QANTEL CORPORATION

BUSINESS ASSEMBLY LANGUAGE

REFERENCE MANUAL

QANTEL

BUSINESS

ASSEMBLY

LANGUAGE

(Q/BAL)

REFERENCE MANUAL

QANTEL Sales Offices: 3474 Investment Blvd., **Hayward,** CA 94545, (415) 783-3410 · 1700 So. El Camino Real, **San Mateo,** CA 94402, (415) 349-8400 · 8622 Bellanca Avenue, **Los Angeles,** CA 90045, (213) 641-8644 · 1 Alexander Lane, Littleton **(Denver),** CO 80121, (303) 770-2411 · 1000 Skokie Blvd., Wilmette **(Chicago),** IL 60091, (312) 256-2800 · P.O. Box 13302, **Kansas City,** MO 64105, (816) 842-0384 · 725 14th St., **Marion,** IO 52303, (319) 377-8277 · 26525 Southfield Rd., **Southfield,** MI 40875, (313) 356-6686 · 500 North Avenue, **Wakefield,** MA 01880, (617) 246-0936 · 1381 Marshall Avenue, **St. Paul,** MN 55104, (612) 646-6573 · 245 Park Avenue, **New York,** NY 10017, (212) 867-5035 · 2316 Seventh Avenue South, **Birmingham,** AL 35233, (205) 252-0114 · 2907 Bay-to-Bay Blvd., **Tampa,** FL 33609, (813) 837-2438.

QANTEL
CORPORATION

MARCH 1, 1973

TABLE OF CONTENTS

SECTION I – GENERAL DESCRIPTION

SECTION II – PROCESSOR FUNCTIONS

SECTION III – ASSEMBLER INSTRUCTION CODING

SECTION IV – ASSEMBLER DIRECTIVES

SECTION V – INPUT/OUTPUT INSTRUCTIONS AND DEVICES

PARAGRAPH		PAGE

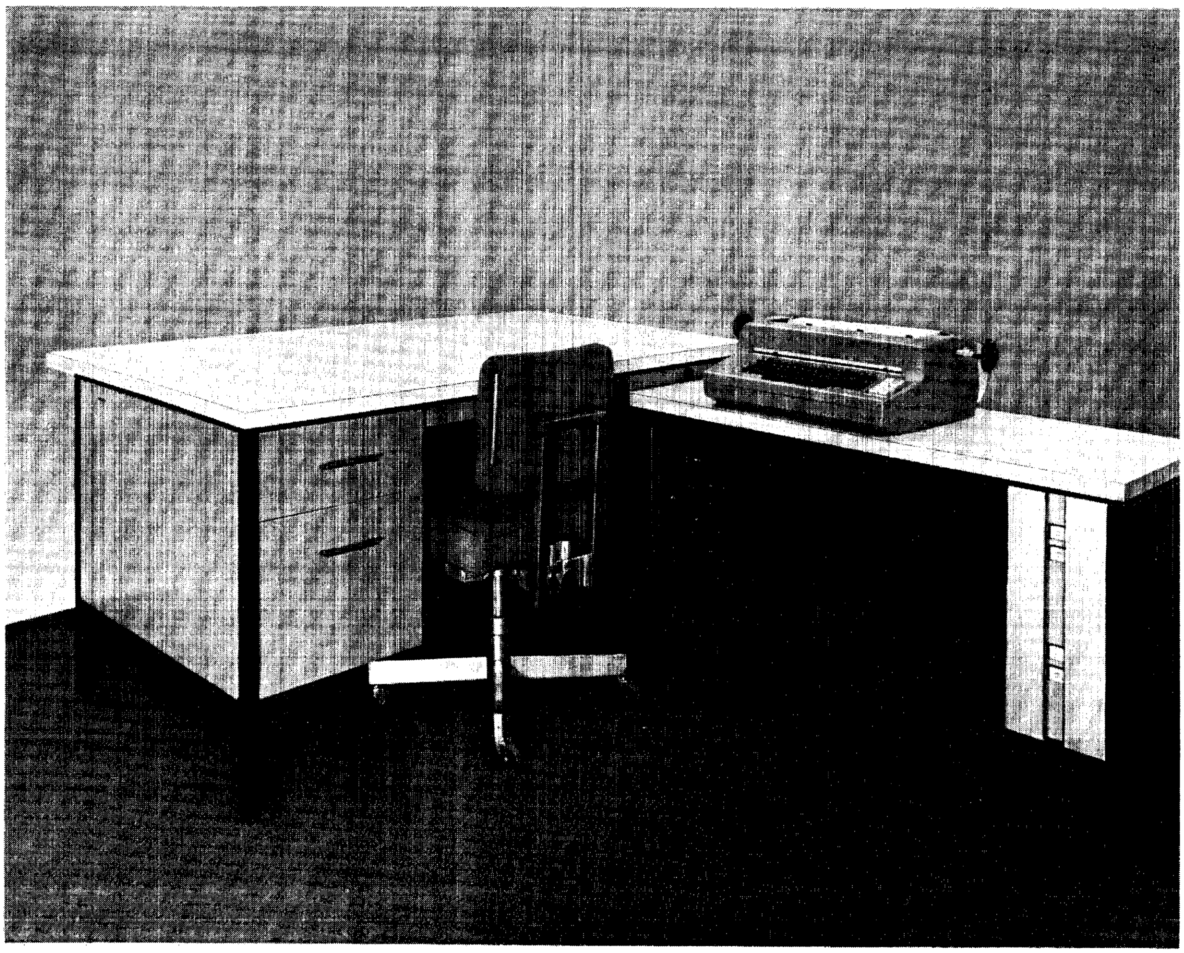
SECTION VI – ASSEMBLER OPERATION

APPENDIXES

LIST OF ILLUSTRATIONS

FIGURE NUMBER		PAGE

LIST OF TABLES



QANTEL ANSWER PROCESSOR SYSTEM

SECTION I

GENERAL DESCRIPTION

1-1. INTRODUCTION

1-2. The QANTEL Business Assembly Language (Q/BAL) is provided by QANTEL Corporation to permit assembly language programming of the QANTEL/ANSWER Processor System. Assembly language programming allows the programmer to write programs using mnemonics instead of machine language instructions and operation codes. The assembler mnemonics are indicative of the particular instructions and the assembler programs contain many aids to greatly simplify programming for the QANTEL/ANSWER Processor System.

1-3. The purpose of this reference manual is to provide the experienced programmer with the information necessary to use the QANTEL Business Assembly Language and to familiarize the programmer with the operation of the QANTEL/ANSWER Processor System standard instructions. The manual is divided into six sections and supporting Appendixes. These sections contain the following information:

a. Section I, General Description - describes the reference manual, lists related publications, provides definitions of terms, describes features and specifications of the QANTEL hardware, and describes the functional operation, library capabilities and object program formats of the QANTEL Business Assembly Language.

b. Section II, Processor Functions – describes machine language processor functions and programming requirements for the QANTEL/ANSWER Processor System.

c. Section III, Assembler Instruction Coding – describes the coding formats and the rules for coding QANTEL Business Assembly Language statements.

d. Section IV, Assembler Directives – describes the instructions used to control the assembler and to produce constants and data areas.

e. Section V, Input/Output Instructions and Devices – describes the various input/output instructions and programming requirements for the various input/output devices.

f. Section VI, Assembler Operation – provides the operating procedures for the Q/BAL Assembler programs.

g. Supporting Appendixes.

1-4. RELATED REFERENCE PUBLICATIONS

1-5. The QANTEL Business Assembly Language Reference Manual is complemented by the following additional QANTEL publications:

a. QANTEL Business Assembly Language Programmers Training Manual – especially useful for teaching new programmers.

b. QANTEL Micro-Assembler Manual – required only for those customers intending to alter the instruction set.

c. Product Specifications for the QANTEL/ANSWER Processor System and I/O Devices – for original equipment manufacturers.

TABLE 1-1. TABLE OF TERMS

TERM	DESCRIPTION
Bit	Single binary digit having the value of zero or one.
Byte	An 8-storage location in memory which may assume any of 256 possible bit configutations (from hex 00 to FF).
Operand	Data represented in a byte, or combination of bytes, that is used in some operation.
Operand Address	Operands in the QANTEL/ANSWER Processor System are addressed by referring to the least significant byte (highest memory location), except in the case of input/output instructions.
Buffer	Refers to a hardware device or main memory positions used for the temporary storage of data. In both cases, the buffer size is described by its length in bytes.
Assembler	Software package that allows programmers to use symbolic language references for instructions and addresses, thereby simplifying the programming task.
Standard Instruction	A micro-program stored in the Read-Only Memory that represents a function found on all QANTEL/ANSWER Systems, e.g., Add Decimal, Edit, Branch and Link, Read, etc.
Micro-Instruction	Single byte instructions contained in the Read-Only Memory (or main memory) that are decoded to preform specific operations.
ROM	Read-Only Memory. A hard-wired control memory that delivers a series of micro-instructions specified (addressed) by the decoded standard instruction. The ROM is installed in the QANTEL/ANSWER Processor System at the factory and cannot be altered.
Fetch/Execute Cycles	Preformance of an instruction is completed by the fetch and Execute cycles. In the fetch cycle, the instruction is read from main memory, and is examined byte-by-byte so that its format can be determined. Once all addresses and operation codes have been examined, the execution cycle performs the actual operation indicated by the instruction.
A and B Operands	Generally, the A operand is the source operand, and the B operand is the resultant or second operand. In an Add Decimal Instruction, for example the A operand is added to the B operand, and the result is placed in the B operand.
IPL	Initial Program Load.
ASCII	American Standard Code for Information Interchange.
Length	The number of consecutive bytes to be operated on by an instruction. Length is variant in several standard QANTEL Instructions, and is counted down as the instruction is executed byte-by-byte. When count zero is reached, the length of the operand field is said to be exhausted. The maximum length of most variable length instructions (except Move), is sixteen bytes, and is represented by zero in the instruction. Minimum length is one byte.

1-6. DEFINITION OF TERMS

1-7. Some of the terms used throughout this reference manual may have different meanings to the readers as a result of their previous experience. Table 1-1 defines some of these terms to make the interpretation of the information as easy as possible for a reader desiring to acquaint himself with the detailed operation of the QANTEL/ANSWER Processor System.

NOTE

Successful operation of the QANTEL/ANSWER Processor System and pre-programmed environment (standard applications software packages) does not require a detailed knowledge of the processor operation.

1-8. SYSTEM DESCRIPTION

1-9. The QANTEL/ANSWER Processor system is designed for small-scale data applications as self-contained units or as intelligent terminals (satellite). The heart of QANTEL Processor System is a serial processor which eliminates the need for complex software, thereby reducing the effort required to implement operational work. Simultaneous input/output and computing is provided through hardware buffering during operation with many of the standard QANTEL peripherals.

1-10. With the QANTEL/ANSWER Processor system, the user has the option to utilize pre-programmed applications, to program at the processor standard instruction levels or, for the experienced staff, to program at the micro-instruction level. Processor organization, together with an extremely basic Read-Only Memory word design, provides the unusual versatility of the QANTEL/ANSWER Processor Systems. The standard internal code of the processor is in ASCII format. However, any desired data format may be used, permitting QANTEL/ANSWER Processor System to operate satellite to any major computer.

1-11. The components of the basic system are normally mounted in a standard L-shaped secretarial desk with I/O typewriter recessed in the desk extension side. The power supply/processor control panel is mounted underneath the extension.

1-12. High-Speed Memory

1-13. Program and data storage within the processor are provided by an eight-bit, variable address, IC main memory that has a complete cycle time of 1.5 microseconds. Cycle time is the time required to transfer one byte of information from memory to the memory register and regenerate the byte back into storage. The main memory is made up of modules, each having 4096 separately addressable eight-bit locations. At the time of this printing memory combinations of one, two, four, six, or eight modules are available.

1-14. Control Memory

1-15. Processor internal control is a function of micro-instructions generated in the Read-Only Memory (ROM). The standard ROM now contains 1536 control words that are used to configure the algorithms of micro-instructions that make up the standard instruction set and disc control instruction logic. Machines that were installed prior to the release of the ANSWER Processor Systems, and that did not include a disc drive, and the additional 512 word disc ROM that was installed with these machines, contained a ROM of 1024 control words. Therefore, these machines will not contain the newest standard instructions. These new standard instructions are noted on the list of QANTEL Standard Instructions shown in Appendix D. On the ANSWER Processor System an additional 512 control words are available, as an option, to supply any other instructions desired by the user. Also, all or part of the micro-instruction complement furnished by the ROM may be optionally specified by the user to meet special demands.

1-16. Hexadecimal System

1-17. Machine language addresses and characters used in the processor are in binary form. Because binary combinations are often difficult to work with and describe, QANTEL publications use the hexadecimal numbering system to represent characters and addresses.

1-18. The Hexadecimal system is a method commonly used to describe the 16 different configurations of four binary bits. Table 1-2 shows how the first ten configurations (in binary sequence) are represented by the decimal numbers zero through nine (0-9). The last six configurations are represented by the alphabetic letters A through F.

TABLE 1-2. HEXADECIMAL NUMBERING SYSTEM

HEXADECIMAL (BASE 16)	BINARY (BASE 2)	DECIMAL (BASE 10)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

NOTE

A complete hexadecimal-to-decimal conversion table is presented in Appendix A at the rear of this manual.

1-19. ASCII

1-20. ASCII is used within the QANTEL/ANSWER Processor Systems. A complete ASCII table is included in Appendix B as an aid to system planning exercises.

1-21. **System Specifications**

1-22. Pertinent specifications for the QANTEL/ANSWER Processor System are listed in table 1-3. More detailed specifications are contained in the respective QANTEL Product Specifications available for original equipment manufacturers.

1-23. **ASSEMBLER DESCRIPTION**

1-24. The Q/BAL Assembler programs described in the following paragraphs perform several general functions. These functions include:

a. Line Entry – accepts each line of the coded assembler language program by means of keyboard entry from the typewriter or 80-column punched cards and checks each statement for correct syntax.

b. Generation of Program Source File – generates an intermediate source program that can be modified using typewriter or card input.

TABLE 1-3. SYSTEM SPECIFICATIONS

PROCESSOR	
Main Memory	IC memory, 4096 eight-bit locations, expandable to 8192, 16384, and 32768, 1.5 microsecond cycle time.
Processor Control	Read-Only Memory with 50 nano-second cycle time.
Physical Dimensions (max. configuration)	26" x 17" x 17½"
INPUT/OUTPUT (Basic System)	
Typewriter	IBM 735 Heavy Duty Selectric 14.7 characters per second Fully buffered (128 characters)

NOTE

The basic QANTEL/ANSWER Processor System may be expanded using additional I/O typewriters, magnetic tape units, disc drives, card readers, printers, communications capabilities, CRT's, optical mark readers, ten-key keyboards, and a programmers control console. For specifications on these varied I/O devices refer to the QANTEL Product Information sheets and engineering specifications.

POWER SUPPLY	
Input Voltages	105-125 vac, 210-250 vac at 48 to 61 HZ
Primary Power Failure Protection	Primary power interruptions of up to eight milliseconds cause no ill effect on system operation
Short Circuit Protection	Fuse protection provided
Physical Dimensions	17" x 17" x 7"
ENVIRONMENTAL CONDITIONS	0 to +40°C (32°F to 104°F), up to 85% relative humidity without condensation

c. Production of Object Program and/or Listing – assembles the program source file and produces a loadable object program and/or a printed listing of the assembled program showing the instructions in both assembler and machine language. A loadable object program, once loaded, forms an executable set of machine instructions and associated data that is capable of performing useful data processing tasks.

1-25. The following paragraphs describe the two passes of the Q/BAL assembler, program update operation, source program library files, and the object program library files.

1-26. **Pass One**

1-27. The QANTEL Business Assembly Language uses two separate programs, run in sequence, to perform the assembly operation. The first assembler program generates the source program and is referred to as Pass One. Operation of the Pass One program performs the following functions:

a. Accepts the Assembly language input from the typewriter or card reader and/or previously created source file (disc or magnetic tape) and checks for errors in format, operation codes, statement syntax and label assignments.

b. Creates a table of the labels or tags used as the program statements are entered and assigns values to these labels.

c. Places the entered program and tag table on the new source file to create the source programs for processing by Pass Two. Refer to figures 1-1 and 1-2.

1-28. PASS ONE ERROR DETECTION

1-29. There are many possible errors that may occur in the syntax of an input statement. Each of these erros is detectable during Pass One operation. When an error is detected during Pass One operation, an appropriate diagnostic message will be generated. The exact mode of presentation for this diagnostic message is dependent on the device used for input of the new or modified source statements.

1-30. When the typewriter is used for input, it is assumed that the programmer (as opposed to an operator without knowledge of the program) is entering the lines of source code. When an error is detected, an appropriate diagnostic message will be printed out, and the program will request that the user re-enter the line in question. This mode of operation is especially useful for small modifications of existing source programs that are resident on magnetic tape or disc. Also, this feature is useful for initial use by a programmer who is not experienced in the use of the QANTEL Business Assembly Language.

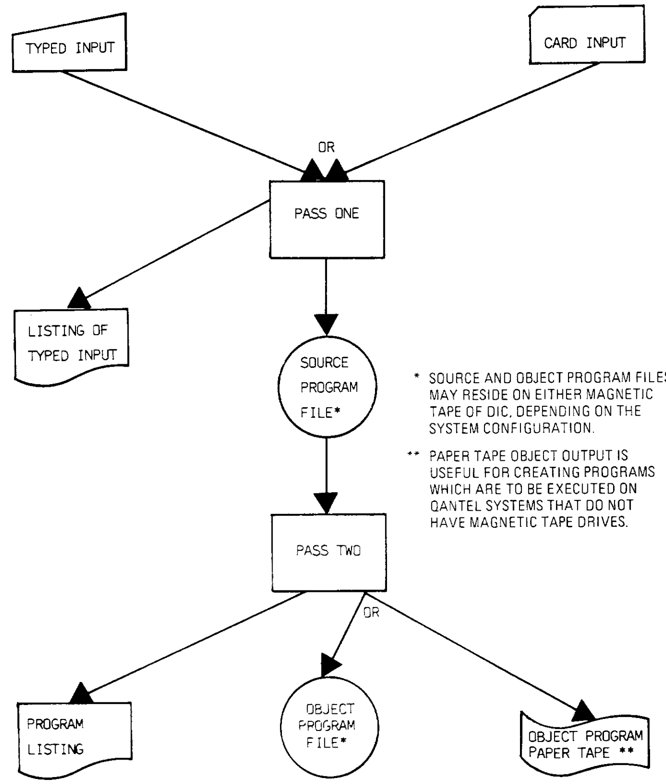


Figure 1-1. Assembler Create Operation

1-31. Using the card reader as an input device to the assembler is especially appropriate when a number of programmers are developing programs using the QANTEL Business Assembly Language, and an operator is available to supervise the assembly of these programs. When using the card reader, the operator cannot correct an erroneous input line at the time the error occurs, therefore, the Pass One program will mark the line in error on the source file, produce an appropriate diagnostic message on the file, and proceed to the next input line. When the resultant source file is processed by Pass Two of the assembler, these diagnostic messages will be printed following the lines in error. This will allow the programmer to code the appropriate corrections to his

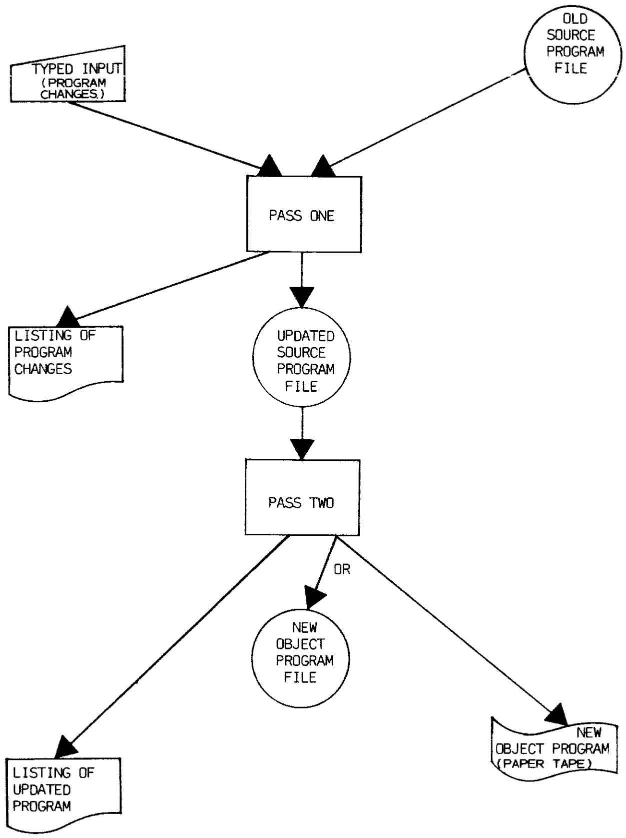


Figure 1-2. Assembler Update Operation

program and resubmit the corrections to the operator for updates. When the lines in error have been replaced or deleted by the subsequent updates of the source tape, the error messages will be ignored and not passed on when an updated source file is created.

1-32. **Pass Two**

1-33. As shown in figure 1-1, the Pass Two program is run subsequent to the Pass One operation to accept the source program file and produce an object program file. During this operation, the assembly language instructions are translated into machine language instructions. The result of the Pass Two operation is an absolute or machine language object program and a printed listing of the program. This listing shows the instructions in both the assembler and machine languages. The production of either the object program or the printed listing may be suppressed at the option of the operator or programmer. The listing of the source and machine language instructions will provide an aid to the programmer for debugging and future modification of the assembled program.

1-34. **Update Operation**

1-35. A previously assembled program may be updated by modifying the old source program, eliminating any need to re-enter the entire program. As shown in figure 1-2, the old source program can be modified by Pass One of the assembler using corrections or changes entered by way of the typewriter or card reader. The result of this update operation is a new, or updated, source program that may be assembled by the Pass Two operation of the Q/BAL Assembler programs. The final result of the update operation is a new source program, a new object program, and a new listing.

1-36. **Source Program Library Files**

1-37. As described in the previous paragraphs, the Pass One operation of the Q/BAL Assembler produces a source program. This source program can reside on either magnetic tape or disc. Some of the advantages of magnetic tape and disc source program files are described in the following paragraphs.

1-38. MAGNETIC TAPE SOURCE PROGRAM FILES

1-39. Magnetic tape source files can be arranged by the assembler and/or the library maintenance programs so that several programs can be contained on one reel of tape. Further details on this are presented in the Assembler Operating Instructions, Section VI. The Pass One operation permits the operator to request the desired source program file from the tape when necessary for updating purposes. Advantages of maintaining multiple source program files on one magnetic tape include a more efficient use of magnetic tape, and an ease of producing a single object file containing all programs of the library. This is especially advantageous when all programs are part of the same system. The primary disadvantage of using magnetic tape for multiple source program storage is the extended processing time required to copy programs not being updated when those programs contain many lines of code.

1-40. DISC SOURCE PROGRAM FILES

1-41. The use of disc for source program files allows assembly of programs on systems configured with one disc drive and one magnetic tape unit. In addition to all of the advantages of magnetic tape source files, the use of disc source files also offers the user the advantage of greater speed and flexibility that is not available when using magnetic tape source files.

1-42. **Object Program Library Files**

1-43. Like source programs, object programs can be maintained on either magnetic tape or disc. However, the assembler will create object files on magnetic tape only. The differences in the speed and convenience of disc over magnetic tape object files is great. This is especially so when it becomes necessary to mount and dismount tape reels. The object program library files can be created or updated by either of the two following methods:

a. Adding, replacing, or deleting individual object programs on the library file by means of the appropriate magnetic tape or disc library maintenance program.

b. Performing the Pass Two operation on the entire source program library file (magnetic tape output only).

1-44. Any object program produced by the QANTEL Business Assembly Language program can be placed on an existing library file using the appropriate QANTEL library maintenance program. Figure 1-3 illustrates how the object programs are fed into the library routine with the existing library file to produce an updated or new library file.

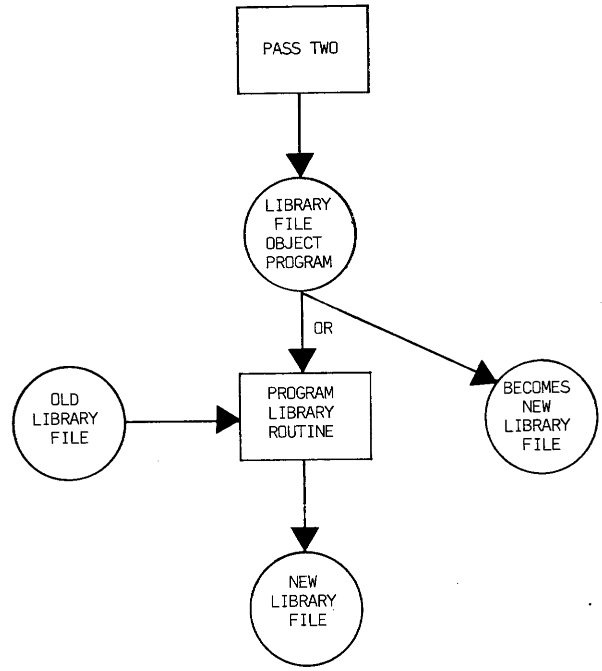
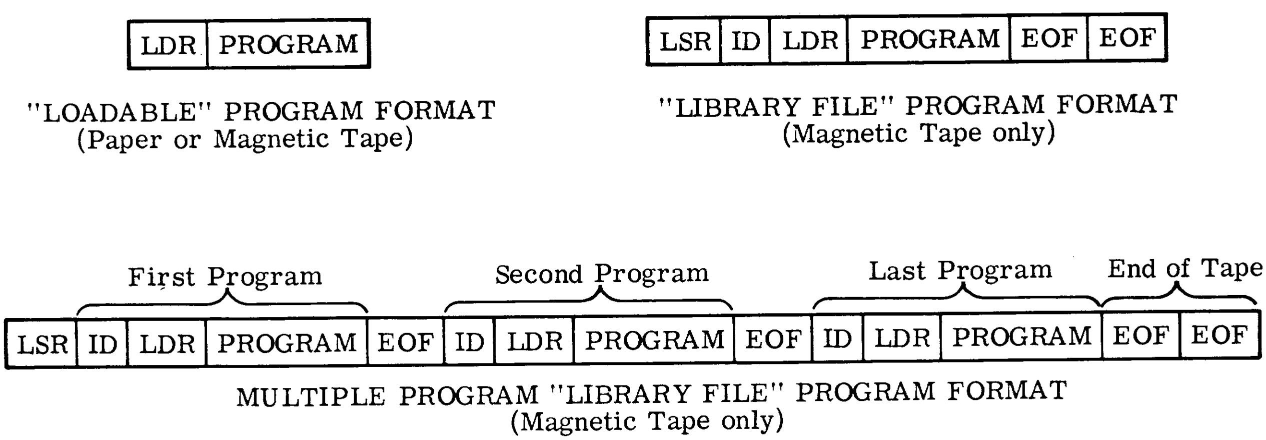


Figure 1-3. Program Library Files

1-45. LOADABLE OBJECT PROGRAMS

1-46. A loadable object program is one that has only its own loader on the first portion of the object tape, immediately preceeding the object program. Refer to figure 1-4. All object programs produced by QANTEL Business Assembly Language programs are of this type. However, the magnetic tape and disc object programs produced by the assembler programs are usually object programs that constitute a library file containing one or more object programs.



LSR = Library Search Routine
LDR = Loader Routine
ID = Program Seven-Character Identification
EOF = End-of-File Indicator (hexadecimal 13)

Figure 1-4. Object Program Formats

1-47. LIBRARY OBJECT FILES

1-48. The library file object program consists of one or more loadable object programs identified by a unique (within the file) seven character program identifiers that precedes the object program loader. See figure 1-4. The first record of the object tape contains a Library Search Routine (LSR) capable of searching the object tape file for a routine specified by the operator. End-of-File (EOF) characters are written on the magnetic tape to mark the end of each object program. Two consecutive EOF characters are written to mark the end of the entire library file. The library file may be used as a separate library or, it may be combined with other library files to construct a larger library file. This is accomplished by using the appropriate program library maintenance routine.

SECTION II

PROCESSOR FUNCTIONS

2-1. GENERAL

2-2. The QANTEL/ANSWER Processor System has 51 standard instructions that are hard-wired in the Read-Only Memory (ROM). See section I for a description of the ROM. These instructions include fourteen input/output processing and control instructions, and 37 data handling, arithmetic and logical, decision and control, and special instructions. This section provides a description of the various processor functions such as data and instruction formats, addressing structure, the interrupt feature, and the various switch settings. Also included are descriptions of all the standard instructions, except for I/O instructions, along with examples of both machine language instruction format and examples of assembly language instruction coding. The I/O instructions are described fully in section V of this manual.

2-3. DATA FORMAT

2-4. The basic unit of information used in the QANTEL/ANSWER Processor System is the byte. Each byte is made up of eight binary bits and can represent an alphabetic letter, a numerical digit, a special character, or a hexadecimal number from 00 to FF. The byte is the smallest addressable unit in the processor, which has a storage capacity of either 4096, 8192, 16384, 24576, or 32768 such bytes. Each of the eight bits within a byte is identified by its binary weight expressed in powers of two as illustrated in figure 2-1. Throughout QANTEL publications, the low order bits are always placed to the right side of the byte. The internal code used in the processor is ASCII. Appendix B of this manual shows a complete ASCII table. ASCII is used to represent all alphameric characters, operands, and special control characters (e.g., communications).

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Figure 2-1. Eight-bit Byte

2-5. INSTRUCTION FORMAT

2-6. Single-Address Instructions

2-7. The QANTEL/ANSWER Processor is a single-address and two-address computer. When single address instructions are used, the instruction operand address becomes the address of the A operand field. An accumulator, which occupies the low order 16 positions of main memory, becomes the implied B operand field. The format of the single-address instruction is basically the same as that of the two-address instruction (described in the following paragraphs), except that it consists of only three (instead of six) bytes. Figure 2-2 illustrates the machine language format of a single-address instruction and figure 2-3 shows a sample of single-address instruction coding.

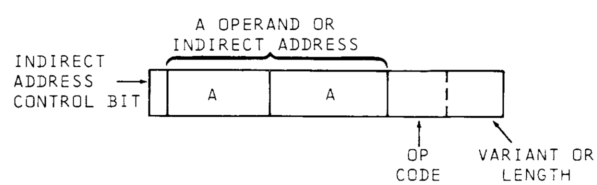


Figure 2-2. Single-Address Instruction Machine Language Format

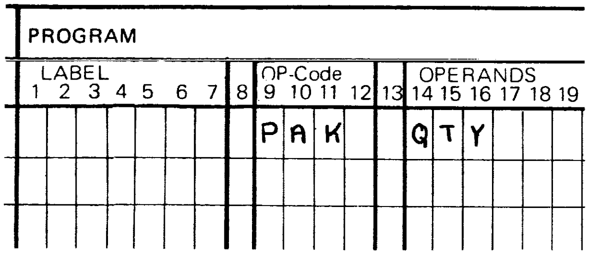


Figure 2-3. Single-Address Instruction Coding Example

2-8. Two-Address Instructions

2-9. All two-address instructions are made possible by the Load Address instruction which indicates to the processor that the instruction is to be treated as the two-address type. The Load Address instruction is described fully in paragraph 2-134. In the fetch cycle for the two-address instruction, the operation code included with the first instruction (Load Address instruction) initiates a sequence that continues fetching, so that both operands are identified prior to the execution.

2-10. In the two-address instruction (with the exception of the Read or Write I/O instruction), the operand address of the Load Address instruction is the address of the B operand field, and the operand address of the second instruction is the address of the A operand field. In either mode of addressing (single-address or two-address), the individual instructions are always three bytes in length, making a total of six bytes for a complete two-address instruction. The two-address instruction has the format shown in figure 2-4. Figure 2-5 shows a coding example of a two-address instruction.

NOTE

Refer to section V of this manual for a complete description of two-address I/O instructions. This includes the different types of Read and Write instructions.

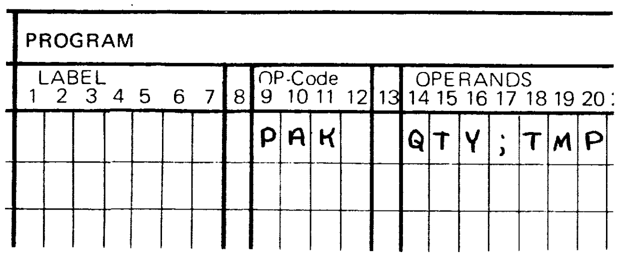


Figure 2-5. Two-Address Instruction Coding Example

NOTE

Individual field definition may differ; however, the definitions are explained as each instruction is examined later in this section.

2-11. Indirect Addressing

2-12. The most significant bit in any instruction is the indirect address control bit. If the indirect address control bit is in the "1" state, indirect addressing is indicated to the processor, and the instruction address is used as source of the operand address. If the most significant bit (of the most significant byte) of the indicated address is also in the "1" state, it is also used as an indirect address and the operation will continue until an operand address is found with the indirect address control bit in the "0" state. Indirect addressing is invaluable when it is necessary to reference a location that it is not convenient to address directly.

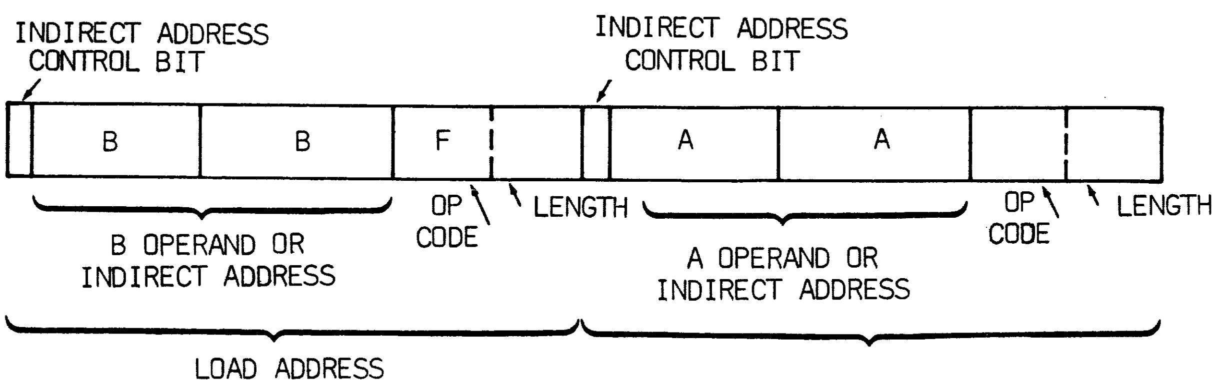


Figure 2-4. Two-Address Instruction Machine Language Format

NOTE

Indirect address looping is a possibility in the QANTEL/ANSWER Processor, and care should be exercised to insure proper setup of instruction formats.

2-13. Indirect addressing is valid for both the single-address and two-address instructions. This powerful feature can be used to index through a table, or for other operations that require changing addresses.

2-14. **Operation Code and Variant**

2-15. The most significant four bits of the least significant byte in any instruction is the operation code. See figures 2-2 and 2-4. The operation code, in conjunction with a variant (if required), indicates the type of operation to be performed by the instruction. The variant occupies the least significant four bits of the instructions, when used. All operation codes and variants are listed with their respective instruction definition and mnemonic in Appendix D at the rear of the manual. In addition, each QANTEL Instruction is described under the heading of Standard Instructions. Description is in sufficient detail to provide the programmer with a working knowledge of the QANTEL Standard Instructions.

2-16. **Instruction Length**

2-17. Instructions with operand that are variable in length (Add, Subtract, Store, etc.) carry length indicators in the instruction. When used, the length indicator occupies the least significant four bits of the instruction as shown in figure 2-6. Also, certain instructions (e.g. Move Numeric) utilize a length only when the two-address form is used. In this case, the length is specified in the Load Address instructions.

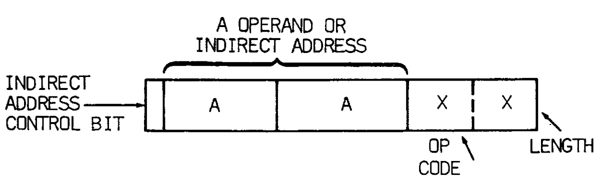


Figure 2-6. Machine Language Representation of Instruction Length

2-18. In the single-address mode, the length of the B operand field is 16 positions (the length of the accumulator).

NOTE

Using the two-address mode is often more economical in elapsed time, especially if the operand length is much less than maximum.

2-19. In all except the Input/Output instructions (see section V), the length indicator specifies the operand field length beginning at the least significant byte (highest memory position).

2-20. A length indicator of zero within a particular instruction specifies the maximum field length of 16 positions. However, a maximum field length of 256 positions can be used in the two-address Move instruction. Refer to paragraph 2-43 for a complete description of the two-address Move instruction. The hexadecimal length indicators are listed in table 2-1 with their respective lengths given in positions.

NOTE

The two-address Move instruction combines the length fields of the Move and associated Load Address instruction into a one byte (eight bits) length indicator. For a maximum length, 0 indicators are used to form a byte of 00_{16},$(00000000)_2$, or 256_{10}.

TABLE 2-1. INSTRUCTION LENGTH INDICATORS

LENGTH INDICATOR (Hexadecimal)	OPERAND FIELD LENGTH (Positions)
0	16
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

2-21. ADDRESSING STRUCTURE

2-22. The addressing structure of the QANTEL/ANSWER Processor System is binary. The lowest possible is zero, and the highest position (decimal) is 4095, 8191, 16383, 24575, or 32767, depending upon the number of memory modules installed in the system. Wrap around does not occur in the QANTEL/ANSWER Processor, and techniques using this operation should not be attempted. Add Binary and Subtract Binary instructions are a part of the standard instruction complement and provide a means of modifying an address. *The operand addresses given in all instructions, except for I/O instructions, specify the location of the least significant byte (highest memory position) of the operand field.* For example, if the operand address is 1000 (decimal), and the specified length is 10 (decimal), the specified operand field occupies positions 991 through 1000 (decimal).

2-23. RESERVED MEMORY

2-24. The first 32 positions of main memory (0 through 31) are reserved for use by the processor during the execution of a program. This portion of main memory serves as intermediate storage, arithmetic operands, I/O control areas, and micro-program utility areas. The reserved area is a convention in the standard QANTEL/ANSWER Processor System and is utilized as described in table 2-2 when the standard instruction set is installed. The locations in reserved memory are addressable by the programmer, and may be required in the normal mode of operations.

NOTE

Reserved memory areas should not be used for temporary storage unless the programmer completely understands processor treatment of these areas.

2-25. INTERRUPT FEATURE

2-26. The QANTEL/ANSWER Processor System can be equipped with a no-cost Interrupt Feature which allows the operator to interrupt the regular program by pressing the FLAG 1 pushbutton on the I/O typewriter. When the operator presses the FLAG 1 pushbutton, and the interrupt is not inhibited, the program instruction in progress is completed and the succeeding program instruction address is stored in positions 16 and 17 of

TABLE 2-2. RESERVED HIGH-SPEED (MAIN) MEMORY ALLOCATIONS

LOCATION (number in decimal)	DESCRIPTION
0-15	Accumulator positions. If the instructions are used in the single-address mode, the implied second operand is the accumulator and its contents.
16-17	Two bytes used to store the current program address when an interrupt occurs.
18	Single byte used to store the contents of the switches and the status of interrupt availability.
19-20	Two bytes which contain the address that replaces the current program address when an interrupt occurs.
21-22	Two bytes which contain the final address-plus-one for an I/O instruction. As the data is taken from or put into main memory, the address is incremented. When the last operation is performed, the address is incremented once more and then stored in bytes 21-22. By using this feature, the programmer can effect consecutive reads or writes to successive core locations from various I/O devices.
23	One byte which receives the I/O Control Byte when the Status-In instruction is executed. Status-In is described in section V of this manual.
24-25	Two bytes used to store the current program address when a branch instruction is executed on the (QANTEL V only), or the address of the match on a Branch Equal instruction on the QANTEL/ANSWER.
26-31	Micro-program utility bytes. Refer to the QANTEL Micro-Assembler Manual.

main memory. The settings of switches one, two, and three are stored in the least significant bits of main memory position 18 (see paragraph 2-30). The most significant bit of position 18 is set to the "1" state to inhibit any succeeding interrupt signals. The processor then branches to the interrupt routine located at the address stored in positions 19 and 20 of main memory. Refer to figure 2-7. Prior to using the Interrupt Feature, the correct interrupt address must be placed in positions 19 and 20 by the program, and the most significant bit of position 18 must be initially set to the "0" state.

NOTE

In systems employing more than one I/O device with an interrupt capability (such as a typewriter and communications modem), the programmer must determine which device is interrupting. Refer to section V.

2-27. After the program interrupt has occurred and the program state has been changed, operations placed in the interrupt routine take place until a Return-From-Interrupt (RTI) instruction is encountered. Normal interrupt routine operations could alter Reserved Memory locations used by the regular program (i.e. the accumulator) so that any instructions affecting these areas should be preceded by a sequence of instructions to save the data. This is especially true if the interrupt routine is to perform I/O instructions, since the termination address of the I/O instruction portion is automatically altered. A simple storing and restoring of the entire area is the simplest and safest way in insuring against these problems.

2-28. The Return-From-Interrupt (RTI) operation restores switches one, two, and three to their former settings (before the interrupt occurred) using the least significant three bits of position 18. It also resets the interrupt inhibit bit (most significant bit of position 18) to the "0" state (interrupt enable), and returns the program to the address stored in positions 16 and 17. Refer to figure 2-8.

2-29. The Interrupt Feature can be inhibited so that any attempted intervention by the operator has no immediate effect on the regular program. The inhibit action is accomplished by placing the most significant bit in main memory location 18 in the "1" state. (This occurs automatically whenever interrupt takes place to prevent a second interrupt.) Setting the bit to the "1" state may be done in the regular program as a result of

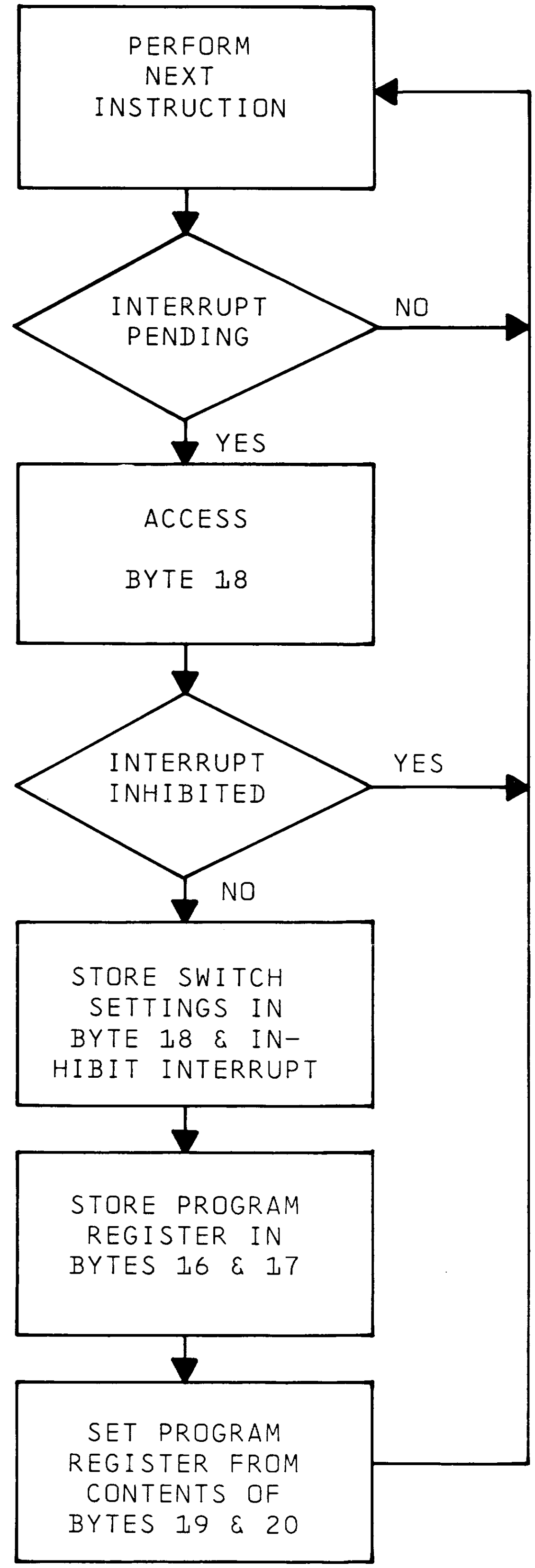


Figure 2-7. Interrupt Sequence in Fetch Cycle

an OR operation, or by simply moving a byte having the most significant bit in the "1" state into position 18. With the Interrupt Feature inhibited, any interrupt signals from the operator (by means of the FLAG 1 pushbutton) do not affect the regular program, but remain pending until the interrupt inhibit bit is reset to the "0" state. At this time, any pending interrupt will be acted upon in the manner previously described.

NOTE

A pending interrupt can be removed by pressing the FLAG 1 pushbutton a second time, so that the interrupt routine will not be entered when the interrupt inhibit bit is reset to the "0" state.

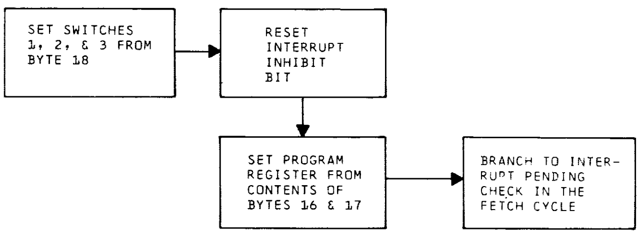


Figure 2-8. Return From Interrupt Instruction

2-30. SWITCH SETTINGS

2-31. The term switch in this reference manual refers to a hardware device within the QANTEL/ANSWER Processor System, which may be set and reset as a result of arithmetic and logical operations. At the conclusion of such an operation, the results can be determined by examining one or more switches and interpreting their settings. The programmer may examine the switch settings by means of the Branch On Overflow, Branch On Minus, Branch On Non-Zero, Branch Equal, and Branch Not Minus instructions. (Branch Not Minus is not available on the QANTEL Processors at this time and will be implemented at some future date.)

NOTE

The Branch instructions must be performed immediately after an arithmetic or logical operation due to the fact that any following instructions (except the Branch instructions) will alter the switch settings to be examined.

2-32. The five Branch instructions listed in the preceding paragraph examine the settings of three (3) separate switches within the processor and act upon these settings to replace the current program address with the branch address.

2-33. Switch One

2-34. Switch one is referred to as the "carry" switch. When an addition takes place in the processor, switch one is set to the "1" state if the result of the addition creates carry (overflow). This condition is input to the next add operation, unless the length has been exhausted, in which case, no add takes place. In this situation, the carry remains and switch one can be examined by the Branch On Overflow instruction.

2-35. Switch Two

2-36. Switch two is referred to as the "minus" switch, and is set to the "1" state if the result of an arithmetic operation is a negative quantity. The setting of switch two may be examined by the Branch On Minus and Branch Not Minus instructions.

2-37. Switch Three

2-38. Switch three is referred to as the "non-zero" switch. When set to the "1" state, switch three indicates that the result of an arithmetic, a logical, or an edit operation is something other than all zero bits. The Branch On Non-Zero instruction examines switch three for the "1" state (non-zero result).

2-39. In addition, switch three can be used to indicate equal or zero result from one of the Compare instructions, or any arithmetic and logical or decision control instructions. A Compare in the QANTEL/ANSWER Processor System is actually a subtract operation, except that neither operand is altered, and the result is reflected in the setting of the switches. For example, if two operands are equal and are compared (subtracted), the result of the operation would be zero, and switch three would be set to the "0" state to indicate the zero result, which in the Compare operation would be an equal result. Conversely, the comparison of two unequal operands would yield a non-zero result, and set switch three to the "1" state. The result of a Compare instruction can be checked for equality by means of either the Branch On Equal or Branch On Non-Zero instruction.

2-40. STANDARD INSTRUCTIONS

2-41. The following paragraphs provide working descriptions of each QANTEL standard instruction. The instructions are divided into five categories as follows:

1. Data Handling Instructions
   a. Move – MOV
   b. Store Accumulator – STA
   c. Load – LD
   d. Edit – EDT
   e. Unedit – UED
   f. Move Numeric – MN
   g. Move Zone – MZ
   h. Shift Bit Left – SBL
   i. Shift Bit Right – SBR
   j. Pack – PAK
   k. Unpack – UPK
   l. Translate – TRN
2. Arithmetic and Logical Instructions
   a. Add Decimal – ADD
   b. Subtract Decimal – SBD
   c. Multiply Decimal – MPY
   d. Divide Decimal – DIV
   e. Add Binary – ADB
   f. Subtract Binary – SBB
   g. And – AND
   h. Or – OR
   i. Exclusive Or – XOR
3. Decision and Control Instructions
   a. Branch On Overflow – BOV
   b. Branch On Minus – BMI (BGT – Branch Greater Than)
   c. Branch On Non-Zero – BNZ (BNE – Branch Not Equal)
   d. Branch Equal – BEQ (BZ – Branch On Zero)
   e. Branch Not Minus – BNM (BLE – Branch Less Than or Equal)
   f. Unconditional Branch – BRU
   g. Halt and Branch – HLT
   h. Branch and Link – BLI
   i. Test Bit – TBT
   j. Compare Decimal – CD
   k. Compare Logical – CMP
   l. Return From Interrupt – RTI
   m. No Operation – NOP
4. Special Instructions
   a. Load Address – LDA
   b. Micro-Instruction Mode – MIM
   c. Search Equal – SEQ
5. Input/Output Instructions
   a. Read – RD
   b. Read and Count – RDC
   c. Read Hex (hexadecimal) – RHX
   d. Read Hex and Count – RHC
   e. Write – WR
   f. Write and Count – WRC
   g. Write Hex – WHX
   h. Write Hex and Count – WHC
   i. Status-In – SIN
   j. Read Status 2 – RS2
   k. Set Read – SRD
   l. Reset I/O – RIO
   m. Device Control – CTL
   n. Seek – SEK

The Input/Output instructions are described in section V.

NOTE

A complete listing of all QANTEL standard instructions with their mnemonics, operation codes, and variants is presented in Appendix D.

2-42. Data Handling Instructions

2-43. MOVE – MOV (OP CODE 6)

2-44. Move (MOV) is a two-address instruction that transfers consecutive bytes from the A operand to the B operand. The B operand field is determined by the Load Address instruction (all two-address instructions contain the Load Address instruction), and the A operand address is taken from the Move instruction. During the operation, the four bit length fields of the two individual instructions (Load Address and Move) are combined within the processor to form one eight-bit length field. The four-bit length field of the Load Address instruction becomes the high order four bits of the composite eight-bit length field, while the four-bit length field of the Move instruction becomes the low order four bits of the eight-bit length field. This feature permits the programmer to move any number of consecutive bytes, up to 256, from the A operand to the B operand. An example of a 255 byte transfer using the Move instruction is shown in figure 2-9. Figure 2-10 is a coding example of this type of operation.

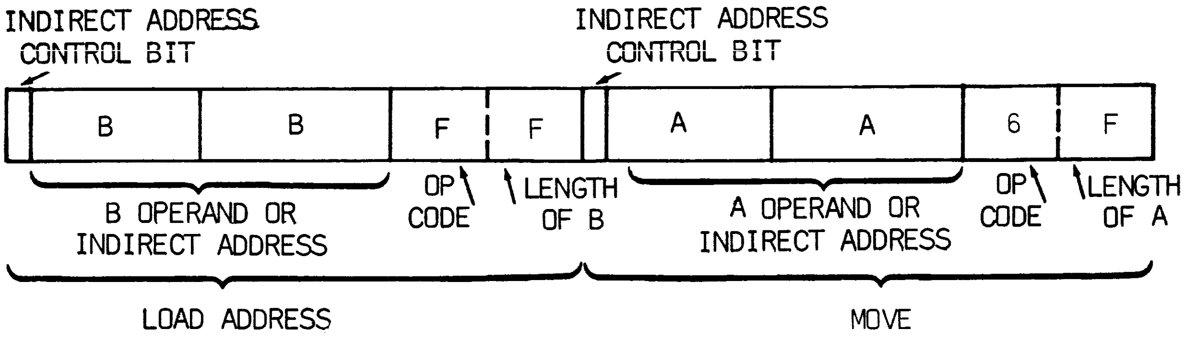


Figure 2-9. Move Instruction Machine Language Format

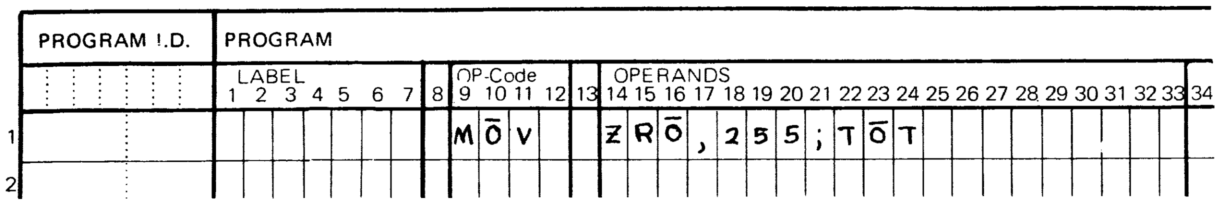


Figure 2-10. Move Instruction Coding Example

2-45. STORE ACCUMULATOR – STA (OP CODE 6)

2-46. Store Accumulator is a single-address instruction which transfers consecutive bytes from the accumulator (implied B operand) to the A operand. The length field in the Store Accumulator instruction determines the number of bytes transferred from the accumulator, beginning with the least significant byte (main memory position 15), and progressing toward the most significant byte, until the A operand length is exhausted. If the Store Accumulator instruction is preceded by a Load Address instruction, the operation transfers data from A to B, instead of from B to A. The operation code for both Move and Store Accumulator is 6. Figure 2-11 shows the machine language format for Store Accumulator and figure 2-12 is a sample of coding.

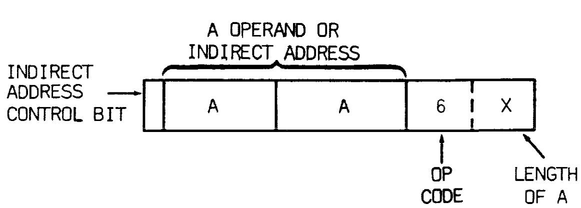


Figure 2-11. Store Accumulator Instruction Machine Language Format

2-47. LOAD – LD (OP CODE 5)

2-48. Load is either a single-address or two-address instruction used to transfer consecutive bytes from the A operand to the B operand. This instruction acts as a clear and add instruction. When used as a single-address instruction, the accumulator is loaded with the contents of the A operand so that the least significant byte of the A operand is transferred to location 15, the least significant position of the accumulator. The operation continues transferring each consecutive byte (from the least significant to the most significant) until the A operand length (as specified in the instruction) is exhausted. If the A operand length is less than the B operand length, decimal zeroes are supplied as additional A operand characters until the B operand length is exhausted. If the A operand length is greater than the B operand length, the number of bytes transferred is equal to the B operand length. Figure 2-13 shows the machine language format for the single-address Load instruction and figure 2-14 is a coding example.

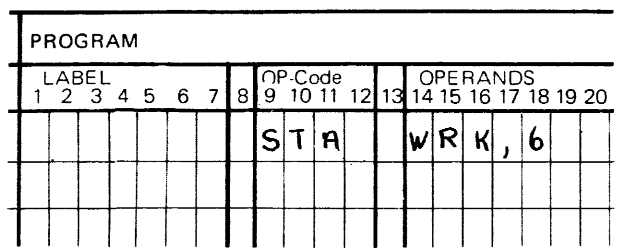


Figure 2-12. Store Accumulator Instruction Coding Example

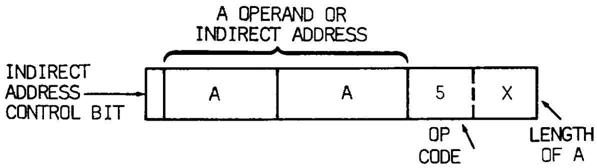


Figure 2-13. Single-Address Load Instruction Machine Language Format

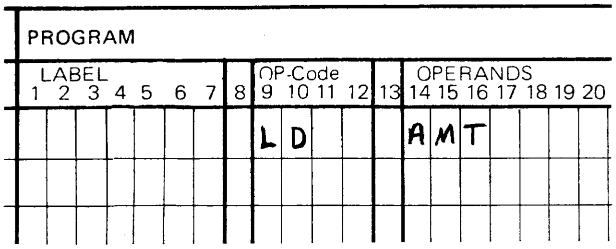


Figure 2-14. Single-Address Load Instruction Coding Example

2-49. In its two-address format, the Load instruction is preceded by the Load Address instruction and permits the transfer of consecutive bytes from the A operand to any specified B operand. This amounts to "floating" the accumulator. The execution cycle of the Load instruction checks the units position of the A operand for sign. The Branch On Minus instruction is used to check for a negative sign. Figure 2-15 shows a coding example of the two-address Load instruction.

2-50. EDIT - EDT (OP CODE 9, VARIANT 9)

2-51. Edit is a single-address or two-address instruction in which the B operand is edited under control of the A operand mask, and is placed in the A operand. That is, the data moves from B to A. In the two-address Edit instruction, the length of the B operand is specified

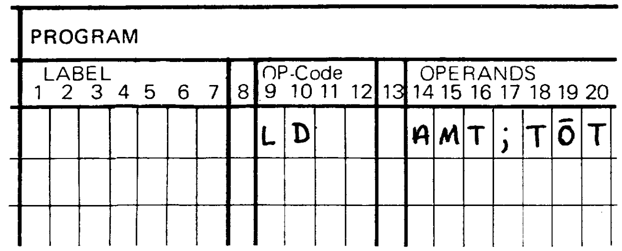


Figure 2-15. Two-Address Load Instruction Coding Example

in the Load Address instruction. Figure 2-16 shows the machine language format of the two-address Edit instruction and figure 2-17 is a coding example. In the single-address Edit instruction, the length is assumed to be the accumulator length. Figure 2-18 shows the machine language format of the single-address Edit instruction and figure 2-19 shows a coding example of the single-address Edit instruction.

2-52. The Edit operation performs a right to left scan of the A operand in order to determine the most significant character of the edit mask. This is done by allowing the least significant position of the mask as a sign indicator, then counting one position to the left for each position of the data field. The count does not include commas or periods. A left to right pass then enters the data into the mask (A operand) field according to the following rules:

1. The most significant character of the mask is used as a fill character.
2. Significance is established by a non-zero character in the B operand, or by a zero in the mask.
3. If significance is not established, the fill character replaces the mask character (including commas and periods).

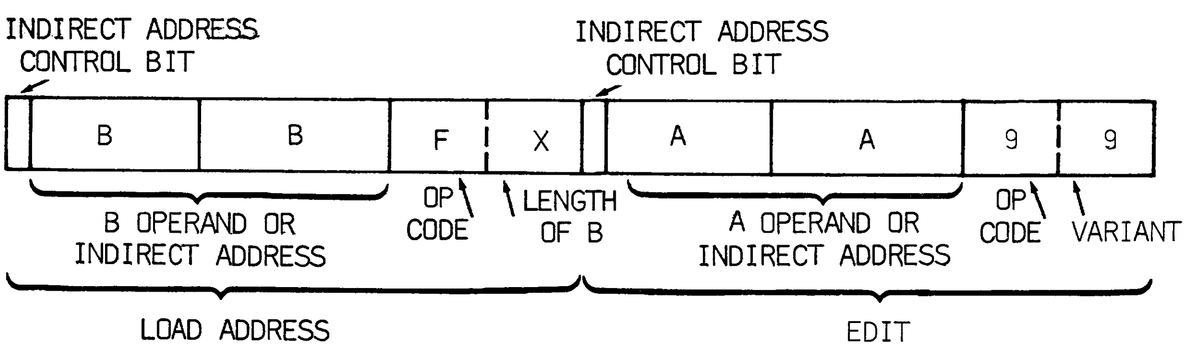


Figure 2-16. Two-Address Edit Instruction Machine Language Format

Line	LABEL	OP-Code	OPERANDS
1		EDT	MSK,7;DAT
2			
3			
4			
5			
6	MSK	DC	'$b̸,b̸b̸Ø.b̸b̸-'
7	DAT	DC	Ø112345
8	PRT	DA	1Ø
9		MŌV	MSK;PRT
10		EDT	PRT,7;DAT
11			

Figure 2-17. Two-Address Edit Instruction Coding Example

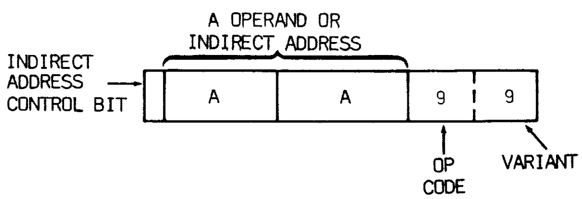


Figure 2-18. Single-Address Edit Instruction Machine Language Format

LABEL	OP-Code	OPERANDS
	EDT	WRK

Figure 2-19. Single-Address Edit Instruction Coding Example

4. If significance is established, the data from the B operand is entered into the mask, unless the mask is a period or comma. If the mask character examined is a period or comma, it is left untouched and the next mask character is examined. This continues until neither a period or comma is encountered, and the data character can be entered into the A operand field.

5. If the data field is negative, the least significant character of the mask is unchanged, and the sign of the least significant digit is changed to positive. If the data field is positive, the least significant position of the mask is replaced by a blank.

2-53. The following example of the Edit operation is provided to further clarify the preceding edit rules. Figure 2-17 shows the coding for the following example.

B operand (data) 0112345

A operand (mask)$b̸,b̸b̸Ø.b̸b̸

A operand result $1,123.45

NOTE

The execution cycle of the Edit instruction checks the B operand for sign. A negative result can be checked with the Branch On Minus instruction.

2-54. UNEDIT - UED (OP CODE 9, VARIANT A)

2-55. Unedit is a single-address or two-address instruction in which the A operand is changed from an edited form to an unedited form, and placed in the B operand. When used as a two-address instruction the Load Address instruction indicates the address and

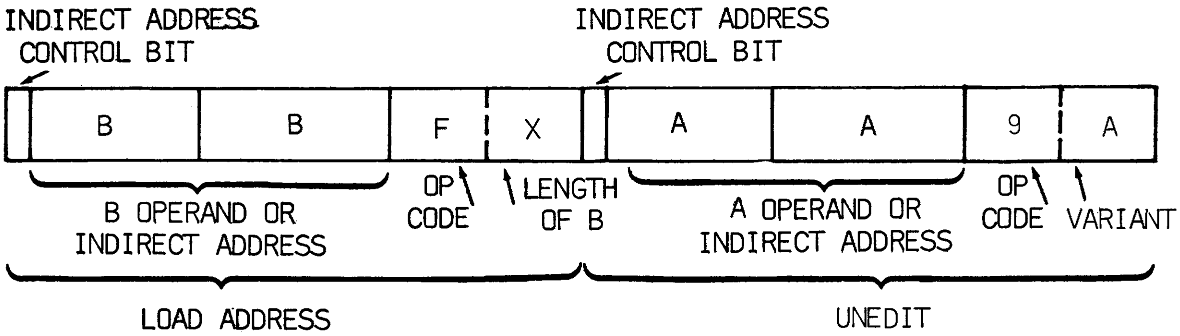


Figure 2-20. Two-Address Unedit (UED) Instruction Machine Language Format

length of the B operand. The length of the A operand must be equal to the length of the B operand. When the Unedit instruction is used in the single-address form, the implied B operand is the accumulator with a length of 16 characters. Figure 2-20 shows the machine language format of the two-address Unedit instruction and figure 2-21 is a coding example of UED. Figure 2-22 shows the machine language format of the single-address Unedit instruction and figure 2-23 is a coding example of UED.

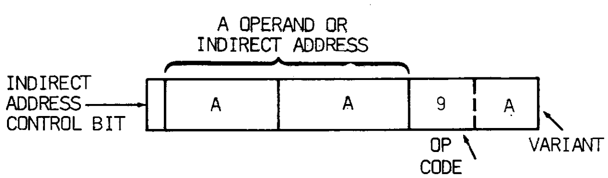


Figure 2-22. Single-Address Unedit Instruction Machine Language Format

2-56. In the unedit operation, the B operand is constructed by scanning the A operand from right to left and moving all numeric characters to the B operand, right justified. The B operand is filled to the left with decimal zeroes. If the scan of the A operand encounters a minus (-) before the least significant digit, a minus zone is placed over the low order byte of the B operand.

2-57. The following of the Unedit operation is provided to further clarify the preceding statements. Figure 2-21 is a coding example of the following Unedit operation.

B operand (miscellaneous data)	XYZ1234Z$X
A operand (edited form)	$1,123.45Ø
B operand result	0000112345

PROGRAM

LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–20)
		UED		WRK

Figure 2-23. Single-Address Unedit Instruction Coding Example

	PROGRAM I.D.	PROGRAM				
		LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–34)
1				UED		DAT,1Ø;MSC
2						
3						
4		DAT		DC		$1,123.45Ø
5		MSC		DA		1Ø
6						

Figure 2-21. Two-Address Unedit (UED) Instruction Coding Example

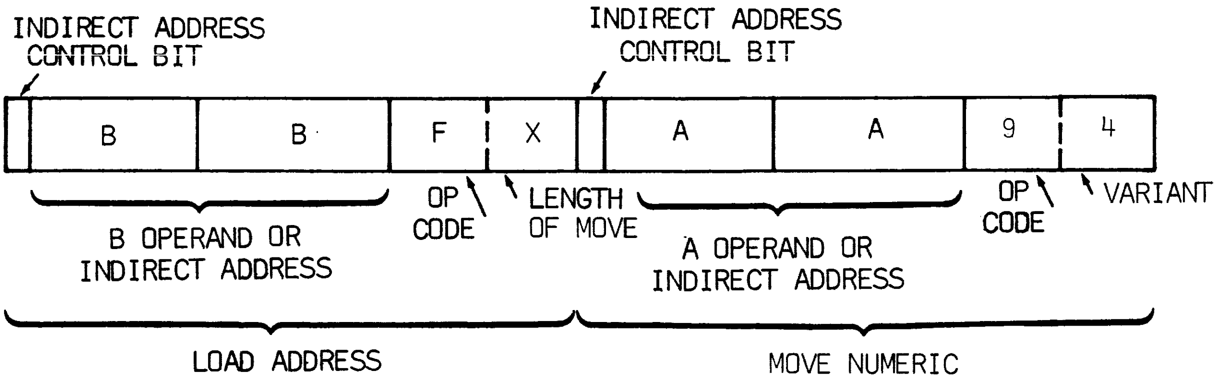


Figure 2-24. Two-Address Move Numeric (MN) Instruction Machine Language Format

---

NOTE

The execution cycle of the Unedit instruction checks the B operand result for sign. Negative sign can be checked with the Branch On Minus instruction.

2-58. MOVE NUMERIC - MN (OP CODE 9, VARIANT 4)

2-59. Move Numeric is a single-address or two-address instruction in which the low order four bits of each A operand byte are transferred to the low order four bits of the corresponding B operand bytes. In the two-address instruction, the B operand is indicated by the operand address of the Load Address instruction, and the length of the move is controlled by the length field of the Load Address instruction. Figure 2-24 shows the machine language format of the two-address Move Numeric instruction and figure 2-25 is a coding example. In the single-address instruction, the implied B operand is the accumulator, and the length is assumed to be the maximum of 16. Figure 2-26 shows the machine language format of the single-address Move Numeric instruction and figure 2-27 is a coding example. During the Move Numeric operation, the A operand and the high order four bits of each byte in the B operand remain unchanged.

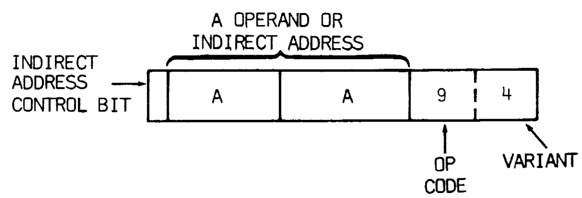


Figure 2-26. Single-Address Nove Numeric (MN) Instruction Machine Language Format

PROGRAM		
LABEL 1 2 3 4 5 6 7 8	OP-Code 9 10 11 12 13	OPERANDS 14 15 16 17 18 19 20
	MN	LTR

Figure 2-27. Single-Address Move Numeric (MN) Instruction Coding Example

2-60. MOVE ZONE - MZ (OP CODE 9, VARIANT 5)

2-61. The Move Zone instruction performs a similar function to that of the Move Numeric instruction, except that the high order four bits of each A operand byte are transferred to the high order four bits of the corresponding B operand bytes. During the Move Zone

	PROGRAM I.D.	PROGRAM		
		LABEL 1 2 3 4 5 6 7 8	OP-Code 9 10 11 12 13	OPERANDS 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
1			MN	LTR,1;LTR-3
2				

Figure 2-25. Two-Address Move Numeric (MN) Instruction Coding Example

operation, the A operand and the low order four bits of each byte in the B operand remain unchanged. The Move Zone and Move Numeric instructions are both written in the basic instruction format, but with different variants in the least significant four bits of the instruction. Figure 2-28 and 2-29 show the machine language format and a coding example of the two-address Move Zone instruction, and figures 2-30 and 2-31 show the machine language format and a coding example of the single-address Move Zone instruction.

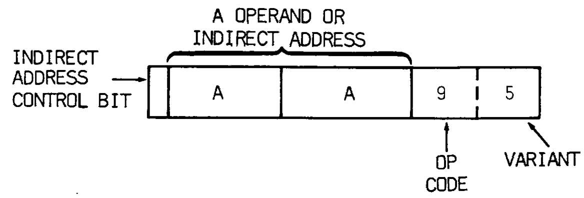


Figure 2-30. Single-Address Move Zone (MZ) Instruction Machine Language Format

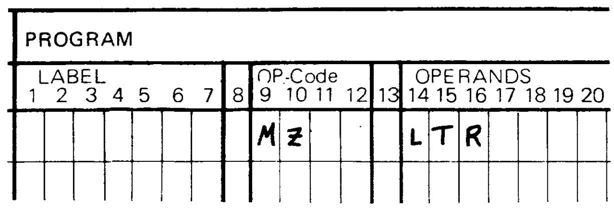


Figure 2-31. Single-Address Move Zone (MZ) Instruction Coding Example

2-62. SHIFT BIT LEFT - SBL (OP CODE 9, VARIANT 7)

2-63. Shift Bit Left is a single-address instruction used to shift the contents of a single byte (indicated by the operand address), one bit to the left. Because the instruction operates upon only one byte, no length information is required or provided in the instruction. During execution of the Shift Bit Left instruction, the byte is internally checked for overflow and non-zero conditions. Overflow may be checked with the Branch On Overflow instruction, and non-zero results may be checked with the Branch On Non-Zero instruction. Figure 2-32 illustrates the results of the Shift Bit Left operation. Figures 2-33 and 2-34 show the machine language format and a coding example of the Shift Bit Left instruction, respectively.

2-64. SHIFT BIT RIGHT - SBR (OP CODE 9, VARIANT 8)

2-65. Shift Bit Right is a single-address instruction used to shift the contents of a single byte, one bit to the right. As in the Shift Bit Left instruction, the operation

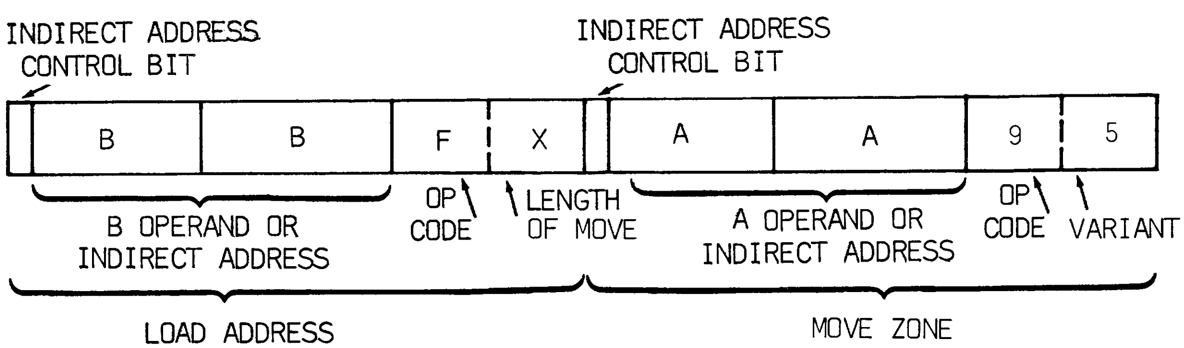


Figure 2-28. Two-Address Move Zone (MZ) Instruction Machine Language Format

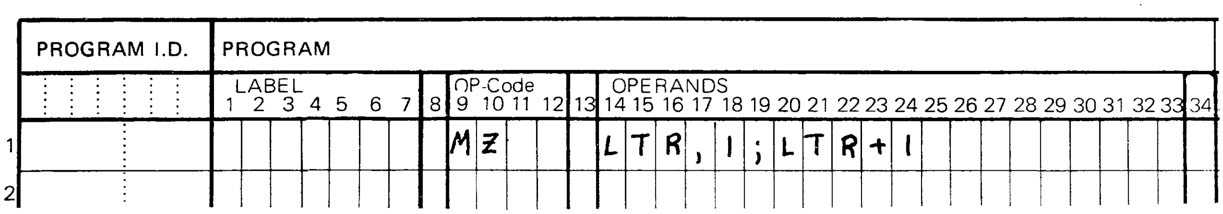


Figure 2-29. Two-Address Move Zone (MZ) Instruction Coding Example

Figure 2-32. Shift Bit Left Operation

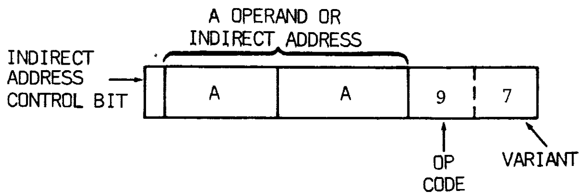


Figure 2-33. Shift Bit Left (SBL) Instruction Machine Language Format

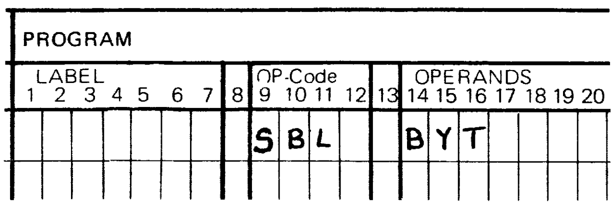


Figure 2-34. Shift Bit Left (SBL) Instruction Coding Example

takes place upon only one byte, eliminating the need for length information in the instruction. No overflow or non-zero checks can be made for the Shift Bit Right instruction. Figure 2-35 illustrates the results of the Shift Bit Right operation. Figures 2-36 and 2-37 show the machine language format and a coding example of the Shift Bit Right instruction, respectively.

2-66. PACK - PAK (OP CODE 9, VARIANT B)

2-67. Pack is a single-address or two-address instruction which alters the A operand from zoned format to

Figure 2-35. Shift Bit Right Operation.

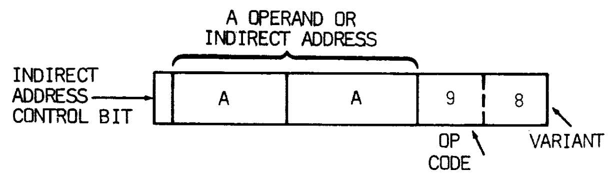


Figure 2-36. Shift Bit Right (SBR) Instruction Machine Language Format

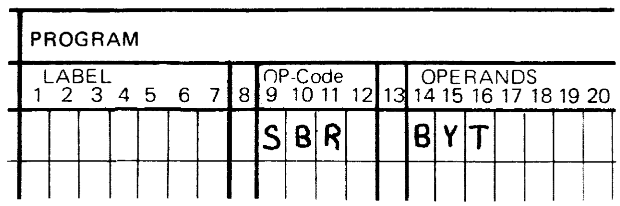


Figure 2-37. Shift Bit Right (SBR) Instruction Coding Example

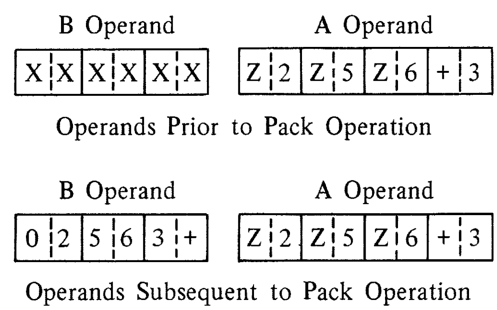


Figure 2-38. Pack Operation

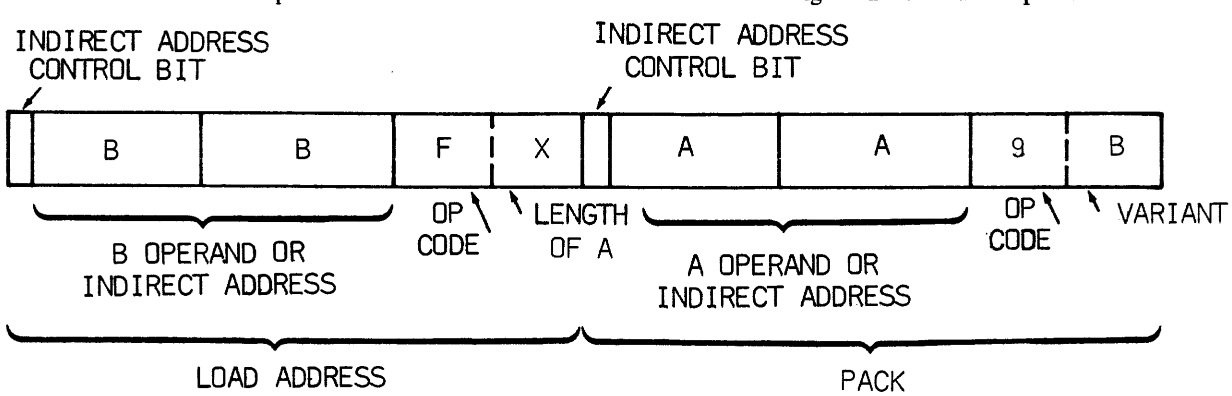


Figure 2-39. Two-Address Pack (PAK) Instruction Machine Language Format

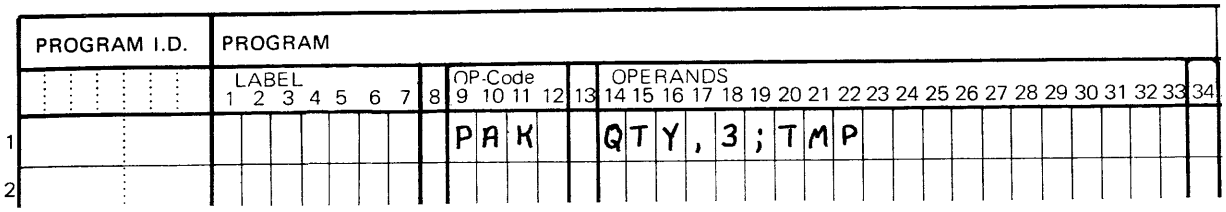


Figure 2-40. Two-Address Pack (PAK) Instruction Coding Example

packed format. The result of the Pack operation is placed in the B operand. As shown in figure 2-38, the zone bits of the least significant byte are interpreted as the sign, and a high order zero is inserted when the length of the unpacked field is an even number of bytes. Figures 2-39 and 2-40 show the machine language format and a coding example of the two-address Pack instruction, respectively.

2-68. When used as a single-address instruction, the B operand is the least significant 9 bytes of the accumulator, and the packed result is placed in these positions, beginning at the least significant position (memory position 15). The length of the unpacked field (A operand) in the single-address instruction is 16 positions. Figures 2-41 and 2-42 show the machine language format and a coding example of the single-address Pack instruction.

2-69. The length of the packed field, in either the single-address or two-address format is equal to $\frac{N}{2} + 1$, where N is equal to the length of the unpacked field (A operand) in the number of bytes.

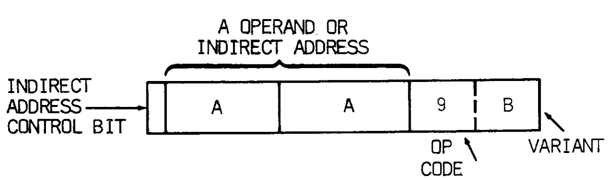


Figure 2-41. Single-Address Pack (PAK) Instruction Machine Language Format

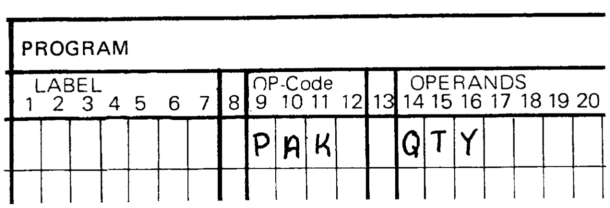


Figure 2-42. Single-Address Pack (PAK) Instruction Coding Example

---NOTE---

The Pack Operation transposes the high and low order bits of the least significant byte of the packed operand.

2-70. UNPACK - UPK (OP CODE 9, VARIANT C)

2-71. Unpack is a single-address or two-address instruction that alters the A operand from packed format to zoned format. The result of the operation is placed in the B operand. The length of the resulting unpacked field (B operand), which must be specific in the Unpack instruction, is equal to 2N − 1, where N is equal to the length of the packed field (A operand) in the number of bytes. In the single-address instruction, the length of the packed field is nine bytes (most significant half byte ignored). Figure 2-43 is an example of the Unpack operation. Figures 2-44 and 2-45 show the machine language format and a coding example of the two-address Unpack instruction. Figures 2-46 and 2-47 show the machine language format and a coding example of the single-address Unpack instruction.

B Operand

X	X	X	X	X	X	X	X	X	X

A Operand

6	1	2	7	5	+

Operands Prior to Unpack Operation

B Operand

Z	6	Z	1	Z	2	Z	7	+	5

A Operand

6	1	2	7	5	+

Operands Subsequent to Unpack Operation

Figure 2-43. Unpack Operation

2-72. TRANSLATE - TRN (OP CODE 8, VARIANT 2)

2-73. The Translate instruction is a seven-byte instruction that allows data stored in main memory to be translated into another form, i.e., ASCII to EBCDIC.

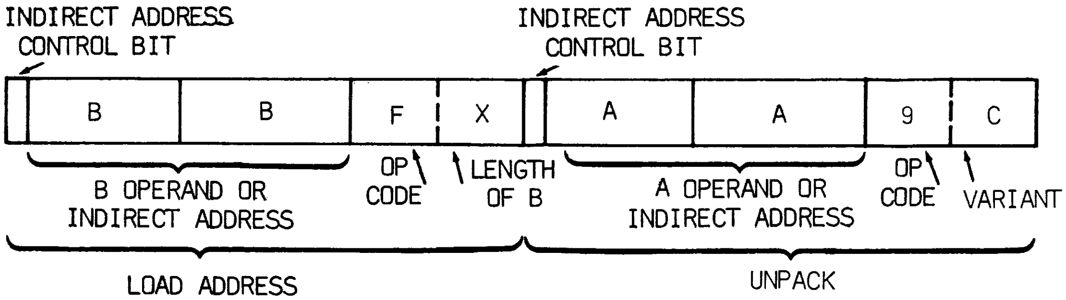


Figure 2-44. Two-Address Unpack (UPK) Instruction Machine Language Format

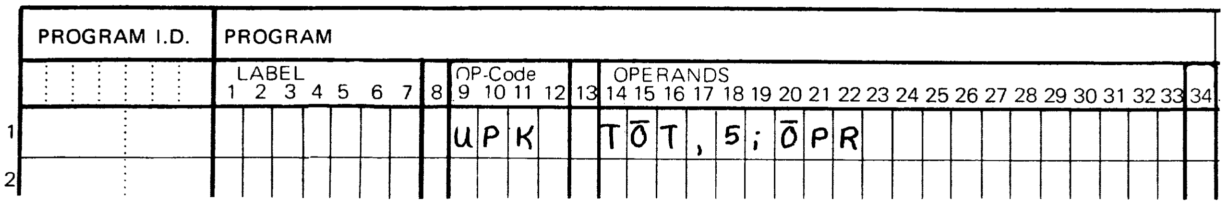


Figure 2-45. Two-Address Unpack (UPK) Instruction Coding Example

This instruction uses a translate table with a maximum length of 256 bytes that is stored in memory at any location divisible by 256, i.e., 256, 512, 1024, etc., as part of the program. The specified data is replaced in memory with the appropriate translation from the translate table by adding the value of the input data to the starting address of the translate table. Figure 2-48 shows the machine language format of the Translate instruction and figure 2-49 illustrates a method for coding this instruction.

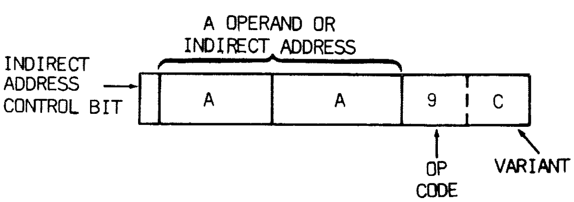


Figure 2-46. Single-Address Unpack (UPK) Machine Language Format

2-74. **Arithmetic and Logical Instructions**

2-75. DECIMAL ARITHMETIC INSTRUCTIONS - GENERAL

2-76. Decimal instructions can be performed as single-address or two-address instructions. In the single-address instruction, no Load Address instruction is used, and the accumulator becomes the implied B operand with a fixed length of 16 positions. The decimal instructions assume operand contents conforming to the

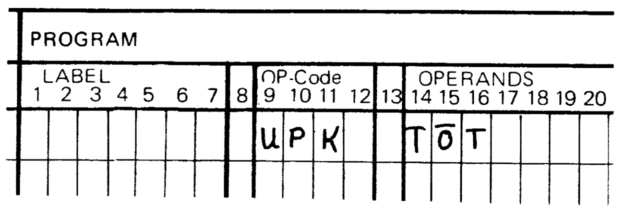
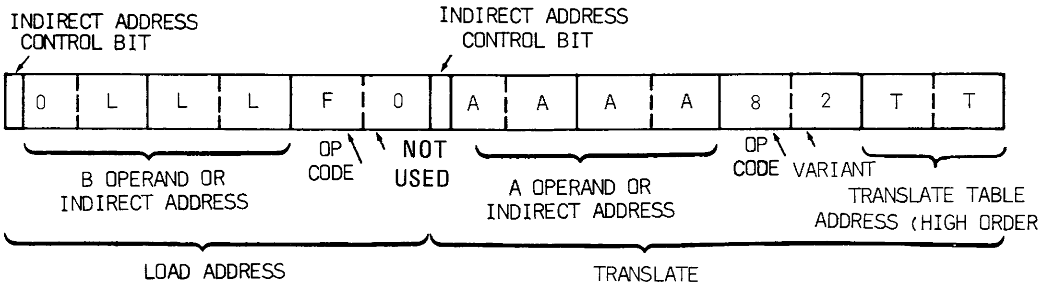


Figure 2-47. Single-Address Unpack (UPK) Instruction Coding Example

ASCII internal format, and no check is made to insure that operands have the correct format except in the case of the divide exception. Operands for decimal operations are signed numbers, with the controlling sign represented by the high order four bits in the low order (least significant) byte of each operand, as shown in the following illustration. After treating the least significant byte as a signed digit, the balance of the decimal operation performs the indicated arithmetic on the low order four bits (numeric digits) of each byte in the operand field. All zone bits except those of the least significant byte, are ignored during the operation. Figure 2-50 illustrates the decimal arithmetic operand field format.

2-77. Decimal instruction operands always have specified lengths and are addressed by the highest memory position (least significant digit) Decimal instructions are always signed operations that follow the rules of algebra, and are performed with the assumption that the field is unpacked with a sign over the least significant byte.



LLL = LENGTH OF INPUT DATA
AAAA = ADDRESS OF INPUT DATA (LOW ORDER)
TT = MOST SIGNIFICANT 8 BITS OF TRANSLATE TABLE ADDRESS (HIGH ORDER), LEAST SIGNIFICANT 8 BITS ARE ASSUMED TO BE 00.

Figure 2-48. Translate (TRN) Instruction Machine Language Format

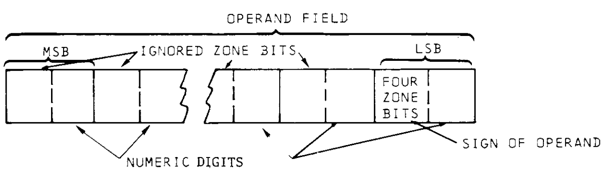


Figure 2-50. Decimal Arithmetic Operand Field

Decimal arithmetic uses the A operand as the first operand, and the B operand as the second and resultant operand. The resultant operand is extended to meet the length requirements of multiply operation results.

2-78. ADD DECIMAL - ADD (OP CODE 1)

2-79. Add Decimal is a single-address or two-address instruction in which the data found in the A operand is decimally (and algebraically) added to the data in the B

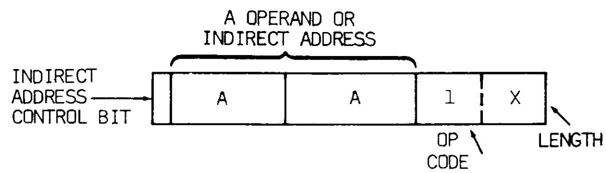


Figure 2-51. Single-Address Add Decimal (ADD) Machine Language Format

```
PROGRAM
LABEL        OP-Code   OPERANDS
             ADD       AMT
```

Figure 2-52. Single-Address Add Decimal (ADD) Instruction Coding Example

```
PROGRAM I.D.  PROGRAM
    LABEL        OP-Code   OPERANDS
1                ORG       $0100
2           INSERT TABLE
3                $
4   CRD          DA        80
5                $
6                TRN       CRD;80
7                DC        $01       TABLE ADDRESS
8                $
9
```

Figure 2-49. Translate (TRN) Instruction Coding Example

operand. The result of the ADD operation is placed in the B operand. In the single-address instruction, the implied B operand is the accumulator, and has a fixed length of 16 positions. Figures 2-51 and 2-52 show the machine language format and a coding example of the single-address Add Decimal instruction. If the A operand field is shorter than the B operand, zeroes are supplied in the A operand until the B operand length has been satisfied.

2-80. In the two-address instruction, the B operand address and length is indicated in the preceding Load Address instruction. Figures 2-53 and 2-54 show the machine language format and a coding example of the two-address Add Decimal instruction. As in the single-address mode, zeroes are provided as the A operand, if necessary, to obtain the same field length as the B operand. A typical two-address Add Decimal is shown in figure 2-54.

NOTE

If the specified length of A operand field is greater than that of the B operand field, the operation is terminated at the end of the B operand field with the truncated results in the B operand field. The truncated results reflect the sum of the numbers considered, with any high order numbers in the A operand ignored. The A operand is never altered in a decimal instruction.

2-81. The results of the Add Decimal operation are also indicated by the internal switch settings described in paragraph 2-30. The switch settings are examined by means of the Branch On Overflow, Branch On Minus, Branch Equal, or Branch On Non-Zero instructions. A Branch On Overflow indicates that when the B operand length was exhausted, carry into the next position was pending. Branch On Minus indicates a negative result, while Branch On Non-Zero indicates a result of something other than zero.

2-82. SUBTRACT DECIMAL - SBD (OP CODE 2)

2-83. The Subtract Decimal instruction shares all the rules and characteristics of the previously described Add Decimal instruction. The two instructions are differentiated by the operation code. Figures 2-55 and 2-56 show the machine language format and a coding example of the two-address Subtract Decimal instruction, and figures 2-57 and 2-58 show the machine language format and a coding example of the single-address instruction.

2-84. MULTIPLY DECIMAL - MPY (OP CODE 3)

2-85. Multiply Decimal is a single-address or two-address instruction in which the B operand is multiplied by the A operand and the result placed in the B operand.

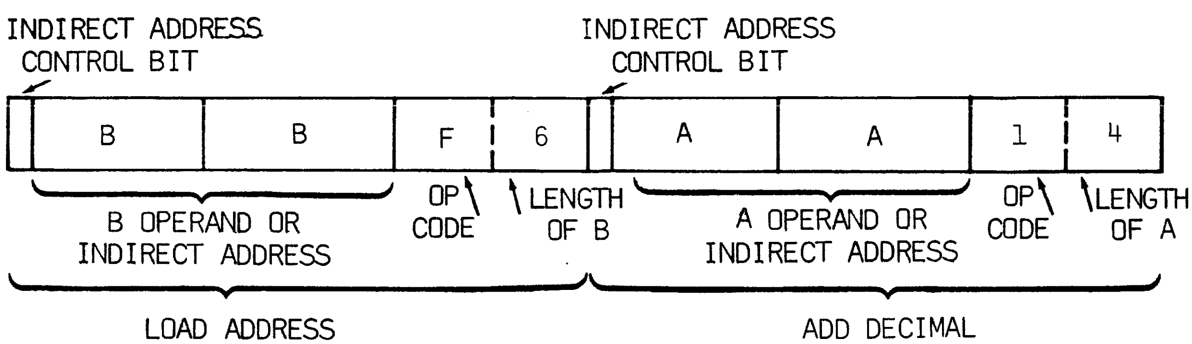


Figure 2-53. Two-Address Add Decimal (ADD) Instruction Machine Language Format

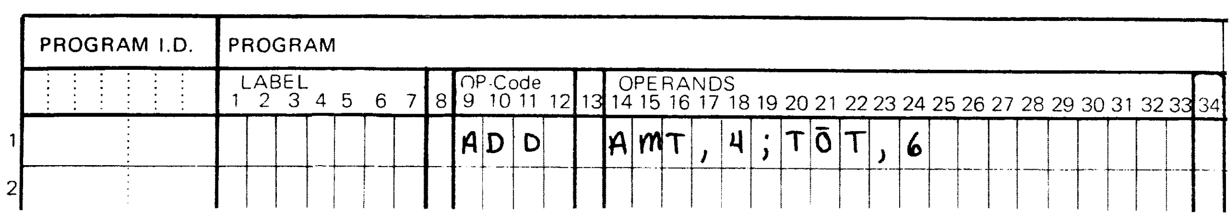


Figure 2-54. Two-Address Add Decimal (ADD) Instruction Coding Example

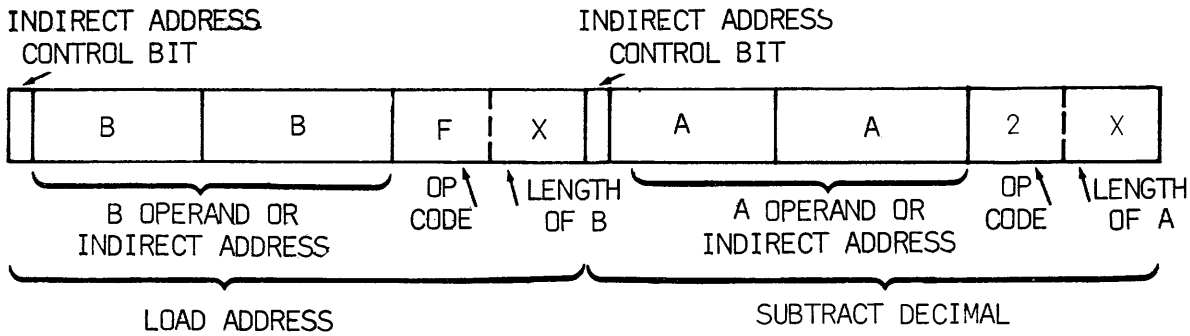


Figure 2-55.
Two-Address Subtract Decimal (SBD) Instruction Machine Language Format

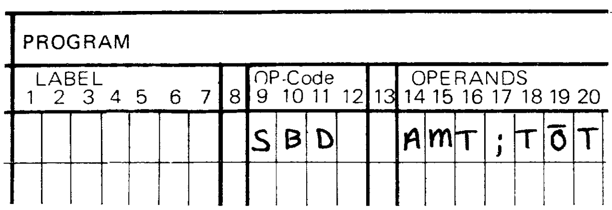


Figure 2-56. Two-Address Subtract Decimal (SBD) Instruction Coding Example

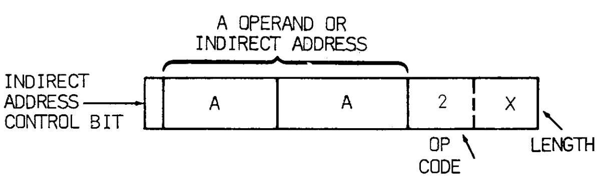


Figure 2-57. Single-Address Subtract Decimal (SBD) Instruction Machine Language Format

In the two-address instruction, the length of the B operand is indicated in the Load Address instruction, while the length of the A operand is indicated in the Multiply Decimal instruction. During the operation, the B operand is extended to a length equal to the sum of the A and B operand lengths. Extension of the B operand takes place in the high order positions. Figures 2-59 and 2-60 show the machine language format and a coding example of the two-address Multiply Decimal instruction.

NOTE

The single-address Multiply Decimal instruction should not be used.

2-86. As in the Add Decimal instruction, internal switch settings may be examined to determine the results of the Multiply operation by using the appropriate Branch instruction.

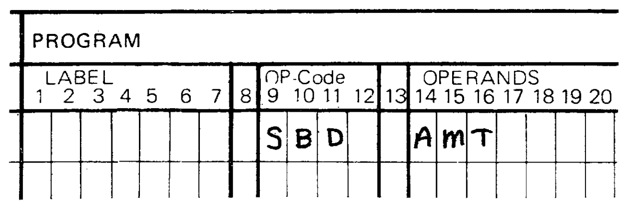


Figure 2-58. Single-Address Subtract Decimal (SBD) Instruction Coding Example

2-87. DIVIDE DECIMAL – DIV (OP CODE 4)

2-88. Divide Decimal is a single-address or two-address instruction in which the B operand is divided by the A operand with the algebraic results going to the B operand. In the two address instruction, the length of the B operand is indicated in the preceding Load Address instruction, while the length of the A operand is indicated in the Divide Decimal instruction. Figures 2-61 and 2-62 show the machine language format and a coding example of the two-address Divide Decimal instruction. In the single-address instruction, the accumulator becomes the implied B operand. After the division, the B operand field contains a signed quotient and remainder. The remainder is equal in length to the divisor and carries the sign of the divident. Figures 2-63 and 2-64 show the machine language format and a coding example of the single-address Divide Decimal instruction.

NOTE

The dividend field must contain sufficient leading zeroes to prevent the occurrence of the divide exception.

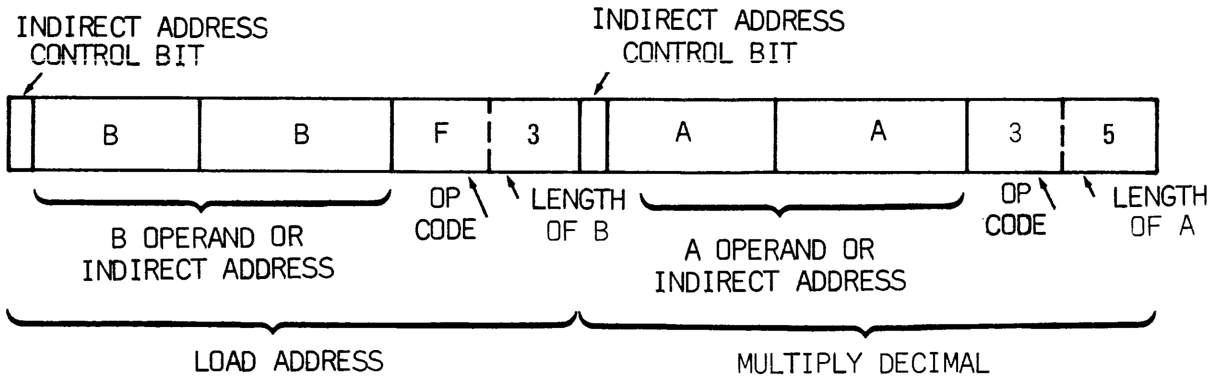


Figure 2-59.
Two-Address Multiply Decimal (MPY) Instruction Machine Language Format

PROGRAM I.D.	LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–)
1			MPY		HRS,3;RAT,5
2					

Figure 2-60. Two-Address Multiply Decimal (MPY) Instruction Coding Example

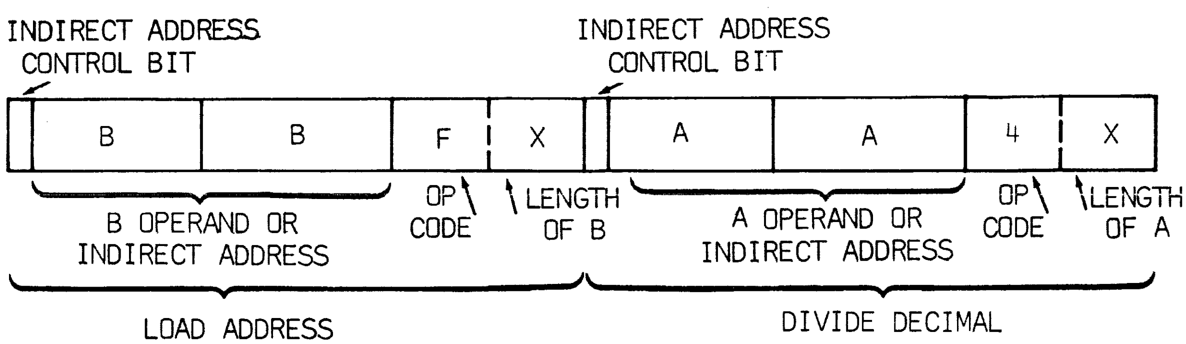


Figure 2-61
Two-Address Divide Decimal (DIV) Instruction Machine Language Format

LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–20)
		DIV		QTY;AMT

Figure 2-62. Two-Address Divide Decimal (DIV) Instruction Machine Language Format

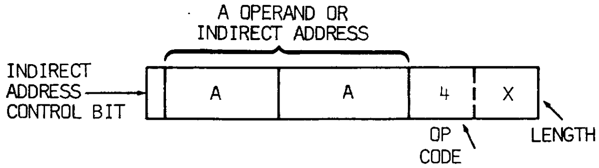


Figure 2-63. Single-Address Divide Decimal (DIV) Instruction Machine Language Format

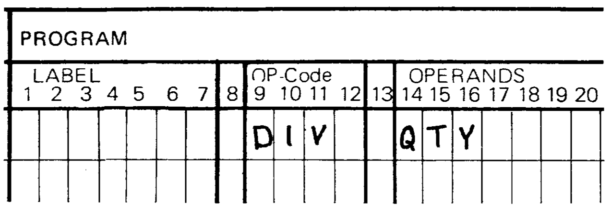


Figure 2-64. Single-Address Divide Decimal (DIV) Instruction Coding Example

2-89. During the divide operation, a trial subtraction is performed (using absolute values) of the divisor from the most significant positions of the dividend. The result of the subtraction should show the tested dividend positions to be less (in value) than the divisor. If the tested dividend positions are not less than the divisor, the divide exception occurs. Placing leading zeroes in the dividend is the simplest method of avoiding the divide exception. Divide exception also occurs when the length field of the divisor is greater than or equal to the length of the dividend.

2-90. If the divide exception is noted during the Divide Decimal operation, the overflow switch (switch one) described in paragraph 2-33 is set to the "1" state, the remainder of the divide operation is suppressed and both operands are left unaltered. Successful and unsuccessful divide formats are snown in the following examples.

$$\frac{323456}{022} = \text{EXCEPTION} \qquad \frac{00323456}{022} = \text{SUCCESSFUL}$$

2-91. ADD BINARY - ADB (OP CODE D)

2-92. Add Binary is a single-address or two-address instruction that is similar to the Add Decimal instruction. In the two-address Add Binary instruction the data stored at the A operand is binary added to the data stored in the B operand and the result of the operation is stored in the B operand. The maximum length for each operand field is 16 characters (bytes). Overflow can be checked by means of the Branch On Overflow instruction. Figures 2-65 and 2-66 show the machine language format and a coding example of the two-address Add Binary instruction. In the single-address instruction the accumulator becomes the implied B operand. The length of the implied B operand is assumed to be 16 characters and the result of the operation is stored in the accumulator. Figures 2-67 and 2-68 show the machine language format and a coding example of the single-address Add Binary instruction. Binary numbers in the QANTEL/ANSWER Processor System are unsigned and arithmetic is performed using absolute numbers.

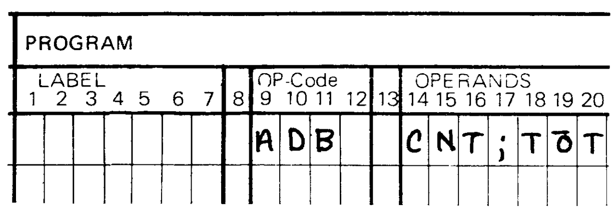


Figure 2-66. Two-Address Add Binary (ADB) Instruction Coding Example

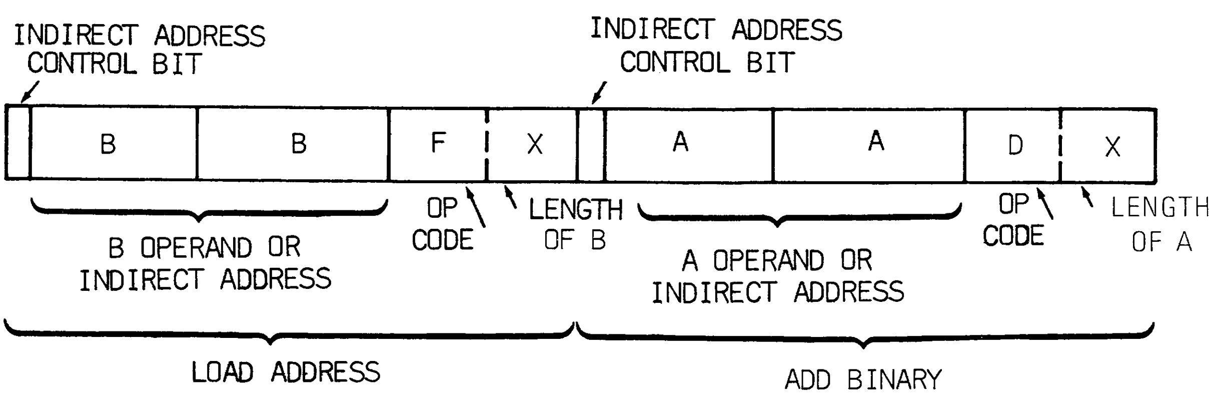


Figure 2-65
Two-Address Add Binary (ADB) Instruction Machine Language Format

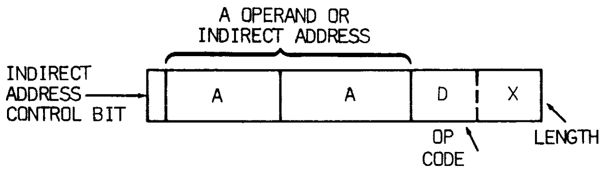


Figure 2-67. Single-Address Add Binary (ADB) Instruction Machine Language Format

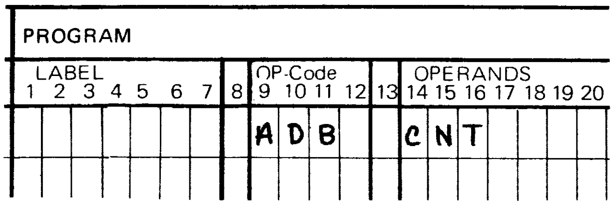


Figure 2-68. Single-Address Add Binary (ADB) Instruction Coding Example

2-93. SUBTRACT BINARY - SBB (OP CODE E)

2-94. Subtract Binary is a single-address or two-address instruction that is similar to the Subtract Decimal instruction. In the two-address Subtract Binary instruction the data stored in the A operand is complemented and added to the data stored in the B operand. The result of the operation is stored in the B operand. Both operands in the two-address instruction require length indicators if the length differs from the implied length of the operand and the maximum length of each is 16 characters. Figures 2-69 and 2-70 show the machine language format and a coding example of the two-address Subtract Binary instruction. In the single-address instruction the data stored in the A operand is complemented and added to the implied B operand, the accumulator. The result of the operation is stored in the accumulator. The length of the implied B operand is assumed to be length of the accumulator, 16 characters. If the A operand is less than the length of the accumulator, hexadecimal zeroes are supplied in the high order positions of the A operand. Figures 2-71 and 2-72 show the machine language format and a coding example of the single-address Subtract Binary instruction.

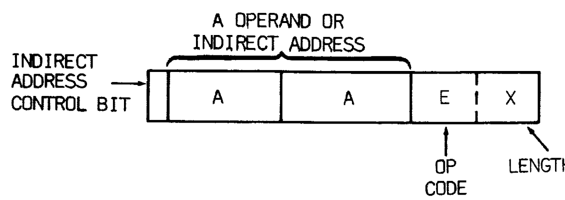


Figure 2-71. Single-Address Subtract Binary (SBR) Instruction Machine Language Format

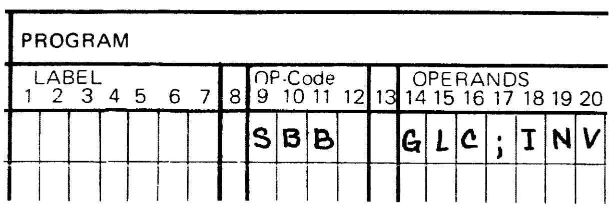


Figure 2-70. Two-Address Subtract Binary (SBB) Instruction Coding Format

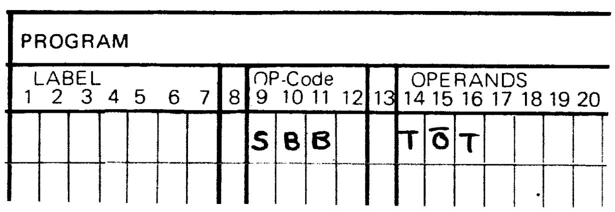


Figure 2-72. Single-Address Subtract Binary (SBB) Instruction coding Example

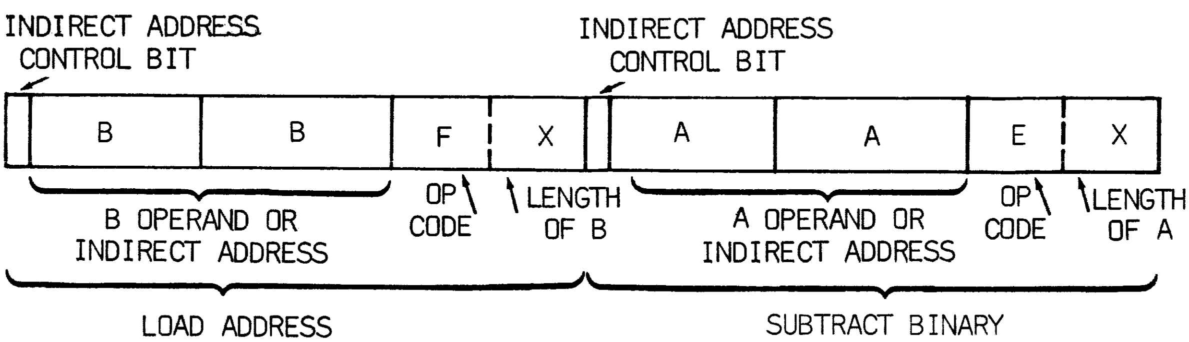


Figure 2-69.
Two-Address Subtract Binary (SBB) Instruction Machine Language Format

2-95. AND - AND (OP CODE 9, VARIANT Ø)

2-96. And is a single-address or two-address instruction in which the bits of the A operand are logically ANDed, bit by bit, with the corresponding bits of the B operand. In the two-address format, both operand fields have the implied length of one (1) position (byte), and if the instruction is in the single-address format, the least significant position (highest memory location – byte 15) of the accumulator becomes the B operand. Figures 2-73 and 2-74 show the machine language format and a coding example of the two-address AND instruction, and figures 2-75 and 2-76 show the machine language format and a coding example of the single-address instruction.

2-97. During the AND operation, the bits of the A and B operands are combined according to the rules shown in table 2-3, with the result of the operation going to the B operand. The A operand remains unaltered. The result of the operation can be checked with the Branch On Non-Zero instruction, and the branch will be taken if a bit match occurs.

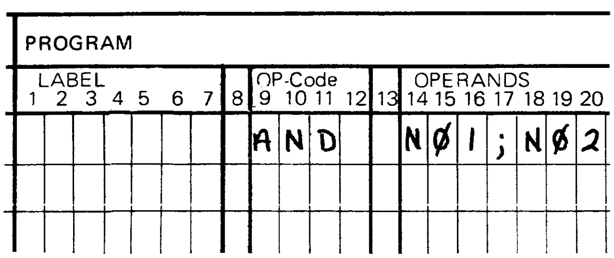


Figure 2-74. Two-Address AND Instruction Coding Example

TABLE 2-3. RULES FOR AND OPERATION

BIT IN A OPERAND	BIT IN B OPERAND	BIT IN RESULT
0	0	0
1	0	0
0	1	0
1	1	1

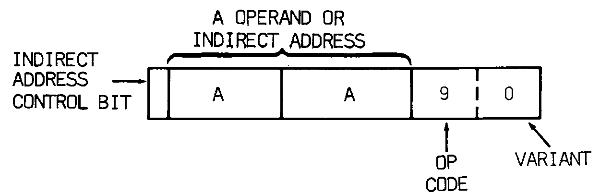


Figure 2-75. Single-Address AND Instruction Machine Language Format

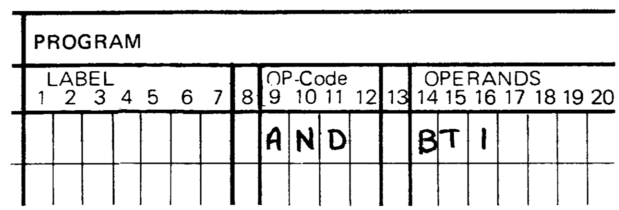


Figure 2-76. Single-Address AND Instruction Coding Example

2-98. OR - OR (OP CODE 9, VARIANT 1)

2-99. The OR instruction is similar to the AND instruction, except that different variants are used and different rules are applied during the operation. The rules used during the OR operation are shown in table

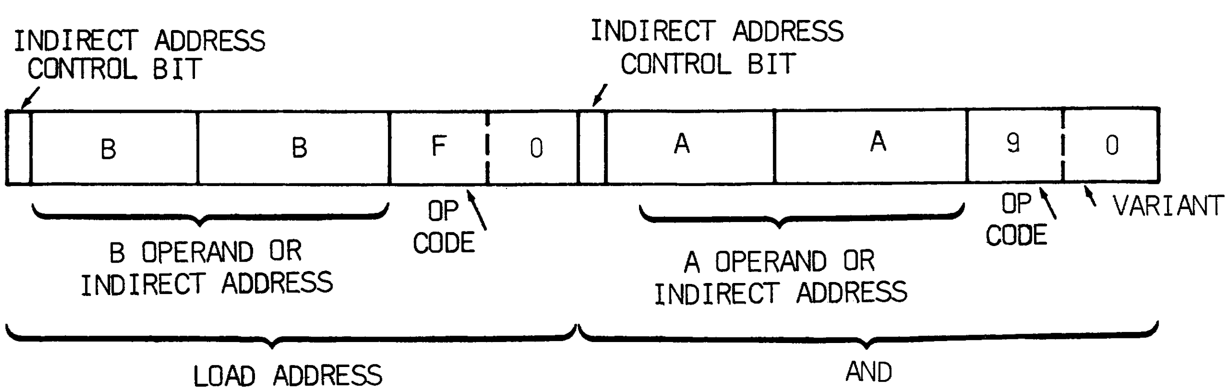


Figure 2-73. Two-Address AND Instruction Machine Language Format

2-4. Figures 2-77 and 2-78 show the machine language format and a coding example of the two-address OR instruction, and figures 2-79 and 2-80 show the machine language format and a coding example of the single-address instruction.

TABLE 2-4. RULES FOR OR OPERATION

BIT IN A OPERAND	BIT IN B OPERAND	BIT IN RESULT
0	0	0
1	0	0
0	1	1
1	1	1

2-100. EXCLUSIVE OR - XOR (OP CODE 9, VARIANT 2)

2-101. The Exclusive OR instruction is similar to the AND instruction, except that different variants are used and different rules are applied during the operation. The rules used during the Exclusive OR operation are shown in table 2-5. Figure 2-81 and 2-82 show the machine language format and a coding example of the two-address XOR instruction, and figures 2-83 and 2-84 show the machine language format and a coding example of the single-address instruction.

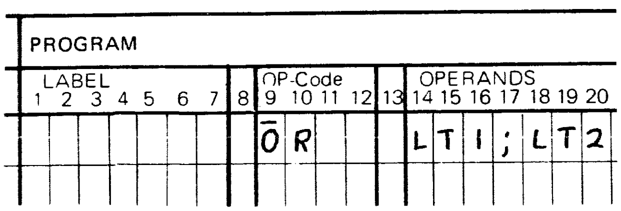


Figure 2-78. Two-Address OR Instruction Coding Example

TABLE 2-5. RULES FOR EXCLUSIVE OR (XOR) OPERATION

BIT IN A OPERAND	BIT IN B OPERAND	BIT IN RESULT
0	0	0
1	0	1
0	1	1
1	1	0

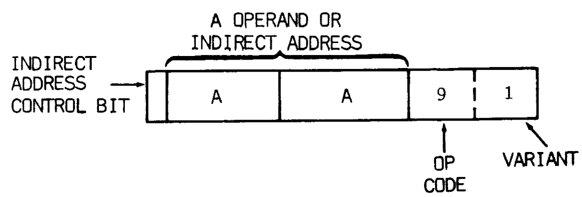


Figure 2-79. Single-Address OR Instruction Machine Language Format

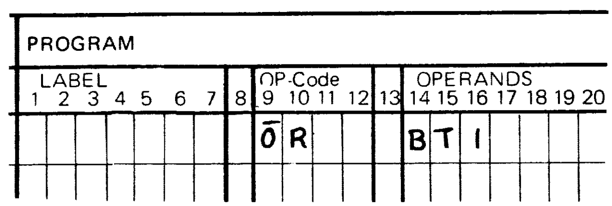


Figure 2-80. Single-Address OR Instruction Coding Example

2-102. **Decision and Control Instructions**

2-103. BRANCH ON OVERFLOW - BOV (OP CODE A, VARIANT 1)

2-104. Branch On Overflow is a single-address instruction used to test the result of the previous arithmetic

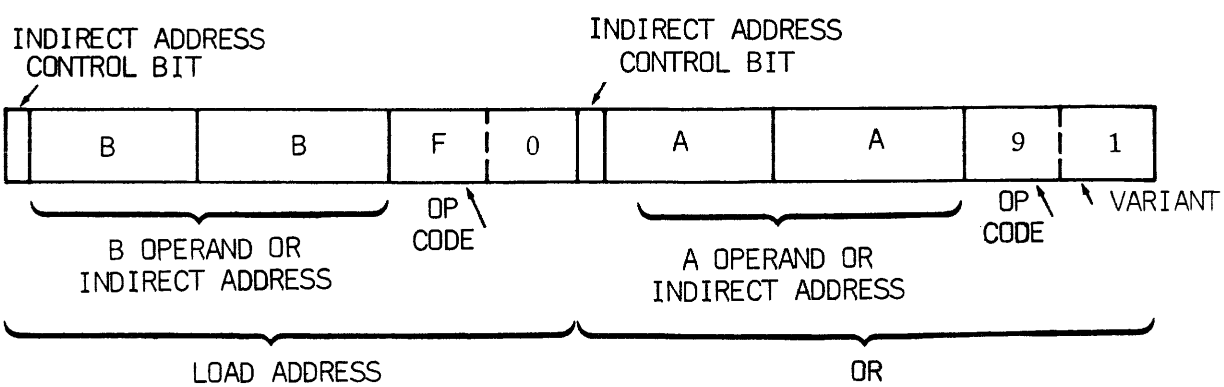


Figure 2-77. Two-Address OR Instruction Machine Language Format

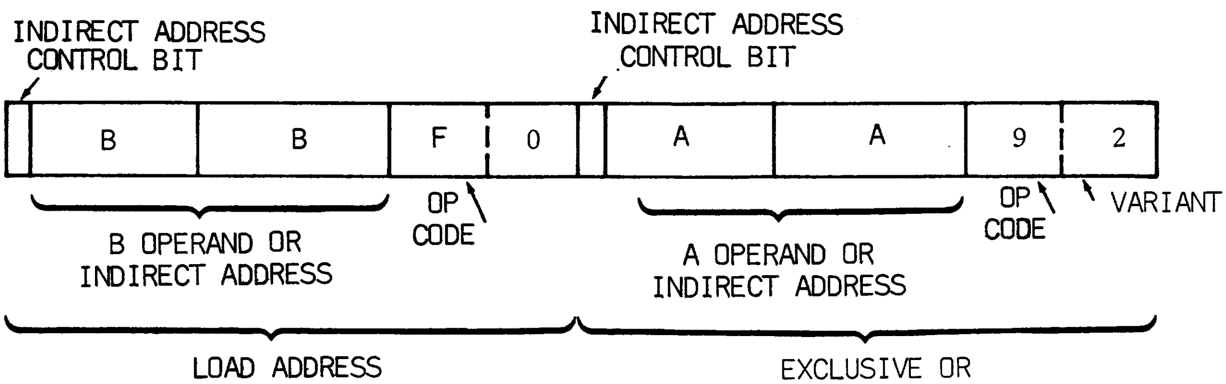


Figure 2-81.
Two-Address Exclusive OR (XOR) Instruction Machine Language Format

PROGRAM

LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–20)
		XŌR		VA1;VA2

Figure 2-82. Two-Address Exclusive OR (XOR) Instruction Coding Example

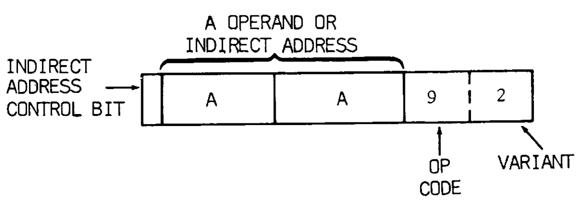


Figure 2-83. Single-Address Exclusive OR (XOR) Instruction Machine Language Format

PROGRAM

LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–20)
		XŌR		VA1

Figure 2-84. Single-Address Exclusive OR (XOR) Instruction Coding Example

operation for an overflow (carry) condition. An overflow condition is indicated within the processor by the setting of switch one as described in paragraph 2-33. When switch one is set to the "1" state by a carry in the result of the previous operation, the current program address is replaced by the branch address given in the Branch On Overflow instruction. Figure 2-85 shows the machine language format of the Branch On Overflow instruction and figure 2-86 is a coding example.

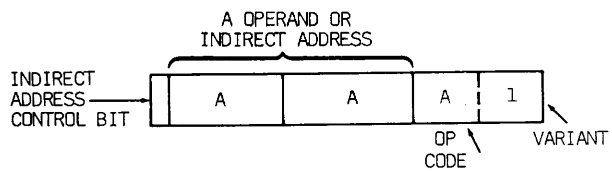


Figure 2-85. Branch On Overflow (BOV) Instruction Machine Language Format

PROGRAM

LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–20)
		ADD		AMT;QTY
		BŌV		ERM
		SBD		TAX;AMT

Figure 2-86. Branch On Overflow (BOV) Instruction Coding Example

2-105. BRANCH ON MINUS - BMI OR BGT (OP CODE A, VARIANT 2)

2-106. Branch On Minus (Branch Greater Than) is a single-address instruction used to test the result of the previous arithmetic operation for a negative result. Negative results are indicated within the processor by the setting of switch two as described in paragraph 2-35. When switch two is set to the "1" state, the current program address is replaced by the branch address given in the Branch On Minus instruction. Figure 2-87 shows the machine language format of the Branch On Minus instruction and figure 2-88 is a coding example.

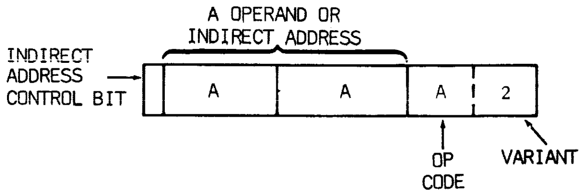


Figure 2-87. Branch On Minus (BMI) Instruction Machine Language Format

PROGRAM

LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–20)
		MPY		AA;AB
		BMI		TYP
		SBD		AC;AB

Figure 2-88. Branch On Minus (BMI) Instruction Coding Example

2-107. BRANCH ON NON-ZERO - BNZ OR BNE (OP CODE A, VARIANT 3)

2-108. Branch On Non-Zero (Branch Not Equal) is a single-address instruction used to test the result of the previous arithmetic, logical, or editing instruction for a result of something other than zero. A non-zero result is indicated within the processor by setting switch three to the "1" state as described in paragraph 2-37. When switch three is set, the current program address is replaced by the branch address given in the Branch On Non-Zero instruction. Figure 2-89 shows the machine language format of the Branch On Non-Zero instruction and figure 2-90 is a coding example.

2-109. BRANCH EQUAL - BEQ OR BZ (OP CODE A, VARIANT 4)

2-110. Branch Equal (Branch On Zero) is a single-address instruction used to test the result of the previous operation for an equal (zero) result. As described in paragraph 2-39, the result of the compare operation when the operands are equal produces a zero result. The Branch Equal instruction checks the setting of switch three for the "0" state (indicating a zero result), and when true, replaces the current program address with the branch address given in the Branch Equal instruction.

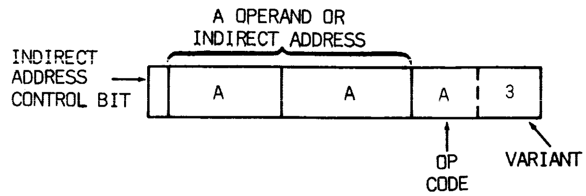


Figure 2-89. Branch On Non-Zero (BNZ) Instruction Machine Language Format

PROGRAM

LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–20)
		SBD		CNT;TŌT
		BNZ		RPT
		ADD		ŌNE;CTR

Figure 2-90. Branch On Non-Zero (BNZ) Instruction Coding Example

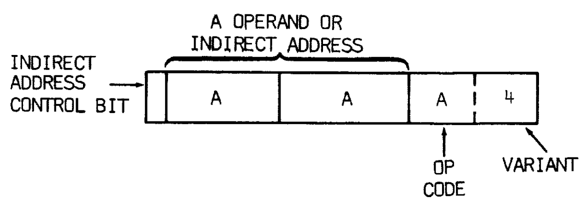


Figure 2-91. Branch Equal (BEQ) Instruction Machine Language Format

PROGRAM

LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–20)
		CD		A1;A2
		BEQ		STR
		ADD		ŌNE;A2

Figure 2-92. Branch Equal (BEQ) Instruction Coding Example

Figure 2-91 shows the machine language format of the Branch Equal instruction and figure 2-92 is a coding example.

2-111. BRANCH NOT MINUS - BNM OR BLE (OP CODE A, VARIANT 5)

2-112. Branch Not Minus (Branch Less Than or Equal) is a single-address instruction used to test the result of the previous arithmetic/compare or edit operation for a positive (not minus) condition. As described in paragraph 2-35, the not minus condition is indicated within the processor by the checking of switch two, the minus switch, for a "0" state, indicating a not minus or positive result. When the switch two is in the "0" state, the current address is replaced by the branch address given in the Branch Not Minus instruction. Figure 2-93 shows the machine language format of the Branch Not Minus instruction and figure 2-94 shows a coding example.

NOTE

This instruction is not presently contained in the QANTEL/ANSWER Processor System and will be implemented at a later date.

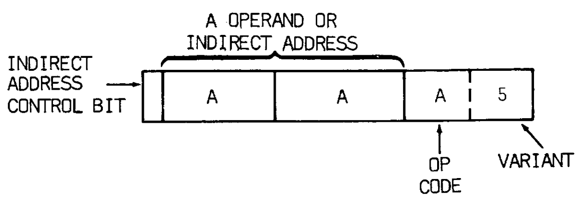


Figure 2-93. Branch Not Minus (BNM) Instruction Machine Language Format

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		SBD		TO1;TO2
		BNM		CNT
		ADD		FNT;TO2

Figure 2-94. Branch Not Minus (BNM) Instruction Coding Example

2-113. UNCONDITIONAL BRANCH - BRU (OP CODE A, VARIANT 7)

2-114. Unconditional Branch is a single-address instruction used to replace the current program address with the branch address given in the instruction. No internal processor conditions are checked to complete this operation. Figure 2-95 shows the machine language format of the Unconditional Branch (BRU) instruction and figure 2-96 is a coding example.

2-115. HALT AND BRANCH - HLT (OP CODE A, VARIANT 8)

2-116. Halt and Branch is a single-address instruction which stops the processor and program immediately upon execution. Pressing the START button on the processor control panel (power supply panel or Programmer's Control Console) causes the program to branch to the address given in the Halt and Branch instruction. Figure 2-97 shows the machine language format for the Halt and Branch instruction and figure 2-98 is a coding example.

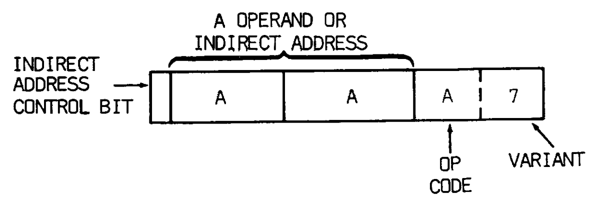


Figure 2-95. Unconditional Branch (BRU) Instruction Machine Language Format

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		CD		A1;A2
		BNZ		RT
		BRU		STR

Figure 2-96. Unconditional Branch (BRU) Instruction Coding Example

2-117. BRANCH AND LINK - BLI (OP CODE A, VARIANT 9)

2-118. Branch and Link is a single-address instruction which stores the current program address and unconditionally branches. Figure 2-99 shows the machine language format and figure 2-100 is a coding example. The link address (current program address) is stored beginning at the address given in the Branch and Link instruction. Subsequently, one byte is skipped and the

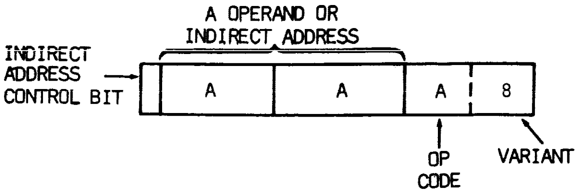


Figure 2-97. Halt and Branch (HLT) Instruction Machine Language Format

PROGRAM

LABEL (1 2 3 4 5 6 7)	8	OP-Code (9 10 11 12)	13	OPERANDS (14 15 16 17 18 19 20)
		HLT		WRT

Figure 2-98. Halt and Branch (HLT) Instruction Coding Example

next instruction is taken from the next memory position. For example, if the address in the instruction is 1000, the link address is stored in 1000 and 1001, position 1002 is skipped, and the next instruction is taken from position 1003. Position 1002 is left for the programmer supplied Unconditional Branch operation code (hexadecimal A7). For example:

0100 1000A9 Branch and Link
0130 Next instruction
Before BLI After BLI
1000 XXXXA7 0130A7

NOTE

A7 is the operation code for Unconditional Branch. Therefore, to return to the original "next" address, the programmer unconditionally branches back to location 1000. This in turn, is an unconditional branch back to the "next" instruction at location 0103.

2-119. TEST BIT - TBT (OP CODE 9, VARIANT 3)

2-120. Test Bit is a single-address or two-address instruction. In the two-address instruction, the bits of the A operand are checked against the corresponding bits of the B operand. The bits are checked to determine if any corresponding two bits in the two operands are both in the "1" state. In this instruction, neither operand is altered and both operands have an implied length of one (1) position (byte). Figure 2-101 shows the machine language format of the two-address instruction and figure 2-102 is a coding example. If the Test Bit instruction is a single-address instruction, the least significant position (highest memory location – byte 15) of the accumulator becomes the B operand. Figure 2-103 shows the single-address instruction machine language format and figure 2-104 is a coding example.

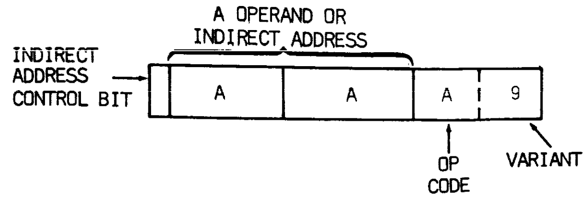


Figure 2-99. Branch and Link (BLI) Instruction Machine Language Format

PROGRAM

LABEL (1 2 3 4 5 6 7)	8	OP-Code (9 10 11 12)	13	OPERANDS (14 15 16 17 18 19 20)
		BLI		WRT
RET		SBD		AØ;A1

Figure 2-100. Branch and Link (BLI) Instruction Coding Example

2-121. If none of the corresponding bits are both in the "1" state, the result of the operation is zero (can be checked using the Branch Equal instruction). Conversely, if any of the corresponding bits are both in the "1" state, the result of the operation is something other than zero (non-zero), and can be checked by using the Branch On Non-Zero instruction.

2-122. COMPARE DECIMAL - CD (OP CODE 9, VARIANT 6)

2-123. Compare Decimal is either a single-address or two-address instruction used to compare the signed decimal values of the A and B operands, without altering either operand. The results of the Compare Decimal operation are indicated by internal switch settings that can be examined by the Branch On Minus, Branch On Non-Zero, and Branch Equal instructions. Because the compare instruction is actually a subtraction of the A operand from the B operand, a negative result indicates that the B operand is smaller than the A operand. This type of result can be checked with the Branch On Minus

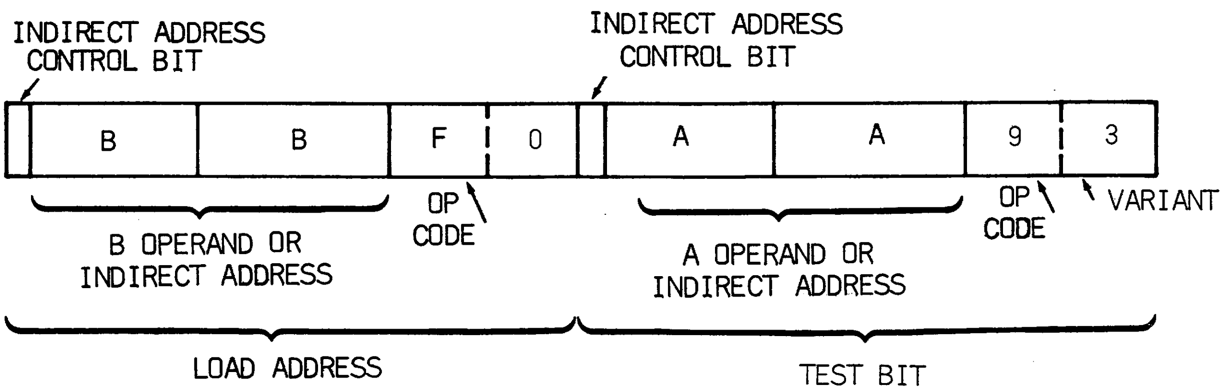


Figure 2-101.
Two-Address Test Bit (TBT) Instruction Machine Language Format

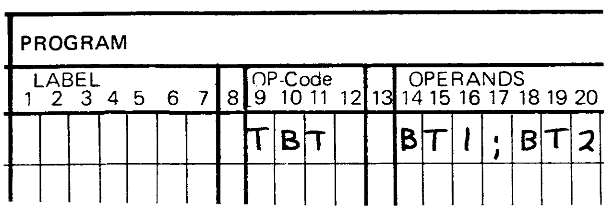


Figure 2-102. Two-Address Test Bit (TBT) Instruction Coding Example

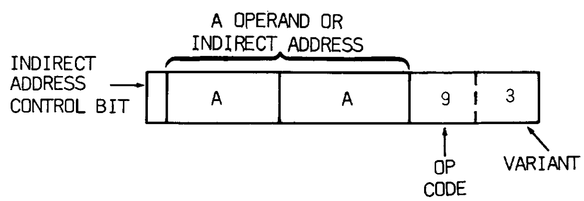


Figure 2-103. Single-Address Test Bit (TBT) Instruction Machine Language Format

instruction. If the compared operands are equal, the result of the internal subtraction would be zero, and can be checked with the Branch Equal instruction.

2-124. In the Compare Decimal instruction, both operand fields must be of equal length or the overflow switch (switch 1) will be set. Figure 2-105 shows the two-address machine language format and figure 2-106 is a coding example. In the single-address format, the implied B operand field is the 16 accumulator positions,

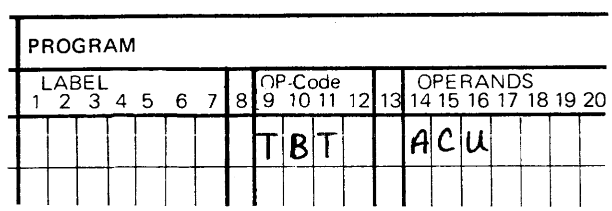


Figure 2-104. Single-Address Test Bit (TBT) Instruction Coding Example

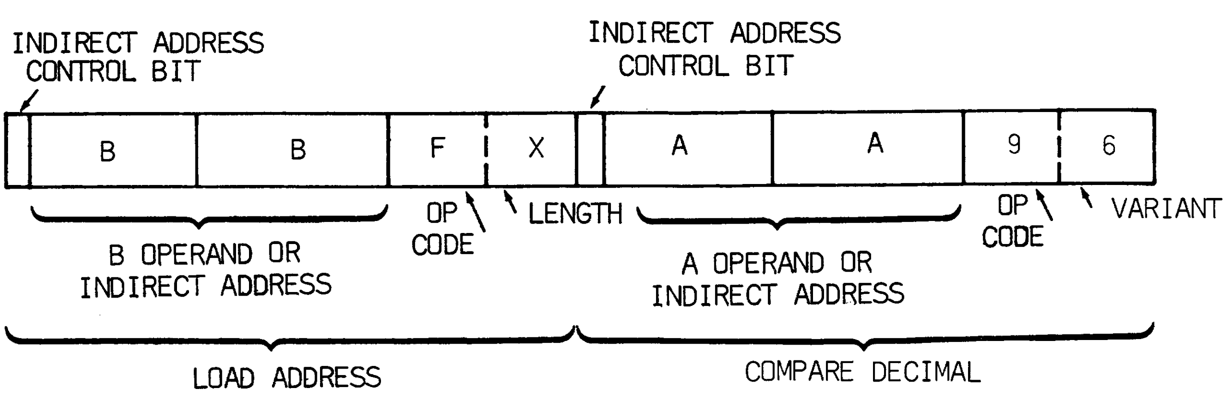


Figure 2-105.
Two-Address Compare Decimal (CD) Instruction Machine Language Format

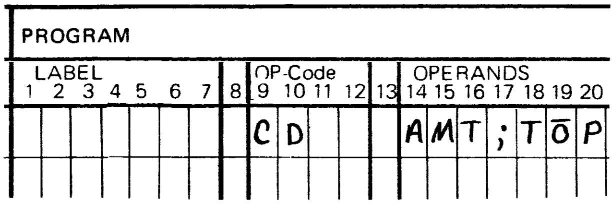


Figure 2-106. Two-Address Compare Decimal (CD) Instruction Coding Example

and the A operand field is 16 memory positions beginning at the memory address (least significant byte) given in the instructions. Figure 2-107 shows the machine language format of the single-address instruction and figure 2-108 is a coding example.

2-125. COMPARE LOGICAL - CMP (OP CODE 7)

2-126. Compare Logical is a single-address or two-address instruction. In the two-address instruction the A and B operands are compared in a bit-by-bit fashion. Figure 2-109 shows the machine language format of the two-address instruction and figure 2-110 is a coding example. In the single-address format, the B operand is the 16 accumulator positions. The length given in the single-address instruction is the length of the A operand. If this length is less than 16 (length of the accumulator),

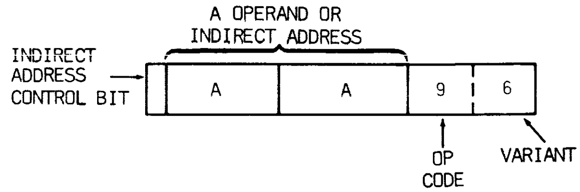


Figure 2-107. Single-Address Compare Decimal (CD) Instruction Machine Language Format

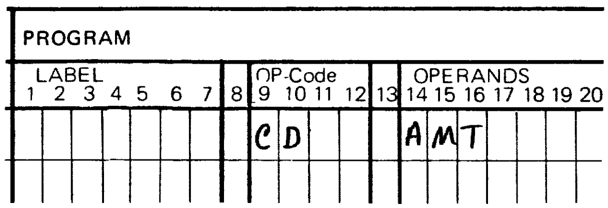


Figure 2-108. Single-Address Compare Decimal (CD) Instruction Coding Example

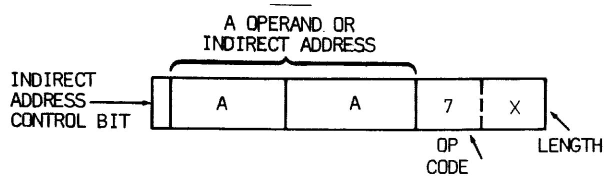


Figure 2-111. Single-Address Compare Logical (CMP) Instruction Machine Language Format

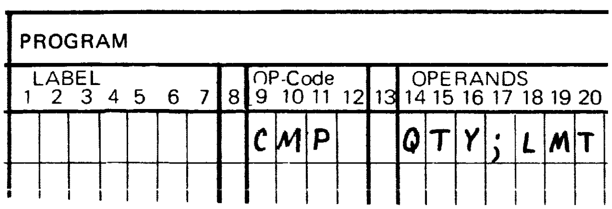


Figure 2-110. Two-Address Compare Logical (CMP) Instruction Coding Example

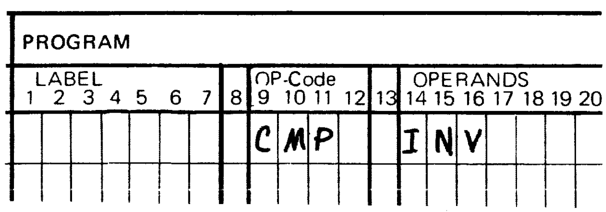


Figure 2-112. Single-Address Compare Logical (CMP) Instruction Coding Example

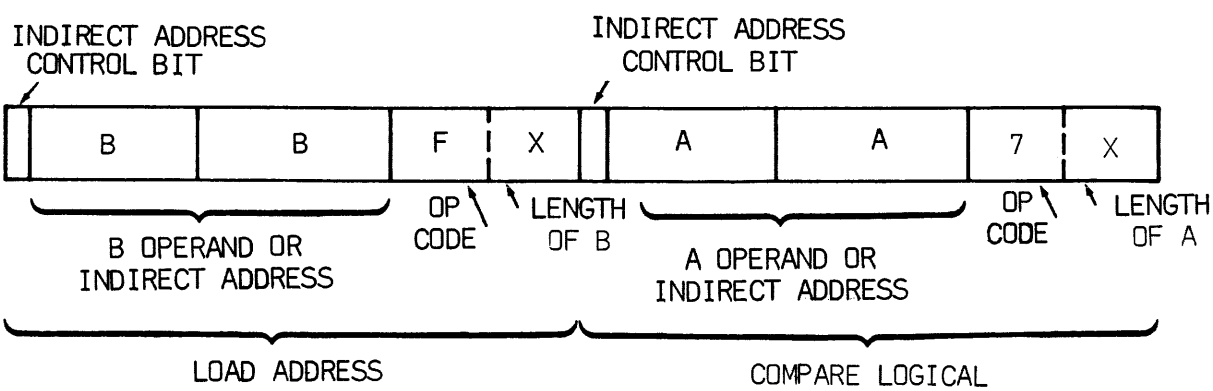


Figure 2-109.
Two-Address Compare Logical (CMP) Instruction Machine Language Format

leading logical zeroes are used in the operation to equalize the length of the operands and facilitate the compare. Leading zeroes are also supplied in the two-address instruction under the same circumstances. Figure 2-111 shows the machine language format of the single-address instruction and figure 2-112 is a coding example. The results of the Compare Logical operation can be checked by means of the branch instructions in a manner similar to that described for the preceding Compare Decimal instruction.

2-127. RETURN FROM INTERRUPT – RTI (OP CODE 9, VARIANT F)

2-128. Return From Interrupt is a single-address instruction at the end of the interrupt routine to bring control back to the regular program address next in sequence before the interrupt occurred. During the Return From Interrupt operation, settings of switches one, two, and three are restored to their former settings, the interrupt inhibit bit is reset, and the program is returned to the address stored in memory positions 16_{10} and 17_{10}. (A detailed description of the Interrupt Feature and its requirements is presented in paragraph 2-25). An example of a typical Return From Interrupt instruction in machine language format is shown in figure 2-113 and figure 2-114 is a coding example.

NOTE

Because all QANTEL/ANSWER Processor System single-address instructions are three bytes in length, the Return From Interrupt instruction is also three bytes in length. However, only the operation code and variant (third byte) are recognized during the operation. As a result, any type of data may be used to fill the first two bytes as long as the Indirect Address control bit is 0.

2-129. NO OPERATION - NOP (OP CODE A, VARIANT 0)

2-130. No Operation is always a single-address instruction which performs no immediate function in the program. Figure 2-115 shows the machine language format and figure 2-116 is a coding example.

2-131. PROGRAMMING NOTE

2-132. The No Operation instruction may be modified to the form of a Branch instruction by subsequent program steps (such as Add Binary, Move Numeric, etc.).

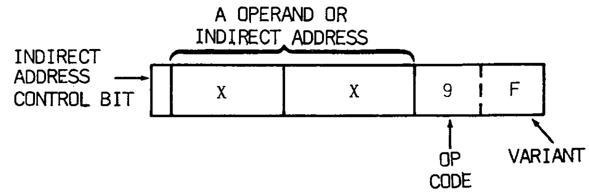


Figure 2-113. Return From Interrupt (RTI) Instruction Machine Language Format

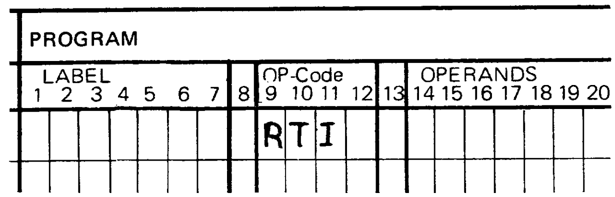


Figure 2-114. Return From Interrupt (RTI) Instruction Coding Example

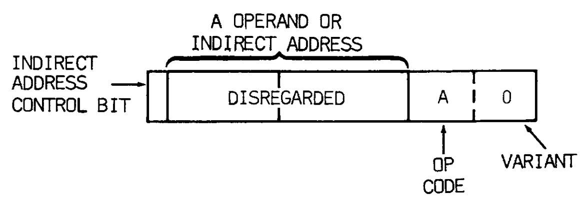


Figure 2-115. No Operation (NOP) Instruction Machine Language Format

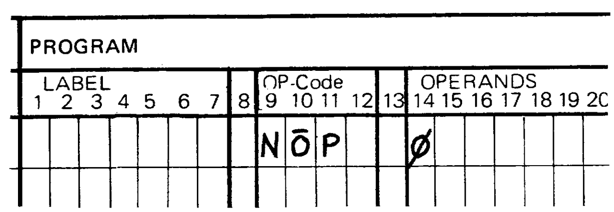


Figure 2-116. No Operation (NOP) Instruction Coding Example

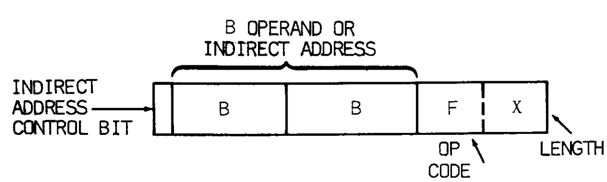


Figure 2-117. Load Address (LDA) Instruction Machine Language Format

2-133. Special Instructions

2-134. LOAD ADDRESS - LDA (OP CODE F)

2-135. The Load Address instruction is contained in the first three bytes of all two-address instructions. When a Load Address is identified, the fetch cycle continues loading the appropriate registers with the next operand address. Early in the fetch cycle, the standard address for the accumulator is created for the B operand address, and is changed only if a Load Address is encountered. By this method, any instruction that is not a Load Address has a B operand address which is the address of the accumulator. The Load Address machine language format is shown in figure 2-117. The Q/BAL assembler program will automatically generate all Load Address instructions required by a program being assembled.

2-136. MICRO-INSTRUCTION MODE - MIM (OP CODE A, VARIANT B)

2-137. The QANTEL/ANSWER Processor System has as a standard feature, the ability to take its control instructions (micro-instructions) from the main memory. The instruction used to enable this feature is called Micro-Instruction Mode, and requires a complete understanding of the QANTEL Micro-Assembler Manual. When this feature is utilized, positions 2048 through 4095 of main memory may be made to act as the control memory. This method may be used in the QANTEL/ANSWER Processor System to control the various communications data sets. The micro-instruction mode is left, and the standard-instruction mode regained, whenever the control instruction from the main memory is a return to the fetch cycle. Figure 2-118 shows the Micro-Instruction Mode machine language format.

2-138. The micro-instruction mode can be used to perform instructions not included in the standard instruction set. These could be arithmetic to a different base or packed decimal arithmetic common on some large systems. QANTEL Corporation has several types of micro routines available for customer use. However, if specific situations require use of the Micro-Instruction Mode instructions, users should contact the QANTEL Programming and Systems Department.

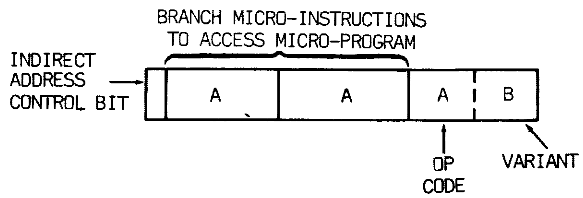
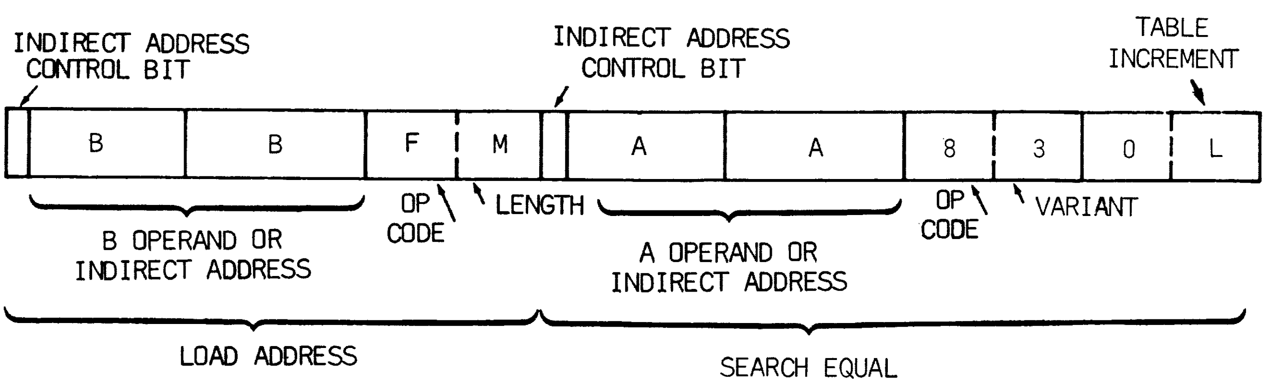


Figure 2-118. Micro-Instruction Mode (MIM) Instruction Machine Language Format



B Operand = ADDRESS OF ARGUMENT (LOW ORDER)
M = LENGTH OF ARGUMENT
A Operand = ADDRESS OF TABLE (HIGH ORDER OF FIRST ENTRY)
L = TABLE INCREMENT (SEARCH PROCEEDS FROM LEFT TO RIGHT UNTIL A MATCH IS FOUND, ADDRESS OF MATCH , LOW ORDER, IS STORED IN 24_{10} AND 25_{10}.)

Figure 2-119. Search Equal (SEQ) Instruction Machine Language Format

2-139. SEARCH EQUAL - SEQ (OP CODE 8, VARIANT 3)

2-140. The Search Equal instruction is a seven-byte instruction that allows the user to search through a table, located at some known position in memory (the A operand is the high order of first entry), in increments of L until a match is found for the argument (the B operand is the argument address and M = argument length). Then the address of the match is stored in positions 24_{10} and 25_{10} in reserved memory. See table 2-2 for a description of Reserved Memory locations. If the increment of search (L) is not 1, then the argument must be property aligned. The maximum length for the table increment is 16_{10} (0016). The programmer, to avoid looping memory if the match is not found, should move (MOV) the argument to the memory address immediately following the last position of the table. This will cause a match to occur, stopping the search operation to prevent looping memory. If a match occurs during the search operation, the address of the match location can be compared against the address of the data stored at the end of the table to determine that the match has occured within the table and not to the data stored at the end of the table. Figure 2-119 shows the machine language format of the Search Equal instruction and figure 2-120 shows an example of Search Equal instruction coding.

PROGRAM I.D.	LABEL (1–7)	OP-Code (9–12)	OPERANDS (14–33)		COMMENTS (35–)
	ARG	DA	4	ARGUMENT	
		~			
		SEQ	TAB;ARG,4		
		DC	$ØB	STEP SIZE	11
		~			
	TAB	EQU	*		
		DC	ØØØ17938764		
		DC	ØØØ22134321		
		DC	ØØØ34159654		
		~			

Figure 2-120. Search Equal (SEQ) Instruction Coding Example

SECTION III

ASSEMBLER INSTRUCTION CODING

3-1. GENERAL

3-2. When the QANTEL Business Assembly Language (Q/BAL) is used to write programs for the QANTEL/ANSWER Processor System, the instructions are coded in a free format assembly language. In order to write programs properly in this free format assembly language the programmer must strictly adhere to the rules set forth in this section. Figure 3-1 shows the coding form used when writing programs for the QANTEL Business Assembly Language. The following paragraphs describe the various portions of the statements used to create the machine executable instructions that make up the object program. Refer to section II of this manual for a detailed description of the operation of those machine executable instructions. Assembler directives used to control the operation of the assembler and the loader, and to generate constants and data areas, are discussed in detail in section IV of this manual.

3-3. LABELS

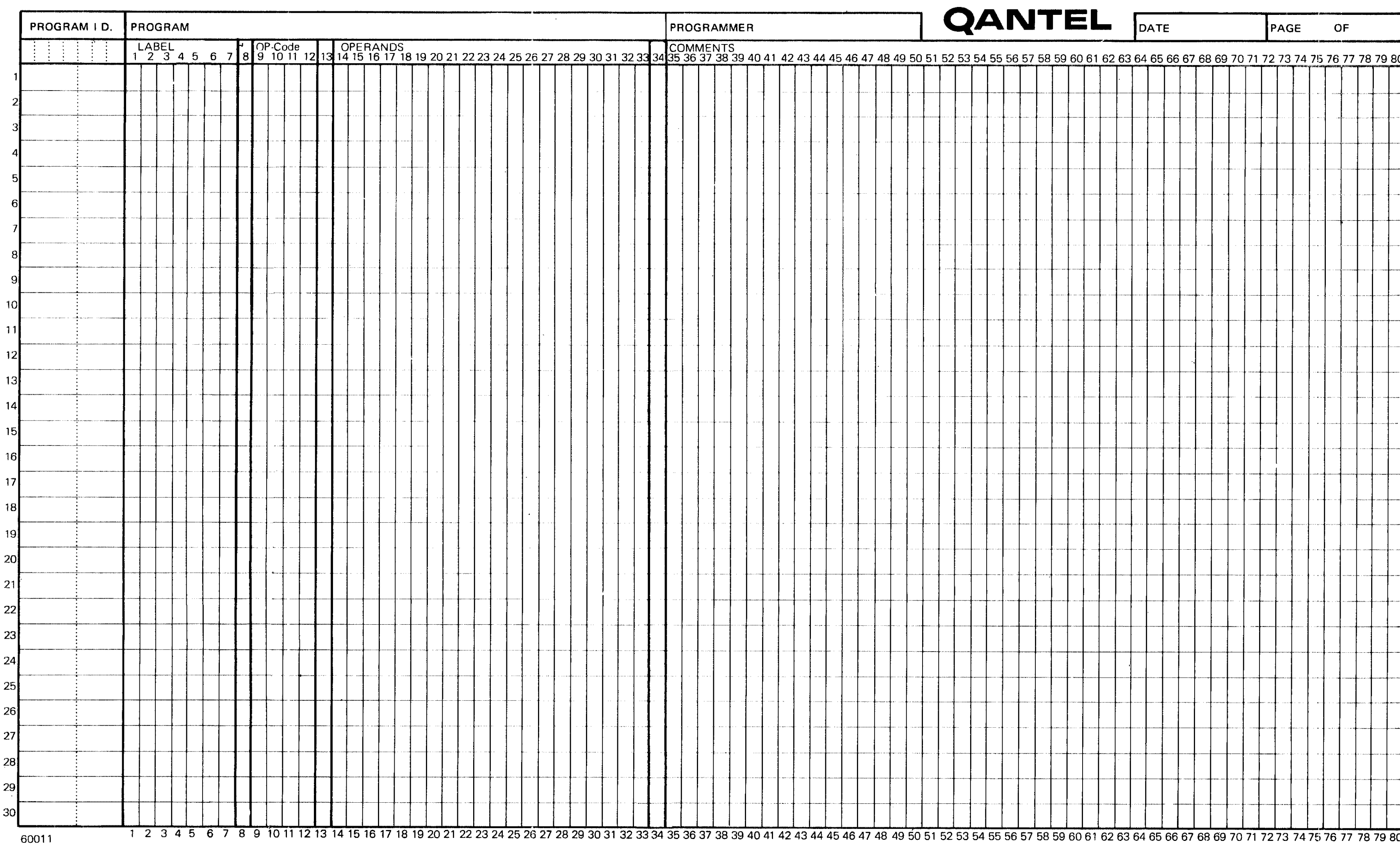
3-4. Labels are programmer-created names that are meaningful to the programmer and, in a restricted sense, to the assembler. When the object file output of the assembler is loaded, the instructions in the object file bear no particular relationship to the specific labels that were chosen by the programmer when writing his original program. To illustrate this (in a program that contains no naming conflicts), the characters that are used for any given labels may be changed to a net set of characters. Providing that no new naming conflicts are created, the resultant machine language program will be identical to a program created using the original labels. The following paragraphs describe the rules which must be followed creating labels for programs written in the QANTEL Business Assembly Language.

3-5. LABEL FORM

3-6. Labels must start with letters A through Z and can contain up to five characters. Characters, other than the first character, must be letters (A to Z) or digits (Ø to 9). Only the first three characters are stored in the assembler's internal tag table. Consequently, each label created by the programmer must be uniquely defined by the first three characters. The additional character positions are provided for programmer convenience. A maximum of 35Ø labels, or tags, may be defined in any one program. Figure 3-2 shows samples of the correct way to code labels, and figure 3-3 shows some incorrect ways to write labels.

	PROGRAM I.D.	PROGRAM												
		LABEL								OP-Code				
		1	2	3	4	5	6	7	8	9	10	11	12	13
1		B	U	F	F	R								
2		T	I	Ø										
3		R	2											
4		U	Ō	2										
5		N	A	M										
6		N	A	M	E									
7		X	1	2	3	4								
8														

Figure 3-2. Correct Labels Coding Example



PROGRAM I D. | PROGRAM | PROGRAMMER | QANTEL | DATE | PAGE OF

LABEL 1 2 3 4 5 6 7 | 8 | OP-Code 9 10 11 12 | 13 | OPERANDS 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 | COMMENTS 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

60011

Figure 3-1. QANTEL Business Assembler Language Coding Form

```
2A          FIRST CHARACTER NUMERIC
BUFFER      EXCEEDS LENGTH REQUIREMENTS
25Ø         FIRST CHARACTER NUMERIC
,4          SPECIAL CHARACTERS NOT ALLOWED
$2A         SPECIAL CHARACTERS NOT ALLOWED
PROGRAM     EXCEEDS LENGTH REQUIREMENTS
```

Figure 3-3. Incorrect Labels Coding Example

NOTE

If the labels NAM and NAME were to appear in the same program they would be treated as the same label. If both of these labels appeared in the same program a naming conflict could exist, or a potential point of confusion could exist for persons reading the listing of the program.

3-7. LABEL VALUE

3-8. Each distinct label in a program, when defined, will possess a value. A label becomes defined when it appears in the label field of some coding statement. The value of the label is stored within the assembler's tag table. These values are usually memory addresses that are significant at program execution time. They may also be I/O device assignments, table lengths, or any other value significant to the programmer. Label values are stored in the assembler tag table in two eight-bit bytes, and are consequently restricted in value to positive integers between 0 and 32767 (hexadecimal 7FFF).

3-9. LABEL LENGTH

3-10. A label obtains a length at the same time that it obtains a value. This length has nothing to do with the nature of the label, but is a property dependent upon the statement which the label identifies. The lengths are stored in the assembler's tag table in a single, eight-bit byte, and are restricted in value to positive integers between 0 and 255. Due to the way these lengths are used in instructions and the manner in which the computer treats length fields, the zero length represents a length of 256. Lengths are generally byte counts for all variable length operations other than I/O instructions. These lengths are used by the assembler in the automatic assignment of lengths to these operations. The lengths that are assigned to labels are discussed, when appropriate, in following paragraphs.

3-11. PROGRAMMER ACCESS TO SYMBOL LENGTHS (.)

3-12. The "length of" operator, period (.), allows the programmer access to the lengths stored in the assembler

```
ER1     DC      'DISC PARITY CHECK'
          ⋮
        WRC     ER1-.ER1+1,Ø;.ER1
```

Figure 3-4. "Length Of" Operator (.) Coding Example tag table. Each label has, in addition to its assigned value, an assigned or implied length. (This length is distinct from the number of characters in the label). This length is stored within the assembler program and may be accessed by the "length of" operator, period (.). The period must be followed by a label with a value and length that has been previously assigned. Figure 3-4 shows a coding example of the use of the "length of" operator. In this example, a Write and Count (WRC) from the Defined Constant (DC) labeled ER1 is done to device 0 (usually the typewriter) for a length of ER1. One of the conveniences the use of the "length of" operation (.) will allow is changing the Defined Constant (DC), in this case an error message, without having to change other portions of the program.

3-13. DECIMAL VALUES

3-14. Decimal Values appear in the input line as strings of decimal digits (0 through 9). These values are distinguished from labels containing digits, e.g., X1234, by the fact they begin with a decimal digit. Decimal values may contain only decimal digits and are quite distinct from decimal constants used in program execution time calculations. Decimal constants are discussed in section IV. Decimal values are limited to integers with a maximum value of 32767. Figure 3-5 shows some samples of the correct ways to code decimal values.

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
ABC		DA		123
CNT		DAC		15
LEN		EQU		1Ø
		WAC		23,Ø;1

Figure 3-5. Correct Coding of Decimal Values

3-15. HEXADECIMAL VALUES

3-16. Hexadecimal values are distinctly different from hexadecimal constants that are used in program execution time calculations. Refer to section IV of this manual for a description of hexadecimal constants. Hexadecimal values must always begin with a dollar sign ($). These values may contain from one to four hexadecimal digits (0 to 9 and A to F). Figure 3-6 shows some examples of correct methods for writing hexadecimal values.

6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20 21 22 23 24 25
		MOV		A;$1FFF
		SBB		HØ1;$17,2
		WRC		BUF,$0;.BUF

Figure 3-6. Correct Coding of Hexadecimal Values

3-17. CURRENT LOCATION OPERAND (*)

3-18. The assembler program contains an internal register that contains the memory location of the high order (low memory address) byte of the current instruction, constant, or data area. This register is accessable to the programmer through the use of the asterisk (*) operand. The asterisk (*) operand is used as though it were a label, but this operand should never appear in the label field. Using the asterisk (*) operand in the label field in this instance would be meaningless. See paragraph 3-34.

3-19. The A operand address is coded as an asterisk (*) in the operand field. When coded in this manner, the A operand address is that of the instruction itself. See figure 3-7 for a coding example. This method of coding the A operand address is used in conjunction with address modification as a convenient means of addressing a point in the program in relation to the current instructions. For example, statement 00380 in appendix E performs a Halt and Branch instruction to the beginning of the next statement. It can be seen that the Halt and Branch instruction occupies location 1067, 1068 and 1069 in memory. Consequently, the asterisk (*) indicates location 1067 because that is the low-

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		HLT		*+3
		HLT		*

Figure 3-7. Current Location Operand (*) Coding Example

numbered memory position of the statement. By modifying this address with the +3, the statement actually causes a Halt and Branch to location 106A, which is the beginning of statement 00400. The asterisk (*) can be used with any address modification.

3-20. EXPRESSIONS

3-21. Expressions are composed of operands separated by the operators plus (+) or minus (-). The operands used in expressions may be labels, length of (.) labels, decimal values, hexadecimal values, or the current location operand (*). The value of any label appearing in the expression must be known to the assembler at the time that portion of the expression is evaluated. In current versions of the assembler some expressions or parts of expressions must be evaluated during Pass One. Labels appearing in such expressions must be defined at the time they are encountered, therefore they must have appeared as labels in previous statements. Figure 3-8 shows some coding examples of expressions.

			OP-Code					OPERANDS											
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
								A	+	B	-	C							
								-	$	E	F								
								.	B	U	F	-	1						
								*	-	6									
								T	A	B	L	E	-	.	T	A	B	L	E
								A	R	G	+	$	E	F					
								+	2										

Figure 3-8. Expressions Coding Examples

3-22. FREE FORMAT CODING

3-23. The fields present in the fixed character positions of previous QANTEL assemblers must be placed in the input lines of a QANTEL Business Assembly Language program in the same basic order. Refer to the coding form figure 3-1. Since the assembler does not require that field entries be made at fixed character positions, various field separators are used to delimit the fields. The major fields will appear justified on the listing if the coding form format is used as an aid to its reading, and the line will be present in the intermediate file exactly as it is originally entered. The following paragraphs describe the various fields used when coding programs with the QANTEL Business Assembly Language.

3-24. First Character Position

3-25. The first character in an input line has a special meaning in several cases. These cases are described in the following paragraphs.

3-26. COMMENT CHARACTER (*)

3-27. If the first character position of an input line contains an asterisk (*) the remainder of the line will be understood to be a comment. Such a line will appear in the listing, but will not cause the generation of any object code. Refer to Appendix E for an example of the use of the comment character.

3-28. PIN ADDRESS CHARACTER (@)

3-29. If the first character position of an input line contains an at sign (@), the label, if present, will be forced to assume the value of the assembler's location counter at the time the statement is processed. This feature is meaningful only when the operation code is Define Constant (DC), Define Area (DA), or Define Address Constant (DAC). Lines 00200 through 00260 of the assembler listing in Appendix E show the use of the pin address character. Without the presence of the pin address character (@), the label takes on the value of the last byte of the area or constant. This is the address normally used when performing arithmetic or logical operations and when moving data within memory. The use of the pin address character (@) will pin the value of the symbol to the first byte of the area or constant. This feature is especially useful when naming strings to be written to an output device since it eliminates the necessity of doing address arithmetic in the I/O instruction. The assembler directives DC, DA, and DAC are explained more fully in section IV of this manual. The use of the pin address character (@) with any other operation codes, or its use when not followed by a label, is meaningless.

3-30. LETTERS (A TO Z)

3-31. If the first character of an input line is a letter (A to Z), a label is assumed to be present. If the first character position contains the pin address character, the second character position should contain the first character of a label.

3-32. SPACE (BLANK)

3-33. If the first position of an input line contains a space (blank) the assembler assumes that no label is present. The first nonspace character position that is encountered in a left-to-right scan of the input must be the first character of a legal operation code or assembler directive.

3-34. DIGITS (Ø to 9) OR SPECIAL CHARACTERS

3-35. The presence of a digit as the first character of an input line when a source file is being created is assumed to be part of a source file number. See section VI, assembler operation, for instruction of what to do in this case. The use of special characters, other than the comment character (*) or pin address character (@) as explained in previous paragraphs, in a label is illegal.

3-36. **Label Field**

3-37. If a label as described in the previous paragraphs, is present at the beginning of an input line, it must satisfy all of the requirements for a legal label. These requirements have been explained in paragraphs 3-3 through 3-12 of this section. Also, the first three characters of the label must not have appeared as the first three characters of any other label already encountered by the assembler in the program being assembled.

3-38. **OP-Code Field**

3-39. The OP-Code (operation code) field of an input line must be preceded by one or more blanks. OP-Codes are two or three character names for the various QANTEL machine operation codes and the assembler directive codes. No naming conflicts will be created if the programmer creates symbols identical to any of the OP-Codes. Only legitimate OP-Codes may appear in the OP-Code field. The various operation codes and assembler directives are described in sections II and IV of this manual.

3-40. **A/B Operand Field**

3-41. The A/B operand field is, in its most general form, composed of an A operand address expression separated from an A operand length/device expression by a comma (,). The A operand length field is separated from a similar B operand by a semicolon (;). The various forms that an A/B operand field may take are described in figure 3-9.

3-42. The instruction operand field requirements are not as simple as the formal specification shown in figure 3-9 represents. There are several classes of instructions, some of which do not possess B operands and others that require B operands. Still others may have an implied B operand (the accumulator, or the first 16 bytes of memory). Length/device expressions are still more complex. Some two-operand instructions require a length for each operand and other two-operand instructions require a length for only the A operand. There are also a group of instructions that require an A operand length if the two-address form is used, but not if the one-address form is used. Another instruction may require a length/device field, but has no A operand. The requirements of all of the various QANTEL instructions and assembler directives are described in sections II, IV, and V of this manual. Also, to make the process of coding the A/B operand easier, a number of aids have been incorporated into the QANTEL Business Assembly Language to ensure that the requirements of each and every instruction are satisfied. These requirements are stored in the tables within the Q/BAL assembler program and if an instruction is entered that does not satisfy these requirements an appropriate error diagnostic will be generated. Also, the programmer is relieved from many of the potential problems associated with operation lengths by the automatic length assignment feature of the Q/BAL assembler programs.

3-43. If an instruction requiring a length assignment is entered, but the length is not assigned, a length will be automatically assigned in most cases. Each defined label possesses a length in addition to its value. When a label is referenced as an operand in an instruction requiring explicit length assignment, and that length is not provided, the assembler program will fetch the required length from the assembler symbol table. There are several important exceptions to this operation. These are: 1) since this assignment applies only to the length field, I/O instructions, which in most cases uses the B operand address to specify the operation length, are excluded from the automatic length assignment, and 2) if the address expression of the A or B operand contains other than a single label, the length, if required by the machine language instruction, must be explicit.

```
               <A/B OPERAND> : : = <A OPERAND>  [;<B OPERAND>]
                 <A OPERAND> : : = <OPERAND>
                 <B OPERAND> : : = <OPERAND>
                   <OPERAND> : : = <ADDRESS EXPRESSION> [ ,<LENGTH/DEVICE EXPRESSION>]
        <ADDRESS EXPRESSION> : : = <SYMBOL> | <SYMBOL> + <DIRECT EXPRESSION>
                                   <SYMBOL> - <DIRECT EXPRESSION> | <DIRECT EXPRESSION> |
  <LENGTH/DEVICE EXPRESSION> : : = <DIRECT EXPRESSION>
         <DIRECT EXPRESSION> : : = <TERM> | <TERM> + <DIRECT EXPRESSION>
                                   <TERM> - <DIRECT EXPRESSION>
                      <TERM> : : = <DIRECT SYMBOL> | <CURRENT LOCATION> | <HEX VALUE> |
                                   <DECIMAL VALUE> | <LENGTH OF SYMBOL>
             <DIRECT SYMBOL> : : = <LABEL>
                 <HEX VALUE> : : = $<HEX DIGIT>  (REFER TO PARAGRAPH 3-15)
                 <HEX DIGIT> : : = 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F
                     <LABEL> : : = <LETTER> {<ALPHA CHAR>}^4_0  (REFER TO PARAGRAPH 3-3)
                    <LETTER> : : = A|B|C|...|X|Y|Z
                    <NUMBER> : : = Ø|1|2|...|8|9
                <ALPHA CHAR> : : = <LETTER> <NUMBER>
                    <SYMBOL> : : = <DIRECT SYMBOL> | <TERM>
          <INDIRECT OPERAND> : : = @<ADDRESS EXPRESSION>
          <CURRENT LOCATION> : : = *  (REFER TO PARAGRAPH 3-17)
           <LENGTH OF LABEL> : : = .<LABEL> (REFER TO PARAGRAPH 3-11)
```

Figure 3-9. A/B Operand Field Format Requirements

3-44. Indirect Addressing

3-45. An indirect address may be defined as an operand containing not the data being referenced, but the address of the data being referenced. In other words, the indirect address identifies the location that contains the desired data. Indirect addressing is accomplished by placing an at sign (@) in the place preceding the address (actual or symbolic) for either the A operand or the B operand, or both. Either the first operand (A operand) or the second operand (B operand) may be indirectly addressed in any instruction. Indirect addressing can be extremely useful when indexing through a table or when address alterations occur during execution of a program. Figure 3-10 shows an example of indirect addressing. In figure 3-10, the indirect contents of LEN is to be moved to the field referenced as WRK. For additional information on indirect addressing refer to section II of this manual and to the QANTEL Business Assembly Language (Q/BAL) Programmers Training Manual.

			OP-Code					OPERANDS											
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
			M	Ō	V			@	L	E	N	,	5	;	W	R	K		

Figure 3-10. Indirect Addressing Coding Example

NOTE

Indirect addresses cannot be modified at program execution time by using relative addressing. Any alteration to an address that is indirect must be made with the Add Binary or Subtract Binary instructions.

3-46. Comment Field

3-47. A comment may be entered following a source statement providing at least one blank separates the comment from any preceding field. Line by line commenting can greatly help the maintainability of programs written in assembly language. While the assembly language statements can often show what is happening, especially if the labels involved have sufficient mnemonic value, the why and how of an assembled program often requires a knowledge of the entire program or system. This knowledge is readily available to the programmer/system designer, but it can be acquired only with great difficulty by another person who needs to modify the program unless meaningful comments have been provided. Such comments are especially necessary when the contents of instructions are modified by the program and when non-standard subroutine returns are used.

SECTION IV

ASSEMBLER DIRECTIVES

4-1. INTRODUCTION

4-2. The QANTEL Business Assembly Language includes several instructions which may be entered as part of the program to perform such functions as beginning the program at a specific memory address, entering constant data into memory with the program, reserving areas of memory, etc. This section describes these assembler directives and provides coding examples where necessary for clarification.

4-3. ORIGIN CONTROL (ORG)

4-4. The Origin Control instruction is used to specify the memory location at which storage of the program begins. As shown in the example of figure 4-1, the OPERANDS portion of the coding form is used to specify the origin address. The Origin Control instruction actually resets the program location counter so that any statements following the instruction (program statements or data) will be loaded into consecutive memory positions following the specified address. An example of the Origin Control statement is also shown in statement 00100 of Appendix E.

NOTE

The only instruction which may be assembled below location 86_{10} is the Defined Area (DA) instruction, as this area of memory is occupied by the standard magnetic tape loader. The loader is automatically placed at the beginning of all object programs. The disc loader, used with object programs residing in a disc resident object library, occupies memory locations below 220_{10}. Define Area (DA) statements should not begin below 27_{10} since these areas may be written by the Processor.

4-5. END CONTROL (END)

4-6. The End Control instruction is used to signify the end of a program to the Q/BAL assembler program, and to indicate the address at which program execution begins. The End Control instruction is coded as shown in figure 4-2, with the OPERANDS portion of the coding form specifying the address or label of the first instruction to be executed. All programs are terminated with the End Control instruction.

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		ORG		4Ø96
		ORG		STR

Figure 4-1. Origin Control (ORG) Instruction Coding

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		END		BGN
		END		1ØØ

Figure 4-2. End Control (END) Instruction Coding

	PROGRAM I.D.	PROGRAM		
		LABEL 1 2 3 4 5 6 7 8	OP-Code 9 10 11 12 13	OPERANDS 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
1		@NAM	DC	'READ HEX TEST
2		DAT	DC	'24APR72'
3		AMT	DC	4591Ø3
4		DAT	DC	$Ø123456789ABCDEF
5				

Figure 4-3. Alphameric, Decimal and Hexadecimal Define Constant (DC) Coding

NOTE

The statement containing the End Control instruction and the branch address or label should never be deleted when updating a program.

4-7. DEFINE CONSTANT (DC)

4-8. The Define Constant instruction permits the programmer to enter necessary data into memory along with the assembled program. The Define Constant instruction is coded as shown in figure 4-3. The statements may or may not be labeled and the DC operation code is entered into the Op-Code portion of the coding form. The OPERANDS field of the first two statements shown in figure 4-3 are examples of the method used for coding text strings. Text strings will always be preceded and followed by an apostrophe ('). The third statement in figure 4-3 shows the coding format for a decimal constant and the fourth statement shows the coding format for hexadecimal constants. The first character of a decimal constant must be a numeric digit (0 through 9). The first character of a hexadecimal constant must be the dollar sign ($). Since each hexadecimal character occupies only one-half of a memory location, two hexadecimal characters are placed in each byte of memory. No attempt should be made to enter an odd number of hexadecimal characters.

4-9. To observe the coding rules for the DC statement, refer to statements 00200 through 00280 in Appendix E. In statement 00200, the defined constant data is a READ HEX TEST (terminated by a carrier return character). These 14 alphabetic characters are coded as shown in the first example of figure 4-3. The label of the DC is NAM and references the first (leftmost) character of the field since the pin address character (@) is used. The label NAM is coded in the LABEL field. The operation code DC is coded in the Op-Code field.

4-10. DEFINE ADDRESS CONSTANT (DAC)

4-11. The Define Address Constant instruction is used to define a two-byte address constant. An example of this is shown in figure 4-4. Upon completion of program assembly of this particular example DAC, the field "ADD" will contain a value two less than the two-byte address of the field labeled STP. If for example, the address of the STP label, coded in figure 4-4 is $102A_{16}$, the defined constant address would be 1028_{16}. Any direct expression may be used for address modification.

PROGRAM		
LABEL 1 2 3 4 5 6 7 8	OP-Code 9 10 11 12 13	OPERANDS 14 15 16 17 18 19 20
ADD	DAC	STP-2
LEN	DAC	.STP
PTR	DAC	Ø,1

Figure 4-4. Define Address Constant (DAC) Coding

4-12. DEFINE AREA (DA)

4-13. The Define Area instruction allows the programmer to reserve specific areas of memory for use during program operation. The assembler reserves these areas of memory for use during program operation by skipping over them instead of inserting program instructions or constant data. Figure 4-5 shows the coding for the DA instruction. An example of a Define Area instruction is also shown in statement 00300 of Appendix E. In the statement shown in figure 4-5, the area being reserved consists of ten memory locations, the highest numbered (right end), of which has the label BF1. The length of the Defined Area is coded as a direct expression, i.e., any symbol used within the expression must be defined in preceding statements. The expression (with a value of up to 0255) actually specified the number of consecutive memory positions to be skipped before entering the next program statement into memory. This allows data areas to overlay executable code (using the ORG statement).

PROGRAM		
LABEL (1–7)	OP-Code (9–12)	OPERANDS (14–20)
BF1	DA	1Ø
BF2	DA	BSZ

Figure 4-5. Define Area (DA) Coding

4-14. EQUATE (EQU)

4-15. Equate is an assembler directive that is used to assign the specified label to a particular memory address. The first example shown in figure 4-6 assigns the value 182 (decimal) to the symbol TA1. The second example assigns the memory location 10 positions before the location having the label BEG to the label TA2 and assigns a length of one (1) to TA2.

NOTE

The Equate assembler directive must appear in the program before any reference to the label assigned by the instruction.

4-16. EXECUTE (EXE)

4-17. Programs which would otherwise exceed the available memory can sometimes be made to fit by coding the program in two or more segments. The segments are separated by the EXE statement and the ORG (Origin Control) statement which are used to assign routines in different segments to the same memory locations. The effect of the EXE statement is to create an end block on the object tape which causes the program loading to terminate and execution to begin at the address or label specified in the EXE statement. The remainder of the program remains positioned on the input device and is loaded up to the next EXE or END statement from tape by moving a constant of 00F2 to location 10 (decimal) and branching to location 26 (decimal). To implement this procedure, locations 26 through 85 (decimal) cannot have been altered by the preceding segment of the program.

4-18. To illustrate the use of the EXE instruction in the overlaying of program segments, an example program is presented in figure 4-7. A typical use of this overlay technique may be to place program initialization routines in the first segment(s), and overlay them after they have been executed.

4-19. TYPEWRITER CONTROL (SKP, TYP)

4-20. Assembler directives are provided to control the I/O typewriter or line printer during the second pass listing of the assembly operation so that the programmer can eliminate the printing of selected portions of the program. By printing the program in a selective manner, such duplicate printing can be avoided to speed up the assembly process.

4-21. The operation codes used are SKP and TYP. Upon encountering the SKP OP Code, the assembler will skip the printing of the succeeding program statements until a TYP OP Code appears in the program. The TYP OP Code causes the assembler to resume printing of

	PROGRAM I.D.	PROGRAM		
		LABEL (1–7)	OP-Code (9–12)	OPERANDS (14–34)
1		TA1	EQU	182
2		TA2	EQU	BEG-1Ø,1
3				

Figure 4-6. Equate Instruction Coding

PROGRAM | PROGRAMMER

```
LABEL   OP-Code  OPERANDS             COMMENTS
*          SAMPLE PROGRAM TO ILLUSTRATE
*             USE OF EXE ASSEMBLER DIRECTIVE
*             FOR PROGRAM
*
TPØ     EQU   Ø         TYPER OUTPUT DEVICE
        ORG   1ØØ
STR     WRC   MS1,TPØ;.MS1
        MOV   CON;1Ø    THIS INSTRUCTION
*                       CAUSES REINITIALIZATION
*                       OF THE LOADER
        BRU   26        BRANCH BACK TO LOADER
@MS1    DC    'BEFORE OVERLAY
CON     DC    $ØØF2
        EXE   STR
        ORG   1ØØ
RST     WRC   MS2,TPØ;.MS2
        HLT   *
@MS2    DC    'AFTER OVERLAY
        END   RST
```

Figure 4-7. Implementation of the Execute (EXE) Instruction

the program statements. (The OP Codes SKP and TYP will always be typed). Generally, if it is known that the program being reassembled does not change up to a certain point, the SKP instruction would be used at the beginning of the program, and the TYP instruction inserted before the corrected area. The SKP and TYP instructions are coded as shown in figure 4-8.

PROGRAM

```
LABEL   OP-Code  OPERANDS
        SKP
        TYP
```

Figure 4-8. Typewriter Control Instructions

SECTION V

INPUT/OUTPUT INSTRUCTIONS AND DEVICES

5-1. INTRODUCTION

5-2. This section contains the basic information required to code read and write instructions, issue commands and properly check the status of each of the various QANTEL I/O devices. The information presented in this section includes a basic description of buffered and unbuffered I/O devices, I/O device status, and descriptions of each of the thirteen standard I/O instructions along with machine language format and coding examples. Also included in this section are descriptions of the methods for issuing commands and checking status for each type of the QANTEL I/O devices.

5-3. BUFFERED I/O DEVICES

5-4. The typewriter unit is an example of a device that is buffered within its respective controller when attached to the QANTEL/ANSWER Processor System. Other devices that are buffered include the discs, printers, 10-key, card reader, Video Display and various Communications Controllers. The buffer is part of the controller hardware, and does not occupy main memory positions. Data to be read from the typewriter keyboard to the processor is first typed serially, byte-by-byte, into the buffer until the buffer-filling operation is terminated. Buffered devices are self-terminating, and termination occurs in the typewriter when the buffer is fully loaded with 128 characters, when the carrier RETURN key is pressed, or when the TERM button is pressed. At this time, the keyboard is locked and the device notifies the processor of the terminated condition by means of a request for service. The processor then issues a Read instruction and empties the buffer, byte-by-byte, placing the data into main memory positions specified in the Read instruction, beginning with the lowest memory address. After the buffer has been emptied into main memory, the programmer can determine the firm status of the device and unlock the keyboard. The operator may then continue to type until termination is again encountered. In this operation (unlock-read-unlock), the keyboard lock-out occurs for only a fraction of a second while the processor receives and stores the buffered data, and would be unnoticeable to the operator.

5-5. Since the QANTEL/ANSWER Processor System is a serially organized processor, a Read instruction to an unloaded buffer in the typewriter would cause the processor to wait for the operator to fill the buffer at typing speed. This delay is not consistent with the buffering concept, so the QANTEL input/output organization includes an instruction (Set Read) that unlocks the typewriter keyboard or other input media and allows the buffer to be filled while the processor is doing other work. This feature eliminates idle processor time and enables the processor to operate independent of the buffered I/O device.

5-6. In the other direction, data is written from the processor main memory to the buffered device in serial form until the buffer is fully loaded, or until count zero is reached. At this time, the I/O device is terminated, and the processor goes on to subsequent instructions while the buffer contents are printed onto the typewriter page, printer page or stored on the disc.

5-7. The typewriter delivers data at a relatively slow speed when compared to the processor (14.7 characters per second for the typewriter, as compared to 136,000 characters per second for the processor). Since the buffer is not capable of receiving new data while still unloading characters to the printing or storage device, another attempt to write data to the previously addressed device would be delayed until the device has emptied the buffer. In this situation, the processor must wait until the pending Write command is taken by the device.

5-8. UNBUFFERED I/O DEVICES

5-9. The tape reader/punch and magnetic tape drives are examples of unbuffered devices which must have the full attention of the processor during any read or write operation. That is, the processor cannot be allowed to do other work as long as the unbuffered device is busy reading or writing data. The busy status of the device (or any I/O device) is reflected within the Status Byte by the read and write bits. Refer to paragraph 5-13.

5-10. Unlike buffered devices, some unbuffered devices are not self-terminating (such as the tape reader/punch). The Read and Write instructions used with an unbuffered device must have the count function, such as Read and Count, and Write and Count. With these instructions, the processor counts down from a predetermined amount as each character is transferred to or from the processor. When count zero is reached (all desired data has been moved), the processor sends a terminate command to the device. At this time, the device completes the current mechanical cycle and indicates to the programmer (by means of the Status Byte) that service by the processor is no longer required, and that the device has finished the operation. Status must be requested by the programmer in order to determine the condition of the device at any given time.

5-11. STATUS

5-12. Status is the term used to describe the condition of an I/O device. That is, if the device is busy, in need of service by the processor, inoperable, finished with all current operations, or involved in the Interrupt routine. The programmer can determine the status of any I/O device by means of the Status-In instruction, subsequently described in paragraph 5-53. When status of a device is requested with the Status-In instruction, the addressed device controller reacts by generating the Status Byte.

5-13. STATUS BYTE

5-14. When requested by the Status-In instruction, the Status Byte is produced by the device and fed to the processor to be stored in location 23 of reserved memory. From this location, the Status Byte can be examined by the program to determine device status. The format of the Status Byte is shown in figure 5-1.

5-15. Each of the eight bits in the Status Byte has a particular significance regarding the condition of the associated I/O device. For example, if the inoperable (27) bit is set, the device is inoperable and requires operator attention.

5-16. Flag Bits

5-17. The high order four bits (2^4 through 2^7) are the flag bits, and can differ in meaning between devices. The flag bits indicate information concerning the devices that are for the program. This could be a device inoperable condition, or a switch on the typewriter that is set by the operator to convey information to the

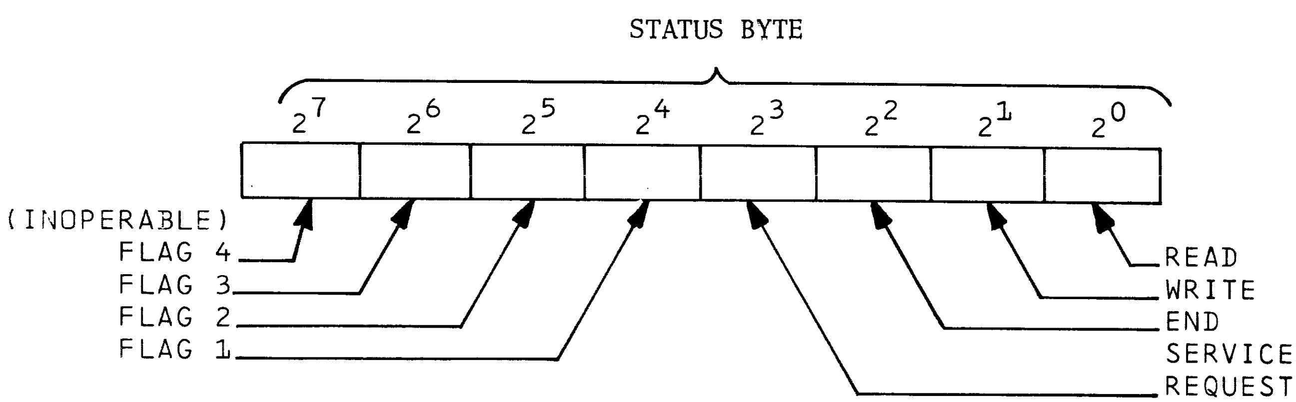


Figure 5-1. Status Byte Format

TABLE 5-1. STATUS BYTE FLAG BITS

FLAG NO. AND I/O CONTROL BYTE BITS	I/O DEVICE: TYPEWRITER	I/O DEVICE: TAPE READER/ PUNCH
Flag 1 Bit 2^4	Indicated Interrupt if Interrupt Feature is installed. Otherwise, Flag 1 as defined in the program	Not used
Flag 2 Bit 2^5	Flag 2 as defined in the program.	Reader inoperable (out of tape or no AC power).
Flag 3 Bit 2^6	Flag 3 as defined in the program	Punch inoperable (out of tape or no AC power).
Flag 4 Bit 2^7	Inoperable.	Total inoperable (no power). Terminates a read or write instruction.

program. The meaning of the flag bits for the typewriter and tape read/punch is described in table 5-1.

5-18. Status Bits

5-19. The low order bits of the Status Byte are device status bits. These bits are used by the micro-program logic in order to perform an I/O operation. They may also be used by the programmer to determine the status of the device prior to issuing an I/O instruction, such as Read or Write to a buffered device. The write operation in the buffered device concerns an I/O typewriter printing off-line (from the buffer). If status is requested during the write operation, the write bit (2^1) in the Status Byte would be set to indicate that the addressed device is busy writing, and that the exact status of the complete operation is not yet known. That is, the typewriter could still become inoperative while emptying the buffer, which could not be indicated by the Status Byte until the failure actually occurred. Consequently, status is not firm until the Read and Write bits are reset to indicate that the operation is complete. This method of checking status may also be used to simply check the success of a read or write operation.

5-20. In the buffered device, the programmer can issue a Set Read command to unlock the keyboard, examine status by means of the Status-In instruction and Status Byte, and when service request is set, issue a Read command. This technique enables the processor to perform other work while the buffer is filling. An attempt to use this technique with an unbuffered device, such as the tape reader/punch, will result in data overrun (loss of data because the device was not serviced in time). In the case of the unbuffered device, the Read and Count instruction should be issued and status should not be requested until after the device no longer reports busy status. Status of the unbuffered device may be checked shortly after termination is initiated. The unbuffered device ties up the processor for the duration of the read or write operation.

5-21. A truth table of the four status bits is shown in table 5-2. This table gives a brief explanation of the status bit configurations in reference to device status.

5-22. Read Sequence

5-23. A flow chart showing the logical procedure performed when reading from a *buffered* device is given in figure 5-2. The flow chart shows that the Set Read command is issued (to unlock the keyboard and allow the operator to type), followed at some time by a Status-In instruction. The Status-In instruction delivers the device status (by means of the Status Byte) to the processor, and check is made for service request. If the device is not yet determined, service will not be requested, so the processor performs other work. Status is examined repeatedly by the program until service request is true. At this time, the Read command is sent to the device and the buffer is emptied into the processor main memory.

TABLE 5-2. STATUS BITS

STATUS BYTE				DEVICE STATUS
2^3 SR	2^2 END	2^1 WR	2^0 RD	
X	X	X	1	Device is busy reading.
X	X	1	X	Device is busy writing.
0	1	0	0	Device is not busy and does not need service.
1	0	0	1	Device requires service.
1	0	1	0	Device requires service.

NOTE

Accessing the Status Byte can be interrupted if the Interrupt Feature is installed. In this situation, it is the responsibility of the interrupt handling procedure to store and restore status, and to determine which device is interrupting (by means of the Status-In instruction and flag 1 of the Status Byte).

5-24. **Firm Status**

5-25. When the I/O device no longer requires service by the processor, i.e., when the buffer has been loaded or emptied by the processor; or in the case of the unbuffered device, when informed by the processor (using the instructions with count) that data transfer is complete, the end bit (2^2) of the Status Byte is set. The programmer has no need to examine this bit. The fact that it is set does *not* indicate that the operation (read or write) is finished. It only indicates that the device no longer needs the processor to continue its task (e.g., during off-line printing from the buffer). When status is requested by the programmer, firm status is indicated by the end bit being set and the read and write bits being reset. Checking the end bit alone has no significant meaning. The read and write bits are reset when the device has actually finished reading or writing the last byte of data.

5-26. Although the read sequence for an unbuffered device is different, (see paragraph 5-20), the firm status technique is the same for all devices. When a write to a paper tape punch is terminated on count (the punch is not self-terminating), the termination, as far as the processor is concerned, occurs about 20 milliseconds before the last character is punched. Since the QANTEL/ANSWER Processor System will execute many instructions in 20 milliseconds, a Status-In instruction would show both the end bit (no more processor assistance needed) set and the write bit set. Again, the programmer should wait until the write bit has been reset before assuming status is firm.

5-27. A flow chart showing the program steps used to determine the success of a read or write operation (by means of obtaining firm status) is given for the buffered device in figure 5-3. After the Read or Write command is issued, status is requested. If the read or write bit of the Status Byte is set, the device is busy and the processor may go on to other work. Status is re-examined until neither read nor write are set. At this time, status is firm and can be evaluated by examining the flag bits for an inoperative or error condition.

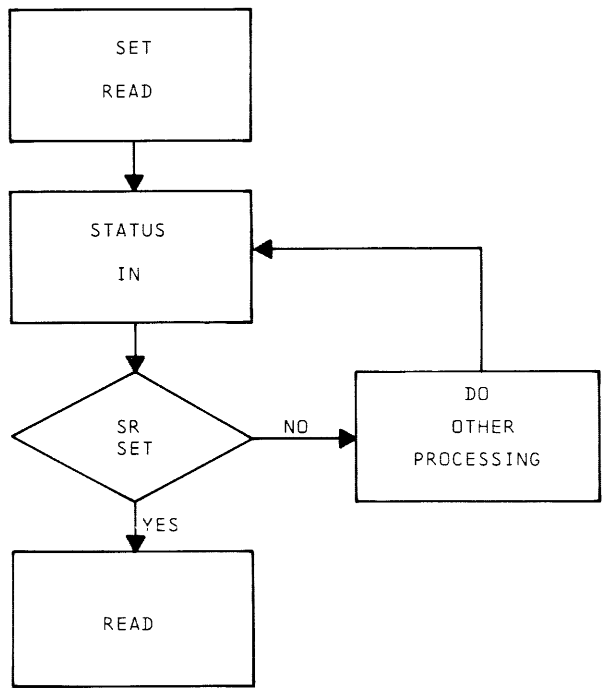


Figure 5-2. Reading a Buffered Device

5-28. **I/O INSTRUCTIONS**

5-29. The QANTEL/ANSWER Processor System is provided with 13 instructions to control operation of the I/O devices. Eight of the 13 instructions are variations of Read and Write, and the other five are I/O control. Each instruction is described separately in the following paragraphs with accompanying illustrations where necessary for clarification. All Read and Write instructions begin at the specified memory address and progress toward the high memory positions.

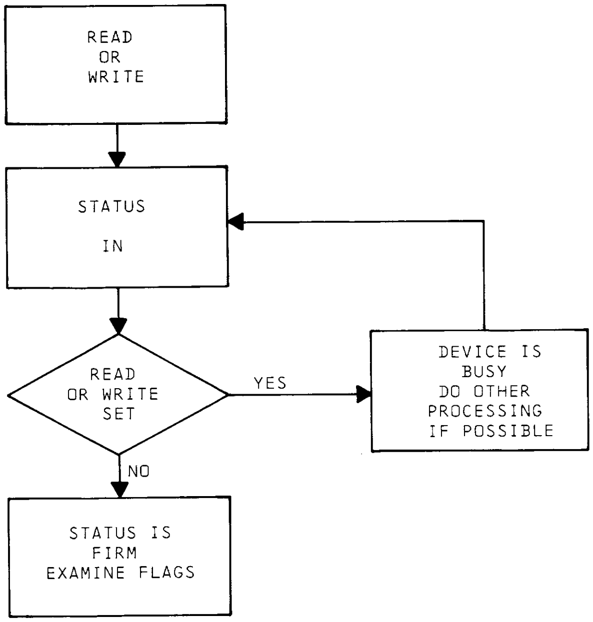


Figure 5-3. Insuring Firm Status

5-30. **I/O Read and Write Instructions**

5-31. READ - RD (OP CODE 0)

5-32. Read is a single-address instruction in which the processor reads data from the addressed device into the main memory. The data is placed in main memory beginning at the A operand address specified in the instruction and progressing toward the higher memory positions. The Read instruction can only be used with a self-terminating device. If used with a computer-terminating device, the device would attempt to read indefinitely, and would tie up the processor for the same length of time. In the Read instruction, the operation code variant is assumed by the operator to be zero. The Read instruction machine language format is shown in figure 5-4 and figure 5-5 is a coding example.

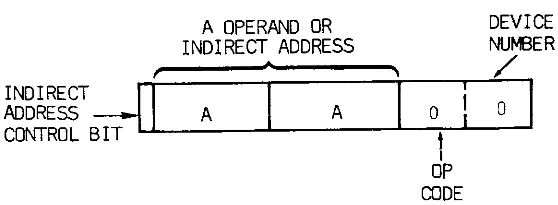


Figure 5-4. Read Instruction (RD) Machine Language Format Example

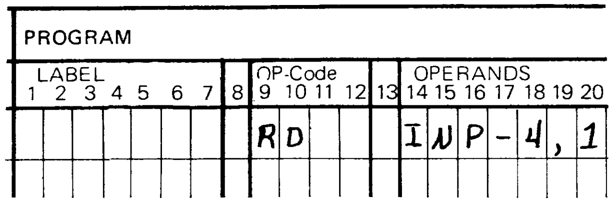


Figure 5-5. Read Instruction (RD) Coding Example

5-33. READ AND COUNT - RDC (OP CODE 0, VARIANT 2)

5-34. Read and Count is a two-address instruction in which the processor reads data from the addressed device into the main memory, beginning at the A operand address specified in the instruction. During the Read and Count operation, the processor counts down to zero from a specified amount as each character is transferred from the device to the processor. (The count is specified in the associated Load Address instruction). When the count reaches zero, the device is terminated by the processor.

5-35. The Read and Count instruction may be used with self-terminating or computer-terminating devices. In the case of the self-terminating device, the terminate may occur before the count reaches zero, such as when the buffer is empty. The terminate (end bit set in the Status Byte) generated by the device ends the Read and Count instruction, and the processor goes on to the next instruction. The Read and Count instruction (written in hexadecimal – machine language) illustrated in figures 5-6 and 5-7 tells device number one to read 80 (decimal) characters into main memory beginning at location 1000 (decimal).

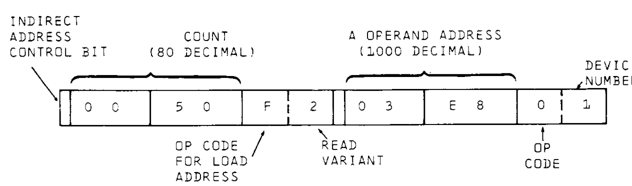


Figure 5-6. Read and Count Instruction (RDC) Machine Language Format Example

5-36. READ HEX - RHX (OP CODE 0, VARIANT 1)

5-37. Read Hex is a two-address instruction in which the processor reads data from the addressed (self-terminating) device into the main memory, beginning at the A operand address specified in the instruction. As each byte (character) is received, it is translated by the

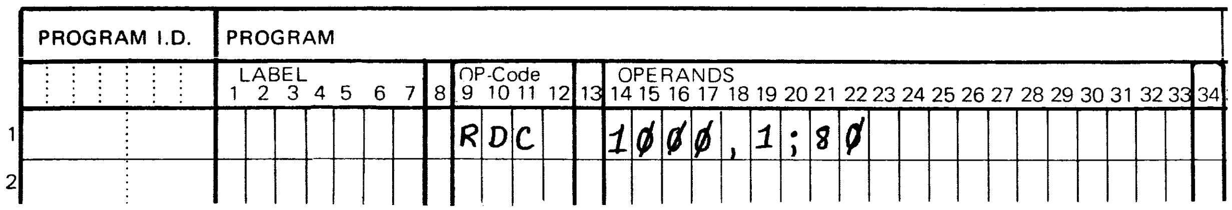


Figure 5-7. Read and Count Instruction (RDC) Coding Example

micro-program into a hexadecimal digit and is then combined into an eight bit byte. By combining digit 1 with digit 2, digit 3 with digit 4, etc., two characters from the device make one new byte for main memory. This method (used in the prceding Read and Count example– permits the programmer to use all eight bits of each memory location to express all 256 possible binary combinations of the eight bits. For further clarification, this method is illustrated in figure 5-8. If an uneven number of characters are read, the last one is lost.

5-38. The machine language format for the Read Hex instruction is shown in figure 5-9 and figure 5-10 is a coding example.

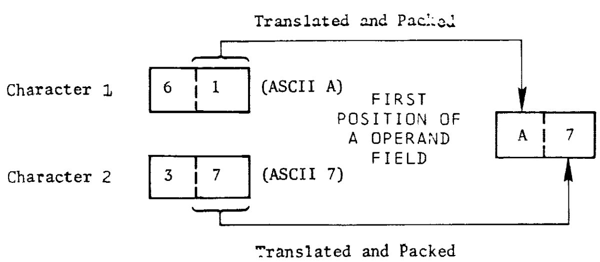


Figure 5-8. Read Hex Instruction Two-Byte Combination Method

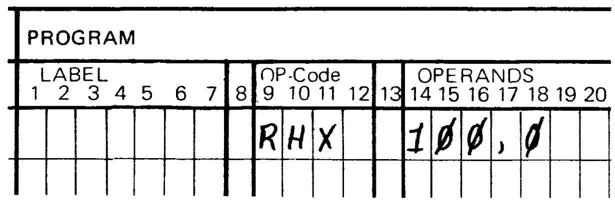


Figure 5-10. Read Hex Instruction (RHX) Coding Example

5-39. READ HEX COUNT - RHC (OP CODE 0, VARIANT 3)

5-40. Read Hex and Count is similar to the Read Hex instruction except that a count is established in the instruction so that it may be used with a computer-terminating device, or to obtain only a certain number of bytes. The count specified in the instruction is the number of main memory positions to be filled, not the number of bytes to be transferred to the processor. If an uneven number of bytes are read, the last byte is lost. The machine language format of the Read Hex and Count instruction is shown in figure 5-11 and figure 5-12 is a coding example.

5-41. WRITE - WR (OP CODE B, VARIANT 0)

5-42. Write is a single-address instruction in which the processor writes data to the addressed device from main

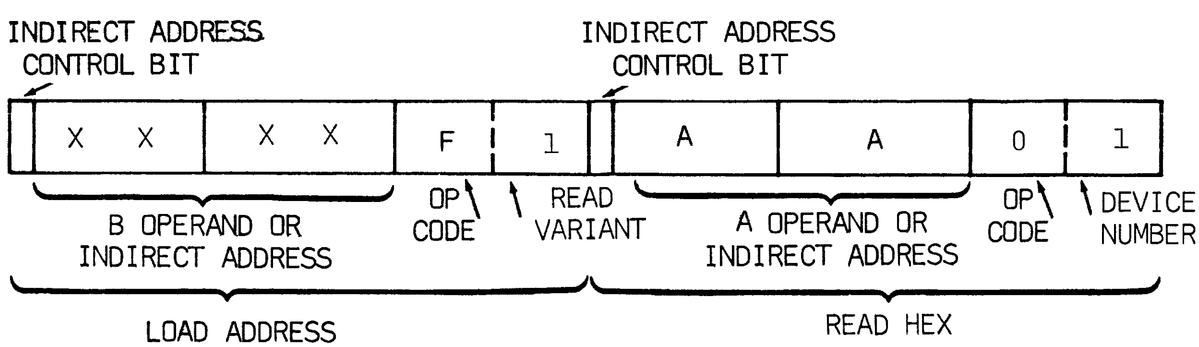


Figure 5-9. Read Hex Instruction (RHX) Machine Language Format

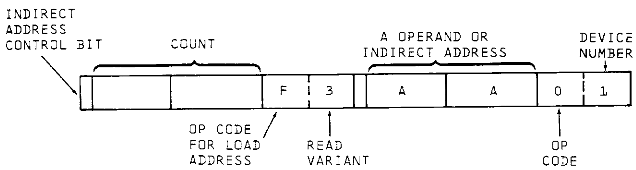


Figure 5-11. Read Hex and Count Instruction (RHC) Machine Language Format

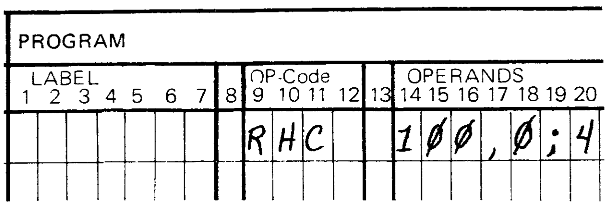


Figure 5-12. Read Hex and Count Instruction (RHC) Coding Example

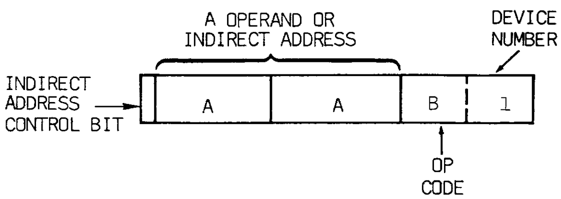


Figure 5-13. Write Instruction (WR) Machine Language Format

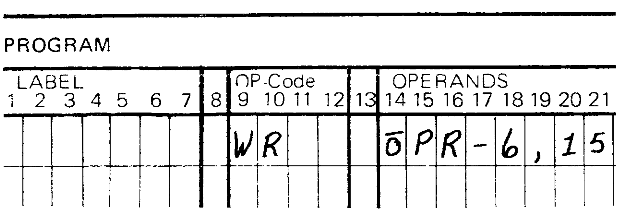


Figure 5-14. Write Instruction (WR) Coding Example

memory. The data is brought from main memory beginning at the A operand address specified in the instruction and progressing toward the higher memory positions. Like the preceding Read instruction, it can only be used with a self-terminating I/O device. The Write instruction machine language format is shown in figure 5-13 and figure 5-14 is a coding example.

5-43. WRITE AND COUNT - WRC (OP CODE B, VARIANT 2)

5-44. Write and Count is a two-address instruction in which the processor writes data to the addressed device from main memory, beginning at the A operand address specified in the instruction. During the Write and Count operation, the processor counts down to zero from a specified amount as each character is transferred from the processor to the device. (The count is specified in the associated Load Address instruction). When the count reaches zero, the device is terminated by the processor. The specified count for the typewriter cannot exceed 128 (decimal).

5-45. The Write and Count instruction may be used with self-terminating or computer-terminating devices. In the case of the self-terminating device, the terminate may occur before the count reaches zero, such as when the buffer is full. The terminate (end bit set in the Status Byte) generated by the device (as a result of the count reaching zero) ends the Write and Count instruction, and the processor goes on to the next instruction. The instruction format for Write and Count is similar to that of the Read and Count instruction, only a different operation code is used. Figure 5-15 shows the machine language format of the Write and Count instruction and figure 5-16 shows some coding examples.

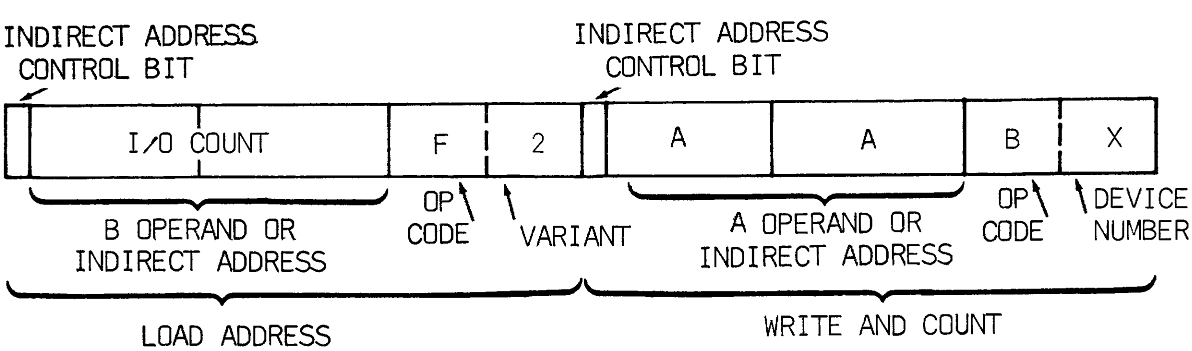


Figure 5-15.
Write and Count Instruction (WRC) Machine Language Format

PROGRAM I.D.	LABEL (1–7)	OP-Code (9–12)	OPERANDS (14–34)
1		WRC	DAR-49,8;5Ø
2		WRC	DHS,DEV;•DAS
3		WRC	@DTH,$D;@CNT
4			

Figure 5-16. Write and Count Instruction (WRC) Coding Example

5-46. WRITE HEX – WHX (OP CODE B, VARIANT 1)

5-47. Write Hex is a two-address instruction in which the processor writes data to the addressed (self-terminating device) from main memory, and is essentially the reverse action of the Read Hex instruction. The Write Hex instruction writes data from main memory beginning at the A operand address specified in the instruction. As each memory position of the processor is accessed, the eight bits are divided so that the most significant four bits are translated and transferred to the device as byte 1, while the least significant four bits are subsequently translated and transferred as byte 2. Similarly, the next position is split to make up bytes 3 and 4, and so on. The format for the Write Hex instruction is similar to that of the Read Hex instruction, only a different operation code is used. Figure 5-17 shows the machine language format of the Write Hex instruction and figure 5-18 is a coding example..

LABEL (1–7)	OP-Code (9–12)	OPERANDS (14–20)
	WHX	RMT,15

Figure 5-18. Write Hex Instruction (WHX) Coding Example

5-48. WRITE HEX AND COUNT - WHC (OP CODE B, VARIANT 3)

5-49. Write Hex and Count is similar to the Write Hex instruction except that a count is established in the instruction so that it may be used with a computer-terminating device, or to output a limited amount of information. The count specified in the instruction is the number of main memory positions to be transferred to the device, not the number of buffer positions to be filled. That is, to fill a 128-character buffer, a count of 64 (decimal) or 40 (hexadecimal) would be written in the instruction. The specified count for the typewriter cannot exceed 128 (decimal). The format for the Write Hex and Count instruction is similar to that of the Read Hex and Count instruction, only a different operation code is used. Figure 5-19 shows the machine language format for the Write Hex and Count instruction and figure 5-20 is a coding example.

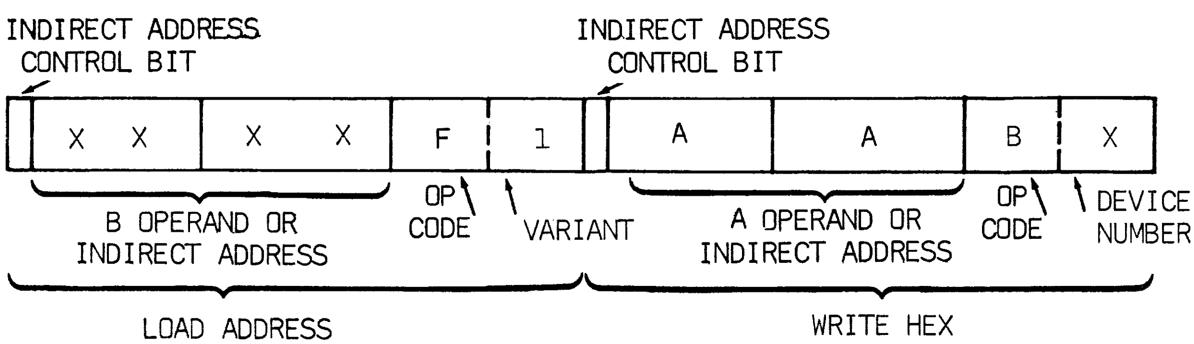


Figure 5-17. Write Hex Instruction (WHX) Machine Language Format

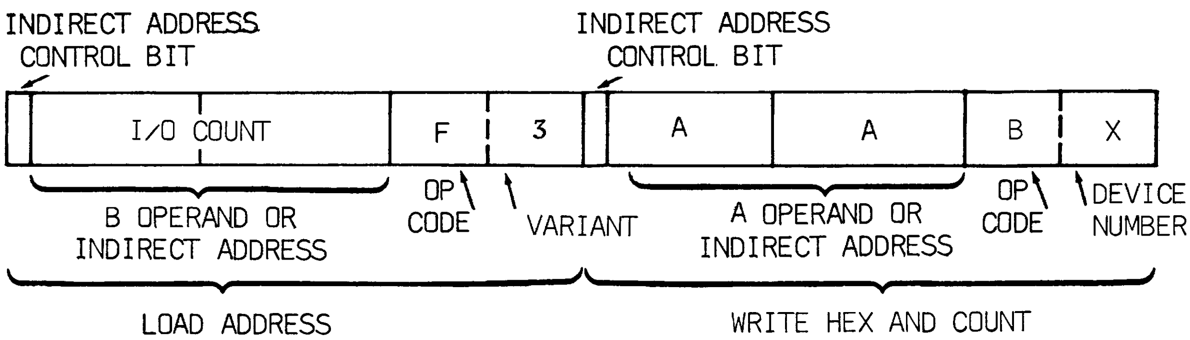


Figure 5-19.
Write Hex and Count Instruction (WHC) Machine Language Format

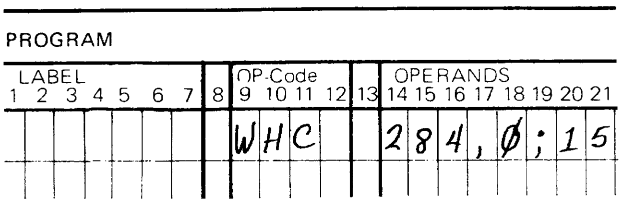


Figure 5-20. Write Hex and Count Instruction (WHC) Coding Example

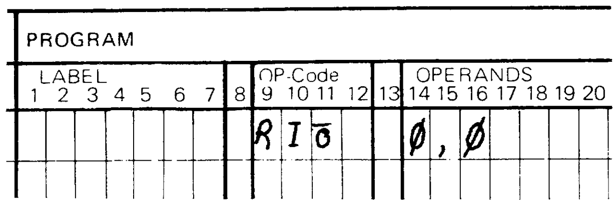


Figure 5-22. Reset I/O Instruction (RIO) Coding Example

5-50. **I/O Control Instructions**

5-51. RESET I/O – RIO (OP CODE 9, VARIANT E)

5-52. Reset I/O is a single-address instruction which may be used to interrupt any current operation being performed by the device, such as reading data to the associated buffer. In this case, execution of the Reset I/O instructions locks the keyboard to the operator and allows the program following the Reset I/O instruction to proceed. The machine language format of the Reset I/O instruction is shown in figure 5-21 and figure 5-22 is a coding example.

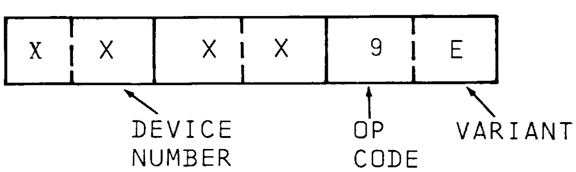


Figure 5-21. Reset I/O Instruction (RIO) Machine Language Format

5-53. STATUS-IN - SIN, SET READ - SRD AND DEVICE CONTROL - CTL (OP CODE 9, VARIANT D)

5-54. The Status-In, Set Read, and Device Control instructions provide different control functions for the addressed device, using only one basic instruction (operation code 9, variant D). Each instruction is a variation of the basic instruction shown in figure 5-23, and is signified by setting the appropriate bit of the most significant byte.

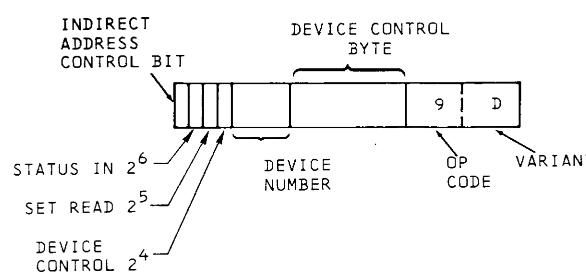


Figure 5-23. Status-In (SIN), Set Read (SRD) and Device Control (CTL) Instructions Machine Language Format

5-55. During the operation of the control instructions, the bits of the most significant byte are examined from left to right, and the first bit found to be set determines the type of instruction. For example, if both the Status-In and Set Read bits are set, the instruction would be taken as Status-In and the Set Read bit disregarded. In this manner, only one of the three control functions can actually be indicated in a single instruction. Status-In, Set Read, and Device Control are

TABLE 5-3. I/O CONTROL INSTRUCTION BITS

MOST SIGNIFICANT BYTE			DESIGNATED INSTRUCTION
2^6	2^5	2^4	
1	X	X	SIN (Status-In)
0	1	X	SRD (Set Read)
0	0	1	CTL (Device Control)

individually specified in the 2^6, 2^5, and 2^4 bits of the most significant byte in the instruction as listed in table 5-3. A functional description of the three commands is provided in the following paragraphs.

5-56. Status-In is set to the addressed device to determine the current status (refer to paragraph 5-11) of the device. Execution of the Status-In instruction causes the addressed device to generate a Status Byte which reflects the current status of the device, and which is subsequently placed in position 23 (decimal) of main memory for examination by the processor. By specifying a Status-In instruction (setting the 2^6 bit in the most significant byte), the Device Control Byte (refer to figure 5-23) may be used by the programmer to further clarify the type of status being checked for. That is, the status of a device may be checked for a particular condition (such as busy, service request, flag 1, etc.) with a single instruction, instead of first requesting status, and then performing a separate examination of the Status Byte to determine which bits are set.

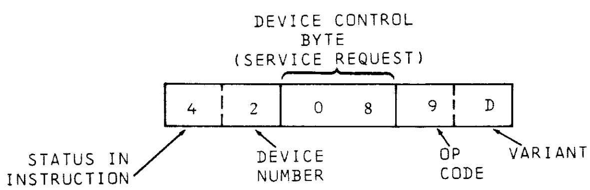


Figure 5-24. Status-In Instruction (SIN) Machine Language Format Example

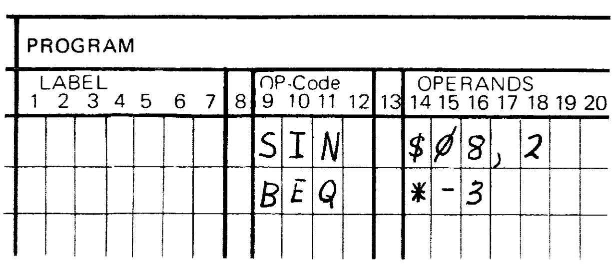


Figure 5-25. Status-In Instruction (SIN) Coding Example

5-57. To check device status for a particular condition, the appropriate bit(s) of the Device Control Byte must be set to enable an automatic Test Bit operation (refer to section II) with the corresponding bits of the Status Byte (described in paragraph 5-13). For example, if it is desired to check the addressed device for service request status, the 2^3 bit of the Device Control Byte must be set to check the 2^3 bit of the Status Byte. The 2^3 bit of the Status Byte, when set, indicates a request for service. The instruction for the example would appear in machine language (hexadecimal) as shown in figure 5-24 and figure 5-25 shows a coding example.

5-58. Set Read is an instruction used to unlock the typewriter keyboard so that the operator may proceed to fill the buffer while the processor is doing other work. The Set Read instruction cannot be used with unbuffered devices. The Set Read instruction is formed by setting the 2^5 bit in the most significant byte, and would appear in machine language (hexadecimal) as shown in figure 2-56. A Read or RIO instruction must be issued to that device before a CTL, SRD, WR, WRC, WHX, or WHC instruction can be used. Figure 5-27 shows a coding example of the Set Read instruction.

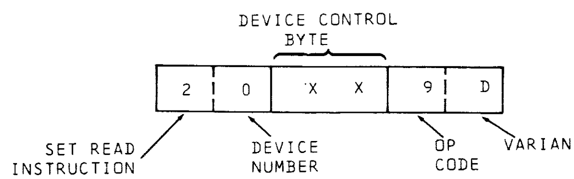


Figure 5-26. Set Read Instruction (SRD) Machine Language Format Example

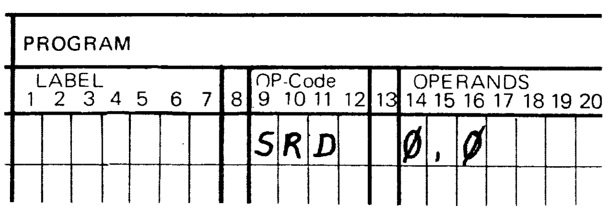


Figure 5-27. Set Read Instruction (SRD) Coding Example

NOTE

In the Set Read instruction, the Device Control Byte has no meaning, and any information included in this byte is disregarded during the operation.

5-59. The Device Control instruction may be used by the programmer to perform various functions with the

TABLE 5-4. DEVICE CONTROL BYTE AND TYPEWRITER SIGNAL LAMPS

DEVICE CONTROL BYTE								OPERATION
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
X	X	X	X	0	1	0	0	None
X	X	X	X	0	1	0	1	Lamp 1 set to on
X	X	X	X	0	1	1	0	Lamp 2 set to on
X	X	X	X	0	1	1	1	Bot lamps set to on
X	X	X	X	0	0	0	0	None
X	X	X	X	0	0	0	1	Lamp 1 set to off
X	X	X	X	0	0	1	0	Lamp 2 set to off
X	X	X	X	0	0	1	1	Both lamps set to off

different I/O devices. Setting the Device Control (2^4) bit of the most significant byte in the I/O control instruction causes the Device Control Byte to be considered during the operation. Within the Device Control Byte, the individual bits can be set to indicate various commands to the addressed device. For example, typewriter signal lamps may be turned on and off by the program to indicate any desired program condition to the operator. Control of the communications lamps is a function of the Device Control Byte, whose bits may be set as listed in table 5-4 to obtain the desired results.

5-60. The Device Control instruction is then used to turn the typewriter lamps on and off by using the Device Control Byte as shown in table 5-4. Figure 5-28 shows an example of the machine language format of a Device Control instruction in which lamp 2 is set to off. Figure 5-29 shows a coding example.

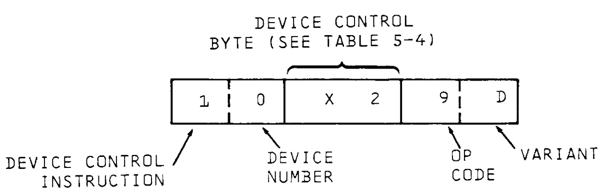


Figure 5-28. Device Control Instruction (CTL) Machine Language Format Example

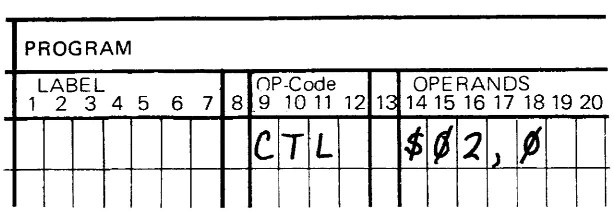


Figure 5-29. Device Control Instruction (CTL) Coding Example

5-61. READ STATUS 2 - RS2 (OP CODE 8, VARIANT 4)

5-62. The Read Status 2 instruction is an I/O control instruction that performs approximately the same function as the Status-In instruction. The exception is that Read Status 2 is used to check the second status byte on certain devices having two status bytes. Devices that have two status bytes are video displays, certain discs and certain QANTEL communications controllers. Additional information on the QANTEL communications controllers is contained in the QANTEL Data Communications Techniques manual. Figure 5-30 shows the machine language format of the Read Status 2 instruction.

NOTE

The Read Status 2 instruction is not available in the QANTEL Processors at this time and will be implemented at a future date.

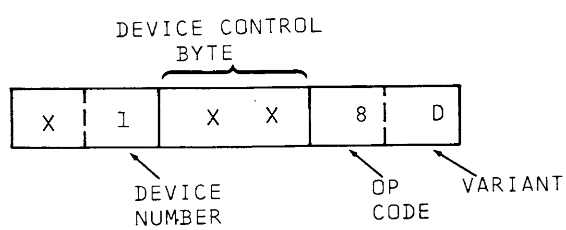


Figure 5-30. Read Status 2 (RS2) Instruction Machine Language Format

5-63. **Initial Program Load (IPL)**

5-64. The term IPL refers to the first instruction or series of instructions performed by the processor. To perform IPL, the operator presses the IPL switch on the processor control panel to generate a general reset to all devices. This action resets the program register to zero and generates a Read Hex instruction addressed to device number zero. The associated main memory address for the Read Hex instruction is zero, and because the current program address (program register) is zero, the data (hexadecimal) to be read into location zero will serve as the first instruction to be fetched.

5-65. The amount of hexadecimal data (instructions) that can be entered from the device into the main memory (beginning at location zero) is determined by the typewriter buffer. In the case of the typewriter, 128 characters can be entered before automatic termination occurs and the instructions are initiated.

5-66. **Programming Notes**

5-67. When the buffer-filling operation on a keyboard device is terminated by a carrier RETURN character, the character is actually placed in the buffer. If the subsequent transfer of buffer contents to main memory is effected by a Read instruction, the carrier RETURN character will be transferred to main memory.

5-68. The Read instruction associated with IPL stores the final address plus one in locations 21 and 22. As a result, there is an effective limit to the number of instructions that could be entered through IPL before initiating some other method for program entry.

5-69. INPUT/OUTPUT DEVICES

5-70. Typewriter

5-71. The typewriter is the basic I/O device. It is supplied as part of the basic QANTEL/ANSWER Processor System and is used to initialize all operations which require the loading of a program. The typewriter is a buffered device having a 128-byte hardware buffer. The buffer allows the operator to enter data while the processor is processing, thereby preventing the relatively slow speed of the typewriter from impairing the performance of the processor. The typewriter will read or write up 128 bytes of data as a result of a single read or write command. Read operations without a specified length are terminated by the typewriter when the buffer is full (with 128 alphabetic/decimal or 64 hexadecimal characters), or by the operator when the TERM or carrier RETURN key is pressed. (Write operations without a specified length are illegal and will cause indeterminate results). Read operations which do have a specified length also allow the operator to fill the buffer. However, those characters which exceed the specified length are not read into memory when the buffer is emptied, but are lost. For example, during a Read with Count, the operator enters more data into the buffer than specified in the instruction, only the specified number of characters will be read into memory when the operator presses the carrier RETURN or TERM key. It is also possible for the operator to press the carrier RETURN or TERM key before the specified number of characters have been entered. The program can check for this condition by examining locations 21 and 22 (decimal) of reserved memory to determine the final address-plus-one of the operation. It should be noted that the difference between the carrier RETURN and the TERM keys is that the carrier RETURN key terminates the read operation and enters a carrier return character into memory (if the specified length allows), while the TERM key simply terminates the read operation without the inclusion of the carrier return character. Those read and write instructions which may be used with the typewriter are indicated in table 5-5.

5-72. TYPEWRITER FLAGS AND SIGNALS

5-73. The typewriter is equipped with three switch/lamps (flags) designated FLG 1, FLG 2, and FLG 3; and is also equipped with two signal lamps designated SIG 1 and SIG 2. This arrangement of flags and signals provides the necessary operator-program communication. The three flags can be selectively set by the operator, and can be examined, separately or collectively, by means of the Status-In instruction described in paragraph 5-56. The signal lamps can be set and reset, separately or together, by the program using the Device Control instruction described in paragraph 5-59.

NOTE

The flags are reset (other than by the operator) by the execution of any read, write, Set Read or Reset I/O instruction.

5-74. The setting of any flags by the operator is reflected in the device status. To check for the presence of a set flag, the program must examine the device status to determine if any of the flags, or a particular flag, have been set. The method of determining typewriter status is described in paragraph 5-75.

5-75. TYPEWRITER STATUS CHECKING

5-76. The status of the typewriter is determined by means of the Status-In (SIN) instruction. The methods of coding this instruction to determine the various possible status conditions are shown in figure 5-31. (Note that column 13 or 14 contains the device number). The SIN instruction is usually followed by a Branch On Non-Zero (BNZ) or Branch Equal (BEQ) instruction, and if the condition checked for is true, the BNZ is executed. Conversely, if the condition is checked for is not true, the BNZ is not executed and the program falls through to the next instruction or a BEQ instruction may be executed.

5-77. The programmer may choose to check more than one status condition in this manner by using several SIN and BNZ statements. Or, he may choose to check two or more conditions with one SIN instruction. For example, if it is desired to determine whether or not the operator has set any of the flags or if the typewriter is inoperable, the SIN statement would be coded as shown in figure 5-32. The $F0 (hexadecimal) entered in columns 8 to 10 is obtained by adding the coding for the individual conditions shown in figure 5-31 (i.e., $10_{16} + 20_{16} + 40_{16} + 80_{16} = F0_{16}$). If any or all of the tested conditions are true, the subsequent BNZ instruction would be executed. (If the program needs to determine precisely which of the four tested conditions is true, further status checking must be performed).

NOTE

When status indicates busy reading or busy writing, the operation is still in progress and the remaining status conditions are subject to change. As a result, status checking for operability should be performed when the device does not indicate busy reading or writing.

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		SIN		$Ø1,Ø
		SIN		$Ø2,Ø
		SIN		$Ø3,Ø
		SIN		$Ø4,Ø
		SIN		$Ø8,Ø
		SIN		$ØC,Ø
		SIN		$1Ø,Ø
		SIN		$2Ø,Ø
		SIN		$3Ø,Ø
		SIN		$4Ø,Ø
		SIN		$5Ø,Ø
		SIN		$6Ø,Ø
		SIN		$7Ø,Ø
		SIN		$8Ø,Ø

Figure 5-31. Typewriter Status-In Instruction Coding

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		SIN		$FØ,Ø

Figure 5-32. Checking Multiple Status Conditions

5-78. TYPEWRITER DEVICE CONTROL

5-79. The Device Control (CTL) instruction is used to operate the signal lamps on the typewriter. The CTL instruction is coded as shown in figure 5-33 to perform the indicated functions, provided the device is not busy.

5-80. TYPEWRITER RESET I/O

5-81. The Reset I/O (RIO) instruction is coded as illustrated in figure 5-34. This instruction performs the function of resetting all flags. It may also be used to terminate any typewriter operation and lock the keyboard to the operator.

5-82. TYPEWRITER SET READ

5-83. The Set Read (SRD) instruction is coded as shown in figure 5-35 and may be used to simply unlock the typewriter keyboard. When the typewriter keyboard is unlocked by the SRD instruction, the operator may enter data into the 128-byte buffer without interrupting the processor. With this method of typewriter operation, the system may perform other processing while the operator enters the data. The subsequent read instruction may be issued at any time to move the data from the typewriter buffer into memory. If the operator has not finished the entry, the processor waits.

TABLE 5-5.
DEVICE AND ALLOWABLE READ AND WRITE INSTRUCTIONS

	I/O DEVICE						
INSTRUCTION (MNEMONIC)	TYPE WRITER	LINE PRINTER	PAPER TAPE RDR/PCH	MAGNETIC TAPE	CARD READER	TEN-KEY KEYBOARD	DISC DRIVE
Read (RD)	yes	no	no	yes	yes	yes	yes
Read and Count (RDC)	yes	no	yes	yes	yes	yes	yes
Read Hex (RHX)	yes	no	no	no	yes	no	no
Read Hex and Count (RHC)	yes	no	no	no	yes	no	no
Write (WR)	no	yes	no	no	no	no	yes
Write and Count (WRC)	yes	yes	yes	yes	no	yes	yes
Write Hex (WHX)	no	yes	no	no	no	no	no
Write Hex and (WHC)	yes	yes	no	no	no	no	no

	PROGRAM I.D.	PROGRAM LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20 21 22 23 24	
1				CTL		$Ø1,Ø	Turn SIG 1 off
2				CTL		$Ø2,Ø	Turn SIG 2 off
3				CTL		$Ø3,Ø	Turn SIG 1 and SIG 2 off
4				CTL		$Ø5,Ø	Turn SIG 1 on
5				CTL		$Ø6,Ø	Turn SIG 2 on
6				CTL		$Ø7,Ø	Turn SIG 1 and SIG 2 on
7							

Figure 5-33. Typewriter Device Control Instruction Coding

PROGRAM LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		RIŌ		Ø,Ø

Figure 5-34. Typewriter Reset I/O Instruction Coding

PROGRAM LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		SRD		Ø,Ø

Figure 5-35. Typewriter Set Read Instruction Coding

5-84. Magnetic Tape Transports

5-85. Data written to the various QANTEL magnetic tape transports is arranged into records and files. The records are separated by inter-record gaps, while files are separated and ended by end-of-file indicators (see paragraph 5-86). Record length is variable and is determined by the character count specified in the Write and Count instruction. Write and Count is the only write instruction used with the magnetic tape transport since the device cannot determine the desired length of the record and terminate the operation itself. The Read and Read with Count instructions may be used with the magnetic tape transport. The Read (without count) instruction causes the device to read one complete record of data and halt at the beginning of the next record. The Read with Count instruction causes one complete record of data to be passed by the read mechanism, but only the specified number of characters are read into memory. Error checking is performed on the entire record regardless of the specified count.

5-86. Records may be any desired length as long as they are at least 12 characters. (Although a 12-character minimum is possible, a 20-character minimum is recommended). Records shorter than 12 characters (bytes) are considered erroneous when read, and are so indicated in the device status as an illegal block length. An exception to the 12-character minimum is the end-of-file (EOF) indicator which is written to the tape unit when it is desired to designate the end of a particular file. The EOF indicator is a one-byte record consisting of 13_{16} (00010011_2), and is written to the device after the last record of the file.

5-87. MAGNETIC TAPE DEVICE CONTROL

5-88. The Device Control (CTL) instruction is used to perform such functions as erase, check read, backspace and rewind. The Erase command causes the transport to erase the next 3½ inches of tape. The Check Read command causes the tape unit to read one record and perform error checking, the result of which is reflected in the device status. This command usually follows a write and backspace operation. The Backspace command causes the tape transport to back up to the beginning of the preceding record. The Rewind command causes the tape transport to rewind the tape to the beginning-of-tape position. The CTL instruction is coded as shown in figure 5-36 to perform the indicated operations.

5-89. MAGNETIC TAPE STATUS CHECKING

5-90. Checking the status of the magnetic tape transport in the most efficient manner is a somewhat more complex procedure than that for the other I/O devices. To simplify the programmer's task of properly using the magnetic tape transport, an Input/Output Control System (IOCS) Utility is provided by QANTEL Corporation. These subroutines include the necessary status checking and are described in the QANTEL Input/Output Control System (IOCS) Operating Instructions.

5-91. Disc Drive

5-92. The QANTEL disc systems can use any of three different sizes of disc, a 7.6M Byte disc, a 30.7M Byte disc and a 60M Byte disc. These units are sectored and organized as follows:

7.6 Byte Disc	30.7M Byte Disc	60M Byte Disc
10 disc surfaces	20 disc surfaces	20 disc surfaces
200 trks/surface	200 trks/surface	400 trks/surface
10 sectors/track	10 sectors/track	10 sectors/track
380 bytes/sector	768 bytes/sector	768 bytes/sector
7.6M Bytes storage	30.72M Bytes storage	61.44M Bytes storage

	PROGRAM I.D.	LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–21)	(22–34)
1				CTL		$ØØ,X	ERASE
2				CTL		$Ø1,X	CHECK READ
3				CTL		$Ø2,X	BACKSPACE
4				CTL		$Ø3,X	REWIND
5							

Figure 5-36. Magnetic Tape Transport Device Control Instruction Coding

BYTE 6	BYTE 5	BYTE 4 / BYTE 3	BYTE 2	BYTE 1
DEVICE NUMBER	TRACK/HEAD GROUP	TRACK (ØØ-99)	HEAD (Ø-9)	SECTOR (Ø-9)

Figure 5-38. Disc Address Field Organization

5-93. The QANTEL/ANSWER Processor System ROM contains a disc Seek instruction (SEK). This instruction is actually a seek and fill buffer command. Read or Write to any sector is usually preceded by a single-address (three byte) Seek instruction with OP Code 8 and Variant 0. Figure 5-37 shows the machine language format of the Seek Instruction.

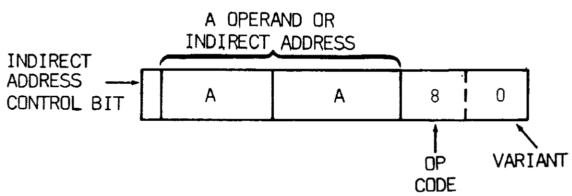


Figure 5-37. Disc Seek Instruction (SEK) Machine Language Format

5-94. The A operand address field of the Seek instruction points at byte six (6) of the actual six byte disc address field. Figure 5-38 shows the organization of the disc address field.

5-95. The information contained in the six bytes of the disc address field are as follows:

a. Byte 1 - decimal sector number Ø through 9 through 9
b. Byte 2 – decimal head number Ø through 9
c. Byte 3 and 4 – decimal number ØØ through 99 designating the track address
d. Byte 5 – 7.6M Byte Disc: decimal number Ø or 1 designating the track group ØØ through 99 or 1ØØ through 199,
   3Ø.7M Byte Disc: decimal number Ø through 3; bit 2^0 indicates head group – Ø if head group Ø through 9 and 1 if head group 1Ø through 19 – bit 2^1 indicates track group,
   60M Byte Disc: decimal number Ø through 7; same as 30.7M Byte Disc except additional head group are indicated by 2^2 bit.
e. Byte 6 – hexadecimal device number with a 3 zone 3Ø to 3F

5-96. DISC STATUS CHECKING AND CONTROL

5-97. When writing programs for disc systems using the QANTEL Business Assembly Language it is recommended that the programmer use the QANTEL Input/Output Control System (IOCS) to perform device Status checking and control. If the programmer desires to write his own status checking and control routines he should refer to figures 5-39 and 5-40 for the coding format of the Status-In instruction for the 7.6M Byte disc and the

	PROGRAM I.D.	LABEL (1–7)	OP-Code (9–12)	OPERANDS (14–26)	
1			SIN	$Ø1,X	READ BUSY
2			SIN	$Ø2,X	WRITE BUSY
3			SIN	$Ø4,X	END
4			SIN	$Ø8,X	SERVICE REQUEST
5			SIN	$1Ø,X	ERROR
6			SIN	$2Ø,X	MARKED SECTOR
7			SIN	$4Ø,X	INVALID SEEK
8			SIN	$8Ø,X	INOPERABLE
9					

Figure 5-39. 7.6M Byte Disc Status-In Instruction Coding

Line	PROGRAM I.D.	LABEL (1–7)	OP-Code (9–12)	OPERANDS (14–19)	
1			SIN	$Ø1,X	READ BUSY
2			SIN	$Ø2,X	WRITE BUSY
3			SIN	$Ø4,X	END
4			SIN	$Ø8,X	SERVICE REQUEST
5			SIN	$1Ø,X	SEEK OR WRITE TERMINATED
6			SIN	$Ø6,X	ERROR, MARK SECTOR, INVALID SEEK
7			SIN	$8Ø,X	INOPERABLE
8					

Figure 5-40. 30.7M Byte and 60M Byte Disc Status-In Instruction Coding

LABEL (1–7)	OP-Code (9–12)	OPERANDS (14–20)	
	CTL	$Ø1,X	Inhibit Track Verify
	CTL	$Ø2,X	Inhibit Read After Write Check
	CTL	$Ø3,X	Inhibit Track Verify and Read After Write Check
	CTL	$1Ø,X	Disable Termination Interrupt (2314 Type Disc Only)
	CTL	$14,X	Enable Termination Interrupt (2314 Type Disc Only)

Figure 5-41. Disc Drive Control Instruction Coding

30.7M Byte or 60M Byte discs, respectively, and to figure 5-41 for the coding format of the Device Control instruction.

5-98. **Card Reader**

5-99. The card reader is a buffered device having an 80-byte hardware buffer. As the card is read, column by column, the Hollerith coded characters are translated to ASCII and stored in the buffer. (A blank column on the card shows up as an ASCII space or blank character). Read operations with the card reader are performed by means of either the Read or the Read with Count instruction. A read operation causes one card to be read into the buffer and then to memory. If the instruction is a Read with Count, only the specified number of characters (up to 80) are transferred from the buffer to memory. Data transfer is terminated by the card reader when the buffer becomes empty, or by the processor when the count is exhausted.

NOTE

The Read with Count instruction could be used to conserve memory in cases where only the front portion of the card needs to be read.

5-100. CARD READER SET READ

5-101. The Set Read (SRD) instruction may be used to read a card into the hardware buffer while the processor operates on the data read from the previous card. (The processor can do a significant amount of processing in the length of time it takes to read a card into the buffer). The SRD instruction causes the card reader to read a single card into the buffer in an off-line mode. A

LABEL	OP-Code	OPERANDS	
	SIN	$Ø1,X	Device busy reading
	SIN	$Ø4,X	Read operation finished
	SIN	$Ø8,X	Service request
	SIN	$1Ø,X	Read error – data unreliable (300 CPM Card Reader Only)
	SIN	$2Ø,X	Feed error – card jam, etc. (300 CPM Card Reader Only)
	SIN	$4Ø,X	Hopper empty/stacker full or hold
	SIN	$8Ø,X	Inoperable, feed error

Figure 5-42. Card Reader Status-In Instruction Coding

	PROGRAM I.D.	LABEL	OP-Code	OPERANDS	
1			SIN	$2Ø,X	Reader out of tape/no ac power
2			SIN	$4Ø,X	Punch out of tape/no ac power
3			SIN	$8Ø,X	Inoperable (both units)
4					

Figure 5-43. Paper Tape Reader/Punch Status-In Instruction Coding

subsequent Read or Read with Count instruction causes the buffer contents to be read into memory.

5-102. CARD READER STATUS CHECKING

5-103. Card reader status is checked by means of the Status-In (SIN) instruction. The methods of coding this instruction to determine the various possible status conditions of the card reader are shown in figure 5-42. In addition, a flowchart of a typical card reader status checking routine is presented in figure 5-44.

5-104. **Paper Tape Reader/Punch**

5-105. The status of the paper tape reader/punch is determined in a manner similar to that of the typewriter, in that the SIN and BNZ instructions are used. The methods of coding this instruction to determine the various possible status conditions of the paper tape reader/punch are shown in figure 5-43. The read and write instructions used with the paper tape reader/punch are listed in table 5-5.

5-106. **Ten-Key Keyboard**

5-107. The ten-key keyboard is a buffered numeric (ASCII) input device. Read operations with the numeric keyboard are performed by means of either the Read or Read with Count instruction. As the operator presses and releases the numeric keys, the data is read into the 31 character buffer. When the terminate (T) key is pressed by the operator, service request is set and the processor reads the buffer into the designated portion of memory.

5-108. TEN-KEY DEVICE CONTROL

5-109. The program can, by means of the Device Control (CTL) instruction, cause the device to produce an audible tone to signal the operator. Coding the CTL instruction to turn the signal on and off is shown in figure 5-45.

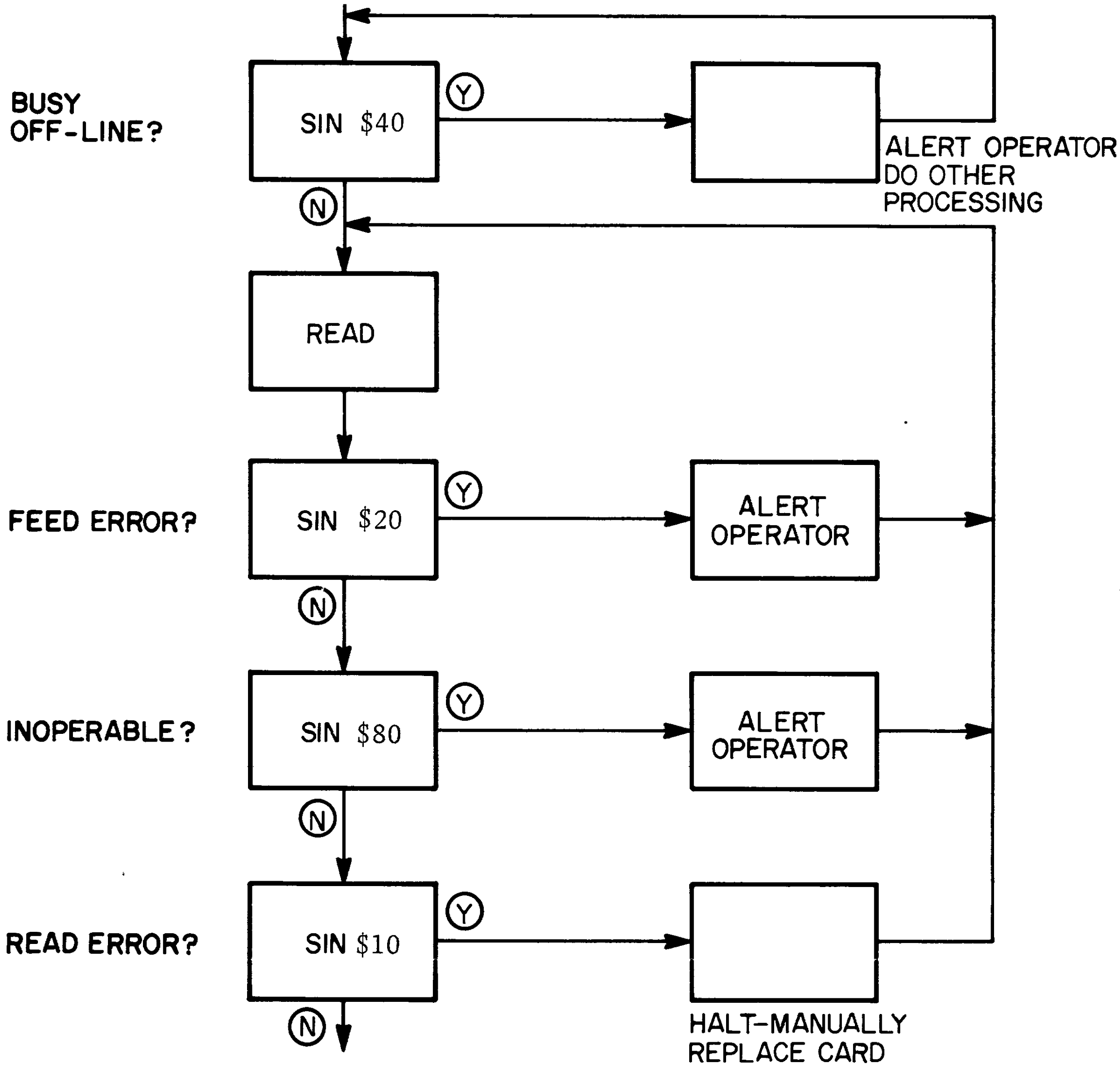


Figure 5-44. Card Reader Status-Checking Flowchart

	PROGRAM I.D.	LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–23)	
1				CTL		$ØØ,X	Turns signal off
2				CTL		$Ø1,X	Turns signal on
3				CTL		$Ø6,X	Disable termination interrupt
4				CTL		$Ø7,X	Enable termination interrupt
5							

Figure 5-45. Ten-Key Keyboard Device Control Instruction Coding

	PROGRAM I.D.	PROGRAM LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–27)	
1				SIN		$Ø1,X	Device busy reading
2				SIN		$Ø4,X	Read terminated
3				SIN		$2Ø,X	F2 set
4				SIN		$4Ø,X	F3 set De
5				SIN		$8Ø,X	Device inoperable
6							

Figure 5-46. Ten-Key Keyboard Status-In Instruction Coding

	PROGRAM I.D.	PROGRAM LABEL (1–7)	8	OP-Code (9–12)	13	OPERANDS (14–25)	
1				CTL		$3Ø,X	Skip to Channel Ø
2				CTL		$31,X	Skip to Channel 1
3				CTL		$32,X	Skip to Channel 2
4				CTL		$33,X	Skip to Channel 3
5				CTL		$34,X	Skip to Channel 4
6				CTL		$35,X	Skip to End-of-Form
7				CTL		$36,X	Skip to Top-of-Form
8				CTL		$37,X	Skip to Next Line Only
9							

Figure 5-47. 60-100 LPM Serial Printer Device Control Instruction Coding

5-110. TEN-KEY STATUS CHECKING

5-111. Status checking is performed by means of the Status-In (SIN) instruction, which is also used to check flags 2 and 3 (F2 and F3). These flags are similar to those used in the typewriter, but can only be turned on during a Read. The operator can set the flag(s) by pressing a button to indicate certain conditions or function to the program. The methods of coding the SIN instruction to determine the various possible status conditions of the ten-key keyboard are shown in figure 5-46.

5-112. **Line Printers**

5-113. QANTEL Corporation has several printers that are used with the QANTEL/ANSWER Processor System, a 60-100 LPM Serial Printer, a 200 LPM Line Printer, a 245-1100 LPM Line Printer and a 700-1800 LPM Line Printer. Data may be written to these printers by means of the Write, Write and Count, Write Hex, and Write Hex and Count instructions. If the Write or Write Hex (without count) instructions are used, the printers will print one line (132 characters) and halt. If a count is specified in the write operation, that number of characters is printed on a single line and the operation is terminated. The printers will automatically advance to the next line after each write operation.

5114. LINE PRINTER DEVICE CONTROL

5-115. The Device Control (CTL) instruction is used to control the Vertical Format Unit contained within the various printers. The Vertical Format Unit uses one-inch

	PROGRAM I.D.	LABEL	OP-Code	OPERANDS	
1			CTL	$ØØ,X	Skip to Top-of-Form (Channel 1)
2			CTL	$Ø1,X	Skip to Next Line Only (Line Feed)
3			CTL	$Ø2,X	Skip to Channel 2 (Vertical Tab)
4			CTL	$2Ø,X	Reset: Allow Termination Interrupt
5			CTL	$6O,X	Set: Allow Termination Interrupt
6					

Figure 5-48. Line Printer Device Control Instruction Coding

	PROGRAM I.D.	LABEL	OP-Code	OPERANDS	
1			SIN	$Ø2,X	Write Busy
2			SIN	$Ø4,X	End
3			SIN	$8Ø,X	Inoperable or No Paper
4					

Figure 5-49. 60-100 LPM Serial Printer Status-In Instruction Coding

	PROGRAM I.D.	LABEL	OP-Code	OPERANDS	
1			SIN	$Ø2,X	Write Busy
2			SIN	$Ø4,X	End
3			SIN	$1Ø,X	Interrupt
4			SIN	$4Ø,X	Always Set
5			SIN	$8Ø,X	Inoperable or No Paper
6					

Figure 5-50. Line Printer Status-In Instruction Coding

punched tape in the form of a loop to control the positioning of the continuous form paper used with the printers. The punched tape loop is created by the programmer or systems person, to meet the needs of the particular program and/or report. The CTL instruction for the 60-100 LPM Serial Printer is coded as shown in figure 5-47 to perform the indicated functions. The CTL instruction for the three line printers is coded as shown in figure 5-48 to perform the indicated functions.

NOTE

If the Vertical Formal Unit finds the top-of-form position before the requested position, the form feeding halts at that point.

5-116. LINE PRINTER STATUS CHECKING

5-117. The status of the printers is determined in a manner similar to that of typewriter, in that the SIN and BNZ instructions are used. The methods of coding this instruction to determine the various possible status conditions of the 60-100 LPM Serial Printer are shown in figure 5-49 and figure 5-50 shows coding for the line printers.

5-118. **System Clock/Interval Timer**

5-119. The System Clock provides time-of-day in 10 millisecond increments up to 23 hours, 59 minutes, 59.99 seconds, thereafter resetting itself to zero and resuming its count. The Interval Timer can be set to a time as large as 99 hours, 59 minutes and 59.99 seconds and uses the Interrupt feature to inform the processor when the interval has elapsed.

5-120. SETTING SYSTEM CLOCK

5-121. Time-of-day is set into the System Clock by a Device Control (CTL) Ø9 instruction followed by a write instruction (eight bytes). The system clock automatically terminates a write-without-count operation after eight bytes have been received. The eight bytes representing time-of-day are to be transmitted with the most significant byte (tens-of-hours) first. Figure 5-51. shows the coding to set time-of-day into the System Clock.

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		CTL		Ø9,3
		WR		TIME,3

Figure 5-51. System Clock Time-Of-Day Setting

5-122. READING INTERVAL TIMER

5-123. The current amount of time remaining in the Interval Timer is read by means of a Device Control (CTL) ØC instruction, followed by a normal read instruction. The device self-terminates on a read-without-count operation after eight bytes have been sent. Figure 5-52 shows the coding to read the interval from the Internal Timer.

PROGRAM

LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20
		CTL		ØC,3
		RD		INTR,3

Figure 5-52. Interval Timer Read Coding

5-124. A desired interval is set into the Timer by a normal write instruction. The device self-terminates a write-without-count operation after eight bytes have been received.

	PROGRAM I.D.	LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20 21 22 23 24	
1				SIN		$Ø1,X	Read Busy
2				SIN		$Ø2,X	Write Busy
3				SIN		$Ø4,X	End
4				SIN		$Ø8,X	Service Request
5				SIN		$1Ø,X	Interrupt (Interval Elapsed)
6				SIN		$2Ø,X	Time Incorrect
7							

Figure 5-53. System Clock/Interval Timer Status-In Instruction Coding

5-125. SYSTEM CLOCK/INTERVAL TIMER STATUS CHECKING

5-126. The status of this device is determined by means of the Status-In (SIN) instruction. The methods of coding this instruction to determine the various possible status conditions of the Clock/Timer are shown in figure 5-53.

NOTE

Timer Incorrect will be set as result of powering up the system, or initiating the "Test" feature. It is reset by a CTL Ø9, Load Time-of-Day. The "Test" feature consists of issuing the device a CTL Ø2 instruction, and causes the clock to speed up so that 24 hours of running is reduced to 5 minutes, 38 seconds.

SECTION VI

ASSEMBLER OPERATION

6-1. **INTRODUCTION**

6-2. This section contains the operating instructions for the QANTEL Business Assembly Language (Q/BAL) assembler programs. Three separate assembler programs are used to assemble programs coded in the QANTEL Business Assembly Language. These programs consist of two Pass 1 programs and one Pass 2 program. These programs are described in complete detail in later paragraphs of this section.

6-3. QANTEL Corporation also supports several library maintenance programs which can be used with both source files and object programs or libraries that are created using the (Q/BAL) assembler programs. Descriptions and operating instructions for these various library maintenance programs are contained in the QANTEL Software Operating Instructions binder. These programs are the Q/BAL Source Converter (LMSDCTX), Source Library Maintenance (LMSDMTX), Object Library Maintenance (OBJTPTP), Object Library Content (OBJDIRT), Disc Object Maintenance (OBJTPDK), Disc Object Library Condense (OBJCOND), and Disc Object Library Directory List and Delete (OBJDIRD).

6-4. **PROGRAM LOADING**

6-5. The Q/BAL Assembler Object programs can reside on either magnetic tape or disc. In order for the programmer or operator to use these programs he must be able to load the program into main memory. The following paragraphs provide initial program load (IPL) procedures used to load the object programs into main memory from either magnetic tape or disc.

6-6. **IPL From Magnetic Tape Library**

6-7. The following procedure should be used to load object programs into main memory from a magnetic tape object library.

a. Mount the magnetic tape library on a tape transport and bring the tape to LOAD point by pressing the LOAD pushbutton twice. When the tape is at LOAD point, the LOAD indicator will light.
b. Press the IPL pushbutton.
c. Using the typewriter, enter the magnetic tape bootstrap 00030X (where X is the device number of the tape transport which has the library mounted on it) and press carrier RETURN.
d. The typewriter will print out PROGRAM ID? Enter the seven character program identifier and press carrier RETURN.
e. The library tape will be searched for the requested program and the program will be loaded into memory.

NOTE

If the typewriter prints out NOT FOUND followed by PROGRAM ID? during program loading, reenter the desired program identifier and press carrier RETURN. If the typewriter prints PARITY ERROR or the SIG 1 and SIG 2 lamps light, the procedure should be repeated starting with step a. If the problem continues, cleaning and/or service for the tape transport, or replacement of the magnetic tape is indicated.

f. When the program loading is completed, program execution will begin.

6-8. IPL From A Disc Library

6-9. The Disc Loader that resides in sectors) and 1 of a disc object library is used to execute programs from the disc. To use the Disc Loader, the following procedures should be follows:

a. Mount the object library disc pack onto the disc drive and close the disc drive cover.
b. Press the POWER ON push-button on the disc drive. The POWER ON indicator should light and the disc pack will begin to rotate.
c. When the disc is up to speed, the 0 indicator will light, approximately 30 seconds after energizing the disc drive. If the SECTOR LOCK indicator comes on, de-energize the disc drive and attempt to turn on again. If the SECTOR LOCK indicator continues to come on, there is a malfunction in the disc drive.
d. When the 0 indicator lights, press the ENABLE ON push-button to enable the disc system. The ENABLE ON indicator will light.
e. Press IPL, enter 0022810X (where X is the disc drive number 0-9 or A-F) and press carrier RETURN. PROGRAM ID? will type out.
f. Enter the seven digit mnemonic of the program to be used and press carrier RETURN. The program will load into memory and when loading is complete, program execution will begin.

NOTE

If the typewriter prints out NOT FOUND followed by PROGRAM ID? during program loading, reenter the desired program identifier and press carrier RETURN. If the SIG 1 and SIG 2 lamps light, the IPL procedure should be repeated starting with step e. If the program continues, cleaning and/or service for the disc drive, or replacement of the disc library is indicated.

6-10. ASSEMBLER PROGRAMS

6-11. The three Q/BAL assembler programs are defined as follows:

LPADC1X *Pass 1* of the assembler using *typewriter or punched card input* to create or update source programs on magnetic tape. *Magnetic tape or disc resident.*

LPADD1X *Pass 1 of the assembler using typewriter, disc or magnetic tape input* to create or update source programs on disc or magnetic. *Magnetic tape or disc resident.*

LPADA2X *Pass 2* of the assembler using magnetic tape or disc to produce an object program on paper tape or magnetic tape. and a source listing on typewriter or printer. *Magnetic tape or disc resident.*

The following paragraphs provide the operating procedures necessary to use the Pass 1 and Pass 2 programs. A description of the error messages and recovery procedures is provided at the end of the operating instructions.

6-12. Assembler Program Operating Instructions

6-13. PASS 1 OPERATION

6-14. The operating procedures for using either of the two Pass 1 programs (LPADC1X or LPADD1X) are identical except for the input/output devices that may be used. To use the Pass 1 programs, load the desired Pass 1 program from the program library and perform the following steps:

a. The program modification base and identifier message followed by CRD RDR DEV #? will type out.
b. If using card input, enter the one digit hexadecimal device number of the card reader. If not using card input, press carrier RETURN.
c. OUTPUT DEVICE NUMBER? will type out. If using the LPADD1X program with disc output, press carrier RETURN and DISC SELECTED ENTER DEVICE NUMBER will type out. Enter the one digit hexadecimal disc device number and press carrier RETURN. If using LPADC1X or LPADD1X without disc output, enter the one digit hexadecimal output device number and press carrier RETURN'
d. UPDATE, YES-NO will type out. If a new program is being assembled, enter NO, press carrier RETURN and proceed with step e. If an existing source program is being updated

enter YES, press carrier RETURN and skip to step g.

e. PROGRAM ID will type out. Enter the seven character mnemonic program identified of the new program being created and press carrier RETURN.

f. 00020 will type out. If the typewriter is being used as the input device (typewriter will always be the input device when creating a new program using LPADD1X) commence entering the program. If the card reader is being used as the input device, type in a space and CRD (CRD), and the assembler will begin to read the new program from cards.

g. INPUT DEVICE NUMBER? will type out. If using LPADD1X with disc input, press carrier RETURN and DISC SELECTED, PROGRAM ID will type out. Enter the seven digit program identifier of the program being updated, press carrier RETURN and skip to step i. If using LPADC1X or LPADD1X without disc input enter the one digit hexadecimal input device number and press carrier RETURN.

h. PROGRAM ID will type out. Enter the seven character mnemonic program identifier and press carrier RETURN.

i. CHANGE ID? YES-NO will type out. If the identifier of the program being updated is to be changed enter YES, press carrier RETURN and continue to step j. Otherwise, enter NO, press carrier RETURN and skip to step k.

j. PROGRAM ID will type out. Enter the new seven character mnemonic program identifier and press carrier RETURN.

k. RENUMBER, YES-NO will type out. If the program is to be renumbered, enter YES and press carrier RETURN. If the program is not being renumbered, enter NO and press carrier RETURN. When not renumbering a program, a new line number must be entered with each new line in the update.

l. 00020 will type out. If the update is being entered from cards, enter CRD and press carrier RETURN. If the typewriter is being used to enter the update, commence entering the update lines appropriate.

m. If it is desired to delete an existing statement from the program, enter the reference number of the statement followed by a comma, followed by a repeat of the reference number. For example, to delete statement number 00240, enter the following: 00240,00240.

NOTE

The statement containing the End Control instruction and the branch address or label should never be deleted when updating a program.

n. If it is desired to delete a group of statements, enter the reference number of the first statement, followed by a comma, followed by the reference number of the last statement in the group to be deleted. For example, to delete statements 00240 through 00280, enter the following: 00240,00280.

o. To terminate the update operation and generate the remainder of the new source program tape without interruption, enter a reference number greater than that of the last statement in the program (e. g. 99999).

6-15. PASS 2 OPERATION

6-16. Load the Pass 2 program from the program library and perform the following steps:

a. The program modification level and identifier message followed by INPUT DEVICE NUMBER? will type out.

b. Mount the source tape or disc.

c. If using magnetic tape source input, enter the one digit hexadecimal device number of the tape drive on which the source tape is mounted, and press carrier RETURN. If using disc source input, press carrier RETURN and DISC SELECTED ENTER DEVICE NUMBER will type out. Enter the one digit hexadecimal disc device number and press carrier RETURN.

d. ASSEMBLE COMPLETE FILE? YES-NO will type out. If a complete object file is being assembled enter YES, otherwise enter NO. Press carrier RETURN.

e. OBJECT OUTPUT? YES-NO will type out. If an object output is desired, enter YES, press carrier RETURN and continue with step f. If no object output is desired, enter NO, press carrier RETURN and skip to step h.

f. OBJECT OUTPUT DEVICE NUMBER? will type out. Enter the one digit hexadecimal output device number and press carrier RETURN.
g. LISTING? YES-NO will type out. If a source listing is desired, enter YES, press carrier RETURN and continue with step h. If no source listing is desired, enter NO, press carrier RETURN and skip to step 1.
h. LISTING DEVICE? T or P will type out. If the typewriter is being used as the listing device, enter T, press carrier RETURN and continue with step i. If the printer is being used, enter P, press carrier RETURN and skip to step j.
i. If YES was entered in step d, and a complete file is being assembled, SET TABS TO 23, 31, 40 will type out and the Pass 2 program will begin assembling. Skip to step n. If NO was entered in step d, and a complete file is not being assembled. SET TABS TO 23, 31, 40 AFTER SPECIFYING PROGRAM ID PLACE PAPER TO TOP OF PAGE will type out. Skip to step 1.
j. PRINTER DEVICE NUMBER will type out. Enter the one digit hexadecimal printer device number and press carrier RETURN.
k. If YES was entered in step d and a complete file is being assembled, the Pass 2 program will begin assembling immediately. Skip to step n. If NO was entered in Step d and a complete file is not being assembled, continue to step 1.

NOTE

When a complete file is being assembled, i.e., the answer to the program question ASSEMBLE COMPLETE FILE? YES-NO was YES, all files on the source tape or disc will be assembled by the Pass 2 program and PROGRAM ID will not be requested.

m. When the selected program has been assembled PROGRAM ID will type out again. If no other programs are to be assembled, press carrier RETURN and continue to the next step. If another program is to be assembled, enter the seven character identifier of the new program, press carrier RETURN and the new program will begin assembling. This step will be repeated until a carrier RETURN is entered after PROGRAM ID is typed out.
n. ADDITIONAL FILES? YES-NO will type out. If additional source files are to be assembled enter Y and continue to the next step. If the assembly is complete, enter N and the tapes will rewind.
o. MOUNT FILE ON INPUT DEVICE will type out and the START/STOP lamp will light. Mount the new source file to be assembled on the input device selected in step c and press START/STOP.
p. If the answer to step d was Y the assembly will begin immediately. If the answer to step d was N, PROGRAM ID will type out. Enter the seven character identifier of the program to be assembled, press carrier RETURN and the selected program will begin assembling. When this assembly is completed, step m will be repeated.

6-17. Error Messages

6-18. When errors are incurred in a program being assembled, the Q/BAL assembler program will type out an error message followed by the line number and the actual erroneous line of coding. This will allow the programmer or operator to enter a corrected line of coding in place of the erroneous coding line. The following list shows the error messages that are produced by the Q/BAL assembler program and describes the meaning of each error message.

MESSAGE	DEFINITION
OP CD	Appears when the Op Code specified by the programmer cannot be found in the assembler program Op Code table, i.e., the Op Code used is not a legitimate machine operation code or a legitimate assembler directive.
REC2NDOPR	Requires second operand – produced when the assembler has been unable to find a second or B operand for an instruction that requires two operands. Example: The Move (MOV) instruction always requires two operands.

MESSAGE	DEFINITION
2NDOPRILL	Second operand illegal – machine operation specified by the programmer may have only one operand and two have been provided. Example: The Store Accumulator (STA) instruction may have only one operand.
OVR/UNDRFLOW	Overflow or underflow – meaning that the evaluation of an addressed expression has produced a value greater than 32,767 or, if signed, a value less than minus 16,384.
LENILL	Length illegal – meaning that the machine operation specified by the programmer does not allow a length. Example: The logical operation Exclusive Or (XOR) does not allow an explicit assignment of length since this operation is always performed on only one byte of memory.
DEVNOTSPEC	Device not specified – produced only for input/output instructions for which the programmer has not specified an explicit device number.
EXCESSHEX	Excessive number of hexadecimal digits – produced when a hexadecimal address value specified by the leading dollar sign contains more than 4 hexadecimal digits.
SYMBOLERR	Symbol error – produced when the assembler attempts to extract an illegal symbol from an input line. Symbols must begin with a letter followed by up to 4 additional characters which must be either letters or digits.
DUPLICATE TAG	Produced when a symbol appears as a label more than once in a program. This message will be produced on all subsequent encounters of that label.
TAG TABLE FULL	Produced when more symbols appear as labels than can be stored in the assembler programs internal symbol table. The maximum number of symbols which may be stored in the symbol table is 350.
UNDEFINED FORCE ZERO	This error message will be produced when a symbol that has not been encountered and defined as a label is used to specify lengths or devices in machine operations, or lengths in a Define Area (DA) statement. If typewriter input is being used this message must be answered either yes (Y or carrier RETURN) or no (N or any other character). If the answer to the question is yes, the assembler program will insert a value of 0 for the value it could not obtain. If the answer is no, the assembler program will retype the line containing error so that it may be reentered.
TOO BIG	Device control specification out of limits – appears when the operand field of a Device Control (CTL) or a Status-In (SIN) instruction exceeds the hexadecimal value FF.

MESSAGE	DEFINITION
ILLCHAR	Illegal character – appears when a Define Constant (DC) statement declared as a hexadecimal constant contains characters other than the letters A – F or the decimal digits 0 – 9.
UNEVN#HEX#S	Uneven numbers of hexadecimal digits – all hexadecimal constants declared in Define Constant (DC) statement must contain pairs of hexadecimal digits. One byte of memory contains one pair of hexadecimal digits.
WHAT'S IT	This message will appear when a Define Constant (DC) statement is analyzed and neither a decimal constant, a hexadecimal constant or a text string can be found.
TOO BIG	This message will appear when a Define Area (DA) statement operand is evaluated and the value obtained is greater than 256.
TOO SMALL	This message is presented when an expression is evaluated for a Define Area (DA) statement and the value obtained is zero or negative.
LENTOOBIG	Length too large – presented when an expression is evaluated for the purposes of obtaining a length for a machine operation and the value obtained is too large to be stored in the file reserved for that length. The limit is 16 for most instructions except for the Move (MOV) instruction which allows a value up to 256. This message will never appear for an I/O instruction.

APPENDIX A. HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895

APPENDIX A. HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (CONTINUED)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

APPENDIX A. HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (CONTINUED)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687

APPENDIX A. HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (CONTINUED)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

APPENDIX A. HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (CONTINUED)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

For numbers outside the range of the table, add the following values to the table figures:

HEXADECIMAL	DECIMAL	HEXADECIMAL	DECIMAL
1000	4096	8000	32768
2000	8192	9000	36864
3000	12288	A000	40960
4000	16384	B000	45056
5000	20480	C000	49152
6000	24576	D000	53248
7000	28672	E000	57344
		F000	61440

APPENDIX B. ASCII CODE

USASCII CODE

B_4-B_1 → / B_8-B_5 ↓		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0001	1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0010	2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0011	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	∧	_
0110	6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	7	p	q	r	s	t	u	v	w	x	y	z	{	¬	}	~	DEL

APPENDIX C. POWERS OF 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25

APPENDIX D
QANTEL STANDARD INSTRUCTION SET

INSTRUCTION	MNEMONIC	OPERATION CODE	VARIANT
Read	RD	0	0*
Read Hex	RHX	0	1*
Read & Count	RDC	0	2*
Read Hex & Count	RHC	0	3*
Add Decimal	ADD	1	
Subtract Decimal	SBD	2	
Multiply Decimal	MPY	3	
Divide Decimal	DIV	4	
Load	LD	5	
Move	MOV	6	
Store Accumulator	STA	6	
Compare Logical	CMP	7	
Seek	SEK	8	0
Disc Bootstrap	- - -	8	1
Translate ***	TRN	8	2
Search Equal***	SEQ	8	3
Read Status 2****	RS2	8	4
And	AND	9	0
Or	OR	9	1
Exclusive Or	XOR	9	2
Test Bit	TBT	9	3
Move Numeric	MN	9	4
Move Zone	MZ	9	5
Compare Decimal	CD	9	6
Shift Bit Left	SBL	9	7
Shift Bit Right	SBR	9	8
Edit	EDT	9	9
Unedit	UED	9	A
Pack	PAK	9	B
Unpack	UPK	9	C
Device Control	CTL	9	D**
Set Read	SRD	9	D**
Status-In	SIN	9	D**
Reset I/O	RIO	9	E
Return From Interrupt	RTI	9	F
No Operation	NOP	A	0
Branch On Overflow	BOV	A	1
Branch On Minus	BMI (BGT)	A	2
Branch On Non-Zero	BNZ (BNE)	A	3
Branch Equal	BEQ (BZ)	A	4
Branch Not Minus***	BNM (BLE)	A	5
Unconditional Branch	BRU	A	7
Halt and Branch	HLT	A	8
Branch and Link	BLI	A	9
Micro Instruction Mode	MIM	A	B
Write	WR	B	0*
Write Hex	WHX	B	1*
Write and Count	WRC	B	2*
Write Hex and Count	WHC	B	3*
Add Binary	ADB	D	
Subtract Binary	SBB	E	
Load Address	LDA	F	

*Variant is located in the Load Address Least significant byte.

**In CTL 2^4 bit of MSB should be set, in SRD 2^5bit of MSB should be set, and in SIN 2^6 bit of MSB should be set.

***Available on newer model processors only.

****Not available at this time, for future implementation.

APPENDIX E
READING AN ASSEMBLER LISTING

```
00020                                  *    READ HEX TEST
00040                                  *
00060                                  *    DEFINE ADDRESS OF ERROR ROUTINE
00080                                  *
00100                                       ORG 4096
00120                                  *
00140                                  *    DEFINE DATA
00160                                  *
00180                                  PDV  EQU 1               PUNCH DEVICE #
00200  1000  72656164206865            @NAM DC  'READ HEX TEST
00220  100E  70757420746170            @MSG DC  'PUT TAPE IN READER PRESS START
00240  102D  656E6420746573            @END DC  'END TEST'
00260  1035  30313233343536            @OUT DC  '0123456789ABCDEF'
00280  1045  0123456789ABCD            DAT  DC  $0123456789ABCDEF
00300  104D                            IN   DA  8
00320  1055  000EF2      1000B0        BGN  WRC NAM,0;.NAM      TYPE NAME
00340  105B  0010F2      1035B1             WRC OUT,PDV;16      PUNCH DATA
00360  1061  001FF2      100EB0             WRC MSG,0;.MSG
00380  1067              106AA8             HLT *+3
00400  106A  0008F3      104D01             RHC IN-.IN+1,PDV;.IN READ DATA
00420  1070  104CF8      105478             CMP IN;DAT          TRANSLATED OK?
00440  1076              0100A3             BNZ 256             GO TO ERR RTN
00460  1079  0008F2      102DB0             WRC END,0;.END
00480  107F              1055A8             HLT BGN
00500                                       END BGN
```

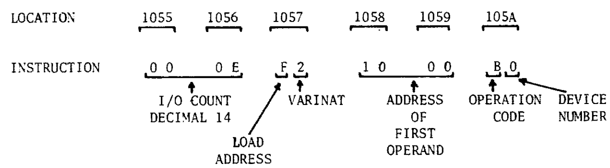
1. REFERENCE NUMBER
2. MEMORY ADDRESS (HEXADECIMAL)
3. MACHINE LANGUAGE DATA
4. MACHINE LANGUAGE INSTRUCTIONS
5. SYMBOLIC LANGUAGE INSTRUCTIONS (CORRESPOND TO CODING SHEET FORMAT)

1. **REFERENCE NUMBER**
A Reference Number is assigned by the Assembler Program, Pass 1, to each statement in the program. The first assigned number is always 00020 and succeeding statement numbers are incremented by 20. During an update operation, the programmer specifies the statement number of the statement to be changed or deleted. If statements are to be added during the update operation, the programmer must specify a number between the preceding and succeeding statement numbers unless such statements follow a deletion or modification of existing statements.
2. **HEXADECIMAL MEMORY ADDRESS**
Hexadecimal memory addresses are assigned by the Assembler Program, Pass 2. The first memory address is the hexadecimal equivalent of the decimal number entered in the Origin Control (ORG) statement. For example, if the ORG statement specifies a start location of 4096 (decimal), the first hexadecimal memory address will be 1000. Instructions and data are stored sequentially starting at the address specified in the Origin Control statement.
Each hexadecimal memory address is the address of the first byte of the instruction field. For example, the instructions at memory address 1055 of the illustrated program is a two-address Write and Count (WRC) instruction. All two-address instructions require six bytes of memory. Therefore, the instruction must occupy locations 1055 through 105A. The hexadecimal address of the next instruction would thus be 105B. Statement number 00380, a Halt and Branch instruction, has the address 1067. Since it is a single-address instruction, it must occupy locations 1067 through 1069. The address of the next sequential instruction will, therefore, be 106A.
The hexadecimal memory addresses of DA's and DC's are also the addresses of the leftmost bytes of the field. For example, statement number 00200 is a DC statement of length 15. Its hexadecimal memory address is 1000. Since the address 1000 is the address of the left-most byte in the field, the DC must occupy locations 1000 through 100D. The address of the next sequential DC must, therefore, be 100E.

NOTE

Recall that labels assigned to instructions refer to the leftmost byte of the three-or-six byte instruction field, while labels assigned to DA's and DC's refer to the right-most byte of the defined field. Recall also that all, except I/O, instructions must reference the rightmost byte of a field and all I/O instructions must reference the leftmost byte of a field.

3. **MACHINE LANGUAGE INSTRUCTIONS**
By means of the Assembler Program, Assembly Language instructions are translated into machine language instructions which ultimately become the loadable object program. The machine language (hex) equivalent is printed along side each symbolic program statement during the listing operation of Pass 2 of the Assembler. The programmer may take corrections to a program by altering the machine language instruction and entering the change into memory with the Control Panel or the Memory Manipulation and Capture Program.
The instruction in statement 00320 appears in the machine language as follows:



The programmer, may, for example, change the device number specified in the instruction by using the Control Panel to address location 105A and entering "BX" where "X" is the desired device number. Refer to the QANTEL Programmer Control Panel Operating Instructions.
4. **MACHINE LANGUAGE DATA**
The first seven bytes of all DC's are printed in the listing in USASCII code. (See Appendix B-USASCII Code). For example, statement number 00180 defines the alphabetic constant, "READ HEXT TEXT$^{C}_{T}$" length 14. The machine language data corresponds directly to the first seven bytes of the constant:

R	E	A	D		H	E
7 2	6 5	6 1	6 4	2 0	6 8	6 5

If the constant being defined is hexadecimal, the machine language data will appear exactly as it is in the DC statement; i.e. statement number 00280. No machine language coding appears for DA's.
5. **SYMBOLIC LANGUAGE INSTRUCTIONS**
The symbolic language instructions correspond exactly to the coding sheet format originally entered by the programmer at the time of the program creation.

APPENDIX F
PROGRAMMING TECHNIQUES

This appendix contains descriptions of programming methods used to circumvent certain problems which may arise in QANTEL programs.

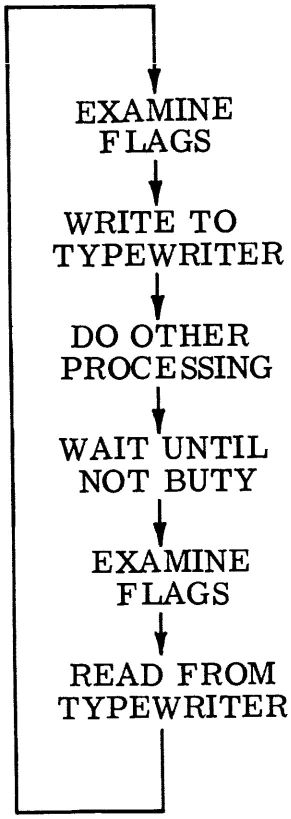
1. Use of the Set Read (SRD) Instruction.
   If a program issues a Set Read instruction to the typewriter and checks for service request (service by processor required) to determine when to issue the Read instruction, a loop will result if the operator presses the TERM (terminate) key without entering any data. To avoid this loop, the program must check for End (service by processor no longer required) if service request is not indicated. The "service by processor no longer required" indicates that no data was entered. A coding example using this method of checking status to avoid the looping is shown in the accompanying diagram.

2. Setting Flags in Programs with Typewriter Output
   In programs with typewriter output, a significant portion of the processing time is spent waiting for the typewriter to become not busy. Using normal programming techniques, it is not possible for the operator to set a flag during this waiting time and have it recognized by the program. (Flags are reset by the write instructions).
   To avoid this problem, each write instruction to the typewriter should be followed by a loop which waits for the typewriter to become not busy. This wait loop need not directly follow the write instruction if there is other processing that can be overlapped, but should precede any subsequent read or write instruction to the typewriter. The following diagram illustrates the proper sequence for examining typewriter flags.

PROGRAM | PROGRAMMER

LABEL 1 2 3 4 5 6 7 | 8 | OP-Code 9 10 11 12 | 13 | OPERANDS 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 | 34 | COMMENTS 35 36 37 38 39 40 41 42 43

```
*                   SET READ CODING
*
AØ          SRD
A1          SIN     $Ø8,Ø
            BNZ     A2      SERVICE REQUEST
            SIN     $Ø4,Ø
            BNZ     A3      END
            BRU     A1
A2          RD      INP,Ø
*                 CODING TO HANDLE DATA
            BRU     AØ
A3          NOP
*                 CODING FOR NO DATA ENTERED
            BRU     AØ
*
```



3. To Fill An Area in Memory with a Particular Character

The programmer may fill an area in memory by coding a DC and entering as many of the fill characters as required. This method is agreeable for small areas, but the filling of larger areas requires a short-cut. The following example shows how to fill a defined area with the desired character. In the example, the area is 25 memory locations in size and is called WRK, will the fill character is a space (hexadecimal 20).

	PROGRAM I.D.	PROGRAM LABEL 1 2 3 4 5 6 7	8	OP-Code 9 10 11 12	13	OPERANDS 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33	34
1		SPA		DC		$2Ø	
2		WRK		DA		25	
3				MŌV		SPA,1;WRK	
4				MŌV		WRK,24;WRK-1	
5							

QANTEL
CORPORATION
3474 Investment Blvd.
Hayward, Ca. 94544
415/783-3410