	0	0	1	X	X	Unlock UT; Set JB; Go to initial state
M	1	1	0	0	0	M	Start job; Unlock UT; Set BZ, Res JB
U	1	X	0	X	X	U	Unlock UT; Set JB, go to initial state

X = State irrelevant
M = States match
U = States do not match
EX = Exchange bit
JB = Job to be done
BZ = Job Busy
RC = Use reserved channel only
CBF = Chan Busy FF
CTF = Chan Term FF
CRF = Chan Reserved FF
* = Applicable only when second IOM has set JB

from the data service unit is then stored in the sixth word of the IOCB, and the IOCB is linked to the Status Queue (SQ).

If this is the last request for this unit, the I/O Queue Head (IOQH) and I/O Queue Tail (IOQT) are nulled, and the UT word is stored unlocked, to complete the termination. If there are more requests, the address of the next IOCB is inserted in the 20-bit address field of the I/O Queue Head. Control is passed to the start section to initiate the request. If the terminating IOCB is the last request for this unit, and the unit is connected to an exchange, then control is passed to the ring-walk section and a search is made to find a request that is waiting to be initiated.

EXCHANGE RING WALK

The following action is taken when an exchange unit terminates and there are no more requests queued for that particular unit.

The Present Unit Designate (PUD) field from UT is saved in the S register. The Unit Designate (NUD) field in the UT word of the terminating device points to the next unit (device) of the exchange. The UT word for this unit is fetched from level-1 memory and status of the busy and job bits is checked. If the unit is busy or there are no jobs waiting service, then the information in the Next Unit Designate (NUD) field in the UT word is used to point to the next unit of the exchange. This process of looking for a request continues until either a request is found or the entire exchange has been walked (NUD=saved PUD). When a unit is found with a serviceable request waiting, the IOQH word for that unit is fetched from memory and the job is started.

AUTOMATIC SERVICE OF DISK JOBS VIA DFO UNITS

IO requests for disk units under control of a Disk File Optimizer (DFO) are entered in the IO Queue for the DFO rather than the IO Queue for the appropriate disk unit. The unit table word for the DFO has bit 39 (DFO) set. The start IO operation for a DFO causes the unit table word to be lock-fetched from memory. If the DFO bit is set, the DFOQ flag is set in the IOM, the unit table word is written to memory and unlocked, and the translator goes to an idle state.

At any time while in idle state, when the DFOQ flag is set, the DFO stack is not full, the DFO is on, the scan bus is not busy (SBY), and no jobs of a higher priority exist, an automatic DFO scan-out is initiated. The Queue Scan (QSN) flag is set and start mode is entered. The DFO unit table word is lock fetched from memory, and the DFO IO Queue Header (IOQH) is read. If the IOQH is null, DFOQ and QSN are reset, the unit table word is written to memory and unlocked, and the translator returns to an idle state. If the OH is not null, scan mode is entered. The CDL word is fetched from IOQH+4.

Several fields are combined to form a scan-out word. The disk address and IOCB address fields form the scan data word, and the EUD field plus device type and function code fields form the scan address word. This information is placed on the scan bus. The UT word is written to memory and unlocked, and the Scan In Process (SIP) flag is set which inhibits the translator from initiating other jobs when in the idle state. The idle state is then entered. When SIP=1 and SBY=0, SIP is reset and scan mode is re-entered.

The UT word is lock fetched from memory, the IOQH is read and the Next Link (NL) field is obtained from the IOCB; then NL is stored in the IOQH. If NL=0, then the IO Queue Tail (IOQT) is nulled and DFOQ is reset. The UT word is written to memory and unlocked, QSN is reset, and the translator returns to the idle state.

A scan-in word is formed and placed on the scan bus when the translator is in the initial state, the scan bus is not busy, and a DFI channel is not busy (served by a DFO). When the Scan-In operation is completed, the translator is notified. The 20 bit address field of the scan-in word is the base address of the IOCB for the next disk-file-EU to be started. The base address plus 4 is used to fetch the CDL information word. The EU Designate field of the CDL word and the UT Base are added, and the result is then used to fetch the UT word of the disk unit from memory. If the UT word busy-bit is zero, the Buffer Descriptor, CDL and IOCW are fetched as before and are sent to the DFI section to start the device. The base address is stored in the IO Queue Head and is used for generating information during the terminate operation of the device. The busy-bit is set and the UT word is stored unlocked in memory and control is returned to the initial state.

If the busy-bit is set in the disk UT word, the IOCB is linked into the tail of the DFO's IO Queue.

DISK-PACK CONTROL

The following actions are taken upon the receipt and examination of result descriptors from Disk Pack units.

When a result descriptor indicating "Seek Initiated" is received, the IOM will not de-link the IOCB or store the result descriptor as in normal terminate operations. Instead, the unit number of the Disk Pack which has begun a seek operation will be stored in a local stack. The contents of this stack are then used to monitor the ready lines of all Disk Pack Units which are currently seeking.

A "Seek Complete" will be detected when the ready line of a seeking Disk Pack returns to the true state. At this time, the Translator will perform a Start I/O for that Disk Pack unit. Since the original job was not de-linked from the job queue when the Disk Pack initiated its seek operation, the same job will be issued a second time, and the data transfer will occur. After this point, all IOM operations proceed the same as for normal peripheral units. If the Disk Pack is issued a conditional I/O command which does not require head positioning, data transfer occurs directly, and the Automatic Disk Pack functions of the IOM are not used.

FAIL MODE OF OPERATION

The fail mode is used to provide the IOM the capability of reporting errors that cannot be associated with a specific request. When an error occurs, such as Scan Bus Error, Memory Error, Home Address Error, Illegal Command, etc., control is passed to the Fail Mode (FM) with appropriate error flags.

A fail result descriptor is built in the fail register. This RD indicates (1) the operational mode when the error occurred, (2) a possible channel number (or memory address, depending on the type of failure), and (3) error flags describing the type of error.

A Fail Unit Designate number (Fail UD = zero) is used with the UT word to access a Fail UT word. Then the QH and Fail UD is used to access the fail I/O Queue Head. The fail result descriptor is placed in the Result Descriptor word of the I/O Control Block (IOCB). The Fail IOCB is then delinked from the queue of Fail CB's and linked to the Status Queue as on a normal termination. An IOM Error Interrupt is then sent by the IOM to the Central Processor designated in the Status Queue Header.

FUNCTIONAL OPERATION OF MIU

The MIU (figure IV-2-2) performs all data and map-word transfers between the IOM and a maximum of 8 system Memory Control Modules, and detects and reports memory error conditions to the requesting functional unit of the IOM (and to the translator when applicable). The MIU manages level-1 memory access requests by the functional units of the IOM on a preassigned priority basis. The access priority scheme for the functional units of the IOM is as follows:

a. First Priority: Data Service requests

b. Second priority: Data Communications Processor interface requests

c. Third Priority: translator requests

When a functional unit of the IOM requires the services of the MIU for the purpose of performing a data transfer, it is required to raise its Access Request Line to the MIU and place a 26-bit Unit Control Word (UCW) on its UCW lines to the MIU. When the requesting unit has priority, the MIU loads the UCW into its control word register and performs one of the following operations:

a. Single data word fetch
b. N-length data word fetch
c. Single word overwrite with flashback
d. Single word protected write
e. Single word protected write with
f. N-length overwrite
g. single word overwrite
h. N-word protected write

Upon determining the type of operation requested, the MIU constructs a Memory Control Word (MCW) and transfers it to memory. Upon transferring the MCW to memory, the MIU is required to perform one of the operations listed below:

a. If a Single Word Store operation was specified: The MIU raises its request lines to the specified Memory Control Module (MCM) in order to alternately transmit the MCW and the data word to be stored to the addressed MCM. The MIU continues to transmit the MCW, followed by the data word to be stored, until an acknowledge signal is received from the MCM.

b. If a Multiple Word Store operation is specified: The MIU raises its request lines to the applicable MCM, and then sends the MCW to the MCM. When the MCM acknowledges receipt of the MCW, the MIU commences the data transfer under the control of the Data Request signal.

c. If a fetch operation is specified: The MIU raises its request lines and sends the MCW to the applicable MCM. When the MCM acknowledges receipt of the MCW, the MIU enables its memory-bus receiver circuits. Information from the MCM will now be accepted by the MIU. However, the MCM is required to transmit a Data Present Strobe pulse to the MIU to cause the information present on the memory bus to be transferred to and detected by the requesting IOM. The Data Present Strobe pulse is required for each word transferred from Memory to a requesting IOM.

While performing a data transfer, the MIU is required to detect and/or report memory error conditions. Memory errors are divided into two categories by the IOM: MIU-detected errors, and memory-detected errors. Memory errors cause termination of the memory request being processed, and the MIU transfers a 3-bit error code to the requesting section. The Translator reports these errors through the Fail Register. Data-service-section units return these errors in the Result Descriptor. A decode of these three bits specifies whether the error is an MIU or memory-detected error.

Errors detected and/or reported by the MIU and their associated 3-bit error reporting codes are listed in descending exclusive order as follows:

a. Store disparity (011) – This error condition is declared if a data transfer from an internal unit is received by the MIU with incorrect parity. The data with incorrect parity is transferred to the memory.

b. L1A Address Residue Error (010) – This error condition is declared if the MIU receives a UCW whose residue bits do not agree with its memory address field configuration (DCP words are not residue checked.)

c. Memory Detected Error (111) – This error condition is declared when the addressed memory module responds with a fail 1 (uncorrectable error) indication to a requestor unit.

d. No Access to Memory (101) – This error condition is declared if the MIU receives no response from the requestor memory module during a waiting period not to exceed 25 microsecond writing period. No response is defined as:

1. Failure to receive, at the MIU, an acknowledged signal from an addressed memory module, or
2. when incomplete data is transferred by an addressed memory module.

e. Fetch Disparity (110) – This error condition is declared if a fetch of data from memory is received by the MIU with incorrect parity.

f. Memory Protect Error (100) – This error condition is declared when the addressed memory module responds with a protect-error signal during a memory protect store operation.

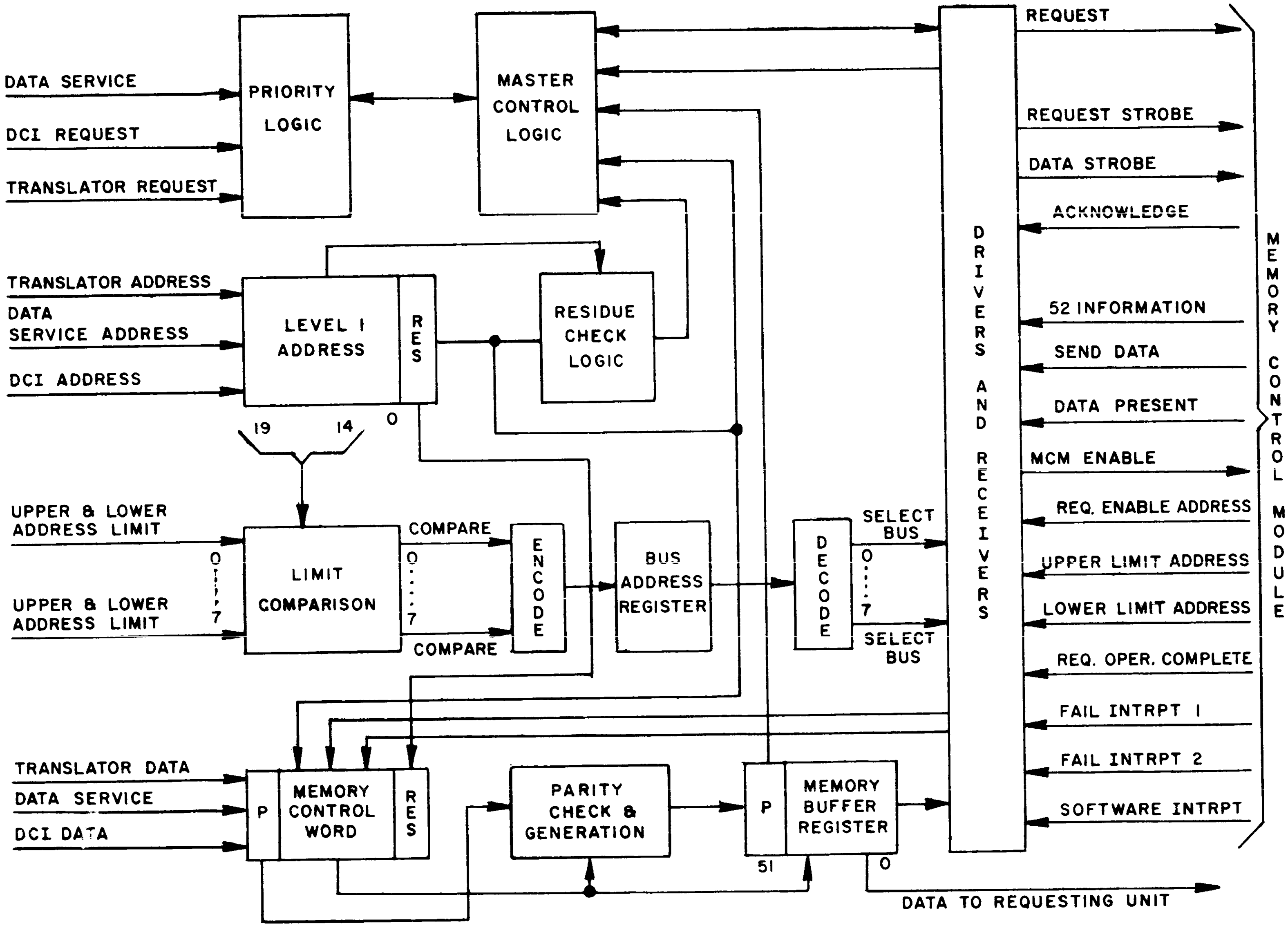
g. Memory Detected Error (001) – This error condition is declared when the addressed memory module responds with a fail 2 (1-bit corrected error) indication to a requestor unit. (This error condition does not cause termination of the memory access operation.)

FUNCTIONAL COMPONENTS OF THE MIU

The MIU consists of nine functional components interfaced as illustrated in figure IV-2-2 and operated as described below:

a. Priority Logic – This section is responsible for granting the services of the MIU to the highest priority requesting unit.

b. Master Control Logic – This section contains the control logic necessary to execute all MIU operations, including the controls required to complete receiver and driver paths.



40167

Figure IV-2-2. Memory Interface Unit

c. Kesidue Check Logic - This section is responsible for checking and verifying the residue bits of the memory addresses transferred from the translator and data service unity.

d. Parity Check and Generate Logic - This section is required to generate odd parity for all words being transferred to memory, and to check for odd parity of all words being fetched from memory.

e. Data Buffer Register - This is a 52-bit register and is used to buffer all data transfers between the requesting area of the IOM and the MIU.

f. Memory Buffer Register - This is a 52-bit register and is used to buffer all input data to memory.

g. L1A and Residue Logic - This section is responsible for constructing and routing the MCW to the data buffer register, as bits of the address.

h. Receivers and Drivers - There are 8 discrete groups of receiver and driver circuits in the MIU, one group per Memory Control Module interface. The state of these groups is determined by master control; only one group is active at any one time.

i. Limit Comparison Logic - This section is responsible for comparing the upper and lower address limits for the memory address: It is comprised of the limit comparison logic, encode logic, bus address register, and decode logic. It is the function of these logic areas to inform the requestor as to which MCM will be servicing the memory request and is accomplished by comparing the 6 most significant bits of the memory address to a known upper and lower limit for the answering MCM.

FUNCTIONAL OPERATION OF PCI

The PCI (figure IV-2-3) enables the IOM to interface with from one to twenty peripheral controllers (PCs), and coordinates data transfers between the PC's and the MIU as directed by the translator section of the IOM. The PCI interfaces with memory by one-word transfers via the MIU. Each PC requires a 1-microsecond service cycle to transfer data. By means of overlapping service cycles, and by use of local memory windows (a one-clock period during which a particular operation may be performed if no higher priority operation is required), it is possible to multiplex all twenty channels.

There are four operational phases of the PCI: (1) channel initiation (2) channel service, (3) memory operations and (4) channel termination. Each of these phases is described as follows. A general block diagram of the PCI is as shown in figure IV-2-3.

CHANNEL INITIATION OPERATION

This phase of PCI operation is controlled by Channel Designate Level (CDL) Control (referred to as CC), and includes all functions required of the PCI to start a device. The channel initiation phase commences when the translator control logic places the first job descriptor word (DSU word) on the translator bus and raises the request line to the PCI. When a local memory window becomes available, the CC strobes the DSU word into the local memory channel allocated to the device to be started, and lowers the PCI-available line to the translator. The translator control logic then lowers the request line and places the second job descriptor word (CDL word) onto the translator bus.

When the next available local memory window occurs, the CC strobes the CDL word into the appropriate location in local memory, strobes the channel number of the new request into initiate queue stack (InQ), and raises the PCI-available line. The translator is thus informed that the entire request descriptor has been received and stored in local memory. If no channel is currently being initiated, the CC selects the highest priority channel in the InQ, transfers this channel number to the CDL stack (CDS) which contains the request descriptor currently being initiated, and resets the INS bit for the selected channel. If no channel requires channel service, the CC checks the busy line of the channel.

If a not busy condition is detected, the CC commences transfer of CDL characters to the appropriate PC at the rate of one character per available service cycle. If the selected channel is busy when initiation is commenced or if the selected channel becomes not busy during transmittal of CDL characters, the request is terminated and appropriate result descriptor information is generated and transferred to the translator (see channel termination). After sending the correct number of CDL characters (4 for standard devices and 8 for disk file devices), the CC raises the start channel bus line to the PC and resets the CDS, and the initiation operations are completed.

CHANNEL SERVICE OPERATION

After completion of the initiation phase, a channel is serviced upon demand at a rate dependent upon the type of peripheral device involved. The PC requests service by raising the access-request line (ARL) to the PCI. The PCI selects the highest priority channel requesting service and generates the appropriate access granted level (AGL). The presence of this signal grants the next service cycle to the accessed peripheral. The service cycle consists of two

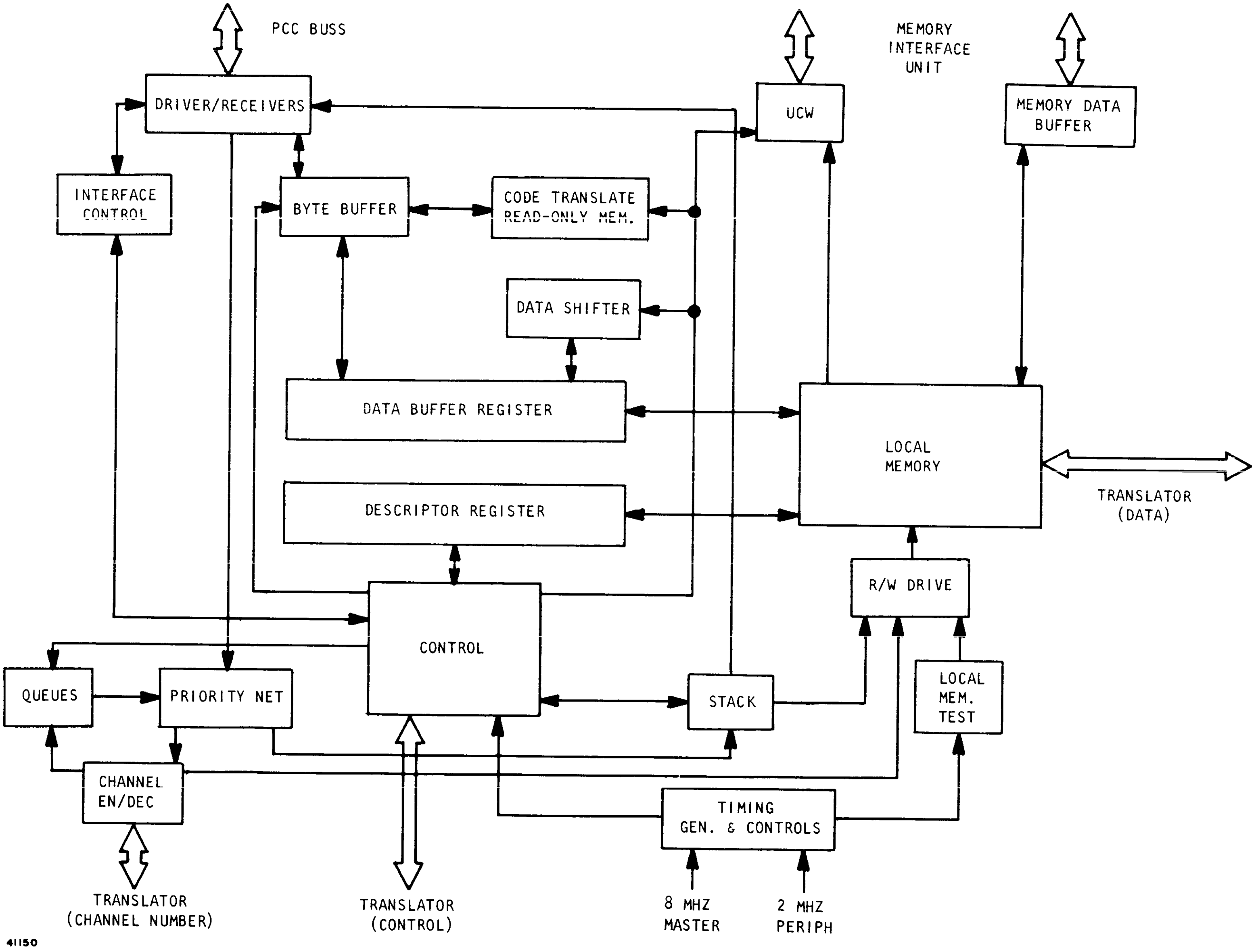


Figure IV-2-3. Peripheral Control Interface

T-time periods (T1 and T2) of 500 nanoseconds each; T1 is used for output to the PC. The AGL signal for the next service cycle is generated during the previous service cycle's T2 time period.

Each data transfer is controlled by a channel descriptor which has been generated from information contained in the DSU word of the job descriptor.

If an error is detected at any time during channel service, the PCI generates the appropriate result descriptor information for the translator (see channel termination) and terminates all operations on that channel.

MEMORY OPERATION

When the PCI determines that one 52-bit data word is required from or is ready to be sent to memory, the channel number selected for the transfer is placed in the Memory Queue (MQ). The MQ is a stack which contains the channel numbers of all channels requiring memory access. If no memory operation is currently being executed, the PCI selects the highest priority job in the MQ, and transfers this number to the Memory Operation Stack (MOS). The PCI then resets the MQ bit for the selected channel, transfers the Unit Control Word to the Unit Control Word Register (UCWR), raises the PCI-memory-request line to the MIU, and, if necessary, transfers data into the Memory Transfer Area (MTA).

Once access to the MIU is granted, the PCI strobe fetches data from the MTA, strobes the data to the appropriate data buffer in local memory, awaits the release signal which indicates that this memory request is completed, and then resets the MOS bit. The memory operation is thus completed.

If the MIU detects an error at any time during this sequence, the error information is transferred to the PCI. The PCI then causes the request to be terminated, and an appropriate result descriptor is generated.

CHANNEL TERMINATION OPERATION

After completing the required data transfer, the PCI sends an I/O complete level to the PC. The PC then returns the result descriptor available level and returns a result descriptor. This result descriptor information, along with the present Channel Descriptor information, is used by the PCI to create the result descriptor word to bring about a normal termination.

Abnormal-termination result descriptors can occur (1) during either channel initiation, channel service, or channel memory operations (when errors are detected by the PCI), (2) during channel memory operation (when errors are detected by either the MIU or the MCM, and (3) during channel initiation or channel service (when errors are detected by the PC). No matter what the source, all result descriptors are treated identically.

Once the result descriptor has been generated, it is stored in the local-memory location allocated to the channel to be terminated. The channel number of this request is strobed to the translator. Should the translator be unable to accept the channel number, the PCI stores this channel information in the Termination Queue (TMQ), which contains all the requests to be terminated. Whenever possible, the PCI selects the highest priority request from this stack, transmits the channel number to the translator, and resets the TMQ bit for the selected channel.

The translator replies to the PCI termination with a read-result-descriptor request, which causes the result word to be placed on the translator bus. This completes the termination operation.

FUNCTIONAL COMPONENTS OF THE PCI

The PCI consists of nine functional components as illustrated in figure IV-2-3. These functional components are described as follows:

a. PCI Local Memory (PCLM) - Consists of twenty 126-bit word locations, one for each PC channel within the PCI. Is used to store the channel descriptors and data for the twenty channels.

b. Descriptor Register (DR) - A 105-bit register which is loaded with the active channel descriptor.

c. Shift Logic (SFT) - Consists of right and left shift logic to properly locate data within the data buffer of the channel descriptor.

d. Byte Buffer (BB) - A 16-bit register which is loaded and unloaded (two 8-bit bytes at a time) during data transfers between the IOM and a PC.

e. Code Translator (CDT) - Consists of the logic circuits necessary to perform EBCDIC to BCL - EXTERNAL or vice-versa, BCL-External to BCL-Internal or vice-versa, ASCII to EBCDIC or vice-versa, and ASCII to BCL-External code translations or vice-versa.

f. Stack Control - Contains stacks whose functions are described in functional operations.

g. Control Section - Contains the CDL control, channel service control, memory transfer control, and termination control which are described in preceding paragraphs.

h. Driver/Receiver Section - Consists of special drivers and receivers used for transmitting and receiving data and controls for the PC.

i. Master Timing and Interface Control (STC) - Consists of logic to control the timing and interface sections and to provide control logic for the control panel.

FUNCTIONAL OPERATION OF DFI

The DFI (figure IV-2-4) enables an IOM to be interfaced with up to eight Disk File Controls (DFC). It consists of two independent, modular sections, each of which is capable of handling 4 data channels; each channel is interfaced to one DFC.

Each section controls data transfers with the DFC's via a 16-bit data bus at a transfer rate of 2 eight-bit characters per transfer time. The transfer rate to memory is 2 words (2 x 48 bits) per transfer time.

Each data channel comprises a four-word data-buffer area, called local data mode memory (LMD), and a 66-bit Channel-Descriptor local Memory (LMC). Upon command from the translator, the DFI initiates requests with its associated DFC's. During data transfer operations, the DFI communicates with the Memory Interface Unit (MIU) to obtain memory accesses. Upon request completion, the DFI notifies the translator of the termination status and awaits re-initiation.

The translator, upon receipt from memory of a disk file job, requests transmission of a job word to the DFI section assigned to handle that job. The selected DFI section loads the new request word into the proper channel-descriptor location in local memory and then releases the translator.

The DFI section, according to priority, reads the channel descriptor of an active request. An active request may be in any one of 3 separate modes. The exact mode is determined by the FAZ field in the channel descriptor. The three modes are (1) channel initiation, (2) channel service, and (3) channel termination.

CHANNEL INITIATION OPERATION

In the channel initiation mode, 48 bits of information are sent to the addressed DFC. This transfer is accomplished as 6 transfers of 8-bits each. These 48 bits are contained in the second request word (job word 2) which was, upon receipt from the translator, stored in the LMD. The W/BP field of the channel descriptor points to the location in the LMD of the information to be transferred to the DFC.

A 16-bit byte, of which only the most significant 8 bits are valid, is loaded into the transfer register (TR) and is then sent to the DFC. After 6 such transfers, a Start Channel Bus (STCB) signal is sent to the DFC to indicate the end of the initiation phase, and the Phase (FAZ) field of the channel descriptor word is incremented by one. If the job word indicates an output operation, two fetches of two words each are requested from memory by way of the MIU during the initiation phase. The first of these two-word fetches is stored in word positions 00 and 01 of the LMD, and the second two-word fetch is stored in word positions 10 and 11 of the LMD.

CHANNEL SERVICE OPERATION

The channel service operation, as indicated by the Phase (FAZ) field of the channel descriptor, consists of transferring 16-bit bytes of information to or from the DFC. If the DFC corresponding to the active channel has raised its access-request-level (ARL) line, the DFI responds by raising its access-granted-level (AGL) line and follows this by granting a one-microsecond service cycle.

If the transfer is to be an input, data is accepted during the first half of the one-microsecond service cycle and written into the Data Local Memory (LMD). If the transfer is to be an output, data is read from the LMD, placed in the transfer register (TR), and put on the data bus during the second half of the one-microsecond service cycle. The location in the LMD of the 16-bit byte to be transferred is determined by the Word/Byte Position (W/BP) field of the channel descriptor. With each 16-bit byte transfer, the W/BP field is incremented by one.

After two words of information have been transferred, memory request is again made by way of the MIU. The current memory address (CL1A) is sent along with certain control bits to the MIU, and if the request is for a read, two words are loaded from the LMD into the two-word buffer. If the request is for a write, two data words from memory will be loaded into the two-word buffer and then into the LMD. At this time the current memory address (CL1A) is compared to the final memory address (FL1A). If they are equal, the channel service phase is completed and the FAZ field is incremented by one; if they are unequal L1A is incremented by 2. The channel descriptor is restored into LMC and the next channel is serviced.

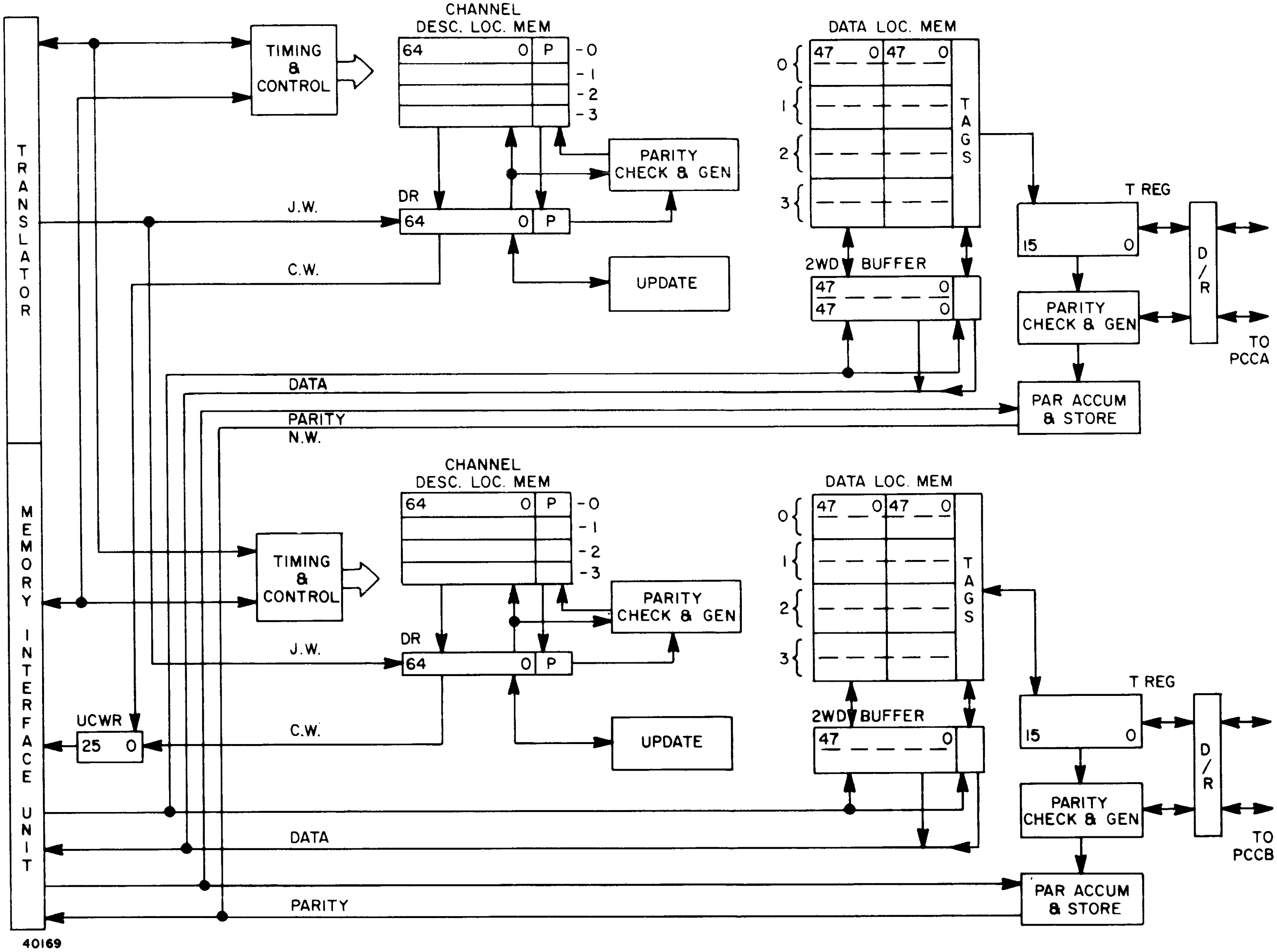


Figure IV-2-4. Disk File Interface

CHANNEL TERMINATION OPERATION

The channel termination operation is initiated when either of the following conditions occurs:

a. Normal completion of transfer (CL1A = FL1A)

b. Detection of an error by the DFI or the Disk File Control (DFC).

If the DFI detects the condition CL1A = FL1A or detects an error, it sends an I/O-complete signal to the DFC. When the DFC detects either the I/O-complete signal or an error, it sends a result descriptor available (RDAV) signal along with the next ARL signal. The DFI accepts the result descriptor and stores it in the FL1A field of the channel descriptor. The DFI sends the terminating channel number to the translator, and then awaits receipt of a Read-Result-Word request. When this request is received at the DFI, the result descriptor is sent to the translator and the channel termination phase is completed.

FUNCTIONAL COMPONENTS OF THE DFI

Figure IV-2-4 illustrates the two DFI sections and their respective interfaces with the Translator, MIU, and Peripheral Control Cabinets (PCC). The two DFI sections are identical and contain the following components:

a. Channel Descriptor Local Memory (LMC) - Provides storage for four 66-bit channel descriptors; one for each DFC channel.

b. Descriptor Register (DR) - Used to store the descriptor of the active channel. The DR contents are used in conjunction with update logic to update the current memory address (CL1A) of the active job and to update various control bits.

c. Update Logic - Used to update the CL1A by two words each time a memory access is requested. It is also used to update the Word/ Byte Position (W/BP) field, residue and phase (FAZ) fields, and various other control bits.

d. Parity Check and Generate Logic - Generates and stores odd parity for each descriptor to be stored in the LMC, and checks for odd parity on all descriptors read from the LMC.

e. Data Local Memory (LMD) - Provides storage for sixteen 48-bit data word locations (two double-word locations for each of the 4 DFC channels). The LMD acts as a buffer for data read from or written onto disk files. Also, during initiation of a request, the LMD contains the 6 CDL characters (48 bits total) which are sent to the DFC.

f. Two-word Buffer - Acts as a buffer for data being transferred between the LMD and the MIU. It contains storage for two 48-bit words.

g. Transfer Register (TR) - A 16-bit register used to buffer all data transfers to or from the DFC.

h. Parity Check, Generate, Accumulate, and Store - Checks and generates parity on data transferred from or to the DFC. When data is sent to the DFC, a parity bit is received from the MIU with each data word, and is stored in the accumulator. The parity bit setting of the accumulator is updated with each 16-bit transfer to the DFC, and is checked against the parity of the last such transfer. When data is received from the DFC, a parity bit is received with each data transfer and is stored in the accumulator. The parity bit setting is updated for each new 16-bit transfer, and a final setting is sent to the MIU for each new 48-bit data word. The MIU then checks parity on the full 48-bit word (three 16-bit transfers per 48-bit word).

FUNCTIONAL OPERATION OF SCI

The SCI (figure IV-2-5) contains the storage and controls required to provide a scan bus for communication with four DCP's and four DFO's. The scan bus to the four DFO's is shared between two IOMs.

The translator initiates scan operations by transmitting a scan control word to the SCI. If a scan-out is required, the translator also transmits the scan-out information to the SCI. Upon completion of the scan operation by the SCI, the translator is notified. In the case of a scan-in operation, the scan-in information is loaded into the translator B register. If an error has been detected by the SCI, error information is loaded into the translator F register.

There are two error conditions which can be reported to the translator by the SCI:

a. Not Ready - If the DFO or DCP addressed by the scan bus does not respond with a ready signal within 3 usec, a not ready error is reported to the translator.

b. Module Errcr - If the DFO or DCP addressed by the scan bus detects an error on a scan-out or scan-in operation, an error signal is transmitted to the SCI. The SCI then reports a scan error to the translator.

SCAN INTERFACE

A DFO is selected by means of the 8-bit EUD code presented over the scan address lines. An EU may be connected to 2 DFOs (directly to one DFO and indirectly via it to the second DFO). Therefore, when the system places an EUD code on the scan bus lines in order to communicate with a DFO, both DFOs could respond were it not for a means of inhibiting the response of one DFO.

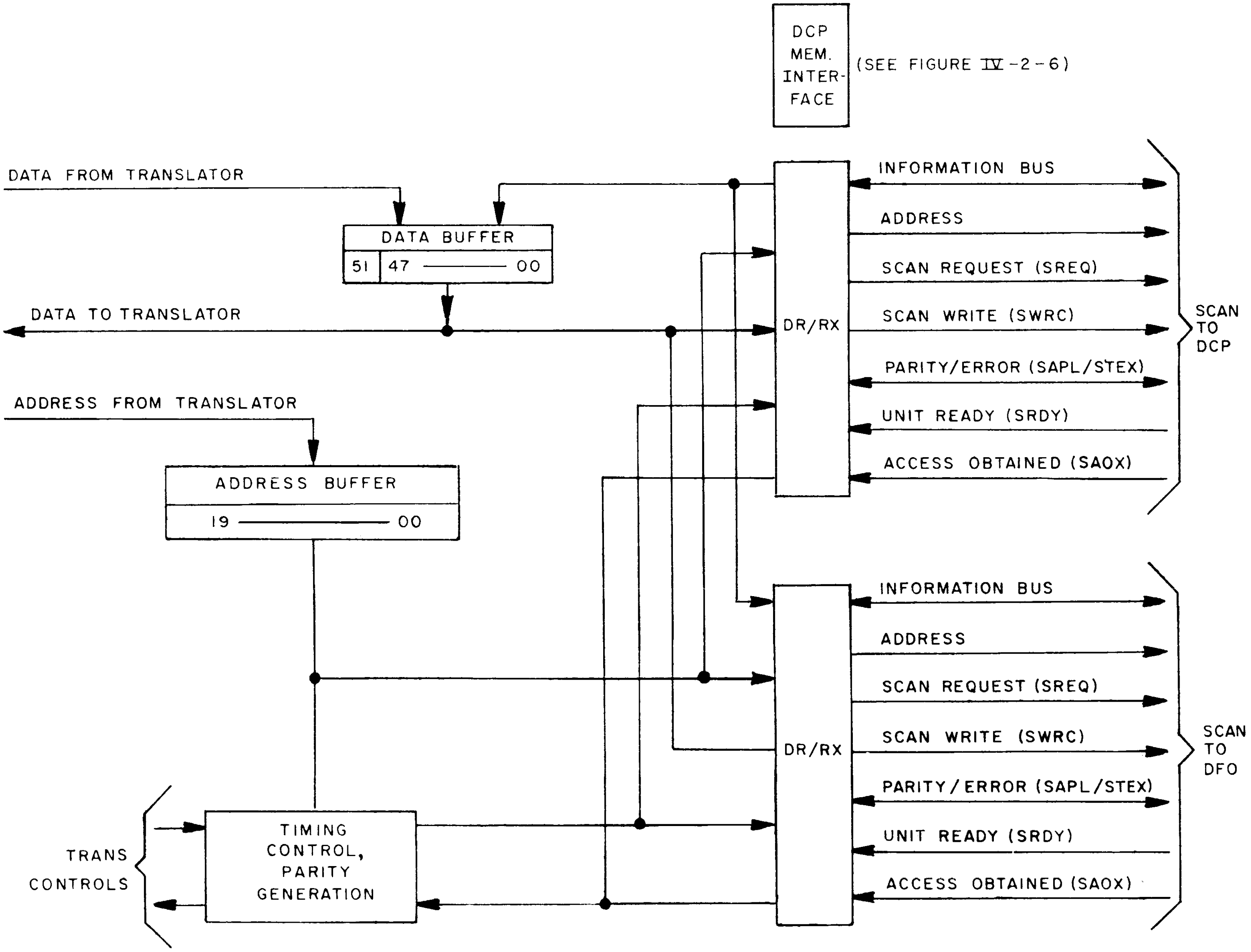


Figure IV-2-5. Scan Bus Interface

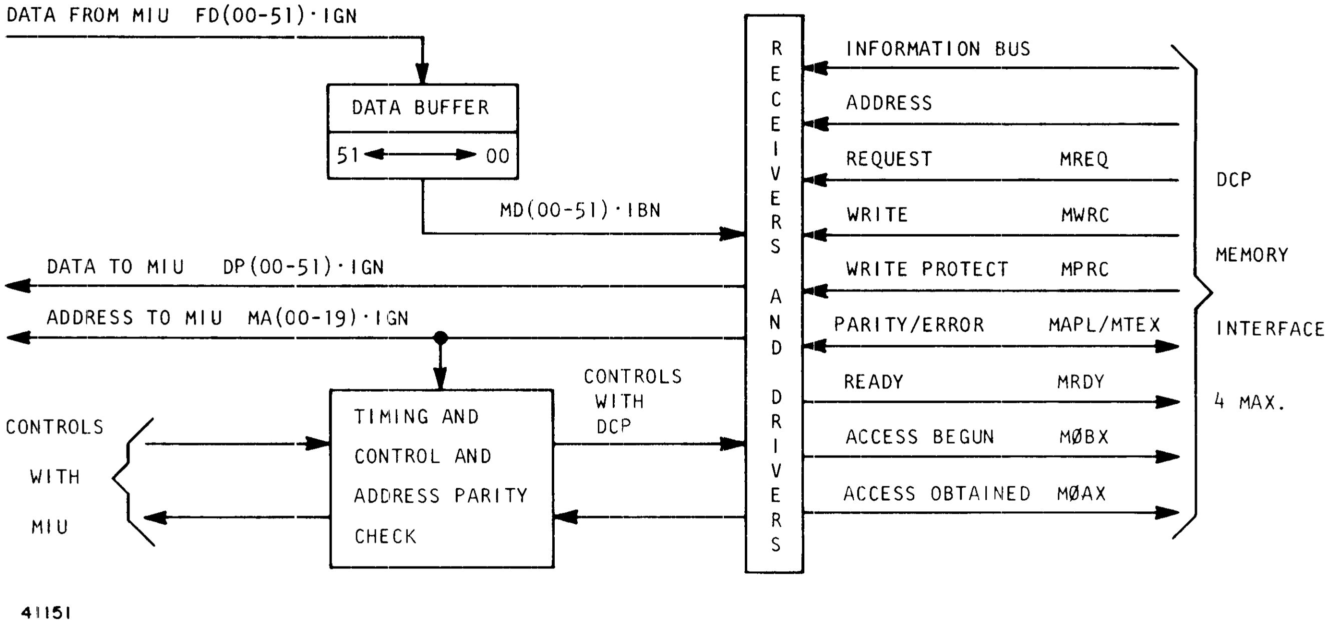


Figure IV-2-6. DCI Unit

This means is bit 7 (ES) of the scan address lines. If bit 7 is low, the DFO connected directly to the referenced EU is selected. If bit 7 is high the DFO connected indirectly to the referenced EU is selected. During execution of the report request operation, the DFO responds regardless of the state of bit 7, if it is not part of a DFO pair. The scan information lines constitute the scan-in and scan-out words during the corresponding operations.

DCP SCAN INTERFACE

During scan-out operations (only scan-out orders are accepted by a DCP), the scan information lines constitute the scan-out word. The SCI provides a maximum of four DCP memory interfaces (see figure IV-2-6) in a DCI unit.

The DCI unit contains all storage capability and controls required to interface with the DCP memory buses. The memory transfer operations performed are:

a. Fetch (one word)

b. Store with flashback (one word)

c. Protected store with flashback (one word).

If an interface is not used by a DCP, it may be used to accommodate a suitable device.

All errors detected by the DCI or MIU for a DCI memory request are transmitted to the DCP that initiated the memory request.

SECTION 3
PERIPHERALS AND CONTROL WORD FORMATS

INPUT/OUTPUT CONTROL WORD (IOCW)

	LINK 47	MINH 43	B/F 39	35	31	27	23	19	15	11	7	3
0 50	ASCII 46	TRA 42	TEST 38	34	30	26	22	18	14	10	6	2
0 49	SA 45	FML 41	T C 37	33	29	25	21	17	13	9	5	1
0 48	I/O 44	MP 40	T L 36	32	28	24	20	16	12	8	4	0

JOB WORD 1 (JBW1) TO DSU

	LINK 47	MINH 43	C T 39	35	31	27	23	19	15	11	7	3
B/F 50	ASCII 46	TRA 42	E X 38	F 34	I N 30	A 26	L 22	18	S T 14	A R 10	T 6	2
T C 49	SA 45	FML 41	T 37	A D 33	D R 29	E 25	S S 21	A 17	D D 13	R E 9	S S 5	1
T L 48	I/O 44	MP 40	36	32	28	24	20	16	12	8	4	0

FIELD	IOCW BIT S	JBW1 BIT S	DESCRIPTION
LINK	47:1	47:1	SIDELINK. When set, indicates a sidelinked I/O is to be performed. Info about sidelink operation is in second word of IOCB.
ASCII	46:1	46:1	ASCII.
SA	45:1	45:1	SOFTWARE ATTENTION. When set, causes ATT bit in result descriptor to be set.
I/O	44:1	44:1	INPUT/OUTPUT. 1 = READ; 0 = WRITE
MINH	43:1	43:1	MEMORY INHIBIT.
TRA	42:1	42:1	TRANSLATE. Settings of ASCII, I/O, TRA and FML bits, taken together, determine what data translation, if any, is to be done.
FML	41:1	41:1	FRAME LENGTH. 1 = 8 bits; 0 = 6 bits.
MP	40:1	40:1	MEMORY PROTECT.
B/F	39:1	50:1	BACKWARD/FORWARD. 1 = BKWD; 0 = FWD.
TEST	38:1	-	TEST.
TCTL	37:2	49:2	TAG CONTROL. 0 = Store SP tags (0); 1 = Store program tags (3); 2 = Tag field transfer; 3 = Store DP tags (2).
CT EXT	-	39:3	COUNT EXTENSION. Number of characters in fractional word part of data buffer.
FINAL ADDRESS	-	36:17	Memory address of last full word of buffer to be accessed by IOM, low 17 bits only.
START ADDRESS	-	19:20	Memory address of first full word of buffer to be referenced; for a backward tape operation, this is *not* the buffer base address.

Job word 1 is built in the IOM and can be read from the B register (Panel 1, Row 3) in T-time 25 of start mode. Job word 2 (JBW2) passed to the data service unit is in all cases the channel described in detail on the following pages.

STANDARD RESULT DESCRIPTOR

	47	43	39	35	31	C C H O	23	19	DVE 15	CTE 11	DPE 7	NTR 3
T 50	46	ME 42	MO 38	RY 34	30	A U N	UN 22	IT 18	NBE 14	IFE 10	ME2 6	BSY 2
A G 49	45	AD 41	DR 37	ESS 33	29	R. T 25	N 21	O. 17	13	BSE 9	ME1 5	ATT 1
48	44	40	36	32	28	24	20	ME3 16	CME 12	CPE 8	DSE 4	EXC 0

Field	BITS	DESCRIPTION
TAG	50:3	0 = Normal; 4 = IOM unable to do sidelinked job.
MEMORY ADDRESS	47:20	Final level 1 memory address (L1A). Hard load RD contains channel used in [32:5].
CHAR COUNT	27:3	No. of last memory word characters validly executed by IOM.
UNIT NO	24:8	Unit No. of device on which job was executed. (Set only for mapped I/O).
ME3,ME2,ME1	16:1 6:1 5:1	MEMORY RELATED ERRORS. Listed in order of decreasing priority.

ME3	ME2	ME1	IOM*	
0	1	1	PER	STORE DISPARITY.
0	1	0	RAE	L1A ADDRESS-RESIDUE ERROR.
1	1	1	F1R	MCM-DETECTED ERROR, FAIL 1 (UNCORRECTABLE).
1	0	1	NOA	NO ACCESS TO MEMORY.
1	1	0	PER	FETCH DISPARITY.
1	0	0	RS1	MEMORY-PROTECT ERROR.
0	0	1	F2R	MCM-DETECTED ERROR, FAIL 2 (CORRECTABLE).

*Panel Light indication in Error Control Register (Panel 2, Row 7).

Unit or DSU error bits 15:9 Depends on value of bit 4 (DSE).

0 = unit related errors (in bits 15:9); see below.
1 = DSU error (in bits 15:9), as follows:

DVE	15:1	DEVICE-DETECTED ERROR.
NBE	14:1	NOT BUSY ERROR.
CME	12:1	COMBINATION ERROR.
CTE	11:1	COUNTER ERROR.
IFE	10:1	INTERFACE ERROR.
BSE	9:1	BUS PARITY ERROR.
CPE	8:1	CONTROL PARITY ERROR.
DPE	7:1	DATA PARITY ERROR (Always set if BSE=1).

DSE	4:1	DATA SERVICE ERROR. 1 = DSU error (in bits 15:9); 0 = Unit related error (in bits 15:9).
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

UNIT RELATED ERRORS

The device RD for any type of unit is returned to the IOM as 3 hex digits from the control. For a PCI operation, the device RD is in the PCI byte buffer (Panel 1, Row 14), as shown below, when the operation completes.

INA8	INA4	INA2	INA1	INB8	INB4	INB2	INB1	INC8	INC4	INC2	INC1	U	U	U	U
15															0

U = UNIT NO. FOR AN EXCHANGE DEVICE, ELSE UNUSED

The INA8 bit should always be on, signifying operation complete. A unit error is signified by INA4=1, and the remaining bits in the device RD are used to make up the RD error field reported to the MCP as shown:

ME3	INC1	INC2	INC4	INC8	INB1	INB2	INB4	INB8	INA1	ME2	ME1	DSE=0	INA2	BSY=0	ATT	EXC
16																0

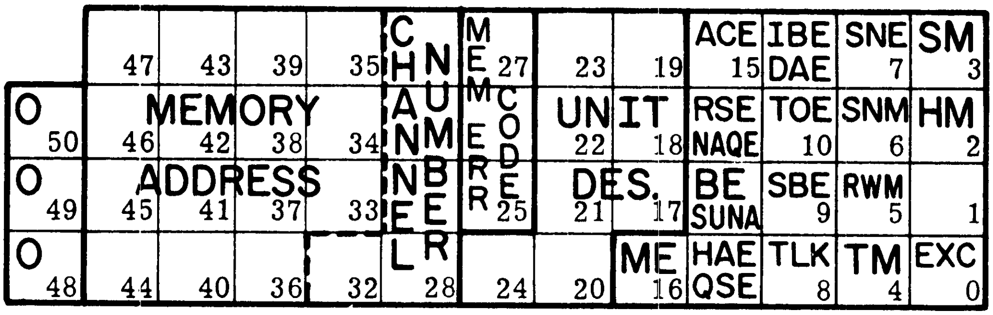
For each PCI channel, the last RD returned to the MCP may be displayed in the RD register (Panel 1, Row 34) of the IOM via local memory operations. Similarly, the last RD from each DFI channel may be displayed (in slightly modified form) in the DFI RD register (Panel 1, Row 10).

RESULT DESCRIPTORS COMMON TO ALL PERIPHERAL DEVICES

0003	ATT (Software Attention)
0005	BSY
0009	NOT READY
000D	WRONG LENGTH DATA TRANSFER (Generated by MCP)
0015	BUSS PARITY (Reported by MCP for hardware RD=0291)
0021	MEMORY FAIL 2 (Correctable)
0041	RESIDUE ERROR
0061	STORE DISPARITY
0291	BUS PARITY (Changed to 0015 by MCP)
10001	MEMORY PROTECT ERROR
10021	NO ACCESS TO MEMORY
10041	FETCH DISPARITY
10061	MEMORY FAIL 1 (Uncorrectable)

Any result descriptor may also have ATT (bit 1) set.

FAIL RESULT DESCRIPTOR



47 43 39 35 27 23 19 ACE 15 IBE DAE 7 SNE 3 SM
O 50 46 42 38 34 22 18 RSE NAQE 10 TOE 6 SNM 2 HM
O 49 45 41 37 33 25 21 17 BE SUNA 9 SBE 5 RWM 1
O 48 44 40 36 32 28 24 20 16 ME HAE QSE 8 TLK 4 TM 0 EXC
MEMORY ADDRESS
CHANNEL NUMBER
MEM ERR CODE
UNIT DES.

41013

FIELD **BITS** **DESCRIPTION**

MEMORY ADDRESS 47:20 This field contains the location in memory that was last accessed at the time of the error. This field is not valid if bit 15 (ACE) is set.

CHANNEL NUMBER 32:5 This field contains a channel number only when bit 15 (ACE) is set.

MEM ERR CODE 27:3 This field contains a memory error code and is valid only when bit 16 (ME) is set. The error bits are interpreted as follows:

1 = Memory detected error.
2 = L1A address residue error.
3 = Store disparity.
4 = Memory protect error.
5 = No access to memory.
6 = Fetch disparity.
7 = Memory detected error.

UNIT DES. 24:8 0 = A Unit Designate of all zeros signifies a Fail Register Result Descriptor.

ME 16:1 MEMORY ERROR. The memory error or MIU detected error is found by decoding bits 27:3 of the Fail Register.

ACE 15:1 ACTIVE CHANNEL STACK ERROR. The address (channel number) of the word in the stack that caused the parity error is contained in bits 47:20.

14:6 If SNE (7:1) = 0, then bits 14:6 are defined as follows:

14:1 RSE RESIDUE ERROR (MEMORY ADDRESS). The address in error is contained in bits 47:20.

13:1 BE BUFFER REGISTER PARITY ERROR.

12:1 HAE HOME ADDRESS ILLEGAL COMMAND.

FIELD **BITS** **DESCRIPTION**

11:1 IBE INITIATE BUSY CHANNEL ERROR. An attempt was made to start a non-exchange channel that was either busy or in the process of being terminated.

10:1 TOE TIME OUT ERROR. A data service time out error.

14:6 If SNE (7:1) = 1, then bits 14:6 represent scan errors and are defined as follows:

14:1 NAQE NO ACCESS TO DFO EXCHANGE.

13:1 SUNA STORAGE UNIT NOT AVAILABLE.

12:1 QSE DFO STACK PARITY ERROR.

11:1 DAE DISK ADDRESS ERROR.

10:1 TOE TIME OUT ERROR. A scan buss time out error.

9:1 SBE SCAN BUSS ERROR. Indicates a parity error on the scan bus.

TLK 8:1 TABLE LOCKED. The translator timed out trying to fetch a locked Unit Table or Status Queue header.

SNE 7:1 SCAN ERROR. When set, indicates that bits 14:6 represent scan errors.

6:5 Indicate the translator mode of operation when the error occurred as follows:

6:1 SNM SCAN MODE.
5:1 RWM RING WALK MODE.
4:1 TM TERMINATE MODE.
3:1 SM START MODE.
2:1 HM HOME ADDRESS MODE.

EXC 0:1 EXCEPTION BIT. Indicates that a "1" exists in the Fail Register.

CARD PUNCH

CARD PUNCH CONTROL

The B 7212 Card Punch Control is used with the B 9213 "300 CPM Punch" which can punch either Binary, Alpha, or EBCDIC code at a rate of 300 cards per minute. Pre-punched cards may be used, but previously punched columns cannot be repunched. The Card Punch has a 1000-card capacity input hopper, and three output stackers (primary, auxiliary, and error) which have a capacity of 1200 card each. Stacker selection is accomplished programmatically.

CDL WORD FORMAT

	47	43	S T 39	35	31	27	23	19	15	11	7	3
O 50	O 46	P 42	A C 38	34	30	26	22	18	14	10	6	2
O 49	CO 45	DE 41	K E 37	33	29	25	21	17	13	9	5	1
O 48	44	40	R 36	32	28	24	20	16	12	8	4	0

41014

FIELD

OP	VAR	ADDR	Operation
23	SIII		Punch BCL
24	SIII		Punch Binary
25	SIII		Punch EBCDIC
99	IIII		Test

S = stacker; 0 = normal, 1 = auxiliary, I = ignored

IOCW FORMATION

Operation	IOCW Bits 46 42 41	CDL OP Code
BCL from Int. BCL	0 1 0	23
BCL from ASCII	1 1 0	23
BCL from EBCDIC	0 1 1	23
Binary (6-bit from 6-bit)	0 0 0	24
EBCDIC from EBCDIC	0 0 1	25
EBCDIC from ASCII	1 0 1	25

RESULT DESCRIPTOR, UNIT ERROR FIELD

To MCP	From Device	Error Type
0081	D00	Punch Check or Memory Access Error
0281	D40	Parity Error
0881	D10	Bus Parity Error

A test-op returns the punch type in bit 10 of the software RD: 0 = Model I; 1 = Model II.

OPERATIONS

BCL (OP 23)

Punch one card on the card punch. The operation is terminated by punching the specified number of words or punching 80 columns. The descriptor word count cannot exceed 10 words for punch BCL. BCL Internal Code, ASCII, or EBCDIC is converted to BCL code by translators in the IOM. The control can include one and only one of the following translators which are used to convert BCL code to BCL card code, ICT card code or BULL card code.

a. BCL-BCL Card Code Translator
b. BCL-ICT Card Code Translator
c. BCL-BULL Card Code Translator

BINARY (OP 24)

Punch one card on the card punch. The operation is terminated by punching the specified number of words or by punching 80 columns. The descriptor word count cannot exceed 20 words for punch binary. A total of 160 six-bit characters of memory are required to punch 80 columns. The contents of each card column are divided into two 6-bit characters. The upper six bits are punched from the first 6-bit character received and the lower six bits from the next 6-bit character. Tag field transfers are compatible with this operator and must not be specified.

CARD PUNCH EBCDIC (OP 25)

Punch one card on the card punch. The operation is terminated by punching the specified number of words or by punching 80 columns. The descriptor word count cannot exceed 13 words for punch EBCDIC. ASCII is translated to EBCDIC by translators in the IOM. The card punch control converts EBCDIC 8-bit code to EBCDIC card code.

TEST (OP 99)

Test the status of the unit and return a result descriptor.

PUNCH CHECK ERROR

When a Punch Check is detected by Card Punch Control 1, the punching of that card is completed. The next card is punched and both cards are sent to the error stacker. The punch check bit is set in the result descriptor returned for the second card.

When Punch Check is detected by Card Punch Control 2, the punching of that card is completed, and it is sent to the error stacker. The Punch Check bit in the result descriptor is set.

The Punch Check bit may be present in the Result Descriptor when addressing a non-present punch, or one that is powered down.

CARD READER CONTROL

The B 7110 Card Reader Control can be used with either B 9111 (800 cpm) or B 9112 (1400 cpm) card readers. The input hopper and the output stacker have a capacity of 2400 cards each. The card readers accept alpha, binary or EBCDIC card codes. The card reader converts alpha card code to BCL, which is then converted into internal BCL or EBCDIC by translators in the I/O Processor. EBCDIC card code is converted to internal EBCDIC by the B 7110 card reader control. When binary punched cards are read no translation is made.

The card readers can read 51–, 60–, or 80-column punched cards. Optional features include the ability to read 40-column Treasury checks and round holes in Postal Money Orders. Cards of varying thickness are acceptable; however, card thickness and length must be consistent during any one run.

CDL WORD FORMAT

	47	43	39	35	31	27	23	19	15	11	7	3
O 50	O 46	P 42	38	34	30	26	22	18	14	10	6	2
O 49	CO 45	DE 41	37	33	29	25	21	17	13	9	5	1
O 48	44	40	36	32	28	24	20	16	12	8	4	0

41017

FIELD

OP	VAR	Operation
20	IIII	Read BCL
21	IIII	Read Binary
22	IIII	Read EBCDIC
99	IIII	Test

I = ignored

IOCW INFORMATION

Operation	IOCW Bits 46 42 41	CDL OP Code
BCL to Int. BCL	0 1 0	20
BCL to ASCII	1 1 0	20
BCL to EBCDIC	0 1 1	20
Binary (60bit to 6-bit)	0 0 0	21
EBCDIC to EBCDIC	0 0 1	22
EBCDIC to ASCII	1 0 1	22

RESULT DESCRIPTOR, UNIT ERROR FIELD

To MCP	From Device	Error Type
0081	D00	Memory-Access Error
0101	C80	Read Check
0281	D40	Validity Check
0381	C80	Read Check and Validity Check
0401	D40	Control Card (generated by IOM)
0881	D10	Bus Parity Error
0889	F10	Bus Parity Error in Initiate Phase*

*3A Control only

OPERATIONS

BCL (OP 20)

Read one card from the card reader. The operation is terminated by reading the specified number of words, or by receiving 80 characters from the reader. The card reader converts BCL card code to BCL code. BCL Code is converted to BCL Internal Control Code, ASCII, or EBCDIC by translators in the IOM.

BINARY (OP 21)

Read one card from the card reader. The operation is terminated by reading the specified number of words, or by receiving 80 columns of information from the reader. The contents of each card column are divided into two 6-bit fields. The upper six bits are stored in memory followed by the lower 6-bits. There is no code translation or invalid code detection Tag field transfers are not compatible with this operator and must not be specified.

EBCDIC (OP 22)

Read one card from the card reader. The operation is terminated by reading the specified number of words, or by receiving 80 characters from the reader. The card reader control converts EBCDIC card code to EBCDIC. EBCDIC is stored as received, or is translated to ASCII by the IOM.

TEST (OP 99)

Test the status of the unit and return a result descriptor.

DISK FILE SUBSYSTEM

A disk file subsystem can include from one to eight controls and from one to twenty electronics units. If more than eight controls are used, they must be divided between IOM's so that not more than 8 controls of 1 subsystem are allocated to one IOM. Model 1A-2, 1C-3, and 1C-4 disks may be mixed in the same subsystem but model 2B disks must have a separate subsystem.

A disk file exchange is required when more than two electronics units are used. A 1 x 2 adapter may be used when two E.U.'s are required. A segment size of 180 bytes (equivalent to 240 6-bit characters) is used in the B 7700 systems. The Disk File Control 4 is used to control disk file electronic units for model 1C and 1A-2 disks.

The Disk File Control 5 is used to control disk file electronic units for model 2B disk.

DISK FILE EXCHANGES

From 1 to 4 disk file controls can be used to form a multiple control subsystem connected to a single IOM.

The 4 x 10 disk file exchange is used to interface disk file control 4 with Model 1 Disk Electronic Units when multiple controls and/or E.U.'s are required in a subsystem. The maximum combination for this exchange is 4 controls and 10 E.U.'s. The number of E.U.'s may be extended up to 20 by adding the 4 x 20 disk file exchange extension and the appropriate adapters.

The 4 x 20 Disk File Exchange 5 is used to interface Disk File Control 5 with the Model 2B Disk Electronic Units when multiple controls and/or E.U.'s are required in a subsystem.

The maximum combination for this exchange is 4 controls and 20 E.U.'s. The number of E.U.'s may be extended up to 8 by adding the 8 x 20 disk file exchange extension and the appropriate adapters.

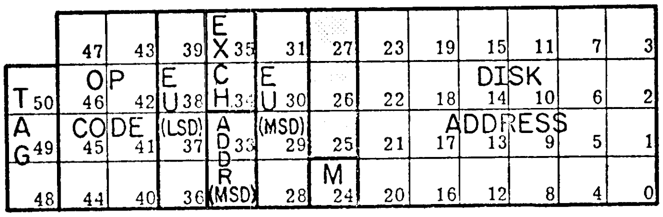
A maximum of 8 controls for a given subsystem are allowed on an IOM. If the 8 x 20 extension is used, more than 1 IOM DF adapter is required and the maximum of 4 controls per DFI adapter must be maintained.

EXCHANGE MODULARITY

The exchanges are modular by the use of adapters. The modularity of the adapters is such that E.U.'s and controls are added in increments of 2 until the maximum complement is reached.

DISK

CDL WORD FORMAT



FIELD

OP	VAR	ADDR	Operation
50	VFGM	AAAAAA	Write
51	VFGM	AAAAAA	Read
52	VFGM	AAAAAA	Check
99	VFii		Test

V = EU No. (LSD of unit designate)
F: (35:2) = MSD of position on exchange, (33:2) = MSD of disk address
G = MSD of EU unit designate, used for DFO operations
M:1 = maintenance segment
A = Disk segment address (6 decimal digits)
i = Ignored

IOCW INFORMATION

OPERATION	IOCW Bits 46 44 43 42 41	CDL OP Code
WRITE	0 0 0 0 1	50
READ	0 1 0 0 1	51
CHECK	0 1 1 0 1	52

RESULT DESCRIPTOR, UNIT ERROR FIELD

To MCP	From Device	Error Type
0081	D00	Memory-Access Error
0101	C80	EU Busy
0201	C40	Write Lockout
0281	D40	Parity Error
0881	D10	Bus Parity Error
8001	C01	Timeout

A test-op returns the diskfile type in [11:2] of the RD to software.
For type IV (DISKI) disk: 0 = 1A2, 1 = 1C3, 3 = 1C4.
For type V (DISKII) disk: 0 = 2B2, 1 = 2B6, 2 = 2B4.

OPERATIONS

WRITE (OP 50)

Write a record on the disk file. The operation is terminated by writing the specified number of words. Incomplete segments are filled out with NULL characters (0000 0000) by the disk file control.

READ (OP 51)

Read a record from the disk file. The operation is terminated by reading the specified number of words.

CHECK (OP 52)

Read a record from the disk file, but do not store any information in memory. If a read error is encountered, the control sets the read error bit in the result descriptor.

ERROR TERMINATION

If, during a read or check operation, a segment parity error is detected, the operation is terminated.

If, during read, check or write operations, a failure to access the IOM is encountered, accesses are terminated and result descriptor returned when the segment is completed. Writes are completed with NULL characters (0000 0000), and a result descriptor stored.

The control will time out if the unit addressed sends no information for more than one disk revolution. The time out bit is set in the result descriptor.

Changing zones or changing disks within a storage unit does not result in a time loss of more than 600 microseconds. Changing storage units can result in a time loss equivalent to one disk revolution.

DISK-PACK DRIVE MEMORY SYSTEM

The Magnetic Actuator Disk-Pack Drive Memory Systems are extremely high-speed, modular, random information storage systems. A basic disk-pack drive memory subsystem includes the disk-pack drive controller, dual disk-pack drive, and the interconnecting cables.

The controller acts upon I/O instructions from the IOM, powers the disk-pack drive, and transfers information between disk-pack drives and the IOM. The controller performs the operation specified by the OP code (and variants) of the CDL Word, and, at the completion of the operation, generates a result descriptor which contains operation and/or error status information.

The disk-pack drive controller with single access capabilities may be used with eight disk-pack spindles (four dual drives) in a one-by-eight configuration, or two groups of eight disk-pack spindles (eight dual drives) in a one-by-16 configuration. Selection of each group is determined by a variant in the I/O descriptor. The disk-pack drive controller with dual access capability may be used in a two-by-eight configuration in which the disk-pack drive controller contains two internal control units. This allows the I/O module to execute two simultaneous operations (two reads, two writes, or a read and a write). This configuration can be expanded to a two-by-16 configuration.

CDL WORD, GENERAL FORMAT

	47	43	U 39	35	31	27	23	19	15	11	7	3
O 50	O 46	P 42	N 38	VAR 34	IAN 30	T 26	22	18	FI 14	LE 10	6	2
O 49	CO 45	DE 41	I 37	33	29	25	21	A 17	DDR 13	ESS 9	5	1
O 48	44	40	T 36	32	28	24	20	16	12	8	4	0

4023

OP	VAR	ADDR	Operation
50	USFV	AAAAAA	Write
51	USFV	AAAAAA	Read
53			Report (reserved for FPM)
54			Unlock (reserved for FPM)
55			Clear (reserved for FPM)
56	USFV	AAAAAA	Initialize
57	USFV	AAAAAA	Verify
58	USFN	AAAAAA	Relocate
99	UVii		Test

U = Unit Designate (LSD); A = Disk Pack Address (six decimal digits)

S1: unconditional/conditional operation

S2: 1 = disable automatic restore function following seek error

F1: 1 = full-track format, 0 = multisector format (full-track format is unimplemented for 225 DPD.)

V = Variant (differs for each operation); N = Spare Sector

i = Ignored

CDL WORD FORMAT, WRITE (OP 50)

	47	43	U 39	S8 35	31	27	23	19	15	11	7	3
O 50	O 46	P 42	N 38	S4 34	30	V4 26	22	D 18	ISK 14	PACK 10	6	2
O 49	CO 45	DE 41	I 37	S2 33	29	25	21	A 17	DDR 13	ESS 9	5	1
O 48	44	40	T 36	SI 32	FI 28	24	20	16	12	8	4	0

S8: 1 = load controller firmware (BX 383 only, overrides all other CDL variants)

S4: 1 = automatic data parity check on all sectors written after write operation.

S2: 1 = disable automatic restore function following seek error.

S1: for BX380, 0 = unconditional operation, 1 = conditional operation.
for BX383, 1 = unconditional operation, 0 = conditional operation.

F1: 1 = full-track format, 0 = multisector format

V4: Reserved for FPM

CDL WORD FORMAT, READ (OP 51)

	47	43	U 39	S8 35	▨	V8 27	23	19	15	11	7	3
O 50	O 46	P 42	N 38	S4 34	▨	V4	22	D 18	ISK 14	PACK 10	6	2
O 49	CO 45	DE 41	I 37	S2 33	▨	SPARE SECTOR	21	A 17	DDR 13	ESS 9	5	1
O 48	44	40	T 36	SL 32	FI 28		20	16	12	8	4	0

S8: 1 = error correction is disabled.

S4: 0 = normal read, 1 = read binary address field.

S2: 1 = disable automatic restore following seek error.

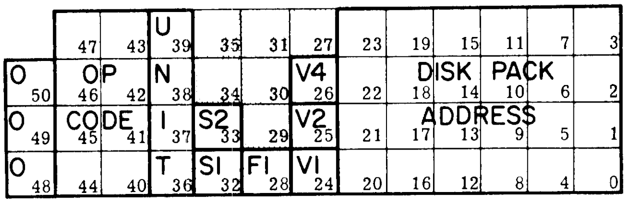
S1: for BX380, 0 = unconditional operation, 1 = conditional operation.
for BX383, 1 = unconditional operation, 0 = conditional operation.

Spare Sector: If S4=1, then value of 1-5 indicates a spare sector. Cylinder referenced will be that containing the specified disk pack address.

F1: 1 = full-track format, 0 = multisector format.

V8,V4: 1, 0 = read log (BX383 only).
V8 = 1 overrides all other CDL variants.
1, 1 = dump buffer (BX383 only).

CDL WORD FORMAT, INITIALIZE (OP 56)



S2: 1 = disable automatic restor function following seek error.

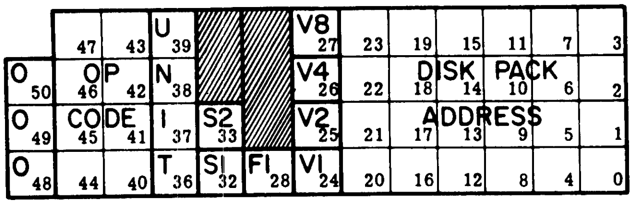
S1: for Bx380, 0 = unconditional operation, 1 = conditional operation.
for BX383, 1 = unconditional operation, 0 = conditional operation.

F1: 1 = full-track format, 0 = multisector format.

V4: 0 = write default pattern into each sector.
BX380, all zeros.
BX383, 215 DPD = "55E7", 225 DPD = "6363".
1 = write provided test data pattern into each sector; first 16 bits of data buffer is used as test pattern.

V2,V1: 0, 0 = initialize entire pack, 0, 1 = initialize designated cylinder,
1, 0 = initialize designated track. Designated track and cylinder are those containing the specified disk pack address.

CDL WORD FORMAT, VERIFY (OP 57)



S2: 1 = disable automatic restore function following seek error.

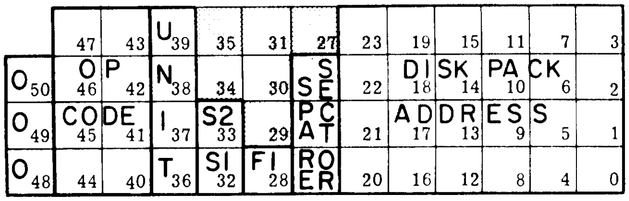
S1: for BX380, 0 = unconditional operation, 1 = conditional operation.
for BX383, 0 = conditional operation, 1 = unconditional operation.

F1: 1 = full-track format, 0 = multisector format.

V8,V4: 0, 0 = check data fields for parity only.
1,0 = compare data fields against default pattern.
x, 1 = compare data fields against last initialization pattern used.

V2,V1: 0, 0 = verify entire pack, terminate on first error.
0, 1 = verify designated cylinder, report all errors.
1, 0 = verify designated track, report all errors.

CDL WORD FORMAT, RELOCATE (OP 58)



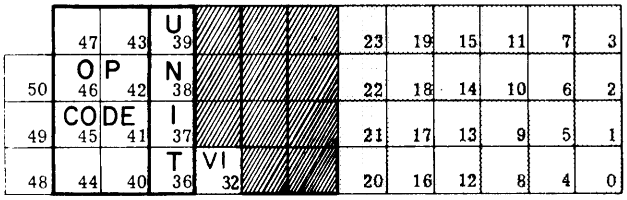
S2: 1 = disable automatic restore function following seek error.

S1: for BX380, 0 = unconditional operation, 1 = conditional operation.
for BX383, 0 = conditional operation, 1 = unconditional operation.

F1: 1 = full-track format, 0 = multisector format

Spare Sector: The sector specified in the address field is relocated to the designated spare sector of the proper cylinder.

CDL WORD FORMAT, TEST (OP 99)



41029

V1: 1 = selected drive is taken off-line for pack removal

READ BINARY ADDRESS OPERATION

A read binary address operation returns the address information written on the pack for the specified sector in the first word of the data buffer as follows:

[47:8] = HEX "FF"
[39:24] = sector address
[15:15] = ignored

The sector address is stored in cylinder - head (track) - sector form and has the following format:
[23:8] Sector
[15:9] = Cylinder
[6:6] = Head (track)
[0:1] = Parity bit (overall parity is odd)

An unused spare sector has the spare number in both the sector and head fields of its address. An unusable (relocated) sector will have all bits on in the sector address returned. An in-use spare sector contains the address of the sector that has been relocated to it, except that the sector field is modified by having the spare number added to it.

SECTOR ADDRESS CONVERSION

To convert a decimal disk pack address to cylinder-head (track) sector form, the algorithms are as follows:

Cylinder = Address DIV C1.
Head = ((Address MOD C1) +5) DIV C2.
Sector = Address MOD C1 (for Head = 0)
Sector = ((Address MOD C1) +5) MOD C2 (for Head = 0)
where
C1 = sectors per cylinder
C2 = sectors per track (head)

For 215 Disk Pack Drives, C1 = 655 and C2 = 33.
For 225 Disk Pack Drives, C1 = 1195 and C2 = 60.

The last five sectors of track (head) 0 of each cylinder are the spare sectors for that cylinder. By convention, the last sector is spare 1, the next to last is spare 2, etc. Thus, for a 225 DPD spare sector 5 for a cylinder is track (head) = 0, sector = 55 for that cylinder.

IOCW

Operation	IOCW Bits						CDL
	46	44	43	42	41	40	OP Code
Write	0	0	0	0	1	0	50
Read	0	1	0	0	1	0/1	51
Initialize	0	0	0	0	1	0	56
Verify	0	1	0	0	1	0/1	57
Relocate	0	0	1	0	1	0	58

RESULT DESCRIPTOR, UNIT ERROR FIELD

BX383			BX380	
To MCP	From Device	Description	To MCP	From Device
0009	E00	Drive Not Ready/Not Present, Nonexistent Address, Unsafe Condition	0009	E00
0081	D00	Memory Access Error	0181	D80
0101	C80	Sector Address Parity Error	0101	C80
0109	E40	Controller in Local Controller Cleared during Operation	0089	F00
0201	C40	Drive Seeking	0201	C40
0301	CC0	First Action with Drive	0301	CC0
0401	C20	Drive Busy	1001	C08
0481	D20	Speed Error during Initialize	1081	D08
0501	CA0	Write Lockout	0501	CA0
0581	DA0	Data Error on Read	0081	D00
0801	C10	Seek Error	0801	C10
0809	E10	Control Malfunction	–	–
0881	D10	Data Parity Error (System to HTC) Write Parity Error on Info Bus	0881	D10
0889	F10	Command Parity Error (System to HTC)	–	–
0901	C90	Sector Timeout	0901	C90
0981	D90	Link Parity Error (DPDC to HTC)	–	–
0A01	C50	Seek Initiated	0A01	C50
0C01	C30	Data 1-bit Error Correction on Read	2801	C14
4101	C82	Address Position Error	1101	C88
8801	C11	Seek Timeout	1801	C18
8881	D11	Data Parity Error (HTC to DPDC) Transmission Parity Error	1181	D88
8889	F11	Command Parity Error (HTC to DPDC)	–	–
C801	C13	Data Correctable Error Retry on Read	–	–

A test-op returns controller and drive information in bits [15:2[of the RD as follows:
0 = drive not present, 1 = BX380 (215 DPD), 2 = BX383 with 215 DPD,
3 = BX383 with 225 DPD.

OPERATIONS

WRITE (OP CODE 50)

Information is written from memory onto disk at the designated file address. Partial sector writes result in a fill of bit zeros in the remainder of the sector.

READ (OP CODE 51)

Information is read from the selected file address into the main memory address. A partial sector read results in the termination of data transfer, but control release does not occur until completion of the total sector read. The controller times out if the read operation is not completed.

INITIALIZE (OP CODE 56)

The controller writes addresses and gaps on all tracks, starting with an index on the track of the designated address. The controller may also write a test pattern consisting of all zeros.

VERIFY (OP CODE 57)

The controller reads from the disk-pack and checks for address errors and information parity errors. The verification of disk sectors begins with the first sector following the index on the track specified by the disk file address and continues through the entire track, cylinder, or disk-pack. The sector positions are verified by counting from the index on each track.

The BCD file address (normal or spare) containing the detected error(s) is reported back between the begin and end memory address (or between the begin address plus 6 bytes and the end address if V $_4$ is 1). Address parity checked for all relocated sectors.

RELOCATE (OP CODE 58)

The controller flags the sector address field of the unusable sector with an error configuration and rewrites this address in the spare sector. The original address field and the relocated address field are located by counting from the index on each track.

NOTE

> An error configuration is one byte of binary "ones" with clock pulses omitted, followed by one byte of binary "ones" (with clock pulses), followed by one byte of binary "ones" with clock pulses omitted.

The error configuration is written into the address field of the designated sector. The standard test data pattern is written into the relocated sector data field. If full-track format is specified, the address is flagged but no relocation takes place.

TEST (OP CODE 99)

The controller returns an appropriate result descriptor indicating the type of drive accessed, the peripheral control type, and (in a shared system configuration), the number of the assigned processor.

EXCEPTION CONDITIONS

SINGLE BIT ERROR CORRECTION (SBE)

A single bit error in a 90-byte data block is detected and corrected during a read operation.

DISK-PACK DRIVE BUSY (TIME-OUT) (DRB)

The operation is terminated if the drive accessed is being used by another controller as an operation is initiated and the drive does not become available within 50 milliseconds.

SPEED ERROR (1081)

The operation is terminated if there is a failure to write a full track of information between index pulses during an initialize operation.

ADDRESS POSITION ERROR (VERIFY) (1101)

A valid address is found at the wrong position relative to the index during a verify operation.

TRANSMISSION PARITY ERROR (1181)

The operation is terminated at the end of the sector being processed if a parity error is detected while data is being transmitted during a write operation.

SEEK TIME-OUT (1801)

The controller terminates the operation when an accessed drive fails to complete the initiated seek within one second.

SEEK ERROR (DSK)

Bit 8 is set and the operation is terminated if a seek is initiated which positions the heads over the wrong cylinder during a read or write operation (as verified by the controller reading the address field from the track on the cylinder in question). Bit 8 is set and the operation is not terminated if a seek is initiated which positions the heads over the wrong cylinder during a verify operation.

SECTOR TIME-OUT (0901)

The operation is terminated if a specified sector address, or spare sector to which that address has been relocated, cannot be found during a read, write, or relocate operation.

WRITE-LOCKOUT (0501)

A write, initialize, or relocate operation is not initiated if the drive to be accessed is in the write lockout state. (The operator failed to press the WRITE ENABLE button on the drive unit.) On a test operation, if the drive is detected to be in a write lockout state, the operation will be executed and bits 5 and 7 will be set in the test result descriptor.

FIRST ACTION (0301)

At the beginning of any operation, if it is detected that this is the first command to a drive that has been powered up, the operation is not initiated.

MEMORY INTERFACE PARITY ERROR (0081)

The controller terminates the operation at the end of the sector being processed if an information parity error is detected on the memory or I/O Processor interface.

CONTROL CLEARED (0089)

If the disk pack controller is cleared during any operation, the operation in process is terminated. The termination is instantaneous and if the controller was writing to the disk at the time Clear was detected, it could destroy the information on the track.

GENERAL INFORMATION

Each disk pack contains 11 disks and 20 recording surfaces, each surface accessed by an individual arm from the actuator. Each disk surface contains 406 tracks. (See figure IV-3-1).

The data transfer is bit-serial. The maximum byte capacity, transfer rate, and other pertinent information for the various disk-pack subsystem are presented in table IV-3-1.

MULTISECTOR-PER-TRACK (STANDARD) FORMAT

There are 33 sectors/track in the multisector format (table IV-3-2), and information within the track is located by sector address (pack address). When a pack is initialized, addresses are written sequentially, sector by sector, throughout each cylinder in the disk-pack.

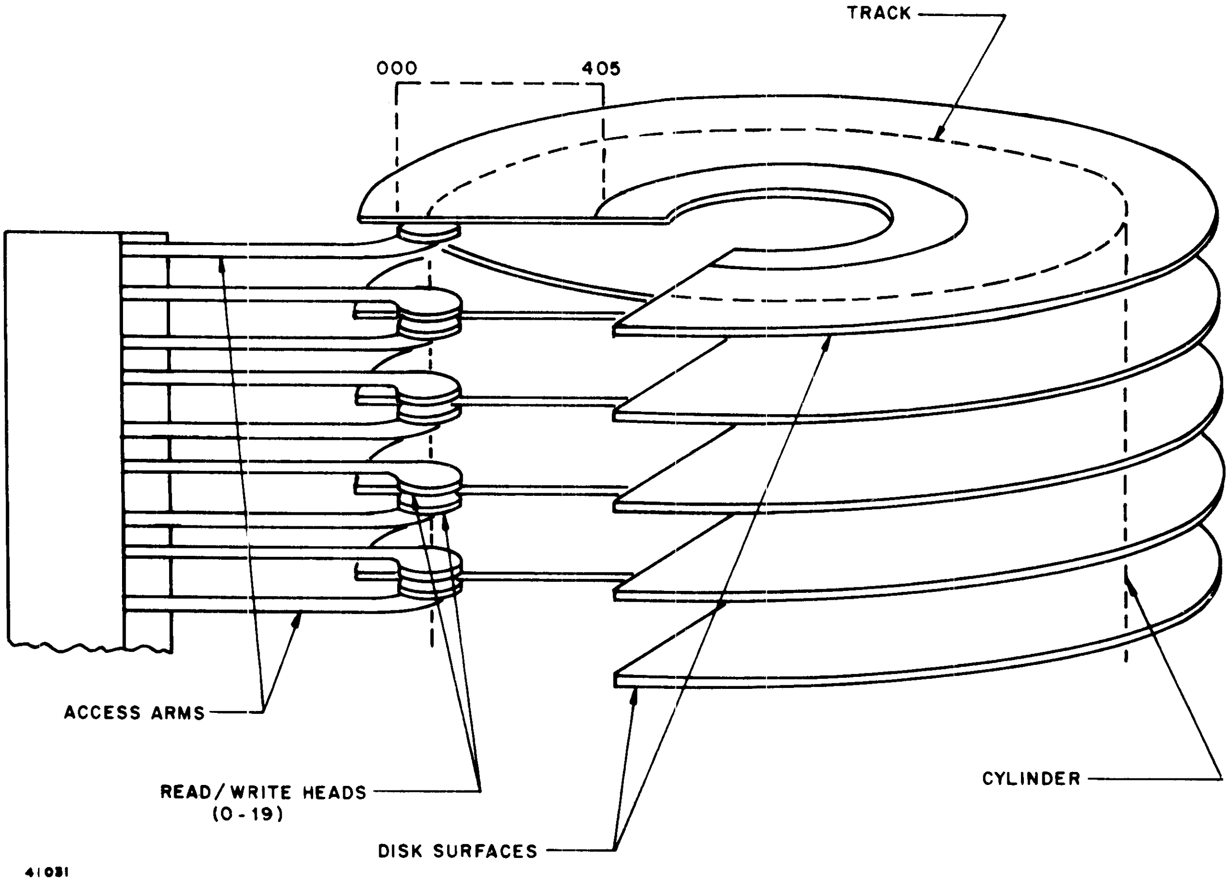


Figure IV-3-1. Disk-Pack Recording Surfaces

Table IV-3-1. Disk-Pack Subsystem Characteristics

DISK PACK DRIVE STYLE NO.	DESCRIPTION	AVERAGE ACCESS TIME (MS)	AVERAGE LATENCY (MS)	STORAGE CAPACITY PER DISK-PACK DRIVE		DATA TRANSFER RATE	PACK DATA		
				MULTI-SECTOR MODE*	FULL TRACK MODE*		MAX. RECORDING DENSITY (BPI)	TRACK DENSITY (TPI)	DISK-PACK STYLE NO.
B 9484-3	Dual drive with single access disk-pack drive controller (B 7380-1)	30	12.5	95.5	121.0	312.5 KB	2200	200	B 9974-1
B 9485-3	Dual drive with dual access disk-pack drive controller (B 7380-2)	30	12.5	95.5	121.0	312.5 KB	2200	200	B 9974-1
B 9486-3	Dual drive add on without disk-pack drive controller	30	12.5	95.5	121.0	312.5 KB	2200	200	B 9974-1
B 9484-4	Dual drive with single access disk-pack drive controller (B 7383-1)	30	12.5	174.4	242.0	625.0 KB	4400	200	B 9974-4
B 9485-4	Dual drive with dual access disk-pack drive controller (B 7383-2)	30	12.5	174.4	242.0	625.0 KB	4400	200	B 9974-4
B 9486-4	Dual drive add on without disk-pack drive controller	30	12.5	174.4	242.0	625.0 KB	4400	200	B 9974-4
B 9486-45	Single drive add on without disk-pack drive controller	30	12.5	87.2	121.0	625.0 KB	4400	200	B 9974-4

*Million eight-bit bytes.

Table IV-3-2. Disk-Pack File Addresses (Burroughs Multi-Sector Format)

	HEAD 0		HEAD 1		HEAD 2		HEAD 18		HEAD 19	
CYL. NO.	SECTOR	FILE ADDRESS	SECTOR	FILE ADDRESS	SECTOR	FILE ADDRESS	SECTOR	FILE ADDRESS	SECTOR	FILE ADDRESS
0	0	000000	0	000028	0	000061	0	000589	0	000622
	1	000001	1	000029	1	000062	1	000590	1	000623
	2	000002	2	000030	2	000063	2	000591	2	000624
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	27	000027	27	000055	27	000088	27	000616	27	000649
Spare Sectors	28	*Addr+1	28	000056	28	000089	28	000617	28	000650
	29	Addr+2	29	000057	29	000090	29	000618	29	000651
	30	Addr+3	30	000058	30	000091	30	000619	30	000652
	31	Addr+4	31	000059	31	000092	31	000620	31	000653
	32	Addr+5	32	000060	32	000093	32	000621	32	000654
1	0	000655	0	000683	0	000716	0	001244	0	001277
	1	000656	1	000684	1	000717	1	001245	1	001278
	2	000657	2	000685	2	000718	2	001246	2	001279
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	27	000682	27	000710	27	000743	27	001271	27	001304
Spare Sectors	28	Addr+1	28	000711	28	000744	28	001272	28	001305
	29	Addr+2	29	000712	29	000745	29	001273	29	001306
	30	Addr+3	30	000713	30	000746	30	001274	30	001307
	31	Addr+4	31	000714	31	000747	31	001275	31	001308
	32	Addr+5	32	000715	32	000748	32	001276	32	001309
405	0	265275	0	265303	0	265336	0	265864	0	265897
	1	265276	1	265304	1	265337	1	265865	1	265898
	2	265277	2	265305	2	265338	2	265866	2	265899
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	27	265302	27	265330	27	265363	27	265891	27	265924
Spare Sectors	28	Addr+1	28	265331	28	265364	28	265892	28	265925
	29	Addr+2	29	265332	29	265365	29	265893	29	265926
	30	Addr+3	30	265333	30	265366	30	265894	30	265927
	31	Addr+4	31	265334	31	265367	31	265895	31	265928
	32	Addr+5	32	265335	32	265368	32	265896	32	265929

* Address of last non-spare sector, plus number of spare (1→5) specified in F variant.

The segment size of Burroughs magnetic actuator disk-pack drives can be selected by the user as either 180-bytes or full track of 7470 or 14,940 bytes. When the 180-byte segment or multi-sector mode is selected, inter-record gaps are introduced and, accordingly, as on all disk-packs where the full-track facility is not used, the 121,000,000-byte capacity will be reduced to 95,500,000 bytes; the 242,000,000 byte capacity will be reduced to 174,400,000 bytes.

SPARE SECTORS

Five sectors are reserved for each cylinder on track 00 (surface under head 00) for relocating data from other sectors in the disk-pack which have been designated as unusable. Data will be relocated between tracks in the same cylinder only.

The method of locating spare sectors so that data is not lost is a software function. If more than five unusable sectors are encountered in one cylinder, the additional unusable sectors are disposed of by software via the XP message.

One format is used to address all sectors which are not spares. This format uses a continuous 6-digit BCD address to designate contiguous sectors beginning at sector 0 (first sector after index) on head 0, cylinder 0 and continuing by sector, head, and cylinder in that order. The BCD file address may be related to the actual disk-pack cylinder head and sector by the following equalities:

a. Burroughs Multi-Sector Format (figure IV-3-2):

For head 0: (655*cylinder) + sector = file address
For other heads: (655*cylinder) + (33*head) + (sector-5) = file address.

b. Single Sector/Track Format (figure IV-3-3):

For all heads: (20*cylinder) + head - file address.

The other format is the single sector per track format (table IV-3-3). In figure IV-3-3 the entire track is shown first, followed by an enlargement of the data and address fields. There are no spares in this format.

Spare sectors (sectors 28 through 32) on surface 0 of each cylinder are addressed with the following six-digit patterns:

a. Two digits of undigit 5's (F) to delimit the field.

b. Three BCD digits to indicate the cylinder on which the spare is located.

c. One BCD digit indicating spare sector 1 through 5 as needed.

UNDIGIT 5	SPARE SECTOR	UNDIGIT 5

LINE PRINTER

PRINTER CONTROL

Four basic line printers are available for use on the B 7700 system.

1. B 9242-11 - 860 LPM, 120 print positions (includes forms self-align feature).

2. B 9243-11 - 1100 LPM, 120 print positions (includes forms self-align feature).

3. B 9246-2 - 1800 LPM, 132 print positions.

4. B 9247-14 - 1100 LPM, 132 print positions (includes 12-channel format tape reader).

Table IV-3-3. Disk-Pack File Addresses (Single Sector/Track Format)

CYL. NO.	HEAD 0	HEAD 1	HEAD 2	HEAD 18	HEAD 19
	FILE ADDRESS	FILE ADDRESS	FILE ADDRESS	FILE ADDRESS	FILE ADDRESS
0	000000	000001	000002	000018	000019
1	000020	000021	000022	000038	000039
2	000040	000041	000042	000058	000059
3	000060	000061	000062	000078	000079
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
405	008100	008101	008102	008118	008119

TRACK FORMAT (MULTI-SECTOR MODE)

COMPLETE TRACK

BOT GAP (A) ADDRESS SYNC - 9 BYTES

24 BYTES (No specific pattern)	5 BYTES Binary "ones"	2 BYTES Binary "zeros"	1 BYTE * "ones"	1 BYTE Binary "ones"
Δ Reserved for variation in detection of index	Sync VFO	Lock phase of VFO	* Clock pulses omitted	Code

SECTOR ADDRESS - 3 BYTES PREAMBLE GAP - 11 BYTES

1 BIT Flag	7 BITS Sector	9 BITS Cyl	6 BITS Head	1 BIT Parity	3 BYTES (No specific pattern)	5 BYTES Binary "ones"	2 BYTES Binary "zeros"	1 BYTE Binary "ones"
Set if sector is a spare					Read to Write switch time	Sync VFO	Lock phase of VFO	Code

DATA FIELD - 184 BYTES (LESS 2 BITS)

90 BYTES Data	11 BITS Check code *	90 BYTES Data	11 BITS Check code *	1 BYTE Count ** character

*Error correction for 90 bytes of data

** 8-bit count of ones - excess 256 (detects multiple bit errors not detected by 11-bit check code)

Figure IV-3-2. Standard Format (Sheet 1 of 2)

POSTAMBLE GAP - 24 BYTES		(B) EOT GAP - 42 to 353 BYTES	
9 BYTES Binary "ones"	15 BYTES Binary "ones"	0 to 311 BYTES (No specific pattern)	42 BYTES Spare
Reserved for space variation in writing data	Reserved for Write to Read switch time	Reserved for speed variation (± 2%)	△

Note: (A) through (B) is repeated 32 times except no postamble on last sector.

ADDRESS FIELD OF SECTOR FLAGGED AS AN ERROR SECTOR (PLUS LAST TWO BYTES OF ADDRESS SYNC)

(PARTIAL) ADDRESS SYNC			
1 BYTE *Binary "ones"	1 BYTE Binary "zeroes"	1 BYTE *Binary "ones"	2 BYTES Binary "zeroes"
*Clock pulses omitted	Code byte changed to "zeroes"		

ADDRESS FIELD OF SECTOR WITH RELOCATED SECTOR INFORMATION

SECTOR ADDRESS FIELD				
1 BIT Flag	7 BITS Sector	9 BITS CYL	6 BITS Head	1 BIT Parity
Bit is a "zero"				

ADDRESS FIELD OF UNUSED SPARE SECTOR

SECTOR ADDRESS FIELD				
1 BIT Flag	7 BITS Sector	9 BITS CYL	6 BITS Head	1 BIT Parity
Bit is a "one"				

Figure IV-3-2. Standard Format (Sheet 2 of 2)

COMPLETE TRACK

BOT GAP (A)	ADDRESS SYNC - 9 BYTES			
24 BYTES (No specific pattern)	5 BYTES Binary "ones"	2 BYTES Binary "zeroes"	1 BYTE * "ones"	1 BYTE Binary "ones"
△ Reserved for variation in detection of index (△)	Sync VFO	Lock phase of VFO	*Clock pulses omitted	Code

SECTOR ADDRESS - 3 BYTES				PREAMBLE GAP - 11 BYTES			
1 BYTE Binary "ones"	9 BITS CYL	6 BITS Head	1 BIT Parity	3 BYTES (No specific pattern)	5 BYTES Binary "ones"	2 BYTES Binary "zeroes"	1 BYTE Binary "ones"
				Read to Write switch time	Sync VFO	Lock phase of VFO	Code

DATA FIELD - 7586 BYTES (LESS 7 BITS)

(A)	(B)		
90 BYTES Data	11 BITS Check code*		1 BYTE Block count**
*Error correction for 90 bytes of data		(A) through (B) repeated 82 times	**8-bit count of "ones" excess 256 (Detects multiple bit errors not detected by 11-bit check code)

EOT GAP - 24 TO 335 BYTES	
0 to 311 BYTES (No specific pattern)	24 BYTES Spare
Reserved for speed variation (± 2%)	△

Figure IV-3-3. Single Sector For Track Format

The B 7240 printer control is used with either B 9242-11, B 9343-11, or B 9246-1 line printer. The B 7243 and B 7247 printer controls are used with the B 9247-13 and B 9247-14 line printers, respectively.

CDL WORD FORMAT

	47	43	S 39	U 35	S 31	C 27	23	19	15	11	7	3
O 50	O 46	P 42	P A 38	N 34	K I 30	H A 26	22	18	14	10	6	2
O 49	CO 45	DE 41	C E 37	I 33	P 29	N N 25	21	17	13	9	5	1
O 48	44	40	36	T 32	T O 28	E L 24	20	16	12	8	4	0

41032

FIELD

OP	VAR	Operation
10	SUNN	Write
11	SUNN	Skip
99	iUii	Test

NN = Skip to Channel (00-11); two decimal digits.
S = Space (0,1,2); ignored if NN = 0.
U = Unit Designate (0 or 1)
i = Ignored

IOCW INFORMATION

Operation	IOCW Bits 46 43 42 41	CDL OP Code
BCL External (no trans.)	0 0 0 0	10
BCL from Int. BCL	0 0 1 0	10
BCL from ASCII	1 0 1 0	10
BCL from EBCDIC	0 0 1 1	10
Space or Skip	0 1 0 0	11

RESULT DESCRIPTOR, UNIT ERROR FIELD

To MCP	From Device	Error Type
0181	D80	Bit-Transfer Error
0281	D40	Buffer Parity Error
0481	D20	Print Check/Code Parity
0801	C10	Low Paper
0881	D10	Bus Parity Error
1001	C08	End-of-Page

OPERATION

BCL (OP 10)

Print one line on the line printer. The length of the line is determined by the number of printer columns (120 or 132) or by printing the specified number of words. Spacing or skipping takes place after printing.

When the 6-bit frame size is selected and control word bit 47 is false and control word bit 42 is true, BCL Internal code is converted to BCL code by a translator in the IOM. When the 8-bit frame size is selected and control word bit 42 is true, EBCDIC is converted to BCL code; when control word bits 47 and 42 are true, ASCII is converted to BCL code.

SPACE (OP 11)

Space as specified by CDL word bits 37:2 (bits 31:8 must be zero).

00	No space
01	Single space
1x	Double space

SKIP (OP 11)

Skip as specified by CDL word bits 31:8 (skip channels 01-11).

TEST (OP 99)

Test the status of the unit and return a result descriptor.

ERROR TERMINATION

The end-of-page bit is set if the current descriptor does not specify a skip and there is a punch in channel 12 of the forms loop. The current descriptor is executed.

BUFFERED PRINTER CONTROL NO. 2

A result descriptor is returned when the printer buffer has been loaded. The print check error bit refers to the line of print associated with the prior descriptor.

MAGNETIC TAPE

TAPE SUBSYSTEM

A magnetic tape subsystem can include up to four tape controls and up to sixteen magnetic tape transports. Within a single tape subsystem, all magnetic tape transports must be used at the same speed; and all controls must be of the same type.

TAPE EXCHANGES, FREE-STANDING UNITS

A magnetic tape exchange is required when more than one control is used or more than 6 magnetic tape transports are used.

MAGNETIC TAPE EXCHANGE NO. 2

This magnetic tape exchange provides the facility for either of two compatible magnetic tape controls to communicate with any of ten magnetic tape transports (all 7-track or all 9-track):

An exchange extension adapter is required for each group of two tape units. An adapter/magnetic tape exchange is required for each ready-status cable (1 or 2) that is connected to an IOM.

MAGNETIC TAPE EXCHANGE NO. 1

This magnetic tape exchange provides the facility of communication between any of four NRZ magnetic tape controls and any of 16 free standing magnetic tape transports.

The basic exchange is equipped for one control unit and no tapes.

TAPE EXCHANGE, CLUSTER UNITS

A cluster is not connected to an external exchange. It can include, as an option, 2x adapter(s) enabling it to operate with two controls. A 2X master cluster provides 2 x 4 capability. The addition of a 2X slave provides 2 x 8 capability.

The controls of the same types (number of heads, and NRZ or PE) can operate with one master cluster (and its matching slave).

2 7-track NRZ, or
2 9-track NRZ, only, or
2 9-track PE only.

Use, if any, of a cluster with two types of stations with the matching controls is permitted. The NRZ controls and stations must be treated as a separate sub-system from phase-encoded (P.E.) stations and controls associated with the same cluster (i.e., separate groups of unit designate numbers).

CDL WORD FORMAT

	47	43	F 39	U 35	31	27	23	19	15	11	7	3
O 50	O 46	P 42	O R 38	N 34	MI 30	SC 26	22	18	14	10	6	2
O 49	CO 45	DE 41	M A 37	I 33	29	25	21	17	13	9	5	1
O 48	44	40	T 36	T 32	28	24	20	16	12	8	4	0

41035

FIELD

OP	VAR	Operation
01	iUVi	Rewind
02	DUVC	Read Forward
03	DUVC	Read Backward
04	DUVi	Erase
06	DUVi	Write
08	DUNN	Space Forward
09	DUNN	Space Backward
99	iUii	Test

i = Ignored
U = Unit Designate (LSD)
NN = No. of Records (2 decimal digits; 00 spaces 100 records)
C = CRC Correction, if bit 27 is on (9 track NRZ only); track in error is in [26:3].
V4 = (Read/Write) = Special BCL translation in PC if bit 30 is on (7 track EVEN parity only).
V2,V1 (Read) = Maintenance variants if [29:2] $\neq$ 0 (ignored for PE tape).
V2 (Write) = Write tapemark if bit 29 is on.
V8 (Read) = Do not store information if bit 31 is on.
V1 (Rewind) = Unload tape if bit 28 is on (6A control only).
V8 (Erase) = Backward pseudo-erase if bit 31 is on.
D = Density and Parity

		800	556@	200	1600*
Density	EVEN	0	2	4	6
Parity	ODD	1	3	5	7

@ = 7-Track Only.
* = 9-Track PE Only.
No. = Unit Selected Density.

IOCW INFORMATION

7-Track			IOCW Bits				CDL
Operation	46	44	43	42	41	39	Op Code
Read Binary (6-bit to 6-bit)	0	1	0	0	0	0/1	02/03
Read BCL into Int. BCL	0	1	0	1	0	0/1	02/03
Read BCL into EBCDIC	0	1	0	1	1	0/1	02/03
Read BCL into ASCII	1	1	0	1	0	0/1	02/03
Write Binary (6-bit to 6-bit)	0	0	0	0	0	x	06
Write BCL from Int. BCL	0	0	0	1	0	x	06
Write BCL from EBCDIC	0	0	0	1	1	x	06
Write BCL from ASCII	1	0	0	1	0	x	06
Erase	x	0	1	x	0	0	04

9-Track Operation	IOCW Bits						CDL
	46	44	43	42	41	39	OP CODE
Read Binary (8-bit to 8-bit)	0	1	0	0	1	0/1	02/03
Read EBCDIC into ASCII	1	1	0	0	1	0/1	02/03
Read ASCII into EBCDIC	1	1	0	1	1	0/1	02/03
Write Binary (8-bit to 8-bit)	0	0	0	0	1	x	06
Write EBCDIC from ASCII	1	0	0	0	1	x	06
Write ASCII from EBCDIC	1	0	0	1	1	x	06
Erase	x	0	1	x	1	0	04
Both							
Rewind	x	0	1	x	x	1	01
Space	x	1	1	x	x	0/1	08/09
Write tapemark	x	0	x	x	x	x	06

x = not used

RESULT DESCRIPTOR, UNIT ERROR FIELD

To MCP	From Device	Error Type
0000	800	Normal Termination by PC on Non-Read Operation
	C20	Normal Termination by PC on Read Operation
0801		Long Block may be subsequently generated by IOM from final L1A address comparison with buffer length
0401		Short Block
0081	D00	Memory-Access Error
0101	C80	Beginning-of-Tape or End-of Tape
0109	E80	Not Ready During Operation*
0201	C40	Write Lockout or End-of-File
0481	D20	Peripheral Interface Parity During Data Transfer*
		Memory Access Error on Read Operation**
0489	F20	Peripheral Interface Parity in Initiate Phase*
0881	D10	Bus Parity Error (System Interface Parity during Data Transfer*)
0889	F10	System Interface Parity in Initiate Phase*
OC01	–	Tape Positioning uncertain during Retry***
OC81	D30	Parity Error
OD81	D80	Parity Error and End-of-Tape
0E81	D70	Parity Error and End-of-File
2001	C04	Non-Present Option (Incorrect Density)
4009	E02	Not Ready, Rewinding
8001	C01	Blank Tape Timeout
8101	C81	Blank Tape Timeout and Beginning-of-Tape
Y001 (Y odd)	COZ (Z> 8)	CRC Correction Requested (9 track NRZ only; track in error is in Z4, Z2 and Z1 (LSB); in Y2, Y4 and Y8 (LSB)

*Model 6A Control only
**Model 5A Control only
***Generated by Software

A test op returns the unit density in [11:2] of the RD to the MCP as follows: 0 = 800 BPI, 1 = 556 BPI, 2 = 200 BPI, 3 = 1600 BPI.

OPERATIONS

REWIND (OP 01)

Rewind the designated tape unit. The control is released and a result descriptor returned after rewind is initiated.

READ OP 02 (FORWARD) OR OP 03 (REVERSE)

Read a record from the designated tape unit. The operation is terminated by detection of an interrecord gap. Information transfer is terminated after reading the specified number of words or by sensing an internal DSU error.

ERASE (OP 04)

Erase in the forward direction on the designated tape unit. The operation is terminated by erasing the number of words specified. No memory cycles are used.

WRITE (OP 06)

Write a record on the designated tape unit. The operation is terminated by writing the specified number of words or a delimiter in the data stream.

WRITE TAPE MARK (OP 06)

Write a tape mark record on the unit designated, when V = 2.

SPACE (OP 08 (FORWARD): OP 09 (REVERSE)

Space 1 to 100 records as specified by the BCD value of NN of the CDL word. If bits NN are all 0's, space 100 records.

TEST (OP 99)

Test the status of the designated unit and return a result descriptor.

BCL ALPHA OPERATION (7-TRACK TAPE WITH EVEN PARITY)

When the 6-bit frame size and even parity are selected, BCL internal code is converted to BCL code on write, and BCL code is converted to BLC Internal Code on read. The BCL "?" code is written (001111).

All above operations are performed by the control.

EXCEPTION CONDITIONS

End of tape does not terminate an operation. The end-of-tape bit is set in the result descriptor after the operation is completed.

On read operations, when a vertical parity error is detected and the 6 bit frame size is selected, a BCL "?" code is stored by the control in memory in place of the code in error.

CRC CORRECTION (9-TRACK, 800 BPI ONLY)

CDL bits V enables CRC correction. The 3 LSB's define the track to be corrected (0-7). The parity track cannot be corrected.

PAPER TAPE PUNCH

PAPER TAPE PUNCH CONTROL

The B 9220 Paper Tape Punch is capable of punching a standard paper tape format in either BCL or Baudot code. The punch accommodates 5-, 6-, or 8-channel tape at a maximum rate of 100 characters per second, punching 10 characters to the inch. Standard tape widths of 11/16, 7/8, and 1 inch may be used in either the oiled paper tape, metalized mylar tape, or laminated mylar tape.

Each paper tape I/O control, reader or punch, can accommodate only one paper tape unit. The controls are the small-size controls which can be set into a PCC cabinet as either a right hand or a left hand control.

CDL WORD FORMAT

	47	43	39	35	F 31	27	23	19	15	11	7	3
O 50	O 46	P 42	38	34	O R 30	26	22	18	14	10	6	2
O 49	CO 45	DE 41	37	33	M A 29	25	21	17	13	9	5	1
O 48	44	40	36	32	T 28	24	20	16	12	8	4	0

41037

FIELD

OP	VAR	ADDR	Operation
48	iiVi		Write
49	iiii		Punch Leader
99	iiii		Test

i = Ignored
V = Format:
0 = 7 bits, even parity generated by control
1 = 6 bits, odd parity generated by control
2 = 8 bits, no parity

IOCW INFORMATION

Operation	IOW Bits 46	43	42	41	CDL OP Code
Write Binary (6-bit from 6-bit)	0	0	0	0	48
Write (8-bit from 8-bit)	0	0	0	1	48
Write EBCDIC from ASCII	1	0	0	1	48
Write ASCII from EBCDIC	1	0	1	1	48
Write BCL from Int. BCL	0	0	1	0	48
Write BCL from ASCII	1	0	1	0	48
Write BCL from EBCDIC	0	0	1	1	48
Punch Leader	0	1	0	0	49

RESULT DESCRIPTOR, UNIT ERROR FIELD

To MCP	From Device	Error Type
0101	C80	Low Tape
0401	C20	Short Block

OPERATIONS

WRITE (OP 48)

Punch one record on the paper tape punch. The operation is terminated by punching the specified number of words or by punching a control code. BCL Internal Code, ASCII, or EBCDIC is converted to BCL code by translators in the IOM. BCL code is converted to BCL paper Tape Code by a translator in the punch.

If bits 31:4 in the CDL word contain a zero, the eighth bit of the character is ignored, and an even parity bit is sent on the eighth line from the control to the paper tape punch. Odd parity is punched on the paper tape.

If bits 31:4 in the CDL word contain a one, the EBCDIC/BCL translator is enabled. The control will terminate the operation when the delimiter (1000 0000) is detected. The delimiter is not sent to the punch. The seventh bit received from memory is ignored, a zero bit is forced on the seventh line, and an odd parity bit is sent on the eighth line. The BCL/BCL paper tape code translator in the paper tape punch is enabled to complete the data transfer. Odd parity is punched on the paper tape.

If bits 31:4 in the CDL word contain a two, the control sends all eight bits received from memory to the paper tape punch. Parity is neither generated not sent.

PUNCH LEADER (OP 49)

Punch the specified number of characters. When the 8-bit frame size is selected, all one characters are punched for each character spaced. When the 6-bit frame size is selected, each character spaced is an all zeros character.

TEST (OP 99)

Test the status of the unit and return a result descriptor.

EXCEPTION CONDITIONS

When the number of words specified are not written due to termination by the punch, the incomplete record bit is set by the control in the result descriptor.

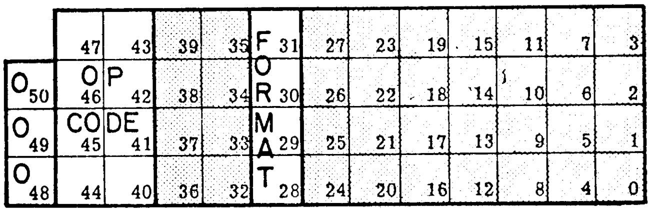
If a memory parity error is encountered during a write operation, the operation is terminated without punching any erroneous characters.

PAPER TAPE READER

PAPER TAPE READER CONTROL

The B 7120 Paper Tape Reader Control is used with the paper tape reader. The B 9120 Paper Tape Reader is capable of reading punched paper tape at a rate of 1000 characters per second and metalized mylar tape or fanfold tape at a rate of 500 characters per second. Baudot and BCL to EBCDIC code translation is automatic. All other codes are read directly into memory and may be translated programmatically. The reader can accommodate 5–, 6–, or 8-channel tape as selected by the operator. Tape widths of 11/16, 7/8, or 1 inch are interchangeable.

CDL WORD FORMAT



	47	43	39	35	F 31	27	23	19	15	11	7	3
O 50	O 46	P 42	38	34	O R 30	26	22	18	14	10	6	2
O 49	CO 45	DE 41	37	33	M A 29	25	21	17	13	9	5	1
O 48	44	40	36	32	T 28	24	20	16	12	8	4	0

FIELD		
OP	**VAR**	**Operation**
40	iiVi	Read
41	iiii	Space Forward
43	iiii	Space Backward
47	iiii	Rewind
99	iiii	Test

i = Ignored
V = format:
0 = 7 bits (odd parity)
1 = BCL/EBCDIC - 6-bit (odd parity)
2 = 8 bits (no parity)

IOCW INFORMATION

	IOCW Bits					CDL
Operation	**46**	**43**	**42**	**41**	**39**	**OP Code**
Read Binary (6-bit into 6-bit)	0	0	0	0	0	40
Read (8-bit into 8-bit)	0	0	0	1	0	40
Read EBCDIC (EBCDIC to ASCII)	1	0	0	1	0	40
Read BCL (Ext. BCL to Int. BCL)	0	0	1	0	0	40
Read BCL (BCL to ASCII)	1	0	1	0	0	40
Read BCL (Ext. BCL to EBCDIC)	0	0	1	1	0	40
Read ASCII (ASCII to EBCDIC)	1	0	1	1	0	40
Space Forward	0	1	0	0	0	41
Space Backward	0	1	0	0	1	43
Rewind	0	1	0	0	1	47

RESULT DESCRIPTOR, UNIT ERROR FIELD

To MCP	From Device	Error Type
0081	D00	Memory-Access Error
0101	C08	Beginning-of-Tape or End-of-Tape
0281	D40	Parity Error
0401	C20	Short Block

OPERATIONS

READ (OP 40)

Read one record from the paper tape reader. The operation is terminated by reading the specified number of words, reading a control code, or encountering end of tape. BCL code is converted to BCL Internal Code, ASCII, or EBCDIC by translators in the IOM. The reader parity-error line is monitored.

NOTE

Control codes are defined by switches on the paper tape reader.

If bits 31:4 in the CDL word contain a zero, only the least significant seven bits of the eight bits received from the reader, together with a high order zero bit are sent to the IOM for possible translation.

If bits 31:4 in the CDL word contain a one, the eight bits received from the reader are transferred to the IOM via the BCL/EBCDIC translator; however, the translator will ignore the two most significant bits. The BCL paper tape code/BCL internal translator in the reader is enabled.

If bits 31:4 in the CDL word contain a two, the eight bits received from the reader are transferred to the IOM.

SPACE FORWARD (OP 41)

Space forward the number of words specified unless stopped by end of tape or a control code.

SPACE BACKWARD (OP 43)

Space backward the number of words specified unless stopped by beginning of tape or a control code.

REWIND (OP 47)

Rewind to beginning of tape.

TEST (OP 99)

Test the status of the unit and return a result descriptor.

EXCEPTION CONDITIONS

When the number of words specified is not read or spaced due to termination by the reader, the incomplete record bit is set by the control in the result descriptor.

Parity errors and memory access errors are reported at the end of an operation.

In the event of a memory access error and the termination of an operation by word count or end of tape, the number of characters spaced is not known. Special-error retry procedures may be required.

SINGLE LINE CONTROL

BURROUGHS TERMINAL COMPUTER MODEL TC500

BURROUGHS INPUT AND DISPLAY TERMINAL MODEL B 9352

Up to 8 of the above units in any combination can be serviced. It is required that the TC500 have the proper "firmware" microprogram to operate in accordance with the communications procedures outlined below. The single line control, figure IV-3-4, performs the following functions:

a. Message heading insertion for messages transmitted to a terminal.

b. Message heading deletion for messages received from the terminal.

c. Generation of vertical and longitudinal parity (even).

d. Checking of vertical and longitudinal parity (even).

e. Code conversion between EBCDIC and ASCII (7-bit).

f. Generation of input request interrupts.

Read and write message formats are defined in figures IV-3-5 and IV-3-6.

CDL WORD FORMAT

	47	43	U 39	35	31	27	23	19	15	11	7	3
O 50	O 46	P 42	N 38	34	30	26	22	18	14	10	6	2
O 49	CO 45	DE 41	I 37	33	29	25	21	17	13	9	5	1
O 48	44	40	T 36	32	28	24	20	16	12	8	4	0

41043

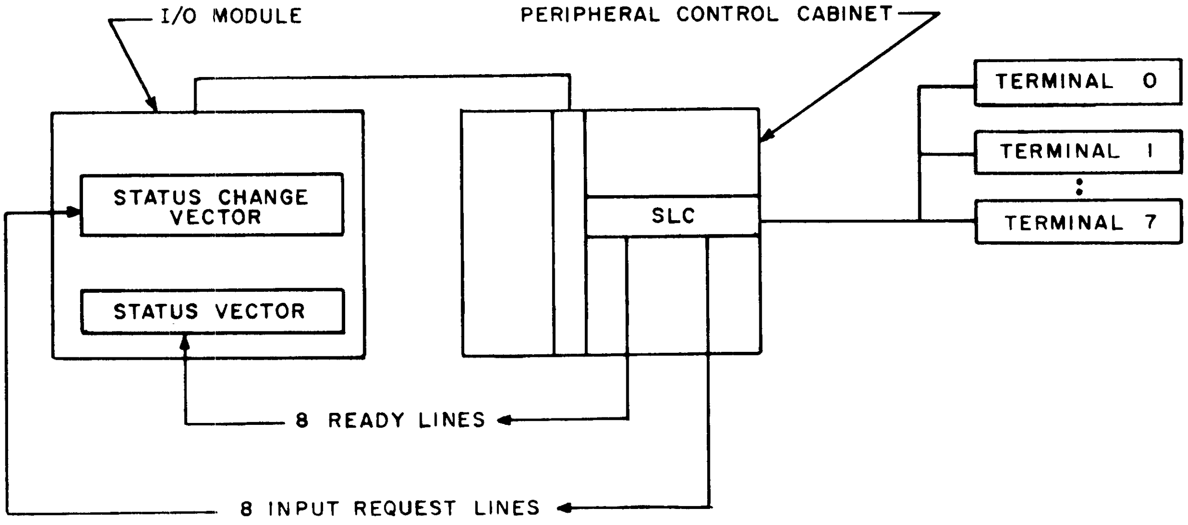
FIELD

OP	VAR	Operation
32	Uiii	Read
34	Uiii	Write
99	Uiii	Test

i = Ignored
U = Unit Designate

IOCW INFORMATION

	IOCW Bits				CDL
Operation	46	44	42	41	OP Code
Read EBCDIC	0	1	0	1	32
Write EBCDIC	0	0	0	1	34



NOTE:
MAXIMUM CABLE LENGTH BETWEEN THE SINGLE LINE CONTROL AND A TERMINAL UNIT IS 50 FEET. THE MESSAGE FORMAT IS A SUBSET OF THE BURROUGHS COMMUNICATION PROCEDURE STANDARD, STANDARD 1284-9006. DATA TRANSMISSION IS ASYNCHRONOUS WITH EACH CHARACTER TRANSMITTED CONSISTING OF 10 BITS AS FOLLOWS: A "0" START BIT, 7 INFORMATION BITS (LEAST SIGNIFICANT BIT FIRST), AN EVEN PARITY AND A "1" STOP BIT.

Figure IV-3-4. Single Line Control Configuration

RESULT DESCRIPTOR, UNIT ERROR FIELD

To MCP	From Device	Error Type
0009	E00	Not Ready to Receive
0081	D00	Memory-Access Error
0281	D40	Parity Error
0401	C20	Control Message
0801	C10	Overflow
0881	D10	Bus Parity Error
8001	C01	Timeout

A hard-wired plug board on the control determines what device type information the test op returns in bits [14:3] of the RD. The MCP uses this information to set the default screen width for displays: 2 = 80 characters (TC500), 1 or 4 = 40 characters (CONRAC) 0 = device not present.

OPERATION

READ (OP 32)

Read an input message from the designated terminal unit until an "End of Text" character is detected or until the area descriptor word count is exhausted, whichever comes first. An ASCII & EBCDIC translator is located in the peripheral control.

The IOM either stores the data as received or translates to ASCII.

NOTE

Each of the 8 terminals which can be connected to a Single Line Control is assigned a system unit designate number. The unit designates must be sequential.

WRITE (OP 34)

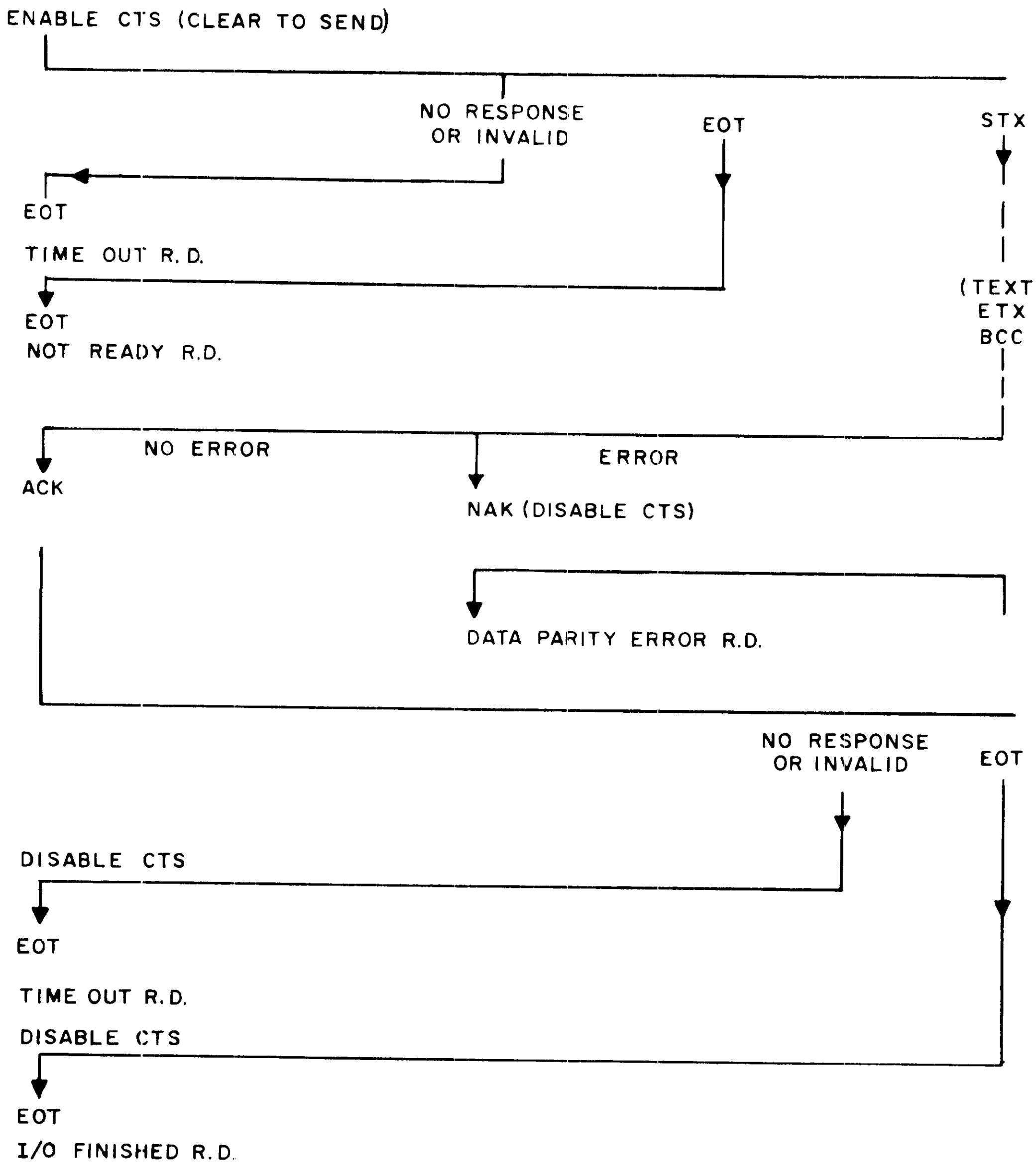
Send a message to the designated terminal unit until an "End of Text" character is detected in the message or until the area descriptor word count is exhausted, whichever comes first. The IOM either transmits data as received, or does an ASCII to EBCDIC translation. An EBCDIC to ASCII translator is enabled in the control.

TEST (OP 99)

Return a result descriptor indicating the type of the designated terminal unit. If no type bit is returned, it means there is no terminal unit connected to the Single Line Control with that unit designation.

SINGLE LINE CONTROL

One adapter is required for each terminal unit connected to a Single Line Control. This adapter is comprised of one level changer card and one cable.



NOTE:

1. STX IS NOT STORED IN THE B7700 MEMORY
2. THE NAK SENT TO THE TC500 OR B 9352 IN THE EVENT OF AN ERROR, CONDITIONS THE TC500 OR B 9352 TO RETRANSMIT THE SAME MESSAGE IN RESPONSE TO THE NEXT READ COMMAND.

Figure IV-3-5. Read Message Format TC500 and B 9352

Data transmission rates of either 1200 or 2400 bits per second can be selected by implementation of the appropriate adapter in the peripheral control. The adapters used for this purpose are Adapter-SLC Timer 2400 and Adapter-SLC Timer 1200.

Either of these pluggable adapters may be installed by a field engineer. The transmission line rate must be the same for all terminals on the same Single Line Control. The rate for the TC500 is limited to 1200 bits per second.

Plug-ons with jumpers in the peripheral control are required to identify the terminal device type associated with each of the 8 possible terminal units. If no device is connected to a particular interface, the corresponding jumper should be removed from the plug-on. See Instructions - Peripheral Control Special No. 1639 7049 for details on jumper locations.

The control receives the receive ready status from the terminals and sends this information to the appropriate status vector in the IOM

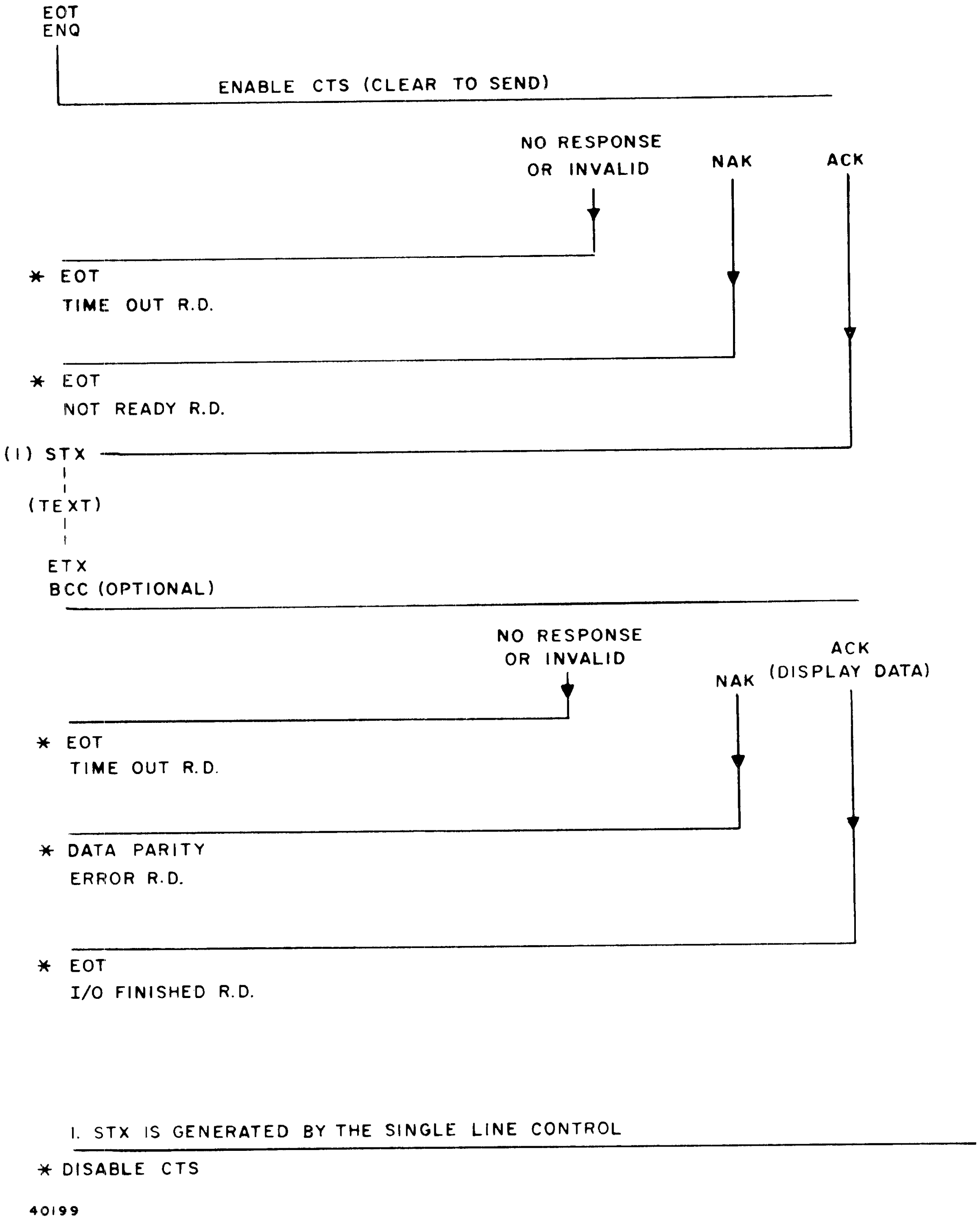


Figure IV-3-6. Write Message Format TC500 and B 9352

via a belted coaxial cable. Input request interrupts initiated by the terminals are sent via individual coaxial cables (1 to 8) to the related IOMS status change vector.

The control contains a one character buffer which receives and sends bit serial information from/to the terminal unit. Transfer of data from and to the IOM is character serial (8 bits).

The control recognizes the "ETX" code (0000011) in an input message which is translated and transferred to the IOM as 0000 0011. Detection of the end of text code terminates the input and stores a result descriptor.

The control recognizes the "ETX" code (0000 0011) in an output message. The end of text code terminates the output and a result descriptor is stored.

If the terminal is in the transmit mode when the control is initiated to do an output, a result descriptor is returned with the not-ready bit set.

There are two delay multi's in the Single Line Control. One multi (DLTO) monitors the line "Turn around" time such that if a terminal does not respond within a time limit, the time out flip-flop (TMOF) is set and a result descriptor generated. If an invalid response is received, the same result occurs without waiting for the multi to time out. The second multi monitors the time elapsed between characters being received from a terminal. If excessive time elapses, the multi (CHTO) times out, setting the time out flip-flop (TMOF) and a result descriptor is generated.

On a read, if the Single Line Control has not gained access to the IOM in time to store a character before the next character is received from the terminal, a memory access error is flagged in the result descriptor, the character is lost, and reading continues. On a write, if the Single Line Control has not gained access to the IOM to fetch another character by the time required to maintain line transmission rates, a memory access error is flagged in the result descriptor, an all zeros (NULL) code is sent to the terminal in place of the character, and writing continues.

CHAPTER V

MEMORY SUBSYSTEM

GENERAL DESCRIPTION OF THE MEMORY SUBSYSTEM

INTRODUCTION

The B 7700 Memory Subsystem provides the main storage for the B 7700 Data Processing System. The memory subsystem stores or supplies words of information as directed by either of two types of requestor: a cnetral processor or an input/output processor.

A B 7700 Memory Subsystem is a modular configuration of one to eight memory modules coupled through a memory requestor switch/interlock network to a maximum of eight memory requestors. (See figure V-1-1.) The memory subsystem can service each requestor in the same manner so that any operation performed for one requestor may also be performed for any other requestor.

A memory module consists of a memory control module (MCM) cabinet which controls either one or two memory storage cabinets (MSC). The MCM controlling one MSC is identified as a 2-MSU memory module. The MCM controlling two MSC's is identified as a 4-MSU memory module. Each MSC contains: two independently addressable 2-1/2 D 2-wire, 1.5-us., core memory storage units (MSU). Each MSU

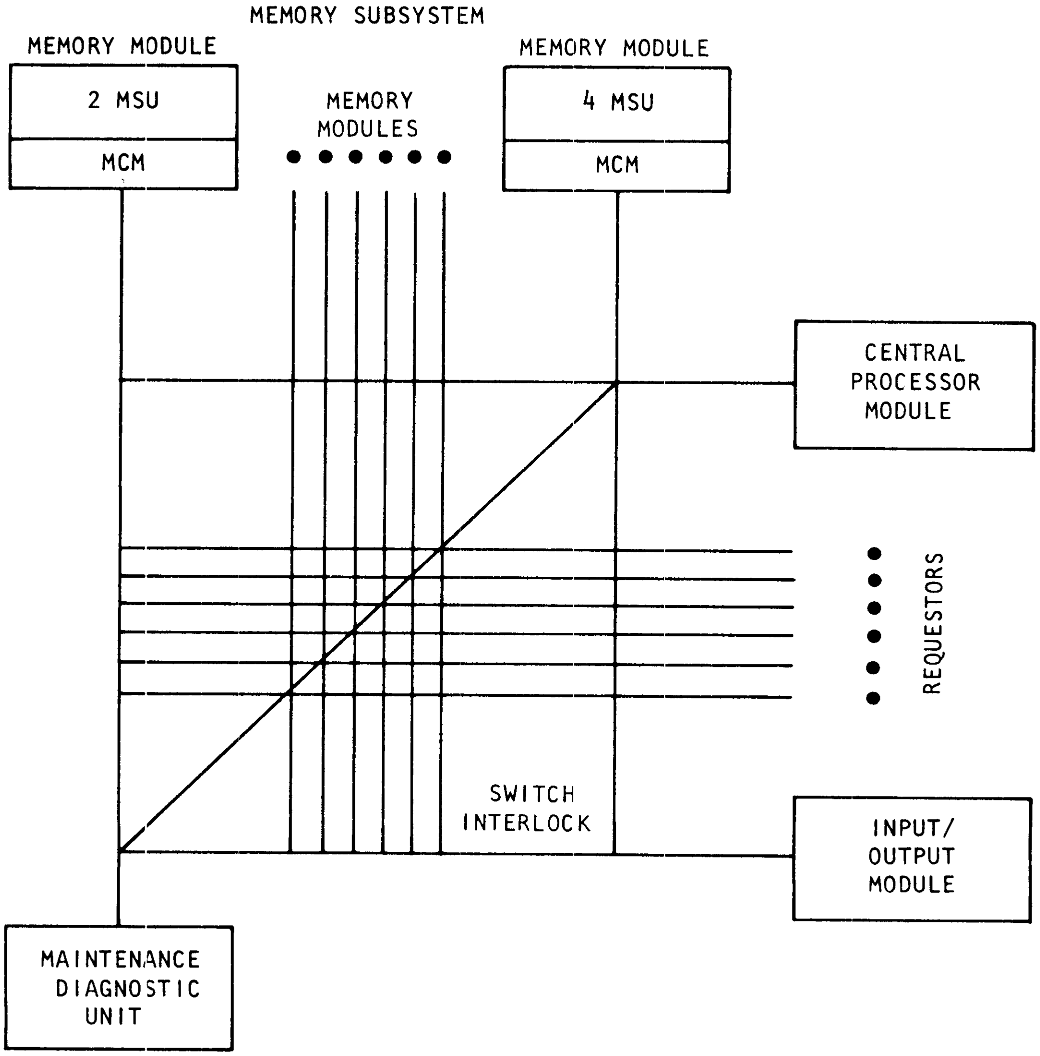


Figure V-1-1. B 7700 Memory System Modularity Diagram

consists of a memory storage module (MSM) with a storage capacity of 65,536 words (393,216 bytes) a memory logic module (MLM) and an independent memory power supply unit (MPU) for each MSU. (See figure V-1-2.)

MEMORY CAPACITY

A B 7700 Memory System may be built with various combinations of the two configurations of memory modules to achieve the desired total memory capacity. Table V-1-1 lists the possible combination of memory sizes.

MINIMUM MEMORY SIZE

The minimum memory size is one 2-MSU memory module of 131,072 words (786,432 bytes). This would be one MCM controlling one MSC containing two MSU's (65,536 words each). For optimum system performance the minimum B 7700 Memory Subsystem recommended is two 4-MSU memory modules.

MAXIMUM MEMORY SIZE

The maximum memory size is 1,048,576 words (6,291,456 bytes) which may be packaged as:

a. Eight 2-MSU modules

b. Four 4-MSU modules

c. A comination of 2-MSU and 4-MSU modules, the total of which equals 1,048,576 words.

MSU RECONFIGURATION

The B 7700 Memory Subsystem is designed with high reliability to minimize the occurrence of failure. Extensive error detection and reporting logic permits early detection and definition of failures. Automatic correction of single-bit parity errors minimizes interruption to the system. The modular design, separate power supplies, and independent interface concept permits soft reconfiguration. In case of an MSU failure, the system can be manually or programmatically reconfigure as follows:

a. The four-MSU memory module will be reconfigured to operate with only two MSU's available to the MCM as the cabinet containing the failed MSU becomes unavailable to the MCM.

b. The two-MSU memory module will be reconfigured to operate with only one MSU available to the MCM.

ADDRESS ALLOCATION

There is no specific assignment order within the system for particular MCM configurations. Memory module address range assignments are based on system requirements and are assigned through use of the memory limits word. For example, any MCM in the system can be assigned the lower (memory zero) address range by setting the memory limits register.

Table V-1-1. B 7700 Memory Module Combinations

Memory Size		Memory Module		Memory Module	
		Quantity		Combination	
Words	Bytes	2-MSU	4-MSU	MCM	MSU
		Type	Type		
131,072	786,432	1	–	1	2
262,144	1,572,864	2	–	2	4
262,144	1,572,864	–	1	1	4
393,216	2,359,296	3	–	3	6
393,216	2,359,296	1	1	2	6
524,288	3,145,728	4	–	4	8
524,288	3,145,728	2	1	3	8
524,288	3,145,728	–	2	2	8
655,360	3,932,160	5	–	5	10
655,360	3,932,160	3	1	4	10
655,360	3,932,160	1	2	3	10
786,432	4,718,592	6	–	6	12
786,432	4,718,592	4	1	5	12
786,432	4,718,592	2	2	4	12
786,432	4,718,592	–	3	3	12
917,504	5,505,024	7	–	7	14
917,504	5,505,024	5	1	6	14
917,504	5,505,024	3	2	5	14
917,504	5,505,024	1	3	4	14
1,048,576	6,291,456	8	–	8	16
1,048,576	6,291,456	6	1	7	16
1,048,576	6,291,456	4	2	6	16
1,048,576	6,291,456	2	3	5	16
1,048,576	6,291,456	–	4	4	16

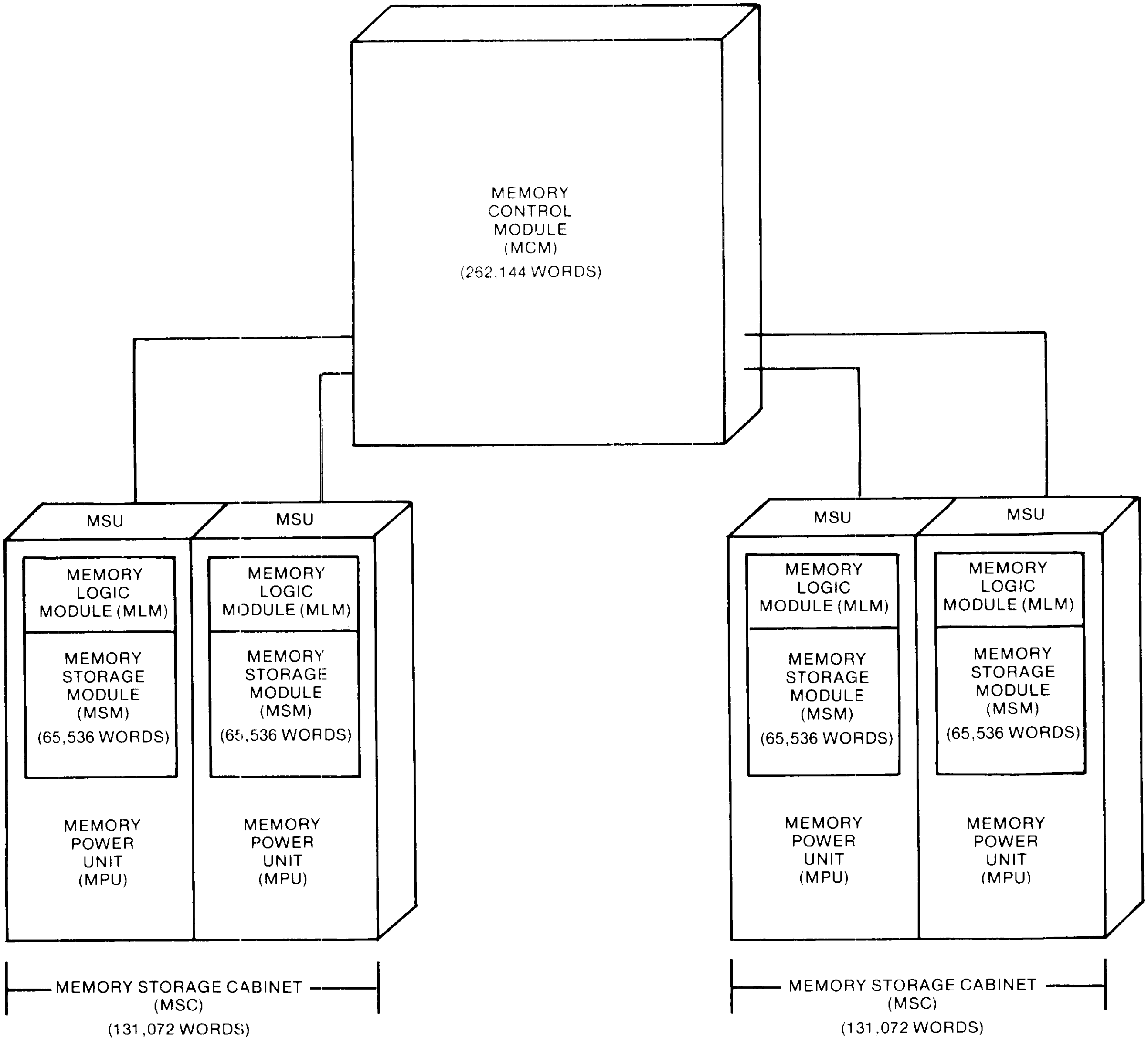

MEMORY
CONTROL
MODULE
(MCM)
(262,144 WORDS)
MSU
MSU
MSU
MSU
MEMORY LOGIC MODULE (MLM)
MEMORY LOGIC MODULE (MLM)
MEMORY LOGIC MODULE (MLM)
MEMORY LOGIC MODULE (MLM)
MEMORY STORAGE MODULE (MSM) (65,536 WORDS)
MEMORY STORAGE MODULE (MSM) (65,536 WORDS)
MEMORY STORAGE MODULE (MSM) (65,536 WORDS)
MEMORY STORAGE MODULE (MSM) (65,536 WORDS)
MEMORY POWER UNIT (MPU)
MEMORY POWER UNIT (MPU)
MEMORY POWER UNIT (MPU)
MEMORY POWER UNIT (MPU)
MEMORY STORAGE CABINET (MSC) (131,072 WORDS)
MEMORY STORAGE CABINET (MSC) (131,072 WORDS)

Figure V-1-2. B 7700 Memory Subsystem

SUBSYSTEM ALLOCATION

The memory capacity can be manually or programmatically allocated into subsystems with respect to designated requestors. For example, MCM's 0 and 2 can be dedicated to requestors 0 and 7 while MCM's 1 and 3 can be dedicated to requestors 1 and 6.

CLOCK RATE AND READ ACCESS TIMES

The B 7700 Memory Subsystem operates at a clock rate of 8.13874 megaHertz. Effective read access time for the MCM is as follows:

a. Single word access is 1.720-us or 0.297-us per byte.

b. Two word access is 1.843-us, or 0.922-us per word, or 0.154-us per byte.

c. Four word access is 2.089-us, or 0.522-us per word, or 0.087-us per byte.

MULTIPLE-WORD TRANSFER (PHASING)

In a multiple-word transfer (known as phasing) words are transferred in bursts of up to four; one word is transferred at each clock cycle.

The maximum number of words which can be phased by the MCM is limited to the number of MSU's being controlled by the MCM. (For example, a four MSU configuration can phase four words, and a two MSU configuration can only phase two words.) If the requested address is less than eight words from the upper address limit of the MCM, the memory operation is limited to a single-word transfer.

These phasing limits do not have to be taken into consideration when a requestor sends a memory request. The requestor simply requests the desired number of words to be transferred, and then decreases the number of words each time a memory transfer is completed. If at the end of the operation the word count is not equal to zero, then another request is made until all the words are transferred.

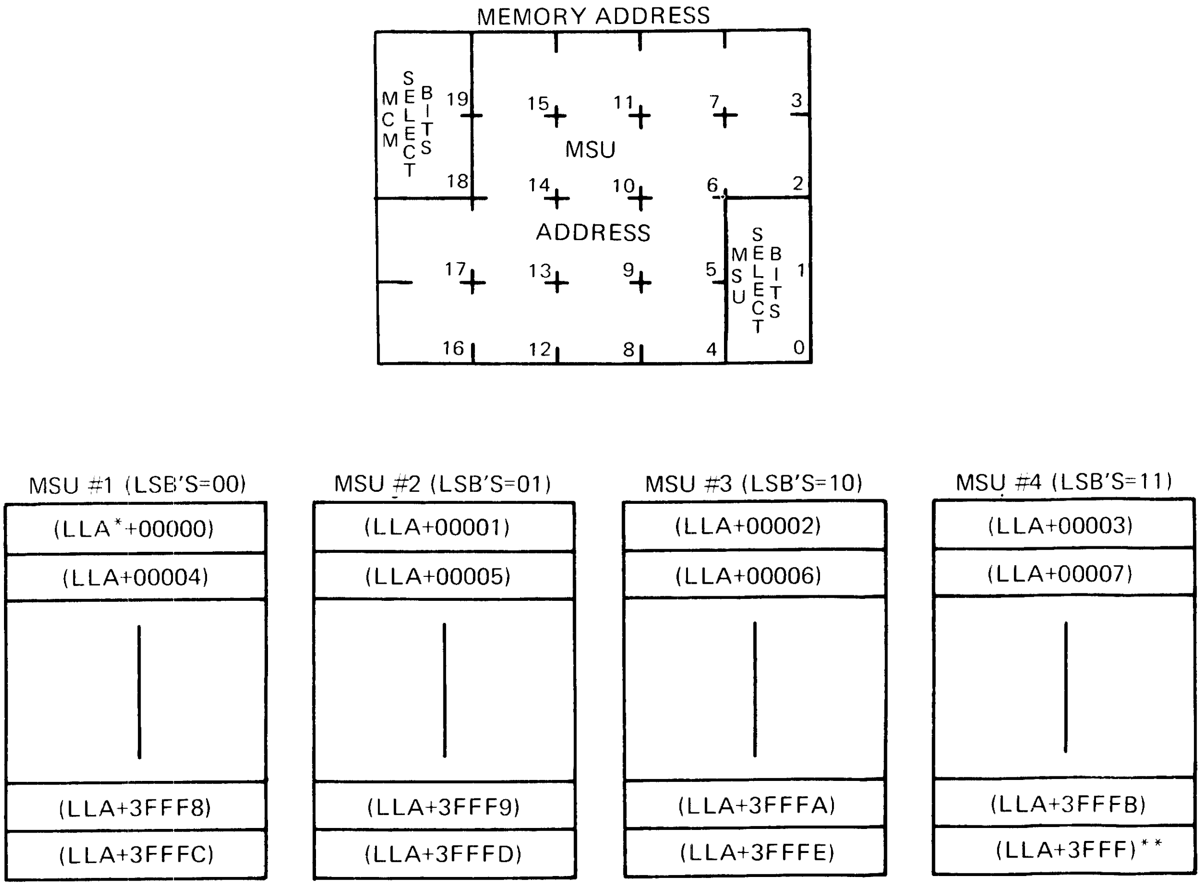
MSU PHASING

A 4-MSU configuration is interfaced so that four consecutive addresses fall in four different memory storage units. (See figure V-1-3.) The effect is that multiple word transfers (phasing) occur in groups of four words each, one at each successive clock. The MCM reads the first word from the MSU defined by the MSU selector bits of the memory address and sends it to the requestor.

As shown in figure V-1-3, the least significant 2 bits (1:2) of the memory address are identified as the MSU select bits. The decoding of these bits is as follows:

2 LSB's	MSU Selection
00	MSU 1
01	MSU 2
10	MSU 3
11	MSU 4

The following 16 bits (17:16) are identified as the MSU Address bits which specify the memory location to be accessed. These bits represent the actual memory address sent to the MSU. The remaining 2 bits (19:2) are identified as the MCM Select bits. These bits determine whether the specified address exists within the MSU's assigned to the MCM.

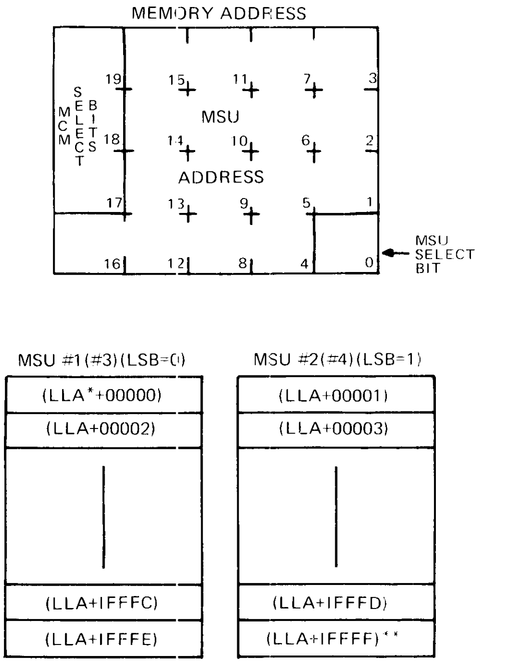


*LLA = LOWER LIMIT ADDRESS

**(LLA+3FFFF) = UPPER LIMIT ADDRESS FOR 4-MSU CONFIGURATION

Figure V-1-3. Four MSU "Phasing" Address Layout

For a 2-MSU configuration (see figure V-1-4), the least significant bit (0:1) of the memory address is identified as the MSU Selector bit. When this bit equals 0, either MSU 1 or MSU 3 is selected; when this bit equals 1, either MSU 2 or MSU 4 is selected. A 2-MSU configuration is interfaced so that 2 consecutive addresses fall in 2 different memory storage units.



MSU #1 (#3) (LSB=0)	MSU #2(#4) (LSB=1)
(LLA*+00000)	(LLA+00001)
(LLA+00002)	(LLA+00003)
(LLA+IFFFC)	(LLA+IFFFD)
(LLA+IFFFE)	(LLA+IFFFF)**

*LLA = LOWER LIMIT ADDRESS

**(LLA+IFFFF) = UPPER LIMIT ADDRESS FOR A 2–MSU CONFIGURATION

Figure V-1-4. Two MSU "Phasing" Address Layout

For a single-MSU configuration (see figure V-1-5), the least significant 16 bits (15:16) represent the actual memory address sent to the MSU. The remaining 4 bits (19:4) are identified as the MCM Selector bits.

MEMORY OPERATIONS TIMING

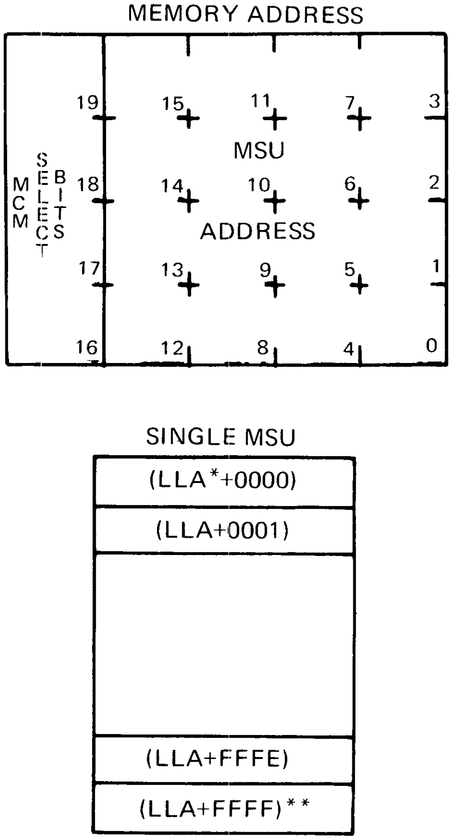
Timing diagram figures V-1-6, V-1-7, and V-1-8 show a general sequence of the major memory operations.

WORD FORMATS

All words used by the B 7700 Mainframe System are 52 bits in length. The 52-bit word consists of 48 bits of information, three tag bits, and an overall parity bit. (See figure V-1-9.)

When information is passed from a requestor to an MCM, the requestor adds a parity bit which produces odd parity on the resultant 52-bit word being transferred. The MCM checks the word it receives for odd parity to verify that an error was not made during transmission.

When an MCM receives a 52-bit word from a requestor, the MCM adds seven bits of error correction code and adds another bit for maintaining odd parity on the overall 60-bit word. The MCM then sends the 60-bit word to the MSU. If a word should be accidentally altered while residing in an MSU, the seven check bits in conjunction with the overall parity bit allows for the detection of the error and provide a means for the automatic correction of errors in which a single bit has been altered. The MCM then sends the original 51-bit word to the requestor.

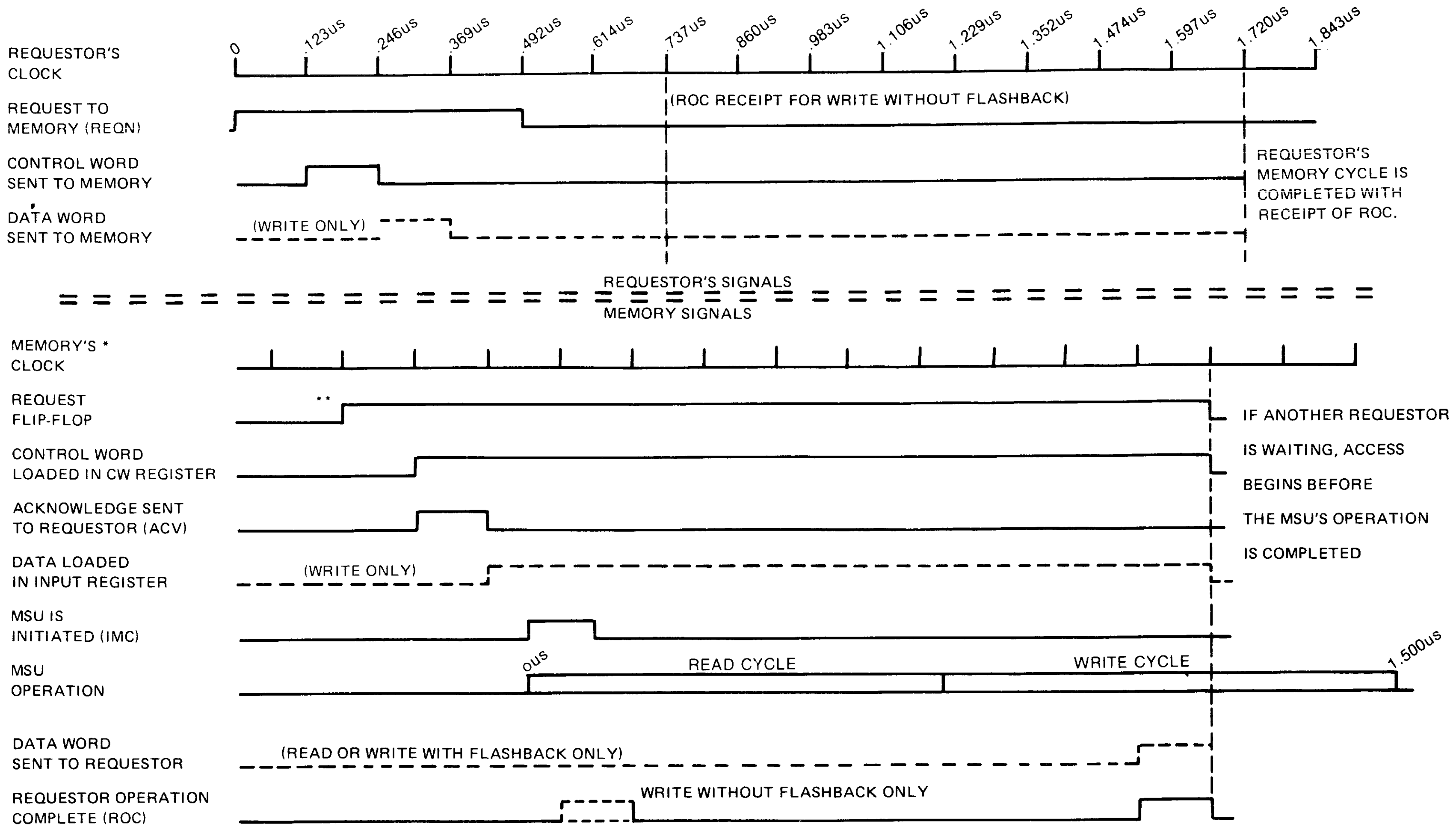


SINGLE MSU
(LLA*+0000)
(LLA+0001)
(LLA+FFFE)
(LLA+FFFF)**

*LLA = LOWER LIMIT ADDRESS

**(LLA+FFFF) = UPPER LIMIT ADDRESS FOR 1 MSU CONFIGURATION

Figure V-1-5. Single MSU Address Layout

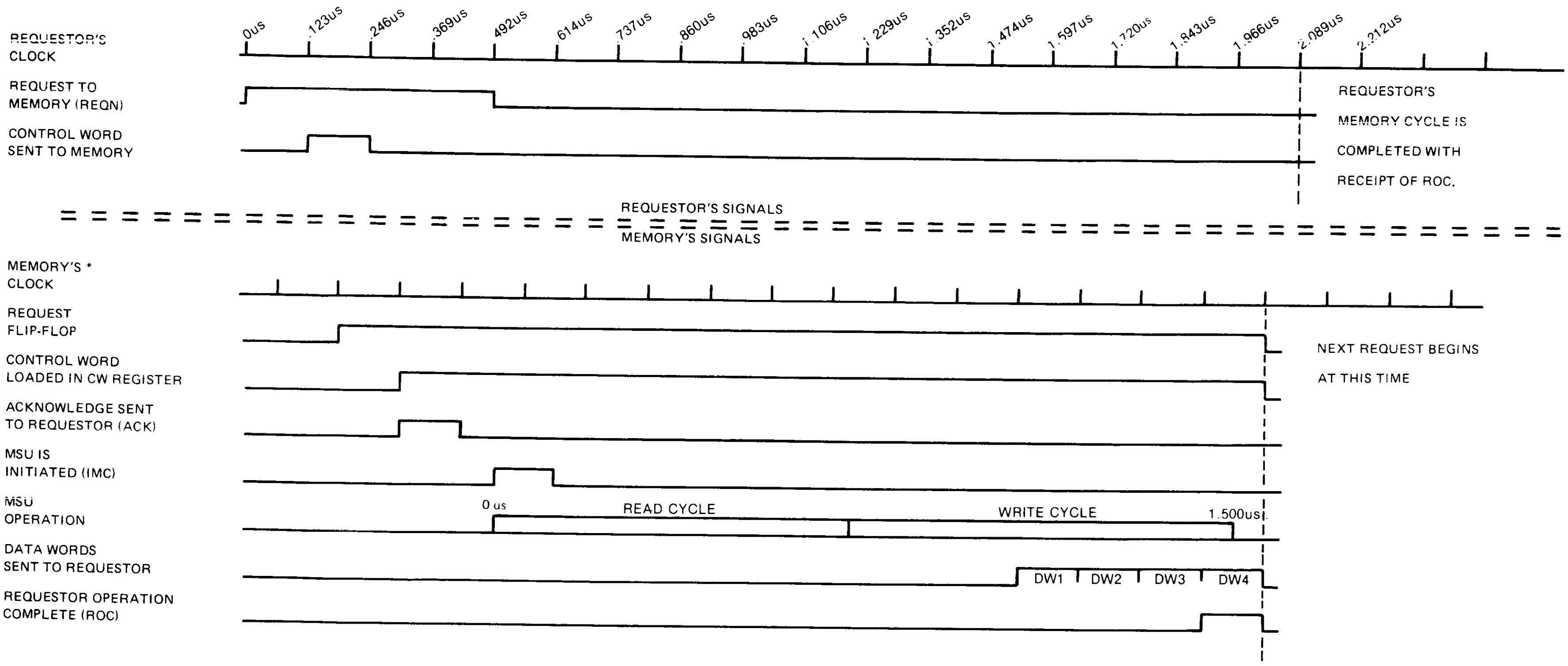

REQUESTOR'S CLOCK
0
.123us
.246us
.369us
.492us
.614us
.737us
.860us
.983us
1.106us
1.229us
1.352us
1.474us
1.597us
1.720us
1.843us
REQUEST TO MEMORY (REQN)
(ROC RECEIPT FOR WRITE WITHOUT FLASHBACK)
CONTROL WORD SENT TO MEMORY
REQUESTOR'S MEMORY CYCLE IS COMPLETED WITH RECEIPT OF ROC.
DATA WORD SENT TO MEMORY
(WRITE ONLY)
REQUESTOR'S SIGNALS
MEMORY SIGNALS
MEMORY'S * CLOCK
REQUEST FLIP-FLOP
**
CONTROL WORD LOADED IN CW REGISTER
ACKNOWLEDGE SENT TO REQUESTOR (ACV)
DATA LOADED IN INPUT REGISTER
(WRITE ONLY)
IF ANOTHER REQUESTOR IS WAITING, ACCESS BEGINS BEFORE THE MSU'S OPERATION IS COMPLETED
MSU IS INITIATED (IMC)
MSU OPERATION
0us
READ CYCLE
WRITE CYCLE
1.500us
DATA WORD SENT TO REQUESTOR
(READ OR WRITE WITH FLASHBACK ONLY)
REQUESTOR OPERATION COMPLETE (ROC)
WRITE WITHOUT FLASHBACK ONLY

NOTES

*MEMORY'S CLOCK IS OUT OF PHASE WITH REQUESTOR'S CLOCK

**THERE IS A .0675 USEC CABLE DELAY BETWEEN MCM'S AND REQUESTOR'S INTERFACE

Figure V-1-6. Single Word Read or Write Operation, Timing Diagram

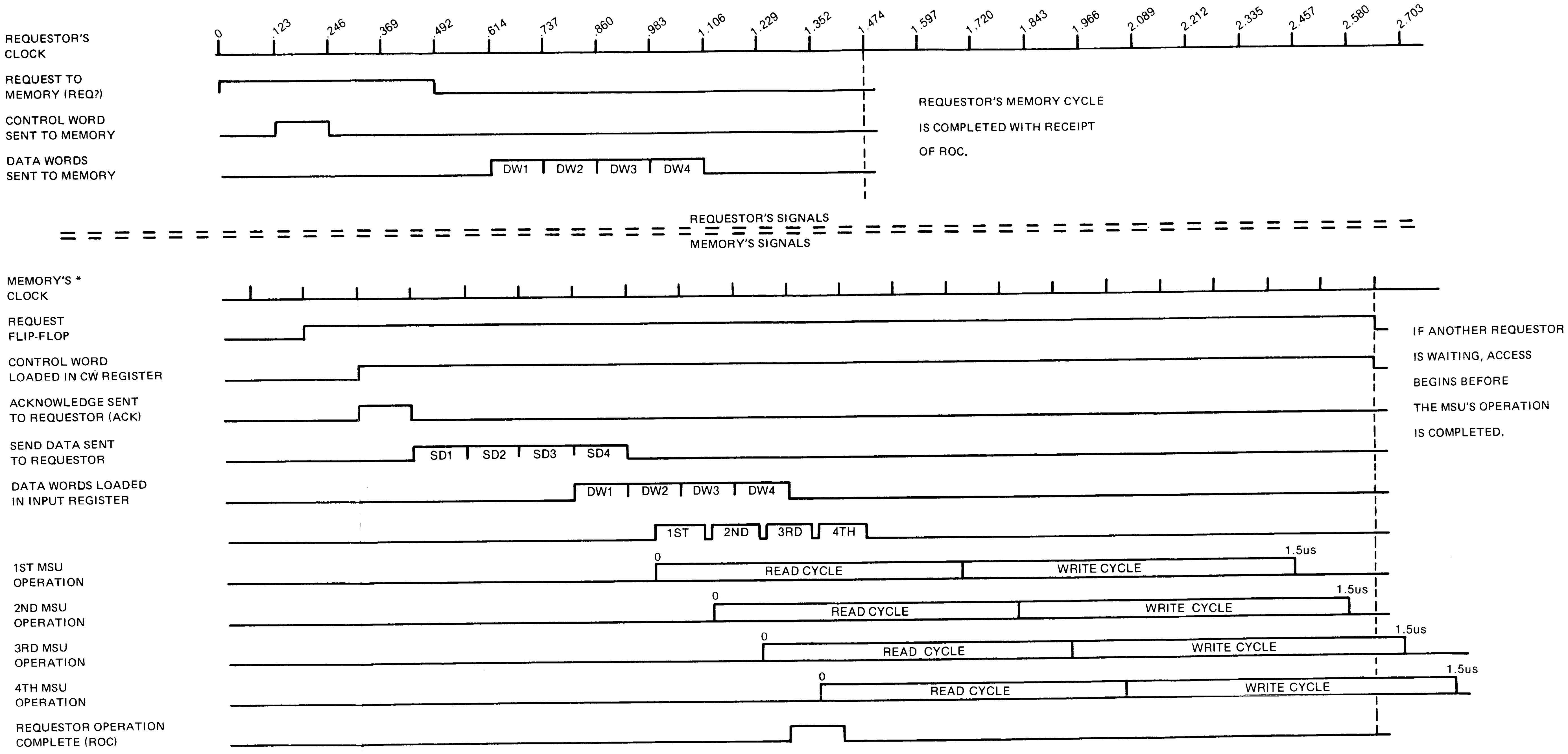


NOTES

*MEMORY'S CLOCK IS OUT OF PHASE WITH REQUESTOR'S CLOCK

**THERE IS A .0675 USEC CABLE DELAY BETWEEN MCM'S AND REQUESTOR'S INTERFACE

Figure V-1-7. Four Word Read Operation, Timing Diagram



NOTES

*MEMORY'S CLOCK IS OUT OF PHASE WITH REQUESTOR'S CLOCK

**THERE IS A .0675 USEC CABLE DELAY BETWEEN MCM'S AND REQUESTOR'S INTERFACE

Figure V-1-8. Four-Word Write Operation, Timing Diagram

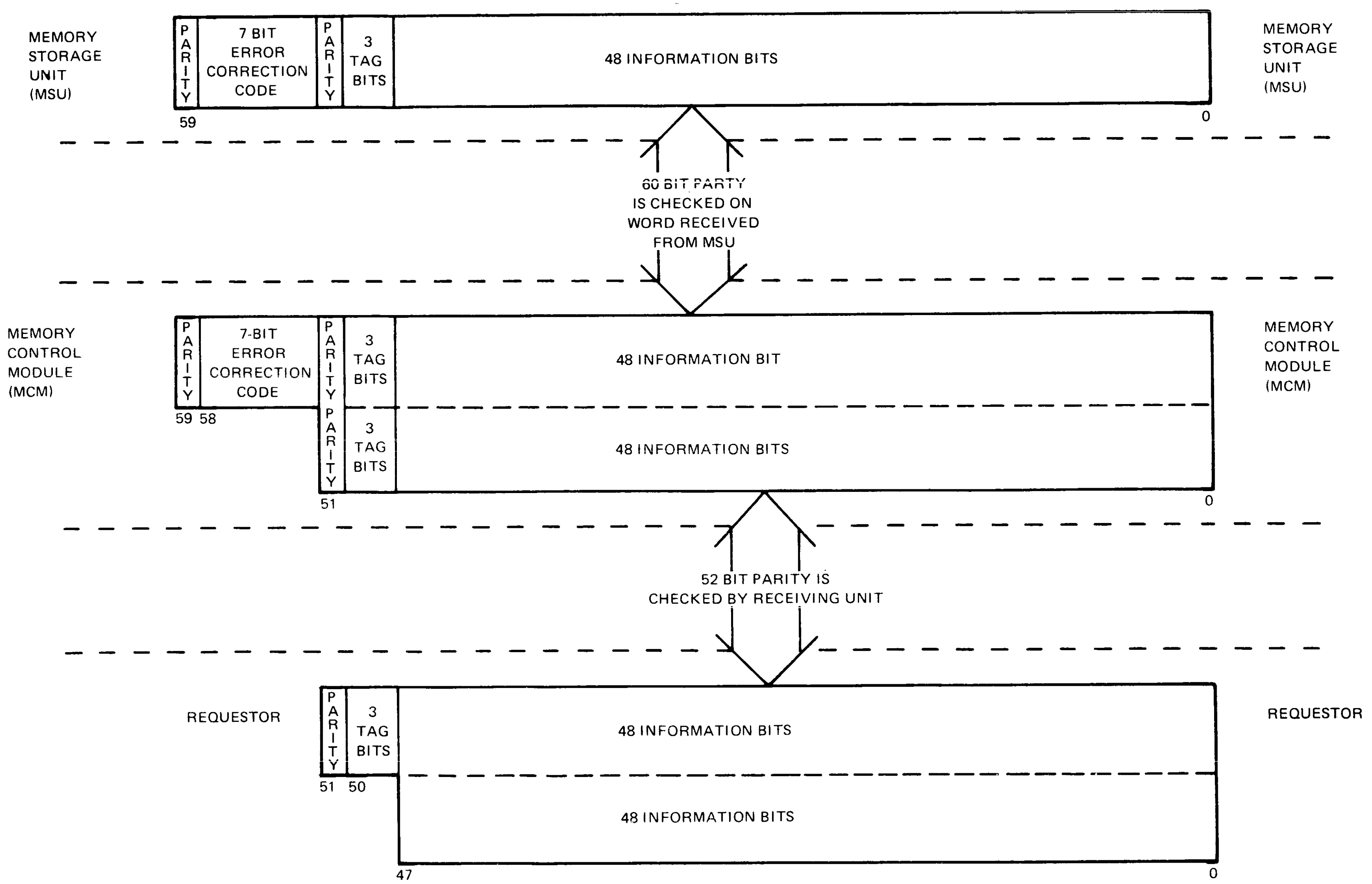
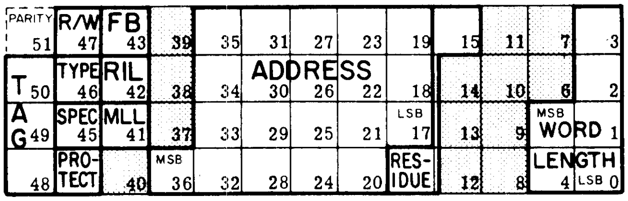

MEMORY STORAGE UNIT (MSU)
PARITY
7 BIT ERROR CORRECTION CODE
PARITY
3 TAG BITS
48 INFORMATION BITS
59
0
MEMORY STORAGE UNIT (MSU)
60 BIT PARTY IS CHECKED ON WORD RECEIVED FROM MSU
MEMORY CONTROL MODULE (MCM)
PARITY
7-BIT ERROR CORRECTION CODE
PARITY
3 TAG BITS
48 INFORMATION BIT
PARITY
3 TAG BITS
48 INFORMATION BITS
59 58
51
0
MEMORY CONTROL MODULE (MCM)
52 BIT PARITY IS CHECKED BY RECEIVING UNIT
REQUESTOR
PARITY
3 TAG BITS
48 INFORMATION BITS
48 INFORMATION BITS
51 50
47
0
REQUESTOR

Figure V-1-9. Data Word Transfer Between Requestor and Memory

MCM CONTROL WORD

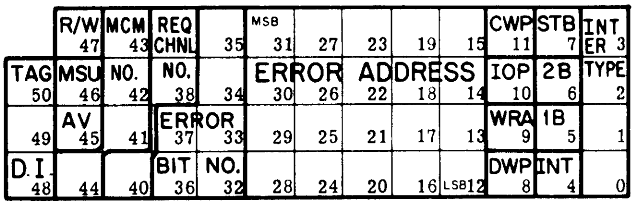
At the start of every memory operation, an MCM control word is transmitted from the requestor to the memory control module. The control word format, bits and fields as received at the memory control module are described below. Table V-1-3 lists the valid operation codes for the MCM.



Field	Bits	Description
PARITY	51:1	The requestor generates odd parity for the 52-bit control word.
TAG	50:3	The Tag bits are not used in the control word.
R/W	47:1	R/W bit: 0 = read (fetch) operation; 1 = write operation.
TYPE	46:1	The Type bit is only set for fail word fetch and N-word protected write operations.
SPEC	45:1	The SPEC (specifier) bit indicates either a single word or multi-word operation: 0 = multi-word operation; 1 = single word operation.
PROTECT	44:1	The protect bit, when "1" indicates that a protected-write operation is to be performed. Protected write only allows the write operation to be performed when bit 48 of the original memory word is off. When bit 48 is on, the write operation is terminated and the contents of the original memory word are not changed.
FB	43:1	The FB (flashback) bit, when a "1" indicates that the original contents of the memory location are to be transferred to the requestor.
RIL	42:2	The RIL (requestor inhibits load) bit is used to specify that a load requestor-inhibit operation is to be performed.
MLL	41:1	The MLL (memory limits load) bit is used to specify that the upper and lower address registers and the MSU available register are to be loaded.
	40:4	Unused.
ADDRESS	36:20	The address bits specify the starting-memory address of the memory operation.
RESIDUE	16:2	The address residue bits indicate the residue value of the 20-bit memory address within the control word.
WORD LENGTH	5:6	The word-length bits indicate the number of words to be transferred during multi-word operations.

MCM FAIL WORD

The MCM fail word contains all pertinent information necessary to identify a hardware failure. The fail word information is locked in a 52-bit fail register until a fetch-the-fail register operation request is made by the requestor or a manual clear operation is performed. The MCM sends a fail 1 interrupt signal to the requestor when an irrecoverable error has occurred or a fail 2 interrupt signal when a one-bit error has occurred. If one of the bits in a memory word was incorrect, the specific bit is corrected by the MCM. The correct memory word is then sent to the requestor together with a fail 2 interrupt signal which allows the requestor to record the error and also to continue processing with correct data. The format, bits, and fields of the MCM fail word are described below:



Field	Bits	Description
D.I	48:1	The delayed interrupt (D.I) bit, when set, indicates that an internal error was detected during the previous memory cycle and had occurred after the requestor operation complete (ROC) signal was sent to the requestor. This interrupt does not occur until the next memory operation is performed.
R/W	47:1	The R/W bit indicates that either a read or write operation was being executed when the error was detected. (Read = 0; write = 1).
MSU AV	46:2	The MSU AV (MSU available) field indicates the number of MSU's in use by the MCM when the error was detected. The field indication is as follows:

Bit 46	Bit 45	
0	0	No MSU is available
0	1	One MSU is available
1	0	Two MSU's are available
1	1	Four MSU's are available

MCM NO.	44:4	The MCM number is preassigned hardware number (from 0 thru 15) that is given to each MCM to identify the specific MCM with the error condition.

REQ CHNL NO. 40:3 The REQ CHNL NO. (requestor-channel-number field) contains the number of the requestor who was using the MCM when the failure occurred. This field is not locked in the fail register if detection of a one-bit error occurs.

ERROR BIT NO. 37:6 The error bit number field is only valid when bit 5 (1-bit error) is set. This field is the binary number of the bit that failed in memory.

ER ADDRS 31:20 The error-address field contains the address of the location that was being accessed if a one-bit or two-bit error occurred. The address is related to one-bit or two-bit errors as follows:

Error Indication		Error Address
2-Bit	1-Bit	Belongs to:
0	1	1-Bit Error
1	0	2-Bit Error
1	1	1-Bit Error

CWP* 11:1 The CWP (control word parity) bit when set, indicates that the MCM has detected incorrect parity on the control word received from the requestor.

IOP* 10:1 The IOP (illegal operation) bit, when set, indicates that the MCM has detected an illegal operation in the control word. These illegal operations are as follows:

(1) Word length = 0

(2) Single-word operation word length greater than 1

(3) Special-request strobe is not sent by the requestor when either memory-limits load or requestor-inhibit load is to be performed.

(4) Illegal-operation code (refer to table V-1-3).

WRA* 9:1 The WRA (wrong address) bit, when set, indicates that the address in the control word did not fall within the upper and lower address limits assigned to the MCM.

DWP* 8:1 The DWP (data-word parity) bit, when set, indicates that a data word containing even parity was received from the requestor.

STB* 7:1 The STB (data-strobe) error bit when set, indicates that the MCM has detected an error in the number of words sent by a requestor during a multiple word transfer.

Table V-1-3. Valid Operation Codes for the MCM

	R/W	TYPE	SPEC	PROT	FB	FIL	MLL
Operation	47	46	45	44	43	42	41
Fetch, Single word	0	0	1	0	0	0	0
Fetch, Multiple word	0	0	0	0	0	0	0
Fetch, Fail register	0	1	1	0	0	0	0
Write, Single word overwrite	1	0	1	0	0	0	0
Write, Multiple word overwrite	1	0	0	0	0	0	0
Write, Single word overwrite with flashback	1	0	1	0	1	0	0
Write, Single word protected write	1	0	1	1	0	0	0
Write, Single word protected write flashback	1	0	1	1	1	0	0
Write, Multiple word protected write	1	1	0	1	0	0	0
Load, Requestor inhibits	1	0	1	0	0	1	0
Load, Memory limits	1	0	1	0	0	0	1

2B*	6:1	A 2B (2-bit) error, when set, indicates that the MCM detected a (non-correctable) multiple bit error from the MSU. If this error occurs, the data transfer to the requestor is completed.
1B**	5:1	A 1B (1-bit) error, when set, indicates that the MCM detected a (correctable) 1 bit error from the MSU.
INT*	4:1	The INT (internal) error bit, when set, indicates an error occurred within the MCM or MSU. This error is further defined by bits 3:4.
INT ERR TYPE	3:4	The internal error type bits define the type of internal error identified by bit 4. The internal error bits are defined as follows:

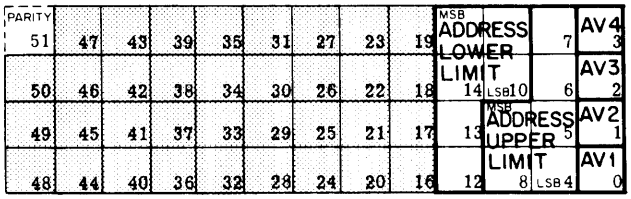
Fail Word Bit **3210**	**Error Type**
0000	MSU Available - indicates that MSU power was interrupted.
0001	Read Available - indicates that the MSU has failed to respond with a Read Available signal during a read operation.
0010	Checker/Generator - indicates that an error had occurred in the MCM Parity Checker/ Generator (data) circuits.
0011	Address Residue - indicates that either a bad address residue was detected in the Control Word received from a requestor, or was generated by the MCM Address Counter during a multiple word transfer operation.
0100	MSU Parity Error - indicates that the MSU detected a parity-error in the address and controls received from the MCM.
0101	Parity Generator (address and control) Error - indicates that an error occurred in the MCM Parity Generator circuits.
0110	Data timer Failure - indicates that a failure occurred in the Data Timer 1-4 circuits.
0111	Data Transfer Control (DTC) Failure - indicates that a failure occurred in the DTC circuit.
1000	MSU Availability Failure - indicates that an illegal MSU Availability configuration exists. The legal configurations of MSU Availability are as follows:

4321	
0001 0010 0100 1000	Single-MSU Configuration
0011 1100	Two-MSU Configuration
1111	Four-MSU Configuration

* Fail 1 interrupt condition
**Fail 2 interrupt condition

MEMORY ADDRESS LIMITS WORD

The Memory Address Limits word changes the MCM and MSU configuration to reflect the number of MSU's available to the MCM as well as the upper and lower address limits. This word follows the special request signal and the memory address limits load word during memory control operations between the MCM and requestor. The format, bits, and fields of the Memory Address Limits are described below:



Field	**Bits**	**Description**
PARITY	51:1	The MCM examines the memory address limits word for odd parity.
	50:35	Unused
ALL	15:6	The address lower limit is the most significant 6 bits of the lowest 20-bit memory address available to this MCM.

AUL	9:6	The address upper limit is the most significant 6 bits of the highest 20-bit memory address available to this MCM.
AV4	3:1	When AV4 is a "1", the MSU designated MSU4 is available to this MCM.
AV3	2:1	When AV3 is a "1", the MSU designated MSU3 is available to this MCM.
AV2	1:1	When AV2 is a "1", the MSU designated MSU2 is available to this MCM.
AV1	0:1	When AV1 is a "1", the MSU designated MSU1 is available to this MCM.

MEMORY REQUESTOR INHIBITS WORD

The Memory Requestor Inhibits word loads the requestor inhibit register with new data to indicate which requestors now have access to the MCM. This word follows the special request signal and the memory requestor inhibits load control word during memory control operations between the MCM and requestor. The format, bits and fields of the Memory Requestor Inhibits word are described below:

PARITY 51	47	43	39	35	31	27	23	19	15	11	RI7 7	RI3 3
50	46	42	38	34	30	26	22	18	14	10	RI6 6	RI2 2
49	45	41	37	33	29	25	21	17	13	9	RI5 5	RI1 1
48	44	40	36	32	28	24	20	16	12	8	RI4 4	RIO 0

Field	Bits	Description
PARITY	51:1	The MCM examines the requestor inhibit word for odd parity.
	50:44	Unused
R17	7:1	When bit R17 is a "1", the requestor who is designated requestor 7 is inhibited from access to the MCM.
R16	6:1	When bit R16 is a "1", the requestor who is designated requestor 6 is inhibited from access to the MCM.
R15	5:1	When bit R15 is a "1", the requestor who is designated requestor 5 is inhibited from access to the MCM.
R14	4:1	When bit R14 is a "1", the requestor who is designated requestor 4 is inhibited from access to the MCM.
R13	3:1	When bit R13 is a "1", the requestor who is designated requestor 3 is inhibited from access to the MCM.
R12	2:1	When bit R12 is a "1", the requestor who is designated requestor 2 is inhibited from access to the MCM.
R11	1:1	When bit R11 is a "1", the requestor who is designated requestor 1 is inhibited from access to the MCM.
R10	0:1	When bit R10 is a "1", the requestor who is designated requestor 0 is inhibited from access to the MCM.

SIGNAL INTERFACE BETWEEN REQUESTOR, MCM, AND MSU

The control and information flow between the requestor, MCM, and MSU is described in the following paragraphs and shown in figure V-1-10.

SIGNAL INTERFACE BETWEEN MCM AND REQUESTOR

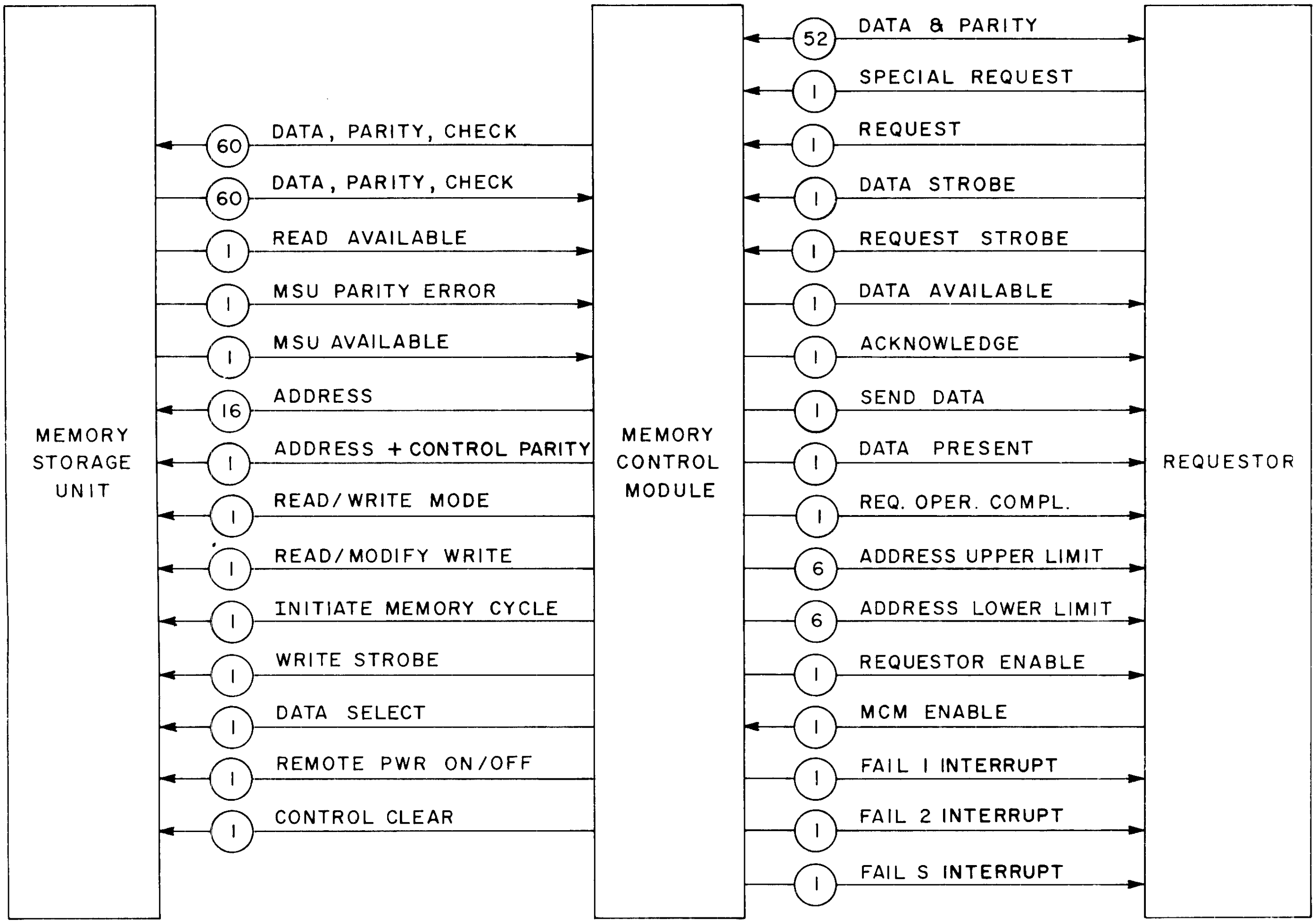
1. Data and, Parity. Data and parity are transferred between a requestor and an MCM via a unique set of 52-bidirectional data lines. These lines are also used for the transmission of the control word.

2. Special-Request Signal (RQSN). A special-request signal (RQSN) is used by a CPM to gain access to a memory control module (regardless of the state of the requestor-inhibits register). The RQSN signal goes "true" in coincidence with the request signal (REQ) whenever a Memory Address Limit Load or Requestor Inhibits Load operation is performed.

3. Request Signal (REQN). A request signal (REQN) is sent by a requestor to select a specific MCM. REQ goes "true" one clock period prior to the request strobe (RSTB) and remains "true" until the receipt of an acknowledge signal (ACK) from the MCM.

4. Data-Strobe Signal (DSTB). A data-strobe signal (DSTB) is sent to inform the MCM that data is to be transmitted over the data lines. The signal is used only in the N-length overwrite and the N-word protected write operations. The data strobe precedes the data word by one clock and its width indicates the number of data words following it.

5. Request-Strobe Signal (RSTB). A request-strobe signal (RSTB) is sent to inform the MCM that a control word is being transferred over the data lines. It is "true" initially one clock period following the start of the request



40265

Figure V-1-10. Requestor-MCU-MSU Interface

signal (REQ). The control word is transmitted in coincidence with the request strobe.

a. Single-word protected write and single-word overwrite: The request strobe (RSTB) will cycle "true" and "false" during successive clock periods. During the "false" period, the data word to be stored is placed on the data lines.

All other operations: The request strobe (RSTB) is "true" one clock period following the request signal (REQN) and remains "true" until the acknowledge signal (ACK) is received.

6. Data-Available Signal (DAV). A data-available signal (DAV) is transmitted to the requestor to indicate that data is available and will be transmitted in the following clock period.

7. Acknowledge Signal (ACK). An acknowledge signal (ACK) of one clock period duration is sent to the requestor to signify that the MCM has accepted the control word and is processing the request.

8. Send-Data Signal (SND). A send-data signal (SND) is sent to the requestor during an N-length overwrite and may be sent during an N-word protected write. The send-data signal indicates the number of data words that must be transmitted to the MCM. The number of words to be transmitted is equal to the number of clock periods the send-data signal is "true."

NOTE

The send data signal will not be transmitted if an attempt is made to write into a protected area during an N-word protected write. Also, the number of data words requested by the MCM must be transferred before a requestor ends his operation.

9. Data-Present Signal (DAPB). Signal DAPB is sent to the requestor to indicate that a valid data word (or words) is being trans-

mitted from the MCM. The DAPB is transmitted in coincidence with the data word. A word is transmitted each clock period that the DAPB is "true."

10. Requestor-Operation Complete Signal (RQOC). The MCM sends a one-clock-period requestor-operation-complete signal (RQOC) to signify the end of the requestor's part of the memory operation.

The following variations apply:

a. Single or N-length fetches; single-word overwrite with flashback: The RQOC is sent coincident with the final clock period of the data-present signal (DAPB).
b. Single-word overwrite; N-length overwrite or N-word protected write: The RQOC signal is sent following the check of parity on the final data word received by the MCM.
c. Single or N-word protected write: The RQOC signal is sent with or following FALS signal if word(s) are protected in N-word protected write.
d. Single word overwrite without flashback: An RQOC is generated following the check of parity on the data word received by the MCM.

11. Address Upper Limit. The Address Upper Limit is the most significant six bits of the highest 20-bit memory access available to this MCM (the least significant 14 bits are assumed to be "1's").

12. Address Lower Limit. The Address Lower Limit is the most significant six bits of the lowest 20-bit memory address available to this MCM (the least significant 14 bits are assumed to be 0's).

13. Requestor-Enable Signal. The MCM sends to the requestor an enable signal which is used to enable or disable communications between the MCM and the requestor.

1. Whenever the MCM is power cycling up or down.
2. Whenever the appropriate requestor inhibit FF is set.

14. MCM-Enable Signal. The requestor sends to the MCM an enable signal which is used to enable or disable communications between the requestor and the MCM. This signal is a steady-state signal which disables communications whenever the requestor is power cycling up or down.

15. Failure Interrupt 1 Signal (FAL1). The MCM transmits a one-clock period FAIL 1 interrupt signal to the requestor if any of the following errors occur:

1. Control word parity
2. Illegal operation code
3. Wrong MCM
4. Data strobe error
5. 2-bit error
6. Internal error

The MCM Fail Register will then be loaded with information to facilitate error analysis.

16. Failure Interrupt 2 Signal (FAL2). The MCM transmits a one-clock period FAIL 2 interrupt to the requestor if a 1-bit error occurs. The MCM Fail Register will then be loaded with information to facilitate error analysis.

17. Software Interrupt (FALS). A one-clock-period software error interrupt is transmitted to the requestor during a single-word or N-word protected write operation if the memory word being examined contains a "1" in bit 48.

SIGNAL INTERFACE BETWEEN MCM AND MSU

1. Data Input Lines (60). Data, error correction code, and overall parity are transferred from MCM to MSU via a set of 60 data lines.

2. Data Output Lines (60). Data, error correction code, and overall parity are transferred from MSU to MCM via a set of 60 data lines.

3. Read Available Signal. The read available signal is sent to inform the MCM that read data is available.

4. MSU Parity Error Signal. The MSU parity error signal is sent by the MSU to indicate that an address and control parity error occurred.

5. MSU Available Signal. The MSU available signal is sent to the MCM to indicate that MSU power is on.

6. Address Lines (16). The MCM sends to the MSU a 16-bit address which specifies the memory location to be accessed.

7. Address And Control Parity. The MCM sends to the MSU an odd parity bit for the 16-address signals, the read/write mode signal, and the read/modify/write signal.

8. Read/Write Mode Signal. The read/write mode signal, when true, indicates that a write operation is to be performed by the MSU.

9. Read/Modify/Write Signal. The read/modify/write signal, when true, in conjunction with the read/write mode signal indicates that a read/modify/write (split cycle) operation is to be performed by the MSU.

10. Initiate Memory Cycle Signal. The initiate memory cycle signal is sent by the MCM to start the MSU operation and to strobe the address and control information into the MSU.

11. Write Strobe Signal. The write strobe signal is used to strobe write data from the MCM into the write register of the MSU.

12. Data Select Write Signal. The data select write signal is sent by the MCM during a read/modify/write operation. This signal is used to specify that data in memory is either restored from the read register or overwritten with new information from the write register.

13. Remote Power On/Off Signal. The remote power an/off signal allows power sequencing of the MSU to be controlled by the MCM.

14. Control Clear Signal. The control clear signal is sent by the MCM to clear various control flip-flops and registers in the MSU.

DEFINITION OF MCM OPERATIONS

The various MCM operations are briefly described in the following paragraphs.

1. Data-Word Fetch (Single or Multiple Word). This operation is a standard fetch of data. If a multiple word fetch is initiated, the data words are transferred to the requestor at the clock rate, i.e., 123 nanoseconds, and within the limits discussed previously.

2. Fail-Word Fetch. This operation is a fetch of the fail register within the MCM. The fail register is cleared as a result of this operation.

3. Single-Word Overwrite with Flashback. This operation is a standard write/read operation. The data from the requestor is written into the addressed location. The original data read out of the address location is transferred back - or flashed back - to the requestor.

4. Single-Word Protected Write (with/without flashback). This operation is a conditional write of data into memory. The data word transferred by the requestor is written into memory only if the address is not protected (i.e., bit 48 of the original word is "0"). The requestor may indicate whether he requires flashback; however, the MCM will unconditionally flash back data to the requestor. The MCM will send a Fail S signal to the requestor if the address was protected.

5. Overwrite (Single or Multiple Word. This operation is a standard write of data into memory. If the operation is an overwrite, the rate of data transfer to the MCM will be controlled by the MCM.

6. Multiple-Word Protected Write (1 > N > 4). This operation is a conditional write of data into memory. The data is written into memory as long as none of the addresses are protected (i.e., bit 48 = 0 for each address). The requestor will transmit the data only upon request of the MCM. The MCM will transmit a Fail S signal to the requestor if any of the addresses were protected, and it will unconditionally flashback data to the requestor.

7. Load Requestor Inhibit Register. This operation is similar to a single-word overwrite with the exception that the data word is transferred to the requestor inhibit register instead of to the MSU. The state of the requestor inhibit register determines which requestors may communicate with the MCM.

8. Load Memory Limit. This operation is similar to a single-word overwrite with the exception that the limits field within the data word is transferred to the memory limit register instead of to the MSU. The memory limits consist of the lower and upper MCM memory addresses and the MSU's available for usage by the MCM.

MCM LOGIC FUNCTIONS

The basic logic functions of the MCM are: priority resolution, data transfer and control, and error protection. (See the block diagram in figure V-1-11.)

PRIORITY-RESOLUTION LOGIC

Priority-resolution logic controls communications between each requestor and the MCM. Lower numbered requestors are given the highest-priority access into memory. Only those requestors selected by the state of the requestor-inhibit register are allowed to be serviced by the MCM. The exception to this rule is that through the use of the special request signal, CPM's are able to override the state of the requestor inhibit register. A requestor is not serviced if the requestor's interface has failed so that other requestors are not locked out. The highest-priority requestor is prevented from obtaining consecutive service if a lower priority requestor is waiting to be serviced.

DATA-TRANSFER-AND CONTROL LOGIC

The data-transfer and control logic provides the sequential control signals required to route the data through the four main data registers (input, output, control word, and memory buffer registers). A brief description of these registers is provided below:

a. Input Register. A 52-bit register used as a temporary buffer register for control words and data words received from memory.

b. Memory Buffer Register. A 60-bit register used as a temporary buffer register for data words transferred to or from MSU's. The memory buffer register. During a fetch the fail-register operation, the fail register information, except bit FR51, is transferred to the MBR before being placed in the output register.

c. Control Word Register. A 52-bit register used to contain the control word transmitted by the requestor.

d. Output Register. A 52-bit register used to buffer data words that are being transmitted to a requestor during a fetch operation. The output register also contains the bit-correction logic required to correct one-bit errors detected by the error-correction logic.

ERROR-DETECTION LOGIC

The error-detection logic detects errors in requestor and MSU data and control interface; detects multiple bit errors; corrects on-bit errors that occur in the MSU during a fetch operation; and detects an internal error if a failure occurs in the check/generator logic.

MSU OPERATIONAL MODES

There are three operational modes provided by the MSU: read/restore, clear/write, and read/modify/write. A brief description of these operational modes is given below:

1. Read/Restore. The MSU reads out data from the memory address defined by the MCM and places the data on the bus to the MCM. The MSU rewrites the information back into the core memory at the defined address.

The following MCM operations use this MSU operation:

a. Single-word fetch

b. N-length word fetch

2. Clear/Write. The MSU reads out data from the memory location addressed by the MCM and places the data on the bus to the MCM. The MSU accepts information from the MCM and stores it into the addressed location.

The following MCM operations use this MSU operation:

a. Single-word overwrite with or without flashback.

b. N-length overwrite.

c. Single-word protected write.

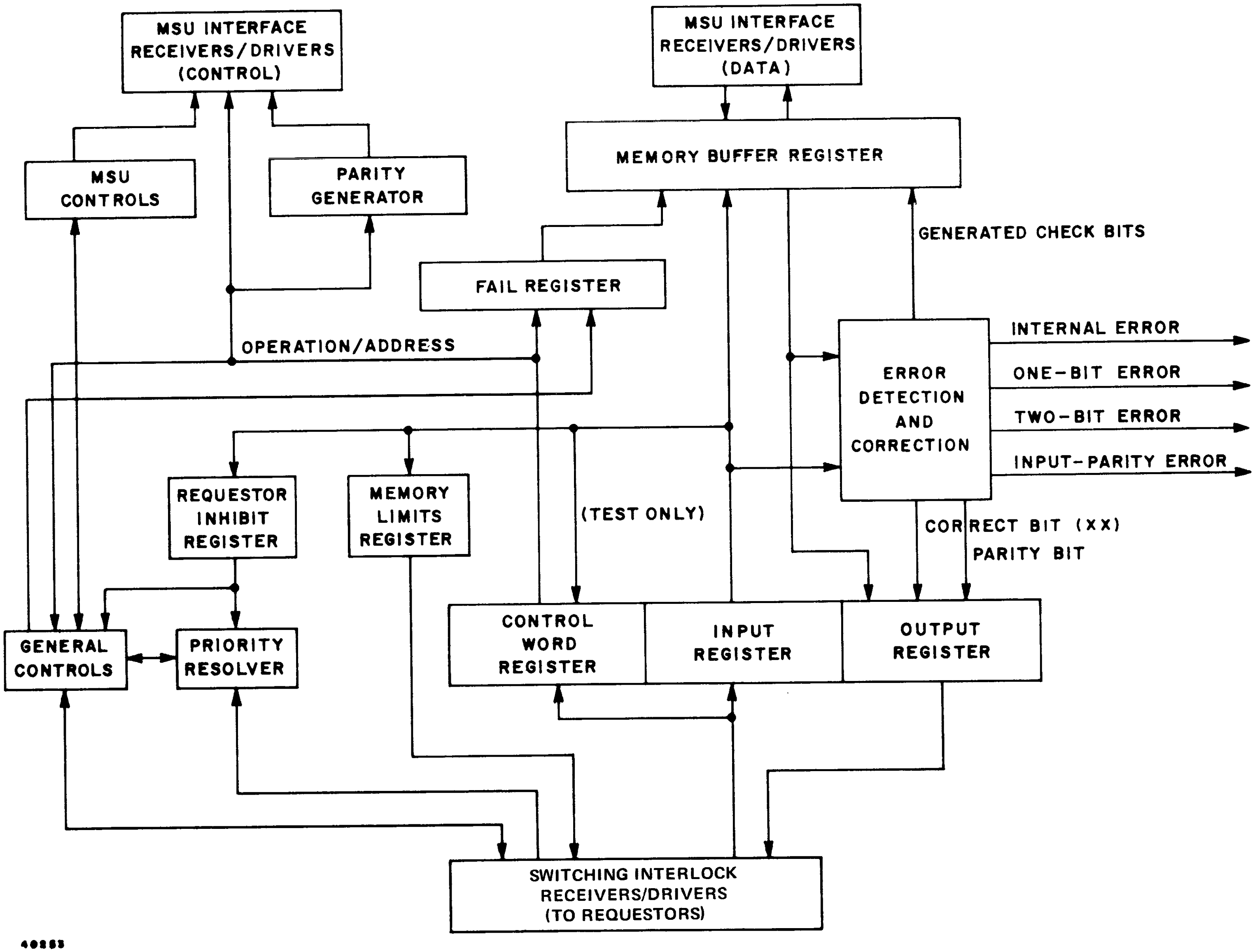
3. Read/Modify/Write. The MSU reads out data from the memory location addressed by the MCM and places the information on the bus to the MCM. The MSU on command from the MCM, stores into the same address either the original information read from memory or information transmitted from the MCM. The N-word protected write uses this MSU operation.

MASTER CLOCK AND SYSTEM DISTRIBUTION

The master clock for a B 7700 System is housed in the MCM cabinet designated MCM-0. Although all MCM's are so configured that they could house the master clock kit, only one master clock is used per system.

The master clock consists of three circuit cards: the crystal-controlled master clock, 2MHz countdown, and crystal-controlled 5MHz clock. The crystal-controlled master clock provides three outputs consisting of the following: a 16MHz signal which is supplied to the CPM's as the clock signal for the program control unit, storage unit, and execution unit; an 8MHz phase-1 signal which is supplied to the CPM's (communications unit only), IOM's, and MDU's as the basic clock signal for internal and interface timing; and an 8MHz phase-2 signal which is supplied to all MCM's as the basic clock signal for internal and interface timing. The 2MHz countdown circuit card steps down the 8MHz phase-1 signal to provide a 2MHz clock signal for the disk file optimizer (DFO). (The DFO does not contain an internal clock generator.) The 5MHz crystal-controlled oscillator provides the clock signal for the data communications processor.

The master clock system obtains its dc input power from special power supplies that are isolated from the normal power supplies in each module. Therefore, if the MCM containing the master clock is shut down, the master clock will continue to drive the other MCM's and system modules.



V-1-11. Memory Control Module Block Diagram

CHAPTER VI

CONTROLS AND INDICATORS

SECTION 1

OPERATING CONTROLS

INTRODUCTION

This section describes the controls and indicators used in the operation of the B 7700 system. These controls and indicators are located on the operators control console and on the cold start/halt load selection cards. The operators control console (figure VI-1-1) consists of a console control panel and one or more supervisory console devices, each of which contains an input keyboard and a video output screen. The cold start/halt load selection cards are mounted in the backplane of each CPM and IOM.

CONSOLE CONTROL PANEL

The console control panel (figure VI-1-2) includes the following switches and indicators.

SYSTEM A/SYSTEM B – This switch selects the portion of the system which will be affected by the remaining switches on the console control panel. Each main frame module (CPM's, IOM's, and MCM's) includes a SYSTEM A/SYSTEM B switch which determines which portion of the system that module belongs to. Only those mainframe modules belonging to the selected portion of the system

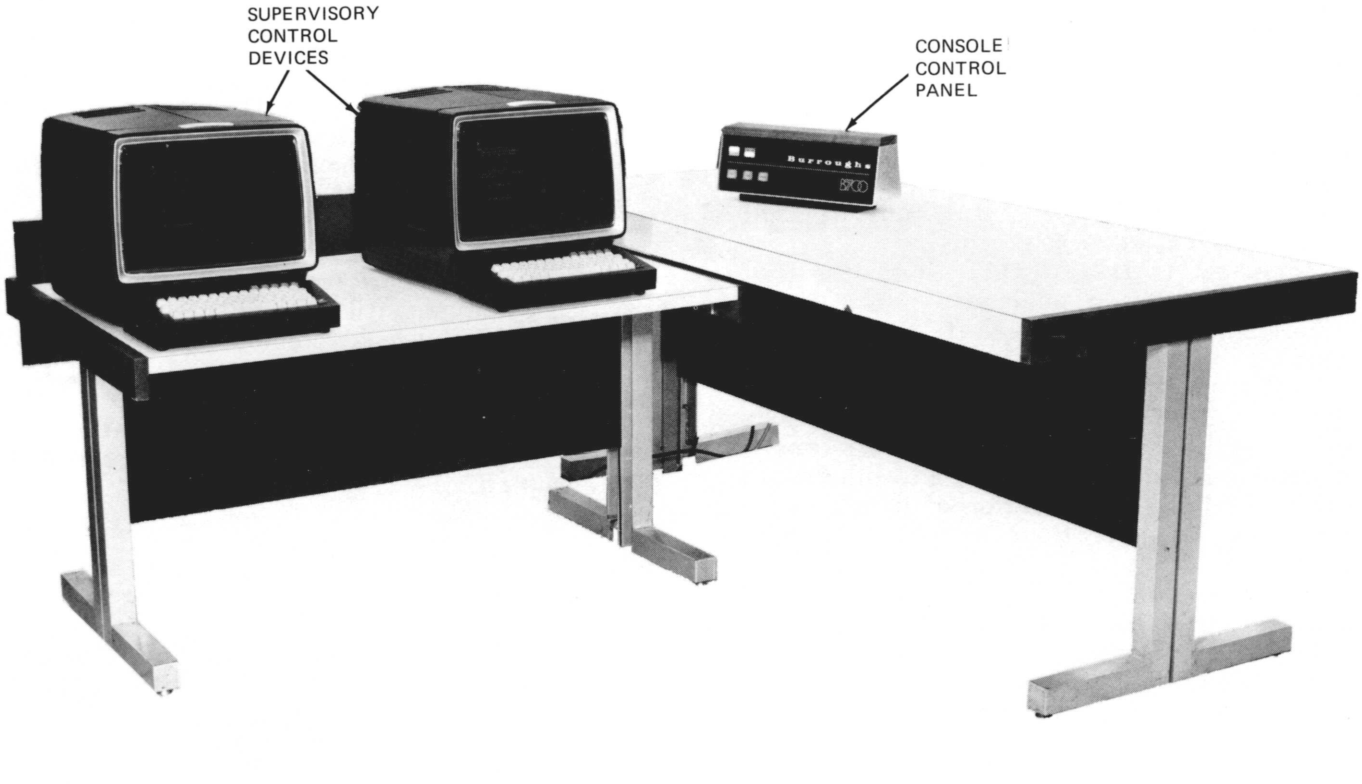


Figure VI-1-1. Operators Control Console

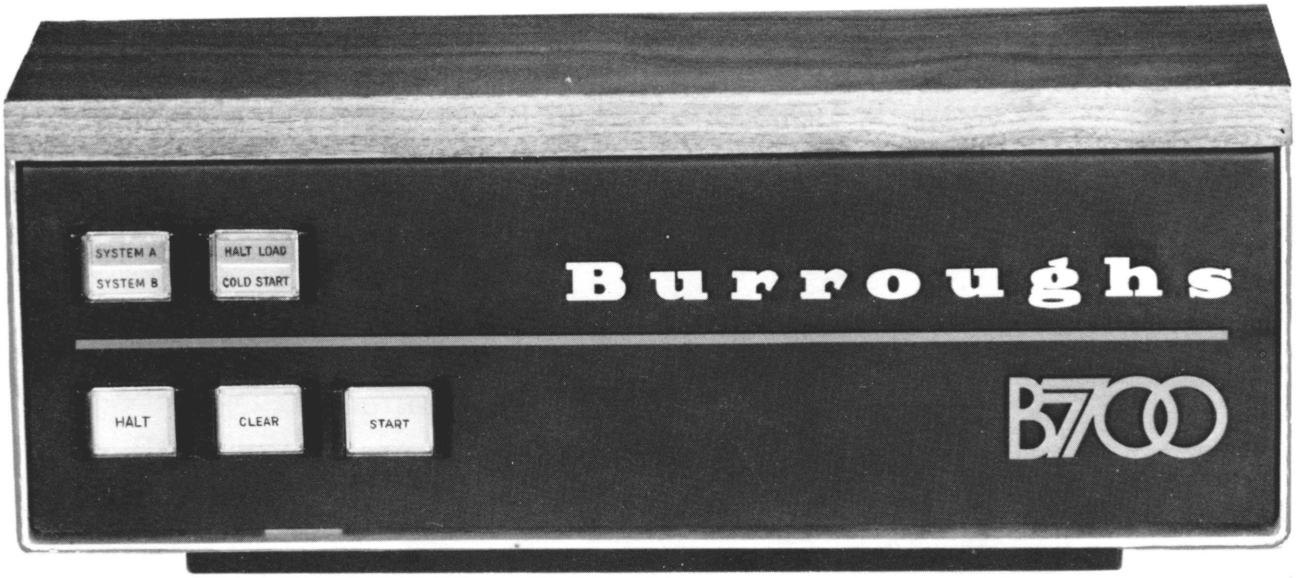


Figure VI-1-2. Console Control Panel

are affected by operation of the console control panel switches.

The SYSTEM A/SYSTEM B switch is located on a switch card (the COLD START/HALT LOAD selection card) mounted on the backplane of a CPM or IOM. The SYSTEM A/SYSTEM B switch is mounted on the cabinet frame at the right side of the panel of an MCM.

DISK LOAD/CARD LOAD – This switch selects the load operation that will be performed when the system is started.

When set to CARD LOAD, indicate that a card load has been selected.

When set to DISK LOAD, indicates that a disk load has been selected.

Note that this switch selects the mode of operation, but does not cause the operation.

HALT – Halts the system in an orderly fashion. Sets CPM's to single instruct mode, which causes a halt at the end of the current program operator, and sets IOM's to inhibit state, which inhibits the initiation of further I/O operations. I/O operations in progress will be completed.

LOAD – Clears all hard registers and flip-flops to the reset state. Sets IOM's to inhibit state, sets IOM Home Address registers to the memory segment indicated by the switch settings, and clears the peripheral controllers. Sets the MCM requestor inhibit, address limits, and availability registers to the switch settings. Sets the CPM's yp control state, control mode 3 in a forced PAUS instruction. Upon receipt of an external interrupt, the CPM will execute code starting at word 8 of the memory segment set into the switch card.

ENABLE – Causes the IOM selected as the load IOM to initiate a read operation from either card or disk, depending on the setting of the COLD START/HALT LOAD switch.

If a card load was selected, cards will be read from the card reader specified by the switch settings until a validity check occurs.

If a disk load was selected, 8192 words will be read from address zero of the disk unit specified by the switch settings.

At the completion of the read operation, the CPM specified by the switch settings will be interrupted.

SUPERVISORY CONSOLE

The supervisory console (figure VI-1-3) contains an input keyboard and a video output screen.

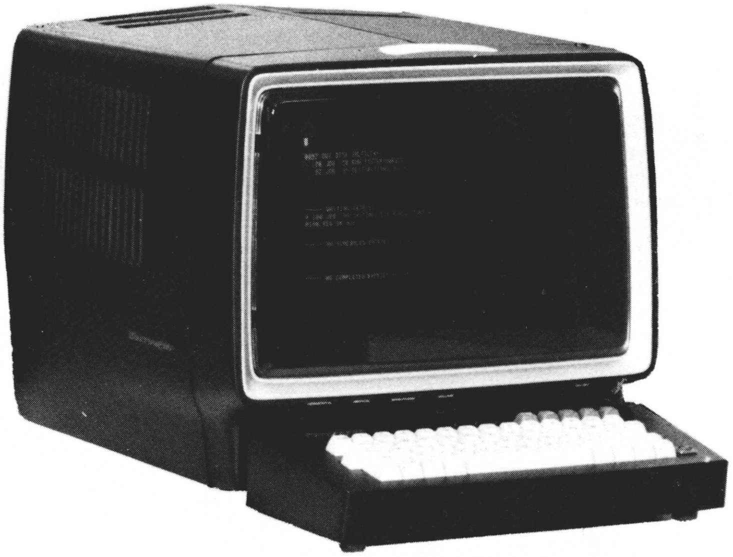


Figure VI-3. Supervisory Console

KEYBOARD CONTROL KEYS

The following is a list of the keyboard control keys and their functions (figure VI-1-4):

XMT mode key and indicator
The transmit (XMT) mode indicator is illuminated when the transmit (XMT) key is depressed. The indicator is extinguished when a transmission from the terminal has been positively acknowledged by the receiving station or the terminal is changed to the local mode by the operator.

RCV mode key and indicator
The receive (RCV) mode indicator, which indicates that the terminal is ready to receive data, is illuminated when the receive (RCV) key is depressed. The indicator is also illuminated when a transmission from the terminal has been successfully completed. The RCV indicator is extinguished when the terminal is switched to local mode, or transmit mode.

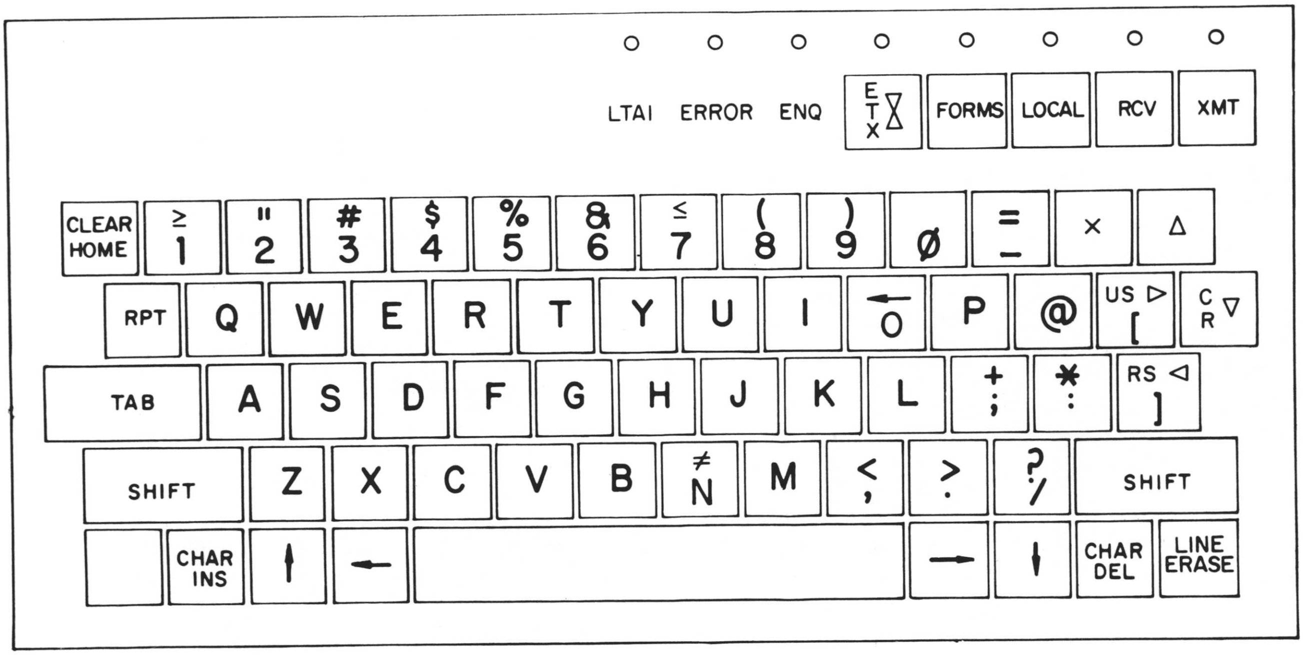


Figure VI-1-4. Keyboard Format

LOCAL mode key and indicator

The local mode indicator is illuminated when the LOCAL key is depressed or by the use of the keyboard when the terminal is in the receive mode with no data being received into the terminal. The indicator is extinguished when the terminal is switched to the receive mode or transmit mode.

ERROR indicator

The error indicator is illuminated when a parity error is detected by the terminal in the data being received, or when the buffer overflows because the received message contains more characters than the character capacity of the display circuits. The error indicator is extinguished by the successful retransmission to the terminal, the receipt of a new message.

ENQ indicator

The enquiry (ENQ) indicator is illuminated when the terminal detects the central processor (CP) attempting to transmit a message to the terminal while the terminal is not in the receive mode. The indicator is extinguished by the operator placing the terminal in the receive or local mode. Also, the audible alarm momentarily sounds when the ENQ indicator is lit.

FORMS push button and indicator

The FORMS indicator is illuminated whenever the terminal is operating in the forms status. The terminal operates in the forms status when the FORMS push button is depressed by the operator and delimiters are present. The FORMS indicator is extinguished either by the receipt of a CP message or by depressing the FORMS push button.

LTAI indicator

The line terminal activity indicator (LTAI) is illuminated whenever data is transmitted from the CP to the terminal on the line. When the terminal responds to the CP, the LTAI indicator is extinguished. In normal operation, the LTAI will blink due to the data line activity. An LTAI which is not illuminated indicates that the CP is not transmitting on that line. An LTAI which remains illuminated indicates that the terminal is not responding.

ETX (X)

The ETX (end-of-text) key causes the ETX symbol (X) to be stored in memory and displayed at the cursor location. When this operation is completed, the cursor is then automatically moved to the home position.

Key

↓ (Line feed)

Line feed is used to move the cursor one line down. When the cursor is in the bottom line, depressing the line feed key causes the cursor to reappear in the top line. This function is disabled when the terminal is in the forms status.

↑ (Reverse Line feed)

Reverse Line feed is used to move the cursor one line up. When the cursor is in the top line, depressing the reverse line feed key causes the cursor to reappear in the bottom line. This function is disabled when the terminal is in the forms status.

← (Backspace)

Backspace is used to move the cursor one character to the left. When the cursor is in the first character position (left edge) of the display, depressing the backspace key causes the cursor to reappear in the last character position (right edge) of the next higher line. When the cursor is in the "home" position (top line, left edge), depressing the backspace key causes the cursor to reappear in the last character position (bottom line, right edge). In forms status, backspace is enabled only to the first character location in an unprotected data field.

→ (Forward space)

Forward space is used to move the cursor one character position to the right. If the cursor is at the right edge of a line, depressing the forward space key causes the cursor to reappear at the left edge, down shifted one line. If the cursor is located in the last character position of the bottom line, depressing the forward space key causes the cursor to reappear in the home position. In forms status, forward space causes the cursor to move from the last position of an unprotected field to the first position of the succeeding unprotected field.

HOME

The HOME key is used to move the cursor to the upper left (home) position. In forms status, HOME causes the cursor to be moved to the first position of the first unprotected data field. The HOME key operates in unshifted mode only.

CLEAR

Depressing the CLEAR key and the shift key will cause all data on the display to be erased except protected data while in the forms mode.

RPT

Repeat (RPT), when depressed along with any alphanumeric key, return, insert, delete, line feed, reverse line feed, backspace, or forward space, causes the repetition of that character or function in successive display and memory locations.

C ▽
R
(Carriage Return)

The carriage return key is used to move the cursor from its position in a line to the first position of the following line. The carriage return symbol (▽) is stored and displayed at the cursor position before the actual carriage return takes place. In forms status, the carriage return key is disabled.

TAB

TAB is used to move the cursor forward to the next tab stop location. The tab stops are located at character positions 1, 9, 17, 25, 33, 41, 49, 57, 65, and 73 of each line. In the forms status, TAB causes the cursor to move forward to the first unprotected character position following the leading delimiter of the next unprotected character field.

US ▷
(Unrestricted Delimiter)

Depressing the US ▷ key and the shift key will cause an unrestricted delimiter symbol (▷) to be inserted into memory and displayed at the cursor location. This symbol has no significance until the terminal is placed in the forms mode. In the forms mode, the data on the screen between the US symbol (▷) and the RS symbol (◁) is interpreted as unprotected data and the US ▷ key on the keyboard is disabled.

RS ◁
(Restricted Delimiter)

Depressing the RS ◁ key and the shift key will cause a restricted delimiter symbol (◁) to be inserted into memory and displayed at the cursor location. This symbol has no significance until the terminal is placed in the forms mode. In the forms mode, the data on the screen between the RS symbol (◁) and the US symbol (▷) is interpreted as protected data and the RS ◁ key on the keyboard is disabled.

LINE ERASE

The LINE ERASE key when depressed, will erase all of the data on a line to the right of, and including the cursor position. In forms status, the LINE ERASE key is disabled.

CHAR INS
(Character Insert)

The character insert key when depressed, inserts a space at the cursor location, and, together with the key of the character to be inserted causes the added character to appear at the cursor location. The succeeding characters are moved one space to the right and downward from line to line. Surplus characters, if any, are shifted off the display at the end of the last line.

CHAR DEL
(Character Delete)

The character delete key is used to remove a displayed character from the cursor location. When the CHAR DEL key is depressed, the succeeding characters are moved one space to the left and upward from line to line.

DISK LOAD/CARD LOAD

The COLD START/HALT LOAD Selection Card (figure VI-1-5) mounted in the backplanes of each CPM and IOM contain switches which are used to identify the CPM, MCM, IOM, card reader, and disk file electronics unit to be used when initializing the system.

The COLD START/HALT LOAD Selection Card contains the following switches:

Select (8) - Selects the load IOM. Must be ON for the load IOM, and OFF for all others in that portion of the system.

A/B (7) - Determines which portion of the system (system A or system B) this module belongs to.

MEMORY NO. (1-6) - Indicates the 16K memory segment to be used for initialization (most significant six bits of the memory address). Should be set to 000000.

CPM INTERRUPT NO. (14-16) - Indicates the channel number (binary) of the load CPM, i.e., the CPM which is to be interrupted following the load operation.

CARD READER CHANNEL NO. (9-13) - The number of the channel (internal to the load IOM) to which the load card reader is connected.

DISK FILE CHANNEL NO. (17-21) - The number of the channel (internal to the load IOM) to which the desired disk file exchange is connected.

EU UNIT DESIGNATE NO. (25-32) - The unit designate number of the desired electronics unit. The transfer from the EU will start at SU 0, zone 0, track 0, segment 0 of the specified EU.

A CPM uses only the MEMORY NO. and A/B switches. An IOM other than the load IOM uses the SELECT switch (must be OFF), the MEMORY NO., A/B, and CPM INTERRUPT NO. switches. The load IOM uses all switches.

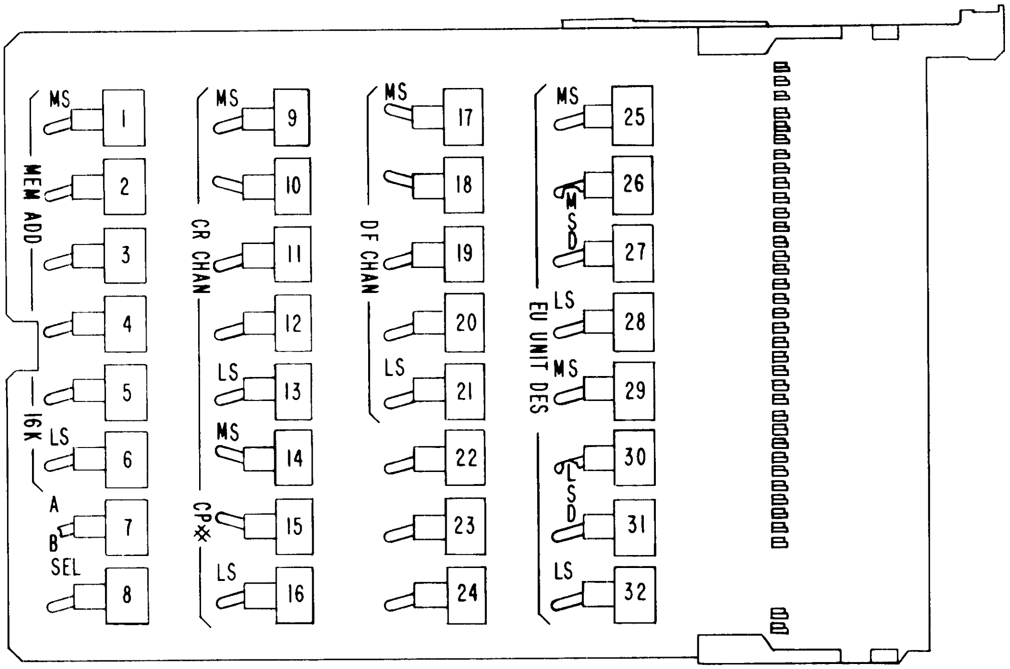


Figure VI-1-5. Cold Start/Halt Load Selection Card

SECTION 2

CENTRAL PROCESSOR MODULE PANELS

This section describes the functions and uses of the controls and indicators on the two panels of the central processor module (CPM). An overall view of the central processor module is shown in figure VI-2-1, and close-ups of the panels are shown in figures VI-2-2 and VI-2-3.

DISPLAY ORGANIZATION

The Central Processor Module has two adjacent panels for display purposes. Each panel has fifteen rows of indicators, a row of 51 toggle switches, and a section of maintenance switches. Legends are printed on the left-hand margin of each panel which indicate the general area a series of rows pertain to. Row numbers are printed on the right-hand margin.

Occasionally, a row of indicators will have two printed row numbers. When this occurs, the single row of indicators is used to display one of two sets of data as specified by a two position toggle switch which is to the left of the row markings. The descriptions of each set either have the format upper switch setting/ lower switch setting or are printed above and below the individual indicators.

The indicators within each row are separated into groups of four by color for hexadecimal grouping purposes as is the row of 51 toggle switches.

PANEL 2 INDICATORS (LEFT-HAND PANEL)

EU DATA SECTION

ROW 1 - ROW 21 DISPLAY SELECTION

The first row of indicators is used to display the contents of either row 1 or row 21. The selection of the row to be displayed is made through use of a toggle switch which is located on the right hand side of the panel adjacent to the first row of indicators and which is labeled EW-C.

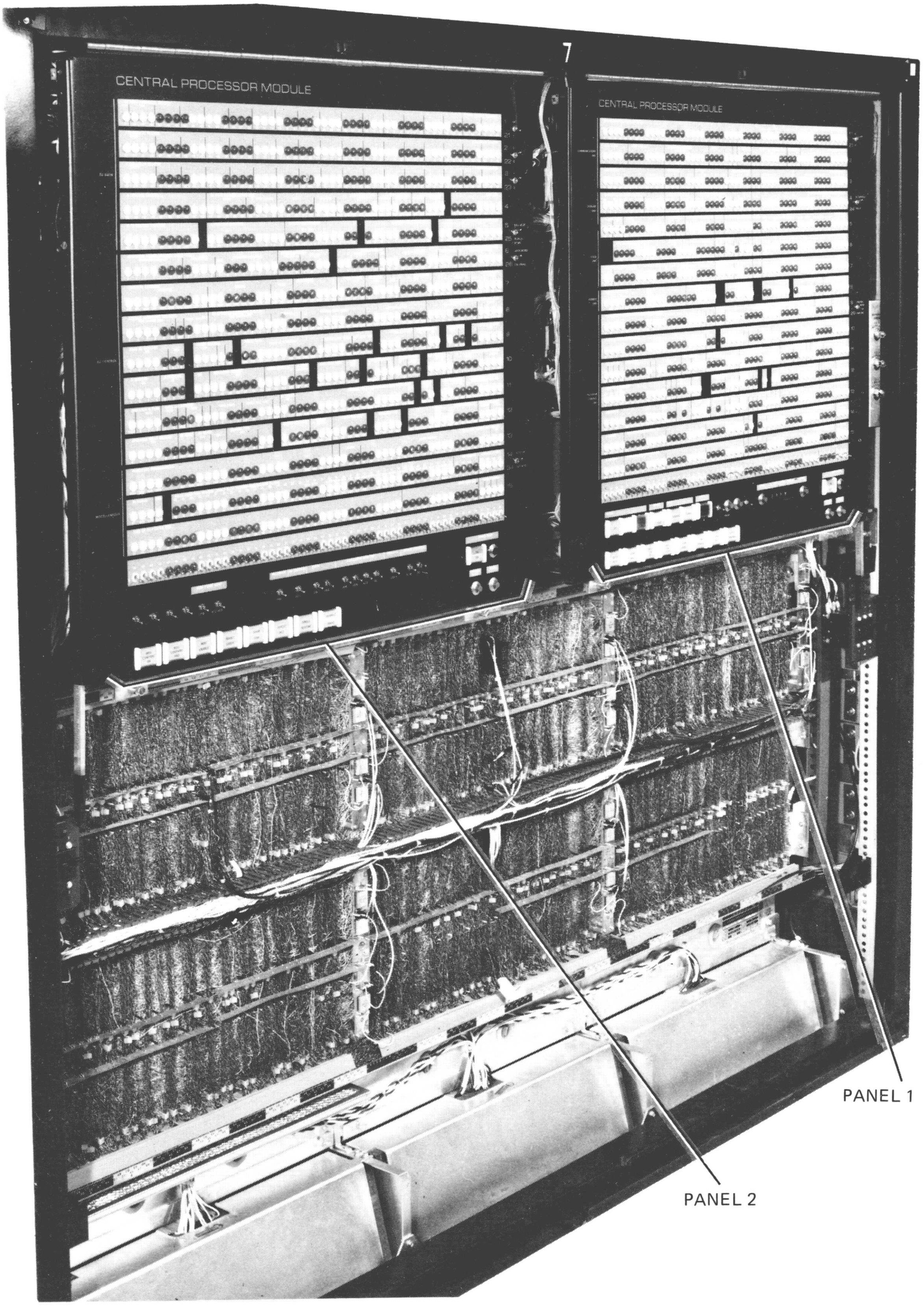
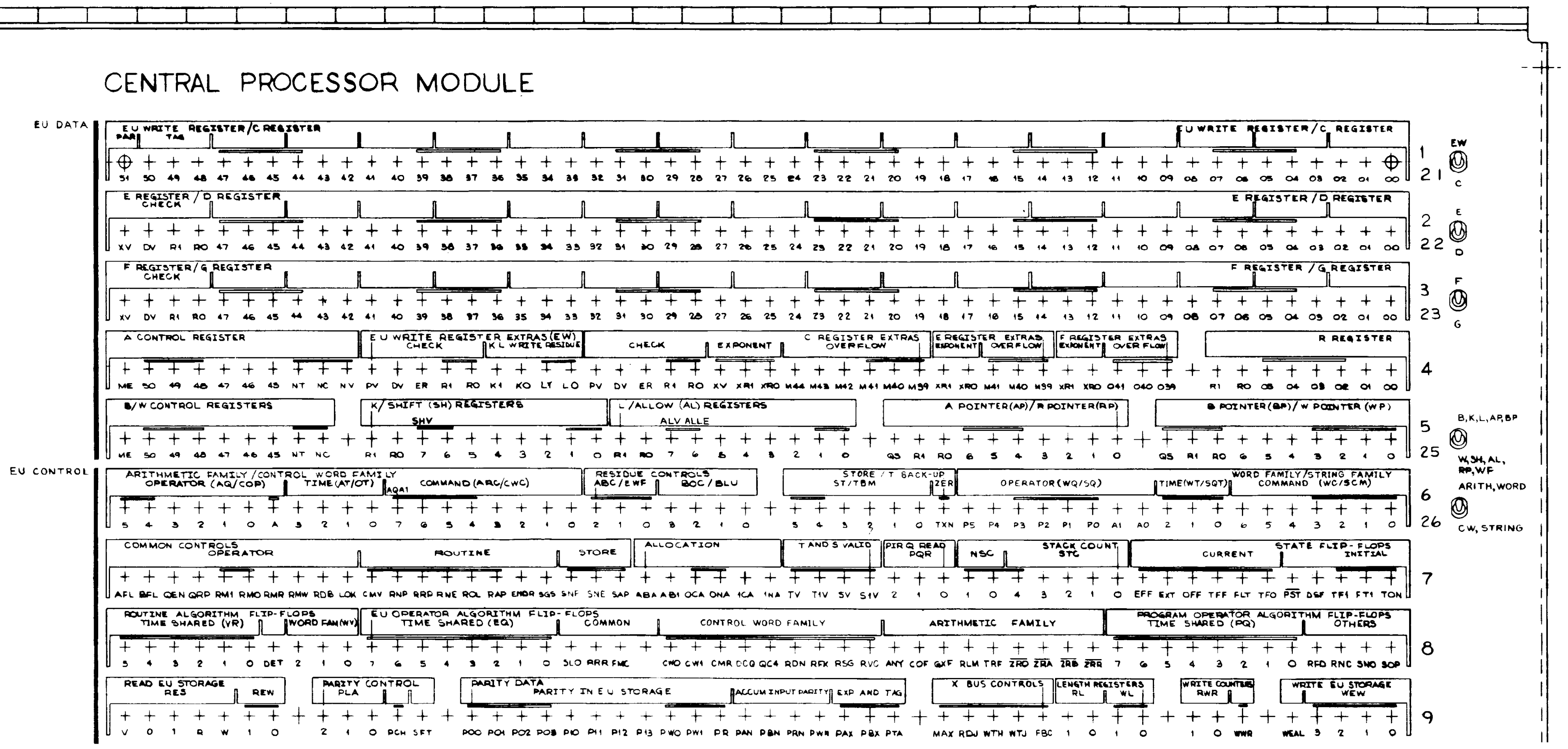


Figure VI-2-1. Central Processor Module

CENTRAL PROCESSOR MODULE


EU DATA
E U WRITE REGISTER/C REGISTER
PAR
TAG
E U WRITE REGISTER/C REGISTER
51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1
21
EW
C
E REGISTER / D REGISTER
CHECK
E REGISTER / D REGISTER
XV DV R1 RO 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
2
22
E
D
F REGISTER/G REGISTER
CHECK
F REGISTER / G REGISTER
XV DV R1 RO 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
3
23
F
G
A CONTROL REGISTER
E U WRITE REGISTER EXTRAS (EW)
CHECK
K L WRITE RESIDUE
CHECK
EXPONENT
C REGISTER EXTRAS
OVERFLOW
E REGISTER EXTRAS
EXPONENT
OVERFLOW
F REGISTER EXTRAS
EXPONENT
OVER FLOW
R REGISTER
ME 50 49 48 47 46 45 NT NC NV PV DV ER R1 RO K1 KO LT LO PV DV ER R1 RO XV XR1 XRO M44 M43 M42 M41 M40 M39 XR1 XRO M41 M40 M39 XR1 XRO O41 O40 O39 R1 RO O5 O4 O3 O2 O1 OO
4
B/W CONTROL REGISTERS
K/ SHIFT (SH) REGISTERS
SHV
L /ALLOW (AL) REGISTERS
ALV ALLE
A POINTER(AP)/R POINTER(RP)
B POINTER(BP)/ W POINTER (WP)
ME 50 49 48 47 46 45 NT NC R1 RO 7 6 5 4 3 2 1 0 R1 RO 7 6 5 4 3 2 1 0 Q5 R1 RO 6 5 4 3 2 1 0 Q5 R1 RO 6 5 4 3 2 1 0
5
25
B,K,L,AP,BP
W,SH,AL,
RP,WF
EU CONTROL
ARITHMETIC FAMILY /CONTROL WORD FAMILY
OPERATOR (AQ/COP)
TIME (AT/OT)
AQA1
COMMAND (ARC/CWC)
RESIDUE CONTROLS
ABC / EWF
BOC / BLU
STORE / T BACK-UP
ST/TBM
ZER
OPERATOR (WQ/SQ)
TIME(WT/SQT)
WORD FAMILY/STRING FAMILY
COMMAND (WC/SCM)
5 4 3 2 1 0 A 3 2 1 0 7 6 5 4 3 2 1 0 2 1 0 3 2 1 0 5 4 3 2 1 0 TXN P5 P4 P3 P2 P1 PO A1 AO 2 1 0 6 5 4 3 2 1 0
6
26
ARITH, WORD
CW, STRING
COMMON CONTROLS
OPERATOR
ROUTINE
STORE
ALLOCATION
T AND S VALID
PIR Q READ
PQR
NSC
STACK COUNT
STC
STATE FLIP-FLOPS
CURRENT
INITIAL
AFL BFL QEN QRP RM1 RMO RMR RMW RDB LOK CMV RNP RRP RNE ROL RAP ENDR SGS SNF SNE SAP ABA AB1 OCA ONA 1CA 1NA TV T1V SV S1V 2 1 0 1 0 4 3 2 1 0 EFF EXT OFF TFF FLT TFO $\overline{\mathrm{PST}}$ DSF TF1 FT1 TON
7
ROUTINE ALGORITHM FLIP-FLOPS
TIME SHARED (VR)
WORD FAM(WV)
E U OPERATOR ALGORITHM FLIP-FLOPS
TIME SHARED (EQ)
COMMON
CONTROL WORD FAMILY
ARITHMETIC FAMILY
PROGRAM OPERATOR ALGORITHM FLIP-FLOPS
TIME SHARED (PQ)
OTHERS
5 4 3 2 1 0 DET 2 1 0 7 6 5 4 3 2 1 0 SLO RRR FMC CWO CW1 CMR DCQ QC4 RDN RFK RSG RVC ANY COF GXF RLM TRF $\overline{\mathrm{ZRO}}$ $\overline{\mathrm{ZRA}}$ $\overline{\mathrm{ZRB}}$ $\overline{\mathrm{ZRR}}$ 7 6 5 4 3 2 1 0 RFD RNC SNO SOP
8
READ EU STORAGE
RES
REW
PARITY CONTROL
PLA
PARITY DATA
PARITY IN E U STORAGE
ACCUM INPUT PARITY
EXP AND TAG
X BUS CONTROLS
LENGTH REGISTERS
RL
WL
WRITE COUNTERS
RWR
WRITE EU STORAGE
WEW
V 0 1 R W 1 0 2 1 0 PCH SFT POO PO1 PO2 PO3 P10 P11 P12 P13 PWO PW1 PR PAN PBN PRN PWR PAX PBX PTA MAX RDJ WTH WTJ FBC 1 0 1 0 1 0 WWR WEAL 3 2 1 0
9

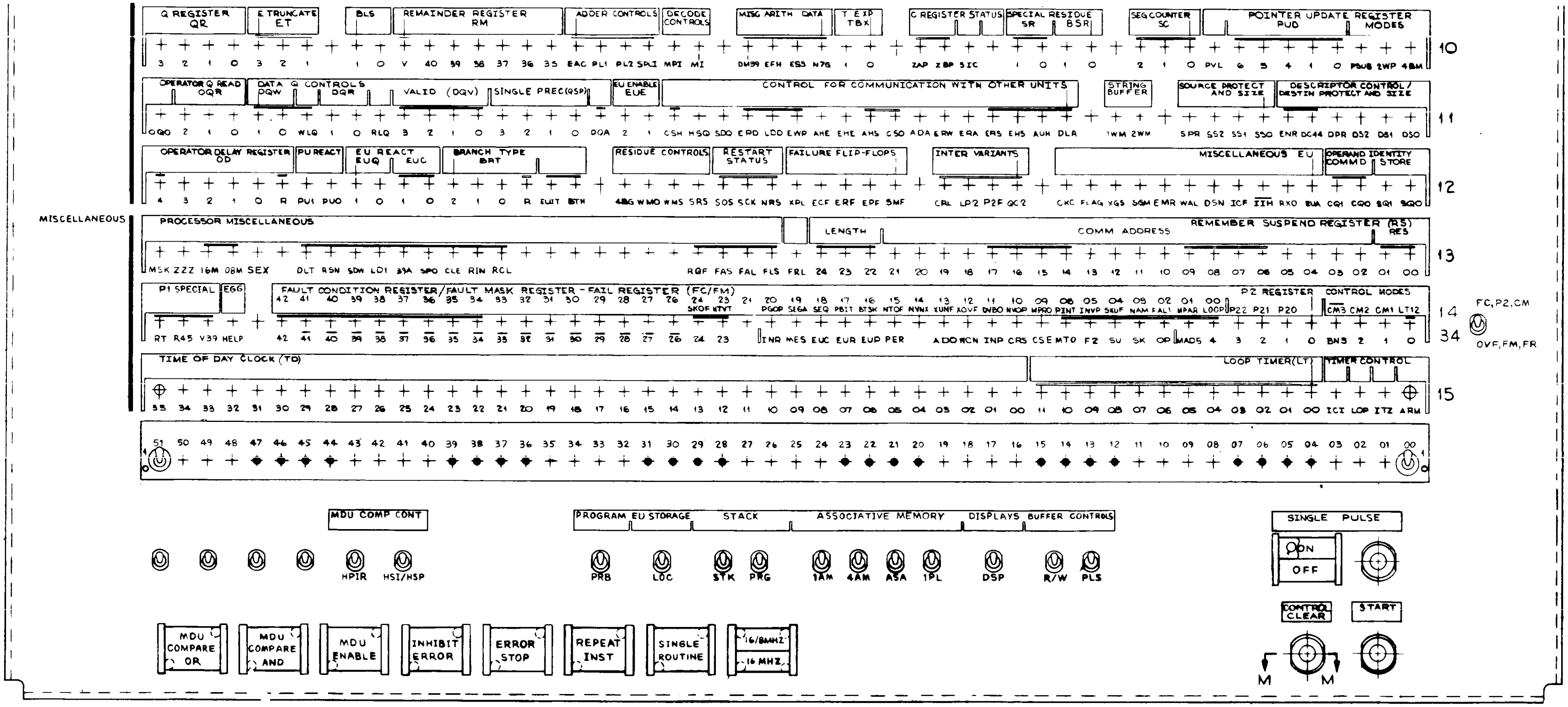
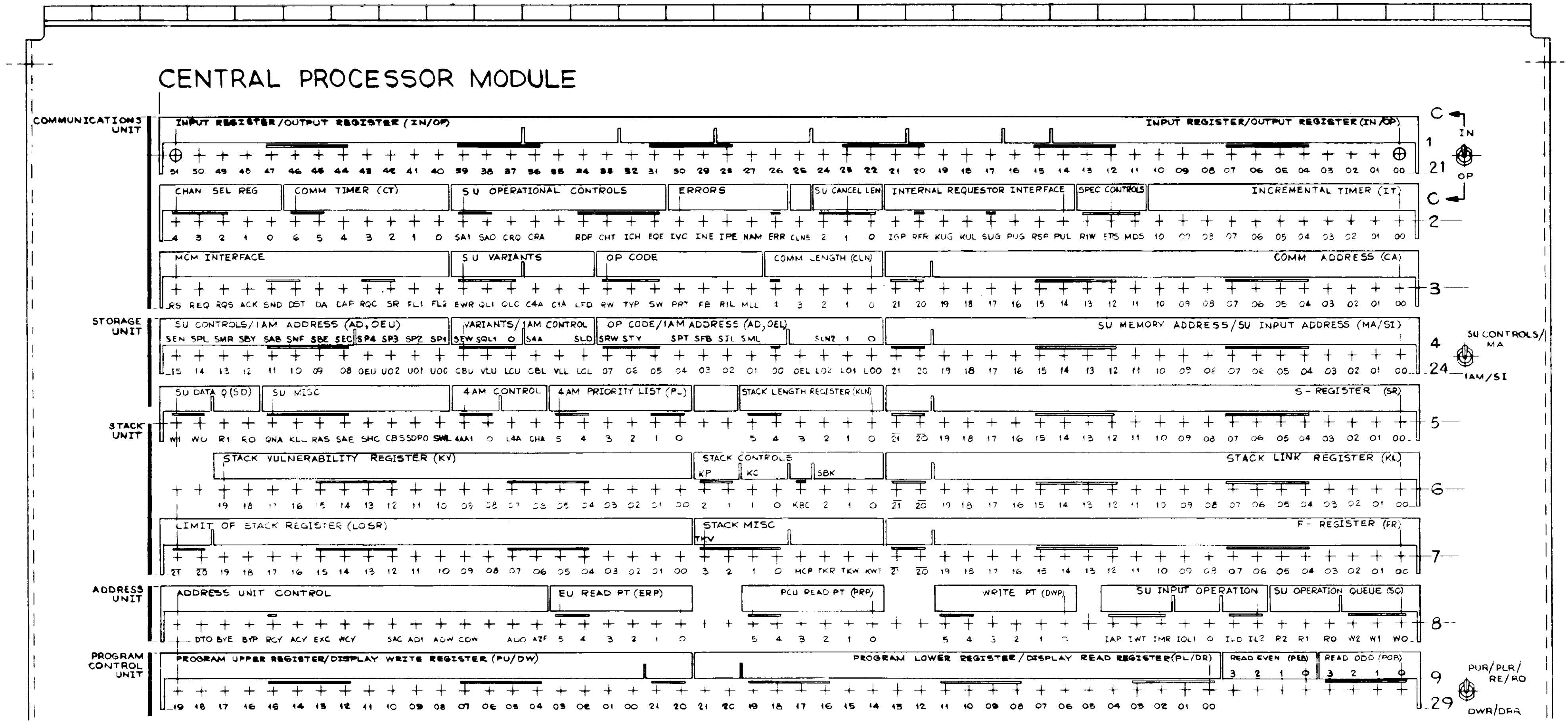


Figure VI-2-2. Panel 2 (Left-Hand Panel) of CPM


CENTRAL PROCESSOR MODULE
COMMUNICATIONS UNIT
STORAGE UNIT
STACK UNIT
ADDRESS UNIT
PROGRAM CONTROL UNIT
INPUT REGISTER/OUTPUT REGISTER (IN/OP)
51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
CHAN SEL REG
COMM TIMER (CT)
SU OPERATIONAL CONTROLS
ERRORS
SU CANCEL LEN
INTERNAL REQUESTOR INTERFACE
SPEC CONTROLS
INCREMENTAL TIMER (IT)
4 3 2 1 0 6 5 4 3 2 1 0 SA1 SAO CRQ CRA RDP CHT ICH EQE IVC INE IPE NAM ERR CLNS 2 1 0 IGP RFR KUG KUL SUG PUG RSP PUL RIW ETS MDS 10 09 08 07 06 05 04 03 02 01 00
MCM INTERFACE
SU VARIANTS
OP CODE
COMM LENGTH (CLN)
COMM ADDRESS (CA)
RS REQ RQS ACK SND DST DA CAF RQC SR FL1 FL2 EWR QLI OLC C4A CIA LFD RW TYP SW PRT FB RIL MLL 4 3 2 1 0 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
SU CONTROLS/IAM ADDRESS (AD,OEU)
SEN SPL SMR SBY SAB SNF SBE SEC SP4 SP3 SP2 SP1
VARIANTS/ IAM CONTROL
SEW SQL1 O S4A SLD
OP CODE/IAM ADDRESS (AD,OEL)
SRW STY SPT SFB SIL SML SLN2 1 0
SU MEMORY ADDRESS/SU INPUT ADDRESS (MA/SI)
15 14 13 12 11 10 09 08 OEU UO2 UO1 UOO CBU VLU LCU CBL VLL LCL 07 06 05 04 03 02 01 00 OEL LO2 LO1 LOO 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
SU DATA Q (SD)
SU MISC
4AM CONTROL
4AM PRIORITY LIST (PL)
STACK LENGTH REGISTER (KLN)
S-REGISTER (SR)
WI WO R1 RO QNA KLL RAS SAE SHC CBS SDPO SWL 4AA1 O L4A CHA 5 4 3 2 1 0 5 4 3 2 1 0 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
STACK VULNERABILITY REGISTER (KV)
STACK CONTROLS
KP
KC
SBK
STACK LINK REGISTER (KL)
19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 2 1 0 KBC 2 1 0 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
LIMIT OF STACK REGISTER (LOSR)
STACK MISC
THV
F-REGISTER (FR)
21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 3 2 1 0 MCP TKR TKW KWI 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
ADDRESS UNIT CONTROL
EU READ PT (ERP)
PCU READ PT (PRP)
WRITE PT (DWP)
SU INPUT OPERATION
SU OPERATION QUEUE (SQ)
DTO BVE BYP RCY ACY EXC WCY SAC ADI ADW CDW AUO AZF 5 4 3 2 1 0 5 4 3 2 1 0 5 4 3 2 1 0 IAP IWT IMR IQL1 O ILO IL2 R2 R1 RO W2 W1 WO
PROGRAM UPPER REGISTER/DISPLAY WRITE REGISTER (PU/DW)
PROGRAM LOWER REGISTER/DISPLAY READ REGISTER (PL/DR)
READ EVEN (PEB)
3 2 1 0
READ ODD (POB)
3 2 1 0
19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 21 20 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
C
IN
OP
C
1
21
2
3
4
24
SU CONTROLS/ MA
IAM/SI
5
6
7
8
9
29
PUR/PLR/ RE/RO
DWR/DRR

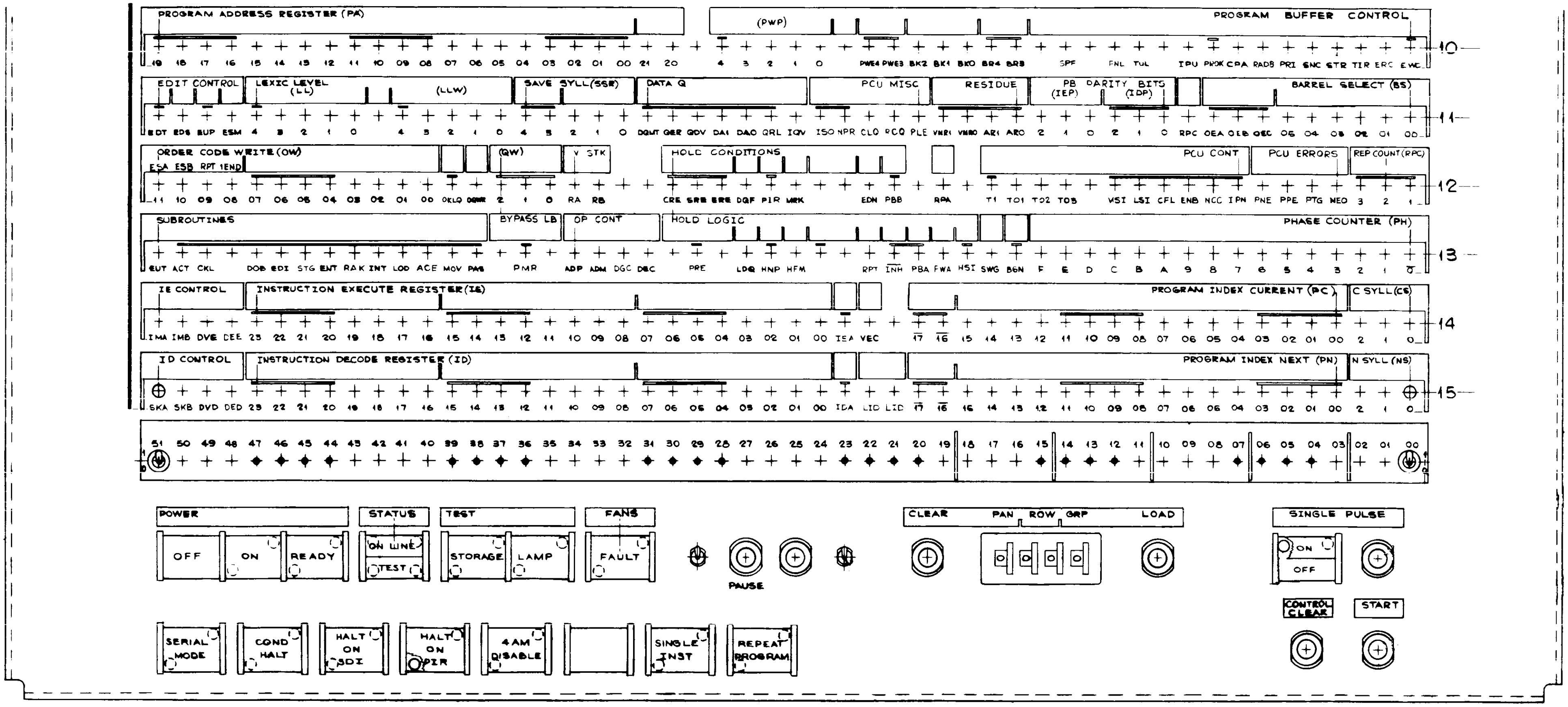

PROGRAM ADDRESS REGISTER (PA)
(PWP)
PROGRAM BUFFER CONTROL
19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 21 20 4 3 2 1 0 PWE4 PWE3 BK2 BK1 BKO BR4 BR3 SPF FNL TUL IPU PROK CPA RADB PRI SNC STR TIR ERC EWC
10
EDIT CONTROL
LEXIC LEVEL (LL)
(LLW)
SAVE SYLL(SSR)
DATA Q
PCU MISC
RESIDUE
PB PARITY BITS
(IEP)
(IDP)
BARREL SELECT (BS)
EDT EDS EUP ESM 4 3 2 1 0 4 3 2 1 0 4 3 2 1 0 DQUT QER QDV DA1 DAO QRL IQV ISO NPR CLO RCQ PLE VNR1 VNRO AR1 ARO 2 1 0 2 1 0 RPC OEA OEB OEC 05 04 03 02 01 00
11
ORDER CODE WRITE (OW)
ESA ESB RPT 1END
(QW)
V STK
HOLD CONDITIONS
PCU CONT
PCU ERRORS
REP COUNT (RPC)
11 10 09 08 07 06 05 04 03 02 01 00 OKLQ OQWR 2 1 0 RA RB CRE SRE ERE DQF PIR MRK EDN PBB RPA T1 TO1 TO2 TO3 VSI LSI CFL ENB NCC IPN PNE PPE PTG NEO 3 2 1
12
SUBROUTINES
BYPASS LB
OP CONT
HOLD LOGIC
PHASE COUNTER (PH)
EUT ACT CKL DOB EDI STG ENT RAK INT LOD ACE MOV PAS PMR ADP ADM DGC DEC PRE LDQ HNP HFM RPT INH PBA FWA HSI SWG BGN F E D C B A 9 8 7 6 5 4 3 2 1 0
13
IE CONTROL
INSTRUCTION EXECUTE REGISTER (IE)
PROGRAM INDEX CURRENT (PC)
C SYLL (CS)
IMA IMB DVE DEE 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 IEA VEC 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 2 1 0
14
ID CONTROL
INSTRUCTION DECODE REGISTER (ID)
PROGRAM INDEX NEXT (PN)
N SYLL (NS)
SKA SKB DVD DED 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 IDA LID LID 17 16 16 14 13 12 11 10 09 08 07 06 06 04 03 02 01 00 2 1 0
15
51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
POWER
OFF
ON
READY
STATUS
ON LINE
TEST
TEST
STORAGE
LAMP
FANS
FAULT
PAUSE
CLEAR
PAN ROW GRP
LOAD
SINGLE PULSE
ON
OFF
CONTROL CLEAR
START
SERIAL MODE
COND HALT
HALT ON SDI
HALT ON PIR
4AM DISABLE
SINGLE INST
REPEAT PROGRAM

Figure VI-2-3. Panel 1 (Right-Hand Panel) of CPM

ROW 1

EU WRITE REGISTER (EW51-0) - The primary input to the Execution Unit (EU) for all data with the exception of operators.

PAR (EW51) - The parity bit of the operand in the EU Write Register (EWR).

TAG (EW50-48) - The tag bits of the operand in the EWR.

ROW 21

C REGISTER (C51-0) - The only output from the EU and the input to EU Local Storage. The following EU error checks occur in this register:

1. Some residue generation.
2. All residue checking.
3. All continuity checking.
4. Some parity generation and checking.

PAR (C51) - The parity bit of the operand in the C Register.

TAG (C50-48) - The Tag bits of the operand in the C Register.

ROW 2 - ROW 22 DISPLAY SELECTION

The second row of indicators is used to display the contents of either row 2 or row 22. The selection of the row to be displayed is made through use of a toggle switch which is located on the right hand side of the panel adjacent to the second row of indicators and which is labeled E-D.

ROW 2

E REGISTER

CHECK

XV - EXV is the "E" Register exponent valid bit which is functional in arithmetic mode only.

DV - EDV is the "E" Register data valid bit which indicates validity for either the entire word when in word mode or for the mantissa when in arithmetic mode.

R1,0 - ER1,0 is the "E" Register residue.

BITS 47-0 - E47-0 is the "E" Register which is one of the inputs to the adder and T bus. This register contains the "A" operand when the processor is in Single Instruct.

ROW 22

D REGISTER

CHECK

XV - DXV is not used.

DV - DDV is the "D" Register data valid bit.

R1,0 - DR1,0 is the "D" Register residue.

BITS 47-0 - D47-0 is the "D" Register which is the input register to the EU Barrel.

ROW 3 - ROW 23 DISPLAY SELECTION

The third row of indicators is used to display the contents of either row 3 or row 23. The selection of the row to be displayed is made through use of a toggle switch which is located on the right hand side of the panel adjacent to the third row of indicator and which is labeled F-G.

ROW 3

F REGISTER

CHECK

XV - FXV is the "F" Register exponent valid bit which is functional in arithmetic mode only.

DV - FDV is the "F" Register data valid bit which indicates validity for either the entire word when in word mode or for the mantissa when in arithmetic mode.

R1,0 - FR1,0 is the "F" Register residue.

Bits 47-0 - F47-0 is the "F" Register which is one of the inputs to the adder and also feeds the S bus. This register contains the "B" operand when the processor is in Single Instruct.

ROW 23

G REGISTER

CHECK

XV - GXV is not used.

DV - GDV is the "G" Register data valid bit.

R1,0 - GR1,0 is the "G" Register residue.

Bits 47-0 - G47-0 is the "G" Register which is the third input to the adder.

ROW 4

A CONTROL REGISTER - Contains control bits for the "A" operand.

ME - AME indicates an "A" word memory error. The memory control word is returned rather than the expected data.

Bits 50-45 - A50-45 correspond to bits 50-45 in an operand.

NT - ANT indicates that "A" is an integer and is true if bits 45-39 in the "A" operand are zero.

NC - ANC indicates that the "A" operand is in the "C" Register. The contents of "C" thus determine the setting of ANC.

NV - ANV indicates that the "A" Control Register has not yet been completely updated.

EU WRITE REGISTER EXTRAS (EW)

CHECK

PV - EWPV is EWR parity valid.

DV - EWDV is EWR data valid.

ER - EWER is EWR word parity error. A memory control word has been substituted for the expected data.

R1,0 - EWR1,0 is the EWR residue.

KL WRITE RESIDUE

K1,0 - "K" Register write residue.

L1,0 - "L" Register write residue.

C REGISTER EXTRAS

CHECK

PV - CPV is the "C" Register parity valid bit.

DV - CDV is the "C" Register data valid bit.

ER - CER is the "C" Register memory error bit. A

ER - CER is the "C" Register memory error bit. A memory control word has been substituted for the expected data.

R1,0 - CR1,0 is the "C" Register residue.

EXPONENT

XV - CXV is the "C" Register exponent valid bit.

XR1,0 - CXR1,0 is the "C" Register exponent residue.

OVERFLOW (M44-39) -- CM44-39 are the "C" Register mantissa overflow bits which are an extension for the arithmetics.

E REGISTER EXTRAS

EXPONENT (XR1,0) - EXR1,0 is the "E" Register exponent residue.

OVERFLOW (M41-39) - EM41-39 is the "E" Register mantissa overflow.

F REGISTER EXTRAS

EXPONENT (XF1,0) - FXF1,0 is the "F" Register exponent residue.

OVERFLOW (041-39) - F041-39 is the "F" Register mantissa overflow.

R REGISTER

R1,0 - RR1,0 is the "R" Register residue.

Bits 05-0, R05-0 is the "R" Register which is used as both a repeat counter and general accumulator.

ROW 5 - ROW 25 DISPLAY SELECTION

The fifth row of indicators is used to display the contents of either row 5 or row 25. The selection of the row to be displayed is made through use of a toggle switch which is located on the right hand side of the panel adjacent to the fifth row of indicators and which is labeled B, K, L, AP, BP - W, SH, AL, RP, WP.

ROW 5

B CONTROL REGISTER - Contains control bits for the "B" operand.

ME - BME indicates a "B" word memory error. The memory control word is returned rather than the expected data.

Bits 50-45 - B50-45 corresponds to bits 50-45 in an operand.

NT - BNT indicates that "B" is an integer and is true if bits 45-39 in the "B" operand are zero.

NC - BNC indicates that the "B" operand is in the "C" Register. The contents of "C" then determine the setting of BNT.

K REGISTER

R1,0 - KR1,0 is the "K" Register residue.

Bits 7-0 - K7-0 is the "K" Register which is the output of the "K" Queue. The "K" Queue contains the variant for normal operators and the starting bit number within a word for single word string operations.

L REGISTER

R1,0 – LR1,0 is the "L" Register residue.

Bits 7-0 – L7-0 is the "L" Register which is the output of the "L" Queue. The "L" Queue contains the variant for normal operators and the length for string ops on single word operations.

A POINTER (AP) – Used with "A" Local storage and will point at the next character to be used in the "A" operand local storage area.

QS – APQS is the "A" pointer output select.

R1,0 – APR1,0 is the "A" pointer residue.

Bits 6-0 – AP6-0 is the "A" operand shift and allow data to allow the pointed data to be obtained.

B POINTER (BP) – Used with "B" local storage and will point at the next character to be used in the "B" operand local storage area.

QS – BPQS is the "B" pointer output select.

R1,0 – BPR1,0 is the "B" pointer residue.

Bits 6-0 – BP06-0 is the "B" operand shift and allow data pointed to be obtained.

ROW 25

W CONTROL REGISTER – General use register.

ME – WME indicates a "W" word memory error. The memory control word is returned rather than the expected data.

Bits 50-45 – W50-45 corresponds to bits 50-45 in an operand.

NT – WNT indicates that "W" is an integer and is true if bits 45-39 in the "W" operand are zero.

NC – WNC indicates that the "W" operand is in the "C" Register. The contents of "C" then determine the setting of WNT.

SHIFT REGISTER (SH) – Shift factors for the EU Barrel in octades and bits.

R1,0 – SHR1,0 is the Shift Register residue.

V – SHV is shift valid.

Bits 5-0 – SH5-0 is the Shift Register.

ALLOW REGISTER (AL) – Allows "n" bits out of the EU Barrel.

R1,0 – ALR1,0 is the Allow Register residue.

V – ALV is Allow Register valid.

LE – ALLE is allow left.

Bits 5-0 – AL05-0 is the Allow Register.

R POINTER (RP) – Used with "R" Local storage which holds the local destination information for string ops and is used for miscellaneous arithmetic partial results.

QS RPQS is the "R" pointer output select.

R1,0 – RPR1,0 is the "R" pointer residue.

Bits 6-0 – RP06-0 is the "R" storage shift and allow data to allow the indicated character to be obtained.

W POINTER (WP) – Used with "W" local storage which holds the local source information for string ops and is used for miscellaneous arithmetic functions.

QS – WPQS is the "W" pointer output select.

R1,0 – WPR1,0 is the "W" pointer residue.

Bits 6-0 – WP06-0 is the "W" storage shift and allow data to allow the indicated character to be obtained.

EU CONTROL SECTION

ROW 6 - ROW 26 DISPLAY SELECTION

The sixth row of indicators is used to display the contents of either row 6 or row 26. The selection of the row to be displayed is made through use of a toggle switch which is located on the right hand side of the panel adjacent to the sixth row of indicators and which is labeled ARITH, WORD-CW, STRING.

ROW 6

ARITHMETIC FAMILY

OPERATOR (AQ)

Bits 5-0 – AQ5-0 is the Arithmetic Family Operator Register which contains the internal micro-op from either the EU Operator Queue, or if that is empty, directly from the OW Register. The contents of the Operator Register will cause the generation of certain commands or routines to be done to complete a required task.

A - AQA is the allow bit for the Arithmetic Family. The Operator Registers of all families are loaded simultaneously, the allow bit indicates when its associated operator register is active.

TIME (AT3-0) - The Arithmetic Family operator Timer.

COMMAND (ARC)

AQA1 - Duplicate of AQA for loading purposes.

Bits 6-0 - ARC6-0 is the Arithmetic Family Command Register which contains the code of a command to perform a simple function (e.g., load a register).

RESIDUE CONTROLS

ABC2-0 - A type of command register for the Adder and Barrel for residue correction.

BOC3-0 - Barrel Residue correction.

WORD FAMILY

OPERATOR (WQ)

P5-0 - WQP5-0 is the Word Family Operator Register which contains the internal micro-op from either the EU Operator Queue, or if that is empty, directly from the OW Register. The contents of the Operator Register will cause the generation of a certain command or routine.

A1,0 - WQA1,0 is the allow bit for the Word Family. The Operator Registers of all families are loaded simultaneously. The allow bit indicates when its associated operator register is active.

TIME (WT2-0) - The Word Family operator Timer.

COMMAND (WC6-0)

The Word Family Command Register which contains the code of a command. STORE
Bits 5-0 - ST5-0 is the Store Command Register. It contains source and destination codes to enable data and address transfers outside of the EU.

STORE

Bits 5-0 - ST5-0 is the Store Command Register. It contains source and destination codes to enable data and address transfers outside of the EU.
ZER - The "R" Register is zero.

ROW 26

CONTROL WORD FAMILY

OPERATOR (COP)

Bits 5-0 - COP5-0 is the Control Word Family Operator Register which contains the internal micro-op from either the EU Operator Queue, or if that is empty, directly from the OW Register. The contents of the Operator Register will cause the generation of a certain command or routine.

A - COPA is the Allow bit for the Control Word Family. The Operator Registers of all families are loaded simultaneously. The allow bit indicates when its associated operator register is active.

TIME (OT3-0) - The Control Word Family operator Timer.

COMMAND (CWC7-0)

The Control Word Family Command Register which contains the code of a command.

RESIDUE CONTROLS

EWF2-0 - E, EWR, or F residue combined with Barrel residue. A code for the Specific type of combination.

BLUE2-0 Barrel or Logic Unit Residue Correction.

STRING FAMILY

OPERATOR (SQ)

P5-0 - SQP5-0 is the String Family Operator Register which contains the internal micro-op from either the EU operator Queue, or if that is empty, directly from the OW Register. The contents of the Operator Register will cause the generation of commands or routines.

A1,0 - SQA1,0 are the Allow bits for the String Family. The Operator Registers of all families are loaded simultaneously. The allow bit indicates when its associated operator is active.

TIME (SQT2-0) - The string Family operator timer.

COMMAND (SCM6-0)

The String Family Command Register which contains the code of a command.

T BACK-UP

TBM - TBM5-0 is the T Back Up which is a coded source for additional data for the T Buss.

TXN - T Back-up Register is not valid for an exponent.

ROW 7

COMMON CONTROLS

OPERATOR

AFL - "A" full (at the start of the current EU operator).

BFL - "B" full (at the start of the current EU operator).

QEN - Last EU operator of any sequence sent by the PCU.

QRP - Send report back to the PCU on the last command of the current operation (when the report bit is set).

RM1-0 - Remember the current operator family and indicates which family is doing the operation.

RMR - Remember return to the Control Word Family. This is used to return when a micro-op calls a micro-op of a different family.

RMW - Remember return to the Word Family. This is used to return when a micro-op of a different family.

RDB - Do a REDB (read "B") operator at the completion of the current operator.

LOK - Lock K and L.

ROUTINE

CMV - Command variant to remember extra operand.

RNP - Routine end of program operator (follows QEN).

RRP - Routine send report (follows QRP)

RNE - Routine end of EU operator. This indicates the final routine of a micro-op.

ROL - Routine operator load which indicates that the EU has generated a micro-op. The operator registers are loaded as a result of the command register contents.

RAP - Routine advance of the PIR Queue read pointer.

ENDR - End of routine.

SGS - Single routine STOP which reflects the status of the Single Routine Stop button.

STORE

SNP - Store Level end of program operator. The result is in the C Register. This flip-flop is set after RNP is set.

SNE - Store Level, end of EU operation.

SAP - Store Level, advance. PIR Queue read pointer (from setting of RAP).

ALLOCATION

ABA - The current A/B allocation in EU local storage.

AB1 - The initial A/B allocation in EU local storage.

OCA - The current read allocation for the first group of EU local storage.

ONA - The initial read allocation for the first group of EU local storage.

1CA - The current read allocation for the second group of EU local storage.

1NA - The initial read allocation for the second group of EU local storage.

T AND S VALID

TV - The T Buss was valid during the previous time.

T1V - T initial valid. The T Buss will be valid at the end of the program operator.

SV - The S Buss was valid during the previous time.

S1V - S initial valid. The S Buss will be valid at the end of the program operator.

PIR Q READ (PQR2-0) - The PIR Queue read pointer. The PIR Queue associates a program operator PIR address with a micro operator for interrupt purposes.

STACK COUNT

NSC - NSC2-0 is the New Stack Count and is used to count the pushes or pops done by a program operator. The NSC is added to STC to arrive at a new stack arrangement at the completion of the program operator.

STC - STC4-0 is the Stack Count which is used to re-establish the stack after an interrupt.

STATE FLIP-FLOPS

CURRENT

EFF - Equality flip-flop.

EXT - Mantissa sign of the "A" operand.

OFF - Overflow.

TFF - True/false flip-flop.

FLT - Float mode-used in edit operations.

TFO - True/false flip-flop occupied (valid).

INITIAL

PST - Processor state-normal mode.

DSF - Different segment-DSF:1=D1 relative, DSF:0=D0 relative.

TF1 - True/false initial, the initial setting of TFF.

FT1 - Float mode initial.

TON - True/false occupied initial.

ROW 8

ROUTINE ALGORITHM FLIP-FLOPS

TIME SHARED (VR) - VR5-0 contain the EU routine variants.

DET - Descriptor size transfer.

WORD FAM (WV) - WV2-0 contain the Word Family variants.

EU OPERATOR ALGORITHM FLIP-FLOPS

TIME SHARED (EQ) - EQ7-0 contain the EU program operator variants.

COMMON

SLO - The store level for the last operator from the EU Operator Queue.

RRR - Remember to return at the end of a routine rather than exiting. This is used when one routine has called another routine.

FMC - Family "C" operator (first hex digit is "A").

CONTROL WORD FAMILY

CWO,1 - Control word remember controls.

CMR - CW Command register valid.

DCQ - Disable CLQ (Clear Q).

QC4 - Remember SIRW.

RDN - Remember different stack number.

RFK - Remember to fetch stack.

RSG - Remember segmented description.

RVC - Remember VALC.

ARITHMETIC FAMILY

ANY - Any 1 in truncated segment. Is there a 1 in E Truncate which is an extension of the "E" Register for arithmetic purposes.

COF - Carry out.

GXF - Greater exponent.

RLM - Right to left mode.

TRF - Truncate.

ZRO - Remember zero.

ZRA - The "A" operand is zero.

ZRB - The "B" operand is zero.

ZRR - The result is zero.

PROGRAM OPERATOR ALGORITHM FLIP-FLOPS

TIME-SHARED (PQ) - PQ7-0 contain the program operator variants.

OTHERS

RFD - Remember flashback for data.

RNC - Remember concatenated NAMC.

SNO - Special action if integer overflow.

SOP - Source is an operand.

ROW 9

READ EU STORAGE

RES

V - RESV is EU read valid.

O - RESO is read within first 4 words of EU storage.

1 - RES1 is read within second 4 words of EU storage.

R - RESR is read within third 4 words of EU storage.

W - RESW is read within fourth 4 words of EU storage.

REW - REW1,0 contains the EU read address within four word groups.

PARITY CONTROL

PLA - PLA2-0 is parity look ahead, a code indicating what type of parity to check.

PCH - Parity check.

STF - Store first time.

PARITY DATA

PARITY IN EU STORAGE

POO-13 - Parity bits for EU local storage in A and B areas.

PWO,1 - Time shared "W" Register parity.

PR - Time shared "R" Register parity.

ACCUM INPUT PARITY

PAN - Accumulate indicated parity for "A".

PBN - Accumulate indicated parity for "B".

PRN - Accumulate indicated parity for "R".

PWN - Accumulate indicated parity for "W".

EXP AND TAG

PAX - "A" operand exponent indicated parity.

PBX - "B" operand exponent indicated parity.

PTA - Tag indicated parity.

X BUS CONTROLS

MAX - Main adder Transfer to the X Buss.

RDJ - Read to "J" storage (auxiliary storage location). When reset, read to "H" storage.

WTH - Write to "H" storage (auxiliary storage location).

WTJ - Write to "J" storage (auxiliary storage location).

FBC - F contains bad C register contents.

LENGTH REGISTERS

RL - RL1,0 contains the "R" storage length in EU Local storage (in number of words).

WL - WL1,0 contains the "W" storage length in EU Local storage (in number of words).

WRITE COUNTERS

RWR - RW1,0 contains the "R" storage write counter.

WWR - The "W" storage write counter.

WRITE EU STORAGE

WEAL - Write allow for EU local storage.

WEW - WEW3-0 contains the write address for EU local storage.

ROW 10

Q REGISTER (QR30-0) - The Trial Quotent Register used in division.

E TRUNCATE (ET3-1) - The "E" Register Truncate bits.

BLS (BLS1,0) - Contain the binary shift left factor for division.

REMAINDER REGISTER (RM)

V - RMV is remainder register invalid.

Bits 40-35 - RM40-35 is the remainder register.

ADDER CONTROLS

EAC - End around carry.

PL1 - Plus 1.

PL2 - Plus 2.

SPL1 - Special plus 1, used when a "double carry" occurs.

DECODE CONTROLS

MP1 - Multiplier plus 1.

M1 - Mode 1 for the scale right operator.

MISC ARITH DATA

DM39 - The "D" Register mantissa bit 39 indicates overflow from the mantissa into the exponent.

EFH - "E" and "F" Registers exponent hold causes the valid bit of the "E" and "F" exponent register's not to be reset.

If this command was not issued, the data valid bit would normally be automatically reset in the next clock eventually resulting in a continuity error.

ESS - EU Local storage Transfer to the "S" Buss (The normal Transfer is to the "T" Buss).

NZG - Zeros were not loaded to the "G" Register.

T EXP (TBX1,0) - The "T" Buss back up exponent used for exponent overflow during arithmetic operations.

C REGISTER STATUS

ZAP - The number of leading zeros in the "C" Register have been counted, and the result has been placed in the "A" pointer.

ZBP - The number of leading zeros in the "C" Register have been counted, and the result has been placed in the "B" pointer.

SIC - The contents of the "S" Buss have been transferred to the "C" Register. (The normal transfer is from the "T" Buss to the "C" Register).

SPECIAL RESIDUE

SR - SR1,0 is the special residue for the "C" Register mantissa (see CSR).

BSR - BSR1,0 is residue for current B word in local storage.

SEG COUNTER (SC2-0) - A segment counter used to keep track of words, used in conjunction with the pointers.

POINTER UPDATE REGISTERS

PVL - Pointer update register valid.

PUD - PUD6-4 is a word update for the EU local storage pointers.

PUD3-0 is a digit update for the EU local storage pointers.

MODES

PSUB-Subtract pointer update register.

2WP - Two word mode for pointer update.

4BM - Four bit mode for EU local storage pointers (normally information is dealt with in three bit digits).

ROW 11

OPERATOR Q READ

OQO - OQR2-0 is the operator queue read register.

DATA Q CONTROLS

DQW - DQW1,0 is the data queue write register.

WLQ - Write into the least significant word of the double precision data queue entry.

DQR - DQR1,0 is the data queue read register.

RLQ - Read from the least significant word of the double precision data queue entry.

VALID (DQV3-0). The data queue valid register is used to allow a inhibit reads and writes. Each double precision data queue entry is represented by a bit in DQV. The bit must be true to enable a read and false to enable a write.

SINGLE PREC (QSP3-0) - Used to indicate single precision operand in each of the data queue entries.

DQA - Data queue read pointer advance.

EU ENABLE (EUE2-1)

Enables the clock for the EU.

CONTROL FOR COMMUNICATION WITH OTHER UNITS

CSH - Conditional stack hold. Enables hold for stack buffer if necessary.

HSQ - Hold for Storage unit data queue available.

SDQ - Store to storage unit data queue.

ERD - EU request for address unit read.

LDD - EU request for address unit quick write.

EWP - Enable address unit write pointer.

AHE - Address unit hold and conditional EU hold for EWR.

EHE - EU hold for EWR data valid.

AHS - Address unit transfers to MAR.

CSO - "C" Register transfer to MAR.

ADA - Address of last EU fetch in AU.

ERW - Unconditional request for EWR.

ERA - Address unit controlled by EU.

ERS - EU request for storage unit.

EHS - EU hold for storage unit available.

AUH - Address Unit hold.

DLA - Disable load associative memory.

1WM - One word mode for string fetching.

2WM - Two word mode for string fetching.

SOURCE PROTECT AND SIZE

SPR - Source memory protect.

SS2-0 - The source size register which contains the character size.

DESCRIPTOR CONTROL/DESTIN. PROTECT AND SIZE

ENR - Edit, not return from interrupt.

DC44 - "D" Control Register bit 44 which indicates that data is segmented.

DPR - Destination protect.

DS2-0 - Destination Size Register which contains the character size.

ROW 12

OPERATOR DELAY REGISTER (OD)

Bits 4-0 - OD4-0 contain the Operator Delay Register which holds the code for an interrupt while the existing command is finished. The delay is for one clock.

R - ODR is the operator delay restore bit which enables restart of a program operator after an interrupt. If ODR is true, then A/B Initial will be left as is, however, if ODR is false then A/B Initial is moved to A/B Current.

PU REACT (PU1,0)

The equivalent of an interrupt caused by a series of micro-ops. This mechanism allows variation of a micro-op.

EU REACT

EUQ - EUQ1,0 is the Eu operator react which is the equivalent of an interrupt caused by a series of micro-ops. This mechanism allows variation of a micro-op.

EUC - EUC1,0 is the EU command react which is similar to EUQ except it allows variation of routines.

BRANCH TYPE

BRT

BITS 2-0 - BRT2-0 contain the branch (interrupt) type.

R - BRTR is branch type restore.

EUIT

EU in Trouble. This mechanism is used for synchronization between the EU and PCU and is evoked when a change in direction occurs and it becomes necessary to call a hardware subroutine.

BTH

Branch type hold which holds the EU while the EU operator queue is invalidated.

RESIDUE CONTROLS

4BG - Four bit residue generator mode for string operators only.

WMO - The word residue mode command which indicates the type of residue.

WMS - Word residue check mode.

SRS - Suppress residue check on sign.

RESTART STATUS

SOS - Save old State.

SOK - State OK.

NRS - No restart. If set on STOP ON ERROR, the micro-op cannot be repeated.

FAILURE FLIP-FLOPS

XPL - Extra pointer load which indicates that an attempt was made to load a pointer while the existing contents were valid. The existing pointer information was thus not shifted somewhere else before the attempted move, or an incorrect hold data valid was issued, or an incorrect hold data valid was issued.

ECF - EU continuity failure.

ERF - EU residue failure.

EPF - EU parity failure.

SMF - String memory failure.

INTER VARIANTS

CRL - Created Length.

LP2 - PCU load P2 if interrupt.

P2F - P2 contains data.

QC2 - Inhibit normal interrupt.

MISCELLANEOUS EU

CKC – Check "C" Register continuity.

FLAG – Remember illegal shift, start, or length.

XGS – Exponent difference greater than selected leading zeros which indicates that the exponent was too large to use only the exponent shift.

SGM – Single routine mode.

EMR – EU force memory reference.

WAL – "W" Register allocation.

DSN – DSF (SD1 relative to D1) next.

$\overline{\text{II}}$H – Inhibit external interrupts (DEXI).

RXO – Remember exponent overflow.

EUA-EU Abort.

OPERAND IDENTITY

COMMD (CQ1,0) is the command operand which acts as a variant on the command.

STORE (SQ1,0) is the "check" operand which acts as a variant on the store.

ROW 13

PROCESSOR MISCELLANEOUS

MSK – Master synchronization flag to keep 8 and 16 MHz cooperative.

ZZZ – Used with MDU COMPARE OR and MDU COMPARE AND switches. When the compare function is satisfied, ZZZ is set to stop the 8 and 16 MHz clocks.

16M – 16 MHz clock.

08M – 8 MHz clock.

DLT – Delete T bus.

RSN – Restart next operator.

SDW – Store to "D" Register word.

LD1 – Last D1 valid in AU location 3B.

39A – 39-bit adder mode.

SPO – String program operator.

CLE – Clear EU controls.

RIN – Repeat instruction.

RCL – Remember CLQ (clear Q) in EU.

RQF – Remember quick fetch (for string ops).

FAS – Fast mode (for long string ops).

FAL – Fail to stay in Fast Mode.

FLS – Failsoft (protected write operation in memory; bit 48 —1).

REMEMBER SUSPEND REGISTER (RS)

Holds the control word if a multiword fetch by the PCU for the program buffer is interrupted. The PCU has the lowest priority with the COMM. Unit.

LENGTH (RS24-22) – The number of words remaining to be read.

Comm Address – (RS21-2) – The main memory address.

RES (RS01,0) – Comm address residue.

ROW 14 - ROW 34 DISPLAY SELECTION

The fourteenth row of indicators is used to display the contents of either row 14 or row 34. The selection of the row to be displayed is made through use of a toggle switch which is located on the right hand side of the panel adjacent to the fourteenth row of indicators and which is labeled FC, P2, CM – OVF, FM, FR.

ROW 14

P1 SPECIAL

RT – Return bit for presence bit P1 parameter.

R45 – Remember RETN bit for presence bit P1.

V39 – Remember VALC bit for presence bit P1.

EGG/HELP – 8 to 16 second timer; used to get out of software loop.

FAULT CONDITION REGISTER

SKOF – Stack overflow.

NTVT – Interval timer.

PGOP – Programmed operator.

SEGA – Segmented array.

SEQ – Sequence error.

PBIT - Presence bit.

BTSK - Bottom of stack.

NTOF - Integer overflow.

NVNX - Invalid index.

XUNF - Exponent underflow.

XOVF - Exponent overflow.

DVBO - Divide by zero.

NVOP - Invalid operant.

MPRO - Memory protect.

PINT - Processor internal.

INVP - Invalid program word.

SKUF - Stack underflow.

NAM - Invalid address (no access to memory).

FAL1 - Memory fail 1 (2 or more bits in error).

MPAR - Memory parity.

LOOP - Loop.

P2 REGISTER (P22-20)
P2 parameter register for interrupts.

CONTROL MODES (CM3-0)
Interrupt management level of the processor.

ROW 34

FAULT MASK REGISTER - FAIL REGISTER (FM)

BITS 42-23 - FM42-23 is the Fault Mask Register which enables or disables recognition of the special (third Priority) and external (fourth priority) interrupts.

INR - Inhibit normal return.

MES - Indicates that incorrect data exists for restart operation. (Data cannot be recovered.)

EUC - EU continuity.

EUR - EU residue.

EUP - EU parity.

PER - Program Unit error.

ADD - Adder unit residue.

WCN - Wrong channel number.

INP - Parity to the Comm. Unit.

CRS - Comm. Unit residue.

CSE - Comm. single error.

MTO - Memory time out.

F2 - One bit error.

SU - Storage Unit.

SK - Stack.

OP - Operation.

MADS5-0 - Memory address for interrupts.

BN3-0 - Box number for interrupts.

ROW 15

TIME OF DAY CLOCK (TD35-0)
The time of day in 2-microsecond intervals.

LOOP TIMER (LT11-0)
A 2-second timer in the processor that is retriggered with the completion of each program operator based in Final Command Final Routine.

TIMER CONTROL
IC1 - Loop timer overflow control.

LOP - LOOP timer overflow.

ITZ - Incremental timer zero.

ARM - Incremental timer overflow.

PANEL 2 SWITCHES (LEFT-HAND PANEL)

MDU COMP CONT

HPIR
This switch with MDU COMPARE AND causes a halt if P1R of an instruction is equal to the value selected in MDU toggle switches.

HSI/HSP
HSI allows the processor to halt on single instruction; HSP allows the processor to stop on a single clock. (This switch is used in conjunction with MDU COMPARE AND or MDU COMPARE OR.)

PROGRAM

PRB

Used in manually reading and writing the program buffer.

EU STORAGE

LOC

Used in manually reading and writing the EU Local Storage.

STACK

STK

Used in manually reading or writing the stack buffer.

PRG

Used with PLS switch to purge the stack buffer. This operation also invalidates the four-word ASM, if the 4AM switch is in the up position.

ASSOCIATIVE MEMORY

1AM

Used in manually reading and writing the one-word Associative Memory data.

4AS

Used in manually reading and writing the four-word Associative Memory data.

ASM

Used with 4AM and PLS switches to advance MAR count by four.

1PL

Used with PLS switch to advance Priority List count.

DISPLAYS

DSP

Used in manually reading and writing the Display Buffer.

BUFFER CONTROLS

R/W

Used with the other storage controls to designate a Read (false) or Write (true) operation.

PLS

A momentary contact switch used for pulsing in conjunction with the other local buffer controls to actually transfer in or out of the local storage.

SINGLE PULSE

ON-OFF

This indicator/switch enables or disables the Single Pulse junction. If this switch is on, the Special Function switch must also be on.

PUSH BUTTON

Used for single pulsing. When this switch is pressed, one 8 MHz and two 16 MHz clocks are generated.

MDU COMPARE OR

This causes a halt if any bit within a combination of bits in the Processor is equal to the value selected in MDU toggle switches.

MDU COMPARE AND

This causes a halt if a combination of bits in the Processor is equal to the value selected in MDU toggle switches.

MDU ENABLE

Enables communication with the MDU.

INHIBIT ERROR

Prevents execution errors from stopping unit.

ERROR STOP

Stops unit on internal hardware errors. If an internal interrupt occurs, bit 6, row 14 is set.

REPEAT INST

Causes loop on instruction that caused unit to stop on error STOPj. Must be pressed after unit has stopped on error.

SINGLE ROUTINE

Causes execution of a single routine within an instruction. (Execution of single EU operator).

16/8 MHZ-16 MHZ

In position 16/8 MHZ, an 8-MHz and two 16-MHz clocks are generated when panel 1 or panel 2 push button is pressed. In position 16 MHZ, one 16-MHz clock is generated when panel 1 or panel 2 push button is pressed.

CONTROL CLEAR

General processor clear.

START

Starts processor. This switch is also used to continue from some types of halt conditions.

PANEL 1 INDICATORS (RIGHT PANEL)

COMMUNICATIONS UNIT

ROW 1 - ROW 21 DISPLAY SELECTION

The first row of indicators is used to display the contents of either row 1 or 21. The selection of the row to be displayed is made through use of a toggle switch which is located on the right hand side of the panel adjacent to the first row of indicators and which is labeled IN-OP.

ROW 1

INPUT REGISTER (IN51-0)
The input register to the Comm Unit which is used for data transfer between the processor and memory.

ROW 21

OUTPUT REGISTER (OP51-0)

The output register from the Comm. Unit which is used for data transfer between memory and the processor.

ROW 2

CHAN SEL REG (CSR4-0)
Contains the number of the physical MCM selected.

COMM TIMER (CT6-0)
Timer for Comm Unit operations.

SU OPERATIONAL CONTROLS
SA1,0 - Start address.

CRQ - Comm request data queue.

CRA - Comm request Associative Memory.

RDP - Remember double precision.

CHT - Comm halt.

ICH - Inhibit comm halt.

EQE - EWR enable.

ERRORS
IVC - Invalid channel.

INE - Internal error.

IPE - Input register parity error.

NAM - No access to memory.

ERR - Comm error.

CLN5
The most significant bit of the comm length (CLN) located in row 3, bits 26-22.

SU CANCEL LENGTH (24-22)
Used with CLN to determine last memory fetch to EWR.

INTERNAL REQUESTOR INTERFACE
IGP - Ignore data parity.

RFR - Request processor fail register.

KUG - Stack granted.

KUL - Stack unit last word.

SUG - Storage unit granted.

PUG - Program unit granted.

RSP - Remember suspended operation.

PUL - Program unit last word.

SPEC CONTROLS
R1W - Remember one word.

ETS - Error transfer sync.

MDS - Maintenance diagnostic.

INCREMENTAL TIMER (IT10-0)
The interval timer.

ROW 3

REQUESTOR INTERFACE
Internal processor requestors to the Comm Unit.

MCM INTERFACE
RS - Request special
REQ - Request
RQS - Request strobe
ACK - Acknowledge
SND - Send data
DST - Data strobe
DA - Data available
DAP - Data present
RQC - Requestor operation complete
SR - Send/Receive
FL1 - Fail one
FL2 - Fail two

SU VARIANTS
EWR - Write into the EU Write Register
QL1,0 - EU Data Queue location
CA4 - Comm load of 4 word ASM
CA1 - Comm load of 1 word ASM
LFD - Look for double precision

OP CODE
RW - Read/Write operation
TYP - Type
SW - Single word
PRT - Protect
FB - Flashback
R1L - Requestor inhibit load
MLL - Memory limit load

COMM LENGTH

CLN5 (ROW 3 BIT 31) and CLN4-0 contain the length of the operation in words.

COMM ADDRESS (CA19-0)

The absolute memory address.

STORAGE AND STACK UNITS

ROW 4 - ROW 24 DISPLAY SELECTION

The fourth row of indicators is used to display the contents of either row 4 or 24. The selection of the row to be displayed is made through use of a toggle switch which is located on the right hand side of the panel adjacent to the fourth row of indicators and which is labeled SU CONTROL MA - 1AM/S1.

ROW 4

SU CONTROLS

SEN - Storage unit enable
SMR - Storage unit memory reference
SBY - Storage unit bypass queue
SAB - Storage unit abort
SNF - Storage unit not first word
SBE - Storage unit busy with Execution Unit
SEC - Storage unit end conditionally
SP4-1 - Storage unit timing

VARIANTS

SEW - Request EWR
SQL1,0 - Location in the EU data queue, passed from the PCU.
S4A - Operation for 4 word ASM
S1A - Operation for 1 word ASM
SLD - Look for double precision word

OP CODE

SRW - Read/Write (reset for fetch and fetch memory fail)
STY - Type (set for memory fail and multiple word store)
SPT - Protect (set for single word store and multiple word store)
SFB - Flashback (set for single word store and read with lock)
SIL - Requestor inhibit load - Memory limit load
SML - Memory limit load
SLN2-0 - Number of words to be transferred

SU MEMORY ADDRESS (MA21-0)

This register receives the absolute memory address from the SU operation queue, AU adder output, or the C register in the EU. (MA21, 20 are residue bits.)

ROW 24

1AM ADDRESS (AD, OEU)

AD-AD15-8 is Associative Memory addressing for local checks.
OEU - Oldest entry upper
U02-22 - Upper pointers for loading information into Associative Memory
CBU - Comm busy with upper portion of Associative Memory
VLU - The upper Associative Memory is valid
LCU - Local in upper
CBL - Comm busy with lower portion of Associative Memory
VLL - The lower Associative Memory is valid
LCL - Local in lower

1AM ADDRESS (AD, OEL)

AD-AD07-0 is Associative Memory addressing for local checks.

OEL - Oldest entry lower

L02-00 - Lower pointers for loading information into Associative Memory.

SU INPUT ADDRESS (S21-0) - This register receives the absolute memory address from the AU adder output. (S21, S20 are residue bits.)

ROW 5

SU DATA Q (SD)

W1-W0 - Write pointer
R1-R0 - Read pointer

SU MISC

QNA - Storage unit data queue
KLC - Stack local check
RAS - Remember above S
SAE - Send address to EWR
SHC - Storage hold for comm
CBS - Comm busy with Storage Unit
SDPO - Storage data queue parity location 0
SWL - Storage operation write last

4AM CONTROL

4AA1 - 4 word Associative Memory address
L4A - Local in 4 word ASM
CHA - Comm has ASM

4AM PRIORITY LIST (PL5-0)

Contains 1 of the 64 words in the priority array. This word contains three two bit codes which represent the three oldest addresses within a selected address block (64 address blocks).

STACK LENGTH REGISTER (KLN5-0)

Contains the stack buffer length currently in use.

S-REGISTER (SR21-0)

Contains the main memory address of the top of stack. (SR21, 20 are the S register residue.)

ROW 6

STACK VULNERABILITY REGISTER (KV19-0)

Contains address of S-4 location in the stack buffer.

STACK CONTROLS

KP2-1 - Stack Unit timing
KC1-0 - 0 Stack command
KBC - Stack busy with comm
SBK2-0 - Storage Unit busy with stack

STACK LINK REGISTER (KL21-0)

Contains the address for the bottom (or oldest) item in the stack buffer. (K21, 20 are the KL register residue.)

ROW 7

LIMIT OF STACK REGISTER (LOSR21-0)

Contains the main memory address of the maximum limit of the stack's assigned area.

STACK MISC

TKV3-0 - Top of stack vulnerability
MCP - Memory copies present
TKR - Top of stack read
TKW - Top of stack write
KWT - Stack write

F-REGISTER (FR21-0)

Contains the address of the current Mark Stack Control Word. (FR21, 20 are the F register residue.)

ADDRESS UNIT

ROW 8

ADDRESS UNIT CONTROL

DTO - Address adder timing

BYE - E.U. is busy with the address adder.

BYP - PCU is busy with the address adder.
- Read cycle

ACY - Add cycle

EXC - Extra cycle

WCY - Write cycle

SAC - Select address couple which is used with a name call or value call.

AD1 - Add one

ADW - Adder to Display Write Register.

CDW - Complement the Display Write Register.

AUO - Address adder overflow.

AZF - Address adder output's all zeros.

EU READ PT (ERP5-0)

The address adder read pointer for the EU which allows the EU to access the addressable registers.

PCU READ PT (PRP5-0)

The address adder read pointer for the Program Unit which allows the PCU to access the addressable registers.

WRITE PT (DWP5-0)

The address adder write pointer. Only the EU can write into the addressable registers.

SU INPUT OPERATION

1AP - Input address present

1WT - Input wait

1MR - Input memory reference

IQL1,0 - The queue location in the EU data queue

ILD - Input look for double precision

IL2 - Input length two

SQ OPERATION QUEUE (SQ)

SQR2-0 - Storage Operator Queue Read pointers.

SQW2-0 - Storage Operator Queue Write pointers.

PROGRAM CONTROL UNIT

ROW 9 - ROW 29 DISPLAY SELECTION

The ninth row of indicators is used to display the contents of either row 9 or row 29. The selection of the row to be displayed is made through use of a toggle switch which is located on the right hand side of the panel adjacent to the ninth row of indicators and which is labeled PUR/PLR/RE/RO - DWR/DRR.

ROW 9

PROGRAM UPPER REGISTER (PU21-0)

Contains the main memory address of the next word to be loaded into the Program Buffer. (PU21, 20 are the Program Upper Register residue.)

PROGRAM LOWER REGISTER (PL21-0)

Contains the main memory address of the oldest active entry in the Program Buffer. (PL21, 20 are the Program Lower Register residue.)

READ EVEN (PEB3-0)

A pointer to the even word in the Program Buffer.

READ ODD (POB3-0)

A pointer to the odd word in the Program Buffer.

ROW 29

DISPLAY WRITE REGISTER (DW21-0)

Input register for all addressable registers. The DWR is one of the inputs to the address adder and is the location into which the "hard" registers are manually read. (DW21, 20 are the Display Write Register residue.)

DISPLAY READ REGISTER (DR21-0)

Output register of the addressable registers. (DR21, 20 are the Display Read Register residue.)

ROW 10

PROGRAM ADDRESS REGISTER (PA21-0)

Contains the main memory address of program code only when a change of direction occurs. The address is then transferred to the Program Upper Register only if a branch point is found to be non-local. (PA21, 20 are the Program Address Register residue.)

PROGRAM BUFFER CONTROL

PWP4-0 – Program Write Pointer.

PWE4, PWE3 – Program Write Edit Pointer.

BK2-0 – Block pointer

BR4, BR3 – Branch pointer

SPF – Stop program fetch

FNL – Force not local

TUL – Transfer program upper to program lower

IPU – Increment program upper

PROK – Program OK (for PA residue check)

CPA – Count Program Address Register

RAD8 – Remember adjust by 8

PRI – Pulse switch flip-flop

SNC – Sync

STR – Start

TIR – Table edit remember

ERC – Edit read control for Program Buffer.

EWC – Edit write control for Program Buffer.

ROW 11

EDIT CONTROL

EDT – Processor is in the table option of Edit Mode.

EDS – Processor is in the single option of Edit Mode.

EUP – Edit Mode is in update variation.

ESM – Edit single micro.

LEXIC LEVEL (LL4-0)

Contains the current Lexic Level.

LLW (LLW4-0)

The Lex Level write register is the input to the Lex Level Queue.

SAVE SYLL (SSR4-0)

The Syllable Save Register is used as interim storage on transfer from the EU to the PCU and is also used to save the normal syllable when entering Edit Mode. The Just two bits of this register (SSR4, SSR3) are the syllable residue, and the remaining three bits (SSR2-0) contain the syllable.

DATA Q

DQMT – Data Queue empty

QER – E.U. Data Queue error.

QDV – E.U. Data Queue valid.

DA1-0 – A two bit index into the EU Data Queue.

QRL – Data Queue Ready Lookout.

IQV – Inhibit EU Data Queue Valid.

PCU MISC

ISO – Inhibit stack overflow

NPR – Alternate P1R

CLQ – Clear queue

RCQ – Remember clear queue

PLE – Pipe line empty

RESIDUE

VNR1,0 - Variant residue

AR1,0 - Address couple residue

PB PARITY BITS

IEP2-0 - Parity bits for syllables located in the IER.

IDP2-0 - Parity bits for syllables located in the IDR.

RPC

Remember program count.

OEA, OEB, OEC

Odd/even flip-flops or the Program Buffer.

BARREL SELECT (BS5-0)

The shift controls for the Program Buffer.

ROW 12

ORDER CODE WRITE (OW)

OW11 (ESA) - Stack location A valid

WL10 (ESB) - Stack location B valid

OW09 (RPT) - Request report from Execution Unit

OW08 (1END) - Instruction end

OW07-00 - Micro operator code

OKLQ

OK to load operator queue.

OQWR

Operator queue write.

OW2-0

Operator queue write pointer.

V (vector) STK

RA - Remember A

RB - Remember B

HOLD CONDITIONS

CRE - Comm Unit request for use of the EU Queue Write Register (EWR).

SRE - Storage Unit request for use of the EWR.

ERE - Execution Unit request for use of the EWR.

NOTE

EWR is the input to the Data Queue in the Execution Unit.

DQF - EU Data queue full.

PIR - Hold for new PIR

MRK - EU has completed a Mark Stack operation.

EDN - EU done

PBB - Program Buffer busy with replenishment.

RPA - Reset PWA flip-flop.

PCU CONT

T1 - Timing for PCU

T01-3 - Timing for PCU

VS1 - Valid storage unit input address

LS1 - Load Storage input register

CFL - Check for program code loop

ENB - Enber/Branch

NCC - Name call concatenate

IPN - Inhibit program next register count

PCU ERRORS

PNE - P1R residue error

PPE - Program word parity error

PTG - Program tag error

NEO - Non-existent operator

REP COUNT (RPC3-1)

Extensions of the Phase Counter.

ROW 13

SUBROUTINES

EUT - An EUIT indication in the PCU set by the EU so that a hardware subroutine can be initiated.

ACT - Hard routine for re-execution of instructions for interrupts.

CKL - Length check routine in Vector Mode.

DOB - Hardware routine for execution of dynamic operand branch.

EDI - Edit interrupt subroutine.

STG - Hardware routine for string operator interrupt processing.

ENT - Hardware routine for Enter Junction during an interrupt.

RAK - React subroutine

INT - Hardware routine for interrupt handling.

LOD - Hardware routine for initialization of the Program Buffer.

ACE - accidental entry

MOV - Hardware routine to accomplish a Move Stack operation.

PAS - Hardware routine to accomplish a Pause Junction.

BY PASS LB

PMR - Program memory reference

OP CONT

ADP - D8, when set, causes issuance of a fetch-to-stack (Pop).

ADM - DA - When set, causes issurance of a push.

DGC - Disable generation of code

DEC - Disable communication to the EU

HOLD LOGIC

PRE - Program Unit requests use of the Execution Unit Write Register.

LDQ - Load Execution Unit Date Queue.

HNP - Hold for new PIR.

HFM - Hold up Program control until Execution Unit completes mark.

RPT - Hold for report from EU.

INH - Inhibit Program Buffer fetches from main memory.

PBA - Program buffer available.

PWA - Program Control Unit requests use of the address adder.

HSI - Hold single instruction.

SWG - Something wrong flip-flop. If set, then whatever phase we are in as determined by the phase counter becomes a special phase. In effect, this causes hesitation in normal operation due to an exception condition (two sequential string operators).

BGN - Begin first phase of instruction.

PHASE COUNTER (PHF-0)

Indicates phase of the program operator.

ROW 14

IE CONTROL

Instruction Execution Register (IER) Control.

IMA - Intermediate stack condition for the "A" operand.

IMB - Intermediate stack condition for the "B" operand.

DVE - Decode Variant Mode operators in the IER.

DEE - Decorde Edit Mode operators in the IER.

INSTRUCTION EXECUTE REGISTER (IER23-0)

A three syllable register which receives the program operator from the IDR and decodes it into micro-operators.

IEA

Instruction execute allow, when reset, disables the decode gates of the Instruction Execute register.

VEC - Vector Mode

PROGRAM INDEX CURRENT (PC17-0)

Contains program index value for the word currently in the IER which, when added to the PBR gives the absolute memory address. (PC17, 16 is program index current residue.)

C SYLL (CS2-0)

The syllable count for the IER.

ROW 15

ID CONTROL

SKA - Stack location A

SKB - Stack location B

DVD - Decode Variant Mode operators in the IDR.

DED - Decord Edit Mode operators in the IDR.

INSTRUCTION DECODE REGISTER (IDR23-0)

A three syllable register which is normally considered to be the "look ahead" station. The IDR is decoded for barrel shifts and counts for "fast" program operators. A combination of IDR and IER action is necessary for the "slower" operators. Decoding of the IDR also provides the initial set-up for stack maintenance and determines if an operator occurred which requires concatenation.

IDA

Instruction decode allow when reset, disables the decode gates of the Instruction Decode register.

LID

Load instruction decode register.

PROGRAM INDEX NEXT (PN17-0)

Contains index value for the word currently in the IDR, which, when added to the PBR gives the main memory address. (PN17, 16 are the Program Index Next Residue).

N SYLL (NS2-0)

The next syllable count for the IDR.

PANEL 1 SWITCHES (RIGHT HAND PANEL)

POWER

OFF - Removes power from the CPM.

ON - Applies power to the CPM.

READY - Indicates that the powering-up sequence is complete.

STATUS

ON LINE - The CPM is in on-line operation. All control switches, with the exception of power, are disabled.

TEST - The CPM is in Text operation. All control switches are enabled.

TEST

STORAGE - Sets or resets the flip-flops and indicators as specified through the toggle and thumb wheel switches.

LAMP - Sets indicators as specified through the thumb wheel switches.

FANS-FAULTS - Indicates that the CPM has sensed a fan failure, and has been automatically powered down.

PAUSE - Sets the CPM to control state, control mode 3 in a forced PAUS instruction.

CLEAR - Zeros the field selected through the Thumb Wheel switches.

PAN - Panel number.

ROW - Indicator row which is printed on the right hand margin of the panel.

GRP - Group. A normal group (1) consists of ten indicators, thus group 0 consists of bits 9-0. Group 9 specifies that action will be taken on the entire selected row.

LOAD - The contents of the toggle switches will be loaded (OR'd) into the location specified through the thumb wheel switches.

SINGLE PULSE

ON-OFF - Enables a disables single pulse action. If this switch is on, the Special Function switch on panel 1 must also be set.

PUSH BUTTON - Used for single pulsing. When this switch is depressed, one sixteen-megacycle clock is generated.

SERIAL MODE

Causes the unit to wait until the EU completes each micro-op before it is given another one. The micro-op thus is taken directly from the OW Register rather than the E.U. Operator Queue.

CONDITIONAL HALT

Causes a halt if the conditional halt operator occurs. Processing may be continued, dependent on the situation by pressing the Start switch.

HALT ON SDI

Causes a halt if Program Index current (panel 1 row 14, bits 18-03) is equal to the value selected in toggle switches 18-03, and FNL (panel 1 row 10, bit 12) is set.

HALT ON PIR

Causes a halt if Program Index Current and C Syll (panel 1 row 14 Bits 18-0) are equal to the value selected in toggle switches 18-0.

4AM DISABLE

Inhibits the use of 4 word Associative Memory.

1AM DISABLE

Inhibits the use of 1 word Associative Memory.

SINGLE INST

Allows a single program operator to be executed each time Start is depressed. This switch is also used to temporarily stop the processor.

REPEAT PROGRAM

Causes the CPM to cycle through the Program Buffer without obtaining more code from main memory.

CONTROL CLEAR

Clears all "hard" registers to the reset state.

START

Resumes execution after a Halt condition.

PANEL OPERATIONS

SETTING/RESETTING OF INDICATORS

The indicators and their associated flip-flops can be manually set or reset either from the CPM display panels or from the MDU. Selection of a row of indicators and a specific group within the row is made through four thumbwheel switches which are located on the bottom portion of the right hand panel, panel 1. The switch allows the selection of a Panel (1 or 2), Row (1-15, 21-23, 27-29, 34), and Group (0-5, 9), "Group" is a group of ten indicators numbered from 0 through 6 from right to left. A group value of nine indicates that the entire row will be acted on.

Once that the row has been selected, it can be modified through the 51 toggle switches at the bottom of each panel. The Clear button, located immediately to the left of the Thumb Wheel switches, causes zeros to be loaded into the selected row. The Load button, located immediately to the right of the Thumb Wheel switches, causes the contents of the toggle switches to be "OR"ed with the appropriate flip-flops. It is thus necessary to clear an area and to load it with new values to RESET a selected indicator.

LAMP TEST

NOTE

The Lamp Test is used to test the indicators only. The flip-flops associated with the indicators are not set.

1. STATUS SWITCH	Place in the TEST position.
2. TEST (SWITCHES)	Place the LAMP switch in the ON position.
3. THUMB WHEEL SWITCHES	Set to Panel=1, Row=N/A, Group=0
4. LOAD SWITCH	PUSH. Indicators on Panel 1 in toggle switch positions 9-0 of all rows should light.
5. THUMB WHEEL SWITCHES	Set to Panel=1, Row=N/A, Group=1
6. LOAD SWITCH	Push. Indicators on Panel 1 in toggle switch positions 19-10 of all rows should light.
7. THUMB WHEEL SWITCHES	Set to Panel=1, Row=N/A, Group=2
8. LOAD SWITCH	Push. Indicators on Panel 1 in toggle switch positions 29-20 of all rows should light.
9. THUMB WHEEL SWITCHES	Set to Panel=1, Row=N/A, Group=3
10. LOAD SWITCH	Push. Indicators on Panel 1 in toggle switch positions 39-30 of all rows should light.
11. THUMB WHEEL SWITCHES	Set to Panel=1, Row=N/A, Group=4
12. LOAD SWITCH	Push. Indicators on Panel 1 in toggle switch positions 49-40 of all rows should light.
13. THUMB WHEEL SWITCHES	Set to Panel=1, Row=N/A, Group=5
14. LOAD SWITCH	Push. Indicators on Panel 1 in toggle switch positions 51, 50 of all rows should light.

15. THUMB WHEEL SWITCHES	Set to Panel=2, Row=N/A, Group=0
16. LOAD SWITCH	Push. Indicators on Panel 2 in toggle switch positions 9-0 of all rows should light.
17. THUMB WHEEL SWITCHES	Set to Panel=2, Row=N/A, Group=1
18. LOAD SWITCH	Push. Indicators on Panel 2 in toggle switch positions 19-10 of all rows should light.
19. THUMB WHEEL SWITCHES	Set to Panel=2, Row=N/A, Group=2
20. LOAD SWITCH	Push. Indicators on Panel 2 in toggle switch positions 29-20 of all rows should light.
21. THUMB WHEEL SWITCHES	Set to Panel=2, Row=N/A, Group=3
22. LOAD SWITCH	Push. Indicators on Panel 2 in toggle switch positions 39-30 of all rows should light.
23. THUMB WHEEL SWITCHES	Set to Panel=2, Row=N/A, Group=4
24. LOAD SWITCH	Push. Indicators on Panel 2 in toggle switch positions 49-40 of all rows should light.
25. THUMB WHEEL SWITCHES	Set to Panel=2, Row=N/A, Group=5
26. LOAD SWITCH	Push. Indicators on Panel 2 in toggle switch positions 51, 50 of all rows should light.
27. TEST (SWITCHES)	Return the LAMP switch to the OFF position.

STORAGE TEST

NOTE

The Storage Test sets or resets both the indicators and their associated flip-flops. It is therefore necessary to either de-commit the CPM before the test and re-commit it following the TEST or Halt/Load following the TEST.

1. STATUS SWITCH	Place in the TEST position.
2. SINGLE PULSE (Panel 2)	Place in the ON position.
3. TEST (SWITCHES)	Place the SORAGE switch in the ON position.
4. THUMB WHEEL SWITCHES	Set to Panel=1, Row=N/A, Group=9
5. PANEL 1 CONTROL CLEAR	Push. All wired indicators on Panel 1 should be OFF (reset).
6. PANEL 1 TOGGLE SWITCHES	Place all switches in the UP (set) position.
7. LOAD SWITCH	All wired indicators on panel 1 should be ON (set).
8. PANEL 1 CONTROL CLEAR	Push. All wired indicators on panel 1 should be OFF (reset).
9. THUMB WHEEL SWITCHES	Set to Panel=2, Row=N/A, Group=9.
10. PANEL 2 CONTROL CLEAR	Push. All wired indicators on Panel 2 should be OFF (reset).
11. PANEL 2 TOGGLE SWITCHES	Place all switches in the UP (set) position.
12. LOAD SWITCH	Push. All wired indicators on Panel 2 should be ON (set).
13. PANEL 2 CONTROL CLEAR	Push. All wired indicators on Panel 2 should be OFF (reset).
14. TEST (SWITCHES)	Restore the STORAGE switch to the OFF position.
15. SINGLE PULSE (Panel 2)	Restore to the OFF position.

NOTE

The CPM must now either be re-committed or Halt/loaded.

PROGRAM BUFFER OPERATION

To perform a program write operation, proceed as follows:

1. STATUS switch (panel 1)	Place in the TEST position.
2. REPEAT PROGRAM switch (panel 1)	Place in the ON position.
3. PROGRAM (panel 2)	Place the PRB switch in the up position.
4. BUFFER CONTROLS (panel 2)	Place the R/W switch in the down position.
5. THUMB WHEEL switches	Set to Panel=1, Row=0, Group=9.
6. CLEAR switch (panel 1)	Press to clear the INPUT REGISTER (IN).
7. Panel 1 toggle switches	Set switches 47-0 in the up position (write one's) for the information to be written into the program buffer. Bits 48 and 49 (tag 3) must be set and bit 51 may be set to obtain good parity (odd).
8. LOAD switch (panel 1)	Press to load the IN register.
9. BUFFER CONTROLS (panel 2)	Toggle the PLS switch once to write into the program buffer.

NOTE

PWP (1-10-24) should be set to indicate that buffer location one is now ready to be written into. Repeat step 9 until all 32 buffer locations are written into.

To perform a program read operation, proceed as follows:

1. PROGRAM (panel 2)	Place the PRB switch in the down position.
2. CONTROL CLEAR (panel 1)	Press to clear all hard registers.
3. PROGRAM (panel 2)	Place the PRB switch in the up position.

NOTE

If indicator $\overline{\text{INH}}$ (1-13-21) is not on, perform steps 4 thru 6; otherwise, proceed with step 7. $\overline{\text{INH}}$ flip-flop must be set in order to read from the program buffer.

4. THUMB WHEEL switches	Set to Panel=1, Row=13, Group=9.
5. Panel 1 toggle switches	Set switch 21 to the up position.
6. LOAD switch (panel 1)	$\overline{\text{INH}}$ indicator on panel 1 should be on (set).
7. BUFFER CONTROLS (panel 2)	Place the R/W switch in the up position.
8. BUFFER CONTROLS (panel 2)	Toggle the PLS switch once to read program buffer location 0 into the EWR register (2-1-00 thru 51). Note that bits 48, 49, and 50 will not appear in EWR.

NOTE

Read Even pointer PEB (1-9-04) should be set. Repeat step 8 and observe that Read Odd pointer POB (1-9-00) is set. Repeat step 8 until all 32 buffer locations are read into the EWR. PEB and POB will alternately count as each buffer location is read.

9. CONTROL CLEAR (panel 1)	Press to clear the processor.

STACK BUFFER OPERATION

To perform a stack buffer write operation, proceed as follows:

1. STATUS switch (panel 1)	Place in the TEST position.
2. SINGLE INST switch (panel 1)	Place in the ON position.
3. BUFFER CONTROLS (panel 2)	Place the R/W switch in the down position.
4. STACK (panel 2)	Place the STK switch in the up position.
5. THUMB WHEEL switches	Set to Panel=1, Row=1, Group=9.
6. CLEAR switch (panel 1)	Press to clear the INPUT REGISTER (IN).
7. Panel 1 toggle switches	Set switches 51-0 for the information to be written into the stack buffer. Bit 51 must be correctly set to obtain good parity (odd).
8. LOAD switch (panel 1)	Press to load IN register.
9. THUMB WHEEL switches	Set to Panel=1, Row=5, Group=0.
10. CLEAR switch (panel 1)	Press to clear the S-REGISTER (SR).
11. Panel 1 toggle switches	Set switches 5-0 to the desired stack buffer address.
12. LOAD switch (panel 1)	Press to load the address into SR05-0.
13. BUFFER CONTROLS (panel 2)	Toggle the PLS switch once to write into the stack buffer.

NOTE

The buffer address is incremented after the write occurs; therefore, to write the next location, repeat step 13.

To perform a stack buffer read operation, proceed as follows:

1. BUFFER CONTROLS (panel 2)	Place the R/W switch in the up position.
2. THUMB WHEEL switches	Set to Panel=1, Row=5, Group=0.
3. Panel 1 toggle switches	Set switches 5-0 to the desired stack buffer address.
4. LOAD switch (panel 1)	Press to load the address into SR05-0.
5. BUFFER CONTROLS (panel 2)	Toggle the PLS switch once to read addressed stack buffer location into the IN register.

NOTE

The buffer address is incremented after the read occurs; therefore, to read the next location, repeat step 5.

DISPLAY BUFFER OPERATION

The following registers can be accessed through the Display Buffer:

Address (Decimal)	Address (Hex)	Register Name	Register Usage
0-31	0-1F	D [N]	Display Registers
33	21	SIR	Source Index Register
34	22	DIR	Destination Index Register
35	23	TIR	Table Index Register
37	25	BOSR	Base of Stack Register
39	27	S1LS	Scratch (Spare Local Storage)
48	30	PBR	Program Base Register
49	31	SBR	Source Base Register
50	32	DBR	Destination Base Register
51	33	TBR	Table Base Register
53	35	SNR	Current Stack Vector Index
54	36	PDR	Current Segment Descriptor Index
55	37	S2LS	Scratch (Spare Local Storage)
56	38	ADZ	Alternate D [0]
57	39	APIR	Alternate Program Index Register
58	3A	ALL1	All ones
59	38	LD1	Last D [1] used as SD1 Base

To perform a display write and read operation, proceed as follows:

1. STATUS switch (panel 1)	Place in the TEST position.
2. SINGLE INST switch (panel 1)	Place in the ON position.
3. DISPLAYS (panel 2)	Place the DSP switch in the up position.
4. BUFFER CONTROLS (panel 2)	Place the R/W switch in the down position.
5. THUMB WHEEL switches	Set to Panel=1, Row=8, Group=9.
6. CLEAR switch (panel 1)	Press to clear WRITE PT (DWP) and PCU READ PT (PRP).
7. Panel 1 toggle switches	Set switches 19-14 and 27-22 to desired display buffer address. (Use same address for both switch settings.)
8. LOAD switch (panel 1)	Press to load the address information into DWP5-0 and PRP5-0.
9. Panel 1, Row 9-29 selection switch	Place in the down position for display of DW and DR.
10. THUMB WHEEL switches	Set to Panel=1, Row=29, Group=9.
11. Panel 1 toggle switches	Set switches 51-32 in the up position (write one's) for the information to be written into DW.
12. LOAD switch (panel 1)	Press to load the DW register.
13. BUFFER CONTROLS (panel 2)	Toggle the PLS switch once and observe that DR contains all one's.

EU LOCAL STORAGE OPERATION

To perform an EU local storage write operation, proceed as follows:

NOTE
The switch locations given in steps 1 thru 9 and 1 thru 7 are located on panel 2 of the CPM.

1. EU STORAGE	Place the LOC switch in the up position.
2. SINGLE ROUTINE switch	Place in the ON position.
3. BUFFER CONTROL	Place the R/W switch in the down position.
4. CONTROL CLEAR switch	Press to clear the processor.
5. START switch	Press once to set up CPM for an EU local storage operation.
6. THUMB WHEEL switches	Set to Panel=2, Row=21, Group=9.
7. Panel 2 toggle switches	Set switches 51-00 in the up position (write one's).
8. BUFFER CONTROLS	Toggle the PLS switch once and observe that WEW0 is set (2-09-00). Repeat this step until the count in the WEW pointer returns to zero.

NOTE
All 16 locations in the EU local storage should contain ones.

9. CONTROL CLEAR switch	Press to clear the processor.

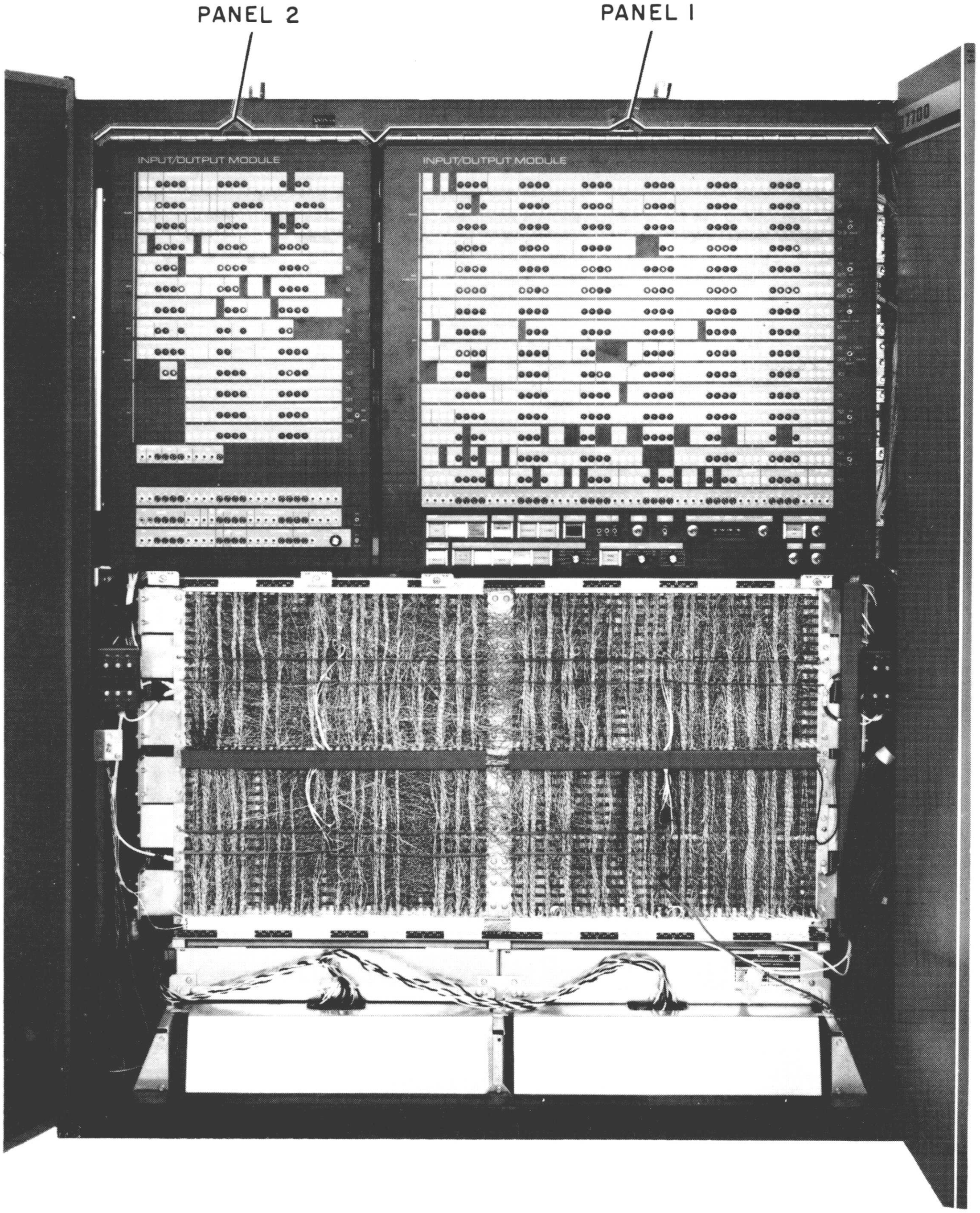
To perform an EU local storage read operation, proceed as follows:

1. START switch	Press once to set up CPM for an EU local storage operation.
2. BUFFER CONTROLS	Place R/W switch in up position.
3. BUFFER CONTROLS	Toggle the PLS switch once and observe that RES0 is set (2-09-50) and C register contains all zero's.
4. BUFFER CONTROLS	Toggle the PLS switch once and observe that RES0 remains set, REW0 is set, and C register contains all one's (bit 51 is off).
5. BUFFER CONTROLS	Toggle the PLS switch until RESR and REW0 are set. Observe that C register contains all one's (bit 51 is on).
6. BUFFER CONTROLS	Toggle the PLS switch until RESW and REW0 are set. Observe that C register contains all one's (bit 51 is off).
7. BUFFER CONTROLS	Toggle the PLS switch until RESW, REW0, and REW1 are set. Observe that C register contains all one's (bit 51 is on).
8. BUFFER CONTROLS	Toggle the PLS switch once and observe that C register contains all one's (bit 51 is on). Repeat this step and observe that C register contains all one's (bit 51 is off).

SECTION 3

INPUT/OUTPUT MODULE PANELS

This section presents an overall view of the input/output module (IOM) in figure VI-3-1 and closeup views of the panels of the input/output module in figures VI-3-2 and VI-3-3.



40902

Figure VI-3-1. Overall View of IOM

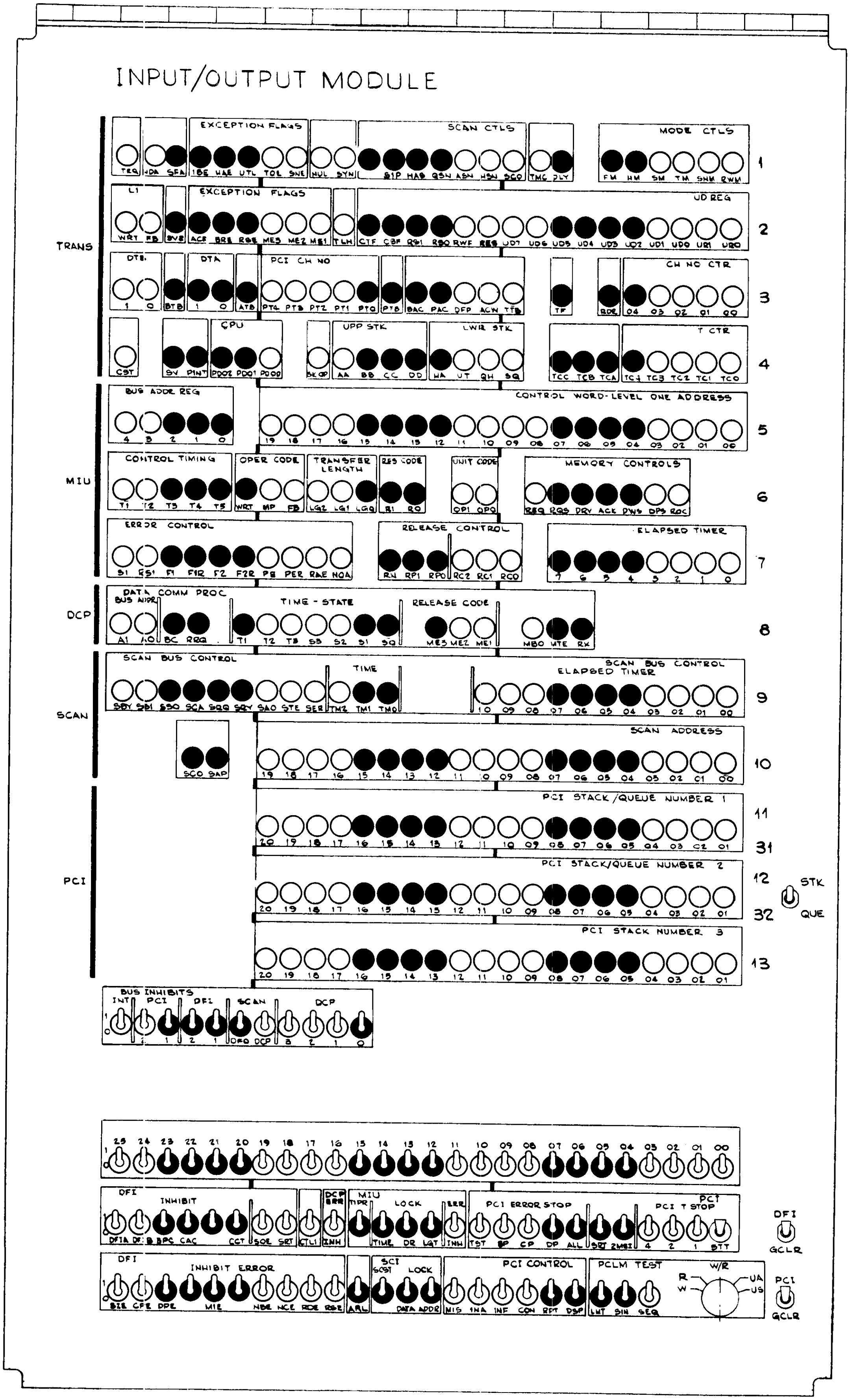
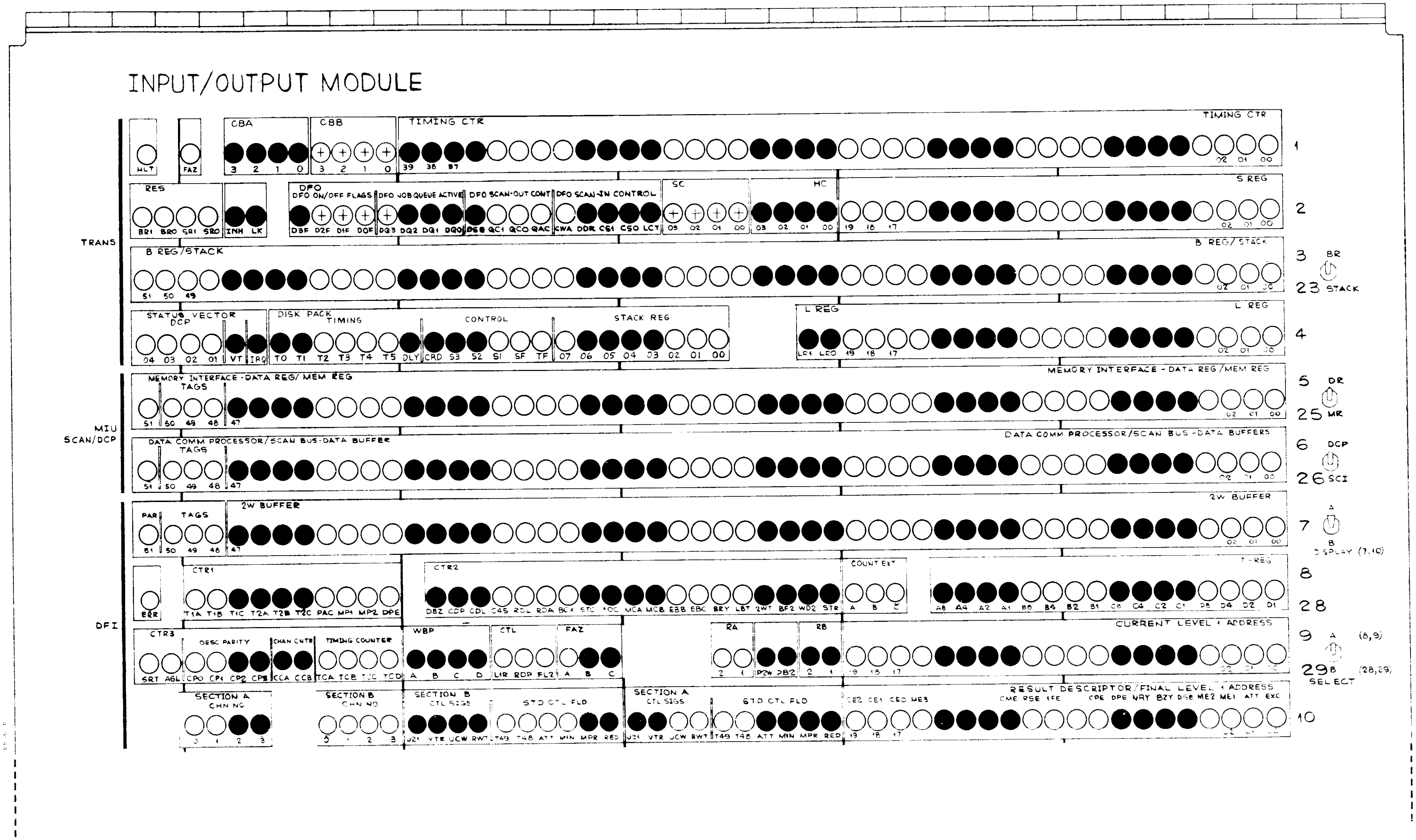


Figure VI-3-2. Left-Hand Panel of IOM


INPUT/OUTPUT MODULE
TRANS
MIU
SCAN/DCP
DFI
HLT
FAZ
CBA
CBB
TIMING CTR
39 38 37
02 01 00
RES
BR1 BR0 SR1 SR0
INH LK
DFO
DFO ON/OFF FLAGS
D3F D2F D1F D0F
DFO JOB QUEUE ACTIVE
DQ3 DQ2 DQ1 DQ0
DFO SCAN-OUT CONT
DS8 QC1 QC0 QAC
DFO SCAN-IN CONTROL
CWA DDR CS1 CS0 LCT
SC
HC
03 02 01 00
S REG
19 18 17
B REG/STACK
51 50 49
STATUS VECTOR
DCP
04 03 02 01 VT IRQ
DISK PACK
TIMING
T0 T1 T2 T3 T4 T5 DLY
CONTROL
CRD S3 S2 S1 SF TF
STACK REG
07 06 05 04 03 02 01 00
L REG
LR1 LR0 19 18 17
MEMORY INTERFACE-DATA REG/MEM REG
MEMORY INTERFACE - DATA REG/MEM REG
TAGS
51 50 49 48 47
DATA COMM PROCESSOR/SCAN BUS-DATA BUFFER
DATA COMM PROCESSOR/SCAN BUS-DATA BUFFERS
PAR
2W BUFFER
CTR1
ERR
T1A T1B T1C T2A T2B T2C PAC MP1 MP2 DPE
CTR2
DBZ CDP CDL C45 RDL RDA BCK STC 10C MCA MCB EBB EBC BRY LBT 2WT BF2 WD2 STR
COUNT EXT
A B C
T-REG
A8 A4 A2 A1 B8 B4 B2 B1 C8 C4 C2 C1 D8 D4 D2 D1
CTR3
SRT AGL
DESC PARITY
CP0 CP1 CP2 CP3
CHAN CNTR
CCA CCB
TIMING COUNTER
TCA TCB TCC TCD
WBP
A B C D
CTL
L1R RDP FL2
FAZ
A B C
RA
2 1
P2W PB2
RB
2 1
CURRENT LEVEL + ADDRESS
SECTION A
CHN NO
0 1 2 3
SECTION B
CHN NO
0 1 2 3
SECTION B
CTL SIGS
J21 VTR UCW RWT
STD CTL FLD
T49 T48 ATT MIN MPR RED
SECTION A
CTL SIGS
J21 VTR UCW RWT
STD CTL FLD
T49 T48 ATT MIN MPR RED
CE2 CE1 CE0 ME3
RESULT DESCRIPTOR/FINAL LEVEL + ADDRESS
CME RSE 1FE CPE DPE NRY BZY DSE ME2 ME1 ATT EXC
1
2
3 BR
23 STACK
4
5 DR
25 MR
6 DCP
26 SCI
7 A
B
DISPLAY (7,10)
8
28
9 A (8,9)
29 B (28,29)
SELECT
10

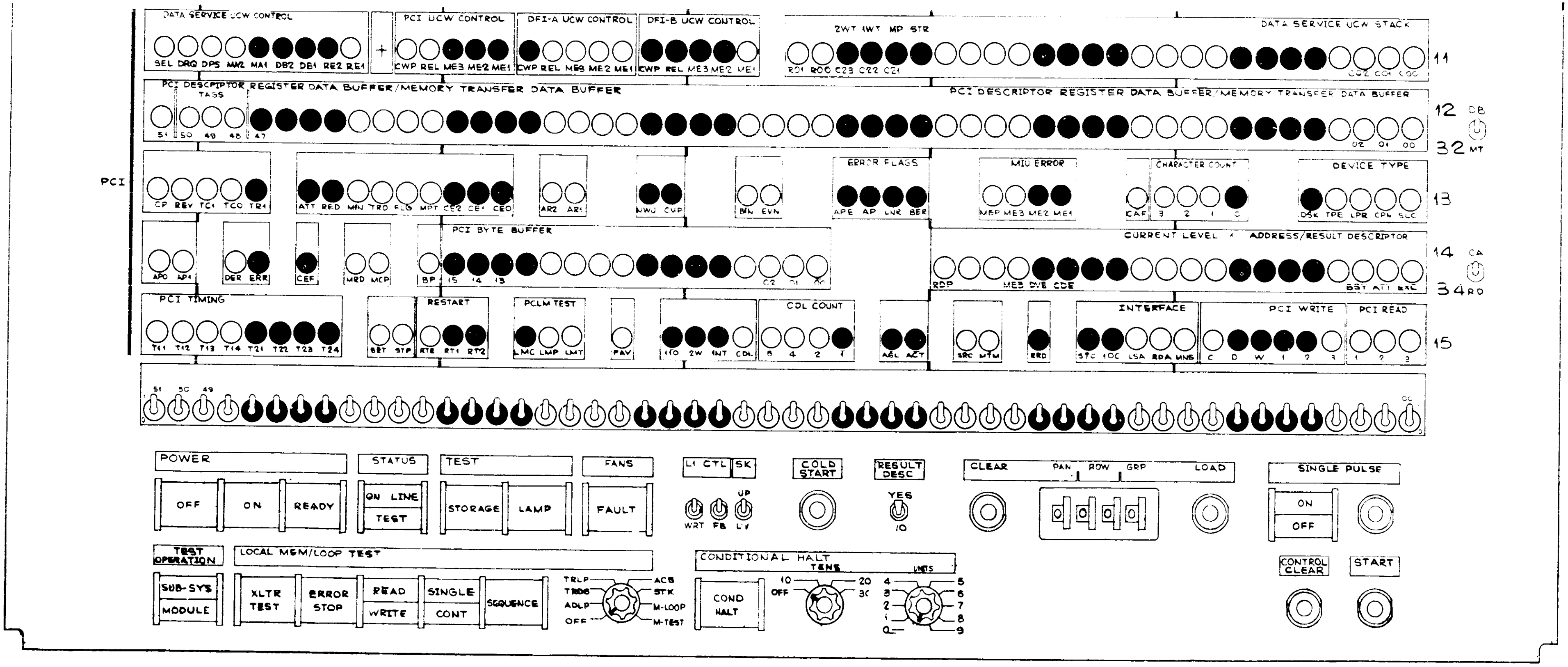


Figure VI-3-3. Right-Hand Panel of IOM

SECTION 4

MEMORY CONTROL MODULE PANEL

This section describes the functions and uses of the controls and indicators on the panel of the memory control module (MCM). An overall view of the memory control module is shown in figure VI-4-1, and a close-up of the panel is shown in figure VI-4-2.

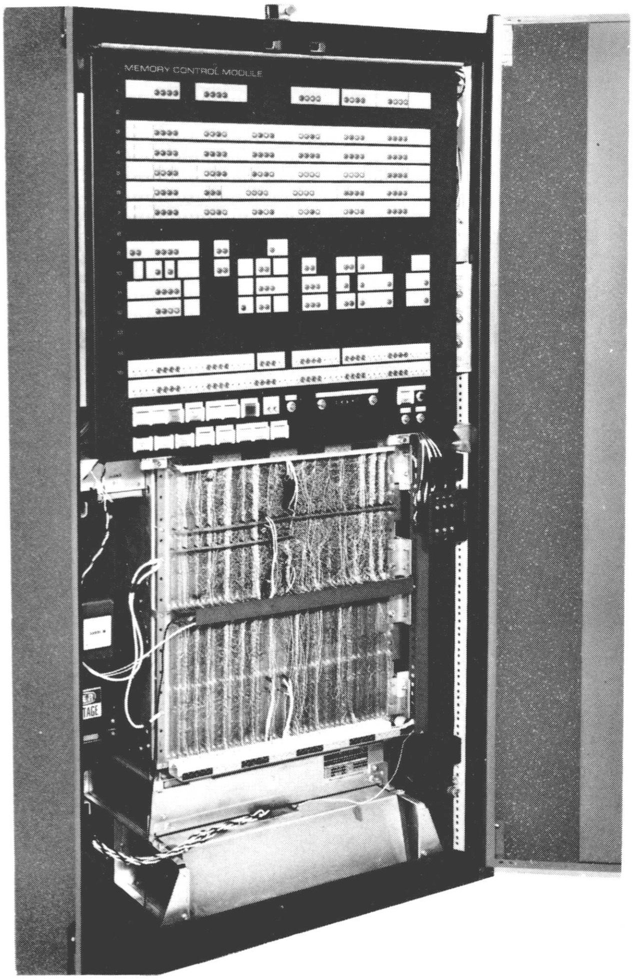


Figure VI-4-1. Overall View of MCM

ROWS 1 THROUGH 12

ROW 1

REQUESTOR

Stores request signals from requestors for evaluation by the priority resolver.

COM FAILURE INHIBITS

Used to prevent a requestor from locking up the priority resolver if he has sent a request without a request strobe or to prevent a requestor from obtaining consecutive services if a lower priority requestor is waiting.

REQUESTOR INHIBIT REG

Used to lock out certain requestors from access.

ADDRESS LIMITS

LOWER

Contains the most significant six bits of the lowest memory address available to the MCM. Can be programmatically loaded by a CPM.

UPPER

Contains the most significant six bits of the highest memory address available to the MCM. Can be programmatically loaded by a CPM.

MSU STATUS

Indicates which of the four possible MSU's are available to the MCM. Can be programmatically loaded by a CPM.

ROW 3

MEMORY BUFFER

The Memory Buffer Register is a 60-bit buffer register for data transferred to or from the MSU's. Bits 52-59, the error-code check bits and overall parity bit, are displayed in Row 12 of the panel.

ROW 4

OUTPUT REGISTER

The Output Register is used to buffer data words that are being transmitted to a requestor.

ROW 5

FAILURE REGISTER

The Failure Register is used to contain all pertinent information necessary to identify and define a failure.

ROW 6

CONTROL WORD REGISTER

The Control Word Register is used to contain the control words transmitted by the requestor.

ROW 7

INPUT REGISTER

The Input Register is used to temporarily buffer words received from a requestor.

ROW 9

INTERNAL ERRORS

Internal Errors Register, used to record internal errors detected by the MCM. The errors detected are:

AVE - MSU Availability Error

DTC - Data Transfer Control (DTC) failure

DTM - Data Timer failure

MPG - Parity Generator (MSU Control) failure

MPE - MSU Parity Error

ADD - Address Counter failure

C/G - Checker/Generator failure

RAV - Read Available failure

MUA - MSU Unavailable

RQS

Special Request flip-flop, indicates that a CPM is making a special request in order to load memory limits or requestor inhibits, or to fetch the fail register. This type of operation bypasses the requestor inhibits.

MBY

MCM Busy flip-flop, used to inhibit access by other requestors until all communications for the using requestor are complete.

HLD

Hold flip-flop, used to allow lockup of the MCM whenever the error stop option is in effect and certain error conditions occur, or during conditional halt operations.

DTC

Data Transfer Control, used for transfer of controls and data within the MCM.

ROW 10

DSP

Data Strobe Problem flip-flop, used to indicate when too many or too few strobes have been sent by requestor.

OV TIM (OC1, OC2)

Overflow Clear Timer, controls the timing for clear/release of MCM from operation requested, due to some lockup type of failure in the requested operation.

CWE

Control Word Error flip-flop, indicates that any of the following errors was detected in the control word:

1. Parity.
2. Wrong MCM Address.
3. Illegal Operation.

EG1

Indicates that the group 1 area of the Fail Register is locked with fail information.

EG2

Indicates that the group 2 area of the Fail Register is locked with fail information.

E2B

Indicates an even number of bits in error in the word received from an MSU.

REQ TIM (RT1, RT2)

Requestor Timer, used in priority resolution. It is initiated each time a new request is made to the MCM.

ROW

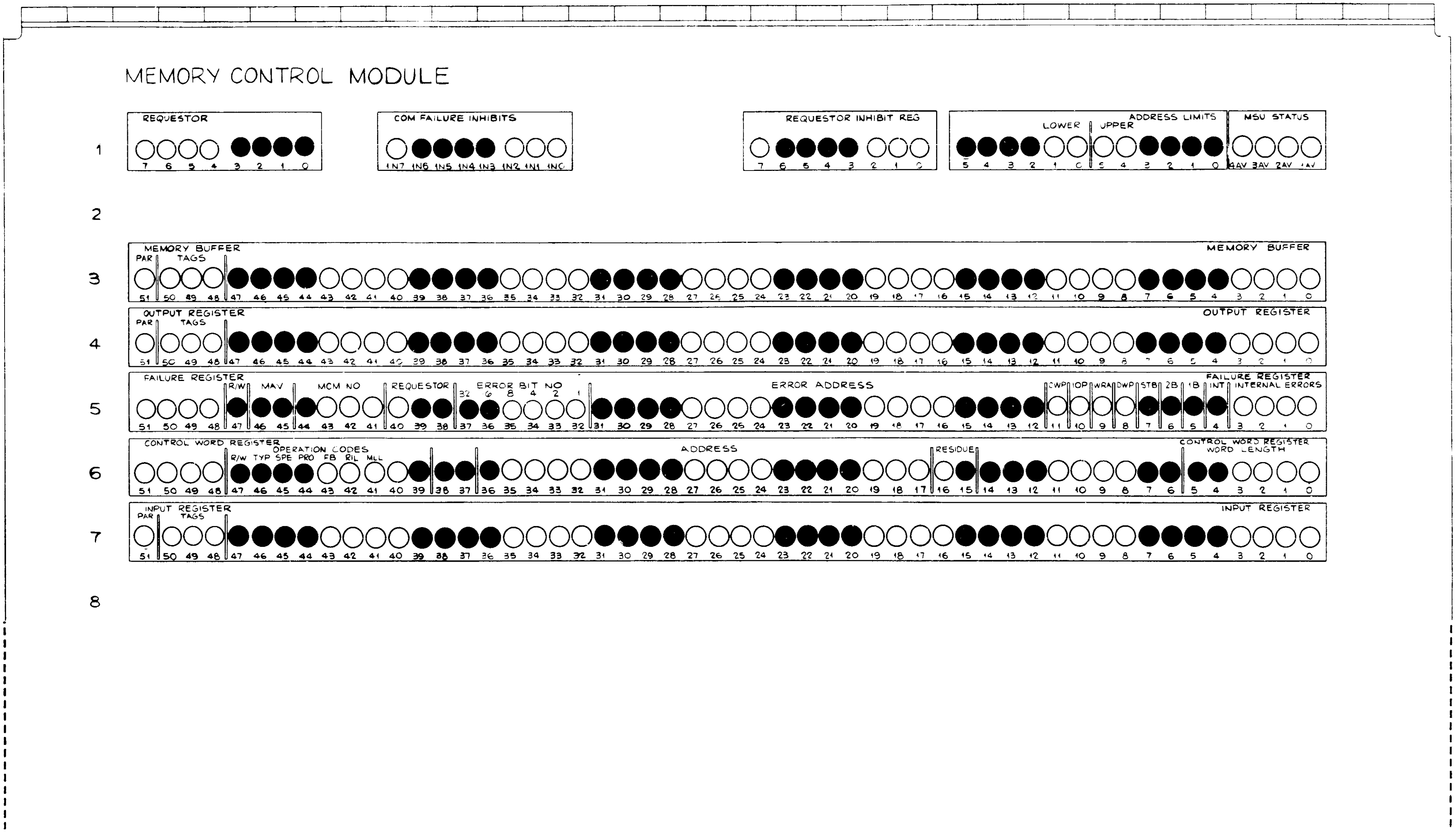
Read or Write flip-flop, used as a basic control during either a read or write operation. Set for a read operation and for the read portion of a protected write or flashback operation. Reset for a write operation.

MPL

Memory Buffer Load flip-flop, indicates that the memory buffer register is loaded, and is used to generate other controls as required by the operation being performed.

IOC

Input/Output Control flip-flop, used to control the direction of data transfer between MCM and requestor. Set during read operation or read portion of a protected write or flashback operation. Reset during a write operation.


MEMORY CONTROL MODULE
1
2
3
4
5
6
7
8
REQUESTOR
COM FAILURE INHIBITS
IN7 IN6 IN5 IN4 IN3 IN2 IN1 INO
REQUESTOR INHIBIT REG
ADDRESS LIMITS
LOWER
UPPER
MSU STATUS
4AV 3AV 2AV 1AV
MEMORY BUFFER
PAR
TAGS
OUTPUT REGISTER
FAILURE REGISTER
R/W
MAV
MCM NO
ERROR BIT NO
32 16 8 4 2 1
ERROR ADDRESS
CWP
IOP
WRA
DWP
STB
2B
1B
INT
INTERNAL ERRORS
CONTROL WORD REGISTER
OPERATION CODES
R/W TYP SPE PRO FB RIL MLL
ADDRESS
RESIDUE
WORD LENGTH
INPUT REGISTER
51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

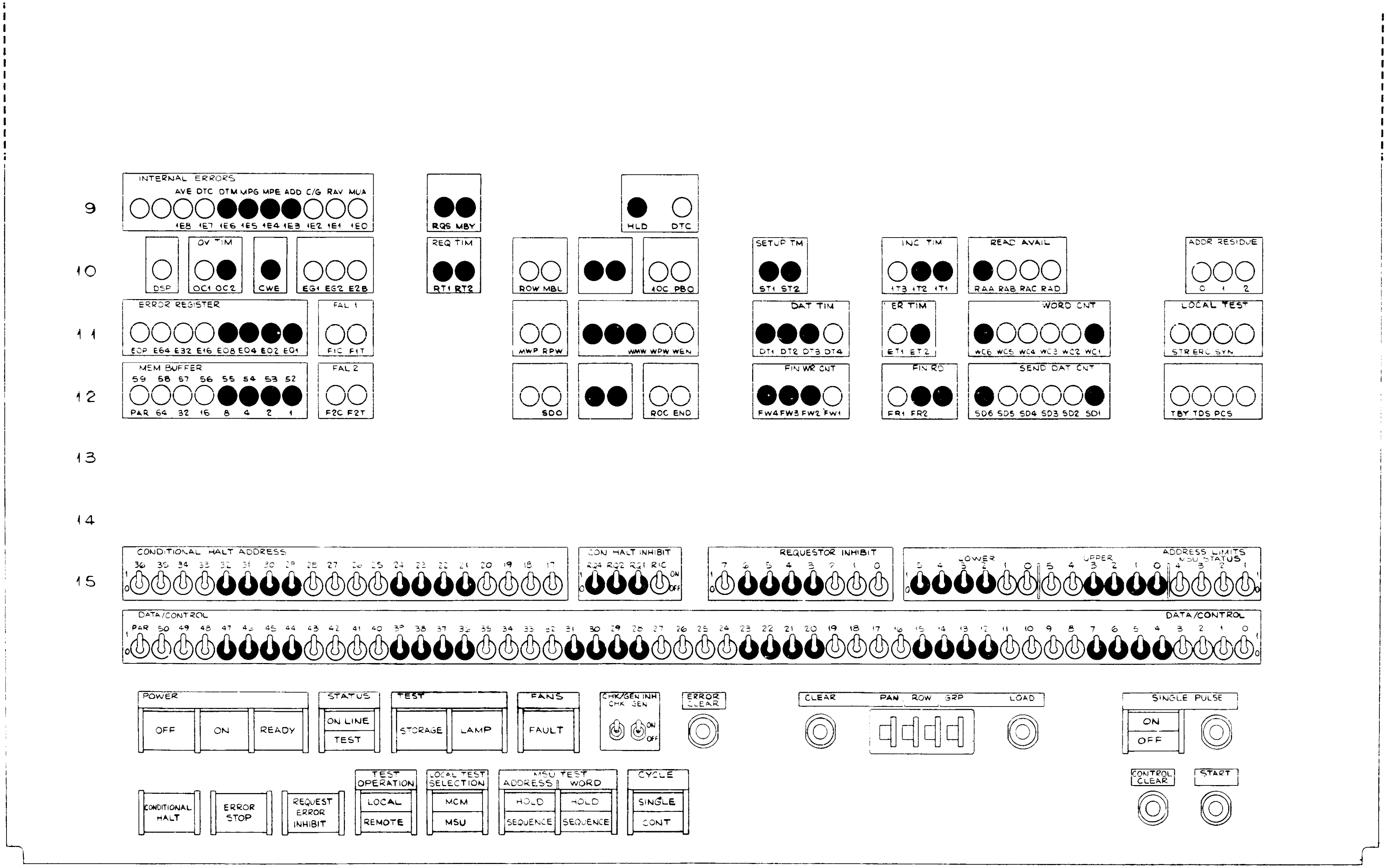


Figure VI-4-2. Panel of MCM

PBO

Presence Bit On flip-flip, indicates that the output register is loaded with data during a read operation.

SETUP TIM (ST1, ST2)

Set-up Timer, used to control initialization of any operation within the MCM.

INC TIM (IT3, IT2, IT1)

Timing counter used to prevent lock-up of MCM for such failures as no read available signal from MSU or no data strobe from requestor in N word write type operations.

READ AVAIL (RAA, RAB, RAC, RAD)

Read Available register, used to store the read available signal transmitted from the selected MSU.

ADDR RESIDUE

Address Residue Counter, used to generate a residue from the address field of a control word. This residue is then compared to the residue field of the control word to detect any error which may have occurred during the transfer.

ROW 11

ERROR REGISTER

Indicates errors detected by the error detection and correction logic. Each bit indicates an error in its respective error correction group.

FAL 1 (F1C, F1T)

Controls the transmission and duration of the FAIL 1 (irrecoverable failure) signal to the requestor.

MWP

Memory Word Protected flip-flop, indicates if a memory word is protected (bit 48 set) during the read portion of a protected write operation.

RPW

Remember Protected Word flip-flop, used to maintain a record that a protected word, 2-bit error, or memory parity error was encountered during a protected write operation.

WMW

Write Multi-Word flip-flop, indicates that timing control of a N-length operation is under control of the requestor data strobe.

WPW

Write Protected Word flip-flop, used to control the writing of a protected word back into its original location on a protected write operation.

WEN

Write Enable flip-flop, used to control writing of data words into the MSU's.

DATA TIM (DT1-DT4)

Data Timer, used to control which MSU is selected during an operation.

ER TIM (ET1, ET2)

Error Timer, used to control single-bit error correction, failure recording, and initiation of failure reporting.

WORD CNT (WC6-WC1)

Word Counter, used to monitor the number of words transferred during any N length operation.

LOCAL TEST

STR

Enables start of any local test operation from the START push button on the MCM panel.

ERC

Allows clear of Fail Register and reset of hold condition in error stop mode of operation.

SYN

Synchronizes start of any local test operation or error clear operation with the 8 MHZ clock.

ROW 12

MEM BUFFER

Memory Buffer Register, used to buffer data transferred to or from the MSU's. This portion of the register contains the check bits and overall parity bit. Bits 0-51 are displayed on Row 3 of the panel.

FAL 2 (F2C, F2T)

Controls the transmission and duration of the FAIL 2 (recoverable failure) signal to the requestor.

SDO

Send Data On flip-flop, used to control the down counting of the Send Data Counter.

ROC

Requestor Operation Complete flip-flop, used to control the sending of the Requestor Operation Complete signal to the requestor.

END

Indicates the end of any operation within the MCM that does not require final read timing.

FIN WR CNT (FW4-FW1)

Final Write Timer, used to control completion of any write operation.

FIN RD (FR1, FR2)

Final Read Timer, used to time out the final portion of a read operation.

SEND DATA CNT (SD6-SD1)

Send Data Counter, used to control the number of words requested from the requestor during a N-length write operation. It is down counted as each data word is sent from the requestor.

It is loaded from the word length field of the control word, limited by the number of MSU's available, or limited to one if the address is less than eight words from the end of an MSU.

TBY

Test Busy signal, acts as local requestor flip-flop for local test operation of MCM.

TDS

Test Data Strobe flip-flop, provides data strobe for local test operation of MCM.

PCS

Generates a non-ringing CLEAR signal for the MSU's from a push-button source at the operator's console.

SWITCHES AND INDICATORS

CONDITIONAL HALT ADDRESS SWITCHES

Select program address to be monitored during any writes into memory.

CONDITIONAL HALT INHIBIT SWITCHES

REQUESTOR INHIBIT CONTROL (RIC)

OFF POSITION

In conditional halt operation lock up of MCM will occur whenever monitored program address is written into.

ON POSITION

Selected requestor is allowed to override conditional halt operation.

REQUESTOR NUMBER SWITCHES (RQ4, RQ2, RQ1)

Select requestor who is allowed to override conditional halt operation.

REQUESTOR INHIBIT SWITCHES

Sets or resets respective FF's each time:

1. Power is cycled up in MCM.
2. CONTROL CLEAR push button is depressed in local test of MCM.
3. CLEAR is depressed on operator's console.

ADDRESS LIMITS SWITCHES

(Lower, Upper, and MSU Status). Same as for requestor inhibit switches.

DATA/CONTROL SWITCHES

Settings used for control word or write data word for local test of MCM.

POWER ON SWITCH

Applies power to MCM and MSU's if in remote.

POWER OFF SWITCH

Removes power from MCM and MSU's if in remote.

POWER READY INDICATOR

Lights at completion of power-on sequence.

STATUS SWITCH

ON LINE POSITION

MCM is in on line operation. All test controls disabled.

TEST POSITION

MCM is in test operation. All test controls may be activated.

TEST

LAMP SWITCH

Enables lamp test of indicators selected by the group thumb-wheel switch.

STORAGE SWITCH

Enables set of all rows equal to pattern set in data/control switches.

FANS FAULT INDICATOR

Illuminates if failure of a cooling fan is detected. If this occurs, the MCM is powered off.

CHECKER INHIBIT SWITCH

Inhibits detection of 1-or 2-bit errors when on.

GENERATOR INHIBIT SWITCH

Inhibits changing of check bits and overall parity bit loaded into memory buffer during write operations.

ERROR CLEAR

Resets the following when depressed:

1. Failure Register.
2. Error Group 1 (EG1) FF.
3. Error Group 2 (EG2) FF.
4. Hold (HLD) FF.

PANEL/ROW/GROUP THUMBWHEELS

PANEL

Selects panel (set to position 1 for MCM).

ROW

Selects register for loading from data/control switches.

GROUP

Selects group for loading from data/control switches or to test in lamp test.

CLEAR PUSH BUTTON

Clears FF's selected by PANEL/ROW/GROUP thumbwheels.

LOAD PUSH BUTTON

Load FF's selected by PANEL/ROW/GROUP thumbwheels.

SINGLE PULSE SWITCH

ON POSITION

8 MHZ clock disabled within MCM.

OFF POSITION 8 MHZ clock enabled within MCM.

SINGLE PULSE PUSH BUTTON

Generates clock pulse each time push button is depressed with SINGLE PULSE switch on.

CONDITIONAL HALT SWITCH

Enables conditional halt operation for writes into memory.

ERROR STOP SWITCH

Enables lock up of MCM for any FAIL1, FAIL2 conditions detected (exception - any detection disabled by requestor error inhibit switch or by CHECKER INHIBIT on/off switch).

REQUESTOR ERROR INHIBIT SWITCH

Disables detection of following errors:
1. Control Word Parity.
2. Wrong MCM Address.
3. Write Data Word Parity Error.
4. Data Strobe Error.

TEST OPERATION SWITCH

LOCAL POSITION

Enables local testing of the MCM (requestors not needed for testing).

REMOTE POSITION

Enables remote testing of the MCM (requestor/requestors needed for testing).

LOCAL TEST SELECTION

MCM POSITION

Enables logic oriented towards testing of the MCM.

MSU POSITION

Enables logic oriented towards testing of MSU's.

MSU TEST SWITCHES

ADDRESS HOLD, WORD HOLD

Single address will be tested within memory.

ADDRESS HOLD, WORD SEQUENCE

Allows word to sequence while address is held (NOTE - this operation should not be used with single MSU).

ADDRESS SEQUENCE, WORD HOLD

Allows test to be executed sequentially on each address within an MSU.

ADDRESS SEQUENCE, WORD SEQUENCE

Allows test to be executed sequentially on each address within the memory available to the MCM.

CYCLE SWITCH

SINGLE POSITION

Only one cycle of test will be executed each time the START push button is depressed.

CONTINUOUS POSITION

Test will be executed continuously.

NOTE

Termination of continuous cycle testing should be accomplished by placing the CYCLE switch in the SINGLE position. It should never be terminated through use of the CONTROL CLEAR push button.

CONTROL CLEAR PUSH BUTTON

Resets the following when pressed:
1. All controls.
2. Control Word Register.
3. Failure Register.

Loads the requestor inhibit and memory limits registers from their respective switches when depressed.

START PUSH BUTTON

Used to initiate any local test of the MCM.

MCM PANEL OPERATIONS

To enable the remaining panel controls the STATUS switch must be in the TEST position. The panel thumbwheel must be in Position 1, since the MCM has only one panel.

LAMP TEST

Turn the LAMP TEST switch on, and place all DATA/CONTROL switches in the 1 position. Set the GROUP thumbwheel to the 0 position and depress the LOAD push button. Flip-flops 0-9 of all rows will light. Stepping the GROUP thumbwheel from positions 0 thru 5 and depressing the LOAD push button at each position will cause the indicators in all rows to light in groups of 10.

STORAGE TEST

Turn on the SINGLE PULSE and STORAGE TEST switches, and place all DATA/CONTROL switches in the 1 position. Press the CLEAR push button, and all indicators should be out. Press the LOAD push button and all indicators light except for spare indicators.

REGISTER LOADING

Place the TEST OPERATION switch in LOCAL. Place the ROW thumbwheel in the position corresponding to the row number of the register to be loaded. Place the GROUP thumbwheel in position 9. Set the desired bit configuration into the DATA/CONTROL switches. Press the CLEAR push button to clear the original contents of the register. Press the LOAD push button, and the contents of the DATA/CONTROL switches are loaded into the selected register.

SINGLE WORD OPERATIONS (READ OR WRITE)

Position the following switches as indicated:

STATUS Switch – TEST position

CHECKER INHIBIT Switch – OFF

GENERATOR INHIBIT Switch – OFF

CONDITIONAL HALT Switch – OFF

ERROR STOP Switch – ON

REQUESTOR ERROR INHIBIT** Switch – OFF

TEST OPERATION Switch – LOCAL position

LOCAL TEST SELECTION Switch – MSU position

ADDRESS SEQUENCE/HOLD* Switch – HOLD position

WORD SEQUENCE/HOLD* Switch – HOLD position

CYCLE* Switch – SINGLE position

SINGLE PULSE Switch – OFF

Load the Control Word Register with a control word indicating the desired operation. The ADDRESS field should contain the desired memory address and the WORD LENGTH field should equal 1. The OPERATION CODE field should be set as follows:

Operation	Bit				
	47	46	45	44	43
Single Word Overwrite	1	0	1	0	0
Single Word Overwrite with Flashback	1	0	1	0	1
Single Word Fetch	0	0	1	0	0
Single Word Protected Write	1	0	1	1	0

All remaining bits in the control word should be zero.

For any variation of write, load the Input Register with the data word to be written. The parity bit should be set correctly for this word.

Press the START push button. The operation will take place.

CONDITIONAL HALT OPERATION

The CONDITIONAL HALT switch should be turned on, the ERROR STOP and REQUEST ERROR INHIBIT switches should be off, and the TEST OPERATION switch should be in the REMOTE position. The CONDITIONAL HALT ADDRESS switches should be set to the memory address which is to be monitored for write operations. The CONDITIONAL HALT INHIBIT switches may be used to allow one requestor to override the conditional halt operation.

*If the ADDRESS and WORD SEQUENCE/HOLD switches are set to the SEQUENCE position, the indicated operation will be repeated for consecutive memory addresses each time the START push button is pressed.

**If the ADDRESS and WORD SEQUENCE/HOLD switches are set to the SEQUENCE position and the CYCLE switch is set to the CONT position, the indicated operation will be repeated for all consecutive address to the limit of the available memory or until an error is detected. If the REQUESTOR ERROR INHIBIT switch is turned OFF, the operation will repeatedly cycle through memory until an error is detected.

SECTION 5

MAINTENANCE DIAGNOSTIC UNIT PANELS

This section presents an overall view of the maintenance diagnostic unit in figure VI-5-1, and close-up views of the panels of the MDU in figures VI-5-2, VI-5-3, and VI-5-4.

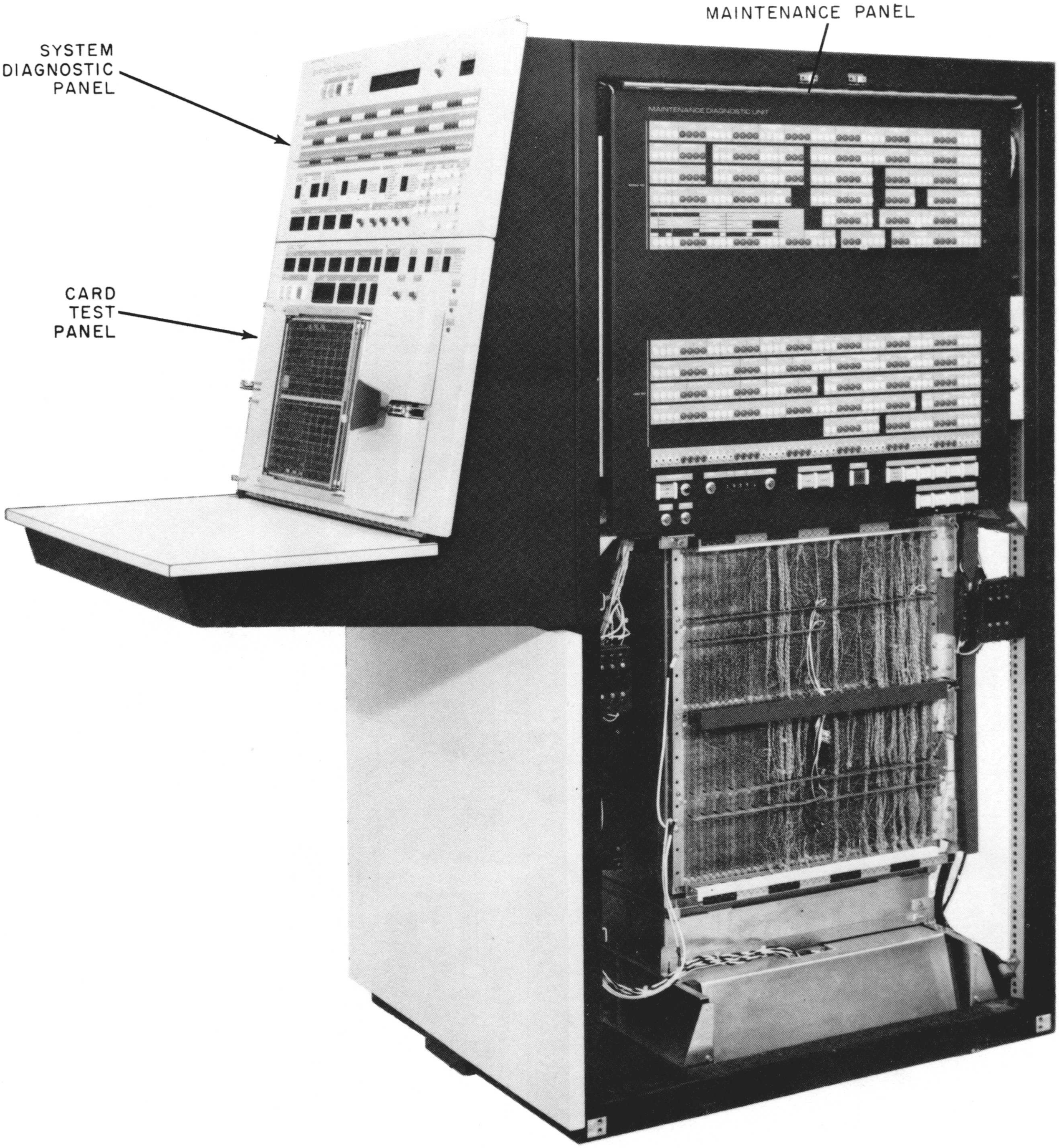
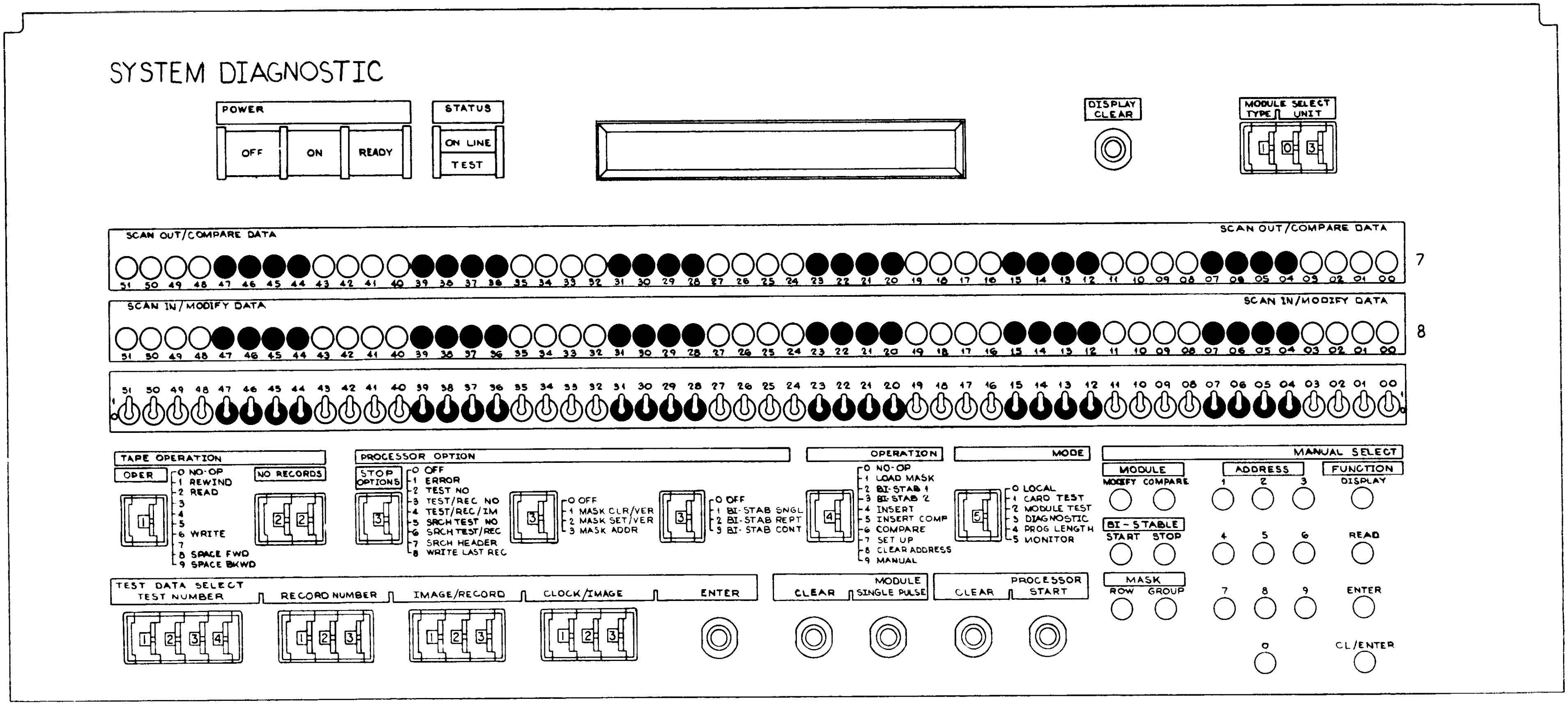


Figure VI-5-1. Maintenance Diagnostic Unit



41006

Figure VI-5-2. System Diagnostic Panel

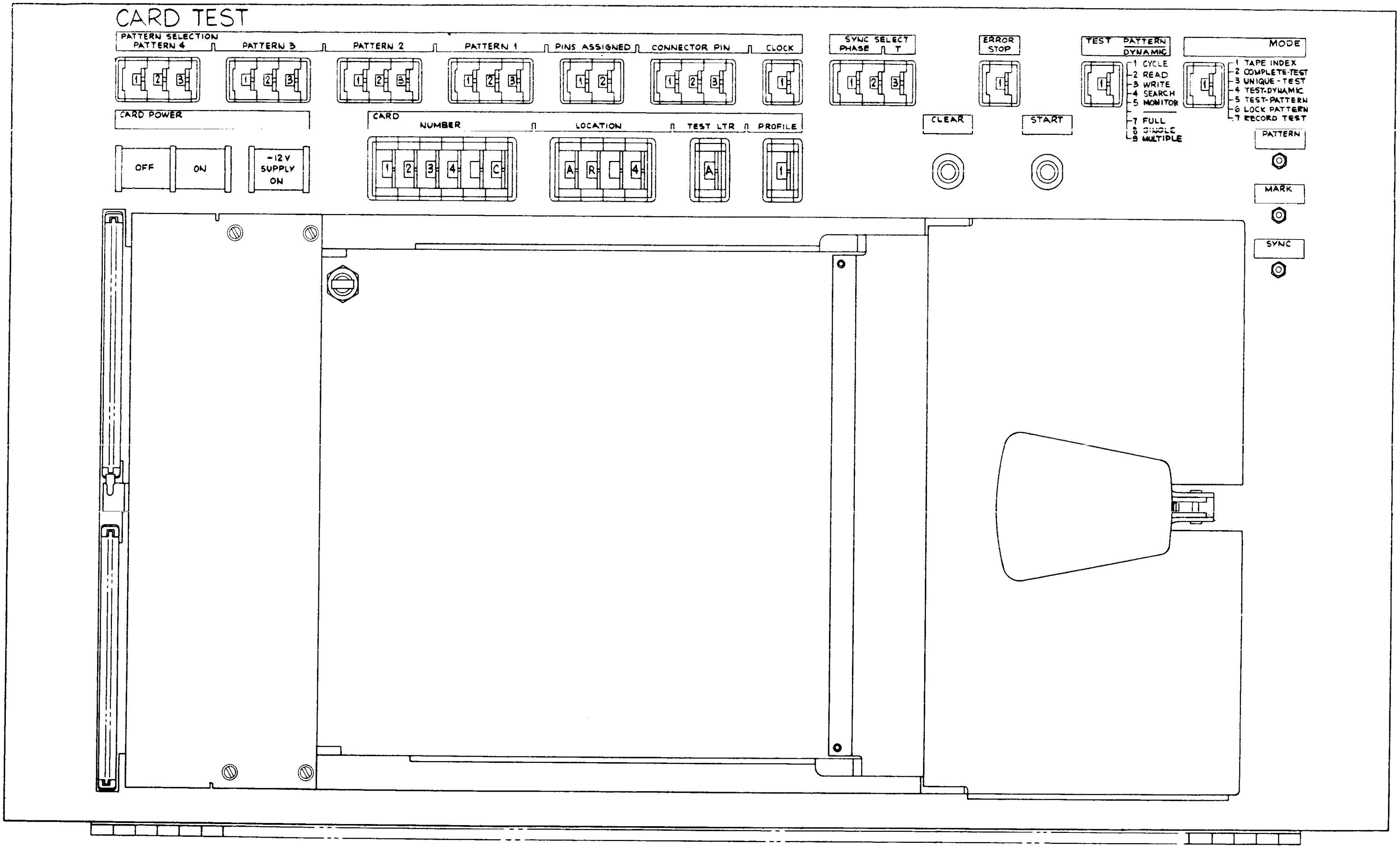
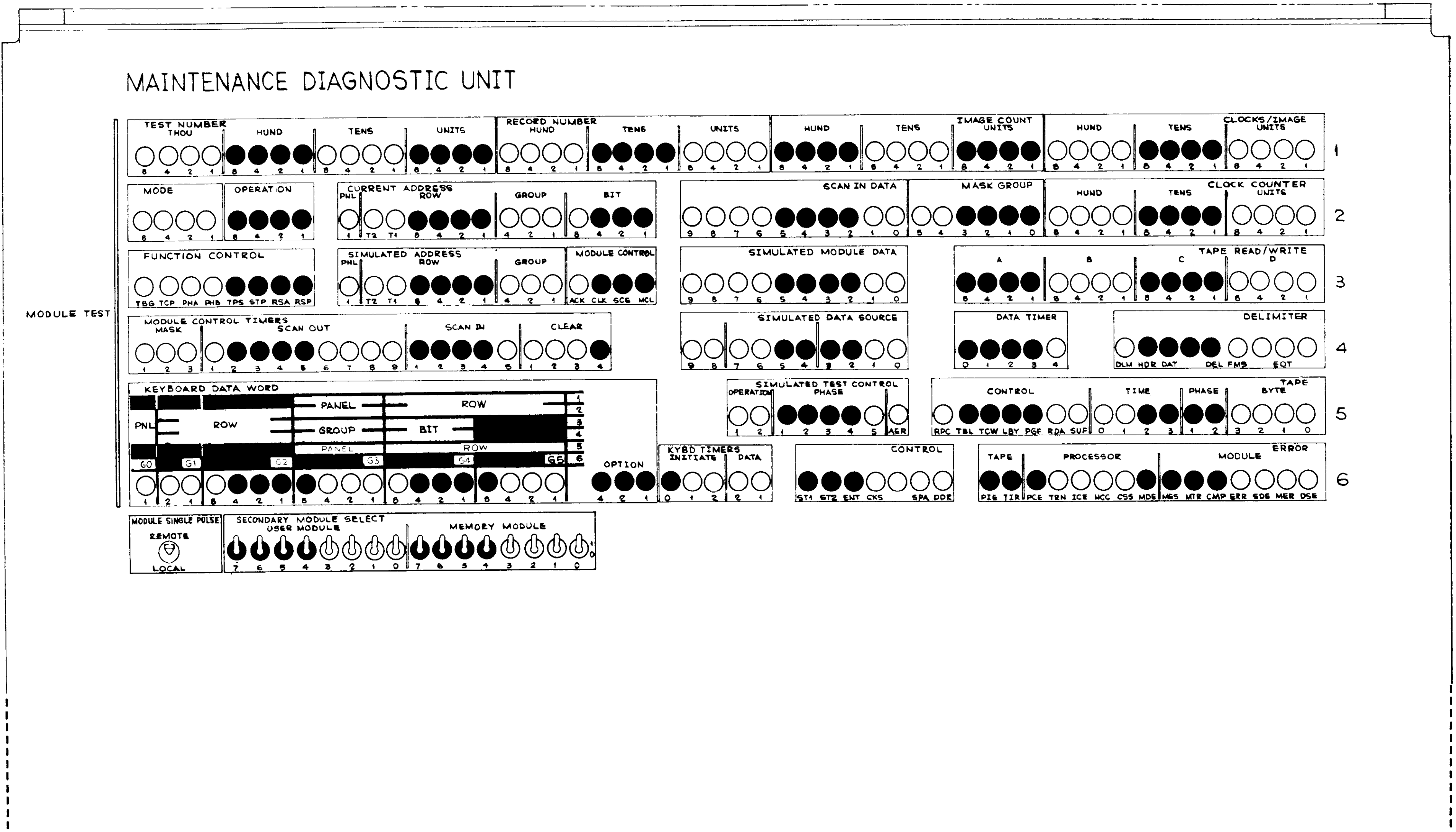


Figure VI-5-3. Card Test Panel


MAINTENANCE DIAGNOSTIC UNIT
MODULE TEST
TEST NUMBER
THOU
HUND
TENS
UNITS
RECORD NUMBER
HUND
TENS
UNITS
IMAGE COUNT
HUND
TENS
UNITS
CLOCKS/IMAGE
HUND
TENS
UNITS
1
MODE
OPERATION
CURRENT ADDRESS
PNL
ROW
GROUP
BIT
T2 T1
SCAN IN DATA
MASK GROUP
CLOCK COUNTER
HUND
TENS
UNITS
2
FUNCTION CONTROL
TBG TCP PHA PHB TPS STP RSA RSP
SIMULATED ADDRESS
PNL
ROW
GROUP
MODULE CONTROL
ACK CLK SCE MCL
SIMULATED MODULE DATA
TAPE READ/WRITE
A
B
C
D
3
MODULE CONTROL TIMERS
MASK
SCAN OUT
SCAN IN
CLEAR
SIMULATED DATA SOURCE
DATA TIMER
DELIMITER
DLM HDR DAT DEL FMS EOT
4
KEYBOARD DATA WORD
PNL
ROW
PANEL
ROW
GROUP
BIT
PANEL
ROW
G0
G1
G2
G3
G4
G5
1 2 3 4 5 6
OPTION
SIMULATED TEST CONTROL
OPERATION
PHASE
AER
CONTROL
RPC TBL TCW LBY PGF RDA SUF
TIME
PHASE
TAPE BYTE
5
KYBD TIMERS
INITIATE
DATA
CONTROL
ST1 ST2 ENT CKS SPA DDR
ERROR
TAPE
PIE TIR
PROCESSOR
PCE TRN ICE HCC CSS MDE
MODULE
MES MTR CMP ERR SDE MER DSE
6
MODULE SINGLE PULSE
REMOTE
LOCAL
SECONDARY MODULE SELECT
USER MODULE
7 6 5 4 3 2 1 0
MEMORY MODULE
7 6 5 4 3 2 1 0
8 4 2 1

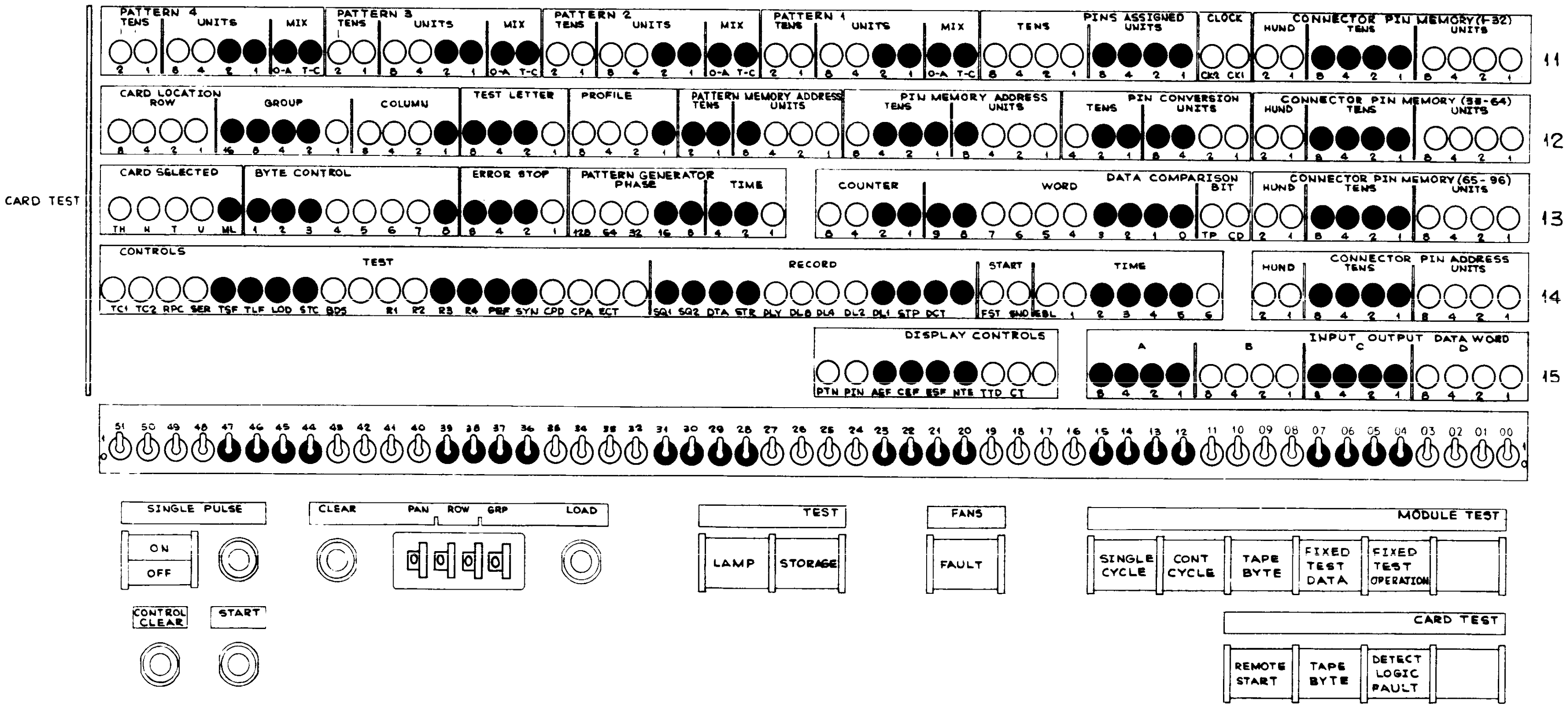


Figure VI-5-4. Maintenance Panel

APPENDIX A

ORDER OF MAGNITUDE CHART

REGISTER BIT SET	DECIMAL	DECIMAL RECIPROCAL	HEX.	OCTAL	BINARY
0	1	1.0	16^0	8^0	2^0
1	2	0.5			
2	4	0.25			
3	8	0.125		8^1	2^3
4	16	0.0625	16^1		
5	32	0.03125			
6	64	0.015625		8^2	2^6
7	128	0.0078125			
8	256	0.00390625	16^2		
9	512	0.001953125		8^3	2^9
10	1024	0.0009765625			
11	2048	0.00048828125			
12	4096	0.000244140625	16^3	8^4	2^{12}
13	8192	0.0001220703125			
14	16384	0.00006103515625			
15	32768	0.000030517578125		8^5	2^{15}
16	65536	0.0000152587890625	16^4		
17	131072	0.00000762939453125			
18	262144	0.000003814697265625		8^6	2^{18}
19	524288	0.0000019073486328125			
20	1048576	0.00000095367431640625	16^5		
21	2097152	0.000000476837158203125		8^7	2^{21}
22	4194304	0.0000002384185791015625			
23	8388608	0.00000011920928955078125			
24	16777216	0.000000059604644775390625	16^6	8^8	2^{24}
25	33554432	0.0000000298023223876953125			
26	67108864	0.00000001490116119384765625			
27	134217728	0.000000007450580596923828125		8^9	2^{27}
28	268435456	0.0000000037252902984619140625	16^7		
29	536870912	0.00000000186264514923095703125			
30	1073741824	0.000000000931322574615478515625		8^{10}	2^{30}
31	2147483648	0.0000000004656612873077392578125			
32	4294967296	0.00000000023283064365386962890625	16^8		
33	8589934592	0.000000000116415321826934814453125		8^{11}	2^{33}
34	17179869184	0.0000000000582076609134674072265625			
35	34359738368	0.00000000002910383045673370361328125			
36	68719476736	0.000000000014551915228366851806640625	16^9	8^{12}	2^{36}
37	137438953472	0.0000000000072759576141834259033203125			
38	274877906944	0.00000000000363797880709171295166015625			
*	549755813887				
39	549755813888	0.000000000001818989403545856475830078125		8^{13}	2^{39}

* FIRST 39 BITS SET. (MAXIMUM INTEGER VALUE ALLOWED).

APPENDIX B

HEXADECIMAL ADDITION TABLE

HEXADECIMAL ADDITION TABLE

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

APPENDIX C

HEXADECIMAL TO DECIMAL CONVERSION TABLES

Table A and table B provide for direct conversion from hexadecimal to decimal numbers in the range of:

Hexadecimal
00000 to FFFFF
or
Decimal
0 to 1048575

Table A provides the decimal value of the first two digits of a five-digit hexadecimal number (nn——), and the table B provides the decimal value of the last three digits of a five-digit hexadecimal number (—NNN).

HEXADECIMAL-DECIMAL CONVERSION TABLE A

x	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
0	0	65536	131072	196608	262144	327680	393216	458752	524288	589824	655360	720896	786432	851968	917504	983040
1	4096	69632	135168	200704	266240	331776	397312	462848	528384	593920	659456	724992	790528	856064	921600	987136
2	8192	73728	139264	204800	270336	335872	401408	466944	532480	598016	663552	729088	794624	860160	925696	991232
3	12288	77824	143360	208896	274432	339968	405504	471040	536576	602112	667648	733184	798720	864256	929792	995328
4	16384	81920	147456	212992	278528	344064	409600	475136	540672	606208	671744	737280	802816	868352	933888	999424
5	20480	86016	151552	217088	282624	348160	413696	479232	544768	610304	675840	741376	806912	872448	937984	1003520
6	24576	90112	155648	221184	286720	352256	417792	483328	548864	614400	679936	745472	811008	876544	942080	1007616
7	28672	94208	159744	225280	290816	356352	421888	487424	552960	618496	684032	749568	815104	880640	946176	1011712
8	32768	98304	163840	229376	294912	360448	425984	491520	557056	622592	688128	753664	819200	884736	950272	1015808
9	36864	102400	167936	233472	299008	364544	430080	495616	561152	626688	692224	757760	823296	888832	954368	1019904
A	40960	106496	172032	237568	303104	368640	434176	499712	565248	630784	696320	761856	827392	892928	958464	1024000
B	45056	110592	176128	241664	307200	372736	438272	503808	569344	634880	700416	765952	831488	897024	962560	1028096
C	49152	114688	180224	245760	311296	376832	442368	507904	573440	638976	704512	770048	835584	901120	966656	1032192
D	53248	118784	184320	249856	315392	380928	446464	512000	577536	643072	708608	774144	839680	905216	970752	1036288
E	57344	122880	188416	253952	319488	385024	450560	516096	581632	647168	712704	778240	843776	909312	974848	1040384
F	61440	126976	192512	258048	323584	389120	454656	520192	585728	651264	716800	782336	847872	913408	978944	1044480

HEXADECIMAL-DECIMAL CONVERSION TABLE B

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
010	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
020	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
030	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
040	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
050	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
060	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
070	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
080	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
090	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
0A0	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
0B0	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
0C0	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
0D0	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
0E0	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
0F0	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
100	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
110	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287
120	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303
130	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
140	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335
150	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351
160	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367
170	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383
180	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399
190	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415
1A0	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431
1B0	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447
1C0	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463
1D0	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479
1E0	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495
1F0	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511
200	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527
210	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543
220	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559
230	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575
240	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591
250	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607
260	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623
270	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639
280	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655
290	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671
2A0	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687
2B0	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703
2C0	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719
2D0	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735
2E0	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751
2F0	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767

HEXADECIMAL-DECIMAL CONVERSION TABLE B (Cont)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
300	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783
310	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799
320	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815
330	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831
340	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847
350	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863
360	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879
370	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895
380	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911
390	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927
3A0	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943
3B0	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959
3C0	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975
3D0	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991
3E0	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

HEXADECIMAL-DECIMAL CONVERSION TABLE B (Cont)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1959	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

HEXADECIMAL-DECIMAL CONVERSION TABLE B (Cont)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

HEXADECIMAL-DECIMAL CONVERSION TABLE B (Cont)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3731	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3827	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

HEXADECIMAL-DECIMAL CONVERSION TABLE B (Cont)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	387i
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3902
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

APPENDIX D

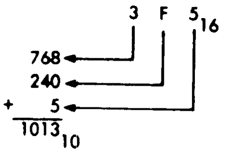
DECIMAL-HEXADECIMAL CONVERSION TABLE

DECIMAL-HEXADECIMAL CONVERSION TABLE

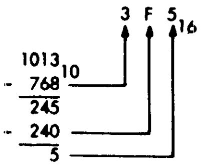
(DECIMAL = H x 16^I)

H↓ I→	6	5	4	3	2	1	0
1	16777216	1048576	65536	4096	256	16	1
2	33554432	2097152	131072	8192	512	32	2
3	50331648	3145728	196608	12288	768	48	3
4	67108864	4194304	262144	16384	1024	64	4
5	83886080	5242880	327680	20480	1280	80	5
6	100663296	6291456	393216	24576	1536	96	6
7	117440512	7340032	458752	28672	1792	112	7
8	134217728	8388608	524288	32768	2048	128	8
9	150994944	9437184	589824	36864	2304	144	9
A	167772160	10485760	655360	40960	2560	160	10
B	184549376	11534336	720896	45056	2816	176	11
C	201326592	12582912	786432	49152	3072	192	12
D	218103808	13631488	851968	53248	3328	208	13
E	234881024	14680064	917504	57344	3584	224	14
F	251658240	15728640	983040	61440	3840	240	15

HEXADECIMAL TO DECIMAL



DECIMAL TO HEXADECIMAL



Hexadecimal to Decimal. Find the decimal value for each hexadecimal digit according to its position. Add these to obtain the decimal equivalent.

Decimal to Hexadecimal. Find the next lower decimal number and its Hexadecimal equivalent. Subtract and use difference to find the next decimal value and hexadecimal equivalent until the complete number is developed.

APPENDIX E

COLLATING INFORMATION

All characters are collated according to their internal binary value. Because the B 7700 has the capability of representing characters internally in BCL, EBCDIC, or USASCII, and because characters are collated according to their internal representation (not necessarily the same as their external mode) a variety of collating sequences is possible. The following table may be used to determine the applicable collating sequence.

Input Mode	Output Mode	Internal Mode	Collating Sequence
BCL	BCL	BCL	BCL (BCL internal)
BCL	EBCDIC	EBCDIC	BCL Translated to EBCDIC
BCL	BCL	EBCDIC	BCL Translated to EBCDIC
EBCDIC	EBCDIC	EBCDIC	EBCDIC
EBCDIC	BCL	EBCDIC	BCL Translated to EBCDIC
EBCDIC	USASCII	EBCDIC	USASCII Translated to EBCDIC
USASCII	USASCII	USASCII	USASCII
USASCII	EBCDIC	EBCDIC	USASCII Translated to EBCDIC
USASCII	BCL	USASCII	BCL Translated to USASCII
USASCII	EBCDIC	USASCII	USASCII Translated to EBCDIC

CHARACTER REPRESENTATION

The BCL, EBCDIC, and USASCII graphics are the same except as follows:

BCL	EBCDIC	USASCII
≥	' (single quote)	'
x (multiply)	! or | or MZ	}
≤	¬ (not)	^
≠	_ (underscore)	
←	| (or)	_
+	PZ (+)	!
[	<	{
]	>	
<	+	
>	-	

A BCL plus sign is never translated to an EBCDIC PZ (plus zero) sign, although the EBCDIC PZ is translated to a BCL plus sign.

EBCDIC 1110 0000 is translated to BCL 00 0000 with an additional flag bit on the next to most significant bit line (7th bit). As the print drums have 64 graphics and space this signal can be used to print the 64th graphic. The 64th graphic is a "CR" for BCL drums and a "¢" for EBCDIC drums.

COLLATING SEQUENCES

EBCDIC			
NUL	+	r	1
SOH	|	s	2
STX	&	t	3
ETX	]	u	4
HT	$	v	5
DEL	*	w	6
VT	)	x	7
FF	;	y	8
CR	¬	z	9
SO	-	PZ	
SI	/	A	
DLE	'	B	
DC1	%	C	
DC2	—	D	
DC3	>	E	
NL	?	F	
BS	:	G	
CAN	#	H	
EM	@	I	
FS	'	MZ(!)	
GS	=	J	
RS	"	K	
US	a	L	
LF	b	M	
ETB	c	N	
ESC	d	O	
ENQ	e	P	
ACK	f	Q	
BEL	g	R	
SYN	h	\	
EOT	i	S	
DC4	j	T	
NAK	k	U	
SUB	l	V	
SP	m	W	
[	n	X	
.	o	Y	
<	p	Z	
(	q	0	

USASCII			
NUL	/	N	u
SOH	(	O	v
STX	)	P	w
ETX	*	Q	x
EOT	+	R	y
ENQ	,	S	z
ACK	-	T	{
BEL	.	U	¦
BS	/	V	}
HT	0	W	~
LF	1	X	DEL
VT	2	Y	
FF	3	Z	
CR	4	[	
SO	5	\	
SI	6	]	
DLE	7	^	
DC1	8	-	
DC2	9	\	
DC3	:	a	
DC4	;	b	
NAK	<	c	
SYN	=	d	
ETB	>	e	
CAN	?	f	
EM	@	g	
SUB	A	h	
ESC	B	i	
FS	C	j	
GS	D	k	
RS	E	l	
US	F	m	
SP	G	n	
|	H	o	
"	I	p	
#	J	q	
$	K	r	
%	L	s	
&	M	t	

BCL	
0	P
1	Q
2	R
3	$
4	*
5	-
6	)
7	;
8	≤
9	(B̄lank)
#	/
@	S
?	T
:	U
>	V
≥	W
+	X
A	Y
B	Z
C	,
D	%
E	≠
F	=
G	]
H	"
I	
.	
[	
&	
(	
<	
←	
×	
J	
K	
L	
M	
N	
O	

LOW

HIGH

EBCDIC COLLATING SEQUENCE

LOW

EBCDIC	Hex.	Internal	Card Code	
Character	Code	Code	Zone	Number
NUL	00	0000 0000	12-0-9-	8-1
SOH	01	0000 0001	12-9-	1
STX	02	0000 0010	12-9-	2
ETX	03	0000 0011	12-9-	3
HT	05	0000 0101	12-9-	5
DEL	07	0000 0111	12-9-	7
VT	0B	0000 1011	12-9-	8-3
FF	0C	0000 1100	12-9-	8-4
CR	0D	0000 1101	12-9-	8-5
SO	0E	0000 1110	12-9-	8-6
SI	0F	0000 1111	12-9-	8-7
DLE	10	0001 0000	12-11-9-	8-1
DC1	11	0001 0001	11-9-	1
DC2	12	0001 0010	11-9-	2
DC3	13	0001 0011	11-9-	3
NL	15	0001 0101	11-9	5
BS	16	0001 0110	11-9-	6
CAN	18	0001 1000	11-9-	8
EM	19	0001 1001	11-9-	8-1
FS	1C	0001 1100	11-9-	8-4
GS	1D	0001 1101	11-9-	8-5
RS	1E	0001 1110	11-9-	8-6
US	1F	0001 1111	11-9-	8-7
LF	25	0010 0101	0-9-	5
ETB	26	0010 0110	0-9-	6
ESC	27	0010 0110	0-9-	7
ENQ	2D	0010 1101	0-9-	8-5
ACK	2E	0010 1110	0-9-	8-6
BEL	2F	0010 1111	0-9-	8-7
SYN	32	0011 0010	9-	2
EOT	37	0011 0111	9-	7
DC4	3C	0011 1100	9-	8-4
NAK	3D	0011 1101	9-	8-5
SUB	3F	0011 1111	9-	8-7
SP	40	0100 0000	(No Punches)	
[	4A	0100 1010	12-	8-2
.	4B	0100 1011	12-	8-3
<	4C	0100 1100	12-	8-4
(	4D	0100 1101	12-	8-5
+	4E	0100 1110	12-	8-6
| (←)	4F	0100 1111	12-	8-7

HIGH

LOW

EBCDIC	Hex.	Internal	Card Code	
Character	Code	Code	Zone	Number
&	50	0101 0000	12-	-
]	5A	0101 1010	11-	8-2
$	5B	0101 1011	11-	8-3
*	5C	0101 1100	11-	8-4
)	5D	0101 1101	11-	8-5
;	5E	0101 1110	11-	8-6
¬ (≤)	5F	0101 1111	11-	8-7
- (Dash)	60	0110 0000	11-	-
/	61	0110 0001	0-	1
, (Comma)	6B	0110 1011	0-	8-3
%	6C	0110 1100	0-	8-4
_ (≠)	6D	0110 1101	0-	8-5
>	6E	0110 1110	0-	8-6
?	6F	0110 1111	0-	8-7
:	7A	0111 1010	-	8-2
#	7B	0111 1011	-	8-3
@	7C	0111 1100	-	8-4
' (≥)	7D	0111 1101	-	8-5
=	7E	0111 1110	-	8-6
"	7F	0111 1111	-	8-7
a	81	1000 0001	12-0-	1
b	82	1000 0010	12-0-	2
c	83	1000 0011	12-0-	3
d	84	1000 0100	12-0-	4
e	85	1000 0101	12-0-	5
f	86	1000 0110	12-0-	6
g	87	1000 0111	12-0-	7
h	88	1000 1000	12-0-	8
i	89	1000 1001	12-0-	9
j	91	1001 0001	12-11-	1
k	92	1001 0010	12-11-	2
l	93	1001 0011	12-11-	3
m	94	1001 0100	12-11-	4
n	95	1001 0101	12-11-	5
o	96	1001 0110	12-11-	6
p	97	1001 0111	12-11-	7
q	98	1001 1000	12-11-	8
r	99	1001 1001	12-11-	9

HIGH

EBCDIC COLLATING SEQUENCE (Cont)

LOW

EBCDIC Character	Hex. Code	Internal Code	Card Code Zone	Number
s	A2	1010 0010	11-0-	2
t	A3	1010 0011	11-0-	3
u	A4	1010 0100	11-0-	4
v	A5	1010 0101	11-0-	5
w	A6	1010 0110	11-0-	6
x	A7	1010 0111	11-0-	7
y	A8	1010 1000	11-0-	8
z	A9	1010 1001	11-0-	9
PZ (+)	C0	1100 0000	12-0	
A	C1	1100 0001	12-	1
B	C2	1100 0010	12-	2
C	C3	1100 0011	12-	3
D	C4	1100 0100	12-	4
E	C5	1100 0101	12-	5
F	C6	1100 0110	12-	6
G	C7	1100 0111	12-	7
H	C8	1100 1000	12-	8
I	C9	1100 1001	12-	9
MZ (!)	D0	1101 0000	11-	0
J	D1	1101 0001	11-	1
K	D2	1101 0010	11-	2
L	D3	1101 0011	11-	3
M	D4	1101 0100	11-	4
N	D5	1101 0101	11-	5
O	D6	1101 0110	11-	6
P	D7	1101 0111	11-	7
Q	D8	1101 1000	11-	8
R	D9	1101 1001	11-	9
\ (CR) (¢)	E0	1110 0000	0-	8-2
S	E2	1110 0010	0-	2
T	E3	1110 0011	0-	3
U	E4	1110 0100	0-	4
V	E5	1110 0101	0-	5
W	E6	1110 0110	0-	6
X	E7	1110 0111	0-	7
Y	E8	1110 1000	0-	8
Z	E9	1110 1001	0-	9

HIGH

LOW

EBCDIC Character	Hex. Code	Internal Code	Card Code Zone	Number
0	F0	1111 0000	-	0
1	F1	1111 0001	-	1
2	F2	1111 0010	-	2
3	F3	1111 0011	-	3
4	F4	1111 0100	-	4
5	F5	1111 0101	-	5
6	F6	1111 0110	-	6
7	F7	1111 0111	-	7
8	F8	1111 1000	-	8
9	F9	1111 1001	-	9

HIGH

BCL COLLATING SEQUENCE (BCL INTERNAL)

BCL	BCL	BCL	BCL Internal	BCL External	Card Code	
Character	Octal	Hex	BA 8421	BA 8421	Zone	Number
0	00	00	00 0000	00 1010	-	0
1	01	01	00 0001	00 0001	-	1
2	02	02	00 0010	00 0010	-	2
3	03	03	00 0011	00 0011	-	3
4	04	04	00 0100	00 0100	-	4
5	05	05	00 0101	00 0101	-	5
6	06	06	00 0110	00 0110	-	6
7	07	07	00 0111	00 0111	-	7
8	10	08	00 1000	00 1000	-	8
9	11	09	00 1001	00 1001	-	9
#	12	0A	00 1010	00 1011	-	8-3
@	13	0B	00 1011	00 1100	-	8-4
?	14	0C	00 1100	00 0000	All other card codes	
:	15	0D	00 1101	00 1101	-	8-5
>	16	0E	00 1110	00 1110	-	8-6
≥	17	0F	00 1111	00 1111	-	8-7
+	20	10	01 0000	11 1010	12	0
A	21	11	01 0001	11 0001	12	1
B	22	12	01 0010	11 0010	12	2
C	23	13	01 0011	11 0011	12	3
D	24	14	01 0100	11 0100	12	4
E	25	15	01 0101	11 0101	12	5
F	26	16	01 0110	11 0110	12	6
G	27	17	01 0111	11 0111	12	7

LOW ↕ HIGH

BCL COLLATING SEQUENCE (BCL INTERNAL) (Cont)

				BCL			
BCL	BCL	BCL	BCL Internal	External	Card Code		
Character	Octal	Hex	BA 8421	BA 8421	Zone	Number	
H	30	18	01 1000	11 1000	12	8	LOW
I	31	19	01 1001	11 1001	12	9	
.	32	1A	01 1010	11 1011	12	8-3	
[	33	1B	01 1011	11 1100	12	8-4	
&	34	1C	01 1100	11 0000	12	-	
(	35	1D	01 1101	11 1101	12	8-5	
<	36	1E	01 1110	11 1110	12	8-6	
←	37	1F	01 1111	11 1111	12	8-7	
x(Mult.)	40	20	10 0000	10 1010	11	0	
J	41	21	10 0001	10 0001	11	1	
K	42	22	10 0010	10 0010	11	2	
L	43	23	10 0011	10 0011	11	3	
M	44	24	10 0100	10 0100	11	4	
N	45	25	10 0101	10 0101	11	5	
O	46	26	10 0110	10 0110	11	6	
P	47	27	10 0111	10 0111	11	7	
Q	50	28	10 1000	10 1000	11	8	
R	51	29	10 1001	10 1001	11	9	
$	52	2A	10 1010	10 1011	11	8-3	
*	53	2B	10 1011	10 1100	11	8-4	
-	54	2C	10 1100	10 0000	11	-	
)	55	2D	10 1101	10 1101	11	8-5	
;	56	2E	10 1110	10 1110	11	8-6	
≤	57	2F	10 1111	10 1111	11	8-7	
Blank	60	30	11 0000	01 0000	-	-	
/	61	31	11 0001	01 0001	0	1	
S	62	32	11 0010	01 0010	0	2	
T	63	33	11 0011	01 0011	0	3	
U	64	34	11 0100	01 0100	0	4	
V	65	35	11 0101	01 0101	0	5	
W	66	36	11 0110	01 0110	0	6	
X	67	37	11 0111	01 0111	0	7	
Y	70	38	11 1000	01 1000	0	8	
Z	71	39	11 1001	01 1001	0	9	
,	72	3A	11 1010	01 1011	0	8-3	
%	73	3B	11 1011	01 1100	0	8-4	
≠	74	3C	11 1100	01 1010	0	8-2	
=	75	3D	11 1101	01 1101	0	8-5	
]	76	3E	11 1110	01 1110	0	8-6	
"	77	3F	11 1111	01 1111	0	8-7	HIGH

COLLATING SEQUENCE - USASCII X3.4-1968

LOW

USASCII Character	Hex Code	Internal Code
NUL	00	0000 0000
SOH	01	0000 0001
STX	02	0000 0010
ETX	03	0000 0011
EOT	04	0000 0100
ENQ	05	0000 0101
ACK	06	0000 0110
BEL	07	0000 0111
BS	08	0000 1000
HT	09	0000 1001
LF	0A	0000 1010
VT	0B	0000 1011
FF	0C	0000 1100
CR	0D	0000 1101
SO	0E	0000 1110
SI	0F	0000 1111
DLE	10	0001 0000
DC1	11	0001 0001
DC2	12	0001 0010
DC3	13	0001 0011
DC4	14	0001 0100
NAK	15	0001 0101
SYN	16	0001 0110
ETB	17	0001 0111
CAN	18	0001 1000
EM	19	0001 1001
SUB	1A	0001 1010
ESC	1B	0001 1011
FS	1C	0001 1100
GS	1D	0001 1101
RS	1E	0001 1110
US	1F	0001 1111
SP	20	0010 0000
| (or)	21	0010 0001
"	22	0010 0010
#	23	0010 0011
$	24	0010 0100
%	25	0010 0101
&	26	0010 0110
'	27	0010 0111
(	28	0010 1000
)	29	0010 1001
*	2A	0010 1010
+	2B	0010 1011
,	2C	0010 1100

HIGH

LOW

USASCII Character	Hex Code	Internal Code
-	2D	0010 1101
.	2E	0010 1110
/	2F	0010 1111
0	30	0011 0000
1	31	0011 0001
2	32	0011 0010
3	33	0011 0011
4	34	0011 0100
5	35	0011 0101
6	36	0011 0110
7	37	0011 0111
8	38	0011 1000
9	39	0011 1001
:	3A	0011 1010
;	3B	0011 1011
<	3C	0011 1100
=	3D	0011 1101
>	3E	0011 1110
?	3F	0011 1111
@	40	0100 0000
A	41	0100 0001
B	42	0100 0010
C	43	0100 0011
D	44	0100 0100
E	45	0100 0101
F	46	0100 0110
G	47	0100 0111
H	48	0100 1000
I	49	0100 1001
J	4A	0100 1010
K	4B	0100 1011
L	4C	0100 1100
M	4D	0100 1101
N	4E	0100 1110
O	4F	0100 1111
P	50	0101 0000
Q	51	0101 0001
R	52	0101 0010
S	53	0101 0011
T	54	0101 0100
U	55	0101 0101
V	56	0101 0110
W	57	0101 0111
X	58	0101 1000

HIGH

COLLATING SEQUENCE - USASCII X3.4-1968 (Cont)

USASCII Character	Hex Code	Internal Code	
Y	59	0101 1001	LOW
Z	5A	0101 1010	
[	5B	0101 1011	
\	5C	0101 1100	
]	5D	0101 1101	
^ (¬)	5E	0101 1100	
-	5F	0101 1111	
‘	60	0110 0000	
a	61	0110 0001	
b	62	0110 0010	
c	63	0110 0011	
d	64	0110 0100	
e	65	0110 0101	
f	66	0110 0110	
g	67	0110 0111	
h	68	0110 1000	
i	69	0110 1001	
j	6A	0110 1010	
k	6B	0110 1011	
l	6C	0110 1100	HIGH

USASCII Character	Hex Code	Internal Code	
m	6D	0110 1101	LOW
n	6E	0110 1110	
o	6F	0110 1111	
p	70	0111 0000	
q	71	0111 0001	
r	72	0111 0010	
s	73	0111 0011	
t	74	0111 0100	
u	75	0111 0101	
v	76	0111 0110	
w	77	0111 0111	
x	78	0111 1000	
y	79	0111 1001	
z	7A	0111 1010	
{	7B	0111 1011	
¦	7C	0111 1100	
}	7D	0111 1101	
~	7E	0111 1110	
DEL	7F	0111 1111	HIGH

COLLATING SEQUENCE - BCL TRANSLATED TO EBCDIC

COLLATING SEQUENCE - BCL TRANSLATED TO EBCDIC

BCL Character	BCL External BA 4321	BCL Hex.	BCL Octal	Translated EBCDIC Code	EBCDIC Hex	Card Code Zone	Card Code Number	
(Blank)	01 0000	10	20	0100 0000	40	-	-	LOW ↑
[	11 1100	3C	74	0100 1010	4A	12	8-4	
.	11 1011	3B	73	0100 1011	4B	12	8-3	
<	11 1110	3E	76	0100 1100	4C	12	8-6	
(	11 1101	3D	75	0100 1101	4D	12	8-5	
+	11 1010	3A	72	0100 1110	4E	12	0	
←	11 1111	3F	77	0100 1111	4F	12	8-7	
&	11 0000	30	60	0101 0000	50	12	-	
]	01 1110	1E	36	0101 1010	5A	0	8-6	
$	10 1011	2B	53	0101 1011	5B	11	8-3	
*	10 1100	2C	54	0101 1100	5C	11	8-4	
)	10 1101	2D	55	0101 1101	5D	11	8-5	
;	10 1110	2E	56	0101 1110	5E	11	8-6	
≤	10 1111	2F	57	0101 1111	5F	11	8-7	
-	10 0000	20	40	0110 0000	60	11	-	
/	01 0001	11	21	0110 0001	61	0	1	
,	01 1011	1B	33	0110 1011	6B	0	8-3	
%	01 1100	1C	34	0110 1100	6C	0	8-4	
≠	01 1010	1A	32	0110 1101	6D	0	8-2	
>	00 1110	0E	16	0110 1110	6E	-	8-6	
?	00 0000	00	00	0110 1111	6F	All other card codes		
:	00 1101	0D	15	0111 1010	7A	-	8-5	
#	00 1011	0B	13	0111 1011	7B	-	8-3	
@	00 1100	0C	14	0111 1100	7C	-	8-4	
≥	00 1111	0F	17	0111 1101	7D	-	8-7	
=	01 1101	1D	35	0111 1110	7E	0	8-5	
"	01 1111	1F	37	0111 1111	7F	0	8-7	
A	11 0001	31	61	1100 0001	C1	12	1	
B	11 0010	32	62	1100 0010	C2	12	2	
C	11 0011	33	63	1100 0011	C3	12	3	
D	11 0100	34	64	1100 0100	C4	12	4	
E	11 0101	35	65	1100 0101	C5	12	5	
F	11 0110	36	66	1100 0110	C6	12	6	
G	11 0111	37	67	1100 0111	C7	12	7	
H	11 1000	38	70	1100 1000	C8	12	8	
I	11 1001	39	71	1100 1001	C9	12	9	↓ HIGH

COLLATING SEQUENCE - BCL TRANSLATED TO EBCDIC (Cont)

BCL Character	BCL External BA 4321	BCL Hex.	BCL Octal	Translated EBCDIC Code	EBCDIC Hex.	Card Code Zone	Card Code Number	
x(mult)	10 1010	2A	52	1101 0000	D0	11	0	LOW
J	10 0001	21	41	1101 0001	D1	11	1	
K	10 0010	22	42	1101 0010	D2	11	2	
L	10 0011	23	43	1101 0011	D3	11	3	
M	10 0100	24	44	1101 0100	D4	11	4	
N	10 0101	25	45	1101 0101	D5	11	5	
O	10 0110	26	46	1101 0110	D6	11	6	
P	10 0111	27	47	1101 0111	D7	11	7	
Q	10 1000	28	50	1101 1000	D8	11	8	
R	10 1001	29	51	1101 1001	D9	11	9	
S	01 0010	12	22	1110 0010	E2	0	2	
T	01 0011	13	23	1110 0011	E3	0	3	
U	01 0100	14	24	1110 0100	E4	0	4	
V	01 0101	15	25	1110 0101	E5	0	5	
W	01 0110	16	26	1110 0110	E6	0	6	
X	01 0111	17	27	1110 0111	E7	0	7	
Y	01 1000	18	30	1110 1000	E8	0	8	
Z	01 1001	19	31	1110 1001	E9	0	9	
0	00 1010	0A	12	1111 0000	F0	-	0	
1	00 0001	01	01	1111 0001	F1	-	1	
2	00 0010	02	02	1111 0010	F2	-	2	
3	00 0011	03	03	1111 0011	F3	-	3	
4	00 0100	04	04	1111 0100	F4	-	4	
5	00 0101	05	05	1111 0101	F5	-	5	
6	00 0110	06	06	1111 0110	F6	-	6	
7	00 0111	07	07	1111 0111	F7	-	7	
8	00 1000	08	10	1111 1000	F8	-	8	
9	00 1001	09	11	1111 1001	F9	-	9	HIGH

COLLATING SEQUENCE - BCL TRANSLATED TO USASCII

LOW

BCL Character	BCL External BA 8421	BCL Hex.	BCL Octal	Translated USASCII Code	USASCII Hex.	Card Code Zone Numbers
Blank	01 0000	10	20	0010 0000	20	- -
←	11 1111	3F	77	0010 0001	21	12 8-7
"	01 1111	1F	37	0010 0010	22	0 8-7
#	01 1011	0B	33	0010 0011	23	- 8-3
$	10 1011	2B	53	0010 0100	24	11 8-3
%	01 1100	1C	34	0010 0101	25	0 8-4
&	11 0000	30	60	0010 0110	26	12 -
≥	01 1111	0F	37	0010 0111	27	- 8-7
(	11 1101	3D	75	0010 1000	28	12 8-3
)	10 1101	2D	55	0010 1001	29	11 8-5
*	10 1100	2C	54	0010 1010	2A	11 8-4
+	11 1010	3A	72	0010 1011	2B	12 0
,	01 1011	1B	33	0010 1100	2C	0 8-3
-	10 0000	20	40	0010 1101	2D	11 -
.	10 1011	3B	53	0010 1110	2E	12 8-3
/	01 0001	11	21	0010 1111	2F	0 1
0	00 1010	0A	12	0011 0000	30	- 0
1	00 0001	01	01	0011 0001	31	- 1
2	00 0010	02	02	0011 0010	32	- 2
3	00 0011	03	03	0011 0011	33	- 3
4	00 0100	04	04	0011 0100	34	- 4
5	00 0101	05	05	0011 0101	35	- 5
6	00 0110	06	06	0011 0110	36	- 6
7	00 0111	07	07	0011 0111	37	- 7
8	00 1000	08	08	0011 1000	38	- 8
9	00 1001	09	09	0011 1001	39	- 9
:	10 1101	0D	55	0011 1010	3A	- 8-5
;	10 1110	2E	56	0011 1011	3B	11 8-6
<	11 1110	3E	76	0011 1100	3C	12 8-6
=	01 1101	1D	35	0011 1101	3D	0 8-5
>	00 1110	0E	16	0011 1110	3E	- 8-6
?	00 0000	00	00	0011 1111	3F	All other card codes
@	00 1100	0C	14	0100 0000	40	- 8-4
A	11 0001	31	61	0100 0001	41	12 1
B	11 0010	32	62	0100 0010	42	12 2
C	11 0011	33	63	0100 0011	43	12 3
D	11 0100	34	64	0100 0100	44	12 4
E	11 0101	35	65	0100 0101	45	12 5
F	11 0110	36	66	0100 0110	46	12 6
G	11 0111	37	67	0100 0111	47	12 7
H	11 1000	38	70	0100 1000	48	12 8
I	11 1001	39	71	0100 1001	49	12 9
J	10 0001	21	41	0100 1010	4A	11 1
K	10 0010	22	42	0100 1011	4B	11 2
L	10 0011	23	43	0100 1100	4C	11 3

HIGH

COLLATING SEQUENCE - BCL TRANSLATED TO USASCII (Cont)

BCL Character	BCL External BA 8421	BCL Hex.	BCL Octal	Translated USASCII Code	USASCII Hex.	Card Code Zone	Card Code Number
M	10 0100	24	44	0100 1101	4D	11	4
N	10 0101	25	45	0100 1110	4E	11	5
O	10 0110	26	46	0100 1111	4F	11	6
P	10 0111	27	47	0101 0000	50	11	7
Q	10 1000	28	50	0101 0001	51	11	8
R	10 1001	29	51	0101 0010	52	11	9
S	01 0010	12	22	0101 0011	53	0	2
T	01 0011	13	23	0101 0100	54	0	3
U	01 0100	14	24	0101 0101	55	0	4
V	01 0101	15	25	0101 0110	56	0	5
W	01 0110	16	26	0101 0111	57	0	6
X	01 0111	17	27	0101 1000	58	0	7
Y	01 1000	18	30	0101 1001	59	0	8
Z	01 1001	19	31	0101 1010	5A	0	9
[	11 1100	3C	74	0101 1011	5B	12	8-4
]	01 1110	1E	36	0101 1101	5D	0	8-6
<	10 1111	2F	57	0101 1110	5E	11	8-7
≠	01 1010	1A	32	0101 1111	5F	0	8-2
x(Mult)	10 1010	2A	52	0111 1101	7D	11	0

LOW ↕ HIGH

COLLATING SEQUENCE - USASCII X3.4-1968 TRANSLATED TO EBCDIC

LOW

USASCII Character	USASCII Hex. Code	Translated EBCDIC Code	EBCDIC Hex. Code
NULL	00	0000 0000	00
SOH	01	0000 0001	01
STX	02	0000 0010	02
ETX	03	0000 0011	03
HT	09	0000 0101	05
DEL	7F	0000 0111	07
VT	0B	0000 1011	0B
FF	0C	0000 1100	0C
CR	0D	0000 1101	0D
SO	0E	0000 1110	0E
SI	0F	0000 1111	0F
DLE	10	0001 0000	10
DC1	11	0001 0001	11
DC2	12	0001 0010	12
DC3	13	0001 0011	13
BS	08	0001 0110	16
CAN	18	0001 1000	18
EM	19	0001 1001	19
FS	1C	0001 1100	1C
GS	1D	0001 1101	1D
RS	1E	0001 1110	1E
US	1F	0001 1111	1F
LF	0A	0010 0101	25
ETB	17	0010 0110	26
ESC	1B	0010 0111	27
ENQ	05	0010 1·101	2D
ACK	06	0010 1110	2E
BEL	07	0010 1111	2F
SYN	16	0011 0010	32
EOT	04	0011 0111	37
DC4	14	0011 1100	3C
NAK	15	0011 1101	3D
SUB	1A	0011 1111	3F
SP	20	0100 0000	40
[	5B	0100 1010	4A
.	2E	0100 1011	4B
<	3C	0100 1100	4C
(	28	0100 1101	4D
+	2B	0100 1110	4E
| (or)	21	0100 1111	4F

USASCII Character	USASCII Hex. Code	Translated EBCDIC Code	EBCDIC Hex. Code
&	26	0101 0000	50
]	5D	0101 1010	5A
$	24	0101 1011	5B
*	2A	0101 1100	5C
)	29	0101 1101	5D
;	3B	0101 1110	5E
^ (¬)	5E	0101 1111	5F
-	2D	0110 0000	60
/	2F	0110 0001	61
¦	7C	0110 1010	6A
,	2C	0110 1011	6B
%	25	0110 1100	6C
_	5F	0110 1101	6D
>	3E	0110 1110	6E
?	3F	0110 1111	6F
`	60	0111 1001	79
:	3A	0111 1010	7A
#	23	0111 1011	7B
@	4C	0111 1100	7C
'	27	0111 1101	7D
=	3D	0111 1110	7E
"	22	0111 1111	7F
a	61	1000 0001	81
b	62	1000 0010	82
c	63	1000 0011	83
d	64	1000 0100	84
e	65	1000 0101	85
f	66	1000 0110	86
g	67	1000 0111	87
h	68	1000 1000	88
i	69	1000 1001	89
j	6A	1001 0001	91
k	6B	1001 0010	92
l	6C	1001 0011	93
m	6D	1001 0100	94
n	6E	1001 0101	95
o	6F	1001 0110	96
p	70	1001 0111	97
q	71	1001 1000	98
r	72	1001 1001	99
~	7D	1010 0001	A1
s	73	1010 0010	A2

HIGH

COLLATING SEQUENCE - USASCII X3.4-1968 TRANSLATED TO EBCDIC (Cont)

USASCII Character	USASCII Hex. Code	Translated EBCDIC Code	EBCDIC Hex. Code	
t	74	1010 0011	A3	LOW
u	75	1010 0100	A4	
v	76	1010 0101	A5	
w	77	1010 0110	A6	
x	78	1010 0111	A7	
y	79	1010 1000	A8	
z	7A	1010 1001	A9	
{	7B	1100 0000	C0	
A	41	1100 0001	C1	
B	42	1100 0010	C2	
C	43	1100 0011	C3	
D	44	1100 0100	C4	
E	45	1100 0101	C5	
F	46	1100 0110	C6	
G	47	1100 0111	C7	
H	48	1100 1000	C8	
I	49	1100 1000	C9	
}	70	1101 0000	D0	
J	4A	1101 0001	D1	
K	4B	1101 0010	D2	
L	4C	1101 0011	D3	
M	4D	1101 0100	D4	
N	4E	1101 0101	D5	
O	4F	1101 0110	D6	
P	50	1101 0111	D7	
Q	51	1101 1000	D8	
R	52	1101 1001	D9	
\	5C	1110 0000	E0	
S	53	1110 0010	E2	
T	54	1110 0011	E3	
U	55	1110 0100	E4	
V	56	1110 0101	E5	
W	57	1110 0110	E6	
X	58	1110 0111	E7	
Y	59	1110 1000	E8	HIGH
Z	5A	1110 1001	E9	
0	30	1111 0000	F0	LOW
1	13	1111 0001	F1	
2	32	1111 0010	F2	
3	33	1111 0011	F2	
4	34	1111 0100	F4	
5	35	1111 0101	F5	
6	36	1111 0110	F6	
7	37	1111 0111	F7	
8	38	1111 1000	F8	HIGH
9	39	1111 1001	F9	

XALGOL COLLATING SEQUENCE (B 5700 BCL)

BCL Character	BCL Octal	BCL Hex	BCL Internal BA 8421	BCL External BA 8421	Card Code Zone	Card Code Number
Blank	60	30	11 0000	01 0000	-	-
.	32	1A	01 1010	11 1011	12	8-3
[	33	1B	01 1011	11 1100	12	8-4
(	35	1D	01 1101	11 1101	12	8-5
<	36	1E	01 1110	11 1110	12	8-6
←	37	1F	01 1111	11 1111	12	8-7
&	34	1C	01 1100	11 0000	12	-
$	52	2A	10 1010	10 1011	11	8-3
*	53	2B	10 1011	10 1100	11	8-4
)	55	2D	10 1101	10 1101	11	8-5
;	56	2E	10 1110	10 1110	11	8-6
≤	57	2F	10 1111	10 1111	11	8-7
-	54	2C	10 1100	10 0000	11	-
/	61	31	11 0001	01 0001	0	1
,	72	3A	11 1010	01 1011	0	8-3
%	73	3B	11 1011	01 1100	0	8-4
=	75	3D	11 1101	01 1101	0	8-5
]	76	3E	11 1110	01 1110	0	8-6
"	77	3F	11 1111	01 1111	0	8-7
#	12	0A	00 1010	00 1011	-	8-3
@	13	0B	00 1011	00 1100	-	8-4
:	15	0D	00 1101	00 1101	-	8-5
>	16	0E	00 1110	00 1110	-	8-6
≥	17	0F	00 1111	00 1111	-	8-7
+	20	10	01 0000	11 1010	12	0
A	21	11	01 0001	11 0001	12	1
B	22	12	01 0010	11 0010	12	2
C	23	13	01 0011	11 0011	12	3
D	24	14	01 0100	11 0100	12	4
E	25	15	01 0101	11 0101	12	5
F	26	16	01 0110	11 0110	12	6
G	27	17	01 0111	11 0111	12	7
H	30	18	01 1000	11 1000	12	8
I	31	19	01 1001	11 1001	12	9
x	40	20	10 0000	10 1010	11	0
J	41	21	10 0001	10 0001	11	1
K	42	22	10 0010	10 0010	11	2
L	43	23	10 0011	10 0011	11	3
M	44	24	10 0100	10 0100	11	4
N	45	25	10 0101	10 0101	11	5

LOW ↑ COLLATING SEQUENCE ↓ HIGH

XALGOL COLLATING SEQUENCE (B 5700 BCL) (Cont)

BCL Character	BCL Octal	BCL Hex	BCL Internal BA 8421	BCL External BA 8421	Card Code Zone	Number
O	46	26	10 0110	10 0110	11	6
P	47	27	10 0111	10 0111	11	7
Q	50	28	10 1000	10 1000	11	8
R	51	29	10 1001	10 1001	11	9
≠	74	3C	11 1100	01 1010	0	8-2
S	62	32	11 0010	01 0010	0	2
T	63	33	11 0011	01 0011	0	3
U	64	34	11 0100	01 0100	0	4
V	65	35	11 0101	01 0101	0	5
W	66	36	11 0110	01 0110	0	6
X	67	37	11 0111	01 0111	0	7
Y	70	38	11 1000	01 1000	0	8
Z	71	39	11 1001	01 1001	0	9
0	00	00	00 0000	00 1010	-	0
1	01	01	00 0001	00 0001	-	1
2	02	02	00 0010	00 0010	-	2
3	03	03	00 0011	00 0011	-	3
4	04	04	00 0100	00 0100	-	4
5	05	05	00 0101	00 0101	-	5
6	06	06	00 0110	00 0110	-	6
7	07	07	00 0111	00 0111	-	7
8	10	08	00 1000	00 1000	-	8
9	11	09	00 1001	00 1001	-	9
?	14	0C	00 1100	00 0000	ALL OTHER CARD CODES	

LOW ↕ HIGH

FORTRAN BCD COLLATING SEQUENCE

BCD Character	Internal Representation		Internal Translation		Card	
	Hex	Binary	Binary	Hex	Zone	Code Number
. (period)	1A	01 1010	0100 1011	4B	12	8-3
)	1B	01 1011	0100 1100	4C	12	8-4
+	1C	01 1100	0101 0000	50	12	
$	2A	10 1010	0101 1011	5B	11	8-3
*	2B	10 1011	0101 1100	5C	11	8-4
;	2E	10 1110	0101 1110	5E	11	8-6
≤	2F	10 1111	0101 1111	5F	11	8-7
- (minus)	2C	10 1100	0110 0000	60	11	
/	31	11 0001	0110 0001	61	0	1
, (comma)	3A	11 1010	0110 1011	6B	0	8-3
(	3B	11 1011	0110 1100	6C	0	8-4
≠	3D	11 1101	0110 1101	6D	0	8-5
>	3E	11 1110	0110 1110	6E	0	8-6
?	3F	11 1111	0110 1111	6F	0	8-7
=	0A	00 1010	0111 1011	7B		8-3
@	0B	00 1011	0111 1100	7C		8-4
≥	0D	00 1101	0111 1101	7D		8-5
÷	0E	00 1110	0111 1110	7E		8-6
"	0F	00 1111	0111 1111	7E		8-7
A	11	01 0001	1100 0001	C1	12	1
B	12	01 0010	1100 0010	C2	12	2
C	13	01 0011	1100 0011	C3	12	3
D	14	01 0100	1100 0100	C4	12	4
E	15	01 0101	1100 0101	C5	12	5
F	16	01 0110	1100 0110	C6	12	6
G	17	01 0111	1100 0111	C7	12	7
H	18	01 1000	1100 1000	C8	12	8
I	19	01 1001	1100 1001	C9	12	9
J	21	10 0001	1101 0001	D1	11	1
K	22	10 0010	1101 0010	D2	11	2
L	23	10 0011	1101 0011	D3	11	3
M	24	10 0100	1101 0100	D4	11	4
N	25	10 0101	1100 0101	D5	11	5
O	26	10 0110	1101 0110	D6	11	6
P	27	10 0111	1101 0111	D7	11	7
Q	28	10 1000	1101 1000	D8	11	8
R	29	10 1001	1101 1001	D9	11	9
S	31	11 0010	1110 0010	E2	0	2
T	33	11 0011	1110 0011	E3	0	3
U	34	11 0100	1110 0100	E4	0	4
V	35	11 0101	1110 0101	E5	0	5
W	36	11 0110	1110 0110	E6	0	6
X	37	11 0111	1110 0111	E7	0	7
Y	38	11 1000	1110 1000	E8	0	8
Z	39	11 1001	1110 1001	E9	0	9
0	00	00 0000	1111 0000	F0		0
1	01	00 0001	1111 0001	F1		1
2	02	00 0010	1111 0010	F2		2
3	03	00 0011	1111 0011	F3		3
4	04	00 0100	1111 0100	F4		4
5	05	05 0101	1111 0101	F5		5
6	06	00 0110	1111 0110	F6		6
7	07	00 0111	1111 0111	F7		7
8	08	00 1000	1111 1000	F8		8
9	09	00 1001	1111 1001	F9		9

Low

High

EXTENDED BINARY CODED DECIMAL INTERCHANGE CODES (EBCDIC)

BITS b8 b7 b6 b5 →					0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
b4	b3	b2	b1	COL → ↓ ROW	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0	0	0	0	0	NUL 12-0-9-8-1	DLE 12-11-9-8-1	11-0-9-8-1	12-11-0-9-8-1	SP BLANK	& 12	– 11	12-11-0	12-0-8-1	12-11-8-1	11-0-8-1	12-11-0-8-1	PZ(+) 12-0	MZ (!) 11-0	\(CR)(¢) 0-8-2	0 0
0	0	0	1	1	SOH 12-9-1	DC1 11-9-1	0-9-1	9-1	12-0-9-1	12-11-9-1	/ 0-1	12-11-0-9-1	a 12-0-1	j 12-11-1	11-0-1	12-11-0-1	A 12-1	J 11-1	11-0-9-1	1 1
0	0	1	0	2	STX 12-9-2	DC2 11-9-2	0-9-2	SYN 9-2	12-0-9-2	12-11-9-2	11-0-9-2	12-11-0-9-2	b 12-0-2	k 12-11-2	s 11-0-2	12-11-0-2	B 12-2	K 11-2	S 0-2	2 2
0	0	1	1	3	ETX 12-9-3	DC3 11-9-3	0-9-3	9.3	12-0-9-3	12-11-9-3	11-0-9-3	12-11-0-9-3	c 12-0-3	l 12-11-3	t 11-0-3	12-11-0-3	C 12-3	L 11.3	T 0-3	3 3
0	1	0	0	4	12-9-4	11-9-4	0-9-4	9-4	12-0-9-4	12-11-9-4	11-0-9-4	12-11-0-9-4	d 12-0-4	m 12-11-4	u 11-0-4	12-11-0-4	D 12-4	M 11-4	U 0-4	4 4
0	1	0	1	5	HT 12-9-5	NL 11-9-5	LF 0-9-5	9-5	12-0-9-5	12-11-9-5	11-0-9-5	12-11-0-9-5	e 12-0-5	n 12-11-5	v 11-0-5	12-11-0-5	E 12-5	N 11-5	V 0-5	5 5
0	1	1	0	6	12-9-6	BS 11-9-6	ETB 0-9-6	9-6	12-0-9-6	12-11-9-6	11-0-9-6	12-11-0-9-6	f 12-0-6	o 12-11-6	w 11-0-6	12-11-0-6	F 12-6	O 11-6	W 0-6	6 6
0	1	1	1	7	DEL 12-9-7	11-9-7	ESC 0-9-7	EOT 9-7	12-0-9-7	12-11-9-7	11-0-9-7	12-11-0-9-7	g 12-0-7	p 12-11-7	x 11-0-7	12-11-0-7	G 12-7	P 11-7	X 0-7	7 7
1	0	0	0	8	12-9-8	CAN 11-9-8	0-9-8	9-8	12-0-9-8	— 12-11-9-8	11-0-9-8	12-11-0-9-8	h 12-0-8	q 12-11-8	y 11-0-8	12-11-0-8	H 12-8	Q 11-8	Y 0.8	8 8
1	0	0	1	9	12-9-8-1	EM 11-9-8-1	0-9-8-1	9-8-1	12-8-1	11-8-1	0-8-1	8-1	i 12-0-9	r 12-11-9	z 11-0-9	12-11-0-9	I 12-9	R 11-9	Z 0-9	9 9
1	0	1	0	10 (A)	12-9-8-2	11-9-8-2	0-9-8-2	9-8-2	[12-8-2	] 11-8-2	12-11	: 8-2	12-0-8-2	12-11-8-2	11-0-8-2	12-11-0-8-2	12-0-9-8-2	12-11-9-8-2	11-0-9-8-2	12-11-0-9-8-2
1	0	1	1	11 (B)	VT 12-9-8-3	11-9-8-3	0-9-8-3	9-8-3	12-8-3	$ 11-8-3	0-8-3	# 8-3	12-0-8-3	12-11-8-3	11-0-8-3	12-11-0-8-3	12-0-9-8-3	12-11-9-8-3	11-0-9-8-3	12-11-0-9-8-3
1	1	0	0	12 (C)	FF 12-9-8-4	FS 11-9-8-4	0-9-8-4	DC4 9-8-4	< 12-8-4	* 11-8-4	% 0-8-4	@ 8-4	12-0-8-4	12-11-8-4	11-0-8-4	12-11-0-8-4	12-0-9-8-4	12-11-9-8-4	11-0-9-8-4	12-11-0-9-8-4
1	1	0	1	13 (D)	CR 12-9-8-5	GS 11-9-8-5	ENQ 0-9-8-5	NAK 9-8-5	(12-8-5	) 11-8-5	(US)(≠) 0-8-5	'(≥) 8-5	12-0-8-5	12-11-8-5	11-0-8-5	12-11-0-8-5	12-0-9-8-5	12-11-9-8-5	11-0-9-8-5	12-11-0-9-8-6
1	1	1	0	14 (E)	SO 12-9-8-6	RS 11-9-8-6	ACK 0-9-8-6	9-8-6	+ 12-8-6	; 11-8-6	> 0-8-6	= 8-6	12-0-8-6	12-11-8-6	11-0-8-6	12-11-0-8-6	12-0-9-8-6	12-11-9-8-6	11-0-9-8-6	12-11-0-9-8-6
1	1	1	1	15 (F)	SI 12-9-8-7	US 11-9-8-7	BEL 0-9-8-7	SUB 9-8-7	|(←) 12-8-7	¬(≤) 11-8-7	? 0-8-7	" 8-7	12-0-8-7	12-11-8-7	11-0-8-7	12-11-0-8-7	DELIMETER 12-0-9-8-7	12-11-9-8-7	11-0-9-8-7	12-11-0-9-8-7

*INTERNAL COLLATING SEQUENCE – 0 0 0 0/0 0 0 0 TO 1111/1111
b8 b7 b6 b5 b4 b3 b2 b1

APPENDIX F

DATA REPRESENTATION

EBCDIC GRAPHIC	BCL	DECIMAL VALUE	EBCDIC INTERNAL	HEX. GRAPHIC	EBCDIC CARD CODE	BCL CARD CODE	OCTAL	BCL INTERNAL	BCL EXTERNAL
BLANK		64	0100 0000	40	No Punches	No Punches	60	11 0000	01 0000
[		74	0100 1010	4A	12 8 2	12 8 4	33	01 1011	11 1100
.		75	0100 1011	4B	12 8 3	12 8 3	32	01 1010	11 1011
<		76	0100 1100	4C	12 8 4	12 8 6	36	01 1110	11 1110
(		77	0100 1101	4D	12 8 5	12 8 5	35	01 1101	11 1101
+		78	0100 1110	4E	12 8 6				11 1010
¦	←	79	0100 1111	4F	12 8 7	12 8 7	37	01 1111	11 1111
&		80	0101 0000	50	12	12	34	01 1100	11 0000
]		90	0101 1010	5A	11 8 2	0 8 6	76	11 1110	01 1110
$		91	0101 1011	5B	11 8 3	11 8 3	52	10 1010	10 1011
*		92	0101 1100	5C	11 8 4	11 8 4	53	10 1011	10 1100
)		93	0101 1101	5D	11 8 5	11 8 5	55	10 1101	10 1101
;		94	0101 1110	5E	11 8 6	11 8 6	56	10 1110	10 1110
¬	≤	95	0101 1111	5F	11 8 7	11 8 7	57	10 1111	10 1111
-		96	0110 0000	60	11	11	54	10 1100	10 0000
/		97	0110 0001	61	0 1	0 1	61	11 0001	01 0001
,		107	0110 1011	6B	0 8 3	0 8 3	72	11 1010	01 1011
%		108	0110 1100	6C	0 8 4	0 8 4	73	11 1011	01 1100
—	≠	109	0110 1101	6D	0 8 5	0 8 2	74	11 1100	01 1010
>		110	0110 1110	6E	0 8 6	8 6	16	00 1110	00 1110
?		111	0110 1111	6F	0 8 7	*	14	00 1100	00 0000
:		122	0111 1010	7A	8 2	8 5	15	00 1101	00 1101
#		123	0111 1011	7B	8 3	8 3	12	00 1010	00 1011
@		124	0111 1100	7C	8 4	8 4	13	00 1011	00 1100
'	≥	125	0111 1101	7D	8 5	8 7	17	00 1111	00 1111
=		126	0111 1110	7E	8 6	0 8 5	75	11 1101	01 1101
"		127	0111 1111	7F	8 7	0 8 7	77	11 1111	01 1111
(+)PZ	+	192	1100 0000	CO	12 0	12 0	20	01 0000	11 1010
A		193	1100 0001	C1	12 1	12 1	21	01 0001	11 0001
B		194	1100 0010	C2	12 2	12 2	22	01 0010	11 0010
C		195	1100 0011	C3	12 3	12 3	23	01 0011	11 0011
D		196	1100 0100	C4	12 4	12 4	24	01 0100	11 0100
E		197	1100 0101	C5	12 5	12 5	25	01 0101	11 0101
F		198	1100 0110	C6	12 6	12 6	26	01 0110	11 0110
G		199	1100 0111	C7	12 7	12 7	27	01 0111	11 0111
H		200	1100 1000	C8	12 8	12 8	30	01 1000	11 1000
I		201	1100 1001	C9	12 9	12 9	31	01 1001	11 1001
(!)MZ	MULT x	208	1101 0000	D0	11 0	11 0	40	10 0000	10 1010
J		209	1101 0001	D1	11 1	11 1	41	10 0001	10 0001
K		210	1101 0010	D2	11 2	11 2	42	10 0010	10 0010
L		211	1101 0011	D3	11 3	11 3	43	10 0011	10 0011
M		212	1101 0100	D4	11 4	11 4	44	10 0100	10 0100
N		213	1101 0101	D5	11 5	11 5	45	10 0101	10 0101
O		214	1101 0110	D6	11 6	11 6	46	10 0110	10 0110
P		215	1101 0111	D7	11 7	11 7	47	10 0111	10 0111

*All other codes

DATA REPRESENTATION

EBCDIC GRAPHIC	BCL	DECIMAL VALUE	EBCDIC INTERNAL	HEX. GRAPHIC	EBCDIC CARD CODE	BCL CARD CODE	OCTAL	BCL INTERNAL	BCL EXTERNAL
Q		216	1101 1000	D8	11 8	11 8	50	10 1000	10 1000
R		217	1101 1001	D9	11 9	11 9	51	10 1001	10 1001
¢		224	1110 0000	E0	0 8 2				00 0000
S		226	1110 0010	E2	0 2	0 2	62	11 0010	01 0010
T		227	1110 0011	E3	0 3	0 3	63	11 0011	01 0011
U		228	1110 0100	E4	0 4	0 4	64	11 0100	01 0100
V		229	1110 0101	E5	0 5	0 5	65	11 0101	01 0101
W		230	1110 0110	E6	0 6	0 6	66	11 0110	01 0110
X		231	1110 0111	E7	0 7	0 7	67	11 0111	01 0111
Y		232	1110 1000	E8	0 8	0 8	70	11 1000	01 1000
Z		233	1110 1001	E9	0 9	0 9	71	11 1001	01 1001
0		240	1111 0000	F0	0	0	00	00 0000	00 1010
1		241	1111 0001	F1	1	1	01	00 0001	00 0001
2		242	1111 0010	F2	2	2	02	00 0010	00 0010
3		243	1111 0011	F3	3	3	03	00 0011	00 0011
4		244	1111 0100	F4	4	4	04	00 0100	00 0100
5		245	1111 0101	F5	5	5	05	00 0101	00 0101
6		246	1111 0110	F6	6	6	06	00 0110	00 0110
7		247	1111 0111	F7	7	7	07	00 0111	00 0111
8		248	1111 1000	F8	8	8	10	00 1000	00 1000
9		249	1111 1001	F9	9	9	11	00 1001	00 1001

NOTES

1. EBCDIC 0100 1110 also translates to BCL 11 1010.
2. EBCDIC 1100 1111 is translated to BCL 00 0000 with an additional flag bit on the most significant bit line (8th bit). This function is used by the unbuffered printer to stop scanning.
3. EBCDIC 1110 0000 is translated to BCL 00 0000 with an additional flag bit on the next to most significant bit line (7th bit). As the print drums have 64 graphics and space this signal can be used to print the 64th graphic. The 64th graphic is a "CR" for BCL drums and a "¢" for EBCDIC drums.
4. The remaining 189 EBCDIC codes are translated to BCL 00 0000 (? code).
5. The EBCDIC graphics and BCL graphics are the same except as follows:

BCL	EBCDIC
≥	' (single quote)
x (multiply)	!
≤	¬ (not)
≠	_ (underscore)
←	

APPENDIX G

PROCESSOR OPERATORS, BY HEXADECIMAL CODE

MODE ID& HEX CODE	OPERATOR NAME	MNEMONIC
(P)00 → 3F	VALUE CALL	VALC
(Z)00 → 3F	VECTOR FETCH	FETCH
(P)40 → 7F	NAME CALL	NAMC
(Z)40 → 7F	VECTOR STORE	STOR
(V)42	SET TWO SINGLES TO DOUBLE	JOIN
(V)43	SET DOUBLE TO TWO SINGLES	SPLT
(V)44	IDLE UNTIL INTERRUPT	IDLE
(V)45	SET INTERVAL TIMER	SINT
(V)46	ENABLE EXTERNAL INTERRUPTS	EEXI
(V)47	DISABLE EXTERNAL INTERRUPTS	DEXI
(V)48	IGNORE PARITY	IGPR
(V)4A	SCAN IN	SCNI
(V)4E	READ PROCESSOR IDENTIFICATION	WHOI
(P)80	ADD	ADD
(P)81	SUBTRACT	SUBT
(P)82	MULTIPLY	MULT
(P)83	DIVIDE	DIVD
(P)84	INTEGER DIVIDE	IDV
(V)84	PAUSE UNTIL INTERRUPT	PAUS
(P)85	REMAINDER DIVIDE	RDIV
(V)85	OCCURS INDEX	OCRX
(P)86	INTEGERIZE, TRUNCATED	NTIA
(P)87	INTEGERIZE, ROUNDED	NTGR
(V)87	INTEGERIZE, ROUNDED, DOUBLE PRECISION	NTGD
(P)88	LESS THAN	LESS
(P)89	GREATER THAN OR EQUAL	GREQ
(P)8A	GREATER THAN	GRTR
(P)8B	LESS THAN OR EQUAL	LSEQ
(V)8B	LEADING ONE TEST	LOG2
(P)	EQUAL	EQUL
(P)8D	NOT EQUAL	NEQL
(P)8E	CHANGE SIGN BIT	CHSN
(P)8F	EXTENDED MULTIPLE	MULX
(V)8F	INTERRUPT CHANNEL N	INCN
(P)90	LOGICAL AND	LAND
(P)91	LOGICAL OR	LOR
(P)92	LOGICAL NEGATE	LNOT
(P)93	LOGICAL EQUIVALENCE	LEQV
(P)94	LOGICAL EQUAL	SAME
(P)95	ESCAPE TO 16-BIT INSTRUCTION	VARI

MODE ID & HEX CODE	OPERATOR NAME	MNEMONIC
(P)96	BIT SET	BSET
(P)97	DYNAMIC BIT SET	DBST
(P)98	FIELD TRANSFER	FLTR
(P)99	DYNAMIC FIELD TRANSFER	DFTR
(P)9A	FIELD ISOLATE	ISOL
(P)9B	DYNAMIC FIELD ISOLATE	DISO
(P)9C	FIELD INSERT	INSR
(P)9D	DYNAMIC FIELD INSERT	DINS
(P)9E	BIT RESET	BRST
(P)9F	DYNAMIC BIT RESET	DBRS
(P)A0	BRANCH FALSE	BRFL
(P)A1	BRANCH TRUE	BRTR
(P)A2	BRANCH UNCONDITIONAL	BRUN
(P)A3	EXIT	EXIT
(P)A4	STEP AND BRANCH	STBR
(P)A5	INDEX AND LOAD NAME	NXLN
(P)A6	INDEX	INDX
(P)A7	RETURN	RETN
(P)A8	DYNAMIC BRANCH FALSE	DBFL
(V)A8	SET MEMORY INHIBITS	SINH
(P)A9	DYNAMIC BRANCH TRUE	DBTR
(P)AA	DYNAMIC BRANCH UNCONDITIONAL	DBUN
(V)AA	SET MEMORY LIMITS	SLMT
(P)AB	ENTER	ENTR
(P)AC	EVALUATE DESCRIPTOR	EVAL
(V)AC	FETCH MEMORY FAIL	FMFR
(P)AD	INDEX AND LOAD VALUE	NXLV
(P)AE	MARK STACK	MKST
(P)AF	STUFF ENVIRONMENT	STFF
(V)AF	MOVE TO STACK	MVST
(P)B0	LIT CALL ZERO	ZERO
(P)B1	LIT CALL ONE	ONE
(P)B2	LIT CALL 8 BITS	LT8
(P)B3	LIT CALL 16 BITS	LT16
(P)B4	PUSH DOWN STACK REGISTERS	PUSH
(V)B4	SET TAG FIELD	STAG
(P)B5	DELETE TOP OF STACK	DLET
(V)B5	READ TAG FIELD	RTAG
(P)B6	EXCHANGE	EXCH
(V)B6	ROTATE STACK UP	RSUP
(P)B7	DUPLICATE TOP OF STACK	DUPL

APPENDIX G (Cont)

MODE ID & HEX CODE	OPERATOR NAME	MNEMONIC
(V)B7	ROTATE STACK DOWN	RSDN
(P)B8	STORE DESTRUCTIVE	STOD
(V)B8	READ PROCESSOR REGISTER	RPRR
(P)B9	STORE NON-DESTRUCTIVE	STON
(V)B9	SET PROCESSOR REGISTER	SPRR
(P)BA	OVERWRITE DESTRUCTIVE	OVRD
(V)BA	READ WITH LOCK	RDLK
(P)BB	OVERWRITE NON-DESTRUCTIVE	OVRN
(V)BB	COUNT BINARY ONES	CBON
(V)BC	LOAD TRANSPARENT	LODT
(P)BD	LOAD	LOAD
(V)BD	LINKED LIST LOOKUP	LLLU
(P)BE	LIT CALL 48 BITS	LT48
(V)BE	MASKED SEARCH FOR EQUAL	SRCH
(P)BF	MAKE PROGRAM CONTROL WORD	MPCW
(V)BF	STOP	STOP
(P)C0	SCALE LEFT	SCLF
(P)C1	DYNAMIC SCALE LEFT	DSLF
(P)C2	SCALE RIGHT TRUNCATE	SCRT
(P)C3	DYNAMIC SCALE RIGHT TRUNCATE	DSRT
(P)C4	SCALE RIGHT SAVE	SCRS
(P)C5	DYNAMIC SCALE RIGHT SAVE	DSRS
(P)C6	SCALE RIGHT FINAL	SCRF
(P)C7	DYNAMIC SCALE RIGHT FINAL	DSRF
(P)C8	SCALE RIGHT ROUND	SCRR
(P)C9	DYNAMIC SCALE RIGHT ROUND	DSRR
(P)CA	INPUT CONVERT, DESTRUCTIVE	ICVD
(P)CB	INPUT CONVERT, UPDATE	ICVU
(P)CC	SET TO SINGLE PRECISION TRUNCATE	SNGT
(P)CD	SET TO SINGLE PRECISION, ROUNDED	SNGL
(P)CE	SET TO DOUBLE PRECISION	XTND
(P)CF	INSERT MARK STACK	IMKS
(E)D0	MOVE WITH INSERT	MINS
(P)D0	TABLE ENTER EDIT, DESTRUCTIVE	TEED
(V)D0	UNPACK SIGNED, DESTRUCTIVE	USND
(E)D1	MOVE WITH FLOAT	MFLT
(P)D1	PACK DESTRUCTIVE	PACD
(V)D1	UNPACK ABSOLUTE, DESTRUCTIVE	UABD
(E)D2	SKIP FORWARD SOURCE CHARACTERS	SFSC
(P)D2	EXECUTE SINGLE MICRO, DESTRUCTIVE	EXSD
(V)D2	TRANSFER WHILE FALSE, DESTRUCTIVE	TWFD
(E)D3	SKIP REVERSE SOURCE CHARACTERS	SRSC
(P)D3	TRANSFER WORDS, DESTRUCTIVE	TWSD
(V)D3	TRANSFER WHILE TRUE, DESTRUCTIVE	TWTD
(E)D4	RESET FLOAT	RSTF
(P)D4	TRANSFER WORDS, OVERWRITE DESTRUCTIVE	TWOD
(V)D4	SCAN WHILE FALSE, DESTRUCTIVE	SWFD
(E)D5	END FLOAT	ENDF
(P)D5	STRING ISOLATE	SISO
(V)D5	SCAN WHILE TRUE, DESTRUCTIVE	SWTD
	(E)D6 MOVE NUMERIC UNCONDITIONAL	MVNU
(P)D6	SET EXTERNAL SIGN	SXSN
(E)D7	MOVE CHARACTERS	MCHR
(P)D7	READ AND CLEAR OVERFLOW FLIP-FLOP	ROFF
(V)D7	TRANSLATE	TRNS
(E)D8	INSERT OVERPUNCH	INOP
(P)D8	TABLE ENTER EDIT, UPDATE	TEEU
(V)D8	UNPACK SIGNED UPDATE	USNU
(E)D9	INSERT DISPLAY SIGN	INSG
(P)D9	PACK UPDATE	PACU
(V)D9	UNPACK ABSOLUTE, UPDATE	UABU
(E)DA	SKIP FORWARD DESTINATION CHARACTERS	SFDC
(P)DA	EXECUTE SINGLE MICRO, UPDATE	EXSU
(V)DA	TRANSFER WHILE FALSE, UPDATE	TWFU
(E)DB	SKIP REVERSE DESTINATION CHARACTERS	SRDC
(P)DB	TRANSFER WORDS, UPDATE	TWSU
(V)DB	TRANSFER WHILE TRUE, UPDATE	TWTU
(E)DC	INSERT UNCONDITIONAL	INSU
(P)DC	TRANSFER WORDS OVERWRITE UPDATE	TWOU
(V)DC	SCAN WHILE FALSE, UPDATE	SWFU
(E)DD	INSERT CONDITIONAL	INSC
(P)DD	EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE	EXPU
(V)DD	SCAN WHILE TRUE, UPDATE	SWTU
(E)DE	END EDIT	ENDE

APPENDIX G (Cont)

MODE ID & HEX CODE	OPERATOR NAME	MNEMONIC
(P)DE	READ TRUE/FALSE FLIP-FLOP	RTFF
(E)DF	CONDITIONAL HALT	HALT
(P)DF	CONDITIONAL HALT	HALT
(V)DF	CONDITIONAL HALT	HALT
(P)E0	TRANSFER WHILE LESS, DESTRUCTIVE	TLSD
(Z)E0	LOAD A	LDA
(P)E1	TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE	TGED
(Z)E1	LOAD A INCREMENT	LDAI
(P)E2	TRANSFER WHILE GREATER, DESTRUCTIVE	TGTD
(Z)E2	LOAD B	LDB
(P)E3	TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE	TLED
(Z)E3	LOAD B INCREMENT	LDBI
(P)E4	TRANSFER WHILE EQUAL, DESTRUCTIVE	TEQD
(Z)E4	LOAD C	LDC
(P)E5	TRANSFER WHILE NOT EQUAL, DESTRUCTIVE	TNED
(Z)E5	LOAD C INCREMENT	LDCI
(P)E6	TRANSFER UNCONDITIONAL, DESTRUCTIVE	TUND
(Z)E6	VECTOR EXIT	VXIT
(P)E7	MULTIPLE-WORD VECTOR MODE	VMOM
(Z)E7	MULTIPLE-WORD VECTOR MODE	VMOM
(P)E8	TRANSFER WHILE LESS, UPDATE	TLSU
(Z)E8	DOUBLE LOAD A	DLA
(P)E9	TRANSFER WHILE GREATER OR EQUAL, UPDATE	TGEU
(Z)E9	DOUBLE LOAD A INCREMENT	DLAI
(P)EA	TRANSFER WHILE GREATER, UPDATE	TGTU
(Z)EA	DOUBLE LOAD B	DLB
(P)EB	TRANSFER WHILE LESS OR EQUAL, UPDATE	TLEU
(Z)EB	DOUBLE LOAD B INCREMENT	DLBI
(P)EC	TRANSFER WHILE EQUAL, UPDATE	TEQU
(P)ED	TRANSFER WHILE NOT EQUAL, UPDATE	TNEU
(Z)ED	DOUBLE LOAD C INCREMENT	DLCI
(P)EE	TRANSFER UNCONDITIONAL, UPDATE	TUNU
(Z)EE	VECTOR BRANCH	VEBR
(P)EF	SINGLE-WORD VECTOR MODE	VMOS
(Z)EF	SINGLE-WORD VECTOR MODE	VMOS
(P)F0	COMPARE CHARACTERS LESS, DESTRUCTIVE	CLSD
(V)F0	SCAN WHILE LESS, DESTRUCTIVE	SLSD
(Z)F0	STORE A	STA
(P)F1	COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE	CGED
(V)F1	SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE	SGED
(Z)F1	STORE A INCREMENT	STAI
(P)F2	COMPARE CHARACTERS GREATER, DESTRUCTIVE	CGTD
(V)F2	SCAN WHILE GREATER, DESTRUCTIVE	SGTD
(Z)F2	STORE B	STB
(P)F3	COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE	CLED
(V)F3	SCAN WHILE LESS OR EQUAL, DESTRUCTIVE	SLED
(Z)F3	STORE B INCREMENT	STBI
(P)F4	COMPARE CHARACTERS EQUAL, DESTRUCTIVE	CEQD
(V)F4	SCAN WHILE EQUAL, DESTRUCTIVE	SEQD
(Z)F4	STORE C	STC
(P)F5	COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE	CNED
(V)F5	SCAN WHILE NOT EQUAL, DESTRUCTIVE	SNED
(Z)F5	STORE C INCREMENT	STCI
(P)F8	COMPARE CHARACTERS LESS, UPDATE	CLSU
(V)F8	SCAN WHILE LESS, UPDATE	SLSU
(Z)F8	DOUBLE STORE A	DSA
(P)F9	COMPARE CHARACTERS GREATER OR EQUAL, UPDATE	CGEU
(V)F9	SCAN WHILE GREATER OR EQUAL, UPDATE	SGEU
(Z)F9	DOUBLE STORE A INCREMENT	DSAI
(P)FA	COMPARE CHARACTERS GREATER, UPDATE	CGTU
(V)FA	SCAN WHILE GREATER, UPDATE	SGTU
(Z)FA	DOUBLE STORE B	DSB
(P)FB	COMPARE CHARACTERS LESS OR EQUAL, UPDATE	CLEU
(V)FB	SCAN WHILE LESS OR EQUAL, UPDATE	SLEU
(Z)FB	DOUBLE STORE B INCREMENT	DSBI

APPENDIX G (Cont)

MODE ID & HEX CODE	OPERATOR NAME	MNEMONIC
(P)FC	COMPARE CHARACTERS EQUAL, UPDATE	CEQU
(V)FC	SCAN WHILE EQUAL, UPDATE	SEQU
(Z)FC	DOUBLE STORE C	DSC
(P)FD	COMPARE CHARACTERS NOT EQUAL, UPDATE	CNEU
(V)FD	SCAN WHILE NOT EQUAL, UPDATE	SNEU
(Z)FD	DOUBLE STORE C INCREMENT	DSCI
(E)FE	NO OPERATION	NOOP
(P)FE	NO OPERATION	NOOP
(V)FE	NO OPERATION	NOOP
(Z)FE	NO OPERATION	NOOP
(E)FF	INVALID OPERATION	NVLD
(P)FF	INVALID OPERATION	NVLD
(V)FF	INVALID OPERATION	NVLD
(Z)FF	INVALID OPERATION	NVLD

APPENDIX H

PROCESSOR OPERATORS BY MNEMONICS

MNEMONIC	MODE ID & HEX CODE	OPERATOR NAME	PAGE
ADD	(P)80	ADD	3-69
BRFL	(P)A0	BRANCH ON FALSE	3-70
BRST	(P)9E	BIT RESET	3-70
BRTR	(P)A1	BRANCH ON TRUE	3-70
BRUN	(P)A2	BRANCH UNCONDITIONAI	3-70
BSET	(P)96	BIT SET	3-70
CBON	(V)BB	COUNT BINARY ONES	3-70
CEQD	(P)F4	COMPARE CHARACTERS EQUAL, DESTRUCTIVE	3-72
CEQU	(P)FC	COMPARE CHARACTERS EQUAL, UPDATE	3-72
CGED	(P)F1	COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE	3-72
CGEU	(P)F9	COMPARE CHARACTERS GREATER OR EQUAL, UPDATE	3-72
CGTD	(P)F2	COMPARE CHARACTERS GREATER, DESTRUCTIVE	3-72
CGTU	(P)FA	COMPARE CHARACTERS GREATER, UPDATE	3-71
CHSN	(P)8E	CHANGE SIGN BIT	3-70
CLED	(P)F3	COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE	3-72
CLEU	(P)FB	COMPARE CHARACTERS LESS OR EQUAL, UPDATE	3-72
CLSD	(P)F0	COMPARE CHARACTERS LESS, DESTRUCTIVE	3-72
CLSU	(P)F8	COMPARE CHARACTERS LESS, UPDATE	3-72
CNEU	(P)FD	COMPARE CHARACTERS NOT EQUAL, UPDATE	3-72
DBFL	(P)A8	DYNAMIC BRANCH FALSE	3-71
DBRS	(P)9F	DYNAMIC BIT RESET	3-70

MNEMONIC	MODE ID & HEX CODE	OPERATOR NAME	PAGE
DBST	(P)97	DYNAMIC BIT SET	3-70
DBTR	(P)A9	DYNAMIC BRANCH TRUE	3-71
DBUN	(P)AA	DYNAMIC BRANCH UNCONDITIONAI	3-70
DEXI	(V)47	DISABLE EXTERNAL INTERRUPTS	3-98
DFTR	(P)99	DYNAMIC FIELD TRANSFER	3-88
DINS	(P)9D	DYNAMIC FIELD INSERT	3-96
DISO	(P)9B	DYNAMIC FIELD ISOLATE	3-88
DIVD	(P)83	DIVIDE	3-69
DLA	(Z)E8	DOUBLE LOAD A	3-105
DLAI	(Z)E9	DOUBLE LOAD A INCREMENT	3-105
DLB	(Z)EA	DOUBLE LOAD B	3-105
DLBI	EB	DOUBLE LOAD B INCREMENT	3-105
DLC	(Z)EC	DOUBLE LOAD C	3-105
DLCI	(Z)ED	DOUBLE LOAD C INCREMENT	3-105
DLET	(P)B5	DELETE TOP OF STACK	3-81
DSA	(Z)F8	DOUBLE STORE A	3-105
DSAI	(Z)F9	DOUBLE STORE A INCREMENT	3-106
DSB	(Z)FA	DOUBLE STORE B	3-106
DSBI	(Z)FB	DOUBLE STORE B INCREMENT	3-106
DSC	(Z)FC	DOUBLE STORE C	3-106
DSCI	(Z)FD	DOUBLE STORE C INCREMENT	3-106
DSLF	(P)C1	DYNAMIC SCALE LEFT	3-80
DSRF	(P)C7	DYNAMIC SCALE RIGHT FINAL	3-80
DSRR	(P)C9	DYNAMIC SCALE RIGHT ROUNDED	3-80
DSRS	(P)C5	DYNAMIC SCALE RIGHT SAVE	3-80
DSRT	(P)C3	DYNAMIC SCALE RIGHT TRUNCATE	3-80

APPENDIX H (Cont)

MNEMONIC	MODE ID & HEX CODE	OPERATOR NAME	PAGE
DUPL	(P)B7	DUPLICATE TOP OF STACK	3-81
EEXI	(V)46	ENABLE EXTERNAL INTERRUPTS	3-98
ENDE	(E)DE	END EDIT	3-104
ENDF	(E)D5	END FLOAT	3-103
ENTR	(P)AB	ENTER	3-88
EQUL	(P)8C	EQUAL	3-79
EVAL	(P)AC	EVALUATE DESCRIPTOR	3-85
EXCH	(P)B6	EXCHANGE	3-80
EXIT		(P)A3 EXIT	3-88
EXPU	(P)DD	EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE	3-73
EXSD	(P)D2	EXECUTE SINGLE MICRO, DESTRUCTIVE	3-73
EXSU	(P)DA	EXECUTE SINGLE MICRO, UPDATE	3-73
FLTR	(P)98	FIELD TRANSFER	3-88
FMFR	(V)AC	FETCH MEMORY FAIL REGISTER	3-101
FTCH	(Z)00 → 3F	VECTOR FETCH	3-106
GREQ	(P)89	GREATER THAN OR EQUAL	3-79
GRTR	(P)8A	GREATER THAN	3-79
HALT	(P)DF	CONDITIONAL HALT	3-96
HALT	(V)DF	CONDITIONAL HALT	3-96
HALT	(E)DF	CONDITIONAL HALT	3-96
ICVD	(P)CA	INPUT CONVERT, DESTRUCTIVE	3-77
ICVU	(P)CB	INPUT CONVERT, UPDATE	3-77
IDIV	(P)84	INTEGER DIVIDE	3-69
IDLE	(V)44	IDLE UNTIL INTERRUPT	3-98
IGPR	(V)48	IGNORE PARITY	3-101
IMKS	(P)CF	INSERT MARK STACK	3-85
INCN	(V)8F	INTERRUPT CHANNEL N	3-102
INDX	(P)A6	INDEX	3-76
INOP	(E)D8	INSERT OVERPUNCH	3-102

MNEMONIC	MODE ID & HEX CODE	OPERATOR NAME	PAGE
INSC	(E)DD	INSERT CONDITIONAL	3-102
INSG	(E)D9	INSERT DISPLAY SIGN	3-102
INSR	(P)9C	FIELD INSERT	3-96
INSU	(E)DC	INSERT UNCONDITIONAL	3-102
ISOL	(P)9A	FIELD ISOLATE	3-88
JOIN	(V)42	SET TWO SINGLES TO DOUBLE	3-96
LAND	(P)90	LOGICAL AND	3-78
LDA	(Z)E0	LOAD A	3-105
LDAI	(Z)E1	LOAD A INCREMENT	3-105
LDB	(Z)E2	LOAD B	3-105
LDBI	(Z)E3	LOAD B INCREMENT	3-105
LDC	(Z)E4	LOAD C	3-105
LDCI	(Z)E5	LOAD C INCREMENT	3-105
LEQV	(P)93	LOGICAL EQUIVALENCE	3-78
LESS	(P)88	LESS THAN	3-79
LLLU	(V)BD	LINKED LIST LOCKUP	3-100
LNOT	(P)92	LOGICAL NEGATE	3-78
LOAD	(P)BD	LOAD	3-77
LODT	(V)BC	LOAD TRANSPARENT	3-77
LOG2	(V)8B	LEADING ONE TEST	3-70
LOR	(P)91	LOGICAL OR	3-78
LSEQ	(P)8B	LESS THAN OR EQUAL	3-79
LT16	(P)B3	LIT CALL 16 BITS	3-78
LT48	(P)BE	LIT CALL 48 BITS	3-78
LT8	(P)B2	LIT CALL 8 BITS	3-78
MCHR	(E)D7	MOVE CHARACTERS	3-103
MFLT	(E)D1	MOVE WITH FLOAT	3-103
MINS	(E)D0	MOVE WITH INSERT	3-102
MKST	(P)AE	MARK STACK	3-85
MPCW	(P)BF	MAKE PROGRAM CONTROL WORD	3-78
MULT	(P)82	MULTIPLY	3-69
MULX	(P)8F	EXTENDED MULTIPLY	3-69
MVNU	(E)D6	MOVE NUMERIC UNCONDITIONAL	3-103
MVST	(V)AF	MOVE TO STACK	3-100
SFDC	(E)DA	SKIP FORWARD DESTINATION CHARACTERS	3-103

APPENDIX H (Cont)

MNEMONIC	MODE ID & HEX CODE	OPERATOR NAME	PAGE
SFSC	(E)D2	SKIP FORWARD SOURCE CHARACTERS	3-103
SGED	(V)F1	SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE	3-97
SGEU	(V)F9	SCAN WHILE GREATER OR EQUAL, UPDATE	3-97
SGTD	(V)F2	SCAN WHILE GREATER, DESTRUCTIVE	3-97
SGTU	(V)FA	SCAN WHILE GREATER, UPDATE	3-97
SINH	(V)A8	SET MEMORY INHIBITS	3-101
SINT	(V)45	SET INTERVAL TIMER	3-98
SISO	(P)D5	STRING ISOLATE	3-82
SLED	(V)F3	SCAN WHILE LESS OR EQUAL, DESTRUCTIVE	3-97
SLEU	(V)FB	SCAN WHILE LESS OR EQUAL, UPDATE	3-97
SLMT	(V)AA	SET MEMORY LIMITS	3-101
SLSD	(V)F0	SCAN WHILE LESS, DESTRUCTIVE	3-97
SLSU	(V)F8	SCAN WHILE LESS, UPDATE	397
SNED	(V)F5	SCAN WHILE NOT EQUAL, DESTRUCTIVE	3-97
SNEU	(V)FD	SCAN WHILE NOT EQUAL, UPDATE	3-98
SNGL	(P)CD	SET TO SINGLE PRECISION, ROUNDED	3-96
SNGT	(P)CC	SET TO SINGLE PRECISION, TRUNCATED	3-96
SPLT	(V)43	SET DOUBLE TO TWO SINGLES	3-96
SPRR	(V)B9	SET PROCESSOR REGISTER	3-99
SRCH	(V)BE	MASKED SEARCH FOR EQUAL	3-100
SRDC	(E)DB	SKIP REVERSE DESTINATION CHARACTERS	3-103
SRSC	(E)D3	SKIP REVERSE SOURCE CHARACTER	3-103
STA	(Z)F0	STORE A	3-105
STAG	(V)B4	SET TAG FIELD	3-98
STAI	(Z)F1	STORE A INCREMENT	3-105
STB	(Z)F2	STORE B	3-105
STBI	(Z)F3	STORE B INCREMENT	3-105
STBR	(P)A4	STEP AND BRANCH	3-71
STC	(Z)F4	STORE C	3-105
STCI	(Z)F5	STORE C INCREMENT	3-105
STFF	(P)AF	STUFF ENVIRONMENT	3-96
STOD	(P)B8	STORE DESTRUCTIVE	3-81
STON	(P)B9	STORE NON-DESTRUCTIVE	3-81
STOP	(V)BF	STOP	3-102
STOR	(Z)40→7F	VECTOR STORE	3-106
SUBT	(P)81	SUBTRACT	3-69
SWFD	(V)D4	SCAN WHILE FALSE, DESTRUCTIVE	3-98
SWFU	(V)DC	SCAN WHILE FALSE, UPDATE	3-98
SWTD	(V)D5	SCAN WHILE TRUE, DESTRUCTIVE	3-98
SWTU	(V)DD	SCAN WHILE TRUE, UPDATE	3-98
SXSN	(P)D6	SET EXTERNAL SIGN	3-96
TEED	(P)D0	TABLE ENTER EDIT, DESTRUCTIVE	3-73
TEEU	(P)D8	TABLE ENTER EDIT, UPDATE	3-73
TEQD	(P)E4	TRANSFER WHILE EQUAL, DESTRUCTIVE	3-83
TEQU	(P)EC	TRANSFER WHILE EQUAL, UPDATE	3-83
TGED	(P)E1	TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE	3-83
TGEU	(P)E9	TRANSFER WHILE GREATER OR EQUAL, UPDATE	3-83
TGTD	(P)E2	TRANSFER WHILE GREATER, DESTRUCTIVE	3-83
TGTU	(P)EA	TRANSFER WHILE GREATER, UPDATE	3-83

APPENDIX H (Cont)

MNEMONIC	MODE ID & HEX CODE	OPERATOR NAME	PAGE
TLED	(P)E3	TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE	3-83
TLEU	(P)EB	TRANSFER WHILE LESS OR EQUAL, UPDATE	3-83
TLSD	(P)E0	TRANSFER WHILE LESS, DESTRUCTIVE	3-83
TLSU	(P)E8	TRANSFER WHILE LESS, UPDATE	3-83
TNED	(P)E5	TRANSFER WHILE NOT EQUAL, DESTRUCTIVE	3-84
TNEU	(P)ED	TRANSFER WHILE NOT EQUAL, UPDATE	3-84
TRNS	(V)D7	TRANSLATE	3-101
TUND	(P)E6	TRANSFER UNCONDITIONAI DESTRUCTIVE	3-84
TUNU	(P)EE	TRANSFER UNCONDITIONAI UPDATE	3-84
TWFD	(V)D2	TRANSFER WHILE FALSE, DESTRUCTIVE	3-84
TWFU	(V)DA	TRANSFER WHILE FALSE, UPDATE	3-84
TWOD	(P)D41	TRANSFER WORDS OVERWRITE, DESTRUCTIVE	3-83
TWOU	(P)DC	TRANSFER WORDS OVERWRITE, UPDATE	3-83
TWSD	(P)D3	TRANSFER WORDS, DESTRUCTIVE	3-82

MNEMONIC	MODE ID & HEX CODE	OPERATOR NAME	PAGE
TWSU	(P)DB	TRANSFER WORDS, UPDATE	3-82
TWTD	(V)D3	TRANSFER WHILE TRUE, DESTRUCTIVE	3-84
TWTU	(V)DB	TRANSFER WHILE TRUE, UPDATE	3-84
UABD	(V)D1	UNPACK ABSOLUTE, DESTRUCTIVE	3-99
UABU	(V)D9	UNPACK ABSOLUTE, UPDATE	3-99
USND	(V)D0	UNPACK SIGNED, DESTRUCTIVE	3-99
USNU	(V)D8	UNPACK SIGNED, UPDATE	3-100
VALC	(P)00 → 3F	VALUE CALL	3-85
VARI	(P)95	ESCAPE TO 16-BIT INSTRUCTION	3-96
VEBR	(Z)EE	VECTOR BRANCH	3-104
VMOM	(Z)EF	VECTOR MODE, MULTIPLE WORD	3-76
VMOS	(Z)E7	VECTOR MODE, SINGLE WORD	3-76
VXIT	(Z)E6	VECTOR EXIT	3-105
WHOI	(V)4E	READ PROCESSOR IDENTIFICATION	3-98
XTND	(P)CE	SET TO DOUBLE PRECISION	3-96
ZERO	(P)B0	LIT CALL ZERO	3-78

APPENDIX I

IOM WORD FORMATS

HA WORD 1, START I/O COMMAND

	LK 47	H O 43	39	35	31	27	23	19	15	11	7	3
T 50	46	M E 42	38	UN 34	IT 30	26	22	18	14	10	6	2
A 49	45	C O 41	37	DES 33	IG- 29	25	21	17	13	9	5	1
G 48	44	D E 40	36	NA 32	TE 28	24	20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
-	46:3	Not used.
HOME CODE	43:4	Defines Start I/O command (0001).
-	39:4	Not used.
UNIT DESIG-NATE	35:8	A unique 8-bit code-used with the UT base address to index and lock fetch from memory the UT word for the device to be started, and used with the QH base address to unlock fetch from memory the QH word, which points to the IOCB base address.
-	27:28	Not used.

HA WORD 1, SET CHANNEL BUSY/RESERVED

	LK 47	H O 43	B/R 39	35	31	C 27	23	19	15	11	7	3
T 50	46	M E 42	38	34	30	H 26	22	18	14	10	6	2
A 49	45	C O 41	37	33	29	N 25	21	17	13	9	5	1
G 48	44	D E 40	36	32	28	O. 24	20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
-	46:3	Not used.
HOME CODE	43:4	Defines Set CH Busy/Set CH Reserved Commands (0010).
B/R	39:1	When reset, further defines command as Set CH Busy; when set, further defines command as Set CH Reserved.

HA WORD 1, SET CHANNEL BUSY/RESERVED

-	38:11	Not used.
CH. NO.	27:5	Identifies one of the 28 possible IOM channels.
-	22:23	Not used.

HA WORD 1, RESET CHANNEL RESERVED

	LK 47	H O 43	B/R 39	35	31	C 27	23	19	15	11	7	3
T 50	46	M E 42	38	34	30	H. 26	22	18	14	10	6	2
A 49	45	C O 41	37	33	29	N 25	21	17	13	9	5	1
G 48	44	D E 40	36	32	28	O. 24	20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines Reset CH Busy/Reset CH Reserved Commands (0011).
B/R	39:1	When reset, further defines command as Reset CH Busy; when set, further defines command as Reset CH Reserved.
—	38:11	Not used.
CH. NO.	27:5	Identifies one of the 28 possible IOM channels.
—	22:23	Not used.

HA WORD 1, LOAD BASE ADDRESS (HA, UT, UOQH, SQ) COMMANDS

	LK 47	H O 43	39	35	31	27	23	19	15	11	7	3
T 50	46	M E 42	38	34	30	26	22	18	ME 14	MO 10	RY 6	2
A 49	45	C O 41	37	33	29	25	21	17	ADD 13	RE 9	SS 5	1
G 48	44	D E 40	36	32	28	24	20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.

HA WORD 1, LOAD BASE ADDRESS (HA, UT, UOQH, SQ) COMMANDS

—	46:3	Not used.
HOME CODE	43:4	Defines: (1) Load Home Address Command (0100) (2) Load Unit Table Address Command (0101) (3) Load I/O Queue Head Address Command (0110) (4) Load Status Queue Address (0111).
—	39:20	Not used.
MEMORY ADDRESS	19:20	The memory address to be stored in the Translator of the IOM to enabl‹ access of the IOM Job Map.

HA WORD 1, DFO SCAN-OUT COMMANDS (CLEAR STACK AND STORE CW REQUEST)

	LK 47	H O 43	39	35	31	27	23	D E 19	D F 15	N 11	ES 7	3
T 50	46	M E 42	38	34	30	26	22	V I C 18	E U 14	U M 10	6	2
A 49	45	C O 41	37	33	29	25	21	E T 17	U N 13	B E 9	T Y 5	1
G 48	44	D E 40	36	32	28	24	20	Y P E 16	I T 12	R 8	P E 4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines the command as Scan Out (DFO or DCP) when 1000.
—	39:20	Not used.
DEVICE TYPE	19:4	Defines the Scan Out command is for a DFO (1001).
DFEU UNIT NMBR and ES (EXCH SELECT)	15:8, 7:1	Together define the DFO by specifying a DFEU unit number and whether it is directly (bit 7=0) or indirectly (bit 7=1) connected to the DFO.
—	6:1	Not used.
TYPE	5:2	Defines the DFO Scan-Out command as Clear the Stack (10) or Store Control Word Request (01).
—	3:4	Not used.

HA WORD 2, DFO SCAN OUT/STORE CONTROL WORD REQUEST COMMAND

	47	43	39	35	31	27	23	19	15	11	7	3
50	46	42	IOCB 38	34	30	26	22	18	DISK 14	10	6	2
49	45	ADDRESS 41	37	33	29	25	21	ADDRESS 17	13	9	5	1
48	44	40	36	32	28	24	20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
—	50:3	Not used.
IOCB ADDRESS	47:20	The base address of the job in memory.
—	27:2	Not used.
DISK ADDRESS	25:26	The disk address to be used for the job.

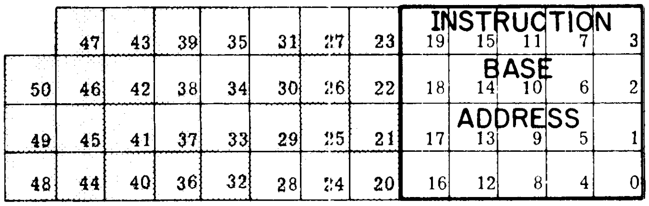
NOTE
This format also represents the format of the Scan Information word sent to the DFO.

HA WORD 1, DCP SCAN-OUT COMMANDS (INITIATE, HALT, SET ATTENTION)

	LK 47	H O 43	39	35	31	27	23	D E 19	15	11	T Y 7	D C 3
T 50	46	M E 42	38	34	30	26	22	V I C 18	14	10	P 6	P 2
A G 49	45	C O 41	37	33	29	25	21	E T 17	13	9	E 5	N O. 1
48	44	D E 40	36	32	28	24	20	Y P E 16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines the command as Scan Out (DFO or DCP) when 1000.
—	39:20	Not used.
DEVICE TYPE	19:4	Defines the Scan Out command is for a DCP (110).
—	15:8	Not used.
TYPE	7:3	Defines the DCP Scan-Out command as Initiate (000), Halt (010), or Set Attention (100).
—	4:1	Not used.
DCP NO.	3:3	Defines the DCP for which the command is intended.
—	0:1	Not used.

HA WORD 2, DCP SCAN-OUT/INITIATE COMMAND



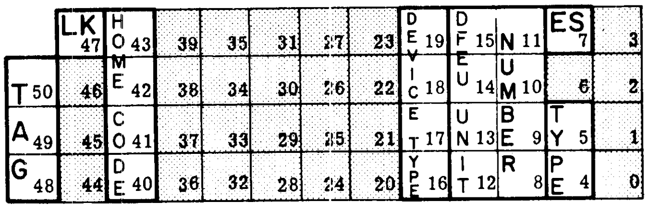
FIELD **BITS** **DESCRIPTION**

— 50:31 Not used.

INSTRUCTION BASE ADDRESS 19:20 Define the memory base addres the DCP code.

NOTE
this format also represents the format of the Scan Information word sent to the DCP.

HA WORD 1, DFO SCAN-IN COMMANDS (QUEUED CONTROL WORD, TOP OF STACK, REPORT)



FIELD **BITS** **DESCRIPTION**

TAG 50:3 Denotes the word is single precision (000).

LK 47:1 When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.

— 46:3 Not used.

HOME CODE 43:4 Defines the command as Scan in (1001).

— 39:20 Not used.

DEVICE TYPE 19:4 Defines the command as for a DFO (1001).

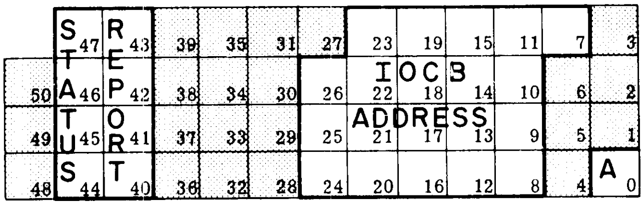
DFEU UNIT NMBR and ES (EXCH SEL) 15:8, 7:1 Together define the DFO by specifying a DFEU unit number and whether it is directly or indirectly connected to the DFO (via an exchange). These fields are not used for the Scan-In DFO Report Command.

— 6:1 Not used.

TYPE 5:2 Defines the DFO Scan-Out command as either Queued Control Word (01), Top of Stack (10), or Report (11).

— 3:4 Not used.

HA WORD 2 (SCAN-IN WORD), SCAN-IN DFO QUEUED CONTROL WORD AND TOP OF STACK COMMANDS



FIELD **BITS** **DESCRIPTION**

— 50:3 Not used.

STATUS REPORT 47:8 Describes the nature of the DFO by bits set as follows:

(1) 47 set = No Access to Exchange
(2) 46 set = SU Not Available
(3) 45 set = Parity Error
(4) 44 set = Disk Address Error
(5) 43 set = Queded Control Word
(6) 42 set = Top of Stack Control Word
(7) 41 set = Stack Empty
(8) 40 set = Control Word Not Available

— 39:13 Not used.

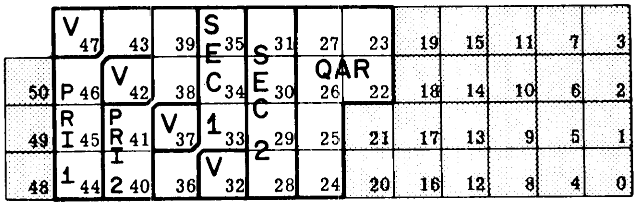
IOCB ADDR 26:20 Defines the memory address of the IOCB.

— 6:6 Not used.

A (ATTEN) 0:1 When set, alerts the IOM to examine the STATUS REPORT FIELD.

NOTE:
This format also represents the format of the Scan Information word received from the DFO.

HA WORD 2 SCAN-IN DFO REPORT COMMAND



FIELD **BITS** **DESCRIPTION**

- 50:3 Not used.

V 47:1 When true indicates connection of an EU/DFO bus at port 1 and the EUs connected to this bus are referenced by the EUD code present on lines 43-46 of the Scan Information Lines.

PRI 1 46:4 EUD code for port 1 bus (bit 46=MSB).

V	42:1	When true indicates connection of an EU/DFO bus at port 2 and the EUs connected to this bus are referenced by the EUD code present on lines 38-41.
PRI 2	41:4	EUD code for port 2 bus (bit 41=MSB).
V	37:1	When true indicates connection of an EU/DFO bus at port 3, and the EUs connected to this bus are referenced by the EUD code present on lines 33-36.
SEC 1	36:4	EUD code for port 3 bus (bit 36=MSB)
V	32:1	When true indicates connection of an EU/DFO bus at port 4, and the EUs connected to this bus are referenced by the EUD code present on lines 28-31.
SEC 2	31:4	EUD code for port 4 bus (bit 31=MSB).

NOTE

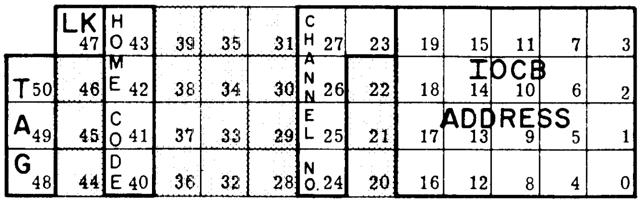
If a given EUD code appears on Scan Information lines 38-41 or 43-46, then the EUs referenced by the code are connected to the responding DFO in a direct manner, but if the EUD code appears on lines 28-31 or 33-36, then the EUs referenced by the EUD code are connected to the responding DFO indirectly (that is, via the other DFO of the DFO-pair).

QAR	27:6	Indicates capacity of memory stack (bit 27=MSB).
-	21:22	Not used.

NOTE

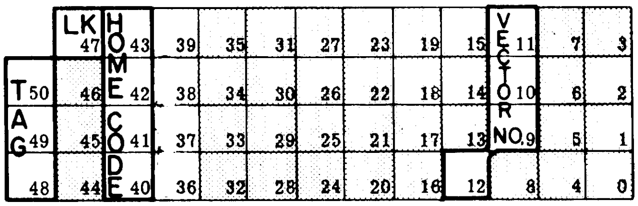
This format also represents the format of the Scan Information word received from the DFO.

HA WORD 1, SYNC I/O COMMAND



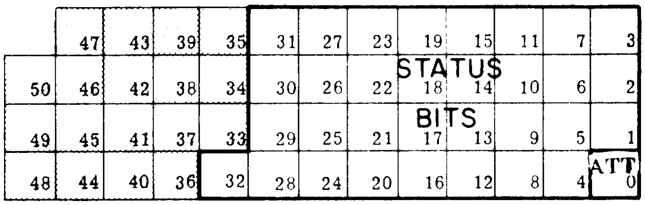
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines the command as Sync I/O.
—	29:12	Not used.
CHANNEL NO.	27:5	Identifies one of the 28 possible IOM channels.
—	22:3	Not used.
IOCB ADDRESS	19:20	The address of the job request in memory.

HA WORD 1, INTERROGATE PERIPHERAL STATUS COMMAND



FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines Interrogate Peripheral Status Command (1011).
VECTOR NO.	12:4	Defines the number of the status vector to be interrogated.
—	8:9	Not used.

HA WORD 2 (STATUS WORD RETURNED), INTERROGATE PERIPHERAL STATUS COMMAND



FIELD	BITS	DESCRIPTION
—	50:18	Not used.
STATUS BITS	32:32	Each bit of this field, when on, indicates the ready status of the associated unit on the vector. (Refer to table I-1 for referencing the ready status vector, ready status bit, and device number of any peripheral device.)
ATT	0:1	When set, alerts the IOM to examine the STATUS BITS field.

TABLE I-1. STATUS VECTOR CROSS REFERENCE

VECTOR BIT NO.	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		VECTOR
"B" REGISTER	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
31 - 0																																	ATTENTION BIT	0
63 - 32																																		1
95 - 64																																		2
UNIT 127 - 96																																		3
DEST. 159 - 128																																		4
191 - 160																																		5
223 - 192																																		6
255 - 224																																		7

TO FIND THE STATUS VECTOR FOR A UD NUMBER, DIVIDE THE UD NUMBER BY 32. THE STATUS VECTOR IS THE INTERGER QUOTIENT AND THE VECTOR BIT IS THE REMAINDER PLUS ONE.

EXAMPLE: UD NUMBER = 95

$$\begin{array}{r@{}l} & 2 \text{ (STATUS VECTOR)} \\ 32\overline{)95} & \\ \underline{64} & \\ 31 + 1 = 32 & \text{ (VECTOR BIT NUMBER)} \end{array}$$

TO FIND THE UD NUMBER, MULTIPLY THE STATUS VECTOR NUMBER BY 32 AND ADD TO THE RESULT THE VECTOR BIT NUMBER MINUS ONE.

EXAMPLE: STATUS VECTOR NUMBER = 2

2 x 32 = 64 + (32 – 1) = 95 (UD NUMBER)

HA WORD 1, INHIBIT IOM, ACTIVATE IOM, AND LOAD DFO FLAGS COMMANDS

	LK 47	H O 43	D F 39	35	31	27	23	19	15	11	7	3
T 50	46	M E 42	O F 38	34	30	26	22	18	14	10	6	2
A G 49	45	C O 41	L A 37	33	29	25	21	17	13	9	5	1
48	44	D E 40	G S 36	32	28	24	20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM services.
HOME CODE	43:4	Defines the command as follows: (1) 1100 = Inhibit IOM (2) 1101 = Activate IOM (3) 1110 = Load DFO Flags
	39:4	DFO Flags (for LOAD DFO FLAGS command only).
	35:36	Not Used.

UNIT TABLE WORD

	LK 47	43	DFO 39	35	31	C H 27	23	19	15	11	C H 7	3
T 50	MGT 46	42	EX 38	FU 34	D 30	N O 26	L C 22	18	NU 14	D 10	N O 6	2
A G 49	DSPK 45	41	JB 37	33	29	B A 25	E X 21	17	13	9	U S 5	1
48	SL 44	40	BZ 36	32	28	S E 24	RC 20	LST 16	12	8	E D 4	ET 0

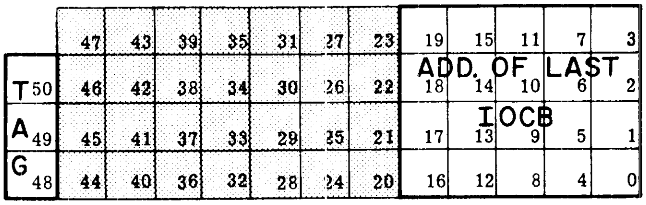
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set, indicates the UT word is being operated on.
MGT	46:1	When set, indicates this job request is for a magnetic tape. (Set by software.)
DSPK	45:1	When set, indicates this job request is for a disk pack. (Set by software.)
SL	44:1	When set, indicates the presence of a side link in IOCB+1.
—	43:4	Not used.
DFO	39:1	When set, indicates unit is under control of a DFO. A ring walk will not be performed with this bit set. (Set by software.)
EX	38:1	When set, indicates the unit is connected to an exchange. A ring walk will be performed (if the job bit is set) with this bit set. (Set by software.) Not used if bit 39 is set.
JB	37:1	When set, indicates that all channels associated with this request were busy, and when a channel becomes free and no further request are queued for that device, this job is to be done. (Set by IOM.) Used only with exch. devices (Bit 38=1). Not used with DFO (Bit 39).
BZ	36:1	When set, indicates that this unit is busy. (Set by IOM.)
FUD	35:8	Points to the First Unit Designate Number connected to the exchange.
CH. NO. BASE	27:5	For units not on an exchange, the number of the channel to which this unit is connected. For units on an exchange, the lowest numbered channel to which the exchange is connected. NOTE: CN 0 and 21 through 23 are unassigned and will cause a fail.
LCEX	22:2	Indicates the 2 least significant bits of the last channel number of the exchange, for the device to be used.
RC	20:1	When set, permits this unit to use a reserved channel.
—	19:3	Not used.
LST	16:1	When set, indicates this is the last Unit Designate on the exchange.
NUD	15:8	Points to the Next Unit Designate number connected to the exchange.
CH. NO. USED	7:5	These bits specify the channel that was used to service the device. (Set by IOM.)
—	2:2	Not used.
ET	0:1	When set, indicates that an error condition has been reported in the current Result Descriptor, and therefore additional jobs should not be initiated for the unit. This bit is normally reset by software.

IOQH WORD

	47	43	39	35	31	27	23	19	AD 15	DRE 11	SS 7	3
T 50	46	42	38	34	30	26	22	18	OF 14	FI 10	RST 6	2
A G 49	45	41	37	33	29	25	21	17	IOC 13	B 9	5	1
48	44	40	36	32	28	24	20	16	12	8	4	0

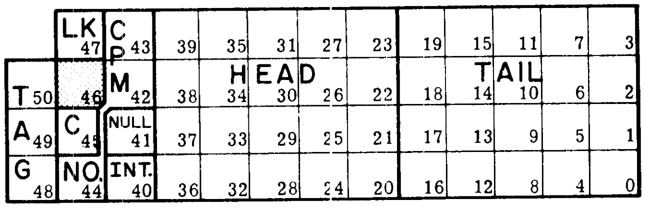
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
—	47:28	Not used.
ADDRESS OF FIRST IOCB	19:20	Address of 1st IOCB in the IOQ. If bits 19-0 are null (zero), the UT word is unlocked and restored to memory.

IOQT WORD



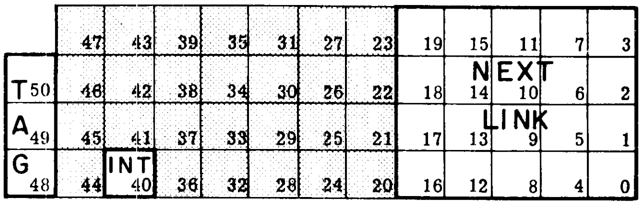
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
—	47:28	Not used.
ADD OF LAST IOCB	19:20	Address of last IOCB in the IOQ.

SQH WORD



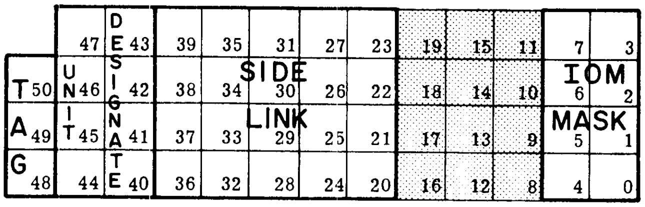
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set, indicates the SQH word is being operated on.
—	46:1	Not used.
C	45:1	Notifies software, when set, that a status change vector has occurred.
CPM NO.	44:3	Points to the CPM that will be interrupted by either channel interrupt or error interrupt.
NULL	41:1	When a 0, indicates that the queue is empty; when a 1, indicates terminated jobs are under queue.
INT	40:1	When set, (set by software) indicates that the CPM number field shall be interrupted upon job termination. (Reset by IOM)
HEAD	39:20	A 20-bit address pointing to the IOCB of the first device terminated. (Not used if bit 41 = 0)
TAIL	19:20	A 20-bit address pointing to the IOCB of the last device terminated. (Not used if 41 = 0)

IOCB WORD O (IOCB I/O LINKAGE (N/L) WORD)



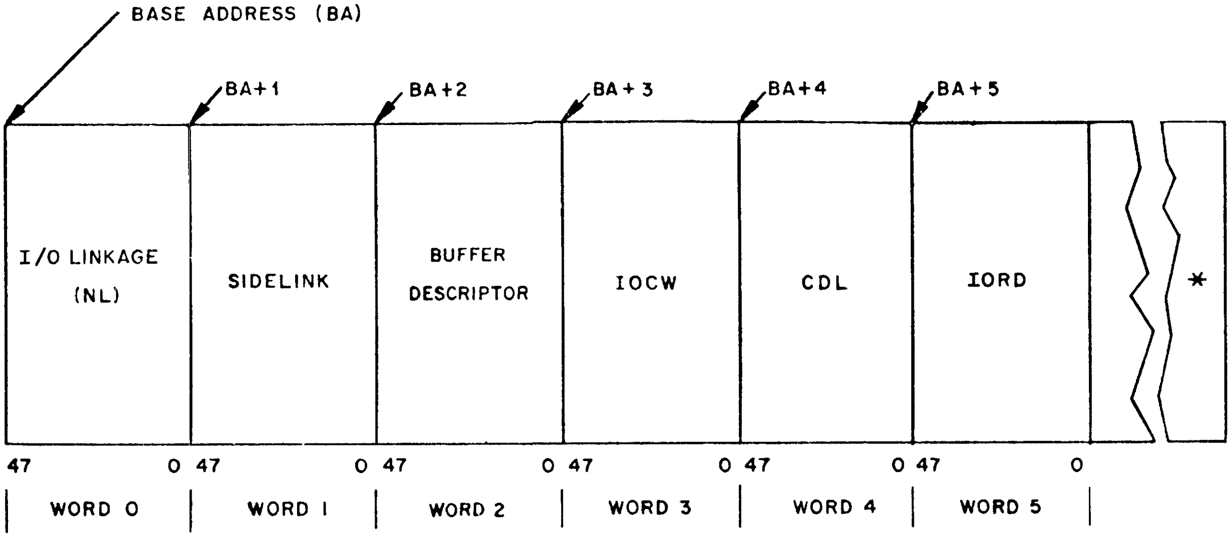
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
—	47:7	Not used.
INT	40:1	When set, notifies the IOM to interrupt the CPM specified in the SQ word upon completion of this job.
—	39:20	Not used.
NEXT LINK	19:20	Memory address of the next job (IOCB) queued for this device.

IOCB WORD 1 (IOCB SIDELINK (SL) WORD)



FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
UNIT DESIGNATE	47:8	Defines the device which is to perform this sidelinked job.
SIDE LINK	39:20	Memory address of the sidelinked job.
-	19:12	Not used.
IOM MASK	7:8	Defines an IOM channel number and thus defines the IOM (or IOM's) which can perform the sidelinked job.

I/O CONTROL BLOCK (IOCB)



* WORDS 6 THRU N ARE RESERVED FOR SOFTWARE USE ONLY

IOCB WORD 2 (IOCB BUFFER DESCRIPTOR (BD) WORD)

	47	43	C T 39	35	31	27	23	19	15	11	7	3
T 50	46	42	E X 38	34	LENGTH 30	26	22	18	BASE 14	10	6	2
A G 49	45	41	T 37	33	29	25	21	17	ADDRESS 13	9	5	1
48	44	40	36	32	28	24	20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
–	47:8	Not used.
CT EXT	39:3	If the length of the buffer includes a fractional part of a word, this field describes the number of characters in that fractional part.
LENGTH	36:17	Describes the length of the buffer in words. (Excess characters are described by the CT EXT field.)
BASE ADDRESS	19:20	Describes the memory address of the first data word of the buffer.

IOCW (IOCB WORD 3)

	ASC 47	MINH 43	B/F 39	35	31	27	23	19	15	11	7	3
T 50	SL 46	TRA 42	T 38	34	30	26	22	18	14	10	6	2
A G 49	SA 45	FML 41	TAG 37	33	29	25	21	17	13	9	5	1
48	I/O 44	MP 40	CTL 36	32	28	24	20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
ASC	47:1	When set, indicates that ASCII translation is required.
SL	46:1	When set, indicates that a sidelink to another IOCW is required. (The address of the new IOCW is stored in bits 0 thru 19 of the IOCB SL word.)
SA	45:1	When set, will cause bit 1 of the result descriptor word (the Exception bit) to be set.
I/O	44:1	When set, indicates that the transfer is to be an input operation. When reset, indicates that the transfer is to be an output operation.
MINH	43:1	When set, indicates that data will not be transferred to/from memory.
TRA	42:1	When set, indicates that internal IOM translation is needed.
FML	41:1	When set, indicates that the frame length is to be 8-bits. When reset, indicates that the frame length is to be 6-bits.

MP 40:1 When set, indicates that a memory protect interrupt will occur if an attempt is made to store into a word in memory which has bit 48 = 1. The store will not occur.

B/F 39:1 When set, indicates a backward operation on a tape unit. When reset, indicates a forward operation on a tape unit.

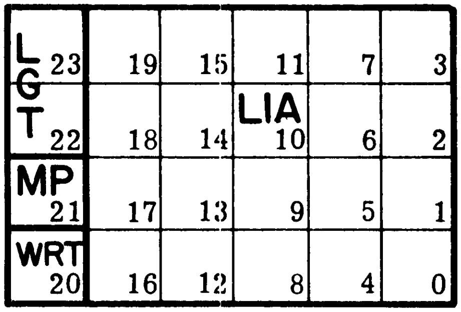
T 38:1 When set, indicates a test operation.

TAG CTL 37:2 Indicates the following:

37	36	
0	0	Store single precision ta
1	1	Store double precision tags
0	1	Store program tags
1	0	Tag field transfer

35:36 Not used.

UNIT CONTROL WORD (UCW)



L G 23	19	15	11	7	3
T 22	18	14	LIA 10	6	2
MP 21	17	13	9	5	1
WRT 20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
LGT	23:2	Specify the total length of the field being transferred as follows:

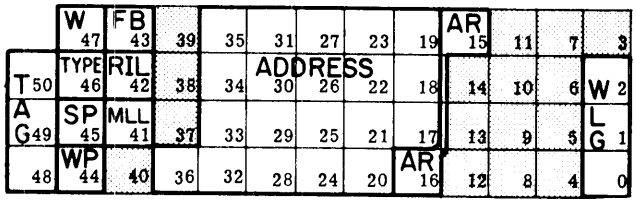
23	22	
0	1	= Transfer 1 word
1	0	= Transfer 2 words

MP 21:1 On a one or two word store, if bit 48 of the information word already stored in that memory location is a one (protected word), memory shall not perform the store but shall send an error signal to the requestor.

WRT 20:1 Shall identify the service request as a Read (WRT=0) or Write (WRT=1) operation.

L1A 19:20 Shall specify the absolute starting memory address of the transfer.

MEMORY CONTROL WORD

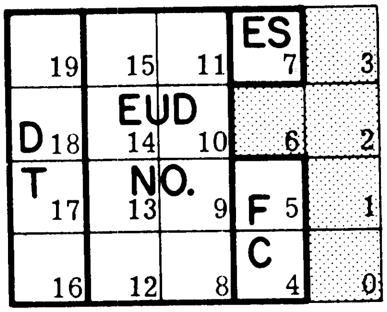


	W 47	FB 43	39	35	31	27	23	19	AR 15	11	7	3
T 50	TYPE 46	RIL 42	38	34	ADDRESS 30	26	22	18	14	10	6	W 2
A G 49	SP 45	MLL 41	37	33	29	25	21	17	13	9	5	L G 1
48	WP 44	40	36	32	28	24	20	AR 16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Not significant for control purposes; examined only for generation of parity.
W (Write)	47:1	When a 0 specifies that a read/restore operation is to be performed. When a 1, specifies that one of the write variations, as defined by the TYPE field, is to be performed.
TYPE	46:1	When the W field is a 1, specifies which write variation is to be performed as follows: when 46=0, a Clear/Write operation shall be performed (the Overwrite and Single-Word protected Write operations use this variation). When 46=1, a Read/Modify/Restore operation shall be performed (the N-Word Protected Write operation uses this variation). When the field is a 0 and 46=1, the contents of the Fail Register are fetched.
SP (Specifier)	45:1	When a 1, indicates that a single-word operation is to be performed. When a 0, indicates that an N-word operation is to be performed.
WP (Write Protect)	44:1	When a 1, indicates that a Protected Write operation is to be performed. It is a 0 if any other type of operation is specified.

FB (Flashback)	43:1	When a 1, specifies that the original contents of the memory location are to be sent to the requestor.
RIL (Requestor Inhibit Load)	42:1	Used in a Single-Word Overwrite operation to specify that a Load Requestor operation is to be performed. When a 1, specifies that the next data word sent to the MCM be loaded into the Requestor Inhibit Register instead of into memory.
MLL (Memory Limits Load)	41:1	When a 1, specifies that the next data word sent to the MCM be loaded into the Memory Limit Registers and the Available Register, instead of into memory.
ADDRESS	36:20	Specify the starting address for the memory operation.
AR (Address Residue)	16:2	Indicate the proper value (00, 01, or 10) that result from changes in the ADDRESS field.
—	14:12	Not used.
WLG (Word Length)	2:3	Indicates the number of words to be transferred during memory operations (2 words maximum).

DFO SCAN ADDRESS WORD (SCAN-IN AND SCAN-OUT)

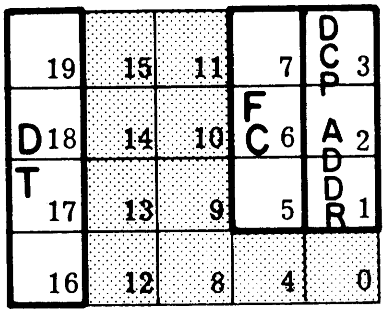


FIELD	BITS	DESCRIPTION
DT	19:4	Identifies the information as for a DFO (1001).
EUD NO. AND ES (EXCHANGE SELECT)	15:8, 7:1	Together define the DFO by specifying a DFEU unit designate number and whether it is directly or indirectly connected to DFO (via an exchange). These fields are not used if Scan-In DFO Report is the job to be implemented.
–	6:1	Not used.
FC	5:2	Function code which defines the operation as follows: (1) During Scan-Out: 5 4 0 1 = Store CW Request 1 0 = Clear-the-Stack (2) During Scan-In: 5 4 0 1 = Queued CW Request 1 0 = Top-of-Stack Request 1 1 = Report Request
	3:4	Not used.

NOTE

The format of the DFO Scan Address word may be related directly to bits 0 through 19 of HA word 1, when HA word 1 contains a command for DFO scan-out or scan-in.

DCP SCAN ADDRESS WORD



FIELD	BITS	DESCRIPTION
DT	19:4	Defines the Scan-Out command is for a DCP (1100).
—	15:8	Not used.
FC	7:3	Defines the DCP Scan-Out command as Initiate (000), Halt (010), or Set Attention (100).
—	4:1	Not used.
DCP ADDR	3:3	Defines the DCP for which the command is intended.
—	0:1	Not used.

NOTE

The format of the DCP Scan Address word may be related directly to bits 0 through 19 of HA word 1, when HA word 1 contains a DCP scan-out command.

cut along dotted line

BURROUGHS CORPORATION
DATA PROCESSING PUBLICATIONS
REMARKS FORM

TITLE: B 7700 Information Processing
Systems, Reference Manual

FORM: 1060233
DATE: 2/12/76

CHECK TYPE OF SUGGESTION:

□ ADDITION □ DELETION □ REVISION □ ERROR

GENERAL COMMENTS AND/OR SUGGESTIONS FOR IMPROVEMENT OF PUBLICATION:

FROM: NAME ____________________
TITLE ____________________
COMPANY ____________________
ADDRESS ____________________
____

DATE ____________

STAPLE

FOLD DOWN SECOND FOLD DOWN

Postage
Will Be Paid
by
Addressee

No
Postage Stamp
Necessary
If Mailed in the
United States

BUSINESS REPLY MAIL

First Class Premit No. 817, Detroit, Michigan

Burroughs Corporation
T10 East-Large System
P.O. Box 203
Paoli, Pa. 19301

Attn: Systems Documentation
TIO EAST

FOLD UP FIRST FOLD UP