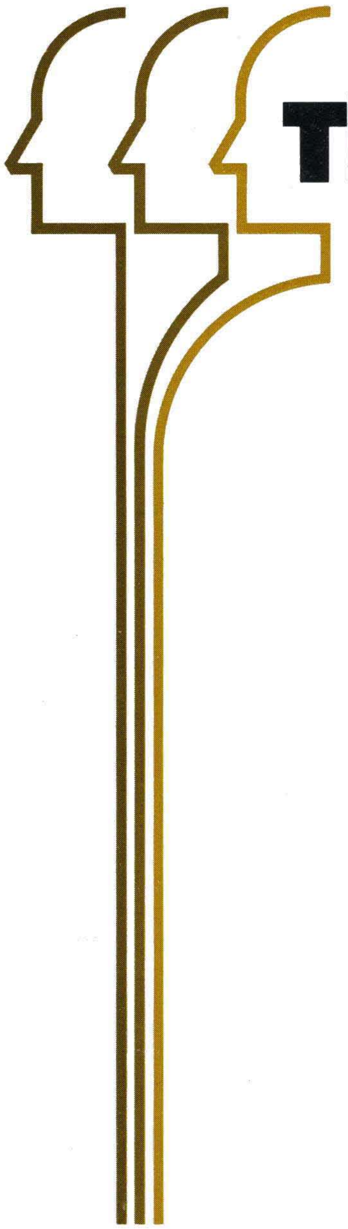


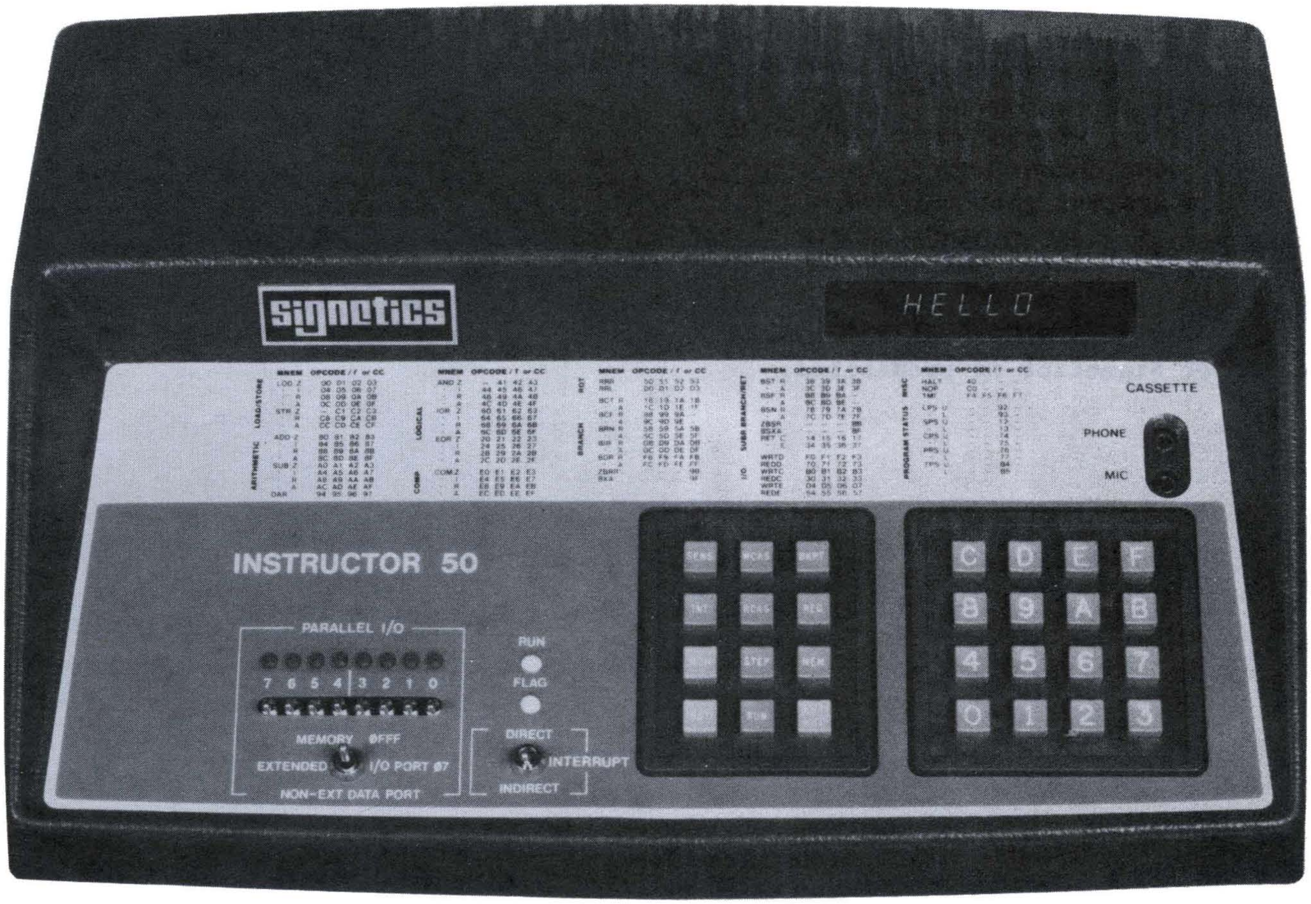
The Instructor 50™

Desktop Computer Users' Guide

signetics

a subsidiary of U.S. Philips Corporation

SIGNETICS INSTRUCTOR 50 USERS' GUIDE



signetics

a subsidiary of U.S. Philips Corporation

Signetics Corporation
P.O. Box 9052
811 East Arques Avenue
Sunnyvale, California 94086
Telephone 408/739-7700

Signetics reserves the right to make changes in the products contained in this manual in order to improve design or performance and to supply the best possible products. Signetics also assumes no responsibility for the use of any circuits described herein, conveys no license under any patent or other right, and makes no representations that the circuits are free from patent infringement. Reproduction of any portion hereof without the prior written consent of Signetics is prohibited.

PREFACE

This manual provides tutorial and reference information on the Signetics INSTRUCTOR 50--a complete, fully assembled and low cost microcomputer system. The INSTRUCTOR 50's computing power is enhanced by the Signetics 2650 microprocessor which is described in detail in Chapter 9.

INSTRUCTOR 50 is designed to assist you in learning programming and in writing, debugging, and testing the programs you develop. There is enough information here to get you started, whether or not you have ever written a program before. The only prerequisite is a familiarity with the 2650 microprocessor. Readers who are not familiar with the 2650's hardware structure and instruction set should read Chapter 9 prior to using the INSTRUCTOR 50.

The microprocessor has brought with it a host of terms which experienced users bandy back and forth with the greatest of ease. For the novice, this "language within a language" can be an obstacle of no small proportions. For the benefit of these people, Chapter 1 is devoted exclusively to microcomputer basics. To further assist you, we've put a glossary in the back to summarize some of the more frequently used buzz words.

CONTENTS

Page

CONTENTS (cont.)

CONTENTS (cont.)

CONTENTS (cont.)

LIST OF ILLUSTRATIONS

1. MICROCOMPUTER BASICS

This chapter introduces certain basic computer concepts. It provides background information and definitions which will be useful in later chapters of this manual. Those already familiar with computers may skip this material.

Before we begin, note that we are using two words: microprocessor and microcomputer. The microprocessor is a device which performs arithmetic, control, and logical operations. The microcomputer, in turn, is a collection of devices that includes a microprocessor, memory, and associated interface circuits to communicate with the "outside" world. Because it has its own microprocessor (the Signetics 2650), memory, latches, counters, buffers, power supply, an operator keyboard and display panel, and a cassette input/output interface, the INSTRUCTOR 50 is a complete and fully operational microcomputer system housed in one single package.

A MICRO DEFINED

Since the microprocessor is a miniaturized, coventional digital computer in integrated circuit (IC) form, a good place to start is with computers. Simply put, A computer is a device capable of automatically carrying out a sequence of operations on data expressed in descrete (digital) or continuous (analog) form. Its purpose is to solve a problem or class of problems; it may be one of control, analysis, or a combination of the two. In digital computers, numbers are represented by the presence of voltage levels or pulses on given lines. A single line defines one bit. A bit is the smallest unit of information in a binary system of notation. It is the choice between two possible states, usually designated one (1) and zero (0). A group of lines considered together is called a "word"; a word may represent a computational quantity (operand) or it may be an instruction specifying how the machine is to operate on computational quantities.

Word/Byte/Nibble

These terms are often misused in describing microprocessor data. For a specific microprocessor, a word is the number of bits associated with the instruction or data length. This can be 4, 8, 16 bits, etc., depending on the machine. A byte commonly refers to an 8-bit word; a byte can be manipulated by a 4, 8, or 16-bit microprocessor. For example, instructions are often provided to deal with byte data in 4 or 16-bit processors. This is called byte handling, and is independent of the natural word size of the machine.

A nibble is 4 bits, and it is rather humorous to consider that it takes two nibbles to make a byte. Nibble (or 4 bit) control can be found on many 8-bit word machines as well as on some 16-bit machines. Four-bit operations are usually associated with Hexadecimal (Hex) or Binary Coded Decimal (BCD) operations. Applications that have a man/machine interface, such as a control keyboard or a numeric display, are good candidates for nibble control.

Binary Notation

One of the problems in communicating with a computer is language. How does an electronic instrument handle and manipulate numbers? The answer is suggested by the nature of all electrical devices: a light bulb is either on or off, a switch is either open or closed, a magnet has a field in one direction or the opposite. For the purpose of understanding computer language, one can think of the "on" condition as being equal to 1 and the "off" state as 0. So the computer, which is made up of literally millions of electronic components, has two numbers it can work with. These numbers, 1 and 0, form all the elements needed in the binary system of notation.

In our more familiar decimal system, the right-hand column of a figure counts numbers up to 9; the column to the left of that registers the number of 10s; the column next to the left registers hundreds--then thousands, millions, and so on. In binary notation, the columns starting at the right register powers of 2 instead of 10. Take the binary number 10110, with successive powers of 2 noted above each column:

16	8	4	2	1
1	0	1	1	0

Adding together the powers of 2 turned "on" in this binary number--16, 4 and 2--we arrive at its decimal equivalent--22. The first eight decimal numbers translated into the binary system look like this:

1	=	1	5	=	101
2	=	10	6	=	110
3	=	11	7	=	111
4	=	100	8	=	1000

Hexadecimal Notation

To deal with large binary numbers, certain simplifications are extremely helpful. To this end, hexadecimal notation is often used. The term "hexadecimal", or hex for short, refers to a shorthand method of expressing a group of four consecutive binary bits by a single digit. Valid digits range from 0 through F, where F represents the highest decimal value (15). See Table 1.1.

Two hexadecimal digits can be used to specify a byte. Hexadecimal notation is very convenient for microprocessors since it gives good counting densities and works very well with the multiples-of-four binary words usually encountered in a microprocessor.

To understand hex notation, take a decimal number like 107_{10}. In binary notation, this becomes 1101011_2. Breaking this number into 4-bit nibbles (half-bytes), you get 0110_2 and 1011_2. The first and most-significant nibble is equal to 6_{16}, while the second and least-significant nibble is equal to B_{16}. Thus, in hexadecimal notation, 107_{10} becomes $6B_{16}$. One way to distinguish hexadecimal numbers from numbers written in other number systems (e.g., decimal, octal, etc.) is to enclose the hex number in single

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Table 1.1: Relationship among decimal, hexadecimal, and binary systems.

quotation marks and precede it by the letter H. Hence, in hex notation $6B_{16}$ would appear as H'6B'. To convert from decimal to hexadecimal, or vice versa, you must first convert the number into binary and then into hexadecimal as previously illustrated.

The INSTRUCTOR 50 uses the hexadecimal number system for entering values. Since the INSTRUCTOR 50 uses 8-bit bytes, two hexadecimal digits can be used to specify a byte. The smallest hexadecimal number is H'00' (00000000_2) and the largest is H'FF' (11111111_2). The INSTRUCTOR 50 still reads only binary numbers; hexadecimal is the user's shorthand, not the microcomputers.

Architecture

A microcomputer looks, architecturally, like any other computer (Figure 1.1). What distinguishes a micro from other computers is the intrinsic power inside each of the five functional boxes. What large-scale computers used to do, minis now handle. Similarly, micros have begun to supplant minicomputers in many applications. Advances in semiconductor technology have made this possible.

The four basic elements of all programmable computers emerge:

- Memory -- A storage unit. In modern computers, memories are implemented with semiconductor or magnetic core systems. Memories can be read only (ROM), for program or data constant storage, or read/write random access (RAM) for program, operand or temporary storage. Data is usually stored in binary notation. The memory is composed of storage space for a large number of words, with each storage space identified by a unique address. The word stored at a given address might be either computational data (operands) or an instruction (such as add, read from memory, etc.).

- Arithmetic & Logic Unit (ALU) -- Performs the arithmetic and/or logical operations on operands or provides partial results within the computer. The simplest ALU consists of a parallel adder and an accumulator. The adder adds (or performs similar logical operations, e.g., OR) two inputs, A and B, and produces the output. The accumulator holds intermediate results of a computation or numbers for a pending computation. The accumulator serves as a temporary storage device.

- Control unit -- Referred to as the brain of any computer because it coordinates all units of the computer in a timed, logical sequence. The control unit generates clock pulses to control and maintain the proper sequence of operations within the microprocessor. It also responds to external signals such as an interrupt request. In fixed-instruction computers, this unit receives instructions from the program memory. These instructions are in sequences, called programs. The control unit is closely synchronized to the memory cycle speed, and the execution time of each fixed instruction is often a multiple of the memory speed.

- Input/Output -- The means by which the computer communicates with a wide variety of devices, referred to as peripherals. They include audio cassette recorders, switches, indicator lamps, teletypewriters, CRT terminals, paper tape units, line printers, A/D or D/A converters, card readers and punches, communication modems, etc. The I/O lines can be connected to intermediate storage devices for use with mass memories, including magnetic discs and large-scale RAM systems.

Program Counter (Jumps, Subroutines, and the Stack)

The instructions that make up a program are stored in the system's memory. The central processor references the contents of memory in order to determine what action is appropriate. This means that the processor must know which location contains the next instruction.

Each of the locations in memory is numbered to distinguish it from all other locations in memory. The number which identifies a memory location is called its Address.

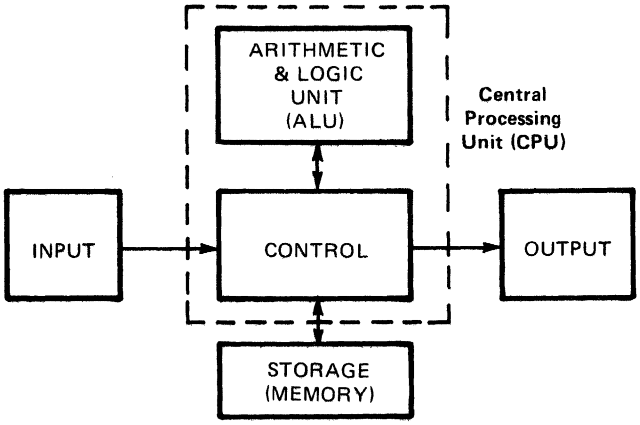


Figure 1.1

The processor maintains a counter which contains the address of the next program instruction. This is called a Program Counter (PC). The processor updates the program counter by adding to the counter each time it fetches an instruction, so that the program counter is always current (pointing to the next instruction). If an instruction takes several words in memory, the PC is incremented by the proper number so that it is always pointing to the first word of the next instruction.

The programmer therefore stores his instructions in numerically adjacent addresses, so that the lower addresses contain the first instructions to be executed and the higher addresses contain later instructions. The only time the programmer may violate this sequential rule is when an instruction in one section of memory is a jump instruction to another section of memory.

A jump instruction contains the address of the instruction which is to follow it. The next instruction may be stored in any memory location, as long as the programmed jump specifies the correct address. During execution of a jump instruction, the processor replaces the contents of its program counter with the address embodied in the jump. Thus, the logical continuity of the program is maintained.

A special kind of program jump occurs when the stored program calls a subroutine. In this kind of jump, the processor is required to "remember" the contents of the program counter at the time that the jump occurs. This enables the processor to resume execution of the main program when it is finished with the last instruction of the subroutine.

A subroutine is a program within a program. Usually it is a general-purpose set of instructions that must be executed repeatedly in the course of a main program. Routines which calculate the square, the sine, or the logarithm of a program variable are good examples of functions often written as subroutines. Other examples might be programs designed for inputting or outputting data to a particular peripheral device.

The processor has a special way of handling subroutines, in order to insure an orderly return to the main program. When the processor receives a call instruction, it increments the Program Counter and stores the counter's contents in a reserved memory area known as the stack. The stack thus saves the address of the instruction to be executed after the subroutine is completed. Then the processor loads the address specified in the call into its Program Counter. The next instruction fetched will therefore be the first step of the subroutine.

The last instruction in any subroutine is a return. Such an instruction need specify no address. When the processor fetches a return instruction, it simply replaces the current contents of the Program Counter with the address on the top of the stack. This causes the processor to resume execution of the calling program at the point immediately following the original call instruction.

Subroutines are often nested; that is, one subroutine will sometimes call a second subroutine. The second may call a third, and so on. This is perfectly acceptable, as long as the processor has enough stack capacity to store the necessary return addresses, and the logical provision for doing so. In other words, the maximum depth of nesting is determined by the depth of the stack itself. If the stack has space for storing three return addresses, then three levels of subroutines may be accommodated.

Processors have different ways of maintaining stacks. For example, some, like the Signetics 2650, have facilities for the storage and return addresses built into the processor itself. Other processors use a reserved area of external memory as the stack and simply maintain a pointer register which contains the address of the most recent stack entry. The external stack allows virtually unlimited subroutine nesting.

Instruction Register and Decoder

Each operation that the processor can perform is identified by a unique byte of data known as an Instruction Code or Operation Code. An eight-bit word used as an instruction code can distinguish between 256 alternative actions, more that adequate for most processors.

The processor fetches an instruction in two distinct operations. First, the processor transmits the address in its Program Counter to the memory. Then the memory returns tha addressed byte to the processor. The CPU stores this instruction byte in a register known as the Instruction Register, and uses it to direct activities during the remainder of the instruction execution.

An eight-bit instruction code is often sufficient to specify a particular processing action. There are times, however, when execution of the instruction requires more information than eight bits can convey.

One example of this is when the instruction references a memory location. The basic instruction code identifies the operation to be performed, but cannot specify the operand address as well. In a case like this, a twoor three-byte instruction must be used. Successive instruction bytes are stored in sequentially adjacent memory locations, and the processor performs two or three fetches in succession to obtain the full instruction. The first byte retrieved from memory is placed in the processor's instruction register, and subsequent bytes are placed in temporary storage; the processor then proceeds with the execution phase. Such an instruction is referred to as variable length.

Address Register(s)

A CPU may use a register pair to hold the address of a memory location that is to be accessed for data. If the address register is programmable (i.e., if there are instructions that allow the programmer to alter the contents of the register), the program can "build" an address in the address register prior to executing a memory reference instruction (i.e., an instruction that reads data from memory, writes data to memory or operates on data stored in memory).

Addressing Modes

An instruction word must convey the operation to be performed (operation code) and the address of the memory location or registers containing the data on which the operation is to be performed (operand). An n-bit instruction may be divided into three basic parts: 1) an operations code, 2) an address mode, and 3) an operand address. The number of bits in each of these parts varies from microprocessor to microprocessor.

The instruction length depends on the machine and the operation being performed. An 8-bit instruction format would allow only $2^8 = 256$ possible combinations of operations and addresses. This is obviously inadequate if a reasonable-size memory is to be accessed. For this reason 2 and 3-byte instructions are frequently used for memory access. Such an instruction is 16 or 24 bits long. In most cases, one byte is used to represent the operations code and address mode portions of an instruction. The number of bits used for each of these and their relative locations within the byte vary from processor to processor. The address mode and operand part of the instruction combine to indicate the location in which the operand is stored. There are numerous modes of addressing the operand. The most important for microprocessors include direct (or absolute), indirect, relative, indexed, and immediate addressing. The address mode portion of the instruction specifies how the address is to be interpreted. These addressing modes are defined as follows:

- Direct Addressing. With direct addressing, the address of the operand is specified directly in the instruction. This is a common form of addressing used in microcomputers. Direct addressing usually requires multiword instructions in 4 or 8-bit microprocessors.

- Indirect Addressing. In this mode, the instruction provides the address at which the address of the operand is to be found. In microprocessors, a form of addressing called register indirect addressing is commonly used. The address is stored in one or more registers within the CPU. In most cases, this architecture allows any location in memory to be addressed with a single-word instruction. Indirect addressing allows modification of the operand address during execution of the program.

- Relative Addressing. In relative addressing, the address is specified by its relation to the program counter. In this mode the address specified in the instruction is added to the number in the program counter to obtain the address of the operand. For example, if the address in the instruction is 11 and the program counter contains 124, then the address of the operand will be 11 + 124 = 135. The use of relative addressing simplifies the transfer of programs to different areas of memory.

Microcomputer memory is frequently structured into pages. A page may consist of 256 words of memory and is frequently located on a single IC. A page structure divides the memory into small blocks. The use of paging reduces the necessity for multiword memory reference instructions. In conjunction with a memory page structure, a form of relative addressing called page relative addressing is frequently used. In page relative addressing, an operand address given in the instruction is interpreted as a location on the same page of memory addressed by the program counter. In page-0 relative addressing, the operand address refers to a location on page 0 of the memory, regardless of the program counter contents.

- Indexed Addressing. This mode is similar to relative addressing. The address specified in the instruction, however, is relative to a prespecified register other than the program counter. This register is called the index. The address given in the instruction is added to the contents of the index register to determine the address of the operand. Indexed addressing is valuable in programs involving tables or arrays of numbers. The address of the first element of the table may be stored in the index register, and all other elements in the table may be addressed in relation to the first element.

- Immediate Addressing. In this mode, the operand is given in the instruction itself. In a microprocessor with only an 8-bit word length this may not be possible. In this case, the memory location immediately following the instruction is often used to store the immediate data.

The Signetics 2650 microprocessor can develop addresses in eight ways:

- Register addressing.
- Immediate addressing.
- Relative addressing.
- Relative, indirect addressing.
- Absolute addressing.
- Absolute, indirect addressing.
- Absolute, indexed addressing.
- Absolute, indirect, indexed addressing.

However, of these eight addressing modes, only four of them are basic. The others are variations due to indexing and indirection. Chapter 9 describes how effective addresses are developed by the 2650 microprocessor.

Extended/Non-Extended I/O

One of the major tasks performed by the CPU portion of a microcomputer is the transfer of data between the CPU and an I/O device. This, of course, is the method used by the computer to communicate with the outside world; e.g., reading data into the processor from a keyboard, cassette tape unit, paper tape reader, etc. or writing data into a CRT display, paper tape punch, cassette recorder, etc.

In most microprocessor-based system, there is essentially only one way that these I/O data transfers take place; i.e., by placing the "address" or identification code of a specific I/O device on the address bus and the data to be written on the data bus. (If its a read operation, the I/O device will place the data to be read on the data bus.) With this arrangement, some mechanism must be provided to examine the address bus during an I/O transfer to determine which specific I/O device is being accessed. This operation requires some type of decoder which can look at up to 8-bits of address data and from this information, generate a signal on a single line which will open a path from the data bus to the individual I/O device specified by the data on the address bus.

This can be a rather complex task and, in fact, is often implemented by a special LSI chip designated specifically for this purpose. In addition to the hardware required, this approach to I/O data transfer also consumes memory space for storing this I/O address. For example, the 2650 requires two eight-bit memory words to implement this type of I/O transfer. One word specifies the operation (Read or Write) and the other specifies the I/O device. In the 2650, this is referred to as an Extended I/O operation.

In addition to the Extended Mode of parallel I/O data transfer, the 2650 can also operate in what is referred to as a Non-Extended mode. In this mode, two different I/O devices can be addressed by a single pin called Data/Control (D/C). This is an output from the 2650 that responds to a specific instruction calling for a Non-Extended I/O operation. This pin and the memory mapped I/O (see Chapter 6) are the only two pins that need be decoded to use this simple form of I/O. When the D/C output is high, it connects the "D" output device to the data bus; when it is low, it connects the "C" output device. Thus, simple SSI gates are the only interface required to enable the 2650 to communicate with I/O devices in the Non-Extended Mode.

In addition to saving hardware, the Non-Extended I/O mode also saves software (or program memory). Each Non-Extended instruction is a single word instruction which contains enough information to specify two different operations (Read or Write) to two different ports (D or C).

One additional benefit in having both Extended and Non-Extended I/O modes is the fact that one can "mix" modes in any given system. For example, assume that a typical system has 20 I/O channels, two of which are used substantially more that the other eighteen. In this system, one could specify the two frequently used channels as Non-Extended channels and address these with single-byte instructions. The other, less frequently used channels would be addressed with Extended instructions.

Another example would be in those situations where a single I/O device has two separate ports for information flow. Quite often, one of these ports is used to handle Control or status information; for example, "start a motor" or "start the timer," etc. The other channel is used for the actual data transfer. In this case, the basic I/O device can be addressed in the Extended Mode with a two-byte instruction and the actual information transferred in a Non-Extended Mode with a single-byte instruction. (In fact, the Data/Control aspects of this dual-port situation is what prompted the nomenclature for the D/C pin.)

SOFTWARE

Software is a term used to describe the programs that make a computer do a specific task. In fact, when used in the context of computers, the word software can be interchanged with the word program. In general, a program is a series of sequential steps (instructions) that accomplish an objective. Even though the specific set of instructions it can use is fixed by its design, a computer is general purpose because it can execute a list of these instructions (a program) to perform some functions, execute another list of instructions to perform some other function, and so on.

In discussions about software and programming, a great deal is often said about programming in some language or another. This is because the way we command the machine is very much like the way we communicate in a written language. We have rules about how we start and end sentences and paragraphs and how we spell words. The way we communicate with a computer is through a programming language, which also has rules of spelling and punctuation, but these rules are much more strictly enforced. If you misspell a few words, your reader will probably understand you anyway. A computer language is not that forgiving and will not produce the desired result if its rules are broken.

Machine Language

There are a number of levels of programming languages. The most basic level is that of the actual machine language. Each instruction is uniquely defined by a binary code (pattern) of ones and zeros. The central processing unit (CPU) examines each instruction code and performs the exact sequence of events to produce the operation defined by that instruction. After an operation has been performed and a problem solved, the computer must then reverse its opening procedure. It must retranslate its machine language and display the answer in a form the person who presented the problem can understand.

The use of machine language is a perfectly reasonable way to program when the application is not too complex and the effort is on a low budget. The INSTRUCTOR 50 is a machine-language microcomputer; making it support assembly language would have considerably raised its cost. The main advantages of machine language programming are that it can be completed without the aid of another program, and it allows the programmer to keep track of and control every detail of the machine operation.

Assembly Language

To make programming easier, assemblers have been developed. An assembler is a computer program that accepts coded instructions or mnemonics that are more meaningful to use and translates them into binary machine code for execution by a computer. The mnemonics used for each instruction are much easier to remember, and they make a listing of the program much easier to read. Assembly language programming allows the programmer to retain complete control over the important details of the computer operation, but takes care of all the drudgery of the binary coding, address calculations, and the like.

Higher-Level Languages

A third category of software is the higher-level languages, such as BASIC and FORTRAN, which come the closest to natural human languages. They are problem-oriented and contain familiar words and expressions; however, they have a very strictly defined structure and syntax. There are two types of support programs associated with higher-level languages: compilers and interpreters. Both types take the higher-level language program the programmer writes and turn it into machine language the computer can use.

Other Software

Other software associated with microprocessors include monitor programs, debug programs, simulators, editors, I/O handlers, diagnostic programs, and loaders. Brief definitions of these programs are provided in the glossary (Chapter 13).

2. GETTING STARTED

Introduction

Welcome aboard the INSTRUCTOR 50--a unique and powerful training tool designed to introduce you to the world of microcomputers in the shortest possible time.

INSTRUCTOR 50 is for computer hobbyists, students, engineers or anyone who wants to learn how to use a microcomputer the easy way, without having to face the drudgery of a long and tedious training program.

INSTRUCTOR 50 is a stand-alone microcomputer based on the Signetics 2650 microprocessor. It includes everything that you need to write, run, and debug machine-language programs. A 12-key Function Control Keyboard and a 16-key Hexadecimal Keyboard are used to enter data and perform various system functions associated with the INSTRUCTOR 50. The INSTRUCTOR 50 User System Executive (USE) monitor program guides you in the use of the system by displaying prompting messages and responses on an eight-digit LED display. All facilities required for program development are built into INSTRUCTOR 50 -- you don't need anything else to start.

Before getting into the details of what makes the INSTRUCTOR 50 tick, let's first take a short shakedown cruise and write a few simple programs. Detailed information on each 2650 instruction is provided in Chapter 9.

Power On and Initial Display

To apply power to the INSTRUCTOR 50, connect the power cord into the rear panel receptacle, and insert the power pack into any standard 115 VAC domestic wall socket. The INSTRUCTOR 50 does not have a power ON/OFF switch. The initial display is the message HELLO, indicating that the INSTRUCTOR 50 is in the monitor mode and ready for use. If the HELLO message does not appear, depress the MON key to initialize the INSTRUCTOR 50. Unplug the power pack to turn the INSTRUCTOR 50 off.

Operating Modes

The INSTRUCTOR 50 has two basic modes of operation, the MONITOR mode and the EXECUTION mode. The MONITOR mode is entered automatically on power up or by depressing the MON key on the function control keyboard. The monitor responds by displaying HELLO. While in the MONITOR mode, you may:

- Enter and alter a program.
- Read in a previously saved program from audio cassette tape.

- Display and alter the contents of the microcomputer's general-purpose working registers and/or Program Status Word (PSW).
- Examine and alter the contents of memory locations.
- Examine and alter the contents of the Program Counter.
- Specify and examine a program breakpoint.
- Step through a program one instruction at a time.
- Save a program on cassette tape.

The EXECUTION mode is entered by depressing the RUN key, the STEP key, or the RESET (RST) key on the function control keyboard. Depressing the RUN key terminates the MONITOR mode and causes program execution to begin at the address specified in the Program Counter. Depressing the STEP key causes the INSTRUCTOR 50 to execute a single instruction and return to the MONITOR mode. When the RST key is depressed, current INSTRUCTOR 50 activity is terminated, and the processor begins program execution at address zero or, in hex notation, H'0000'.

Keying in and Entering Values

Address and data parameters are entered into the INSTRUCTOR 50 via the hexadecimal keyboard using the hex notation described in Chapter 1. When entering an address, you may enter as many as four hex digits starting with the most-significant digit of the address. Leading zeroes need not be entered; if less that four digits are entered, the leading digits are automatically zeroed. Data values consist of one or two hex digits, with the most-significant digit entered first. If only one digit is entered, the most-significant digit is automatically zeroed.

Correcting Entry Errors

The numbers keyed in appear in the address or data display field and can be edited prior to depression of a funciton key by simply keying in the correct characters. The display shifts to the left each time a new character is entered, and characters shifted out of the field are disregarded. Only the last digits entered are retained, so that an error in entry can be corrected by entering the correct data.*

For example, if you were entering an address and you depressed 121 instead of the correct value of 120, the display would read:

```
.Ad. = 121
```

* Data values entered during operation in the FAST PATCH command mode cannot be corrected in this manner. See description of the FAST PATCH command in Chapter 5.

To recover from this error, simply key in the correct value by depressing the following hex keys:

(0) (1) (2) (0)

The correct value would then be displayed as indicated below.

.Ad. = 0120

The Prompt Light

A dot or period in the left-most position of the display (e.g., .Ad. =) is a prompt signal. It indicates that the INSTRUCTOR 50 is ready to accept a data or address value.

Entering and Executing a Simple Program

To demonstrate the use of the INSTRUCTOR 50, let's write a simple program, enter it, and execute it. Prior to writing the program, we must decide what task or operation we want the program to perform.

Let's say we want to "show the operation of an 8-bit binary counter on the INSTRUCTOR 50's output port indicator LEDs". The flowchart for performing this task is shown in Figure 2.1.

The DELAY block shown in the flowchart provides a time interval between new values of the binary count in order to observe the counting action on the port indicators. This can be implemented in several ways, depending on the delay required.* We will use a double-loop technique, with the outer loop counting the number of excursions through the inner loop.

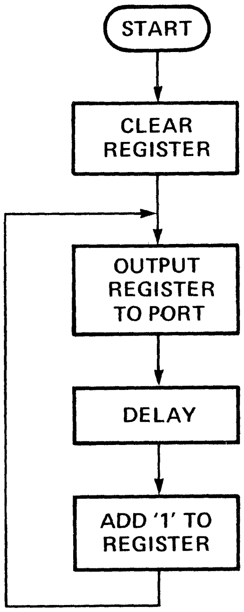


Figure 2.1: Flowchart for Binary Counter Program

*See Signetics 2650 Applications Note AS52 - General Delay Routines.

The next step is to select registers for the binary counter and the delay loop counters, and to select an output port for the display operation. Let's arbitrarily make the following assignments:

Register 0 = Binary counter
Register 1 = Outer loop counter
Register 2 = Inner loop counter
Port D = Output display port

We are now ready to write the program:

ADDRESS	HEX VALUE	LABEL	INSTRUCTION	COMMENTS
00	75,11	START	CPSL C + RS	Operations without Carry, Reg. bank 0
02	20		EORZ,R0	Clear R0
03	F0	OUT	WRTD,R0	Output R0 to D
04	05,20	LOOP1	LODI,R1 H'20'	Initialize outer loop
06	06,20	LOOP2	LODI,R2 H'00'	Initialize inner loop
08	FA,7E	SELF	BDRR,R2 SELF	Count inner loop
0A	F9,7A		BDRR,R1 LOOP2	Count outer loop
0C	84,01		ADDI,R0 H'01'	Add 1 to R0
0E	1F,00,03		BCTA,UN OUT	Go back to output

Let's begin entering the program using the INSTRUCTOR 50's FAST PATCH command, which is used for entering long hex data strings. The FAST PATCH mode is enabled by depressing the (REG) key followed by the (F) key:

KEY	DISPLAY	COMMENTS
(MON)	HELLO	Enter monitor mode
(REG) (F)	.Ad. =	Enter FAST PATCH
(0) (ENT/NXT)	.0000	Enter starting address
(7) (5)	.0000 75	Begin program entry.
(1) (1)	.0001 11	
(2) (0)	.0002 20	
•		
•		
•		
(1) (F)	.000E 1F	
(0) (0)	.000F 00	
(0) (3)	.0010 03	
(ENT/NXT)	.0010 03	Terminate FAST PATCH

We will now verify correct entry by using the DISPLAY & ALTER MEMORY command:

KEY	DISPLAY	COMMENTS
(MEM)	.Ad. =	Display and Alter memory
(0) (ENT/NXT)	.0000 75	Address entered, data displayed
(ENT/NXT)	.0001 11	
(ENT/NXT)	.0002 20	
(ENT/NXT)	.0003 F0	
•		
•		
•		
(ENT/NXT)	.0010 03	Verification complete

ERROR CORRECTION TECHNIQUE

If an error is detected during verification, it can be corrected by entering the correct value before depressing the (ENT/NXT) key. For example:

KEY	DISPLAY	COMMENTS
•		
•		
(ENT/NXT)	.0003 F8	Error. Data should be F0.
(F) (0)	.0003 F0	Correct data entered.
(ENT/NXT)	.0004 05	New data deposited.
•		
•		

EXERCISING THE PROGRAM

We are now ready to exercise the program. Before proceeding, make certain that the Interrupt Select Switch which is accessible from the bottom side of the case is in the keyboard position (towards the center). The Port Address Select Switch is placed in the NON-EXTENDED Port D position, and, since the program begins at address zero, the (RST) key is depressed to initiate execution. The program operation can be observed on the I/O port indicators.

CHANGING THE PROGRAM PARAMETERS

We can use the INSTRUCTOR 50 facilities to change the program parameters or to observe the internal operation of the program. For example, to change the delay time, we can change the delay constant at address H'05' with the DISPLAY AND ALTER MEMORY command.

KEY	DISPLAY	COMMENTS
(MON)	HELLO	Return to monitor mode.
(MEM)	.Ad. =	Display and Alter memory.
(5) (ENT/NXT)	.0005 20	Address entered, data shown.
(4) (0)	.0005 40	New constant entered.
(ENT/NXT)	.0006 06	New constant deposited.
(RST)		Program re-started.

The counter now operates about half as fast as before.

We can observe the internal operation of the program by using a breakpoint, which will stop program execution at a selected instruction and return to the monitor mode. Let's watch the outer delay loop operate by placing a breakpoint at address H'0A'. To enable the breakpoint during program execution, the program must be started via the RUN command. Before running the program, the starting address (H'00') must be entered by using the DISPLAY AND ALTER PROGRAM COUNTER (PC) command:

KEY	DISPLAY	COMMENTS
(MON)	HELLO	Return to monitor.
(BKPT) (A)	b.P. = A	Breakpoint entered.
(ENT/NXT)		
(REG) (C) (0)	.PC = 0	Enter starting address.
(RUN)	-000A F9	Start execution. Program stops at breakpoint and returns to monitor.
(REG) (1)	.rl = 3F	Rl has decremented by 1.
(RUN)	-000A F9	Execute again.
(REG) (1)	.rl = 3E	Rl has decremented again.
(RUN) (REG) (1)	.rl = 3D	And again.
(BKPT) (BKPT)	b.P =	Breakpoint removed.
(RUN)		Program runs without stopping.

EXAMPLE 2: THE BILLBOARD PROGRAM

Example 2 is a program that makes use of the User Display Routine described in Chapter 6. The User Display Routine moves an eight-byte message from a user program to the display buffer and then displays the message. In our sample program, the selected message will reappear on the display panel at regular intervals to give the effect of a rotating billboard.

The following program listing is self-explanatory and contains all the necessary parameters for entering and executing the program. If you are not familiar with program listings, the hex values are located in the third column from the left under the word OBJECT. Figure 2.2 is a flowchart of the billboard program.

```
LINE ADDR  OBJECT  E SOURCE

0002                 *********************************************************************
0003                 *
0004                 *PROGRAM WRITTEN BY JOHN KEENAN
0005                 *
0006                 *THIS PROGRAM IS WRITTEN FOR THE INSTRUCTOR 50
0007                 *
0008                 *THIS PROGRAM DISPLAYS THE MESSAGE IN THE DISPLAY BUFFER
0009                 *
0010                 *THE MESSAGE WILL WILL REAPPEAR ON THE DISPLAY PANEL
0011                 *AT REGULAR INTERVALS TO GIVE THE EFFECT OF A ROTATING
0012                 *BILLBOARD.
0013                 *
0014                 *THE MAXIMUM MESSAGE LENGTH IS 254 CHARACTERS
0015                 *THE MESSAGE IS ENTERED STARTING AT LOCATION H'101'. PROGRAM LABEL 'MSG'
0016                 *THE END OF MESSAGE IS INDICATED BY THE VALUE OF H'FF' AS THE LAST
0017                 *CHARACTER OF THE MESSAGE.
0018                 ***********************************************************************
0019                 * STANDARD SYMBOL DEFINITION - THIS FILE MAY BE APPENDED TO THE
0020                 *                              FRONT OF ANY USER'S SOURCE DECK
0021                 *   REGISTER EQUATES
0022 0000            R0      EQU     0       REGISTER 0
0023 0001            R1      EQU     1       REGISTER 1
0024 0002            R2      EQU     2       REGISTER 2
0025 0003            R3      EQU     3       REGISTER 3
0026                 *  CONDITION CODES
0027 0001            P       EQU     1       POSITIVE RESULT
0028 0000            Z       EQU     0       ZERO RESULT
0029 0002            N       EQU     2       NEGATIVE RESULT
0030 0002            LT      EQU     2       LESS THAN
0031 0000            EQ      EQU     0       EQUAL TO
0032 0001            GT      EQU     1       GREATER THAN
0033 0003            UN      EQU     3       UNCONDITIONAL
0034                 *  PSW LOWER EQUATES
0035 0000            CC      EQU     H'00'   CONDITIONAL CODES
0036 0020            IDC     EQU     H'20'   INTERDIGIT CARRY
0037 0010            RS      EQU     H'10'   REGISTER BANK
0038 0008            WC      EQU     H'08'   1=WITH 0=WITHOUT CARRY
0039 0004            OVF     EQU     H'04'   OVERFLOW
0040 0002            COM     EQU     H'02'   1=LOGIC 0=ARITHMETIC COMPARE
0041 0001            C       EQU     H'01'   CARRY/BORROW
0042                 *  PSW UPPER EQUATES
0043 0080            SENS    EQU     H'80'   SENSE BIT
0044 0040            FLAG    EQU     H'40'   FLAG BIT
0045 0020            II      EQU     H'20'   INTERRUPT INHIBIT
0046 0007            SP      EQU     H'07'   STACK POINTER
0047                 * END OF EQUATES
0048                 ***********************************************************************
```

```
LINE ADDR  OBJECT  E SOURCE

0050 0000                   ORG     0       SET THE BEGINNING OF PROGRAM TO LOCATION 0
0051                 *
0052 0000 7510       BEGIN  CPSL    RS      SET LOWER REGISTER BANK
0053 0002 0560       START  LODI,R1 H'60'   LOAD THE DELAY COUNTER
0054 0004 7710       DISPL  PPSL    RS      SET THE UPPER REGISTER BANK
0055 0006 0501              LODI,R1 <MSG    LOAD THE UPPER BYTE OF MESSAGE POINTER ADDRESS
0056 0008 0E0100            LODA,R2 PNTR    LOAD THE LOWER BYTE OF MESSAGE POINTER ADDRESS-1
0057 000B 0701              LODI,R3 01      LOAD THE 1 PASS COMMAND PARAMETER TO THE DISPLAY
0058                 *                      ROUTINE
0059 000D BBE6              ZBSR    *USRDSP EXIT TO THE DISPLAY ROUTINE FOR 1 PASS
0060 000F 7510              CPSL    RS      SELECT THE LOWER REGISTER BANK
0061 0011 FD0004            BDRA,R1 DISPL   DECREMENT THE DELAY COUNTER AND CHECK FOR END
0062                 *                      IF NOT AT END, DISPLAY THE SAME MESSAGE UNTIL
0063                 *                      COUNT = 0
0064 0014 0E0100            LODA,R2 PNTR    LOAD POINTER TO MESSAGE
0065 0017 8601              ADDI,R2 1       INCREMENT IT
0066 0019 CE0100            STRA,R2 PNTR    SAVE THE NEW POINTER VALUE
0067 001C 0E6109            LODA,R0 MSG+8,R2        LOAD THE NEXT CHARACER TO BE DISPLAYED
0068 001F E4FF              COMI,R0 H'FF'   IS IT THE END OF MESSAGE CHARACTER?
0069 0021 9C0002            BCFA,EQ START   IF NO, GO DISPLAY THE MESSAGE ROTATED LEFT 1
0070                 *                      CHARACTER
0071 0024 0400              LODI,R0 0       IF YES,  RESET MESSAGE POINTER TO BEGINNING OF
0072                 *                      MESSAGE
0073 0026 CC0100            STRA,R0 PNTR    SET POINTER TO BEGINNING
0074 0029 1F0002            BCTA,UN START   GO DISPLAY THE MESSAGE FROM THE BEGINNING
0075                 *
0076                 *
0077 002C                   ORG     H'100'  THIS IS THE DATA AREA FOR THE PROGRAM
0078                 *
0079 0100            PNTR   RES     1       1 LOCATION TO SAVE THE LEAST SIGNIFICANT BYTE OF
0080                 *                      MESSAGE POINTER
0081                 *
0082                 *
0083                 *THIS IS THE INITIAL MESSAGE IN THE BUFFER
0084                 *
0085                 *THE MESSAGE IS 'HI THIS IS THE 2650....HI THIS'
0086                 *
0087                 *
0088 0101 14011707   MSG    DATA    H'14,01,17,07,14,01,05,17,01,05,17' HI THIS IS
     0105 14010517
     0109 010517
0089 010C 07140E17          DATA    H'07,14,0E,17,02,06,05,00'          THE 2650
     0110 02060500
0090 0114 1A1A1A1A          DATA    H'1A,1A,1A,1A,14,01,17,07,14,01,05' ....HI THIS
     0118 14011707
     011C 140105
0091 011F FF                DATA    H'FF'                               END OF MESSAGE FLAG
0092                 *
0093                 *
0094 0120                   ORG     H'1FE6' LOCATION OF POINTER IN MONITOR TO DISPLAY ROUTINE
0095 1FE6            USRDSP EQU     $
0096                 *
0097                 *
0098 0000                   END     BEGIN

 TOTAL ASSEMBLY ERRORS = 0000
```

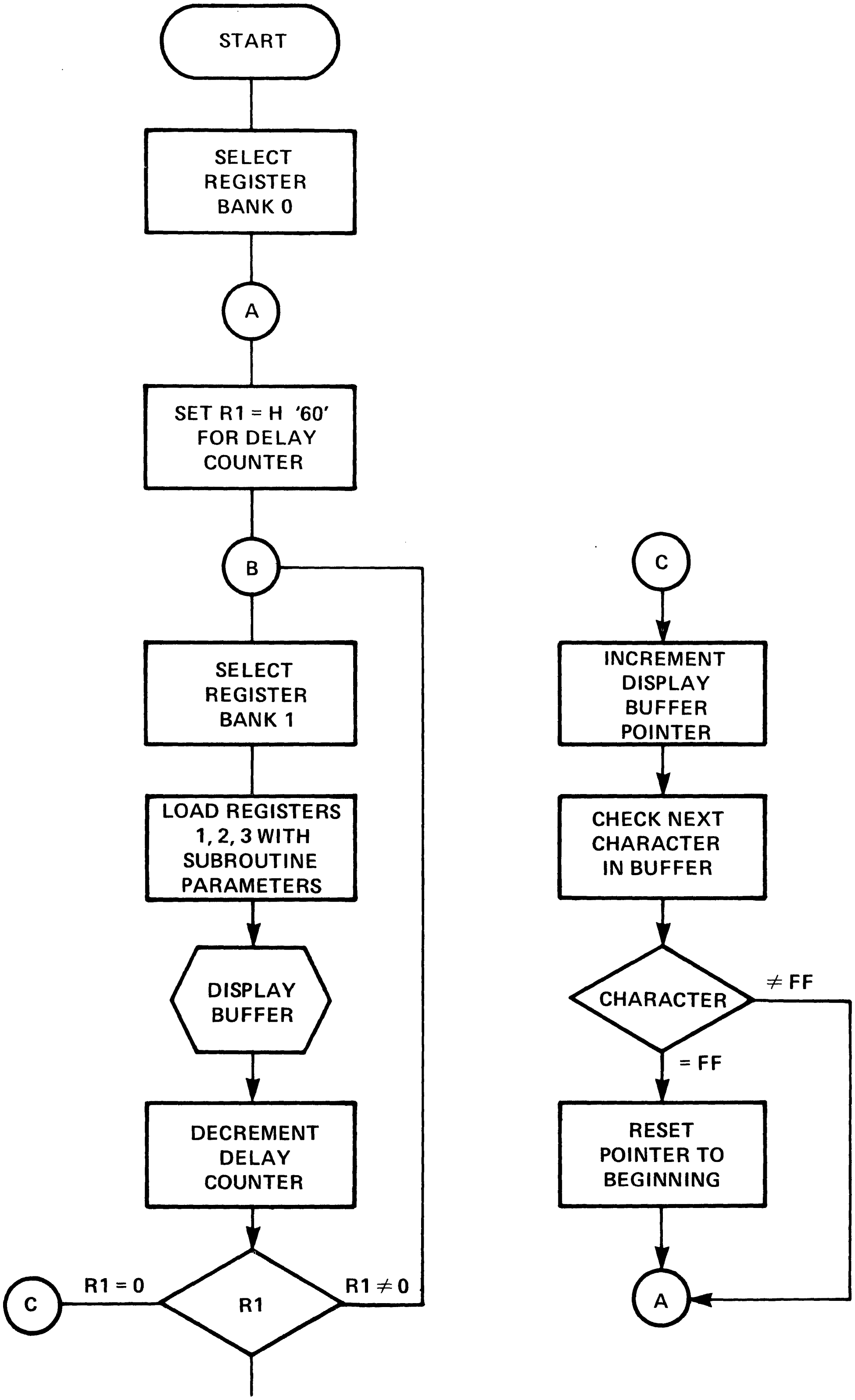

START
SELECT REGISTER BANK 0
A
SET R1 = H '60' FOR DELAY COUNTER
B
SELECT REGISTER BANK 1
LOAD REGISTERS 1, 2, 3 WITH SUBROUTINE PARAMETERS
DISPLAY BUFFER
DECREMENT DELAY COUNTER
R1
R1 = 0
R1 ≠ 0
C
C
INCREMENT DISPLAY BUFFER POINTER
CHECK NEXT CHARACTER IN BUFFER
CHARACTER
≠ FF
= FF
RESET POINTER TO BEGINNING
A

Figure 2.2: Flowchart for Billboard Program.

Let's begin entering the hex values shown in the program listing starting at memory location H'0000'. We will again use the FAST PATCH command for entering values.

Program Entry & Verification

KEY(S)	DISPLAY	COMMENTS
(MON)	HELLO	Enter monitor mode
(REG) (F)	.Ad. =	Enter FAST PATCH
(0) (ENT/NXT)	.0000	Enter starting address.
(7) (5)	.0000 75	Begin program entry.
(1) (0)	.0001 10	
(0) (5)	.0002 05	
(6) (0)	.0003 60	
•		
•		
•		
(1) (F)	.0029 1F	
(0) (0)	.002A 00	
(0) (2)	.002b. 02	
(ENT/NXT)	002b. 02	Terminate FAST PATCH.

We will now verify correct entry by using the DISPLAY & ALTER MEMORY command:

KEY(S)	DISPLAY	COMMENTS
(MEM)	.Ad. =	Display and Alter Memory
(0) (ENT/NXT)	.0000 75	Address entered; data displayed.
(ENT/NXT)	.0001 10	
(ENT/NXT)	.0002 05	
(ENT/NXT)	.0003 60	
•		
•		
•		
(ENT/NXT)	.0029 1F	
(ENT/NXT)	.002A 00	
(ENT/NXT)	.002b. 02	Verification complete.

Setting a Pointer

Our next step will be to set a message pointer at memory location 100 to indicate that our message will begin at address 101.

KEY(S)	DISPLAY	COMMENTS
(MEN)	.Ad. =	Display and Alter Memory
(1) (0) (0)	.Ad. = 100	Location of message pointer address entered.
(ENT/NXT)	.0100 1A	Previous contents of memory location 100 is 1A.
(0) (0)	.0100 00	Contents changed to 00.
(ENT/NXT)	.0101 00	Message pointer set.

Entering a Message

Now that the message pointer has been entered and set, we can begin entering our message starting at memory location 101. We will re-enter the FAST PATCH mode prior to message entry and then begin entering the message: HI THIS IS THE 2650....HI THIS. Note that the first two words of our message are repeated to give the effect of a rotating billboard.

Refer to Figure 3.2 for the hex value corresponding to each character in our message.

KEY(S)	DISPLAY	COMMENTS
(REG) (F)	.Ad. =	Enter FAST PATCH
(1) (0) (1)	.Ad. = 101	Starting address entered.
(ENT/NXT)	.0101	Starting address set.
(1) (4)	.0101 14	Hex value for letter H
(0) (1)	.0102 01	Hex value for letter I
(1) (7)	.0103 17	Hex value for blank or space
(0) (7)	.0104 07	Hex value for letter T
(1) (4)	.0104 14	Hex value for letter H
(0) (1)	.0106 01	Hex value for letter I
(0) (5)	.0107 05	Hex value for letter S
(1) (7)	.0108 17	Hex value for blank or space
(0) (1)	.0109 01	Hex value for letter I
(0) (5)	.010A 05	Hex value for letter S
(1) (7)	.010b. 17	Hex value for blank or space
(0) (7)	.010C 07	Hex value for letter T
(1) (4)	.010d. 14	Hex value for letter H
(0) (E)	.010E 0E	Hex value for letter E
(1) (7)	.010F 17	Hex value for blank or space
(0) (2)	.0110 02	Hex value for numeral 2
(0) (6)	.0111 06	Hex vlaue for numeral 6
(0) (5)	.0112 05	Hex value for numeral 5
(0) (0)	.0113 00	Hex value for numeral 0
(1) (A)	.0114 1A	Hex value for dot or period
(1) (A)	.0115 1A	Hex value for dot or period
(1) (A)	.0116 1A	Hex value for dot or period
(1) (A)	.0117 1A	Hex value for dot or period
(1) (4)	.0118 14	Hex value for letter H
(0) (1)	.0119 01	Hex value for letter I
(1) (7)	.011A 17	Hex value for blank or space
(0) (7)	.011b. 07	Hex value for letter T
(1) (4)	.011C 14	Hex value for letter H
(0) (1)	.011d. 01	Hex value for letter I
(0) (5)	.011E 05	Hex value for letter S
(F) (F)	.011F FF	End of message flag.

Exercising the Program

To execute the program, depress the following keys:

(REG) (C) (0) (RUN)

You can now observe the movement of our message on the display panel. We can use the INSTRUCTOR 50 facilities to vary the speed of message movement by changing the constant at memory location 0003 from 60 to another value. To accomplish this, proceed as follows:

KEY(S)	DISPLAY		COMMENTS
(MON)	HELLO		Return to monitor mode.
(MEM)	.Ad. =		Display and alter memory.
(3) (ENT/NXT)	.0003	60	Address entered; data shown.
(F) (F)	.0003	FF	New constant entered.
(ENT/NXT)	.0004	77	New constant deposited.
(REG) (C) (0)			Program re-started.
(RUN)			

EXAMPLE 3: CLOCK PROGRAM FOR THE INSTRUCTOR 50

Example 3 is a clock program that makes use of the Display routine described in Chapter 6. Since this program incorporates features described in later chapters of this manual, you may wish to skip it for now and return to it when you are more familiar with the INSTRUCTOR 50.

The following program listing is self-explanatory and contains all the necessary parameters for entering and executing the program. As in the previous example, the hex values in the program listing are located in the third column from the left under the word OBJECT.

NOTE: When entering the hex values from the listing, note the gaps at address locations 0005 and 0006. These gaps are used to accommodate an interrupt routine. No Operation (NOP) instructions (e.g., C0) may be inserted in these gaps.

Let's begin entering the program using the program listing as a guide:

KEY(S)	DISPLAY		COMMENTS
(MON)	HELLO		Enter monitor mode.
(REG) (F)	.Ad. =		Enter FAST PATCH
(0) (ENT/NXT)	.0000		Enter starting address
(7) (6)	.0000	76	Begin program entry.
(2) (0)	.0001	20	
(1) (F)	.0002	1F	
(0) (0)	.0003	00	
(9) (5)	.0004	95	
(C) (0)	.0005	C0	NOP entered in gap.
(C) (0)	.0006	C0	NOP entered in gap.
(0) (C)	.0007	0C	
•			
•			
•			
(1) (B)	.00A5	1b.	
(6) (E)	.00A6	6E	
(ENT/NXT)	.00A6	6E	Terminate FAST PATCH

Now that we have entered the program, let's enter the time of day, execute the program, and observe the clock on the display panel. For demonstration purposes, we will use the time of day specified on page 1 of the program listing; that is : 3:45:27 AM.

KEY(S)	DISPLAY	COMMENTS
(REG) (F)	.Ad. =	Enter FAST PATCH
(1) (0)		.0100 Enter starting
(0) (ENT/NXT)		address.
(1) (7)	.0100 17	
(8) (3)	.0101 83	
(0) (4)	.0102 04	
(8) (5)	.0103 85	
(0) (2)	.0104 02	
(0) (7)	.0105 07	
(1) (7)	.0106 17	
(0) (A)	.0107 0A	
(ENT/NXT)		Terminate FAST PATCH

Depress (RST) and observe the clock on the display panel.

```
LINE ADDR  OBJECT  E SOURCE

0002                 ********************************************************************
0003                 *
0004                 * CLOCK PROGRAM FOR INSTRUCTOR 50
0005                 *
0006                 * THIS PROGRAM IMPLEMENTS A CLOCK ON THE INSTRUCTOR 50.
0007                 * TO RUN THE PROGRAM, THE 'DIRECT/INDIRECT' SWITCH MUST
0008                 * BE IN THE 'DIRECT' POSITION AND THE 'INTERRUPT SELECT'
0009                 * SWITCH MUST BE IN THE 'AC LINE' POSITION.
0010                 *
0011                 * THE DISPLAY FORMAT IS AS FOLLOWS:
0012                 *              HH.MM.SS A/P
0013                 * AFTER ENTERING THE PROGRAM INTO THE INSTRUCTOR 50,
0014                 * THE INITIAL TIME MUST BE ENTERED INTO LOCATIONS H'100'
0015                 * TO H'107' USING THE EXAMINE/ALTER MEMORY COMMAND OR THE
0016                 * FAST PATCH COMMAND.  THE DATA ENTERED IN THESE LOCATIONS ARE
0017                 * THE DISPLAY CODES DESCRIBED IN FIGURE 3.2 OF THIS
0018                 * MANUAL.  FOR EXAMPLE, TO INITIALIZE THE TIME TO 3:45:27 AM,
0019                 * PROCEED AS FOLLOWS ON THE INSTRUCTOR 50:
0020                 *                      REG F   ENTER FAST PATCH
0021                 *                      100 E/N ENTER STARTING ADDRESS
0022                 *                      17      BLANK CODE
0023                 *                      83      3.  CODE
0024                 *                      04      4 CODE
0025                 *                      85      5.  CODE
0026                 *                      02      2 CODE
0027                 *                      07      7 CODE
0028                 *                      17      BLANK CODE
0029                 *                      0A      A CODE
0030                 *                      E/N     TERMINATE FAST PATCH MODE
0031                 * AFTER INITIAL TIME VALUE IS ENTERED, DEPRESS 'RES' KEY TO
0032                 * BEGIN PROGRAM EXECUTION.
0033                 *
0034                 *******************************************************************
0035                 *   REGISTER EQUATES
0036 0000            R0     EQU     0       REGISTER 0
0037 0001            R1     EQU     1       REGISTER 1
0038 0002            R2     EQU     2       REGISTER 2
0039 0003            R3     EQU     3       REGISTER 3
0040                 *  CONDITION CODES
0041 0001            P      EQU     1       POSITIVE RESULT
0042 0000            Z      EQU     0       ZERO RESULT
0043 0002            N      EQU     2       NEGATIVE RESULT
0044 0002            LT     EQU     2       LESS THAN
0045 0000            EQ     EQU     0       EQUAL TO
0046 0001            GT     EQU     1       GREATER THAN
0047 0003            UN     EQU     3       UNCONDITIONAL
0048                 *  PSW LOWER EQUATES
0049 0000            CC     EQU     H'00'   CONDITIONAL CODES
0050 0020            IDC    EQU     H'20'   INTERDIGIT CARRY
0051 0010            RS     EQU     H'10'   REGISTER BANK
0052 0008            WC     EQU     H'08'   1=WITH 0=WITHOUT CARRY
0053 0004            OVF    EQU     H'04'   OVERFLOW
0054 0002            COM    EQU     H'02'   1=LOGIC 0=ARITHMETIC COMPARE
0055 0001            C      EQU     H'01'   CARRY/BORROW
0056                 *  PSW UPPER EQUATES
```

```
LINE ADDR  OBJECT  E SOURCE

0057 0080            SENS   EQU     H'80'   SENSE BIT
0058 0040            FLAG   EQU     H'40'   FLAG BIT
0059 0020            II     EQU     H'20'   INTERRUPT INHIBIT
0060 0007            SP     EQU     H'07'   STACK POINTER
0061                 * END OF EQUATES
0062                 *
0063                 ************************************************************
0064                 *
0065                 * PROGRAM BEGINS HERE. LOCATE AT STARTING ADDRESS H'0000'
0066                 *
0067                 ************************************************************
0068                 *
0069 0000                   ORG     0       PROGRAM STARTS AT H'0000'
0070                 *
0071 0000 7620       START  PPSU    II      INHIBIT INTERRUPTS
0072 0002 1F0095            BCTA,UN DISP    BRANCH AROUND INTERRUPT ROUTINE TO DISPLAY
0073                 *
0074 0005                   ORG     7       LOCATION OF INTERRUPT ROUTINE
0075                 *
0076 0007 0C0108     CLOCK  LODA,R0 FRAC    GET FRACTIONAL SECONDS
0077 000A 8401              ADDI,R0 1       ADD 1 SINCE INTERRUPTED EVERY 60TH OF A SEC
0078 000C CC0108            STRA,R0 FRAC    RESTORE THE VALUE
0079 000F E43C              COMI,R0 60      HAVE WE COUNTED ONE SECOND?
0080 0011 16                RETC,LT         NO-RETURN TO DISPLAY
0081                 *                      YES-MUST INCREMENT SECONDS
0082 0012 20                EORZ    R0      SET R0 TO ZERO
0083 0013 CC0108            STRA,R0 FRAC    SET FRAC TO ZERO
0084 0016 0C0105            LODA,R0 USEC    GET UNIT SECONDS
0085 0019 8401              ADDI,R0 1       ADD 1
0086 001B CC0105            STRA,R0 USEC    PUT IT BACK
0087 001E E40A              COMI,R0 10      IS UNIT SECONDS 10?
0088 0020 16                RETC,LT         NO-RETURN TO DISPLAY
0089 0021 20                EORZ    R0      SET R0 TO ZERO
0090 0022 CC0105            STRA,R0 USEC    SET UNIT SEC TO ZERO
0091 0025 0C0104            LODA,R0 TSEC    GET TENS OF SECONDS
0092 0028 8401              ADDI,R0 1       ADD 1
0093 002A CC0104            STRA,R0 TSEC    PUT IT BACK
0094 002D E406              COMI,R0 6       REACHED 60 SECONDS?
0095 002F 16                RETC,LT         NO-RETURN TO DISPLAY
0096                 *                      YES-MUST INCREMENT MINUTES
0097 0030 20                EORZ    R0      SET R0 TO ZERO
0098 0031 CC0104            STRA,R0 TSEC    SET TENS OF SEC TO ZERO
0099 0034 0C0103            LODA,R0 UMIN    GET UNIT MINUTES
0100 0037 8401              ADDI,R0 1       ADD 1
0101 0039 CC0103            STRA,R0 UMIN    PUT IT BACK
0102 003C E48A              COMI,R0 H'8A'   REACHED 10 MINUTES?
0103 003E 16                RETC,LT         NO-RETURN TO DISPLAY
0104 003F 0480              LODI,R0 H'80'   SET R0 TO '0.'
0105 0041 CC0103            STRA,R0 UMIN    SET UNIT MIN TO '0.'
0106 0044 0C0102            LODA,R0 TMIN    GET TENS OF MINUTES
0107 0047 8401              ADDI,R0 1       ADD 1
0108 0049 CC0102            STRA,R0 TMIN    PUT IT BACK
0109 004C E406              COMI,R0 6       REACHED ONE HOUR?
0110 004E 16                RETC,LT         NO-RETURN TO DISPLAY
0111                 *                      YES-MUST INCREMENT HOURS
0112 004F 20                EORZ    R0      SET R0 TO ZERO
```

```
LINE ADDR  OBJECT  E SOURCE

0113 0050 CC0102          STRA,R0 TMIN    SET MINUTES TO ZERO
0114 0053 0C0101          LODA,R0 UHRS    GET UNIT HOURS
0115 0056 8401            ADDI,R0 1       ADD 1 TO UNIT HOURS
0116 0058 CC0101          STRA,R0 UHRS    PUT IT BACK
0117                *                     MUST CHECK IF HAVE REACHED '13' HRS
0118 005B E483            COMI,R0 H'83'   IS UNIT HRS A 3 (83=3.)?
0119 005D 1812            BCTR,EQ HRS13   BRANCH IF YES
0120                *                     NO - MUST CHECK IF REACHED '12' HRS TO
0121                *                        CHANGE A TO P OR VICE-VERSA
0122 005F E482            COMI,R0 H'82'   IS UNIT HRS A 2 (82=2.)?
0123 0061 181F            BCTR,EQ HRS12   BRANCH IF YES
0124 0063 E48A            COMI,R0 H'8A'   IS UNIT HRS 10 (A=10)?
0125 0065 16              RETC,LT         NO-RETURN TO DISPLAY
0126 0066 0480            LODI,R0 H'80'   SET R0 TO '0.'
0127 0068 CC0101          STRA,R0 UHRS    PUT IN UNIT HOURS
0128 006B 0401            LODI,R0 1       MUST SET TENS OF HRS TO 1
0129 006D CC0100          STRA,R0 THRS    STORE IN TENS OF HRS
0130 0070 17              RETC,UN         RETURN TO DISPLAY
0131                *                     FOLLOWING CODE CHANGES '13' HOURS TO
0132                *                        '1' HOUR
0133 0071 0C0100    HRS13 LODA,R0 THRS    CHECK IF '3' OR '13'
0134 0074 E417            COMI,R0 H'17'   IS THRS A BLANK?
0135 0076 14              RETC,EQ         YES -IT IS '3', NOT '13'-RETURN TO DISPLAY
0136                *                     NO-MUST BE '13' SO CHANGE TO '1'
0137 0077 0417            LODI,R0 H'17'   CODE FOR BLANK
0138 0079 CC0100          STRA,R0 THRS    SET TENS OF HRS TO BLANK
0139 007C 0481            LODI,R0 H'81'   CODE FOR '1.'
0140 007E CC0101          STRA,R0 UHRS    STORE IN UNIT HOURS
0141 0081 17              RETC,UN         RETURN TO DISPLAY
0142                *                     FOLLOWING CODE CHANGES A TO P AND
0143                *                        VICE-VERSA AT '12' HOURS
0144 0082 0C0100    HRS12 LODA,R0 THRS    FIRST CHECK IF '2' OR '12'
0145 0085 E417            COMI,R0 H'17'   IS TENS OF HRS A BLANK?
0146 0087 14              RETC,EQ         YES, MUST BE '2'-RETURN TO DISPLAY
0147 0088 040A            LODI,R0 H'0A'   LOAD CODE FOR 'A'
0148 008A EC0107          COMA,R0 AMPM    IS SYMBOL NOW AN 'A'?
0149 008D 9802            BCFR,EQ SYMB    NO-IT IS 'P' SO MUST CHANGE TO 'A'
0150 008F 0410            LODI,R0 H'10'   CODE FOR 'P' SINCE AMPM NOW 'A'
0151 0091 CC0107    SYMB  STRA,R0 AMPM    STORE NEW SYMBOL
0152 0094 17              RETC,UN         RETURN TO DISPLAY
0153                *
0154                * THIS IS THE DISPLAY ROUTINE
0155                *
0156 0095 75FF      DISP  CPSL    H'FF'   CLEAR ALL BITS OF PSL
0157 0097 7702            PPSL    COM     SET COM=1 FOR ARITH COMPARES
0158                *                     PRESET REGISTERS FOR MONITOR DISPLAY ROUTINE
0159 0099 0500            LODI,R1 <THRS-1  UPPER PART OF DISPLAY BUFFER LOCATION-1
0160 009B 06FF            LODI,R2 >THRS-1  LOWER PART
0161 009D 0701            LODI,R3 1       CODE FOR ONE SCAN AND RETURN
0162 009F 7620            PPSU    II      INHIBIT INTERRUPTS WHILE DISPLAYING
0163 00A1 BBE6            ZBSR    *H'1FE6'        GO TO MONITOR DISPLAY ROUTINE
0164                *                     AFTER ONE SCAN THROUGH DISPLAY, PROGRAM WILL
0165                *                        CONTINUE EXECUTION HERE
0166 00A3 7420            CPSU    II      ENABLE INTERRUPTS - IF INTERRUPT HAS
0167                *                        OCCURRED WILL GO TO INTERRUPT ROUTINE AND THEN
0168                *                        RETURN TO NEXT INSTRUCTION
```

```
LINE ADDR  OBJECT  E SOURCE

0169 00A5 1B6E              BCTR,UN DISP    CONTINUE DISPLAYING
0170                 *
0171                 * MEMORY AREA FOR DISPLAY BUFFER
0172                 *
0173 00A7                   ORG     H'100'  START DISPLAY BUFFER AT H'100'
0174                 *
0175 0100            THRS   RES     1       TENS OF HOURS
0176 0101            UHRS   RES     1       UNIT HOURS
0177 0102            TMIN   RES     1       TENS OF MINUTES
0178 0103            UMIN   RES     1       UNIT MINUTES
0179 0104            TSEC   RES     1       TENS OF SECONDS
0180 0105            USEC   RES     1       UNIT SECONDS
0181 0106 17         SPACE  DATA    H'17'   BLANK SPACE
0182 0107            AMPM   RES     1       A OR P SYMBOL
0183                 *
0184                 * END OF DISPLAY BUFFER
0185                 *
0186 0108            FRAC   RES     1       60THS OF SEC COUNTER
0187                 *
0188 0000                   END     START

 TOTAL ASSEMBLY ERRORS = 0000
```

The simple programs described above are designed to demonstrate some of the capabilities of the INSTRUCTOR 50 and to give you a feel for how the system works. Additional programming examples are presented in subsequent sections of this manual.

3. SYSTEM OVERVIEW

Introduction

A simplified block diagram of the INSTRUCTOR 50 system is shown in Figure 3.1. Major system components include:

- 2650 8-bit, N-channel microprocessor
- 2656 System Memory Interface (SMI)
- Sixteen-key hexadecimal keyboard
- Twelve-key function selection keyboard
- Eight-digit, 7-segment display
- Audio tape cassette interface
- S100-compatible expansion bus
- User System Executive (USE) monitor
- Debugging aids
- On-board user Input/Output
- Forced jump logic
- 512 bytes of on-board user RAM
- Crystal-controlled system clock

2650 Microprocessor

The 2650 processor is a single-chip microprocessor made using an ion-implanted, N-channel silicon-gate process. It has a fixed command set of 75 instructions, operates on 8-bit parallel data and can address 32,768 bytes of memory. All bus outputs of the 2650 are three-state and can drive either one 7400-type load, or four 74LS loads.

The 2650 contains a total of seven general-purpose registers, each eight bits long. They may be used as source or destination for arithmetic operations, as index registers, and for Input/Output (I/O) data transfers.

The processor instructions are one, two, or three bytes long, depending on the instruction. Variable length instructions tend to conserve memory space since a one or two-byte instruction may often be used rather than a three-byte instruction. The first byte of each instruction always specifies t e operation to be performed and the addressing mode to be used. Most instructions use six of the first eight bits for this purpose, with the remaining two bits as an operation code.

The 2650 has a versatile set of addressing modes used for locating operands for operations and an interrupt mechanism which is implemented as a single level, address vectoring type. Address vectoring means that an interrupting device can force the processor to execute code at a device-determined location in memory.

Detailed hardware and software information on the 2650 microprocessor is provided in Chapter 9.

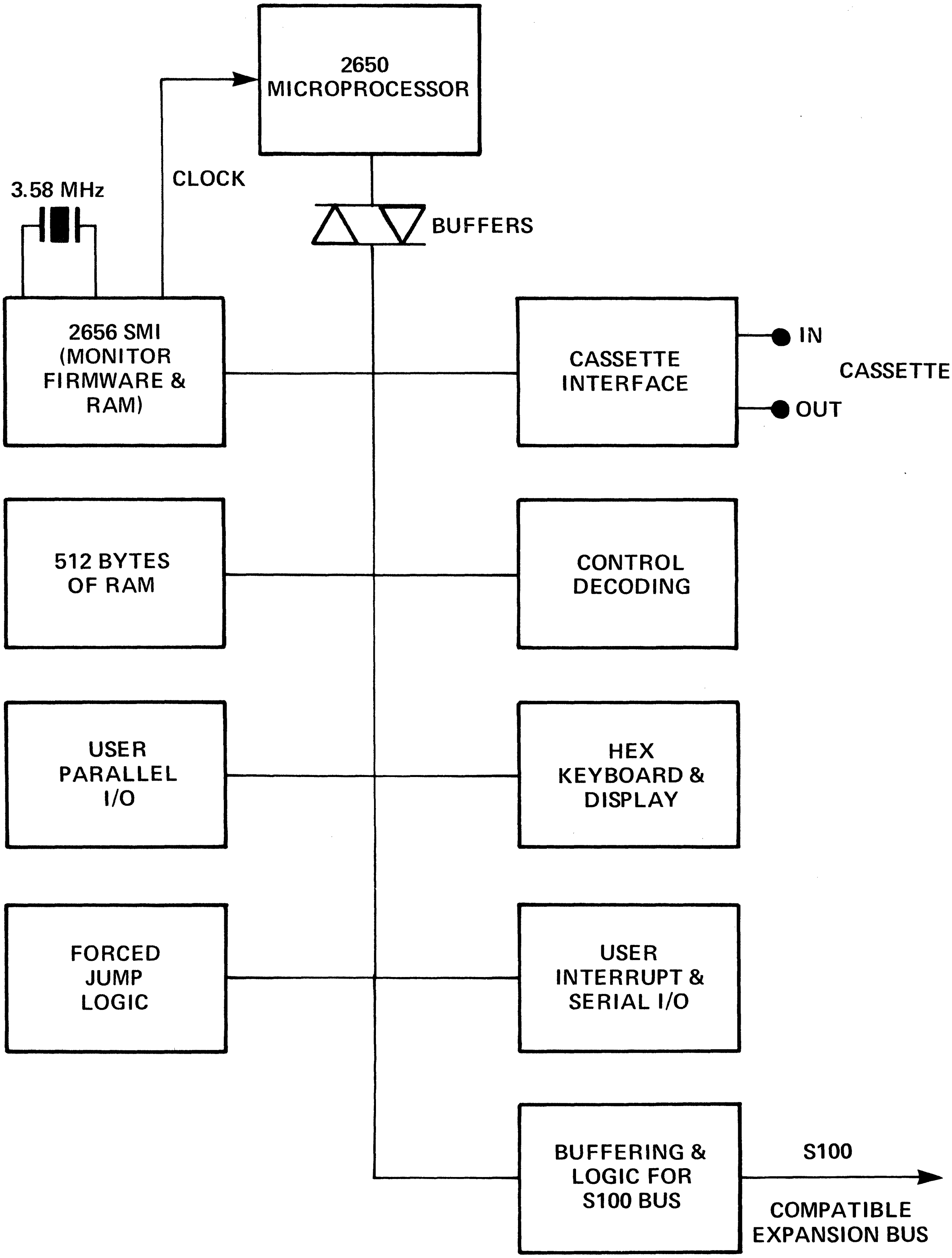


Figure 3.1: Instructor 50 Basic Block Diagram

2656 System Memory Interface

The Signetics 2656 System Memory Interface (SMI) contains Read-Only Memory (ROM), Random-Access Memory (RAM), and a programmable I/O port.

Two notable features are onboard decoders that make it possible to place the ROM and RAM anywhere in the memory space and an I/O port that can be set up as either a bidirectional port or as chip-select lines. The chip-select capability eliminates a great deal of the TTL that usually surrounds microprocessors. The 2K USE monitor, 128 bytes of scratchpad memory, I/O decode logic, and the system clock are housed in the 2656 SMI.

Keyboards

A 16-key hexadecimal keyboard and a 12-key function control keyboard enable you to communicate with the INSTRUCTOR 50. Both the hexadecimal keyboard and the function keyboard are under control of the USE monitor. The monitor performs a scanning process to determine what key has been depressed and what action is to be taken by the INSTRUCTOR 50 as a result of the depression. A functional description of the various controls and indicators is provided in Chapter 4.

Display Panel

The 8-digit, 7-segment display panel provides responses to input commands and guides you in the use of the INSTRUCTOR 50 by displaying prompting messages describing the data that must be entered.

Messages or responses are displayed using the seven-segment display font illustrated in Figure 3.2. Note that the characters 'b' and 'd' are always displayed with the right-hand decimal point attached in order to distinguish these characters from the number '6'.

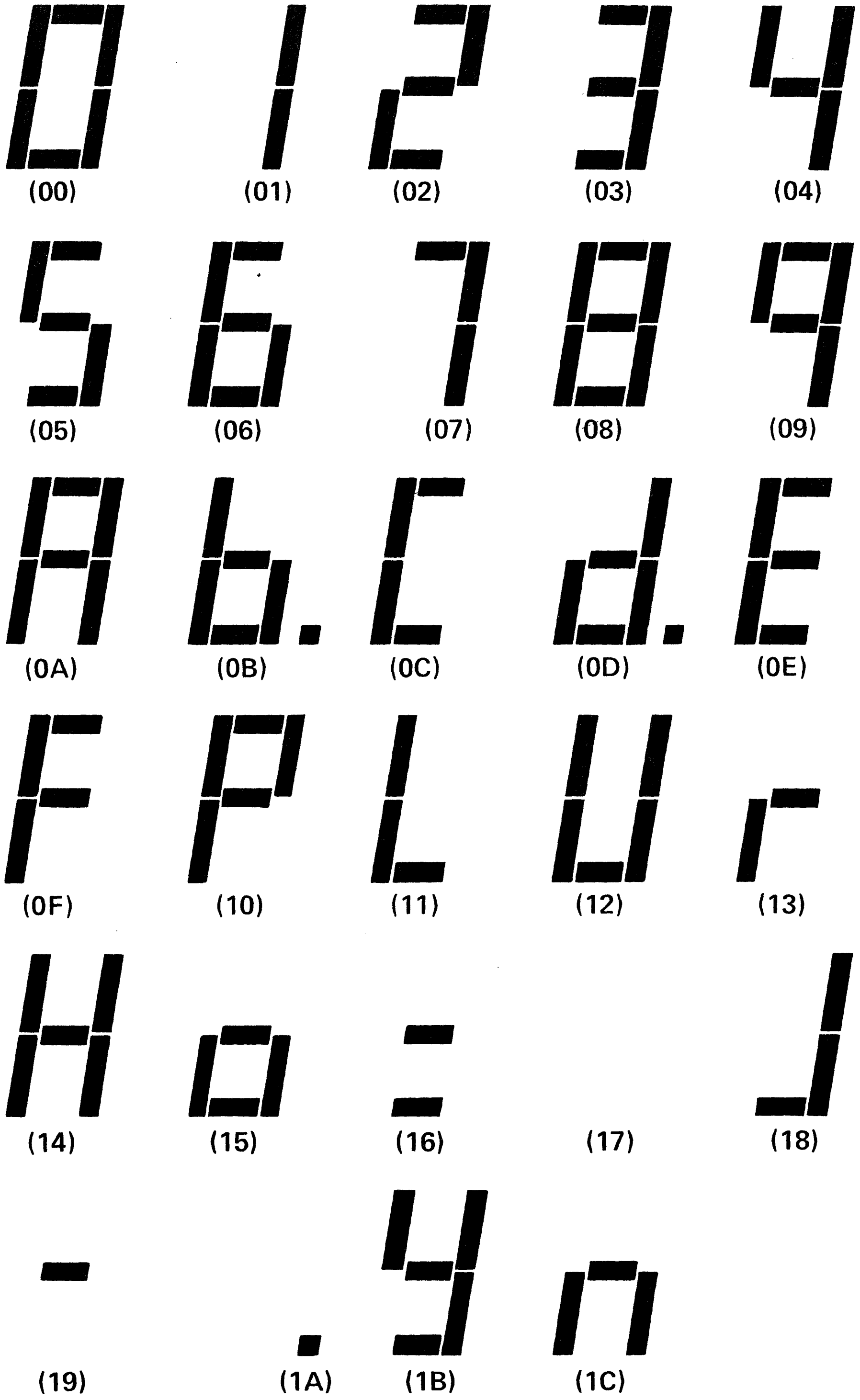
Figure 3.2 also shows the hexadecimal code required in the monitor's display buffer to display the character illustrated. To display a character with a right-hand decimal point attached, H'80' must be added to the value given. For example, H'07' will display '7', while H'87' will display '7.'. Refer to Chapter 6 for additional information on the use of the monitor's display subroutine.

Audio Cassette Interface

An audio cassette interface lets you load and store programs into and out of RAM. The storage medium is any audio cassette recorder.

S100-Compatible Expansion Bus

The INSTRUCTOR 50 includes an S100-compatible expansion bus connector so that other standard products, such as additional memory or prototyping cards, can be used with the system. This connector carries all of the 2650's I/O signals in addition to control signals required by the S100 bus. (See Chapter 7.)



( ) INDICATES THE HEX VALUE USED AS THE INTERNAL DISPLAY CODE.

NOTE: IF 80_{16} IS ADDED TO ANY CODE, A DECIMAL POINT WILL APPEAR WITH THE CHARACTER.

Figure 3.2: Instructor 50 Display Font

Monitor Firmware

The USE (User System Executive) monitor supervises operation of the INSTRUCTOR 50 and allows you to enter and alter programs, execute these programs in continuous or single-step modes, and perform a number of auxiliary functions. Monitor commands are entered via the control keys and the hexadecimal keyboard, and responses are displayed on the monitor display.

A basic flowchart of the monitor is shown in Figure 3.3. The monitor normally idles in the scan display and keyboard mode. If a key closure is detected during the scan, the monitor verifies that this is a new key closure (that any previously depressed key had been released), extinguishes the display, performs a keyboard debounce function, and then performs the requested function. The monitor then resumes the display and keyboard scan.

Monitor functions are terminated by depressing a new function key. Interrupts are inhibited while the monitor is running.

Debugging Aids

Two key features incorporated into INSTRUCTOR 50 are designed specifically for program debugging. These features are:

1. The ability to set a breakpoint that automatically interrupts execution of programs at any point without loss of hardware or software status.

2. The ability to step through a program one instruction at a time.

When a breakpoint is encountered during program execution or when a single instruction is executed in the single-step mode, control is returned to the monitor at which time you may examine the 2650 registers, the Program Status Word (PSW), and the program counter to determine the status of the microcomputer. You can then continue execution, set a new breakpoint, or resume the single-step operation. While in the monitor mode, you may change any register value, including the PSW and program counter, and you may alter memory locations.

On-Board User I/O

Both parallel and serial I/O are available in the INSTRUCTOR 50. The parallel I/O port provides 8 switch inputs and 8 individual Light-Emitting Diodes (LEDs) as a latched output port. A single LED is attached to the processor's FLAG output, and the SENS key on the function control keyboard allows you to test the processor's SENSE input. Additionally, you may exercise interrupt operation by using the interrupt (INT) key on the function control keyboard. See Chapter 6 for a discussion of the INSTRUCTOR 50's I/O capabilities.

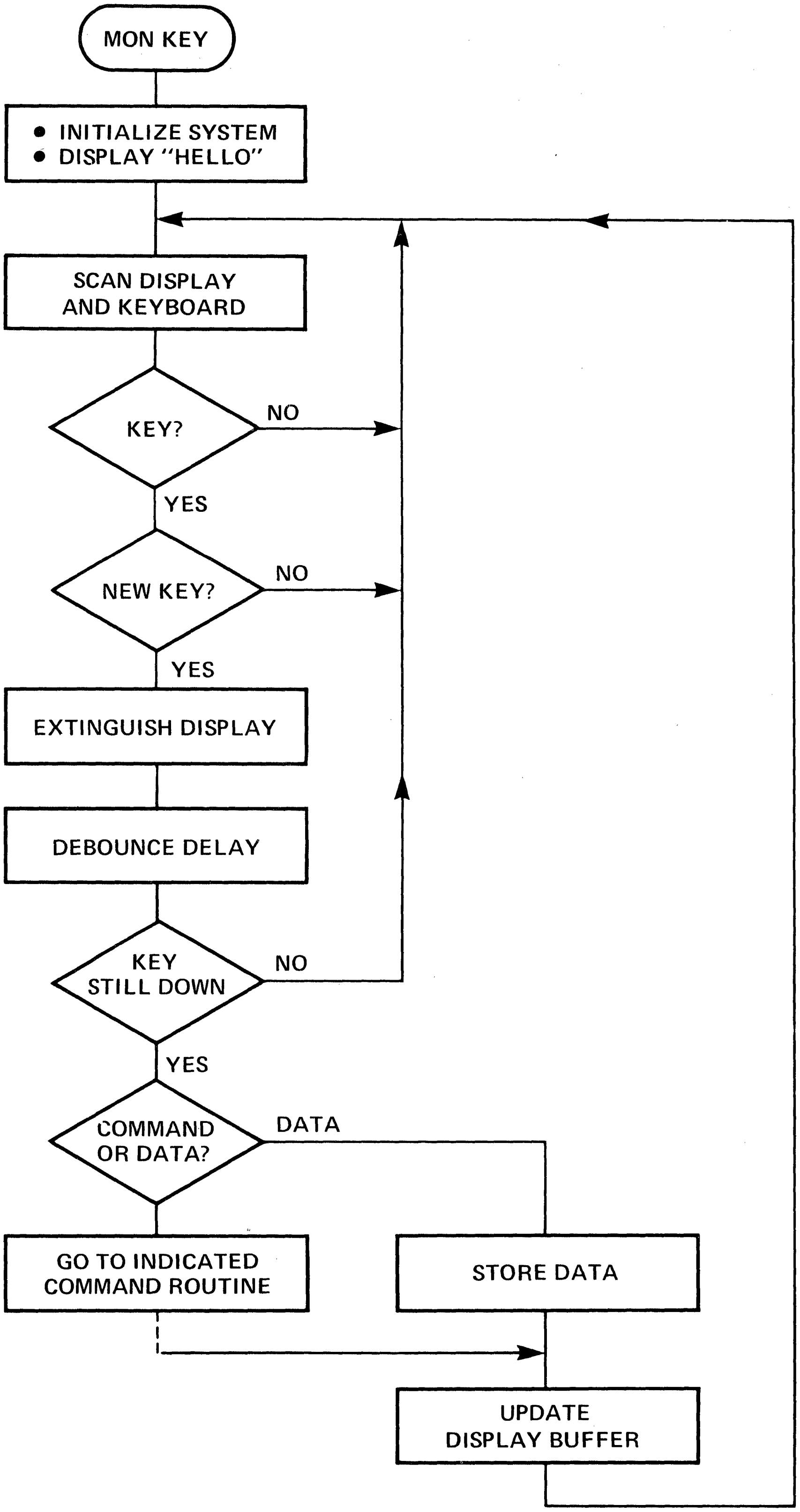


Figure 3.3: Basic Use Monitor Flow Chart

Forced Jump Logic

The Forced Jump Logic performs the following functions:

- Entry into the MONITOR mode when power is applied to the INSTRUCTOR 50 or when the MON key is depressed.

- Re-entry to the MONITOR mode after executing one instruction in single-step operation or upon detection of a breakpoint.

Memory and I/O Organization

512 bytes of RAM storage is provided for storing user programs and data. The RAM area may be expanded via the expansion bus connector.

Partitioning of the INSTRUCTOR 50's memory and I/O locations is illustrated in Figure 3.4. The supplied user memory occupies locations H'0000' to H'01FF' and may be expanded to occupy locations H'0200' - H'0FFF' and H'2000' - H'7FFF'. The extended I/O ports from H'00' to H'F7' are available for program use. Ports H'F8' to H'FF' and memory locations H'1000' to H'1FFF' are reserved for the USE monitor.

An additional 64 bytes of RAM storage is available to user programs for storing data values. This additional storage space occupies memory locations H'1780' to H'17BF'. Because of the way the USE monitor operates, instructions should not be stored at these locations.

The INSTRUCTOR 50 I/O data port is assigned one of three locations, depending on the setting of the Port Address Select Switch. These are memory address H'0FFF', extended I/O address H'07', or non-extended I/O Port D.

Clock Circuitry

The 2656 SMI provides the clock circuitry for the INSTRUCTOR 50. A 3.579545 MHz crystal is used to provide the reference frequency.

Internal Power Supply

The INSTRUCTOR 50 uses a self-contained A-C power pack that produces 8 VAC @ 1.5A. An on-board rectifier and regulator reduces this to 5 VDC. A jumper option permits the use of an alternate 8 VDC source. The INSTRUCTOR 50 may be plugged into any standard 115 VAC domestic wall socket. (European models require 220 VAC primary power.)

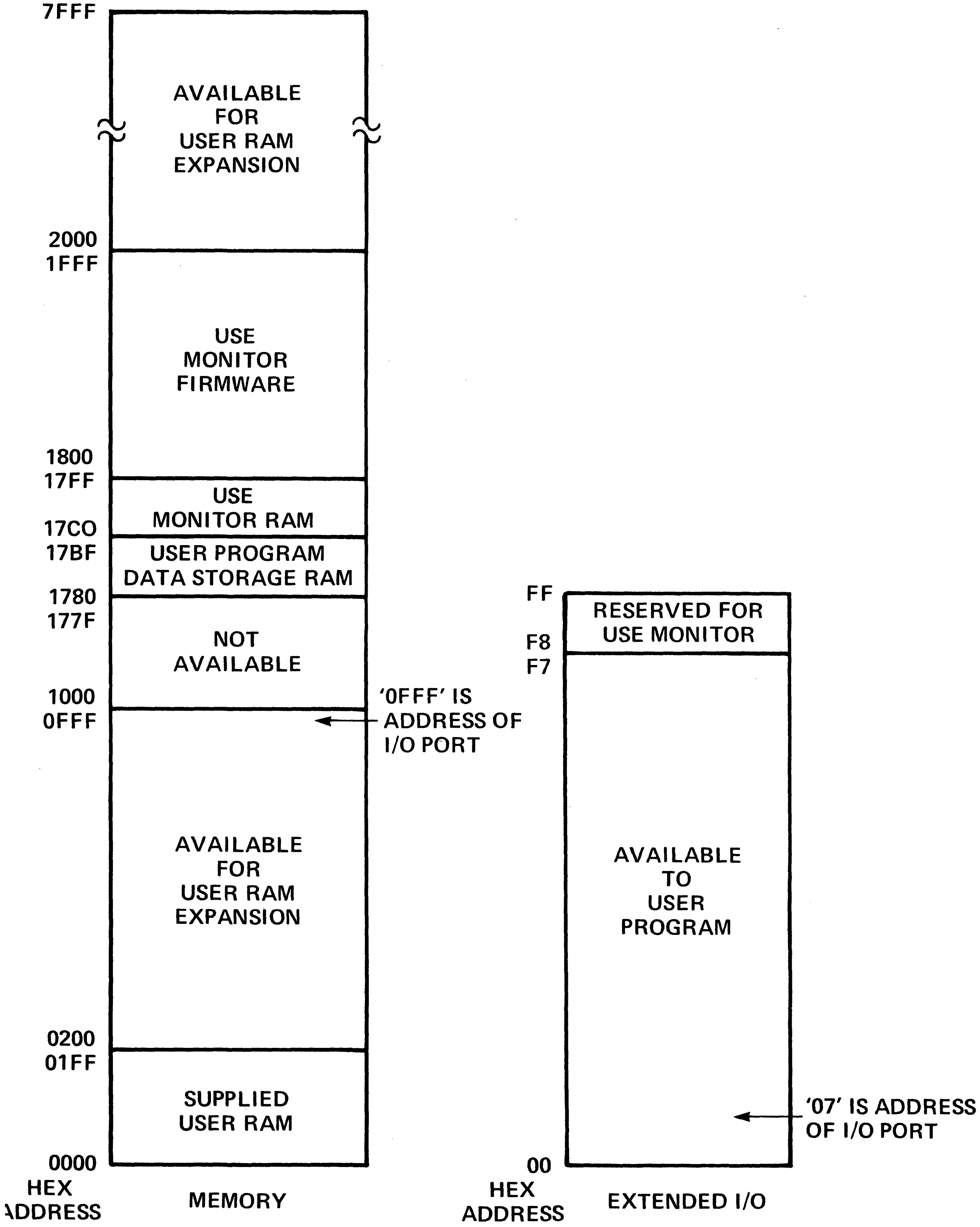


Figure 3.4: Memory And I/O Organization

4. CONTROLS AND INDICATORS

Introduction

This chapter provides a brief functional description of the various keys, switches and indicators associated with the INSTRUCTOR 50. See Figure 4.1.

The 12-key Function Control Keyboard and the 16-key Hexadecimal keyboard enable you to communicate with, enter data, and perform the various system functions associated with the INSTRUCTOR 50. The 8-digit display is used by the USE monitor to display responses to keyed input commands. The other switches and indicators are associated with various INSTRUCTOR 50 facilities.

Function Control Keyboard

The keys in the left-most column of the function control keyboard (SENS, INT, MON, and RST) are used primarily for system control. All other keys on this keyboard perform functions associated with entry, execution, and debugging of programs.

The RST and MON keys are active at all times. All other keys except SENS and INT are normally active only during the monitor mode. Depressing these keys while executing your program has no effect. The SENS and INT keys are active only during execution of a program and have no effect on monitor operation.

However, you may take advantage of the INSTRUCTOR 50's keyboard and display facilities by incorporating calls to the monitor subroutines controlling these devices as part of your program. See Chapter 5 for a description of these subroutines.

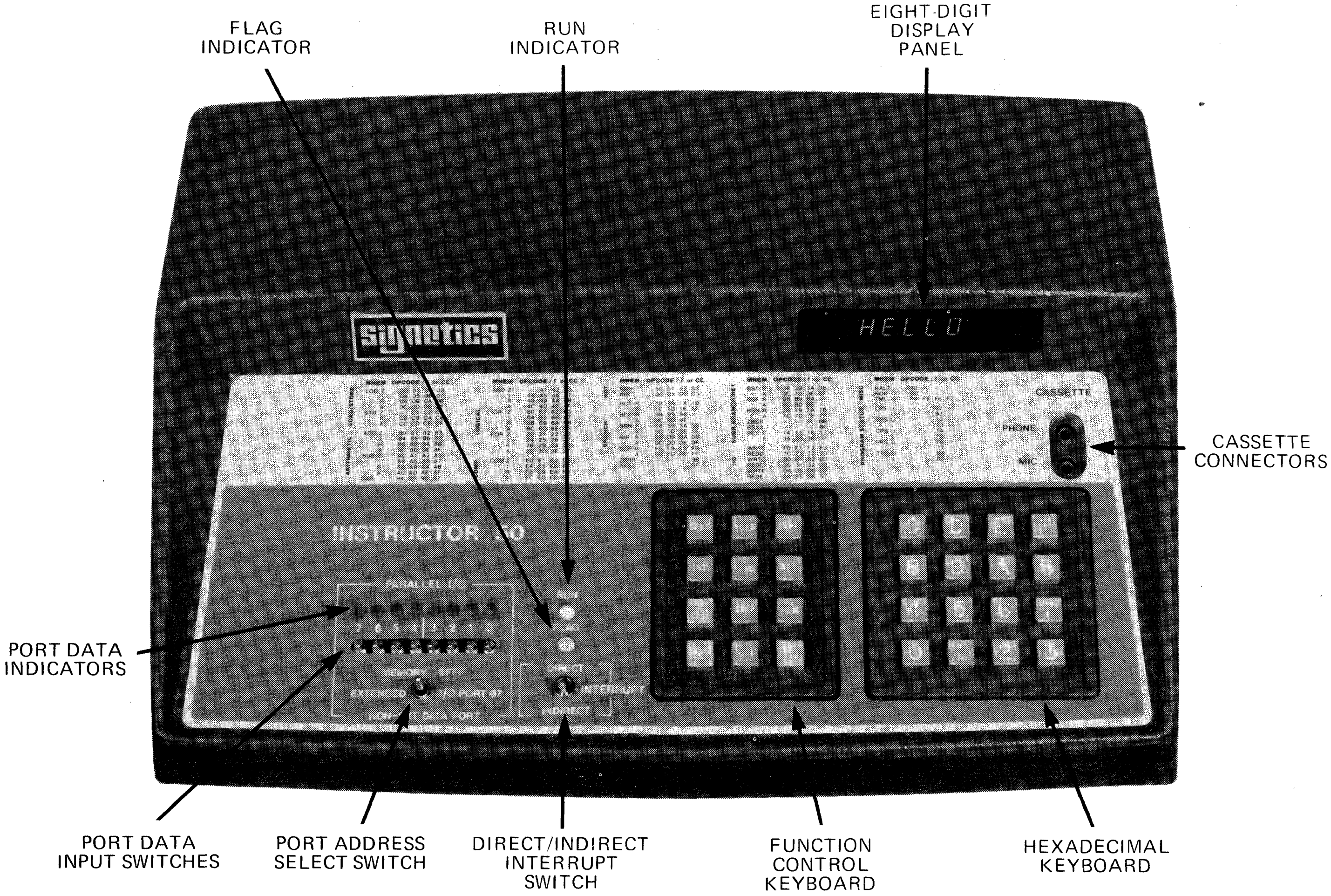


Figure 4.1: Controls and Indicators

<u>KEY</u> <u>FUNCTION</u>

SENS Controls the SENSE input to the 2650 when executing a user program. The SENSE input is normally a logic '0'. Depressing the SENS key will set the SENSE input to a logic '1'.

INT Allows you to manually interrupt the processor when executing a program. When this key is depressed, an interrupt sequence begins, resulting in the processor being vectored to or through memory location 07. The Direct/Indirect switch on the INSTRUCTOR 50 panel determines whether an instruction at location 07 is executed (Direct), or whether location 07 contains a branch address to another location in the user memory (Indirect). A switch accessible through a cutout in the bottom panel permits interrupts to be controled by the AC line input frequency. See Chapter 5 for more information on INT options.

MON Termintes any operation in process and causes the forced jump logic to output a jump instruction sequence resulting in an entry to the monitor mode. The response to a depression of the MON key is the message HELLO on the display panel.

RST When this key is depressed, any current operation is terminated, and a RESET signal is applied to the 2650 causing program execution to begin at address zero. The system does not enter the monitor mode when this key is depressed.

WCAS Allows programs to be transferred from the INSTRUCTOR 50 memory to audio cassette tape.

RCAS Allows programs to be transferred from audio cassette tape to the INSTRUCTOR 50 memory.

STEP Causes the 2650 to execute a single program instruction and return to the monitor mode, displaying the address of the next instruction to be executed on the monitor display.

RUN Depressing this key terminates the monitor mode and causes program execution to begin at previously specified address. Program execution continues until (1) a breakpoint is encountered; (2) the RST or MON keys are depressed; or (3) the program executes a WRTC or HALT instruction.

BKPT Allows you to specify and examine a program breakpoint.

REG Places the INSTRUCTOR 50 in the Display and Alter Registers mode. In this mode, you may examine and alter the contents of the 2650's general-purpose registers, the program counter value, and the value of the Program Status Word (PSW). This key is also used to initiate entry into the ADJUST CASSETTE and FAST PATCH commands. See Chapter 5.

MEM Places the INSTRUCTOR 50 in the Display and Alter Memory mode. In this mode you may specify memory locations that you wish to examine, and you may alter the contents of these memory locations.

ENT NXT Enters keyed-in data into memory or registers and also causes the contents of the next sequential memory or register location to be displayed. The use of this key during the various monitor operations is described in the detailed command descriptions, Chapter 5.

Hexadecimal Keyboard

The 16-key hexadecimal keyboard (0 through 9 and A through F) is used to enter address and data parameters as required. This keyboard is also used in conjunction with the REG key on the function control keyboard to enable certain commands. See detailed command descriptions, Chapter 5.

Eight-Digit Hex Display Panel

The 8-digit display panel is used by the monitor to display prompting messages and responses to keyed input commands. It also displays prompting messages to guide you in the operation of the INSTRUCTOR 50.

Port Data Input Switches

These eight switches are used to specify a byte of input data at the parallel I/O port. This value is read when the 2650 executes a read I/O port instruction.

Port Data Indicators

The eight I/O port LEDs reflect the current value in the parallel output port latch. This latch is loaded with the contents of an internal register by a write I/O port instruction.

Direct/Indirect Interrupt Switch

This switch determines whether the 2650 executes a direct or indirect branch to subroutine when it acknowledges an interrupt request.

Port Address Select Switch

This switch selects the manner in which the parallel I/O port is addressed. The three modes are: non-extended I/O - Port D, extended I/O at port address 07_{16}, and memory mapped I/O at address $OFFF_{16}$.

FLAG Indicator

This LED indicates the current value of the FLAG bit in the 2650's Program Status Word. If the FLAG bit is a one, the LED is on. If the FLAG bit is a zero, the LED is off.

RUN Indicator

The RUN indicator reflects the operating status of the 2650. When the 2650 is executing either the monitor program or a user program, the RUN light is on. The RUN light is off when the 2650 has executed a HALT instruction or when the PAUSE line of the S100 interface has been driven low.

5. COMMAND DESCRIPTIONS

Introduction

This chapter describes the various commands available to the INSTRUCTOR 50 user. These commands include:

DISPLAY AND ALTER REGISTERS
DISPLAY AND ALTER MEMORY
FAST PATCH
DISPLAY AND ALTER PROGRAM COUNTER
BREAKPOINT
STEP
WRITE CASSETTE
ADJUST CASSETTE
READ CASSETE
RUN
RESET

In this chapter, each pair of facing pages discusses a single command. The left-hand page is devoted to text, while the right-hand page actually shows what is displayed on the monitor display panel when specific keys are depressed. The circled numbers imbedded in the text on the left-hand page correspond with the circled numbers on the right-hand page.

A discussion of the INSTRUCTOR 50's error messages is presented at the end of this chapter.

DISPLAY AND ALTER REGISTERS

FUNTION: This command allows you to inspect and alter, if desired, the contents of the 2650's general-purpose registers and/or Program Status Word (PSW).

PROCEDURE:

1. Depress the $\boxed{\text{REG}}$ key ① followed by the register address corresponding to the first register to be inspected, ② according to the following table:

REGISTER ADDRESS	REGISTER
0	R0
1	R1, bank 0
2	R2, bank 0
3	R3, bank 0
4	R1, bank 1
5	R2, bank 1
6	R3, bank 1
7	PSU
8	PSL

2. The contents of the register are displayed as two hex digits in the data field of the display. ②

3. The register contents may be modified at this time by keying in a new value followed by $\boxed{\text{ENT/NXT}}$. The numbers keyed in and appearing in the DATA display field are displayed there only and can be edited by simply keying in the correct characters ⑨. The display shifts to the left each time a new character is entered, and characters shifted out of the two-digit field are lost. The hex value appearing on the display is deposited in the register when the $\boxed{\text{ENT/NXT}}$ key is depressed. ⑩

4. When the $\boxed{\text{ENT/NXT}}$ key is depressed after step 2 or 3, the next higher register in sequence will be displayed as in step 2 ③ unless the PSL is being displayed, in which case R0 will be the next register displayed. ⑩

5. The command is terminated by initiating any other command.

6. If the keys 9, B, D, or E are depressed following $\boxed{\text{REG}}$ in step 1, the key depression will be ignored. If the keys A, C, or F are depressed, the INSTRUCTOR 50 will enter the ADJUST CASSETTE, DISPLAY AND ALTER PROGRAM COUNTER, or FAST PATCH commands, respectively. See appropriate command descriptions.

DISPLAY AND ALTER REGISTERS

EXAMPLES

	KEY	DISPLAY	COMMENTS
①	REG	r =	Awaiting register address
②	4	.r4 = 7E	R1, bank 1 = H'7E'
③	ENT/NXT	.r5 = 0F	R2, bank 1 = H'0F'
④	ENT/NXT	.r6 = 13	R3, bank 1 = H'13'

Example A: Examine contents of R1 - R3 of bank 1

	KEY	DISPLAY	COMMENTS
⑤	REG	r =	Awaiting register address
⑥	7	.PU = 04	PSU = H'04'
⑦	ENT/NXT	.PL = 53	PSL = H'53'
⑧	4 8	.PL = 48	Wrong data entered
⑨	4 0	.PL = 40	Correct data entered
⑩	ENT/NXT	.r0 = 72	Entered data deposited in PSL and R0 contents displayed.

Example B: Examine contents of PSW and change contents of PSL to H'40'

DISPLAY AND ALTER MEMORY

FUNCTION: Allows you to examine and optionally alter the contents of memory locations individually. This command is particularly useful when you are debugging your program and wish to examine, verify and/or change the contents of memory locations.

PROCEDURE:

1. Depress the $\boxed{\text{MEM}}$ key ① followed by the address of the memory location to be inspected. ② If fewer than four digits are entered, the digits entered are used as the least-significant hexadecimal digits of the address. ② If more than four digits are entered, the last four digits are used as the address.

2. Depress the $\boxed{\text{ENT/NXT}}$ key ③ to display the contents of the specified memory location. The contents are displayed as two hexadecimal digits in the data field of the display.

3. You amy continue to examine the contents of sequential memory locations by depressing the $\boxed{\text{ENT/NXT}}$ key. ④ If you wish to alter the contents of any memory location, enter the new data via the hexadecimal keyboard. ⑧ Only the last two digits entered are retained, so that an error in entry can be corrected by entering the correct data. To deposit the new data into the specified memory location, you may either depress the $\boxed{\text{ENT/NXT}}$ key or transfer control to a new function by depressing a function key. ⑨

   Each time new data is specified, the monitor performs a read-after-write check to verify that you are not attempting to write into a ROM area or into non-existent memory. If the check fails, error message 3 is displayed. To recover from this error, depress the $\boxed{\text{MEM}}$ key and repeat the cycle correctly.

DISPLAY AND ALTER MEMORY

EXAMPLES

	KEY	DISPLAY	COMMENTS
①	MEM	.Ad. =	Awaiting memory address
②	1 0	.Ad. = 10	10 = Address of memory location to be examined
③	ENT/NXT	.0010 02	H'02' = contents of memory location 0010
④	ENT/NXT	.0011 FF	Address and contents of next sequential memory location

Example A: Examine contents of memory location 0010, and move to next sequential memory location.

	KEY	DISPLAY	COMMENTS
⑤	MEM	.Ad. =	Awaiting memory address
⑥	2 2	.Ad. = 22	Address of memory location to be examined
⑦	ENT/NXT	.0022 06	H'06' = Contents of memory location 0022
⑧	0 5	.0022 05	Desired contents of memory location 0022 entered and displayed.
⑨	REG	.r =	H'05' deposited into memory location 0022, Display and Alter Memory command is terminated, and monitor enters Display and Alter Registers command.

Example B: Examine contents of memory location 0022, change data, and transfer control to another function.

FAST PATCH

FUNCTION: The FAST PATCH command allows you to enter long strings of data into memory from the hexadecimal keyboard. Once the starting address is selected, data is loaded into memory sequentially--one byte for every two hex keys depressed. Once keyed in, data may not be changed in the FAST PATCH mode. To change data, you must use the DISPLAY AND ALTER MEMORY command or re-enter the FAST PATCH command starting at the address where the change is required.

PROCEDURE

1. To enter the FAST PATCH command, depress the REG key ① on the function control keyboard followed by F on the hexadecimal keyboard. ②

2. Enter the desired starting address on the hexadecimal keyboard. ③

   NOTE: You may bypass this step and go directly to step 3 to begin at a known starting address. The starting address is known under any one of the following conditions:

   a) When a file has been read into memory from a cassette tape by the INSTRUCTOR 50. The file's starting address will be the beginning address for the FAST PATCH.

   b) The address from which the last exit from the DISPLAY AND ALTER MEMORY or FAST PATCH command took place.

3. Depress the ENT/NXT key ④ on the function control keyboard to set the starting address. Data may now be entered into the specified address.

4. Enter desired data for the displayed address as two hex digits. ⑤ Continue entering data in this manner until all data is entered. The INSTRUCTOR 50 automatically increments the memory address as data is entered. ⑥ ⑦ ⑧ ⑨

5. Exit the FAST PATCH mode by depressing ENT/NXT or another function key. ⑩

6. A read-after-write check is performed as each byte is deposited. The INSTRUCTOR 50 will display Error 3 if data cannot be stored.

FAST PATCH

EXAMPLE

	KEY(S)	DISPLAY	COMMENTS
1	REG	r =	
2	F	.Ad. =	Awaiting starting memory address.
3	1 0	.Ad. = 10	Starting address entered.
4	ENT/NXT	.0010	Starting address set.
5	1 2	.0010 12	Data entry.
6	1 3	.0011 13	
7	1 4	.0012 14	
8	1 5	.0013 15	
9	1 6	.0014 16	
10	MEM	.Ad. =	Exit from FAST PATCH mode.

Enter Data String "12 13 14 15 16" into Successive Memory Locations Starting at Address H'10'

DISPLAY AND ALTER PROGRAM COUNTER

FUNCTION: The DISPLAY AND ALTER PROGRAM COUNTER command allows you to examine or change the address of the first instruction to be executed by the 2650 during execution of a RUN or STEP command.

PROCEDURE:

1. To enter the DISPLAY AND ALTER PROGRAM COUNTER command, depress the REG key ① on the function control keyboard followed by C on the hexadecimal keyboard. ②

2. The display will show the current Program Counter (PC) value as four hexadecimal digits. ②

3. If you want to change the PC address, enter the desired address on the hexadecimal keyboard. ③

   NOTE: For a multiple-byte instruction, the address entered is the address of the first byte.

4. Depress any command key ④ on the function control keyboard to set the desired starting address. If the ENT/NXT key is used, the INSTRUCTOR 50 transfers control to the DISPLAY AND ALTER REGISTERS command.

DISPLAY AND ALTER PROGRAM COUNTER

EXAMPLE

	KEY	DISPLAY	COMMENTS
①	REG	r =	
②	C	.PC = 0015	0015 = present contents of Program Counter.
③	1 7	.PC = 17	Starting address changed to 0017.
④	ENT/NXT	r =	Sets new starting address, and transfers control to DISPLAY AND ALTER REGISTERS command.

Set Starting Address for RUN Command to H'0017'

BREAKPOINT

The BREAKPOINT COMMAND allows you to enter, clear, or examine a program breakpoint. A breakpoint returns system control from the executing porgram to the monitor and enables you to examine the state of the memory and processor registers, make modifications, if desired, and continue program execution from the point of interruption.

PROCEDURE:

1. Depress the BKPT key on the function control keyboard ① to place the INSTRUCTOR 50 in the breakpoint mode.

2. The monitor will display either:

   a) A blank data field if a breakpoint address was not specified previously. ①

   b) The address of the breakpoint previously entered. ⑤

3. Enter the desired breakpoint address on the hexadecimal keyboard. ② If the desired address is already displayed, as in step (2b), re-entry is not required.

   NOTE: If a breakpoint is set at a multiple-byte instruction, the address specified for the breakpoint should be the address of the first byte.

4. Depress the ENT/NXT key ③ or another function key ④ to set the breakpoint at the address displayed.

5. To clear a breakpoint, depress the BKPT key twice in succession. ⑤ ⑥

NOTE: The breakpoint is inserted into your program when you enter the execution mode via the RUN command. When the breakpoint is encountered during program execution, the breakpoint address and contents are displayed, preceded by a "-" (minus) sign. The instruction at the breakpoint address is restored and executed prior to this display, and the Program Counter is updated to the address of the instruction following the breakpoint.

ERROR MESSAGES

During specification of the breakpoint address, the INSTRUCTOR 50 may display one of the following error messages:

ERROR 1 If the user attempts to specify a breakpoint address in the INSTRUCTOR 50's ROM address space or in non-existent memory. To clear this error, depress BKPT once.

ERROR 2 If the user attempts to enter a new breakpoint address after having set a previous breakpoint address by depression of the ENT/NXT key. To clear this error, depress any function key. The original breakpoint address will be saved.

BREAKPOINT

EXAMPLE

	KEY(S)	DISPLAY	COMMENTS
1	BKPT	.b.P =	No previous breakpoint specified. Waiting for breakpoint address.
2	4 4	.b.P = 44	Breakpoint address entered.
3	ENT/NXT	b.P = 0044	Breakpoint address set.
4	REG	r =	Breakpoint address set by exiting to another function.
5	BKPT	.b.P = 0044	Breakpoint address displayed.
6	BKPT	b.P =	Breakpoint cleared.

Set Breakpoint at Address H'0044' and then clear it.

STEP

FUNCTION: Causes the 2650 to execute a single instruction and return to the MONITOR mode, displaying the address of the next instruction to be executed on the monitor display.

PROCEDURE:

1. Enter the address of the first instruction to be executed as described under DISPLAY AND ALTER PROGRAM COUNTER command. ①

2. Depress the $\boxed{\text{STEP}}$ key. ② The INSTRUCTOR 50 will execute a single instruction and display the address of the next instruction to be executed and the data at that address.

3. At this point you may examine and alter memory and/or register values if desired by using the appropriate commands.

4. Continue as in step 2 to repeat the single-step operation. ③ ④

5. To exit the single-step mode, depress any function key. ⑤

6. Note that a breakpoint, if entered, is ignored during single-step operation.

The single-step sequencer and the forced jump logic are used in this mode of operation. Following is the sequence of operations executed by the monitor when the $\boxed{\text{STEP}}$ key is depressed:

a) The monitor SINGLE STEP flag is set.

b) Register contents previously stored upon entry to the monitor are restored to the 2650.

c) The monitor executes a "hidden single step" to determine how many cycles are contained in the instruction to be stepped.

d) The monitor permits execution of one user program instruction by counting the predetermined number of cycles.

e) The registers (R0 - R3, R1' - R3' and PSW) are saved.

f) The Program Counter is updated to the next instruction.

g) The address in the Program Counter and data at that address are displayed. The SINGLE STEP flag is cleared.

h) The monitor exits to the KBD SCAN routine to await user's input.

STEP

EXAMPLE

	KEY(S)	DISPLAY	COMMENTS
1	REG C 8 ENT/NXT	r =	Starting address H'0008' entered.
2	STEP	000A 42	Single step executed.* Next instruction is at H'000A', and op-code is H'42' (ANDZ, R2).
3	STEP	000B CC	Next instruction op-code is H'CC' (STRA, RO).
4	STEP	000E 20	Next instruction op-code is H'20' (EORZ, RO).
5	REG	r =	Exit single step.

Single step three instructions starting at address H'0008'

* Since the displayed address is two greater than the starting address (H'000A' - H'0008' =2), the first instruction executed was a two-byte instuction.

WRITE CASSETTE

FUNCTION: The WRITE CASSETTE command allows you to write programs and data from memory onto cassette tape. Any good quality audio cassette tape recorder may be used as the output device. The data transfer rate is approximately 300 bits per second.

PROCEDURE:

General Installation

- Connect the INSTRUCTOR 50's Cassette-Out Jack to the microphone (MIC) input of the cassette deck using the appropriate cable supplied with the INSTRUCTOR 50 package.

- Install tape in transport.

- Make certain that the tape is positioned so that previously recorded files will not be destroyed when the WCAS command is issued.

- Adjust recorder's input level control, if one is provided, to normal recording level.

Operation

1. Depress the $\boxed{\text{WCAS}}$ key ① to place the INSTRUCTOR 50 in the WRITE CASSETTE mode.

2. Enter the lower (beginning) address of the file to be written. ②

3. Depress the $\boxed{\text{ENT/NXT}}$ key ③ to set the lower address.

4. Enter the upper (ending) address of the file to be written. ④

5. Depress the $\boxed{\text{ENT/NXT}}$ key ⑤ to set the upper address.

6. Enter the program start address (the address at which you want your program to begin executing). ⑥

7. Depress the $\boxed{\text{ENT/NXT}}$ key ⑦ to set the start address.

8. Enter the file identification (ID) number. ⑧
   NOTE: The file ID may be any hex value between 00 and FF. If no ID is entered, the default file number is 00.

9. Place the cassette deck in the RECORD mode.

10. Depress $\boxed{\text{ENT/NXT}}$ key. ⑨ This starts a five-second delay prior to actual memory dump to tape. The INSTRUCTOR 50 flashes the FLAG Indicator at one-second intervals during this delay. The message HELLO is displayed ⑨ when data transfer to tape is completed.

WRITE CASSETTE

11. During the recording process, a visual indication of the 'dump' can be observed on the I/O port indicators by placing the I/O Port Address Select Switch in the EXTENDED (center) position.

Tape Deck Shutdown

- Turn the audio tape recorder off.
- If the tape deck has a counter, note its value for future reference.
- Disconnect tape deck and remove and store tape cartridge.

Error Messages

The INSTRUCTOR 50 will display the message 'Error 7' if the value of the specified upper address is less than the value of the lower address.

EXAMPLE

	KEY(S)	DISPLAY	COMMENTS
①	WCAS	L.Ad. =	Waiting for lower address of file to be written onto tape.
②	0	L.Ad. = 0	Lower address entered.
③	ENT/NXT	U.Ad. =	Lower address set. Waiting for upper address.
④	7 6	U.Ad. = 76	Upper address entered.
⑤	ENT/NXT	S.Ad. =	Upper address set. Waiting for start address.
⑥	1 0	S.Ad. =10	Start address entered.
⑦	ENT/NXT	.F =	Start address set. Waiting for file number.
⑧	1	.F = 1	File ID entered.
⑨	ENT/NXT	HELLO	File address set. Write data to cassette tape completed.

Write a file to tape with the following parameters:
File Number = 1
Beginning Address = 0
Ending Address = H'76'
Program Start Address = H'10'

ADJUST CASSETTE

FUNCTION: The ADJUST CASSETTE command allows you to adjust the output level of a cassette recorder for proper interface to the INSTRUCTOR 50 during a READ CASSETTE operation.

While most conventional audio cassette recorders are compatible for use with the INSTRUCTOR 50, the playback volume control must be accurately adjusted to ensure proper detection of data by the INSTRUCTOR 50. Otherwise, the data signal may be distorted (volume too high) or may drop below detection thresholds (volume too low).

PROCEDURE:

General Installation

1. Check to ensure that the cassette recorder's playback heads and transport mechanism are clean and free from any obstructions.

2. Install tape in transport and rewind to an area known to contain a previously recorded file.

3. Connect the INSTRUCTOR 50's PHONE jack to the cassette deck's PHONE or SPEAKER output jack using the appropriate cable supplied with the INSTRUCTOR 50 package.

Operation

1. Place the INSTRUCTOR 50 in the ADJUST CASSETTE mode by depressing the REG key on the function control keyboard followed by A on the hexadecimal keyboard. ①

2. Start playback of previously recorded data.

3. Adjust tape deck VOLUME or LEVEL control. The following three digits will be displayed intermittently during the adjustment process:

   U Increase volume

   d. Decrease volume

   - volume control adjusted correctly

4. When a minus sign (-) ③ is displayed, the audio cassette's playback volume is properly adjusted.

5. During the adjust process, the I/O Port indicators can also be used to observe data being read by the INSTRUCTOR 50 if the I/O Port Address Switch is placed in the EXTENDED (center) position. The display has the following significance:

All LEDs OFF	Indicates proper operation or no data.

Some negative number (LED bit 7 ON) — Indicates that the playback level is too low - not enough pulses.

Some positive number (LED bit 7 OFF) — Indicates that the playback level is too high. Tape "noise" is being detected - too many pulses.

6. When level is properly set, turn off the cassette deck.

7. Depress the $\boxed{\text{MON}}$ key ④ to exit from the ADJUST CASSETTE routine.

EXAMPLE

	KEY(S)	DISPLAY	COMMENTS
①	$\boxed{\text{REG}}$ $\boxed{\text{A}}$	$\boxed{\text{U}}$	Places INSTRUCTOR 50 in the ADJUST CASSETTE mode. Increase playback level.
②		$\boxed{\text{d.}}$	Decrease playback level.
③		$\boxed{\text{-}}$	Playback level properly set.
④	$\boxed{\text{MON}}$	$\boxed{\text{HELLO}}$	Exit ADJUST CASSETTE mode.

READ CASSETTE

FUNCTION: The READ CASSETTE command allows you to read files previously stored on cassette tape using the WRITE CASSETTE command and store these files in the specified RAM locations.

PROCEDURE:

General Installation

1. Check to ensure that the cassette recorder's playback heads and transport mechanism are clean and free from any obstructions.

2. Install tape in transport and rewind to desired file location.

3. Connect the INSTRUCTOR 50's PHONE jack to the cassette deck PHONE or SPEAKER output jack using the appropriate cable supplied with the INSTRUCTOR 50 package.

4. Adjust playback level to setting previously determined to be proper by ADJUST CASSETTE operation (See ADJUST CASSETTE command).

Operation

1. Depress the $\boxed{\text{RCAS}}$ key ① to place the INSTRUCTOR 50 in the READ CASSETTE mode.

2. Depress one or two hex digits ② corresponding to the file number desired to be read back.

   NOTE: The user may elect to read the first file encountered by omitting this step.

3. Depress the $\boxed{\text{ENT/NXT}}$ key ③ to set the file ID number.

4. Start the cassette deck in playback mode. The reading of data by the INSTRUCTOR 50 can be visually observed on the I/O Port indicators by placing the I/O Port Address Switch in the EXTENDED (center) position.

5. When the reading of the specified file is completed, the INSTRUCTOR 50 will display the HELLO message. ④

6. Turn off the audio cassette deck.

7. Data read from tape will be placed at consecutive memory locations starting at the beginning address specified when the file was created. The Program Counter (PC) will be set to the address specified as the program start address when the file was created.

READ CASSETTE

Error Messages

During the read-in process, any one of the following error messages may be displayed:

- Error 4 - Cassette Block Check Character (BCC) error
- Error 5 - Read Cassette Memory Write Error
- Error 6 - Read Cassette character from tape not ASCII HEX

EXAMPLE

	KEY(S)	DISPLAY	COMMENTS
1	RCAS	.F =	Places the INSTRUCTOR 50 in the READ CASSETTE mode. Waiting for file ID number.
2	1	.F = 1	File ID number entered.
3	ENT/NXT		Sets file ID number. Begins reading data into memory.*
4		HELLO	File is fully loaded into memory.

*Flashing I/O Port indicators at this point indicate that the file is being read.

RUN

FUNCTION: Terminates the monitor mode and causes program execution to begin at the address specified in the Program Counter. Program execution continues until 1) a breakpoint is encountered, 2) the RST or MON key is depressed, or 3) the user program executes a WRTC (Write to Port C) or HALT instruction.

The RUN command allows program execution to begin at any point in the user program. It is particularly valuable, when used in conjunction with a set breakpoint, for debugging sections of a program. When the RUN key is depressed, the INSTRUCTOR 50 performs the following actions:

1. If a breakpoint was set, the WRTC code is inserted at the specified breakpoint address and a monitor 'BREAKPOINT ENABLED' flag is set. This flag distinguishes a breakpoint 'WRTC' from any other 'WRTC' in the user program when control is returned to the USE monitor by the forced jump logic upon execution of a WRTC instruction.

2. The processor registers are restored to the last values existing when control was returned to the USE monitor after a breakpoint or single step, or to the values specified by you in a DISPLAY AND ALTER REGISTERS operation.

3. The INSTRUCTOR 50 switches to the execution mode by jumping to the address specified in the Program Counter. This address will be the address of the next instruction following a breakpoint or single step, or the address specified by you in a DISPLAY AND ALTER PROGRAM COUNTER operation.

RESET

FUNCTION: When the RST (RESET) key is depressed, current INSTRUCTOR 50 activity is terminated immediately, and the processor begins program execution at address H'0000. Breakpoint and single-step flags, if set, are ignored. A high (logic one) level appears on the expansion connector RESET pin for as long as the key remains depressed.

When the RESET key is used to initiate program execution from location H'0000', the initial processor register values are unknown, and a breakpoint, if previously specified, is not inserted in the user program. Program execution continues until any one of the following occurs:

1. The RESET key is depressed again.

2. A HALT instruction (H'40') is executed. Upon detection of a HALT instruction, the processor halts until the RESET key is depressed again or, if the Interrupt Inhibit PSW bit was not set, until an interrupt occurs.

3. A WRTC Instruction is executed or the MON key is depressed. Control is transferred to the USE monitor and the HELLO message is displayed. When control is returned to the monitor, the address of the last memory fetch is saved in the Program Counter, and register values are saved in monitor RAM. These may be examined by using the appropriate commands.

4. The processor's PAUSE input is raised high via the expansion connector. When this occurs, the RUN indicator light is extinguished. Program execution will begin at the next instruction when PAUSE goes low.

ERROR MESSAGES

The USE monitor incorporates extensive error checking firmware. If an error is encountered while attempting to execute a command, a message of the form 'Error n' is presented on the monitor display. Error messages are summarized in Table 5.1.

●	Error 1	BREAKPOINT CANNOT BE SET.
●	Error 2	INVALID COMMAND.
●	Error 3	ALTER OR PATCH MEMORY WRITE ERROR.
●	Error 4	CASSETTE BCC ERROR.
●	Error 5	READ CASSETTE MEMORY WRITE ERROR.
●	Error 6	CHARACTER FROM TAPE NOT ASCII HEX.
●	Error 7	START ADDRESS GREATER THAN STOP ADDRESS.
●	Error 8	KEYBOARD HAS 2 KEYS IN COLUMN DOWN.
●	Error 9	NEXT SINGLE STEP IS INTO MONITOR.

TABLE 5.1: Error Messages

Additional information on each of the above error messages is presented in the following paragraphs.

Error 1 *BREAKPOINT CANNOT BE SET*

The display message Error 1 indicates that an attempt was made to set a breakpoint at a memory address which is not RAM. A breakpoint is entered by inserting the WRTC,R0 code H'B0' into the memory address specified. A read-after-write check is then performed. If this test fails, the error message is displayed.

Error 2 *INVALID COMMAND*

The display message Error 2 indicates that an incorrect command sequence was entered via the keyboard.

Error 3 *ALTER OR PATCH MEMORY ERROR*

The display message Error 3 indicates that an attempt was made to change the data at a memory address which is not RAM. When changing memory data during an Alter Memory or Patch Memory operation, a read-after-write check is performed. If this test fails, the error message is displayed.

Error 4 *CASSETTE BCC ERROR*

When data is written on tape with the WRITE CASSETTE command, a Block Check Character (BCC) is appended to the end of the file. The BCC is recalculated when data is read back with a READ CASSETTE command and compared with the BCC recovered from the tape. If the BCC's do not match, the message Error 4 is displayed, indicating that some problem has occurred in reading the tape.

Error 5 *READ CASSETTE MEMORY WRITE ERROR*

Data read back from the tape is stored in the INSTRUCTOR 50 at consecutive memory locations starting at the address specified in the tape file. A read-after-write check is performed on each byte stored. If the test fails, the message Error 5 is displayed.

Error 6 *CHARACTER FROM TAPE NOT ASCII HEX*

Data written on tape uses the ASCII code for the characters 0 through F. The display message Error 6 indicates that a non-hex character was recovered from the tape. Correct adjustment of playback level should be verified using the ADJUST CASSETTE command.

Error 7 *START ADDRESS GREATER THAN STOP ADDRESS*

The display message Error 7 indicates that the start address in the WRITE CASSETTE command is greater that the specified stop address. The operation cannot be performed.

Error 8 *KEYBOARD HAS 2 KEYS IN COLUMN DOWN*

The Error 8 message is displayed when the monitor detects that two keys are depressed simultaneously. The monitor cannot decode the action desired.

Error 9 *NEXT SINGLE STEP IS INTO MONITOR*

Single-step operation in the memory area reserved for the USE monitor (H'1000' - H'1FFF') is not permitted and will cause unpredictable results if executed. The display message Error 9 is a warning that such a single-step operation was attempted.

6. USING THE INSTRUCTOR 50

Restrictions on Using the 2650 Instruction Set

When writing programs, the INSTRUCTOR 50 user has the complete 2650 microprocessor instruction set at his disposal. However, because of the interaction between the USE monitor and user hardware and software, certain restrictions must be observed:

1) The USE monitor reserves the WRTC, Rx instruction (H'B0' - H'B3') to indicate the location of a breakpoint in a user program. If this instruction is executed in a user program, control of the system will return to the monitor, and the message HELLO will be displayed.

2) If a HALT instruction (H'40') is executed, processor operation will terminate. This is indicated by the RUN indicator being extinguished. The only ways to reinitiate operation are to depress the RST key or, if interrupts were not inhibited, to cause an interrupt by depressing the INT key.

   If a breakpoint is set at a HALT instruction location, the monitor will prevent execution of the HALT, and normal operation will continue.

3) The top of memory page zero is occupied by the USE monitor program. Therefore, the ZBSR and ZBRR instructions with negative displacements should not be used unless entry into the monitor program is desired. The same applies to interrupt vectors with negative displacements.

4) The USE monitor uses three levels of the 2650 subroutine Return Address Stack (RAS) in its operation. Since the RAS is limited to eight levels, user programs being developed under control of the USE monitor should be limited to a maximum of five levels of subroutines, including interrupt levels.

Using Interrupts

Interrupts provide a method of interfacing a synchronous program to asynchronous external events. An Interrupt Request forces the 2650 to temporarily suspend execution of the program currently running and branch to an interrupt service routine. Upon completion of the interrupt service routine, the 2650 resumes execution of the interrupted program.

The INSTRUCTOR 50 provides three methods of interrupting the 2650. The first method is a manual interrupt using the INT key on the function keyboard. The second method uses a 60Hz signal derived from the INSTRUCTOR 50's power supply to generate interrupt requests once every 16.7 ms. This option accommodates user programs that require a real-time clock. (For European systems, the real-time clock interrupts occur at a 50Hz rate or once every 20 ms). The third method of interrupting the INSTRUCTOR 50 is via the S100 bus interface. This section decribes the 2650's interrupt mechanism and provides details on selecting the interrupt options.

The 2650's interrupt mechanism can be selectively enabled or disabled at various points in a user program by setting or clearing the Interrupt Inhibit (II) bit of the processor's Program Status Word (PSW). If the Interrupt Inhibit bit has been set, the 2650 ignores interrupt requests. The Interrupt Inhibit bit may be cleared (thus enabling interrupts) in any of the following four ways:

1) By resetting the processor (depressing the RST key);

2) By executing a Clear Program Status Upper (CPSU) instruction with the proper mask value;

3) By executing a Return from Subroutine and Enable Interrupt (RETE) instruction; or

4) By executing a Load Program Status, Upper (LPSU) instruction.

The interrupt mechanism of the 2650 operates with a vectored interrupt. When the processor accepts an interrupt request, it responds by issuing an INTerrupt ACKnowledge (INTACK). Upon receipt of INTACK, the interrupting device responds by placing an "interrupt vector" on the 2650 data bus. This vector is used as the address, relative to byte zero, page zero, of a branch to subroutine instruction. The interrupt vector may specify either direct or indirect addressing. A vector that specifies direct addressing causes the 2650 to execute a subroutine branch to the address specified by the vector. If an indirect address is specified, the interrupt vector points to the first of two successive memory locations (interrupt vector and interrupt vector + 1) where the address of the interuupt subroutine is stored. In this case, the processor first fetches the two interrupt subroutine address bytes and then branches to the subroutine. Thus, a direct interrupt vector transfers the program to any location from -64 to +63 relative to byte zero, page zero, and an indirect interrupt vector can transfer the program to any location within the 2650's 32K addressing range.

If the Interrupt Inhibit bit has been cleared, the INSTRUCTOR 50 responds to an interrupt request with the following sequence of events:

1) The 2650 completes execution of the current instruction.

2) The processor sets the Interrupt Inhibit bit of the PSW (=1).

3) The first byte of a Zero Branch to Subroutine Relative (ZBSR) instruction is inserted in the 2650's internal instruction register.

4) The processor issues INTACK and waits for an interrupt vector to be returned on the data bus.

5) The INSTRUCTOR 50's interrupt logic places the interrupt vector (H'07' or H'87') on the data bus. Whether the interrupt vector specifies

direct (H'07') or indirect (H'87') addressing is determined by the setting of the Direct/Indirect switch on the front panel. If the switch is in the Direct position, the next instruction executed is the instruction at address H'07'. If the switch is in the Indirect position, the next instruction executed is at the address contained in H'07' and H'08'.

6) The 2650 executes the ZBSR instruction. The address of the next instruction in the interrupted program is stored in he 2650's internal subroutine address stack, and the stack pointer is incremented.

7) When the interrupt subroutine is terminated with an RETE or RETC instruction, the 2650 decrements the stack pointer, replaces the current value of the Program Counter with the address previously stored in the subroutine stack, and resumes execution of the interrupted program.

Since the INSTRUCTOR 50 interrupt logic vectors interrupt requests through memory address H'07', user programs that support direct interrupts must place the first byte of the interrupt subroutine at this address. If indirect subroutines are used, the address of the interrupt subroutine must be stored at memory locations H'07' and H'08'.

As interrupts may occur at any point in a user program, it is entirely possible that they will affect the contents of the 2650's internal registers with unpredictable results for the interrupted program. This probelm can be solved in two ways. The first way is to tightly control the portions of a user program that can be interrupted by selectively setting and clearing the Interrupt Inhibit (II) bit in the PSW. The second method is to save the 2650's internal registers and Program Status Word upon entering the interrupt subroutine and restoring them before returning from the subroutine.

The INSTRUCTOR 50's interrupt modes can be selected by a combination of switch settings and a jumper option on the printed circuit board. As mentioned previously, the Direct/Indirect switch on the INSTRUCTOR 50's front panel determines whether the interrupt vector generated by the interrupt logic specifies direct or indirect addressing. Whether the 2650 responds to the INT key or the 60 Hz real-time clock is determined by the setting of a slide switch located at the bottom of the INSTRUCTOR 50 case. Optionally, devices external to the INSTRUCTOR 50 can generate interrupt requests via the S100 bus interface. To accomplish this, a jumper option described in the last part of this section is used.

Following are two programming examples that make use of the INSTRUCTOR 50's interrupt facilities:

Example 1 - Direct Interrupt

This example is a complete program that first clears the parallel I/O port lights and then maintains a binary counter at the I/O port lights. The count is incremented each time the INT key is depressed. Prior to running this program, you must place the Direct/Indirect switch in the Direct position, and the I/O port address select switch in the Non-Extended position.

Address	Data	Instruction Mnemonic	Comment
0000	76,20	PPSU H'20'	Set II - inhibit interrupts.
0002	75,08	CPSL H'08'	Operations without carry.
0004	1F,00,0A	BCTA,UN,H'000A'	Branch over interrupt subroutine.
0007	84,01	ADD1,R0,H'01'	Increment R0 (counter).
0009	17	RETC,UN	Return from interrupt subroutine.
000A	20	EORZ,R0	Clear R0 (counter).
000B	F0	WRTD,R0	Write R0 to the lights (non-extended).
000C	74,20	CPSU H'20'	Clear II (open interrupt window).
000E	76,20	PPSU H'20'	Set II (close interrupt window).
0010	1F,00,0B	BCTA,UN H'000B'	Branch back to WRTD.

Example 2 - Indirect Interrupt

This example performs the same function as above but uses indirect interrupts. The interrupt subroutine starts at address H'100'. This address is contained in program locations H'07' and H'08'. Prior to running this program, you must place the Direct/Indirect switch in the indirect position but retain the I/O port address select switch in the non-extended position.

Address	Data	Instruction Mnemonic	Comments
0000	76,20	PPSU H'20'	Set II - Inhibit Interrupts.
0002	75,08	CPSL H'08'	Operations without carry.
0004	1F,00,09	BCTA,UN H'0009'	Branch over interrupt address.
0007	01,00	ACON H'0100'	Interrupt routine address.
0009	20	EORZ,R0	Clear counter.
000A	F0	WRTD,R0	Write R0 to the lights.
000B	74,20	CPSU H'20'	Clear II - enable interrupts.
000D	1F,00,0D	BCTA,UN H'000D'	Loop forever.
0100	84,01	ADD1,R0 H'01'	Increment counter.

```
0102          F0           WRTD,R0               Write R0 to the lights.

0103          37           RETE,UN               Return and enable interrupts.
```

Using the I/O Switches and Lights

The 2650 provides several methods for monitoring the status of and controlling the operation of external I/O devices. One such method unique to the 2650 is a serial I/O port formed by the SENSE input pin and the FLAG output pin on the processor. The 2650 also has provisions for two types of parallel I/O instructions, called extended and non-extended. The non-extended I/O instructions are one-byte instructions that allow a user program to read from and write to two eight-bit I/O ports: port C and port D. The two-byte extended I/O instructions expand the 2650's I/O capabilities to 256 bidirectional I/O ports.

In addition to the 2650 instructions specifically intended for I/O operations, you may choose to use the memory mapped I/O mode. This mode is implemented by assigning a memory address to an I/O device. While a memory mapped I/O port requires more decode logic than either an extended or a non-extended port, it can ba accessed by the full range of 2650 memory referencing instructions. (Refer to Chapter 9 for a description of the 2650 I/O control modes.)

The INSTRUCTOR 50 includes features that demonstrate all of the 2650's I/O modes. These features are described as follows:

FLAG and SENSE I/O

The 2650's FLAG and SENSE pins are associated with the flag and sense bits of the processor's internal Program Status Word (PSW). The SENSE bit of the PSW always reflects the signal level on the SENSE pin. Likewise, the level on the FLAG pin always reflects the current value of the flag bit in the PSW.

The user may manually control the value of the sense bit in the PSW using the SENS key on the function control keyboard. When the SENS key is depressed, the SENSE bit is a one. Otherwise, the SENSE bit is a zero.

The INSTRUCTOR 50's Flag indicator on the front panel is driven by the FLAG pin on the 2650, providing a visual indication of the FLAG bit's current value. The FLAG light is on if the FLAG bit is a one, and the light is off if the FLAG bit is a zero.

Non-Extended I/O

The 2650 can control two bidirectional I/O ports with four single-byte instructions: WRTC, WRTD, REDC and REDD. These instructions move data between port C, port D and the 2650's internal registers.

The INSTRUCTOR 50's front panel parallel I/O port can be assigned as non-extended port D by placing the Port Address select switch in the NON-EXTENDED position. In this position, the I/O port can be accessed with the WRTD and REDD instructions. This allows you to manually enter data with the input switches by including a REDD instruction in your program. Similarly, your program can write a data value to the output LEDs by executing a WRTD instruction.

Extended I/O

The 2650 can control up to 256 bidirectional I/O ports with the double-byte instructions WRTE and REDE. The second byte of these instructions specifies the extended I/O port address. The INSTRUCTOR 50's parallel I/O port can be assigned as extended address H'07' by placing the Port Address switch in the EXTENDED position. In this mode, the parallel I/O port can be accessed with WRTE and REDE instructions that specify an extended address H'07' in their second byte.

Memory Mapped I/O

Memory mapped I/O is simply a matter of decoding a memory address for enabling an I/O port. To demonstrate this I/O mode, the INSTRUCTOR 50's Port Address select switch can be placed in the MEMORY position. This assigns the parallel I/O port a memory address of H'OFFF'. Thus, any memory reference instruction that specifies H'OFFF' as the source or destination will access the front panel parallel I/O port. When an instruction reads location H'OFFF', the value contained in the specified register will appear in the port output LEDs.

CALLING MONITOR SUBROUTINES

Now that you are familiar with the 2650 instruction set and have successfully entered a few simple programs, you are undoubtedly ready and anxious to make use of some of the more powerful features provided by the INSTRUCTOR 50. For example, you might want to write a decimal add program using the INSTRUCTOR 50's keyboard and eight-digit display. By calling subroutines within the monitor program, the display can be used to request the two numbers to be added, and the hex keyboard can be used to enter the numbers. After the two numbers have been entered, and their sum calculated, another monitor subroutine can be called to display the results of the addition. This section describes these subroutines and provides examples in their use.

In addition to subroutines that provide easy access to the INSTRUCTOR 50's keyboard and display, the monitor program contains other subroutines that are useful in writing application programs. Refer to the program listing in Chapter 11 for additional information on other subroutines.

The monitor subroutines are called with Zero Branch to Subroutine Relative (ZBSR) instructions. The ZBSR instruction specifies a subroutine relative to byte zero, page zero. The relative addressing range is -64 to +63. Since the 2650 uses an 8K page addressing scheme, ZBSR instructions with a negative offset (relative address) wrap back around to the top of the first 8K page. The top of the first 8K page in the INSTRUCTOR 50 is located within the monitor program and contains a table of indirect subroutine addresses. Thus, the monitor subroutines can be called by ZBSR instructions that specify indirect addressing and have the negative offset that points to the desired subroutine. The addresses required to call the various monitor subroutines are included in the description of each subroutine.

The subroutine descriptions include a list of the 2650 registers used in their execution. Unless otherwise specified, the contents of these registers will contain meaningless data when the subroutine returns control to the user program. Therefore, registers that contain important user program information must be stored in a memory location before the monitor subroutine is called.

When calling monitor subroutines, caution must be exercised to avoid overflowing the 2650's internal 8-level subroutine stack. Since some of the user-accessible subroutines call other subroutines within the monitor program, each subroutine description includes the number of other subroutines called during its execution. This information allows you to calculate the number of subroutine stack levels required by your program and insures that this number never exceeds eight.

MOVE SUBROUTINE

Calling Instruction:

Mnemonic	Hex Value
ZBSR *MOV	BB,FE

Registers Used:

R1 = Message Pointer - 1 (high-order byte)
R2 = Message Pointer - 1 (low-order byte)

Subroutine Levels Used: 0

Function:

MOVE fetches an eight-byte message within the user's program and stores the eight bytes in the monitor's display buffer. When combined with the DISPLAY subroutine, MOVE allows you to write messages on the INSTRUCTOR 50's eight-digit display. Any of the INSTRUCTOR 50's characters can be used in assembling a message.

Operation:

Before calling MOVE, you must store an eight-byte message within your program. The location of the sequential message bytes is transferred to MOVE by storing the address of the first message byte in R1 and R2 prior to calling the subroutine. Because of the algorithm used to implement the MOVE subroutine, it is necessary to subract one from the message pointer before it is stored in R1 and R2. Following is an example of the MOVE subroutine call and a list of the hexadecimal values for the INSTRUCTOR 50's display characters.

EXAMPLE OF MOVE SUBROUTINE CALL

```
Address     Data       Instruction Mnemonic        Comments

   •
   •
   •
0010        05,00      LODI, R1 H'00'              Load message pointer -1 in  R1
                                                   and R2.

0012        06,FF      LODI, R2 H'FF'              (H'100' - 1 = H'00FF').


0014        BB,FE      ZBSR *MOV                   Call MOVE.  The message bytes
                                                   stored in locations  0100-0107
                                                   are transferred to  the  moni-
                                                   tor's display buffer.
   •
   •
   •
0100        17                                     = blank (first  byte  of  mes-
                                                   sage)

0101        14                                     = H
0102        0E                                     = E
0103        11                                     = L
0104        11                                     = L
0105        00                                     = 0
0106        17                                     = blank
0107        17                                     = blank  (last  byte  of  mes-
                                                   sage).
```

Hex Values of Display Characters

Character	Value	Character	Value	Character	Value
*0.0	H'00'	*A	H'0A'	*H	H'14'
*1.I	H'01'	*B	H'0B'	*0	H'15'
*2	H'02'	*C	H'0C'	*=	H'16'
*3	H'03'	*D	H'0D'	*BLANK	H'17'
*4	H'04'	*E	H'0E'	*J	H'18'
*5.5	H'05'	*F	H'0F'	*-	H'19'
*6.G	H'06'	*P	H'10'	*	H'1A'
*7	H'07'	*L	H'11'	*Y	H'1B'
*8	H'08'	*U	H'12'	*N	H'1C'
*9	H'09	*R	H'13'		

DISPLAY SUBROUTINE

Calling Instruction:

Mnemonic	Hex Value
ZBSR *DISPLY	BB,EC

Registers Used:

R0,R1,R2,R3

On entry R0 = Display Command
On exit R0 = Key Value (optional)

Subroutine Levels Used: 0

Function:

When used with the MOVE subroutine, DISPLAY writes messages on the INSTRUCTOR 50's eight-digit display. DISPLAY reads the message stored in the monitor's display buffer with MOVE and writes the message on the display. Optionally, DISPLAY can be used to read the function and data keyboards and return the value of a depressed key.

Operation:

DISPLAY has three modes of operation that are selected by writing a command byte in R0 prior to calling the subroutine. The DISPLAY commands and the functions they specify are summarized below:

Value Placed in R0	Function
H'00'	Displays message in display buffer until a function or data key is depressed. Returns the value of the depressed key in R0.
H'01'	Makes one pass through the DISPLAY subroutine and does not read the keyboards. A single pass through the DISPLAY subroutine will not produce a visible display. Hence, when this command is used, it should be part of a loop that calls DISPLAY a sufficient number of times to illuminate the message.
H'80'	This command is identical to the H'00'command except that the decimal point of the most-significant (far-left) digit is illuminated.

The function and data key values returned in R0 when operating in response to commands H'00' and H'80' are listed in the following table. This is followed by an example of the MOVE and DISPLAY subroutine calls that displays the message HELLO until the [RUN] key is depressed.

Data Values for Command and Data Keys

Key	Value	Key	Value	Key	Value
0	H'00'	8	H'08'	WCAS	H'80'
1	H'01'	9	H'09'	BKP	H'81'
2	H'02'	A	H'0A'	RCAS	H'82'
3	H'03'	B	H'0B'	REG	H'83'
4	H'04'	C	H'0C'	STEP	H'84'
5	H'05'	D	H'0D'	MEM	H'85'
6	H'06'	E	H'0E'	RUN	H'86'
7	H'07'	F	H'0F'	ENT/NXT	H'87'

Example of Move and Display Subroutine Calls

Address	Data	Instruction Mnemonic	Comment
0010	05,00	LODI,R1 H'00'	Place message table pointer minus one in R1 and R2.
0012	06,FF	LODI,R2 H'FF'	
0014	BB,FE	ZBSR *MOV	Call the Move subroutine.
0016	04,00	LODI,R0 H'00'	Place command byte in R0.
0018	BB,EC	ZBSR *DISPLY	Call the DISPLAY subroutine.
001A	E4,86	COMI,R0 H'86'	Compare returned key code to RUN code. If equal, branch to address H'XXXX'.
001C	1C,XX,XX	BCTA,EQ H'XXXX'	
001F	1C,00,16	BCTA,UN H'0016'	If not equal, loop back and wait for next key.
•			
•			
•			
0100	17		First byte of message table = blank
0101	14		= H
0102	0E		= E
0103	11		= L
0104	11		= L
0105	00		= 0
0106	17		= blank
0107	17		Last byte of message table = blank.

USER DISPLAY SUBROUTINE

Calling Instruction:

Mnemonic	Hex Value
ZBSR *USRDSP	BB,E6

Registers Used:

R0, R1, R2, R3

```
On entry R3 = Display Command
         R1 = Message Pointer -1 (high order)
         R2 = Message Pointer -1 (low order)
On exit  R0 = Key value (optional)
```

Subroutine Levels Used: 2

Function:

USER DISPLAY combines the functions of MOVE and DISPLAY. That is, USER DISPLAY moves an eight-byte message from a user program to the display buffer and then displays the message. As with DISPLAY, this subroutine may be used to read the function and data keyboards.

Operation:

Before calling USER DISPLAY , you must load the first address of your message table (-1) in R1 and R2. Additionally, R3 must be loaded with the desired command as in the DISPLAY subroutine.

The following example of a USER DISPLAY subroutine call displays the message HELLO until the RUN key is depressed. (This example is functionally equivalent to the example for the DISPLAY subroutine).

Example of a USER DISPLAY Subroutine Call

Address	Data	Instruction Mnemonic	Comment
0010	05,00	LODI,R1 H'00'	Place message table pointer -1 in R1 and R2.
0012	06,FF	LODI,R2 H'FF'	
0014	07,00	LODI,R3 H'00'	Place command byte in R3.
0016	BB,E6	ZBSR *USRDSP	Call USER DISPLAY.
0018	E4,86	COMI,RO H'86'	Compare returned key value to RUN's value.
001A	1C,XX,XX	BCTA,EO H'XXXX'	Branch to XX,XX if equal.
001D	1F,00,10	BCTA,UN H'0010'	If not equal, loop back and get another key.
• • •			
0100	17		First byte of message table = blank
0101	14		= H
0102	0E		= E
0103	11		= L
0104	11		= L
0105	00		= 0
0106	17		= blank
0107	17		Last byte of message table = blank

NIBBLE SUBROUTINE

Calling Instruction:

Mnemonic	Hex Value
ZBSR *DISLSD	BB,F4

Registers Used:

```
R0 and R2
On entry:  R0 = byte (high-order nibble, low-order nibble)
On exit:   R0 = high-order nibble
           R1 = low-order nibble
```

Subroutine Levels Used: 1

Function:

NIBBLE takes an eight-bit byte and separates it into two bytes, each containing one of the original four-bit nibbles. This subroutine is useful in user programs that display a register or memory data value on the INSTRUCTOR 50 display. The NIBBLE subroutine is an invaluable aid in converting binary data to hexadecimal values.

Operation:

The byte to be separated in passed to NIBBLE in R0. NIBBLE then takes the least-significant four bits (low-order nibble) from R0 and places them in the four least-significant bits of R1. When NIBBLE returns program control to your program, R0 contains the low-order nibble, and R1 contains the high-order nibble. The most - significant four bits of both R0 and R1 contain zeros. A functional example of NIBBLE is shown below. This is followed by an example of a NIBBLE subroutine call.

Functional Example of NIBBLE

On entry: R0 = F3

On exit: R0 = 0F
R1 = 03

Example of NIBBLE subroutine Call

Address	Data	Instruction Mnemonic	Comment
0000	70	REDD,R0	Read I/O port (Non-Extended) into R0.
0001	BB,F4	ZBSR *DISLSD	Call NIBBLE subroutine.
0003	CD,01,07	STRA,R1 H'01,07'	Store low-order nibble in message table.
0006	CC,01,06	STRA,R0 H'01,06'	Store high-order nibble in message table.
0009	05,00	LODI,R1 H'00'	Place message table pointer (-1) in R1 and R2.
000B	06,FF	LODI,R2 H'FF'	
000D	04,00	LODI,R0 H'00'	Place display command in R0.
000F	BB,E6	ZBSR*USRDSP	Call USER DIAPLAY subroutine. Displays previous Port D value. Allows new I/O value to be set up in switches. Exits when any key is depressed.
0011	1B,6D	BCTR,UN H'6D'	Loop back to 0000 and get new I/O value.
0100	13		= "R" (first byte of message table).
0101	0E		= "E"
0102	0D		= "D"
0103	0D		= "D"
0104	16		= "="
0105	17		= "blank"
0106	17		= "blank" (high-order nibble will be stored here).
0107	17		= "blank" (low-order nibble will be stored here).

INPUT DATA SUBROUTINE

Calling Instruction:

Mnemonic	Hex Value
ZBSR *GNP	BB,FA

Registers Used:

On entry: R0 = Input Command

On exit:
R0 = Two Data Key Values
R1 = Two Data Key Values (optional)
R2 = Function Key Value
R3 = Data Entered Indicator

Subroutine Levels Used: 1

Function:

INPUT DATA displays the contents of the display buffer and scans the data keyboard for data entry. As data is keyed in, the subroutine writes the input data in the least-significant digits of the display. When a function key is depressed, USER DISPLAY returns to the main program with the input data and function key values in the 2650's internal registers.

Operation:

INPUT DATA has two selectable modes of operation. Mode selection is made by writing an input command byte in R0 before calling the subroutine. The input command bytes and the functions they specify are listed as follows:

Value Placed in R0	Function
H'00'	Displays a four-digit message and accepts four digits of data. As each data value is keyed in, it is displayed in the least-significant (right-most) display digit, and previously entered values are shifted left. Data entry is terminated and program control is returned to the user program when a function key is depressed. If less than four data values are entered, zeros are inserted in the non-entered digit positions.
H'01'	Identical to H'00' except that a five-digit message is displayed, and two digits of data are input from the data keyboard.

The four or five-digit message to be displayed by INPUT DATA must be placed in the monitor's display buffer before INPUT DATA is called. The message characters displayed are taken from the first four or five bytes of the eight-byte message table transferred to the display buffer by the MOVE subroutine.

The data values input to INPUT DATA are returned to the main program in R0 for the two-digit input mode and to R0 and Rl for the four-digit input mode. In the two-digit input mode, the most-significant data value entered is returned to the most-significant nibble of R0, and the least-significant data value is returned to the least-significant nibble of R0. In the four-digit input mode, the two most-significant data values are returned in Rl, and the two least-significant data values are returned to R0.

When data entry is terminated with a function key depression, the value of the function key is returned to R2, and a data entered indicator value is returned to R3. If no data has been entered before a function key is depressed, R3 will contain the value H'7F'. If data has been entered, R3 will contain a value of H'00'. The following example illustrates how data is returned to the user program. (This is followed by an example of an INPUT DATA call.)

Example of Data Entry and Register Contents on Return

From Input Data Subroutine

Keys	Display	Comments
(1) (2) (3) (4)	PLUS	Initial display on subroutine entry.
(RUN)	PLUS1234	Data values entered.
		Data entry terminated and program control returned to user program.

Register Contents on Return from Input Data Subroutine

Register	Contents	Comments
R0	H'34'	Least-significant data values
R1	H'12'	Most-significant data values
R2	H'86'	Value of RUN key
R3	H'00'	Indicates valid data in R0 and R1

Example of Input Data Subroutine Call

Address	Data	Instruction Mnemonic	Comments
0050	05,00	LODI,R1 H'00'	Place message table pointer (-1) in R1 and R2.
0052	06,FF	LODI,R2 H'FF'	
0054	BB,FE	ZBSR *MOV	Call MOVE to transfer message table to display buffer.
0056	04,00	LODI,R0 H'00'	Place input command in R0 (H'00' = 4 digits).
0058	BB,FA	ZBSR *GNP	Call INPUT DATA subroutine.
•			
•			
•			
0100	10		First byte of message table = P
0101	11		= L
0102	12		= U
0103	05		= S
0104	17		= blank
0105	17		= blank
0106	17		= blank
0107	17		Last byte of message table = blank.

NOTE: Since the input command requests four digits of input data, only the first four message table bytes (0100 - 0103) are displayed.

MODIFY DATA SUBROUTINE

Calling Instruction:

Mnemonic	Hex Value
ZBSR *GNPA	BB,FC

Registers Used:
R0,R1,R2,R3

On entry: R0 = Input command

On exit:
R0 = Two Data Key Values
R1 = Two Data Key Values
R2 = Function Key Values
R3 = Data Entered Indicator

Subroutine Levels Used: 1

Function:

MODIFY DATA is very similar to INPUT DATA. The major difference is that the initial display message can use all eight digit positions on the INSTRUCTOR 50 display panel. MODIFY DATA enables a program to display data values that were previously entered with INPUT DATA and allows these data values to be modified.

Operation:

As with INPUT DATA, MODIFY DATA has two modes of operation that are selected by writing an input command byte in R0 prior to calling the subroutine. The input commands and their respective functions are listed below:

Value Placed in R0	Resulting Function
H'00'	Displays an eight-digit message and accepts four digits of data. After the first data key is depressed, the four least-significant digits of the display are cleared. Each new data value entered is then displayed in the least-significant display digit, and previously entered values are shifted left. Data entry is terminated when a function key is depressed.
H'01'	Identical to H'00' except that when the first data key is depressed, the three least-significant display digits are cleared, and two digits of data may be entered.

The eight-digit message to be displayed must be transferred to the monitor's display buffer with MOVE before MODIFY DATA is called. The values for the data entered indicator are the same for MODIFY DATA as for INPUT DATA. That is, R3 contains H'00' if R0 and R1 contain valid data and H'7F' if a function key was depressed before data was entered. The following example illustrates operation of MODIFY DATA. This is followed by an example of a MODIFY DATA subroutine call.

Data Entry and Register Contents on Return

From Modify Data

Data Input Key	Display	Comments
	JOB = 01	Initial display on subroutine entry.
(2)	JOB = 2	Least-significant three digits are cleared and new data is displayed.
(RUN)		Data entry is terminated, and program control is returned to user program.

Register Contents on Return from MODIFY DATA

Register	Contents	Comments
R0	H'02'	Data entered is returned in R0.
R1	H'XX'	Data in R1 is meaningless.
R2	H'86'	Value of RUN key.
R3	H'00'	Indicates valid data in R0.

Example of MODIFY DATA Subroutine Call

Address	Data	Instruction Mnemonic	Comment
0034	05,00	LODI,R1 H'00'	Place message table pointer (-1) in R1 and R2.
0036	06,FF	LODI,R2 H'FF'	
0038	BB,FE	ZBSR *MOV	Call MOVE to transfer the message table to the display buffer.
003A	04,01	LODI,R0 H'01'	Place input command in R0 (H'01' = 2 digits).
003C	BB,FC	ZBSR *GNPA	Call MODIFY DATA subroutine.
•			
•			
•			
0100	17		First byte of message table = "blank"
0101	18		= "J"
0102	15		= "O"
0103	0B		= "B"
0104	16		= "="
0105	17		= "blank"
0106	00		= "0" previously entered data value.
0107	01		= "1" previously entered data value.

Jumper Options

The INSTRUCTOR 50's versatility is enhanced by jumper options on the printed circuit board. These options allow you to modify the system's basic configuration. The jumpers are accessible through cutouts at the bottom of the INSTRUCTOR 50's plastic housing. Figure 6.1 identifies the location of the various jumpers and their configuration. The factory supplied configurations are identified by asterisks (*) in the jumper pin description tables.

Jumper A - Interrupt Selection

As described previously, a switch at the bottom of the INSTRUCTOR 50 allows you to select interrupts from the interrupt key on the function keyboard or from the input line frequency clock. Jumper 'A' provides additional interrupt flexibility by allowing interrupt requests from external logic via the bus interface connector. If this option is exercised, interrupt requests from external logic will result in a vectored interrupt through memory address H'0007'. The setting of the DIRECT/INDIRECT switch on the front panel determines whether an externally generated interrupt request results in a direct or indirect subroutine branch. The pin descriptions for jumper 'A' are defined in the following table:

JUMPER A Pin Descriptions

Pins Connected	Description
1-2*	Normal operation. The 2650 recognizes interrupt requests from the interrupt key or the real-time clock, depending on the position of S6.
2-4	Bus interface. The 2650 recognizes interrupt requests from the interface signal VIO (pin 4). The interrupt latch is set on the rising edge of VIO.
2-39 3-4	Bus interface inverted. This configuration is identical to the 2-4 option except that the interrupt latch is set on the falling edge of VIO.

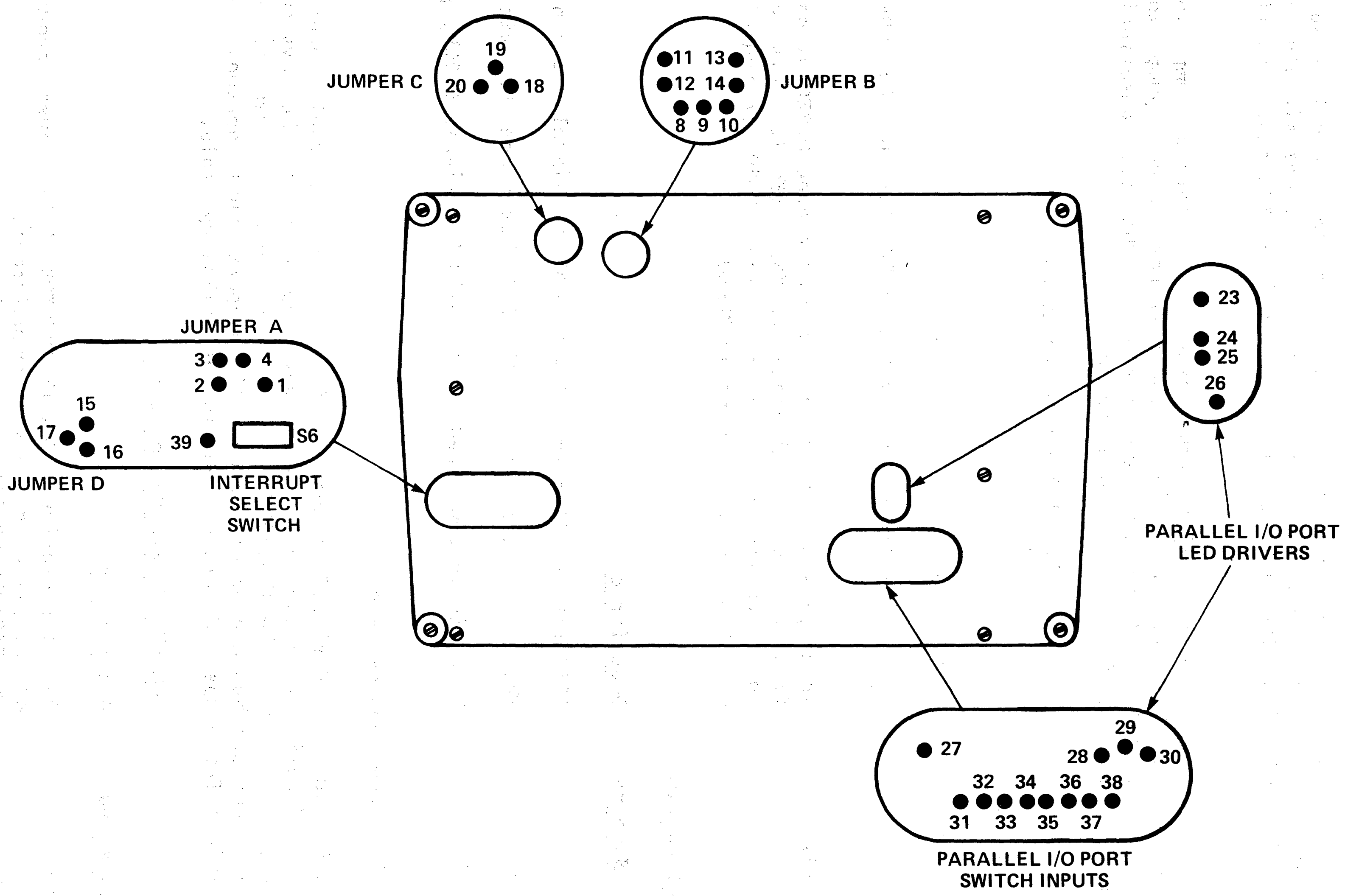


Figure 6.1: Jumper Locations

Jumper B - S100 Clock Select

The bus interface includes three pins for S100 interface clock requirements. The jumper 'B' option allows you to select between two clock signals generated by the INSTRUCTOR 50. The first clock is the same 895 KHz clock available to the 2650.

The second clock is the 2650 OPREQ signal gated by the forced jump logic enable (i.e., the OPREQ clock is inhibited whenever the forced jump logic has control of the 2650's address and data busses). The pin descriptions for jumper 'B' are defined in the following table:

JUMPER B Pin Definitions

Clock Source Pins

Pin Numbers	DESCRIPTION
11,12	These pins are driven by the INSTRUCTOR 50's 895 KHz system clock.
13,14	These pins are driven by the conditioned OPREQ signal. The frequency is approximately 303 KHz. (NOTE: This clock is not a continuous frequency. Some 2650 instructions are executed without generating OPREQ).

S100 Clock Pins

Pin Numbers	Description
8	This pin is connected to the S100 bus signal 01, pin 25.
9	This pin is connected to the S100 bus signal 02, pin 24.
10	This pin is connected to the S100 bus signal CLOCK, pin 49.

Jumper C - Power Source Select

The INSTRUCTOR 50 is designed to operate with its own internal power supply used in conjunction with the wall transformer supplied with the system. Optionally, the input to the INSTRUCTOR 50's 5-volt regulator can be supplied from the interface bus connector. Jumper 'C' supports this option. The pin descriptions for jumper 'C' are defined in the following table:

JUMPER 'C' Pin Definitions

Pin Connected	Description
18-20*	Normal operation. The INSTRUCTOR 50's power requirements are supplied by the wall transformer.
18-19	The INSTRUCTOR 50's power requirements are supplied by an 8-volt unregulated D-C source applied via the bus interface connector.

Jumper D - Cassette Output Selection

The INSTRUCTOR 50's cassette interface provides two recording signal levels. Jumper 'D' selects between a 30mV rms record level and a 300mV rms record level. The pin descriptions for jumper 'D' are defined in the following table:

Jumper 'D' Pin Definitions

Pins Connected	Description
15-17*	This option provides a 30mV rms record level to the cassette.
16-17	This option provides a 300mV rms record level to the cassette.

7. SYSTEM EXPANSION

Introduction

Microprocessors have had a tremendous impact on the hobbyist computer market. Beginning with Altair's 8800 home computer, the hobbyist market has literally exploded with new products. These new products include not only basic computers but a host of small support systems or peripheral boards. The first peripheral boards were simple memory expansion boards, but today there are a wide variety of peripherals available. There are television interfaces for computer graphics, floppy disc interfaces for mass storage, and even a board that synthesizes human speech.

The majority of these peripheral boards are designed to be compatible with the Altair 8800 bus. As more and more Altair 8800-compatible systems were introduced, this microcomputer bus was given an industry wide name, the S100 bus.

The INSTRUCTOR 50's S100 interface (an edge connector at the back of the unit) transforms a simple learning device into a small system computer limited only by the number and type of peripheral boards used. Moreover, the powerful program/data entry and debug facilities of the basic INSTRUCTOR 50 are extended to any device connected to the S100 bus interface.

Because the Altair 8800 home computer was based on the 8800, many of the S100 bus signals are essentially 8080 signals. Many of these signals, such as the two-phase clock and negative supply voltage, are not required by state-of-the-art microprocessors like the 2650. Hence, the INSTRUCTOR 50's S100 interface bus is not pin-for-pin compatible with Altair's original bus. However, the INSTRUCTOR 50's interface bus contains the most commonly used signals and can be easily connected to the majority of S100 peripherals. In addition to the common S100 bus signals, spare pins on the S100 pin bus have been assigned 2650 signals (e.g., OPREQ, R/W, and M/IO). Thus, custom interfaces can be designed with the 2650 control logic, instead of the more cumbersome 8080 interface bus logic. In short, the INSTRUCTOR 50's S100 interface opens up the entire universe of home computer peripherals to owners of the INSTRUCTOR 50 training system.

The INSTRUCTOR 50 bus interface signals are described in Table 7.1.

TABLE 7.1

INSTRUCTOR 50 INTERFACE BUS SIGNALS
(*Indicates a 2650 bus signal)

Pin #	Mnemonic	Signal Description
1	+8V	Positive 8 volts, unregulated. This line provides +8 volts to the INSTRUCTOR 50 when Jumper C selects the interface bus as the system power source.
2	+16V	Positive 16 volts. This line is reserved for +16 volts that may be required for a S100 peripheral board. +16V is neither required for or generated by the INSTRUCTOR 50.
3	XRDY	External Ready. XRDY is returned by an external device when it has completed a data transfer with the 2650. On board the INSTRUCTOR 50 XRDY becomes OPACK for the 2650.
4	VIO	Vectored Interrupt #0. VIO provides an external interrupt request when Jumper A is wired for external interrupts. VIO is latched and generates either an indirect or direct interrupt (selected by the DIRECT/INDIRECT switch) through address H'0007'.
5	Not used	
6	Not used	
7	Not used	
8	Not used	
9	Not used	
10	Not used	
11	Not used	
12	R/W*	Read/Write. A 2650 control signal that indicates whether the processor is performing a read or write operation with an external peripheral board. As with all of the 2650 control signals, R/W is valid only when OPREQ is true. NOTE: An asterisk (*) indicates non-S100 2650 control signals.
13	WRP*	Write Pulse. A 2650 control signal that is generated during memory or I/O write sequences. WRP may be used to strobe data into the selected device.

14	M/IO*	Memory/Input-Output. A 2650 signal that is generated during memory or I/O write sequences. WRP may be used to strobe data into the selected device.
15	RESET*	Reset. When driven high, RESET performs the same operation as depressing the RST key on the INSTRUCTOR 50 front panel. That is, the 2650 is reset and begins executing the user program at location H'0000'.
16	RUN/WAIT*	Run/Wait. A 2650 control signal that indicates whether the 2650 is in the wait state or is executing a program.
17	PAUSE*	Pause. This 2650 control signal input is provided for Direct Memory Access (DMA) operations. When driven high, this signal causes the 2650 to enter the WAIT state after completing the instruction currently being executed.
18	Not used	
19	Not used	
20	Not used	
21	Not used	
22	Not used	
23	Not used	
24	01	Phase 1 Clock. 01 may be driven by the 895 KHz system clock or the 2650 OPREQ signal depending on the configuration of the Jumper B option.
25	02	Phase 2 Clock. 02 may be driven by the system clock or OPREQ depending on the configuration of Jumper B.
26	Not used	
27	Not used	
28	Not used	
29	A5	Address Bit 5
30	A4	Address Bit 4
31	A3	Address Bit 3
32	A15	Address Bit 15. Since the 2650 has an address range of 32K, this line is grounded.
33	A12	Address Bit 12

34	A9	Address Bit 9
35	DO1	Data Out Bit 1
36	DO0	Data Out Bit 0
37	A10	Address Bit 10
38	DO4	Data Out Bit 4
39	DO5	Data Out Bit 5
40	DO6	Data Out Bit 6
41	DI2	Data In Bit 2
42	DI3	Data In Bit 3
43	DI7	Data In Bit 7
44	Not used	
45	SOUT	Output. SOUT indicates that the address bus contains the address of an output I/O device. The addressed device may accept the value on the data bus when PWR (pin 77) is active.
46	SINP	Input. SINP indicates that the address bus contains the address of an input I/O device. The selected device should return its data when PDBIN (pin 78) is active.
47	SMEMR	Memory Read. This signal indicates that the address bus contains the address of a memory location and that the 2650 is performing a memory read operation.
48	Not used	
49	CLOCK	System Clock. Depending on the configuration of Jumper B, this line is driven by the 895 KHz system clock or the 2650 OPREQ output.
50	GND	System Ground.
51	+8V	Positive 8 volts. This line provides +8V to the INSTRUCTOR 50 when Jumper C selects the interface bus as the system power source.
52	-16V	Negative 16 volts. This line is reserved for -16 volts that may be required by a S100 peripheral board. Not supplied with the INSTRUCTOR 50.
53	Not used	
54	Not used	

55	D0*	Data Bus Bit 0 -	In addition to the 2650 control signals, the INSTRUCTOR 50 interface bus also includes a bidirectional data bus. The 2650 signals form a subset of the Interface bus that can be used to interface the INSTRUCTOR 50 to breadboard peripherals with a minimum of interconnect wires.
56	D1*	Data Bus Bit 1 -	
57	D2*	Data Bus Bit 2 -	
58	D3*	Data Bus Bit 3 -	
59	D4*	Data Bus Bit 4 -	
60	Not used		
61	D5*	Data Bus Bit 5 -	
62	D6*	Data Bus Bit 6 -	
63	D7*	Data Bus Bit 7 -	
64	UOPREQ*	User Operation Request - OPREQ, a 2650 control signal, indicates that the processor's address bus, data bus, and other control signals are valid. OPREQ may be used to latch the data bus for write operations and enable input device bus drivers for read operations.	
65	INTACK*	INTERRUPT ACKNOWLEDGE. The 2650 returns INTACK to an interrupting device in response to an INTERRUPT REQUEST. Upon receipt of INTACK, the interrupting device drives the data bus with a relative branch address and asserts either XRDY or PRDY. These signals become the 2650 status signal OPACK.	
66	FLAG*	FLAG. This line contains the 2650 single-bit output port.	
67	USENSE*	USER SENSE. USENSE is the 2650 single-bit input port. FLAG and SENSE are part of the PROGRAM STATUS WORD.	
68	MWRITE	MEMORY WRITE. MWRITE indicates that data is to be written into the memory location addressed by the current value of the ADDRESS BUS.	
69	Not used		
70	Not used		
71	Not used		
72	PRDY	PROCESSOR READY. PRDY is logically OR'd with XRDY to form the 2650 status signal OPACK. PRDY is returned by an addressed device (either memory or I/O) or an interrupting device when the requested data transfer has been completed.	
73	PINT	PROCESSOR INTERRUPT. PINT is an S100 signal that corresponds to the 2650 INTERRUPT REQUEST signal. The 2650 acknowledges PINT when it completes the instruction it was executing when PINT was driven low. The 2650 does not recognize PINT if it is in the WAIT state or if the INTERRUPT INHIBIT bit of the PSW is reset. PINT can be used to release the 2650 from the HALT state.	

74	Not used	
75	Not used	
76	Not used	
77	PWR	PROCESSOR WRITE. PWR indicates that the data bus is valid and may be accepted by the addressed memory location or output device.
78	PDBIN	PROCESSOR DATA BUS IN. PDBIN indicates that the 2650 is readng data from the addressed memory location or input device. PDBIN may be used to enable the selected device's data bus drivers.
79	A0	Address Bit 0
80	A1	Address Bit 1
81	A2	Address Bit 2
82	A6	Address Bit 6
83	A7	Address Bit 7
84	A8	Address Bit 8
85	A13	Address Bit 13
86	A14	Address Bit 14
87	A11	Address Bit 11
88	DO2	Data Out Bit 2
89	DO3	Data Out Bit 3
90	DO7	Data Out Bit 7
91	DI4	Data In Bit 4
92	DI5	Data In Bit 5
93	DI6	Data In Bit 6
94	DI1	Data In Bit 1
95	DI0	Data In Bit 0
96	Not used	
97	Not used	
98	Not used	

99 POR POWER ON RESET. POR is an output signal that indicates that power has been applied to the INSTRUCTOR 50 and the system is being reset. POR may be used to reset peripheral boards on the Interface Bus.

100 GND GROUND. System Ground.

8. THEORY OF OPERATION

Introduction

The INSTRUCTOR 50 is typical of modern microcomputers, reflecting many of the recent advances in microprocessor technology. For example, the current trend in microcomputer design is to replace logic functions implemented with Small-Scale Integration (SSI) and Medium-Scale Integration (MSI) circuits with complex Large-Scale Integration (LSI) microprocessor support circuits. This trend is exemplified in the INSTRUCTOR 50 which makes use of the 2650 microprocessor and the 2656 System Memory Interface. These two chips alone constitute a basic microcomputer. Beyond this two-chip microcomputer, the remainder of the circuits on the INSTRUCTOR 50 Printed Circuit Board are devoted to providing the microcomputer with man-machine and machine-machine interfaces.

This chapter describes the hardware and software associated with the INSTRUCTOR 50 system. The intent is not to give a detailed exposition for maintenance purposes. The INSTRUCTOR 50 comes fully assembled and debugged ready to be plugged in and used and requires little or no maintenance. Rather, the intent is to introduce you to the basic fundamentals of modern microcomputer design.

Basic Concept

The functional heart of computers in general and microcomputers in particular is the system program. The program is a logical sequence of machine instructions that monitor system status, and, based on that status, decides what control actions to take. A computer's Central Processing Unit (CPU) is a device that reads instructions from program storage and, by executing the instructions, performs all of the arithmetic and logical operations required by the system program. The CPU also provides the system program with the physical means to access and control the system's I/O functions. The INSTRUCTOR 50's CPU is the 2650 microprocessor.

The 2650 fetches instructions from program storage and communicates with the system I/O circuits via its address bus, control bus, and data bus. As the 2650 executes each instruction, the address and control bus values specify the device to be communicated with (memory location, I/O device, etc.), and the data bus serves as an information conduit between the processor and the selected device. This information transfer scheme defines the system's basic architecture illustrated in Figure 8.1.

Considerable savings in parts count was realized by decoding the I/O device addresses within the 2656 SMI.

Thus, when the 2650 executes an instruction that references an I/O device (e.g., the parallel I/O port), that device's address is asserted on the address bus, and the Programmable Gate Array within the SMI decodes the address and generates the I/O device's enable signal. Thus enabled, the selected I/O device either accepts data from or returns data to the 2650 over the data bus. As the 2650 executes each instruction, it selects the device

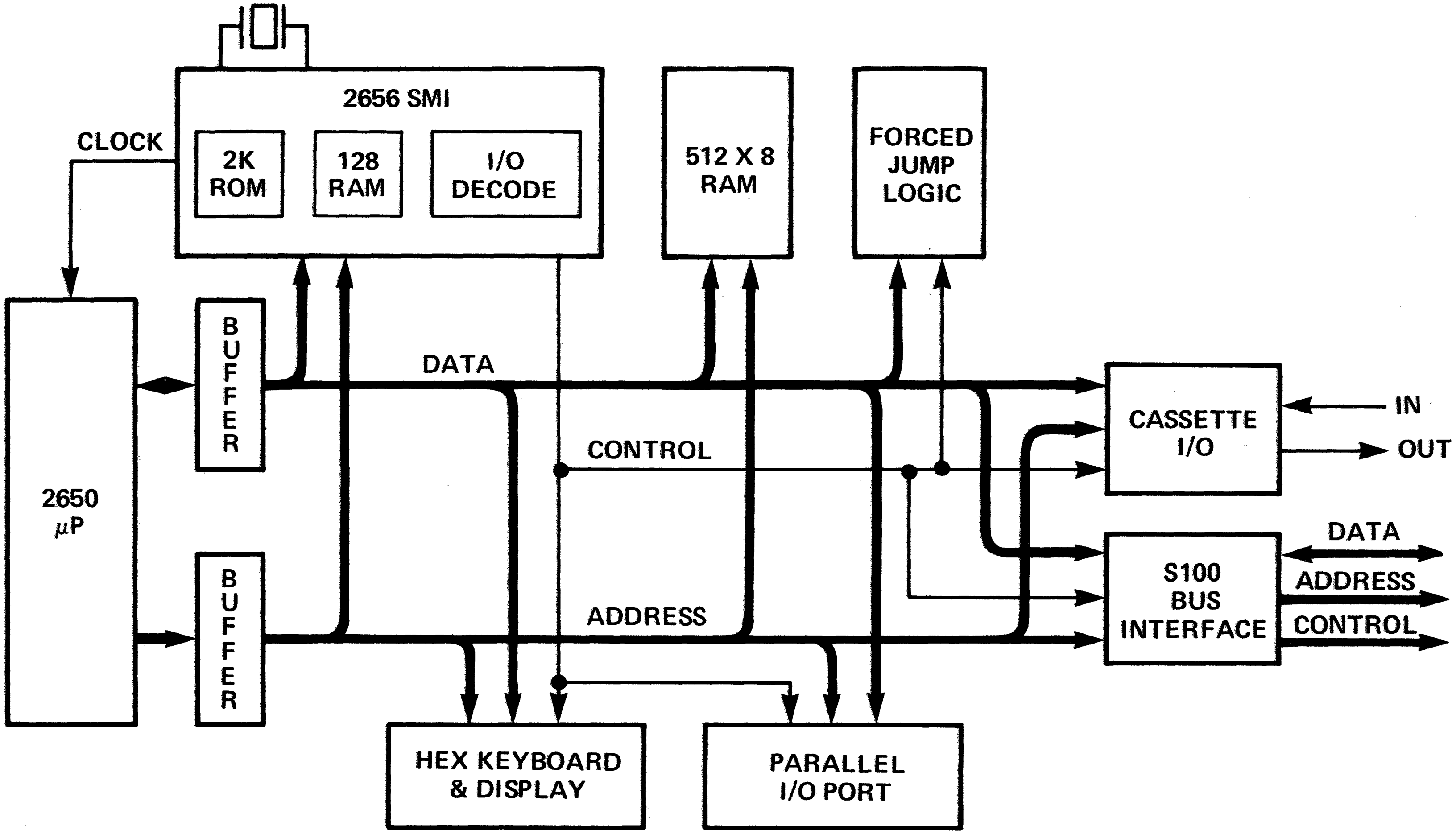


Figure 8.1: Basic Instructor 50 Architecture

specified by the instruction (program storage, user RAM, an I/O device, etc.) with the address bus and communicates with the selected device via the data bus.

Detailed Block Diagram Description

A detailed block diagram of the INSTRUCTOR 50 is presented in Figure 8.2. This section gives a description of each of the major functional blocks illustrated in Figure 8.2

The Microcomputer

As mentioned previously, the basic microcomputer consists of the 2650 microprocessor and the 2656 System Memory Interface (SMI). The 2650 provides the following functions:

8-bit ALU — The Arithmetic Logic Unit performs all of the arithmetic and logical operations required for program execution.

Program Counter — The program counter is used to generate program storage addresses.

Interrupt Logic — The interrupt logic performs all functions required to respond to an interrupt request from an external device.

Internal Registers — The 2650's seven internal registers provide temporary data storage and serve as a link between the ALU and external data storage, such as RAM locations and I/O devices.

Bus Interface Logic — The bus interface logic distinguishes between memory and I/O device addresses and specifies the direction of data transfers between the processor and external data storage.

The 2650 microprocessor is surrounded with bus drivers (buffers). Because the 2650 is fabricated using an MOS process, its output pins can drive only one TTL load. The bus drivers buffer the 2650 outputs and are able to drive all of the loads on the INSTRUCTOR 50's busses.

The buffered 2650 data, address and control busses are connected directly to the 2656 SMI. The SMI contains the 2K monitor program, 128 bytes of scratchpad RAM, a system clock generator, and an eight-bit I/O port. The eight-bit I/O port is controlled by a mask Programmable Gate Array (PGA). As configured for the INSTRUCTOR 50, the PGA decodes the address bus and provides eight I/O chip enables for the user RAM and I/O devices. Table 8.1 lists the functions of these outputs.

All of the monitor program's scratchpad memory requirements are met by the SMI's 128 byte RAM. In fact, the monitor only requires 64 bytes, thus leaving the remaining 64 bytes for user storage. It should be noted, however, that while the INSTRUCTOR 50 enables you to access these 64 bytes of the SMI's RAM with the DISPLAY AND ALTER MEMORY command and the FAST PATCH command, the SINGLE STEP and BREAKPOINT commands are not supported within this memory

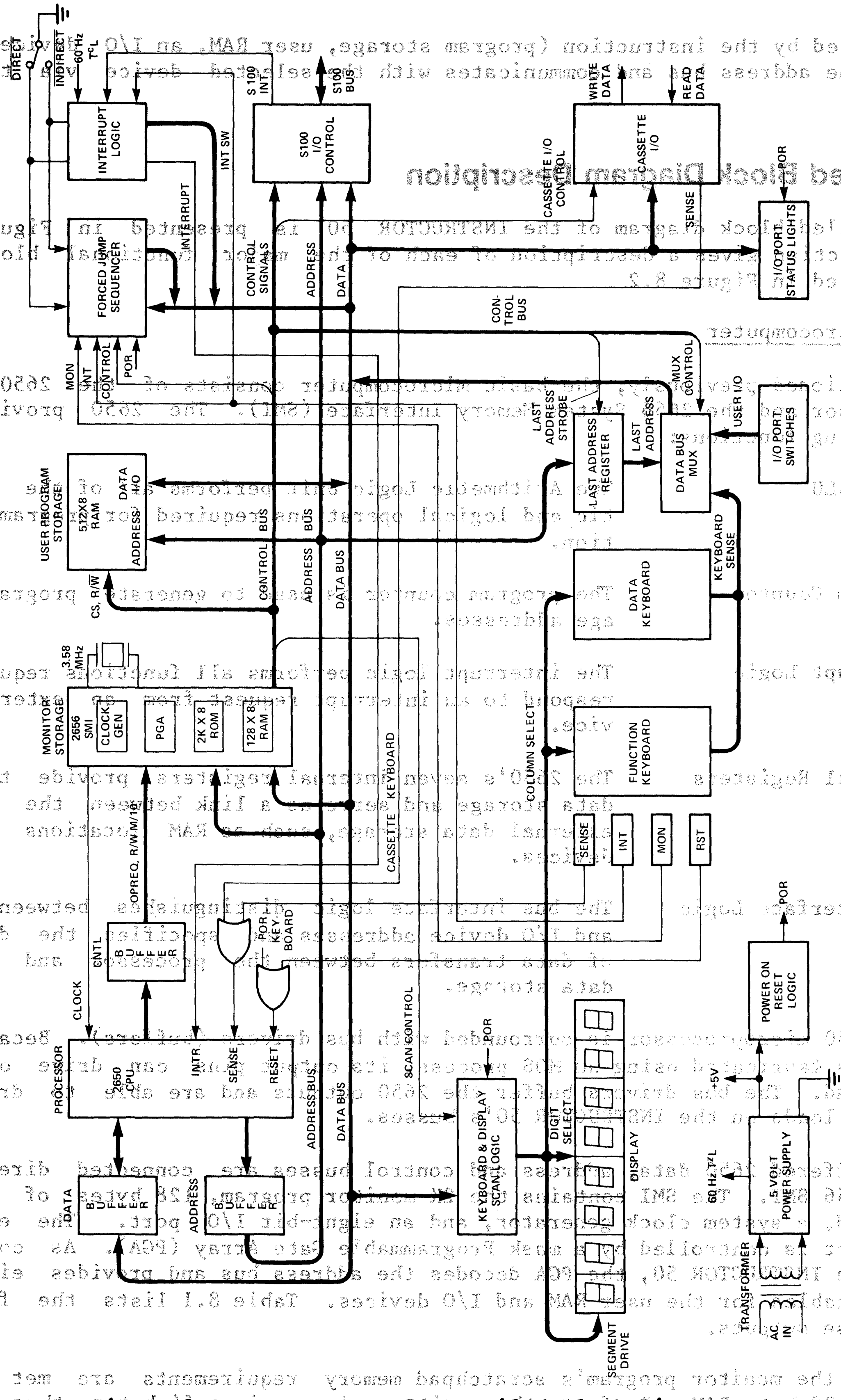


Figure 8.2: Instructor 50 Detailed Block Diagram

space. Hence, these 64 bytes should be used for data storage only. That is, user programs should be stored in user RAM or on an S100 memory expansion board.

INSTRUCTOR 50 Memory Allocation

Figure 8.3 is a memory map of the INSTRUCTOR 50's addressable memory space. The memory map is divided into four 8K pages reflecting the addressing architecture of the 2650. The first page, page zero, contains the user RAM and the SMI ROM and RAM. The second, third, and fourth pages are available for user memory expansion or memory mapped I/O.

The user RAM is formed by four 256 x 4 RAMs (Signetics 2112's) that are enabled by the SMI chip-enable lines mentioned previously. Chapter 7 described how S100 memory boards can be added to the INSTRUCTOR 50.

Table 8.1

CONTROL SIGNALS GENERATED BY THE SMI

Signal	Function
RAM0CE	RAM 0 chip enable: this signal enables the lower 256 bytes of user RAM.
RAM1CE	RAM 1 chip enable: this signal enables the upper 256 bytes of user RAM.
PORTFX	PORTFX goes low whenever the 2650 executes an extended I/O instruction with an address between H'F8' and H'FF', inclusive. This signal enables the INSTRUCTOR 50's I/O device addresses to be decoded with just three address bits.
USRPORT	USRPORT goes low whenever the 2650 accesses the parallel I/O port with an extended I/O instruction (address H'07').
USRMEM	This signal goes low when the 2650 executes a memory reference instruction that specifies address H'0FFF'. USRMEM enables the parallel I/O port when the port address select switch is in the MEMORY position.
DI/O	DI/O goes low when the 2650 executes a non-extended I/O instruction that specifies port D. If the port address select switch is in the NON-EXTENDED position, DI/O enables the parallel I/O port.
CI/O	CI/O goes low when the 2650 executes a WRTC instruction. This signal is used by the forced jump logic for breakpoint detection.
MON	MON goes low whenever the 2650 fetches an instruction or data value within the monitor's address space (H'17C0' and H'1FFF').

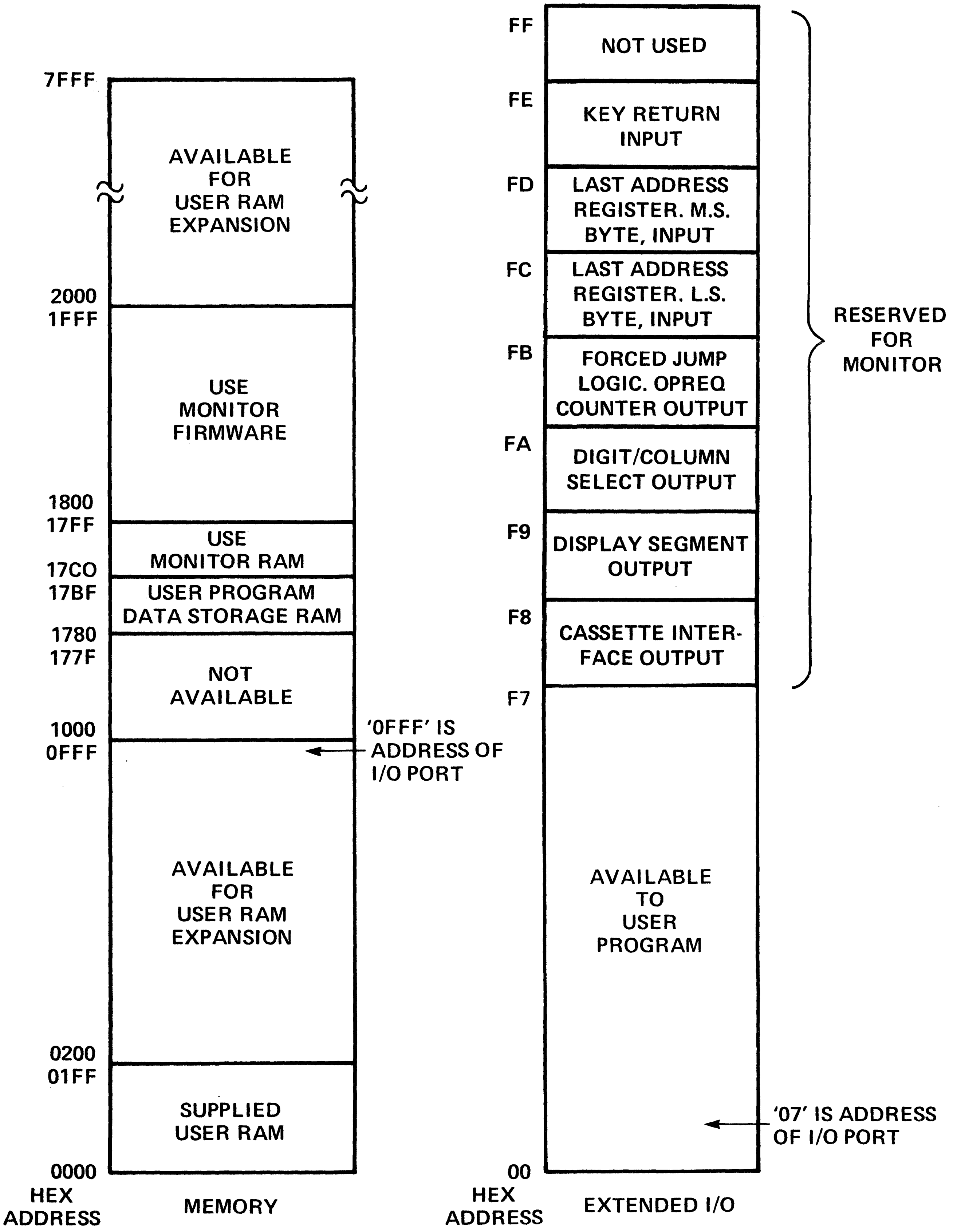

7FFF
AVAILABLE FOR USER RAM EXPANSION
2000
1FFF
USE MONITOR FIRMWARE
1800
17FF
USE MONITOR RAM
17CO
17BF
USER PROGRAM DATA STORAGE RAM
1780
177F
NOT AVAILABLE
1000
0FFF
'0FFF' IS ADDRESS OF I/O PORT
AVAILABLE FOR USER RAM EXPANSION
0200
01FF
SUPPLIED USER RAM
0000
HEX ADDRESS
MEMORY
FF
NOT USED
FE
KEY RETURN INPUT
FD
LAST ADDRESS REGISTER. M.S. BYTE, INPUT
FC
LAST ADDRESS REGISTER. L.S. BYTE, INPUT
FB
FORCED JUMP LOGIC. OPREQ COUNTER OUTPUT
FA
DIGIT/COLUMN SELECT OUTPUT
F9
DISPLAY SEGMENT OUTPUT
F8
CASSETTE INTER-FACE OUTPUT
F7
RESERVED FOR MONITOR
AVAILABLE TO USER PROGRAM
'07' IS ADDRESS OF I/O PORT
00
HEX ADDRESS
EXTENDED I/O

Figure 8.3: Memory and I/O Organization

Parallel I/O Port

The parallel I/O port consists of an output latch, input switches, and port address decode logic. The port address decode logic generates a port enable whenever one of the three following conditions are met.

1) The 2650 executes a WRTD or REDD instruction.

2) The 2650 executes either a WRTE or REDE instruction that specifies H'07' as an extended I/O address.

3) The 2650 executes a memory reference instruction that specifies location H'0FFF'.

The Port Address switch selects one of these signals as the parallel I/O port enable.

Whenever the I/O port is enabled and the R/W control line specifies a write operation, the value on the data bus is strobed into the I/O port output latch. This latch drives the I/O port indicator LEDs.

The I/O port switches are one of four inputs to a data bus multiplexer. Whenever the I/O port is enabled and the R/W line indicates a read operation, the I/O switch levels are asserted on the data bus via the data bus multiplexer.

Keyboard and Display Logic

The INSTRUCTOR 50's primary man-machine interface consists of an output device, the eight-digit display, and two input devices - the function and data entry keyboards. Together they provide an inexpensive human interface to the microcomputer.

The display digits consist of seven discrete LEDs arranged in a rectangular array or bars and an eighth LED that serves as a decimal point. There are several methods of driving a seven-segment display with a microprocessor. The most straightforward approach is to provide a separate output port latch to drive each individual display. With this approach, the microprocessor simply writes a byte to each output port, corresponding to the segments required to form the desired character. While the direct drive approach is the simplest to conceptualize, it also requires the most hardware to implement. However, the basic rule of thumb in microcomputer design is to eliminate as much system hardware as possible with program logic. Toward this end, an alternate display drive method that requires only two output ports is used in the INSTRUCTOR 50.

The first output port (extended port F9) is a latch that drives the segment select lines connected in parallel to each of the eight digits. The second output port (extended port FA), an eight-bit latch, enables only one digit at a time. With this structure, the segment select lines can be time shared among the eight digits. The 2650 first enables a digit with the digit select output port and then writes that digit's character segments in the segment select output port. The process is repeated for each digit in a sequential fashion. If each digit is illuminated at a sufficiently fast frequency, about 100 Hz, the entire eight-digit display appears flicker free. Thus, considerable savings in display drive hardware is realized by substituting program complexity for output ports.

Because of the display's high-current requirements, the two output port latches require current buffering. A darlington transistor array on the output of each latch supplies the required current.

There are several methods of interfacing a microcomputer to an input keyboard. Here again the primary objective is to minimize the system hardware by placing as much of the control logic in the program as possible. The keyboard scan approach used by the INSTRUCTOR 50 arranges the two keyboards in a matrix. Since each function and data key is actually a two-terminal switch, a matrix can be formed by grouping the terminals of each switch into columns and rows. This organization is illustrated in Figure 8.4

Referring to Figure 8.4, the column select signals, COL 1-COL 6, are driven by an output port (extended port FA), and the four sense signals, KRO-KR3, serve as the inputs to an input port (extended port FE). Given this structure, the 2650 can scan the keyboard to detect a switch closure as follows:

1) The processor writes a byte to the column select output port that drives one of the column select lines low.

2) The processor reads the row sense input port. If any of the keys in the selected column are depressed, a low is sensed on the corresponding row sense line.

3) The process is repeated for each column.

The keyboard interface column select operation is identical to that of the display digit select. Hence, a single output port serves both interfaces. The row sense input port is another input to the data bus multiplexer. When the 2650 executes an REDE instruction that specifies the row sense input port, the row sense signals are returned to the processor on the data bus via the multiplexer.

Referring again to Figure 8.4, you will notice that four of the function keys, SENS, INT, MON, and RST, are not included in the switch matrix. The reason for their absence is that the functions they perform are independent of the monitor program. Since RST resets the 2650, this switch is connected to the 2650's RESET pin (after being OR'ed with the power on reset signal). Likewise, the SENS key is connected to the 2650 SENSE input pin. (Actually the 2650 SENSE pin is used for both the SENS key and the audio cassette interface. The signal presented to the 2650 depends on whether or not the 2650 is reading data from cassette). The INT key is connected directly to the INSTRUCTOR 50 interrupt logic, and the MON key is connected to the forced jump logic. The operation of these two keys is described under forced jump logic.

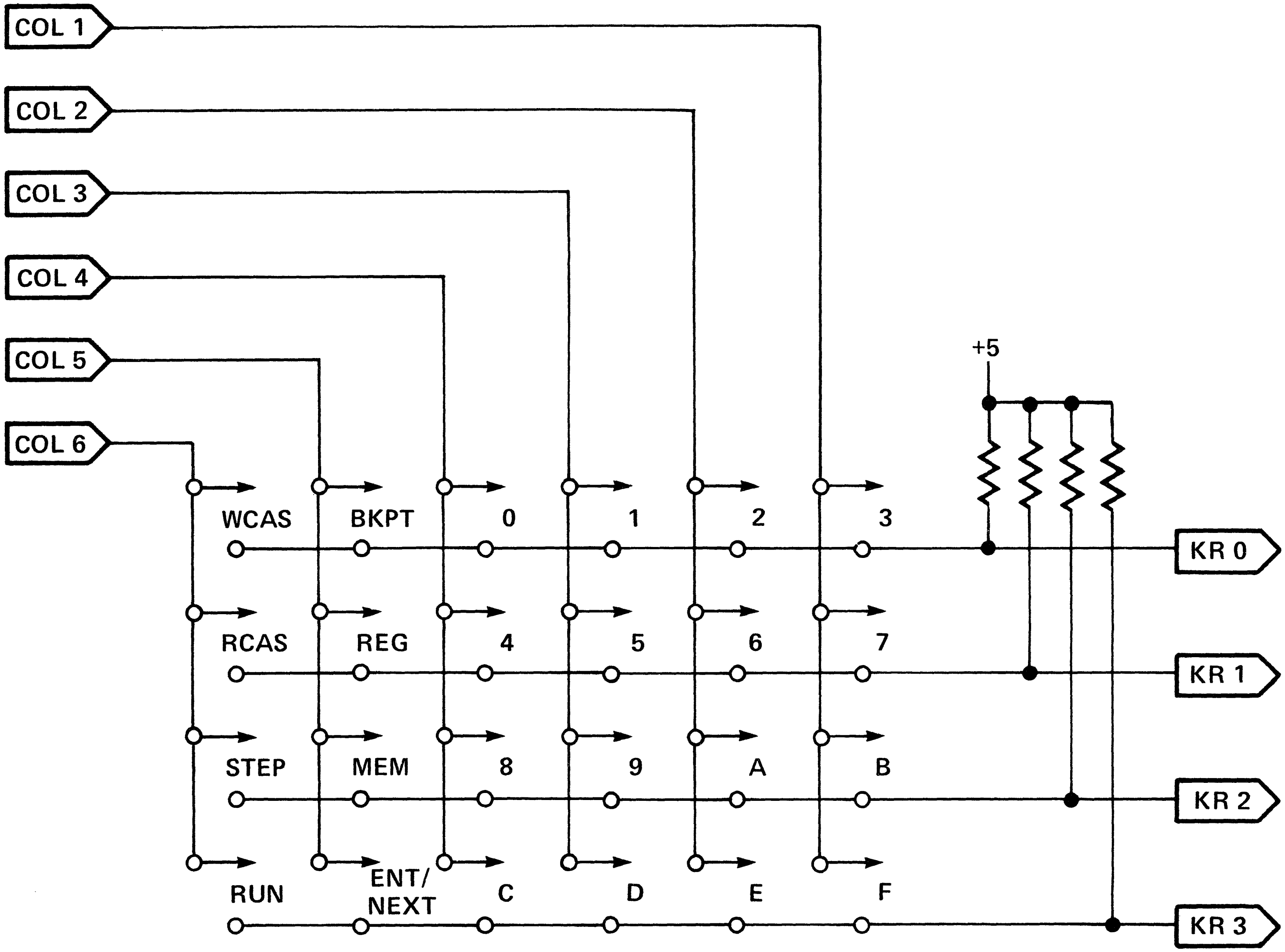


Figure 8.4: Keyboard Layout

Bit Assignments for Keyboard and Display Ports

Figure 8.5 gives the bit assignments for the ports associated with the keyboard and display circuits.

PORT F9 -- SEGMENT OUTPUT

7	6	5	4	3	2	1	0
SEG DP	G	F	E	D	C	B	SEG A

1 = SEGMENT ON
0 = SEGMENT OFF

PORT FA -- KEYBOARD COLUMN AND DISPLAY DIGIT SELECT OUTPUT

		BKPT REG MEM ENT/NXT	WCAS RCAS STEP RUN	3 7 B F	2 6 A E	1 5 9 D	0 4 A C	KEYS
7 (LEFT)	6	5	4	3	2	1	0 (RIGHT)	DISPLAY DIGIT

1 = COLUMN/DIGIT selected
0 = COLUMN/DIGIT not selected

PORT FE -- KEY RETURN INPUT

				C D E F RUN ENT/NXT	8 9 A B STEP MEM	4 5 6 7 RCAS REG	0 1 2 3 WC BKPT

Figure 8.5

The Cassette Interface

The cassette interface is unique among the INSTRUCTOR 50's I/O devices in that it communicates with an analog system, a cassette tape recorder. It converts microprocessor-generated logic signals into an audio waveform for recording data, and converts the audio waveform returned from the recorder into a digital pulse stream that can be decoded by the processor when data is being read from the cassette.

The INSTRUCTOR 50 uses a two-bit output port (extended port F8) for recording data onto cassette tape and a single-bit input port for reading the data back. Figure 8.6 illustrates the record waveforms required by this technique. The two signals, FREQ and ENV, are provided by a two-bit output port (port F8, bits 3 and 4, respectively). These signals are combined with an open-collector NAND gate to form the write signal for the cassette. As shown in Figure 8.6, six pulses are used to record a 'zero' on the cassette, and three pulses to record a 'one'. The only exception to this recording format is the last bit of a byte. Six additional pulses are recorded for the last bit of a byte to mark byte boundaries (i.e., a one is nine pulses and a zero is twelve pulses).

Since only a single bit input port is required to read data back from cassette, the 2650's SENSE pin is used for this purpose. Bit 7 of port F8 is used to switch the SENSE input from the keyboard to the cassette interface when a Read Cassette operation is in progress. However, before the audio input is presented to the SENSE pin, it is digitized by a Schmidt trigger. The Schmidt trigger has about 1.5 volts of hysteresis that provides the read logic with necessary noise immunity.

Interrupt Logic

The INSTRUCTOR 50 can respond to interrupt requests from three possible sources: the INT key, the real-time clock derived from the power supply line frequency, or the S100 bus interface. As mentioned previously, interrupt source is determined by a switch located at the bottom of the INSTRUCTOR 50 case. This switch selects between the INT key and the real-time clock. A jumper option enables interrupt requests from the S100 bus interface.

The selected interrupt request source is input to a flip-flop that is set when an interrupt request is received. The output of the flip-flop is connected to the INTREQ pin on the 2650. The 2650 responds to an interrupt request by asserting INTACK. INTACK, in turn, enables a tri-state drive that places the interrupt vector H'07' or H'87', depending on the position of the DIRECT/INDIRECT switch on the data bus. INTACK also resets the interrupt request flip-flop.

Forced Jump Logic

The INSTRUCTOR 50's Breakpoint and Single Step commands are implemented with a combination of firmware and hardware control. This hardware portion is called the forced jump logic. The forced jump logic returns program control to the monitor whenever a breakpoint is detected, after a single user instruction has been executed in the step mode, when the MON key is depressed, and when power is initially applied to the INSTRUCTOR 50.

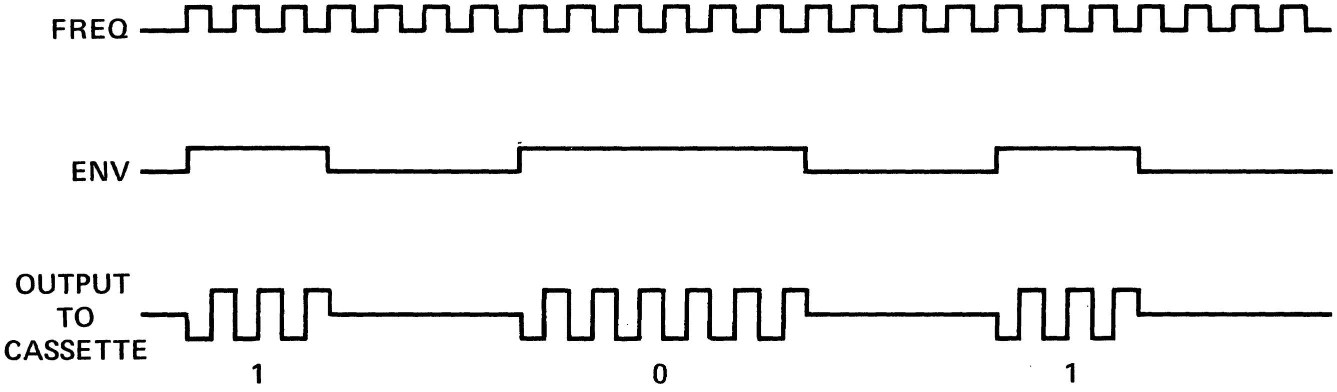


Figure 8.6: CASSETTE RECORD WAVEFORMS

The forced jump logic consists of the following logical elements:

1) The Return to Monitor Sequencer - This sequencer is responsible for returning program control to the monitor when the 2650 is executing a user program. The sequencer consists of a programmable counter and a 32 x 8 PROM. The PROM contains the data values of an absolute branch instruction. When the sequencer is active, the forced jump logic disables the INSTRUCTOR 50's normal instruction fetch mechanism and returns the absolute branch instruction stored in the PROM. The 2650 initializes the sequencer by loading the counter via extended output port FB.

2) The Last Address Register - The Last Address Register (LAR) saves the last address issued by a user program before program control is returned to the monitor. This address points to the next instruction that the user program would execute if the return to monitor had not been activated. The monitor program reads the LAR to determine where the user program should resume execution after a STEP command has been completed or when a breakpoint is encountered. The monitor reads the least-signigicant byte of the LAR by addressing port FC, and the most-significant byte by addressing port FD.

3) Control Logic - The control logic performs general housekeeping functions such as loading the LAR, integrating interrupt requests with the return to monitor state sequencer, and loading the programmable counter.

The forced jump logic is enabled when power is first applied to the INSTRUCTOR 50, when the MON key is depressed, when a breakpoint is detected, and when the monitor program executes the STEP command. The resulting action taken by the forced jump logic when one of these events occurs is described below.

POWER ON (POR) OR MON KEY DEPRESSION

When power is applied to the INSTRUCTOR 50 or when the MON key is depressed, the 2650 is reset. The 2650 responds to a reset by clearing its internal program counter and fetching the instruction located at byte zero, page zero. However, when the 2650 places address H'0000' on the address bus, the forced jump logic disables the normal memory access mechanism and returns a NOP instruction value to the 2650 via the data bus. The 2650 executes the NOP and attempts to fetch an instruction at the next sequential address H'0001'. This instruction fetch generates an operation request (OPREQ). OPREQ is used to increment the sequencer counter. In this state, the return to monitor sequencer places the first byte of an unconditional branch instruction on the data bus. When the 2650 receives the BCTA, UN op-code, it generates two more OPREQs to fetch the branch address. Each OPREQ increments the counter and the PROM places the beginning address of the monitor, H'1800', on the data bus. At this point the 2650 executes the branch to monitor, and the forced jump logic returns to the idle state.

BREAKPOINT DETECTION

If the user has specified a breakpoint, the monitor program inserts a WRTC instruction at the breakpoint address specified. When the 2650 executes the WRTC instruction, a control signal is generated that produces the same results as the POR signal, and program control is returned to the monitor. A monitor software flag distinguishes this entry from a POR or MON key entry and causes a branch to the breakpoint routine.

SINGLE STEP

The execution of a single 2650 instruction in response to the STEP key is an excellent example of combined firmware/hardware control. When the STEP key is depressed, the monitor program fetches the instruction pointed to by the Program Counter and calculates the number of OPREQs required to execute the instruction. The OPREQ counter (an extended I/O port) is then loaded with a value that corresponds to the number of OPREQs. The monitor then restores the user's program registers and status and branches to the instruction to be stepped. When the 2650 executes the instruction, the OPREQ counter, beginning at the present count, addresses "dummy states" of the return to monitor sequencer. That is, the locations addressed are not output on the data bus. When the last OPREQ of the instruction occurs, the output of the return to monitor PROM is enabled, and subsequent OPREQs return the unconditional branch to monitor instruction bytes to the processor.

If an interrupt request should occur during execution of the STEP instruction, the 2650 waits until the instruction has been completed before asserting INTACK. Conditioned by the forced jump control logic, INTACK becomes an address bit for the return to monitor PROM. While INTACK is high, another address bit reflects the position of the DIRECT/INDIRECT switch. In concert, these two address bits force the sequencer into one of two interrupt handling sequences: one for direct interrupts and another for indirect interrupts.

S100 Bus Interface

The S100 bus interface consists of tri-state drivers and receivers and a Field Programmable Gate Array (FPGA) which produces the S100 bus signals from logical combinations of 2650 control signals. Unfortunately, the S100 bus is far from standardized. Many of the signals are repetitious and different peripheral manufacturers make different demands of the bus. The FPGA enables you to modify the bus interface to meet any specific needs you may encounter. A detailed description of the S100 bus interface is given in Chapter 7.

System Power

The INSTRUCTOR 50 obtains its system power from one of two possible sources. The first source is an A-C wall transformer supplied with the INSTRUCTOR 50. The transformer provides the INSTRUCTOR 50 with 8 VAC (rms). On board, the A-C input is rectified, and the resulting D-C voltage is applied to a three-terminal regulator. The regulator supplies 5 VDC at 1.5 amps, the system power requirements of the INSTRUCTOR 50. The user may optionally change a wire jumper at the bottom of the printed circuit board to select unregulated 8 VDC from the S100 bus interface as input to the regulator.

In addition to the rectifier, the A-C input to the system is also applied to the resistive divider network. The reduced A-C voltage is input to a comparator that outputs a 60 Hz real-time clock (50 Hz in Europe and Japan). This real-time clock is available to the interrupt request logic via a select switch at the bottom of the printed circuit board. The wall transformer can be used to drive the real-time clock even if system power is derived from the S100 bus interface.

The USE Monitor

Without question, the most important component of any microcomputer (or any computer for that matter) is the system program. Every function or operation performed by a microcomputer is accomplished by executing a sequence of instructions within the system program.

Basically, the USE monitor is a collection of separate routines -- one routine for each system command. A brief functional description of several routines with illustrative examples is provided in Chapter 5. This section provides a brief description of the command executive - a section of the monitor program that links the various command routines into a cohesive system program.

Figure 8.7 is a flowchart of the command routine executive section of USE. Whenever the forced jump logic returns program control to the monitor, monitor execution begins at H'1800', the first address of the executive. Beginning at this address, the first operation is to save the 2650 registers and Program Status Word. (These values are restored before program control is transferred to the user program). The next operation is to check certain software flags to determine how the forced jump logic was enabled. If it was triggered by a breakpoint (WRTC instruction), program control is returned by the breakpoint routine. Similarly, if the forced jump logic was activated by the completion of a single-step sequence, program control is returned to the single-step routine. The alternatives to these two entry modes are power on and MON key depression. If the executive was entered via either of these two modes, the executive clears the breakpoint and step flags, since they may be on even if entry to the monitor was via power-on. Next, the display buffer pointer is set to the "HELLO" message table, and the DISPLAY subroutine is called. The monitor remains in this routine until a function key is depressed.

Upon returning from the DISPLAY subroutine, R0 contains the function key value. This value is used as an index to fetch a command routine address from the command address table. The address thus accessed is used for an absolute branch to one of the command routines. The executive is re-entered from any command routine when a function key is depressed. Hence, a new command address is accessed, and the monitor again branches to the specified command routine. Refer to the USE Program Listing in Chapter 11 for detailed information on the USE routines.

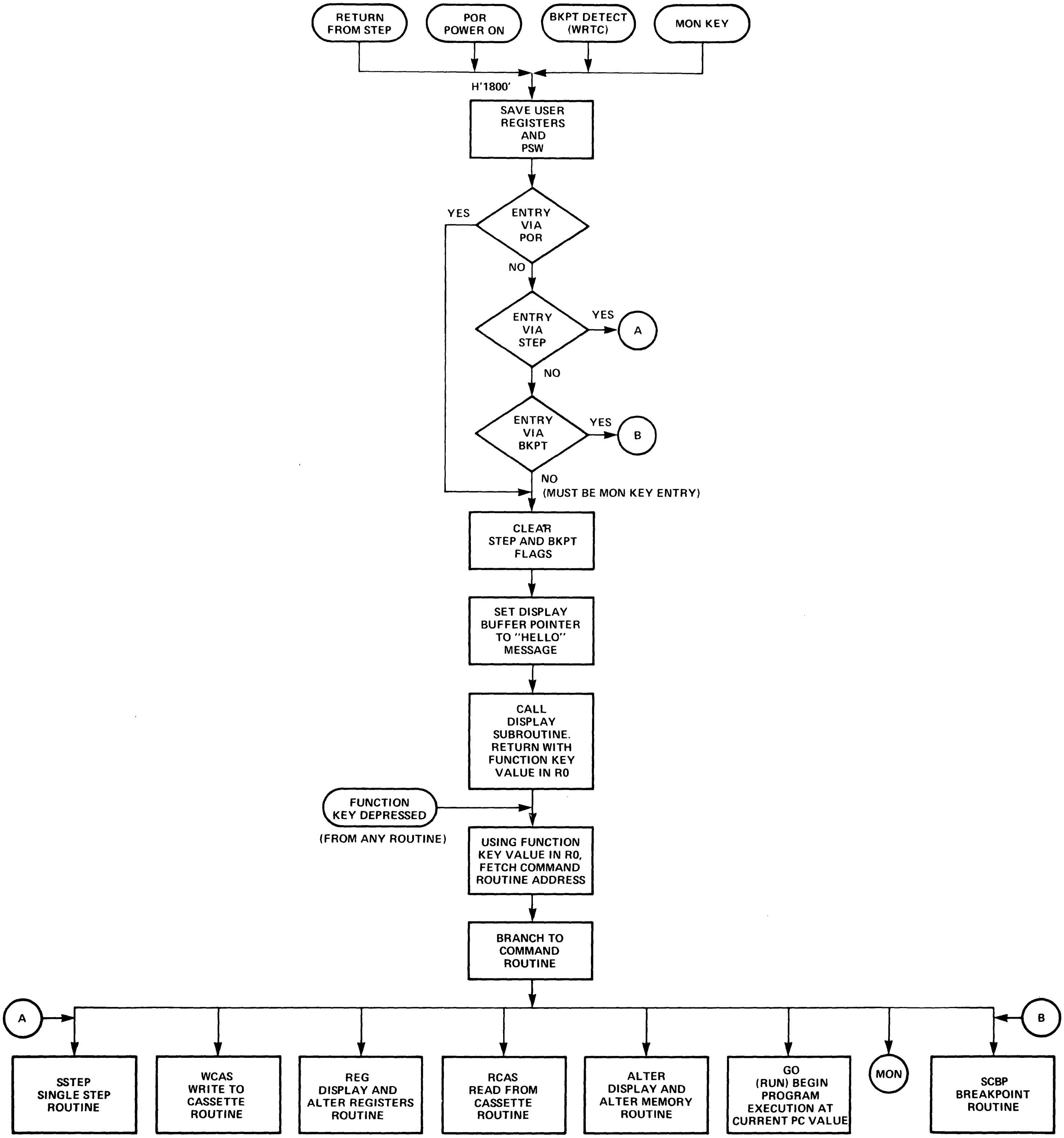

RETURN FROM STEP
POR POWER ON
BKPT DETECT (WRTC)
MON KEY
H'1800'
SAVE USER REGISTERS AND PSW
ENTRY VIA POR
YES
NO
ENTRY VIA STEP
YES
A
NO
ENTRY VIA BKPT
YES
B
NO (MUST BE MON KEY ENTRY)
CLEAR STEP AND BKPT FLAGS
SET DISPLAY BUFFER POINTER TO "HELLO" MESSAGE
CALL DISPLAY SUBROUTINE. RETURN WITH FUNCTION KEY VALUE IN R0
FUNCTION KEY DEPRESSED
(FROM ANY ROUTINE)
USING FUNCTION KEY VALUE IN R0, FETCH COMMAND ROUTINE ADDRESS
BRANCH TO COMMAND ROUTINE
A
B
SSTEP SINGLE STEP ROUTINE
WCAS WRITE TO CASSETTE ROUTINE
REG DISPLAY AND ALTER REGISTERS ROUTINE
RCAS READ FROM CASSETTE ROUTINE
ALTER DISPLAY AND ALTER MEMORY ROUTINE
GO (RUN) BEGIN PROGRAM EXECUTION AT CURRENT PC VALUE
MON
SCBP BREAKPOINT ROUTINE

Figure 8.7: USE Command And Routine Executive

9. THE 2650 MICROPROCESSOR

Introduction

The 2650 processor is a general purpose, single chip, fixed instruction set, parallel 8-bit binary processor. A general purpose processor can perform any data manipulations through execution of a stored sequence of machine instructions. The processor has been designed to closely resemble conventional binary computers, but executes variable length instructions of 1 to 3 bytes in length.

The 2650A microprocessor is functionally identical to the 2650, but it incorporates a new chip design which provides improved operating margins. All references to the 2650 in this section apply to the 2650A as well.

The 2650 contains a total of 7 general purpose registers, each 8 bits long. They may be used as source or destination for arithmetic operations, as index registers, and for I/O transfers.

The processor can address up to 32,768 bytes of memory in 4 pages of 8,192 bytes each. The processor instructions are 1, 2 or 3 bytes long, depending on the instruction. Variable length instructions tend to conserve memory space, since a 1- or 2-byte instruction may often be used rather than a 3-byte instruction. The first byte of each instruction always specifies the operation to be performed and the addressing mode to be used. Most instructions use 6 of the first 8 bits for this purpose, with the remaining 2 bits forming the register field. Some instructions use the full 8 bits as an operation code.

The 2650/2650A instruction set consists of 75 basic instructions, of which about 40% are arithmetic instructions. This class contains the Boolean, arithmetic and compare operations, each of which may be executed using any one of eight addressing modes. Another 30% of the instruction set includes I/O instructions, instructions for performing operations on the two status registers, a Decimal Adjust instruction and the Halt instruction.

Utilizing multiple addressing modes greatly increases coding efficiency, allowing functions to be performed using fewer instructions than less powerful machines. The resulting reduction in routine execution time and memory capacity requirements directly translates into improved system performance and reduced memory cost.

In addition to the microprocessor itself, a number of support circuits and development tools are also required to design and test microprocessor-based systems. A growing complement of circuits and hardware and software development aids are available from Signetics.

Features:

Low System Cost

- Low cost N-channel products
- Intrinsic advantages of single +5V supply

- Uses standard low cost memories
- Low cost interfacing

Ease of Use

- Easy interfacing
- Conventional instruction set
- Ease of programming

Wide Range of Applications

- General purpose capability
- Powerful architecture
- Powerful instruction set
- Flexibility
- Expanding family of support devices

2650 Microprocessor Characteristics

General

- Single chip 8-bit processor
- Signetics' silicon gate N-channel technology
- Single +5V power supply
- Low power consumption
- Single phase TTL-compatible clock
- Static operation: No minimum clock frequency
- Clock frequency: 1.25 MHz maximum
- Cycle time: 2.4us minimum
- Standard 40-pin DIP

Interfaces

- TTL-compatible inputs and outputs-no external resistors required.
- Tri-state bus outputs for multiprocessor and direct memory access systems.
- Asynchronous (handshaking) memory and I/O interface.
- Accepts wide range of memory timing.
- Interfaces directly with industry standard memories.
- Powerful control interface.
- Single-bit direct serial I/O path.
- Parallel 8-bit I/O capability.

Architecture

- 8-bit bidirectionsl tri-state data bus.
- Separate tri-state address bus.
- 32,768-byte addressing range.
- Internal 8-bit parallel structure.
- Seven 8-bit addressable general purpose registers.
- Eight-level on-chip subroutine return address stack.
- Program status word for flexibility and enhanced processing power.
- Single-level hardware vectored interrupt capability.
- Interrupt service routines may be located anywhere in addressable memory.

Instruction Set

- General purpose instruction set with substantial capabilities in arithmetic, character manipulation and control and I/O processing
- Fixed instruction set
- 75 instructions
- Up to 8 addressing modes
- True indexing with optional auto increment/decrement
- 1, 2 or 3-byte instructions
- 1 and 2-byte I/O instructions
- Selective test of individual bits
- Powerful instruction set and addressing modes minimize memory requirements.

Internal Organization

The block diagram of the 2650 series, Figure 9.1, shows the major internal components and the data paths that interconnect them. In order for the processor to execute an instruction, it performs the following general steps:

1. The Instruction Address Register provides an address for memory.
2. The first byte of an instruction is fetched from memory and stored in the Instruction Register.
3. The Instruction Register is decoded to determine the type of instruction and the addressing mode.
4. If an operand from memory is required, the operand address is resolved and loaded into the Operand Address Register.
5. The operand is fetched from memory and the operation is executed.
6. The first byte of the next instruction is fetched.

The Instruction Register (IR) holds the first byte of each instruction and directs the subsequent operations required to execute each instruction. The IR contents are decoded and are used in conjunction with the timing information to control the activation and sequencing of all the other elements on the chip. The Holding Register is used in some multiple-byte instructions to contain further instruction information and partial absolute addresses.

The Arithmetic Logic Unit (ALU) is used to perform all of the data manipulation operations, including load, store, add, subtract, AND, inclusive-OR, exclusive-OR, compare, rotate, increment and decrement. It contains and controls the Carry bit, the Overflow bit, the Interdigit Carry and the Condition Code register parts of the Program Status Word.

The Register Stack contains 6 registers that are organized into two banks of three registers each. The Register Select bit (RS) of the Program Status Word picks one of the two banks to be accessed by instructions. In order to accommodate the register-to-register instructions, register zero (R0) is outside the array. Thus, register zero is always available along with one set of three registers.

The Instruction Address Register (IAR) holds the address of the next instruction byte to be accessed. The Address Adder is used to increment the instruction address and to calculate relative and indexed addresses. The Operand Address Register stores operand addresses and sometimes contains intermediate results during effective address calculations.

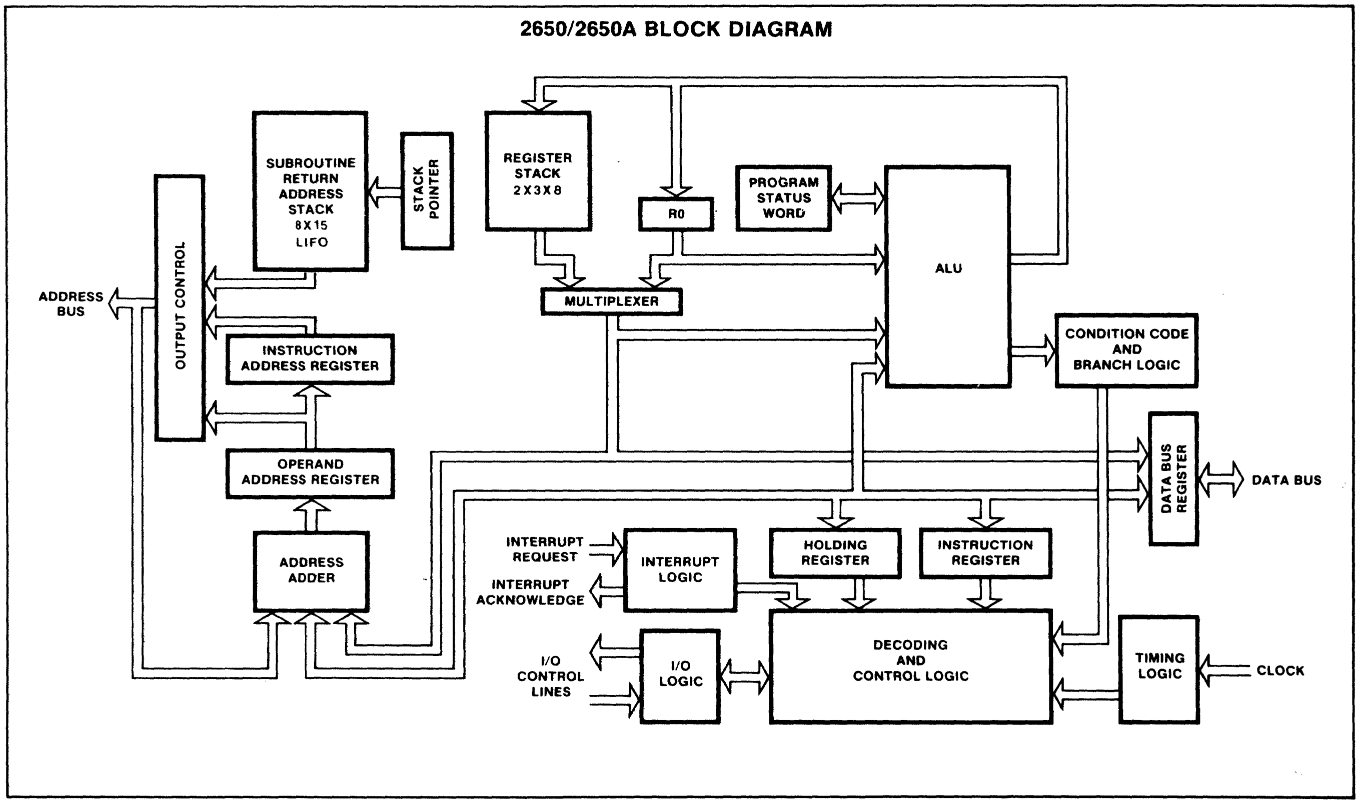


Figure 9.1

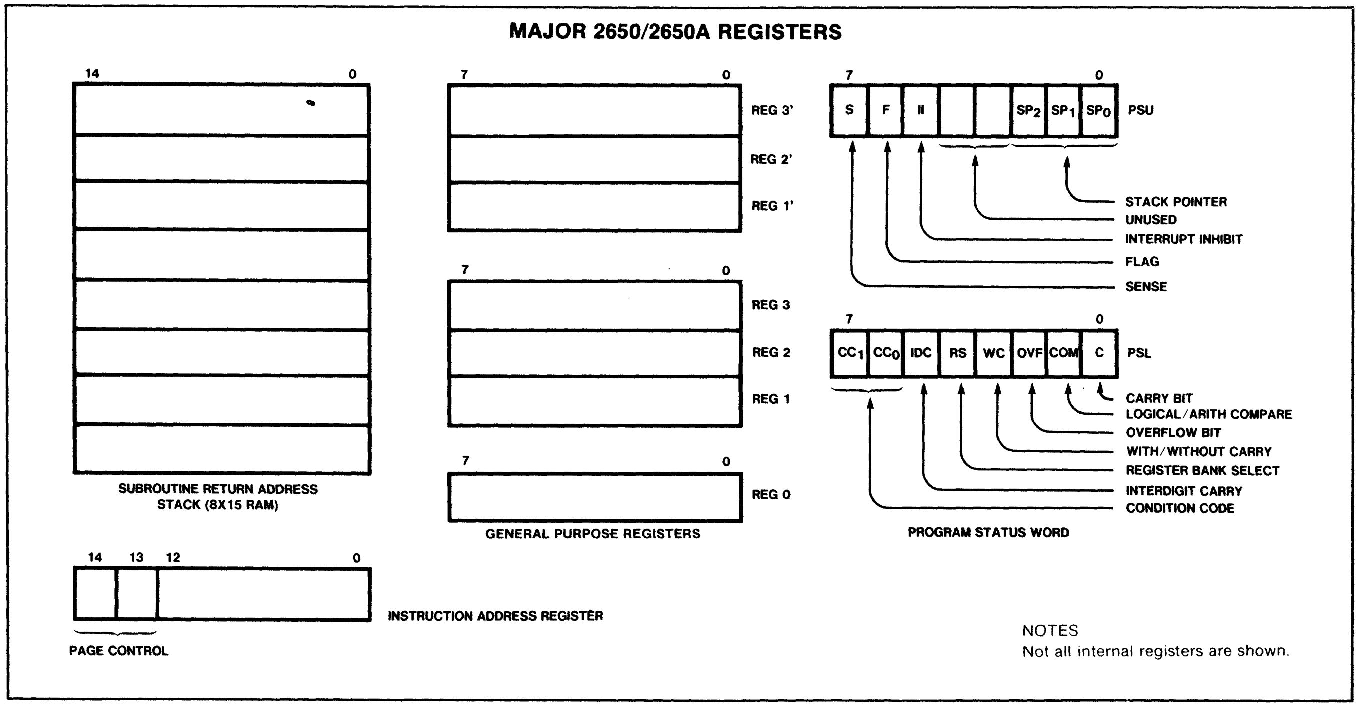


Figure 9.2

The Return Address Stack (RAS) is an 8-level, Last-In, First-Out (LIFO) memory which receives the return address whenever a Branch-to-Subroutine instruction is executed. When a Return instruction is executed, the RAS provides the last return address for the processor's IAR. The stack contains 8 levels of storage so that subroutines may be nested up to 8 levels deep. The Stack Pointer (SP) is a 3-bit wraparound counter that indicates the next available level in the stack. It always points to the current return address. Placing the RAS on the chip allows efficient ROM-only systems to be implemented in some applications.

Figure 9.2 summarizes the 2650 internal registers as seen by the programmer.

Program Status Word

The Program Status Word (PSW) is a major feature of the 2650/2650A which greatly increases its flexibility and processing power. The PSW is a special purpose register within the processor that contains status and control bits. It is 16 bits long and is divided into two bytes called the Program Status Upper (PSU) and Program Status Lower (PSL).

The PSW bits may be tested, loaded, stored, preset or cleared using the instructions which effect the PSW. The Sense bit, however, cannot be set or cleared because it is directly connected to pin 1. The PSW is organized as follows:

PSU	7	6	5	4	3	2	1	0
	S	F	II	Not Used	Not Used	SP2	SP1	SP0

S	Sense	SP2	Stack Pointer Two
F	Flag	SP1	Stack Pointer One
II	Interrupt Inhibit	SP0	Stack Pointer Zero

PSL	7	6	5	4	3	2	1	0
	CC1	CC0	IDC	RS	WC	OVF	COM	C

CC1	Condition Code One	WC	With/Without Carry
CC0	Condition Code Zero	OVF	Overflow
IDC	Interdigit Carry	COM	Logical/Arithmetic Compare
RS	Register Bank Select	C	Carry/Borrow

Sense (S)

The Sense bit in the PSU reflects the logic state of the input to the processor at pin 1. The Sense bit is not affected by the LPSU, PPSU or CPSU instructions.

Flag (F)

The flag bit is a simple latch that drives the FLAG output (pin 40) on the processor.

Interrupt Inhibit (II)

When the Interrupt Inhibit bit is set (II = 1), the processor will not recognize an incoming interrupt. When interrupts are enabled (II = 0), and an interrupt signal occurs, the inhibit bit in the PSU is automatically set. When a Return-and-Enable instruction is executed, the inhibit bit is automatically cleared.

Stack Pointer (SP)

The three stack pointer bits are used to address locations in the Return Address Stack (RAS). The SP designates the stack level which contains the current return address. The SP bits are organized as a binary counter which is automatically incremented with execution of Branch-to-Subroutine instructions and decremented with execution of Return instructions.

Condition Code (CC)

The Condition Code is a 2-bit register which is set by the processor whenever a general purpose register is loaded or modified by the execution of an instruction. Additionally, the CC is set to reflect the result of a Compare instruction or a Test instruction.

The following table indicates the setting of the condition code whenever data is set into a general purpose register. The data byte is interpreted as an 8-bit, two's complement number:

REGISTER CONTENTS	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

For Compare instructions, the data is compared as two 8-bit absolute numbers if the COM bit of the Program Status Lower byte is set to indicate "logical" compare (COM = 1). If the COM bit indicates "arithmetic" compare (COM = 0), the comparison instructions interpret the data bytes as two 8-bit two's complement binary numbers. The CC indicates the result of the comparison as follows:

REGISTER TO MEMORY COMPARE INSTRUCTION	REGISTER TO REGISTER COMPARE INSTRUCTION	CC1	CC0
Reg X greater than Memory	Reg 0 greater than Reg X	0	1
Reg X equal to Memory	Reg 0 equal to Reg X	0	0
Reg X less than Memory	Reg 0 less than Reg X	1	0

The test instructions set the CC to indicate whether the bits in the selected register that correspond to the one's in the mask (second byte of the test instruction) are all one's or not all one's. The following table describes the condition code setting for the test instructions:

RESULTS	CC1	CC0
All of the selected bits are 1s	0	0
Not all of the selected bits are 1s	1	0

The CC is never set to "1" by normal processor operations, but it may be explicitly set to "11" through LPSL or PPSL instruction execution.

Interdigit Carry (IDC)

For BCD arithmetic operations, it is sometimes essential to know if there was a carry from bit 3 to bit 4 during the execution of an arithmetic instruction.

The IDC reflects tha value of the interdigit carry from the previous add or subtract instruction. After any add or subtract instruction execution, the IDC contains the carry or borrow out of bit 3.

The IDC is also modified upon execution of Rotate instructions when the WC bit in the PSW is set. The IDC will reflect the same information as bit 5 of the operand register after the rotate is executed.

Register Select (RS)

There are two banks of general purpose registers with three registers in each bank. The Register Select bit is used to specify which set of general purpose registers will be currently used. Register 0 is common and is always available to the program. An individual instruction may address only 4 registers, but the bank select feature effectively expands the available on-chip registers to 7. When the Register Select bit is 0, registers 1, 2 in register bank 0 will be accessible, and when the bit is 1, registers 1, 2 and 3 in register bank 1 will be accessible.

With/Without Carry (WC)

This bit controls the execution of the Add, Subtract and Rotate instructions.

Whenever an Add or a Subtract instruction executes, the following bits are either set or cleared: Carry/Borrow (C), Overflow (OVF) and Interdigit Carry (IDC). These bits are set or reset without regard to the value of the WC bit. However, when WC = 1, the previous value of the carry bit affects the result of an Add or Subtract instruction, i.e., the carry bit is either added to (Add instruction) or subtracted from (Subtract instruction) the result of the operation.

Whenever a Rotate instruction executes with WC = 0, only the 8 bits of the rotated register are affected. However, when WC = 1, the following bits are also affected: Carry/Borrow (C), Overflow (OVF) and Interdigit Carry (IDC). The Carry /Borrow bit is combined with the 8-bit register to make a 9-bit rotate (see Figure 9.3). The Overflow bit is set whenever the sign bit (bit 7) of the rotated register changes from a 0 to a 1 and is cleared otherwise. The Interdigit Carry bit is set to the new value of bit 5 of the rotated register.

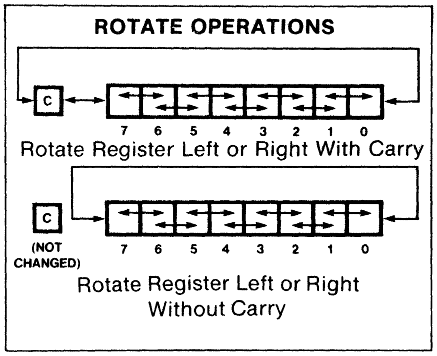

ROTATE OPERATIONS
C
7 6 5 4 3 2 1 0
Rotate Register Left or Right With Carry
C
(NOT CHANGED)
7 6 5 4 3 2 1 0
Rotate Register Left or Right Without Carry

Figure 9.3.

Compare (COM)

The Compare Control bit determines the type of comparison that is executed with the Compare instructions. Either logical or arithmetic comparisons may be made. The arithmetic compare assumes that the comparison is between 8-bit, two's complement numbers (-128 to +127). The logical compare assumes that the comparison is between 8-bit positive binary numbers (0 to +255). When COM is set to 1, the comparisons will be logical, and when COM is set to 0, the comparisons will be arithmetic. See Condition Code (CC).

Carry (C)

The Carry bit is set or cleared by the execution of Add or Subtract instructions. The Carry bit is set to 1 by an Add instruction that generates a carry and a Subtract instruction that does not generate a borrow out of the high-order bit of the ALU. Conversely, an add that does not generate a carry causes the C bit to be cleared as does a subtract instruction that generates a borrow.

Even though a borrow is indicated by a zero in the Carry bit, the processor will correctly interpret the zero during subtract with borrow operations. For a subtract without borrow operation (WC = 0), the processor automatically provides the proper borrow input into the ALU. However, if operations with carry are being performed (WC = 1), the Carry bit must be preset to a 1 by a PPSL instruction in order for the result of a single byte subtraction (or the result of the first subtraction of a multiple byte subtraction) to be correct.

The Carry bit may also be set or cleared by Rotate instructions as described earlier under "With/Without Carry."

To perform an Add with Carry or a Subtract with Borrow, the WC bit must be set (WC = 1).

Overflow (OVF)

The Overflow bit is set during Add instruction execution whenever the two initial operands have the same sign but the result has a different sign. Adding operands with different signs cannot cause overflow. Example: A binary +124 (01111100) added to a binary +64 (01000000) produces a result of (10111100) which is interpreted in two's complement form as a -68. The true answer would be 188, but that answer cannot be contained in the set of 8-bit, two's complement numbers used by the processor, so the OVF bit is set.

The overflow bit is also set during Subtract instruction execution whenever the minuend and the subtrahend have different signs, but the result has a sign that is different from the sign of the minuend. Subtraction of operands with the same signs cannot cause overflow. These conditions are summarized in Table 9.1

TABLE 9.1

SIGN (BIT 7)			ADD OVF	SUB OVF
Operand 1	Operand 2	Result		
+	+	+	0	0
+	+	–	1	0
+	–	+	0	0
+	–	–	0	1
–	+	+	0	1
–	+	–	0	0
–	–	+	1	0
–	–	–	0	0

The Rotate instructions will cause an overflow if the WC bit is set and the sign bit changes from a zero to a 1 as a result of the rotate. If the WC bit is not set, the OVF bit is not affected.

SIGN (BIT 7)		OVF
Before Rotate	After Rotate	
+	+	0
+	–	1
–	+	0
–	–	0

Memory Organization

The 2650/2650A can address memory in locations 0_{10} to $32,767_{10}$. As may be seen in the Instruction section of this chapter, most direct addressing instructions have 13 bits allocated for the direct address. Since 13 bits can only address locations 0_{10} to $8,191_{10}$, a paging system is used to accommodate the entire address range.

The memory may be thought of as being divided into 4 pages of 8,192 bytes each. The addresses in each page are as shown below.

PAGE	START ADDRESS	END ADDRESS	
0	000000000000000	001111111111111	0_{10}-8191_{10}
1	010000000000000	011111111111111	8192_{10}-$16,383_{10}$
2	100000000000000	101111111111111	$16,384_{10}$-$24,575_{10}$
3	110000000000000	111111111111111	$24,576_{10}$-$32,767_{10}$

The low order 13 bits in every page range through the same set of numbers. These 13 bits are the 13 bits addressed by Non-Branch instructions and are also the same 13 bits which are brought out of the 2650/2650A on the address lines ADR0-ADR12.

The two high-order bits of the 15-bit address are known as the page bits. The page bits when examined by themselves represent, in binary, the number of the memory page. Thus, the address 010000001101101 is known as address location 109_{10} in page 1. The page bits, which correspond to ADR13 and ADR14, are brought out of the 2650/2650A on pins 19 and 18.

There are no instructions to explicitly set the page bits. They are set through execution of Direct or Indirect, Branch or Branch-to-Subroutine instructions. These instructions (see Instruction section) have 15 bits allocated for the address field. When such an instruction is executed, the two high-order address bits are set into the page bit latches in the 2650/2650A processor and will appear on ADR 13 and ADR 14 during direct memory accesses until they are specifically changed by another instruction of the branch type.

For memory access from Non-Branch instructions, the 13-bit direct address will address the corresponding location within the current page only. However, the Non-Branch Memory Access instructions may access any byte in any page through indirect addressing which provides the full 15-bit address. In the case of Non-Branch instructions, the page bits are only temporarily changed to correspond to the high order 2 bits of the 15-bit indirect address used to fetch the argument byte. Immediately after the memory access, ADR13 and ADR14 will revert to their previous value.

The consequences of this page address system may be summarized by the following statements:

1. The Reset signal clears both page latches, i.e., ADR13 and ADR14 are cleared to zero.

2. All Non-Branch Direct Memory Access instructions address memory within the current page.

3. All Non-Branch Memory Access instructions may access any byte of addressable memory through use of indirect addressing which temporarily changes the page bits for the argument access. The page bits revert back to their previous state immediately following instruction execution.

4. All Direct and Indirect Addressing Branch instructions set the page bits to correspond to the high order 2 bits of the 15-bit address.

5. Programs may not flow across page boundaries. They must branch to set the page bits.

6. Interrupts always drive the processor to page zero (see Interrupt Mechanism section of this chapter).

Interface

Pin Configuration

The 2650/2650A is packaged in a standard dual-in-line 40-pin package. Figure 9.4 illustrates the pin configuration for the 2650/2650A, and Table 9.2 summarizes the characteristics of the interface signals.

Signal Descriptions

RESET (Pin 16)

The RESET signal is used to cause the 2650/2650A to begin processing from a known state. RESET will normally be used to initialize the processor after powerup or to restart a program. RESET clears the Interrupt Inhibit control bit, clears the internal interrupt-waiting signal and initializes the IAR to zero. RESET is normally low during program execution, and must be driven high to activate the reset function. The leading and trailing edges may be asynchronous with respect to the clock, but the Reset signal must be at least 3 clock periods long. If RESET alone is used to initiate processing, the first instruction will be fetched from Memory location page zero, byte zero after the RESET signal is removed. Any instruction may be programmed for this location including a branch to some program located elsewhere.

Processing can also be initiated by combining an interrupt with a reset. In this case, the first instruction to be executed will be at the interrupt address.

CLOCK (Pin 38)

The CLOCK signal is a positive-going pulse train that determines the instruction execution rate. Three clock periods comprise a processor cycle. Direct instructions are 2, 3 or 4 processor cycles long, depending on the specific type of instruction. Indirect addressing adds 2 processor cycles to the direct instruction times.

ADR (Pins 2-14, 18-19)

The Address signals form a 15-bit path out of the processor and are used primarily to supply memory addresses during memory operations. The addresses remain valid as long as OPREQ is on so that no external address register is required. For extended I/O operations, the low order 8 bits of the ADR lines are used to output the immediate byte of the instruction which typically is interpreted as a device address.

The 13 low order lines of the address are used only for address information. The 2 high order address lines are multiplexed with I/O control information. During memory operations, the lines serve as memory addresses. During I/O operations, they serve as the D/C and E/NE control lines. Demultiplexing is accomplished through use of the Memory/IO control line (see D/C and E/NE below).

ABBREVIATION	PINS	TYPE	FUNCTION	SIGNAL SENSE
GND	1	INPUT	Ground	GND = 0
VCC	1	INPUT	+5 Volts ± 5%	VCC = 1
RESET	1	INPUT	Chip Reset	RESET = 1, causes reset
CLOCK	1	INPUT	Chip Clock	CLOCK = 0 (low), CLOCK = 1 (high)
PAUSE	1	INPUT	Temp. Halt execution	PAUSE = 0, temporarily halts execution
INTREQ	1	INPUT	Interrupt Request	INTREQ = 0, requests interrupt
OPACK	1	INPUT	Operation Acknowledge	OPACK = 0, acknowledges operation
SENSE	1	INPUT	Sense	SENSE = 0 (low) or SENSE = 1 (high)
ADREN	1	INPUT	Address Enable	ADREN = 1 drives into third state
DBUSEN	1	INPUT	Data Bus Enable	DBUSEN = 1 drives into third state
DBUS0-DBUS7	8	IN/OUT	Data Bus	DBUSn = 0 (low), DBUSn = 1 (high)
ADR0-ADR12	13	OUTPUT	Address 0 through 12	ADRn = 0 (low), ADRn = 1 (high)
ADR13 or E/NE	1	OUTPUT	Address 13 or Extended/Non-Extended	Non-Extended = 0, Extended = 1
ADR14 or D/C	1	OUTPUT	Address 14 or Data/Control	Control = 0, Data = 1
OPREQ	1	OUTPUT	Operation Request	OPREQ = 1, requests operation
M/IO	1	OUTPUT	Memory/IO	IO = 0, M = 1
R/W	1	OUTPUT	Read/Write	R = 0, W = 1
FLAG	1	OUTPUT	Flag Output	FLAG = 1 (high), FLAG = 0 (low)
INTACK	1	OUTPUT	Interrupt Acknowledge	INTACK = 1, acknowledges interrupt
RUN/WAIT	1	OUTPUT	Run/Wait Indicator	RUN = 1, WAIT = 0
WRP	1	OUTPUT	Write Pulse	WRP = 1 (pulse), causes writing

Table 9.2 INTERFACE SIGNALS

2650/2650A PIN CONFIGURATION

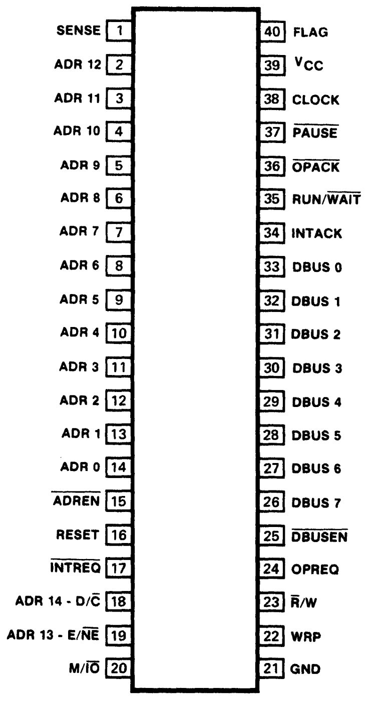

SENSE 1
ADR 12 2
ADR 11 3
ADR 10 4
ADR 9 5
ADR 8 6
ADR 7 7
ADR 6 8
ADR 5 9
ADR 4 10
ADR 3 11
ADR 2 12
ADR 1 13
ADR 0 14
$\overline{\text{ADREN}}$ 15
RESET 16
$\overline{\text{INTREQ}}$ 17
ADR 14 - D/$\overline{\text{C}}$ 18
ADR 13 - E/$\overline{\text{NE}}$ 19
M/$\overline{\text{IO}}$ 20
40 FLAG
39 V_{CC}
38 CLOCK
37 $\overline{\text{PAUSE}}$
36 $\overline{\text{OPACK}}$
35 RUN/$\overline{\text{WAIT}}$
34 INTACK
33 DBUS 0
32 DBUS 1
31 DBUS 2
30 DBUS 3
29 DBUS 4
28 DBUS 5
27 DBUS 6
26 DBUS 7
25 $\overline{\text{DBUSEN}}$
24 OPREQ
23 $\overline{\text{R}}$/W
22 WRP
21 GND

Figure 9.4

ADREN (Pin 15)

The Address Enable signal allows external control of the tri-state address outputs (ADRO-ADR12). When ADREN is driven High, the address lines are switched to their third state and show a high output impedance. This feature allows wired-OR connections with other signals. The ADR13 and ADR14 lines which are multiplexed with other signals are not affected by ADREN.

When a system is not designed to utilize the feature, the ADREN input may be connected permanently to a low signal source.

DBUS (Pins 26-33)

The Data Bus signals form an 8-bit bidirectional data path in and out of the processor. Memory and I/O operations use the data bus to transfer the write or read data to or from memory or the I/O device.

The direction of the data flow on the data bus is indicated by the state of the R/W line. For write operations, the output buffers in the processor output data to the bus for use by memory or by external devices. For read operations, the buffers are disabled and the data condition of the bus is sensed by the processor. The output buffers may also be disabled by the DBUSEN signal.

The signals on the data bus are positive true signals, i.e., a one is a high level and a zero is low.

DBUSEN (Pin 25)

The Data Bus Enable signal allows external control of the tri-state data bus output drivers. When DBUSEN is driven high, the data bus will exhibit a high output impedance. This allows wired-OR connection with other signals.

When a system is not designed to utilize this feature, the DBUSEN input may be permanently connected to a low signal source.

OPREQ (Pin 24)

The Operation Request output is the coordinating signal for all external operations. The M/IO, R/W, E/NE, D/C and INTACK lines are operation control signals that describe the nature of the external operation when the OPREQ line is true. The DBUS and ADR bus also should not be considered valid except when OPREQ is in the high, or on state.

OPREQ will stay on until the external operation is complete, as indicated by the OPACK input. The processor delays all internal activity following an OPREQ until the OPACK signal is received.

OPACK (Pin 36)

The Operation Acknowledge signal is a reply from external memory or I/O devices as a response to the Operation Request signal from the processor. OPREQ is used to initiate an external operation. The affected external device indicates to the processor that the operation is complete by returning the OPACK signal. This procedure allows asynchronous functioning of external devices.

If a memory operation is initiated by the processor, the memory system will provide an OPACK when the requested memory data is valid on the data bus or when the Memory Write operation is completed. If an I/O operation is initiated by the processor, the addressed I/O device may respond with an OPACK as soon as the write data is accepted from the data bus, or after the read operation is completed. If an I/O operation is initiated by the processor, the addressed I/O device may respond with an OPACK as soon as the write data is accepted from the data bus, or after the read operation is completed. However, in order to avoid slowing down the processor when using memories or I/O devices that are just fast enough to keep the processor operating at full speed, the OPACK signal must be returned before the external operation is completed. Any OPACK that is returned within 640ns following an OPREQ will not delay the processor. Data from a read operation can return up to 850ns after an OPREQ is sent and still be accepted by the processor. If all devices will always respond within these time limits, the OPACK line may be permanently connected in the on (low) state. Whenever an OPACK is not available within that time, the processor will delay instruction execution until the first clock following receipt of the OPACK. All output line conditions remain unchanged during the delay, and the processor does not enter the wait state. OPACK is true in the low state and false in the high state.

M/IO (Pin 20)

The Memory/IO output is one of the operation control signals that defines external operations. M/IO indicates whether an operation is memory or I/O, and should be used to gate read or write signals between the 2650/2650A and memory or I/O devices.

The state of M/IO will not change while OPREQ is high. The high state corresponds to a memory operation, and the low state corresponds to an I/O operation.

R/W (Pin 23)

The Read/Write output is one of the operation control signals that defines external operations. R/W indicates whether an operation is read or write. It controls the nature of the external operation and indicates whether the bidirectional DBUS is driving or receiving data. R/W should not be considered valid until OPREQ is on, and the state of the R/W line does not change as long as OPREQ is on.

The high state corresponds to the write operation and the low state corresponds to the read operation.

D/C (Pin 18)

The Data/Control output is an I/O signal which is used to discriminate between the execution of the two types of 1-byte I/O instructions. There are four 1-byte I/O instructions: WRTC, WRTD, REDC, REDD. When Read Control or Write Control is executed, the D/C line takes on the low state which indicates Control (C). When Read Data or Write Data is executed, the D/C line takes on the high state, indicating Data (D). "Data" and "Control" are identifiers only and are not indicative of the type of information which is transferred.

D/C is multiplexed with a high-order address line. When the M/IO line is in the I/O state, the ADR14-D/C line should be interpreted as "D/C." When the

M/IO line is in the M state, the ADR14-D/C line should be interpreted as memory address bit 14.

When the processor responds to an interrupt request with an INTACK, the state of the control lines is equivalent to that occurring during a Read Control operation. Thus, port C may be used to input the interrupt address vector to the data bus. If this type of operation is not desired, INTACK must be used to inhibit the reading of port C.

E/NE (Pin 19)

The Extended/Non-Extended output is the operation control signal that is used to discriminate between 2-byte and 1-byte I/O operations. There are 6 I/O instructions: REDE, WRTE, REDC, REDD, WRTC, WRTD. When either of the 2-byte I/O instructions is executed (REDE, WRTE), the E/NE line takes on the high state or "extended" indication. When any of the 1-byte I/O instructions is executed, the line takes on the low state or "non-extended" indication. Thus, E/NE indicates the presence or absence of valid information on the 8 low-order address lines during I/O operations. E/NE is multiplexed with a high-order address line. When the M/IO line is in the I/O state, the ADR13-E/NE line should be interpreted as "E/NE." When the M/IO line is in the M state, the ADR13-E/NE line should be interpreted as memory address bit 13. E/NE should not be considered valid until: (a) OPREQ is on, and (b) M/IO indicates an I/O operation.

FLAG (Pin 46)

The FLAG output indicates the state of the FLAG bit in the PSW. Any change in the FLAG bit is reflected by a change in the FLAG output. A 1 in the FLAG bit will give a high level on the FLAG output pin. The LPSU, PPSU and CPSU instructions can change the state of the FLAG bit. The FLAG output is always a valid indication of the state of the FLAG bit without regard for the status of the processor or control signals. Changes in the FLAG bit are synchronized with the last cycle of the changing instruction.

SENSE (Pin 1)

The SENSE line provides an input line to the 2650/2650A that is independent of the normal I/O bus structures. The SENSE signal is connected directly to one of the bits in the program status word. It may be stored or tested by an executing program. When a Store (SPSU) or Test (TPSU) instruction is executed, the SENSE line is sampled during the last cycle of the instruction.

Through proper programming techniques, the SENSE signal may be used to implement a direct serial data input channel or it may be used to present any bit of information that the designer chooses.

The SENSE input and FLAG output facilities provide the simplest method of communicating data in or out of the 2650/2650A processor, as neither address decoding nor synchronization with other processor signals is necessary.

PAUSE (Pin 39)

The PAUSE input provides a means for temporarily stopping the execution of a program. When PAUSE is driven low, the 2650/2650A finishes the instruction in progress and then enters the wait state, causing the RUN/WAIT output to go low. When PAUSE goes high, program execution continues with the next instruction, and RUN/WAIT returns to the high state. If PAUSE is turned on and then off again before the last cycle of the current instruction begins, program execution continues without PAUSE. The PAUSE line must be held on until RUN/WAIT goes low or the processor may continue without pausing. If both PAUSE and INTREQ occur prior to the last cycle of the current instruction, the interrupt will be recognized, and an INTACK will be generated immediately following release of PAUSE. The next instruction to be executed will be a ZBSR to service the interrupt.

If an INTREQ occurs while the 2650/2650A is in a wait state due to PAUSE, the interrupt will be acknowledged and serviced after execution of the next normal instruction following release of PAUSE.

RUN/WAIT (Pin 35)

The RUN/WAIT output signal indicates the Run/Wait status of the processor. The wait state may be entered by executing a Halt instruction or by turning on the PAUSE input. At any other time, the processor will be in a run state.

When the processor is executing instructions, the line is in the high or run state; when in the wait state, the line is held low.

The Halt-initiated wait condition can be changed to run by a RESET or an Interrupt. The PAUSE-initiated wait condition can be changed to run by removing the PAUSE input.

If a RESET occurs during a PAUSE-initiated wait state and the PAUSE remains low, the processor will be reset, fetch one instruction from page zero byte zero and return to the wait state. When the PAUSE is eventually removed, the previously fetched instruction will be executed.

INTREQ (Pin 17)

The Interrupt Request input (normally high) is a means for external devices to change the flow of program execution. When the processor recognizes an INTREQ, i.e., INTREQ is driven low, it finishes the instruction in progress, inserts a ZBSR instruction into the IR, turns on the Interrupt Inhibit bit in the PSU, and then responds with INTACK and OPREQ signals. Upon receipt of INTACK, the interrupting device may raise the INTREQ line and present a data byte to the processor on the DBUS. The required byte takes the same form as the second byte of a ZBSR instruction. Thus, the interrupt initiated Branch-to Subroutine instruction may have a relative target address anywhere within the first or last 64 bytes of memory page 0. If indirect addressing is specified, a branch to any location in addressable memory is possible.

The relative address presented by the interrupting device is handled with a normal I/O Read sequence using the usual interface control signals. The addition of the INTACK signal distinguishes the Interrupt Address operation from other operations that may take place as part of the execution of the interrupted instruction. At the same time that it acknowledges the INTREQ, the processor automatically sets the bit that inhibits recognition of further interrupts. The Interrupt Inhibit bit may be cleared anytime during the interrupt service routine, or a Return-and-Enable instruction may be used to enable interrupts upon leaving the routine. If an INTREQ is waiting when the Interrupt Inhibit bit is cleared, it will be recognized and processed immediately without the execution of an intervening instruction.

INTACK (Pin 34)

The Interrupt Acknowledge signal is used by the processor to respond to an external interrupt. When an INTREQ is received, the current instruction is completed before the interrupt is serviced. When the processor is ready to accept the interrupt, it sets INTACK to the high, or on, state along with OPREQ. The interrupting device then presents a relative address byte to the DBUS and responds with an OPACK signal. INTREQ may be turned off anytime following INTACK. INTACK will fall after the processor receives the OPACK signal.

WRP (Pin 22)

The Write Pulse output is a timing signal from the processor that provides a positive-going pulse in the middle of each requested write operation (memory or I/O) and a high level during read operations. The WRP is designed to be used with Signetics' 2606 memory circuits to provide a timed chip enable signal. For use with memory, it may be gated with the M/IO signal to generate a memory write pulse.

Because the WRP pulse occurs during any write operation, it may also be used with I/O write operations where convenient.

Signal Timing

The clock input to the 2650/2650A provides the basic timing information that the processor uses for all its internal and external operations. The clock rate determines the instruction execution time, except to the extent that external memories and devices slow the processor down. The maximum clock rate of the standard 2650/2650A is 1.25 megacycles (1 clock period = 800ns minimum). One unique feature of the 2650/2650A is that the clock frequency may be slowed down to dc, allowing complete timing flexibility for interfacing. This feature permits single stepping the clock which can greatly simplify system checkout. It also provides an easy method to halt the processor. Each 2650/2650A cycle is comprised of 3 clock periods. Direct instructions require either 2, 3 or 4 processor cycles for execution and, therefore, vary from 4.8 to 9.6us in duration.

OPREQ is the master control signal that coordinates all operations external to the processor. Many of the other signal interactions are related to OPREQ. The timing diagrams (Figures 9.5, 9.6 and 9.7) assume that the clock periods are constant and that OPACK is returned in time to avoid delaying instruction execution. In that case, OPREQ will be high for 1.5 clock periods and then will be low for another 1.5 clock periods.

The interface control signals have been designed to allow implementation of asynchronous interfaces for both memory and input/output devices. The control signals are relatively simple and provide the following advantages: no external synchronizing is necessary, external devices may run at any data rate up to the processor's maximum I/O data rate, and, because data signals are furnished with guard signals, the external devices are often relieved of the necessity of latching information.

The timing diagrams (Figures 9.5, 9.6, and 9.7) are for illustrative purposes only and are not meant to convey precise timing relationships. Consult the 2650/2650A data sheet for detailed DC and AC parameter information.

Memory Read

The timing for a typical Memory Read operation is shown in Figure 9.5. When reading memory, the 2650/2650A simultaneously switches OPREQ to the high state, M/IO to M (memory), R/W to R (read), and places the memory address on lines ADR0-ADR14. Even though the ADR13 and ADR14 lines are multiplexed with I/O control information, they contain valid address data during memory operations, so special demultiplexing or gating circuitry is not required.

Once the memory logic has determined the simultaneous existence of the signals mentioned above, it places the true data corresponding to the given address location on the data bus (DBUS0-DBUS7), and returns an OPACK signal to the processor. The processor, recognizing the OPACK, strobes the data into the receiving register and lowers OPREQ. This completes the Memory Read sequence.

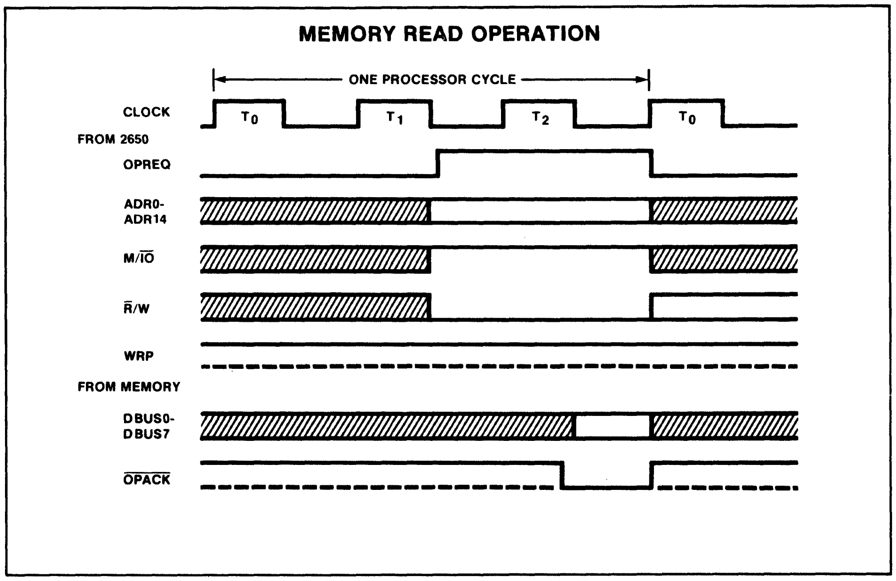


Figure 9.5

If the OPACK signal is delayed by the memory device, the processor waits until it is received. OPREQ is lowered only after the receipt of OPACK. The memory device should raise OPACK after OPREQ falls. If the memory will always respond within the allowed time, the OPACK input may be left permanently in the low state.

Memory Write

The signals involved with the processor's Memory Write sequence are similar to those used in the Memory Read sequence with the following exceptions:

1. The R/W signal is in the write state; and
2. The WRP signal provides a positive-going pulse during the write sequence which may be used as a chip enable, write pulse, etc.

Figure 9.6 demonstrates the signals that occur during a memory Write operation.

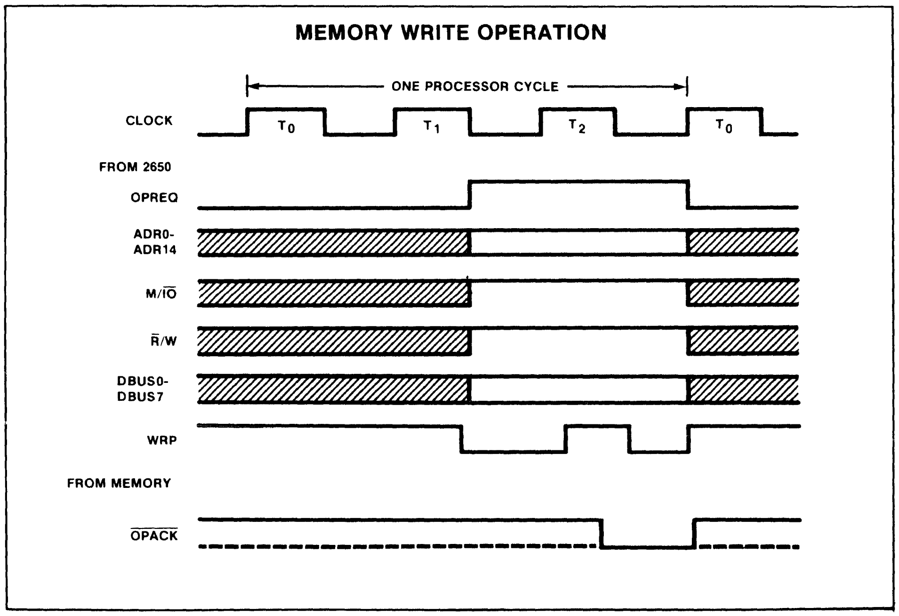


Figure 9.6

I/O Device Read

The timing sequences for the I/O Read instructions are the same as the Memory Read sequences with the following exceptions: The M/IO signal is switched to IO, the ADR13 signal becomes the E/NE (Extended/Non-Extended) signal, and for Non-Extended instructions, the ADR14 signal becomes the D/C (Device/Control) signal. The address lines only contain valid information for extended instructions.

Figure 9.7 shows the signals that occur for an I/O Device Read operation.

I/O Device Write

The timing sequences for I/O Write operations are similar to those shown in Figure 9.7 for an I/O Read operation except that the R/W signal is in the wait (high) state and the WRP signal provides a positive-going pulse during the OPREQ time. In addition, the data bus signals are provided by the 2650/2650A.

A Minimal System Example

The 2650/2650A has been designed for low cost, easy interfacing, which is illustrated by a minimal cȯnfiguration shown in Figure 9.8. This system has a Teletype interface, 1024 bytes of ROM, and 256 bytes of RAM, yet requires only 7 standard integrated circuit packages. The ROM can contain a bootstrap loader and I/O driver programs for the Teletype. Other programs could reside in ROM or be read into RAM via the Teletype. An alternative to the 2608 n-channel MOS ROM is the 82S115 bipolar PROM which offers a 512X8 organization. Only one +5 volt power supply is required for this system. The advantages of conceptual simplicity and minimum system costs of the 2650/2650A approach will become obvious to the system designer, particularly when compared with alternative microprocessor products.

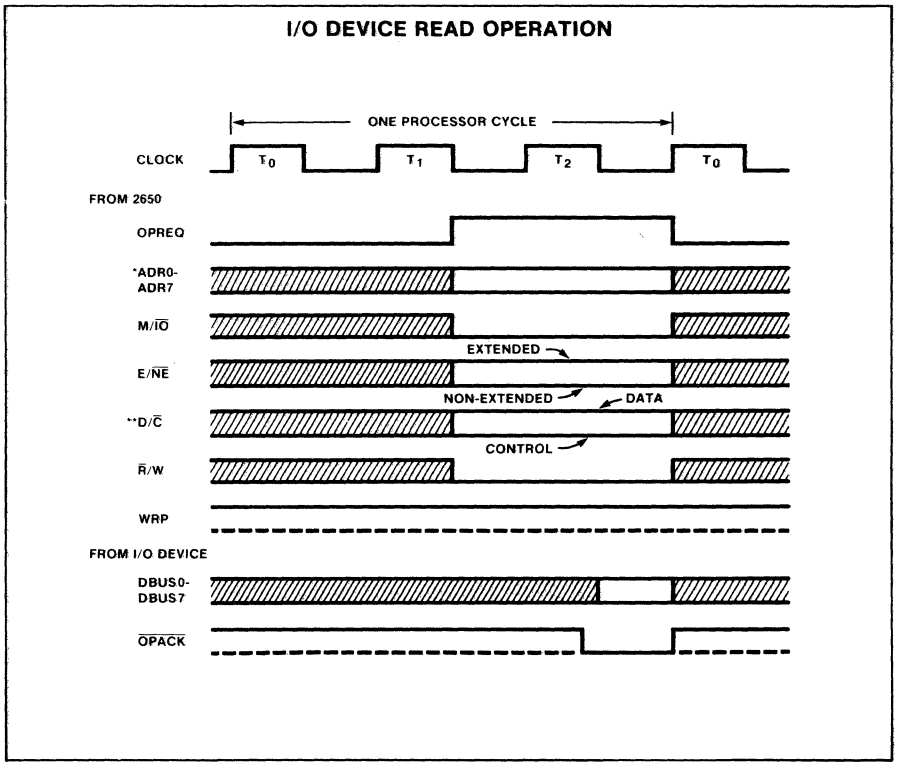
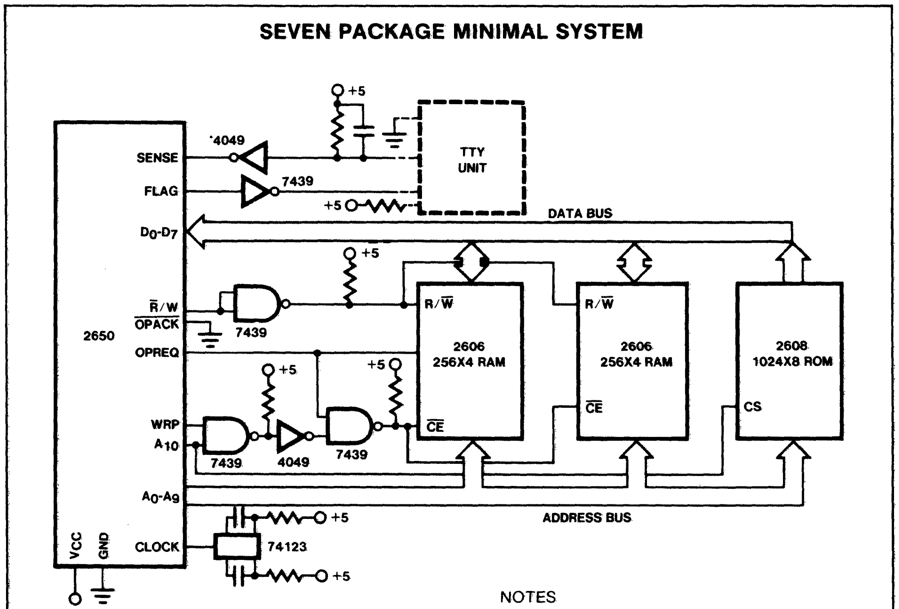


Figure 9.7



NOTES

1. One +5V supply seven IC packages
2. *CMOS receiver used for high noise immunity.

Figure 9.8

Input/Output Facilities

The 2650/2650A processor provides several mechanisms for performing input/output functions. They are Flag and Sense, Non-Extended I/O instructions, Extended I/O instructions and Memory I/O. These four facilities are described below.

Flag and Sense I/O

The 2650/2650A has the ability to directly output 1 bit of data without additional address decoding or synchronizing signals.

The bit labeled "Flag" in the Program Status Word is connected through a TTL-compatible driver to the chip output at pin 40. The Flag output always reflects the value in the Flag bit. When a program changes the Flag bit through execution of an LPSU, PPSU, or CPSU instruction, the bit will be set or cleared during the last cycle of the instruction that changes it.

The Flag bit may be used conveniently for many different purposes. The following is a list of some possible uses:

1. A serial output channel
2. An additional address bit to increase addressing range.
3. A switch or toggle output to control external logic.
4. The origin of a pulse for polling chains of devices.

The Sense bit performs the complementary function of the Flag and is a single bit direct input to the 2650/2650A. The SENSE input, pin 1, is connected to a TTL-compatible receiver and is then routed directly to a bit position in the Program Status Word. The bit in the PSW always represents the value of the external signal. It may be sampled anytime through use of the TPSU or SPSU instructions.

This input to the processor may be used in many ways. The following is a list of some possible uses:

1. A serial input channel.
2. A sense switch input.
3. A break signal to a processing program.
4. An input for yes/no signaling from external devices.

Non-Extended I/O

There are four 1-byte I/O instructions: REDC, REDD, WRTC and WRTD. They are all referred to as non-extended because they can communicate only 1 byte of data, either into or out of the 2650/2650A.

REDC and REDD cause the input transfer of 1 byte of data. They are identical except for the fact that the D/C signal is in the D state for REDD and in the C state for REDC. Similarly, the instructions WRTC and WRTD cause an output transfer of 1 byte of data. The D/C line discriminates between the 2 pairs of input/output instructions, and can be used as a 1-bit device address in simple systems.

The IO and NE signals inform the devices outside the 2650/2650A that a 1-byte I/O instruction is being executed. The D/C line indicates which pair of the

1-byte I/O instructions are being executed; D implies either WRTD or REDD, and C implies either WRTC or REDC. Finally, the R/W signal level specifies whether a read or a write is being performed.

Extended I/O

There are two 2-byte I/O instructions: REDE and WRTE. When these instructions are executed, the second byte of the instruction is output on the low order address lines ADR0-ADR7. REDE causes the byte of data then on the data bus to be strobed into the register specified in the instruction to be output on the data bus.

The 2-byte I/O instructions are similar to the 1-byte I/O instructions except the D/C line is not considered, and the data from the second byte of the I/O instruction appears on the address bus during the time that OPREQ is valid. The data on the address bus is intended to convey a device address, but may be utilized for any purpose.

Memory I/O

The 2650/2650A user may choose to transfer data into or out of the processor using the memory control signals. The advantage of using this technique is that the data can be read or written by the program with memory reference instructions, and data may be directly operated upon with the arithmetic and logical instructions. The memory reference instructions can use the various addressing modes provided by the 2650/2650A, such as indexing and indirect addressing.

To make use of this technique, the designer must assign memory addresses to I/O devices and design the device interfaces to respond to the same signals as memory.

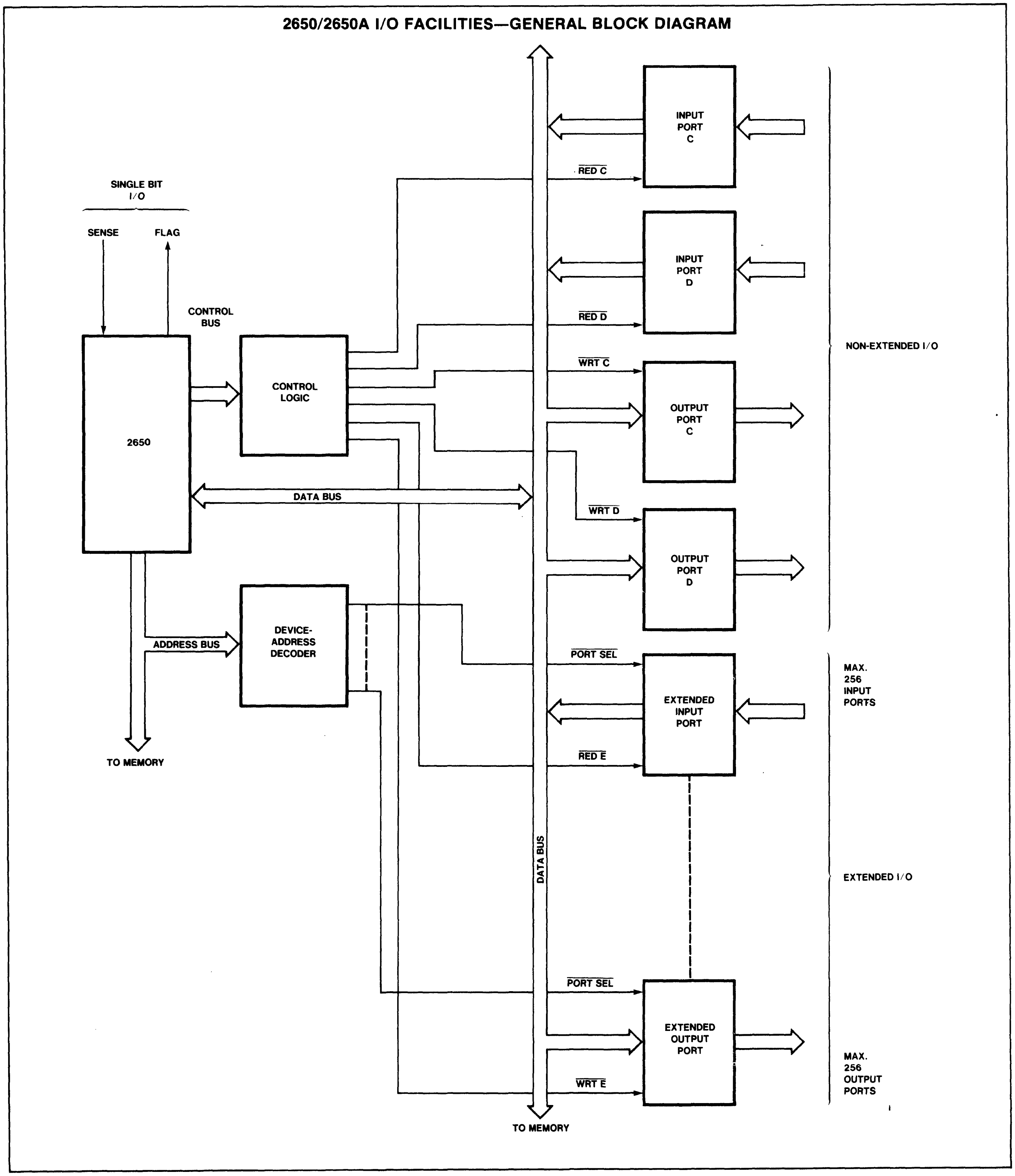
A possible disadvantage of this method is that it may be necessary to decode more address lines to determine the device address than with other I/O facilities.

Table 9.3 summarizes the I/O signal states for the various types of I/O facilities, and Figure 9.9 illustrates the types of I/O available with the 2650/2650A.

TYPE OF I/O OPERATION	OPREQ	M/$\overline{\text{IO}}$	$\overline{\text{R}}$/W	ADR0-ADR7	ADR13 (E/$\overline{\text{NE}}$)	ADR14 (D/$\overline{\text{C}}$)
Sense input	X	X	X	X	X	X
Flag output	X	X	X	X	X	X
Extended read	H	L	L	Second byte	H	X
Extended write	H	L	H	of instruction		X
Non-extended read C	H	L	L	X	L	L
Non-extended read D	H	L	L	X	L	H
Non-extended write C	H	L	H	X	L	L
Non-extended write D	H	L	H	X	L	H
Memory I/O read	H	H	L	ADR0-ADR7	ADR13	ADR14
Memory I/O write	H	H	H	ADR0-ADR7	ADR13	ADR14

X = Don't care

Table 9.3 I/O INTERFACE SIGNAL STATE



2650/2650A I/O FACILITIES—GENERAL BLOCK DIAGRAM

Figure 9.9

Interrupt Mechanism

The 2650/2650A has been implemented with a single level, address vectoring interrupt mechanism. There is 1 interrupt input pin. When an external device generates an Interrupt signal (INTREQ), the processor is forced to transfer control to any of 128 possible memory locations as determined by an 8-bit vector supplied by the interrupting device on the data bus. The device may also return an Indirect Address signal which causes the processor to enter an indirect addressing sequence. This enables a device to direct the processor to execute code anywhere within addressable memory.

Upon recongnizing the Interrupt signal, the processor automatically sets the Interrupt Inhibit bit in the Program Status Word. This inhibits further interrupts from being recognized until the interrupt routine is finished executing and a Return-and-Enable instruction is executed or the Inhibit bit is explicitly cleared.

When the Inhibit bit in the PSW is set (II=1), the processor will not recognize an interrupt input. The Interrupt Inhibit bit may be set under program control (LPSU, PPSU) and is automatically set whenever the processor accepts an interrupt. The Inhibit bit may be cleared in 3 ways:

1. By a Reset operation.

2. By execution of an appropriate Clear or Load PSU instruction (CPSU, LPSU).

3. By execution of a Return-and-Enable instruction.

The sequence of events for an Interrupt operation is as follows:

1. An executing program enables interrupts.

2. The external device initiates an Interrupt with the INTREQ line.

3. The processor finishes executing the current instruction.

4. The processor sets the Inhibit bit in the PSW.

5. The processor inserts the first byte of a ZBSR (Zero Branch-to Subroutine, Relative) instruction.

6. The processor accesses the data bus to fetch the second byte of the ZBSR instruction.

7. The interrupting device responds to the processor-generated INTACK (Interrupt Acknowledge) by supplying the requested second byte.

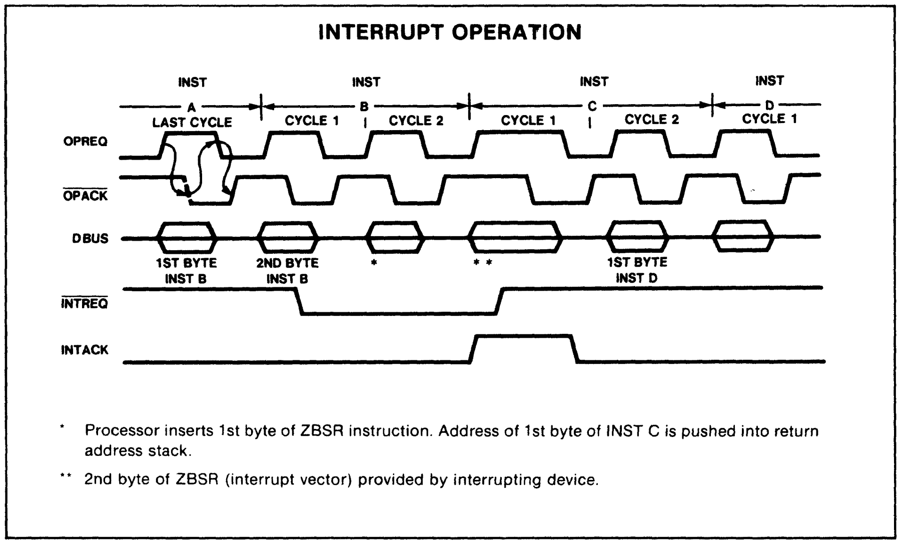
8. The processor executes the Zero Branch-to-Subroutine instruction, saving the address of the next sequential instruction in the RAS, and proceeds to execute the instruction at the address relative to page 0, byte 0 given by the interrupting device.

9. When the Interrupt routine is complete, a Return instruction (RETC, RETE) pulls the address from the RAS and execution of the interrupted program resumes. If the instruction is an RETE, interrupts are again enabled.

Since the interrupting device specifies the interrupt subroutine address in the standard relative address format, it has considerable flexibility with regard to the interrupt procedure. It can point to any location that is within +63 or -64 bytes of page 0, byte 0 of memory. (Negative relative addresses wrap around the memory, modulo 8,192$_{10}$ bytes.) The interrupting device also may specify whether the subroutine address is direct or indirect by providing a zero or one to DBUS7 (pin 26).

The vectored interrupt technique requires that each interrupting device contain the hardware required to provide the address vector to the processor. In some cases, an overall reduction in system hardware may be realized by implementing a "polled" interrupt scheme. In this case, the INTACK generated as a response to any interrupt is used to force an address byte on the data bus. The program at this address sequentially polls all devices to determine the interrupting device, and then branches to a program to service that device. The disadvantages of this technique are increased coding requirements and slower response to the interrupt.

The timing diagram in Figure 9.10 illustrates how the interrupt system works in the processor. The execution of the instruction labeled "A" has been proceeding before the start of this diagram. The last cycle of instruction A is shown. Notice that, as in all external operations, the OPREQ output eventually causes an OPACK input, which, in turn, allows OPREQ to be turned off. The arrows show this sequence of events. The last cycle of instruction A fetches the first byte of instruction B from memory and inserts it into the Instruction Register.


INTERRUPT OPERATION

* Processor inserts 1st byte of ZBSR instruction. Address of 1st byte of INST C is pushed into return address stack.

** 2nd byte of ZBSR (interrupt vector) provided by interrupting device.

Figure 9.10

Assume that instruction B is a 2 cycle, 2-byte instruction such as ADD. Since the first byte has already been fetched by instruction A, the first cycle of instruction B is used to fetch the second byte of instruction B. Had the interrupt not occurred during instruction B, it would have fetched the first byte of the next sequential instruction during its second (last) cycle.

Since an Interrupt occurred, however, the processor uses the last cycle of B to jam the Interrupt instruction (ZBSR) execution into the instruction register. Notice that the INTREQ input can arrive at any time prior to the last (second) cycle of execution of instruction B and that execution of instruction B is completed.

Instead of being the next sequential instruction following B, instruction C is the execution of the Interrupt. The first cycle of C is used to fetch the second byte of the ZBSR instruction from the DBUS as provided by the interrupting device. This request is indicated by the presence of the INTACK control signal. The INTREQ may then be removed. When the device responds with the requested byte, it uses a standard Operation Acknowledge procedure (OPACK) to so indicate to the processor. During the second cycle of instruction C the processor executes the ZBSR instruction, and fetches the first byte of instruction D which is the first instruction of the interrupt subroutine.

Subroutine Linkage

The on-chip stack, along with the Branch-to Subroutine and Return instructions, provide the facility to transfer control to a subroutine. The subroutine can return control to the program that branched to it via a Return instruction.

The stack is eight levels deep and operates on a last-in, first-out basis. This means that a routine may branch to a subroutine, which may branch to another subroutine, which may branch to another subroutine, etc., eight times before any Return instructions are executed.

When designing a system that utilizes interrupts, it should be remembered that the processor jams a ZBSR into the IR and then executes it. This will cause an entry to be pushed into the on-chip stack like any other Branch-to-Subroutine instruction and may limit the stack depth available in certain programs.

When branching to a subroutine, the following sequence of events occur:

1. The address in the IAR is used to fetch the Branch-to-Subroutine instruction and is then incremented in the Address Adder so that it points to the instruction following the subroutine branch.

2. The Stack Pointer is incremented by ones so that it points to the next Return Address Stack location.

3. The contents of the IAR are stored in the stack at the location designated by the Stack Pointer.

4. The operand address contained in the Branch-to-Subroutine instruction (the address of the first instruction of the subroutine) is inserted into the IAR.

When returning from a subroutine, this sequence of events occurs:

1. The address in the IAR is used to fetch the return (RETC, RETE) instruction from memory.

2. When the Return instruction is recognized by the processor, the contents of the stack entry pointed to by the Stack Pointer is placed into the IAR.

3. The Stack Pointer is decremented by one.

4. Instruction execution continues at the address now in the IAR.

Condition Code Usage

The 2-bit register called the Condition Code is incorporated in the Program Status Word. It may be seen in the description of the 2650/2650A instructions that the Condition Code (CC) is specifically set by every instruction that causes data to be transferred into a general-purpose register and by Compare and Test instructions.

The reason for this design feature is that after an instruction executes, the CC contains a modest amount of information about the byte of data which has just been manipulated. Thus, when a program loads a register with a byte of unknown data, the Condition Code setting indicates whether the byte is positive, negative or zero. A negative indication, for example, implies that bit 7 is set to one.

Consequently, a data manipulation operation, when followed by a conditional branch, is often sufficient to determine desired information without resorting to a specific test, thus saving instructions and memory space.

Start-up Procedure

The 2650/2650A must be started in an orderly fashion to assure that the internal control logic begins in a known state.

Assuming power is applied to the chip and the clock input is running, the easiest way to start is to apply a Reset signal for at least three clock periods. When the Reset signal is removed, the processor will fetch the instruction at page 0, byte 0 and commence instruction execution.

To start processing at a different address, a more complex start-up procedure may be employed. If an Interrupt signal is applied initially along with the Reset, processing will commence at the address provided by the interrupting device. Recall that the address provided may include a bit to specify indirect addressing, and therefore the first instruction executed may be anywhere within addressable memory. The Reset and Interrupt signal may be applied simultaneously and when the Reset is removed, the processor will execute the usual interrupt signal sequence as described in "Interrupt Mechanism."

Instructions

Addressing Modes

An addressing mode is a method the processor uses for developing argument addresses for machine instructions.

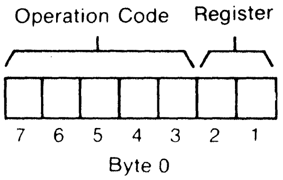
The 2650/2650A processor can develop addresses in eight ways:

- Register addressing
- Immediate addressing
- Relative addressing
- Relative, indirect addressing
- Absolute addressing
- Absolute, indirect addressing
- Absolute, indexed addressing
- Absolute, indirect, indexed addressing

However, of these eight addressing modes, only four are basic. The others are variations due to indexing and indirect addressing. The basic addressing mode of each instruction is indicated in the first line of each detailed instruction description. The following text describes how effective addresses are developed by the processor.

Register Addressing

All register-to-register instructions are one byte in length. Instructions utilizing this addressing mode appear in this general format:

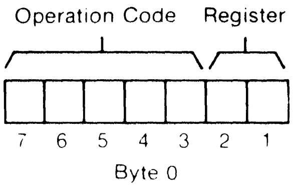


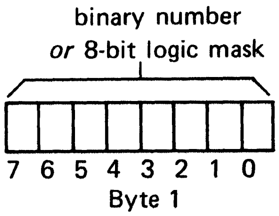
Since there are only two bits designated to specify a register, register zero always contains one of the operands while the other operand is in one of the three registers in the currently selected bank. Register zero may also be specified as the explicit operand giving instructions such as: LODZ R0.

In 1-byte register addressing instructions which have just one operand, any of the currently selected general-purpose registers or register zero may be specified, e.g., RRL, R0.

Immediate Addressing

All immediate addressing instructions are two bytes in length. Usually, the first byte contains the operation code and register designation, while the second byte contains data used as the argument during instruction execution. In some cases, the entire eight bits of the first byte are used for the operation code.

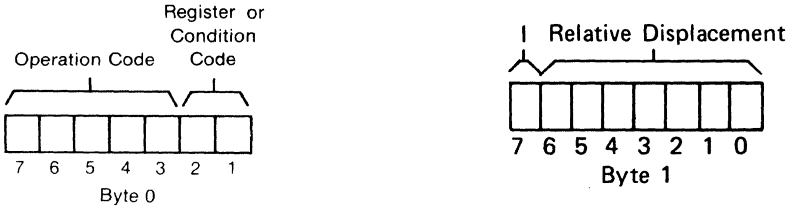




The second byte, the data byte, may contain a binary number or a logic mask depending on the particular instruction being executed. Any register may be designated in the first byte.

Relative Addressing

Relative addressing instructions are all two bytes in length and may be of the branch or non-branch type. The format of relative addressing instructions is:



For branch type instructions, the first byte contains the operation code and the condition code value for which the branch will take place. For non-branch instructions, one argument is a register and the second argument is the contents of a memory location, and the first byte contains the operation code and register designation.

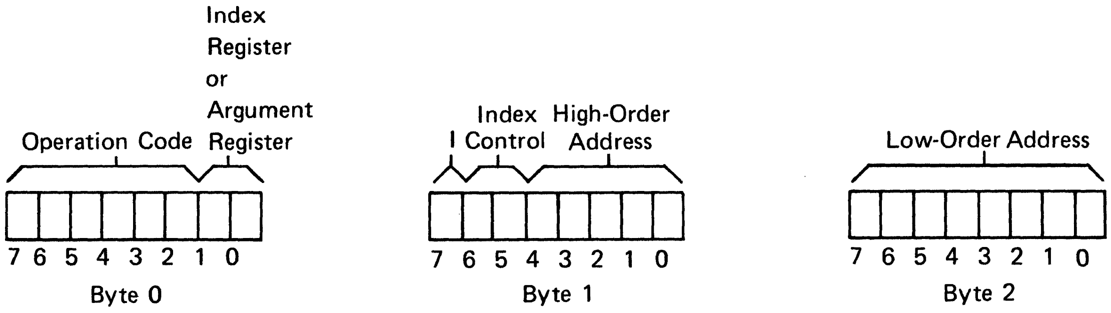
For either type, the second byte contains the relative address. Bits 0-6, byte 1, contain a 7-bit two's complement binary number which can range from -64 to +63. This number is used by the processor to calculate the effective address. The effective address is calculated by adding the address of the first byte following the Relative Addressing instruction to the relative displacement in the second byte of the instruction.

If bit 7, byte 1 is set to 1, the processor will enter an indirect addressing cycle, where the actual operand or branch address will be accessed from the effective address location. See Indirect Addressing.

Two of the branch instructions (ZBSR, ZBRR) allow addressing relative to page 0, byte 0 of memory. In this case, values up to +63 reference the first 62 bytes of page 0 and values up to -64 reference the last 64 bytes of page 0.

Absolute Addressing for Non-Branch Instructions

Non-branch, absolute addressing instructions are all 3 bytes in length and are memory reference instructions. One argument of the instruction is a register, designated in bits 1 and 0, byte 0; the other argument is the contents of a memory location. The format of these instructions is:



Bits 4-0, byte 1 and 7-0, byte 2 contain the absolute address and can address any byte within the same page that contains the instruction.

The index control bits, bits 6 and 5, byte 1, determine how the effective address will be calculated and possibly which register will be the argument during instruction execution. The index control bits have the following interpretation:

INDEX CONTROL		MEANING
Bit 6	Bit 5	
0	0	Non-indexed address
0	1	Indexed with auto-increment
1	0	Indexed with auto-decrement
1	1	Indexed only

When the index control bits are 0 and 0, bits 1 and 0 in byte 0 contain the argument register designation and bits 0 to 4, byte 1 and bits 0 to 7, byte 2 contain the effective address. Indirect addressing may be specified by setting bit 7, byte 1 to a one.

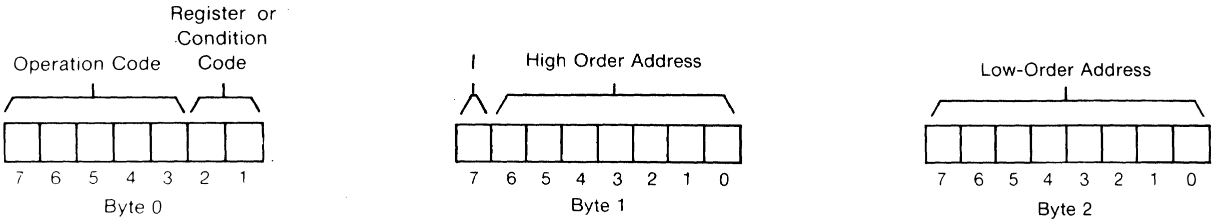
When the index control bits are 1 and 1, bits 1 and 0 in byte 0 designate the index register and the argument register implicitly becomes register 0. The effective address is calculated by adding the contents of the index register (interpreted as an 8-bit positive integer) to the address field. If indirect addressing is specified, the indirect address is accessed and then the value in the index register is added to the indirect address. This is commonly called post indexing.

When the index control bits contain 0 and 1, the address is calculated by the processor exactly as when the control bits contain 1 and 1 except a binary 1 is added to the contents of the selected index register before the calculation of the effective address proceeds. Similarly, when the index control bits contain 1 and 0, a binary 1 is subtracted from the contents of the selected index register before the effective address is calculated.

Indexing across page boundaries is not allowed. This is true even if indirect addressing with indexing is specified. Attempts to index across the top of a page will result in an address at the bottom of the same page.

Absolute Addressing for Branch Instructions

The 3-byte, absolute addressing, branch instructions deviate slightly in format from non-branch absolute addressing instructions as shown below:



The notable difference is that bits 6 and 5, byte 1, are no longer interpreted as index control bits, but instead are interpreted as the high order bits of the address field. This means that there is no indexing allowed on most absolute addressing branch instructions. However, indexed branches are possible through use of the BXA and BSXA instructions. Bits 6 and 5, byte 1, are used to set the current page register, thus enabling programs to directly transfer control to another page (see Memory Organization, BXA and BSXA instructions, and Indirect Addressing).

Indirect Addressing

Indirect addressing means that the argument address of an instruction is not specified by the instruction itself, but rather the argument address will be found in the 2 bytes pointed to by the address field, or relative address field, of absolute or relative addressing instructions. In both cases, the processor will enter the indirect addressing mode when the bit designated "I" is set to 1. Entering the indirect addressing sequence adds 2 cycles (6 clock periods) to the execution time of an instruction.

Indirect addresses are 15-bit addresses stored right- justified in 2 contiguous bytes of memory. As such, an indirect address may specify any location in addressable memory (0-32,767). The high order bit of the 2-byte indirect address is not used by the processor. In the case of absolute addressing, with indexing specified, the value of the index register is added to the indirect address, not to the value in the address field of the instruction.

Only single level indirect addressing is implemented. The examples below demonstrate indirect addressing.

In Figure 9.11, the LODA instruction in memory locations 10, 11, and 12 specifies indirect addressing (bit 7, byte 1, is set). Therefore, when the instruction is executed, the processor takes the address field value, H'51', and uses it to access the 2-byte indirect address at 51 and 52. Then using the contents of 51 and 52 as the effective address, the data byte containing H'67' is loaded into register 2.

The example In Figure 9.12 is, in a fashion, similar to the previous example; the relative address is used to access the indirect address which points to the data byte. When the LODR instruction is executed, the data byte contents, H'67', will be loaded into register 2.

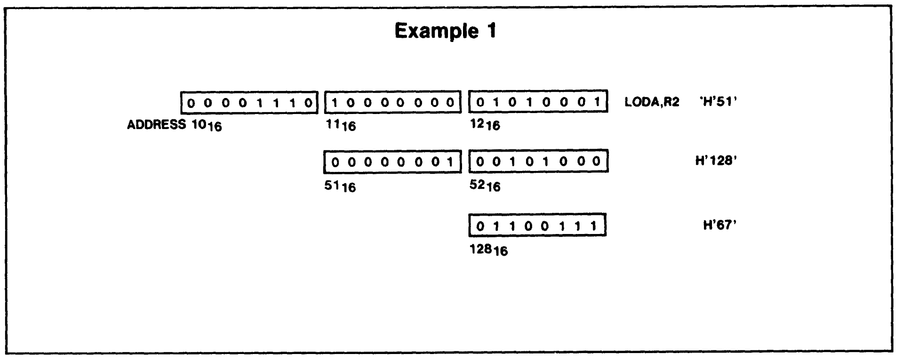


Figure 9.11

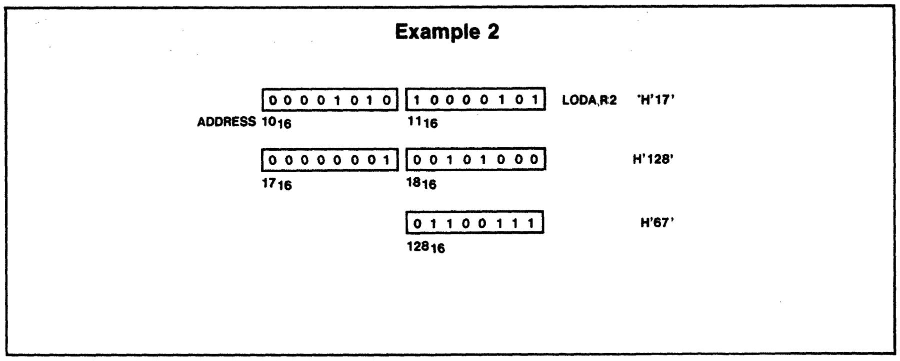


Figure 9.12

Instruction Format Exceptions

There are several instructions which are detected by decoding the entire 8 bits of the first byte of the instruction. These instructions are unique and may be noticed in the instruction descriptions. Examples are HALT, CPSU and CPSL.

Of this type of instruction, 2 operation codes were taken from otherwise complete sets, thus eliminating certain possible operations. The cases are as follows:

(Not okay) (Okay	STRZ 0 NOP	Storing register zero into register zero is not implemented; the operation code is used for NOP (no operation).
(Not okay) (Okay)	ANDZ 0 HALT	AND of register zero with register zero is not implemented; the operation code is used for HALT.

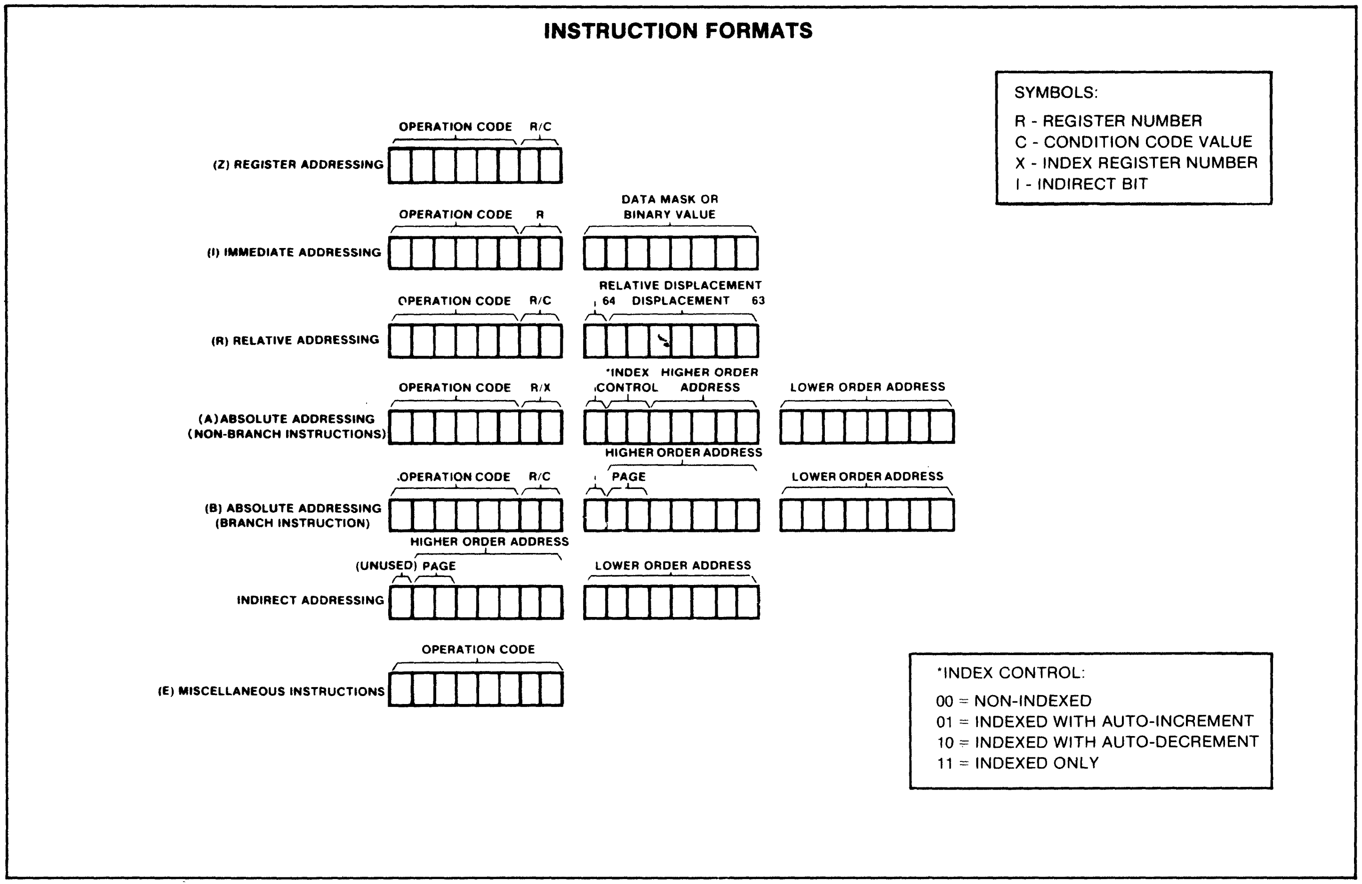

INSTRUCTION FORMATS
SYMBOLS:
R - REGISTER NUMBER
C - CONDITION CODE VALUE
X - INDEX REGISTER NUMBER
I - INDIRECT BIT
OPERATION CODE
R/C
(Z) REGISTER ADDRESSING
OPERATION CODE
R
DATA MASK OR BINARY VALUE
(I) IMMEDIATE ADDRESSING
OPERATION CODE
R/C
RELATIVE DISPLACEMENT
64
DISPLACEMENT
63
(R) RELATIVE ADDRESSING
OPERATION CODE
R/X
*INDEX CONTROL
HIGHER ORDER ADDRESS
LOWER ORDER ADDRESS
(A) ABSOLUTE ADDRESSING (NON-BRANCH INSTRUCTIONS)
HIGHER ORDER ADDRESS
OPERATION CODE
R/C
I
PAGE
LOWER ORDER ADDRESS
(B) ABSOLUTE ADDRESSING (BRANCH INSTRUCTION)
HIGHER ORDER ADDRESS
(UNUSED)
PAGE
LOWER ORDER ADDRESS
INDIRECT ADDRESSING
OPERATION CODE
(E) MISCELLANEOUS INSTRUCTIONS
*INDEX CONTROL:
00 = NON-INDEXED
01 = INDEXED WITH AUTO-INCREMENT
10 = INDEXED WITH AUTO-DECREMENT
11 = INDEXED ONLY

Figure 9.13

THIS PAGE INTENTIONALLY BLANK

Introduction

The 2650/2650A uses variable-length instructions that are 1, 2 or 3-bytes long. The instruction length is determined by the nature of the operation being performed and the addressing mode being used. Thus, the instruction can be expressed in 1 byte when no memory operand addressing is necessary, as with register-to-register or rotate instructions. On the other hand, for direct addressing instructions, 3 bytes are allocated. The relative and immediate addressing modes allow 2-byte instructions to be implemented.

The 2650/2650A uses explicit operand addressing; that is, each instruction specifies the operand address. The first byte of each 2650/2650A instruction is divided into three fields and specifies the operation to be performed, the addressing mode to be used and, where appropriate, the register or condition code mask to be used.

In the instsruction descriptions which follow, the mnemonic assembler format as well as the execution time are listed. In the mnemonics, parentheses are used to indicate options. The parentheses are not included when the option is desired. With regard to execution time, note that non-branch type instructions specifying indirect addressing require an additional 2 cycles (6 clock periods) for execution. Branch type instructions specifying indirect addressing require an additional 2 cycles for execution only if the branch is taken.

Symbols and Abbreviations Used

a	Address Value
C	Carry bit
CC	Condition Code bits
COM	Compare bit
EA	Effective Address
F	Flag bit
I	Indirect addressing bit
IAR	Instruction Address Register
IC	Index Control bits
IDC	Inter-Digit Carry bit
II	Interrupt Inhibit bit
OVF	Overflow bit
PSL	Program Status Lower byte
PSU	Program Status Upper byte
PSW	Program Status Word
r	Register ($0 \leq r \leq 3$)
RS	Register Bank Select bit
RO	Register zero
S	Sense bit
SP	Stack Pointer
v	A value
WC	With Carry bit
x	Index register value ($0 \leq x \leq 3$)
X	Index register value with optional auto-increment (,+) or auto-decrement (,-)
(A)	Contents of A

((A))	Contents of location addressed by A
(a:b)	Bits a through b
:	Is compared to
←	Is replaced by

Calculating Effective Addresses

The "Effective Address' (EA) of an instruction depends on the addressing mode used and whether the indirect and/or indexing options are invoked. The following rules apply for calculating the EA. Exceptions to these rules are detailed in the instruction descriptions.

$EA = a$	; if $I = 0$, $IC = 00$
$EA = (a)$	; if $I = 1$, $IC = 00$
$EA = a + (x)$	; if $I = 0$, $IC = 11$
$EA = a + (x) + 1$	; if $I = 0$, $IC = 01$
$EA = a + (x) - 1$	; if $I = 0$, $IC = 10$
$EA = (a) + (x)$	; if $I = 1$, $IC = 11$
$EA = (a) + (x) + 1$	; if $I = 1$, $IC = 01$
$EA = (a) + (x) -1$	; if $I = 1$, $IC = 10$

Absolute Addressing - Branch Instructions

$EA = a$	; if $I = 0$
$EA = (a)$	; if $I = 1$

Relative Addressing

Relative instructions (except ZBRR and ZBSR) are calculated relative to the current value of the Instruction Address Register (IAR). Note that when the calculation of EA is made, the IAR value is the first address following the instruction, or equivalently, the address of the first byte of the relative addressing instruction plus two.

$EA = (IAR) + a$	; if $I = 0$
$EA = ((IAR) + a)$	; if $I = 1$

Note that 'a' is treated as a two's complement value, so that forward or backward EA's can be generated.

ADDA,r (*)a(,X) ADD ABSOLUTE

Addressing: Absolute

Operation Codes: 8C - 8F

Binary Coding:

1	0	0	0	1	1	r or X	
7	6	5	4	3	2	1	0

I	IC		a high order				
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time: 4 cycles (12 clock periods)

Operation: $(r) \leftarrow (r) + (EA)$; if $(WC) = 0$
$(r) \leftarrow (r) + (EA) + (C)$; if $(WC) = 1$

DESCRIPTION: This 3-byte instruction causes the contents of register r and the contents of the byte of memory pointed to by the effective address to be added together in a true binary adder. The 8-bit sum replaces the contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

NOTE
Add with Carry may be performed. See With/Without Carry and Carry in description of Program Status Word.

PSW Bits Affected: C, CC, IDC, OVF

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ADDI,r v ADD IMMEDIATE

Addressing: Immediate

Operation Codes: 84 - 87

Binary Coding:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 2 cycles (6 clock periods)

Operation: $(r) \leftarrow (r) + v$; if (WC) = 0
$(r) \leftarrow (r) + v + (c)$; if (WC) = 1

DESCRIPTION: This 2-byte instruction causes the contents of register r and the contents of the second byte of this instruction to be added together in a true binary adder. The 8-bit sum replaces the contents of register r.

NOTE
Add with carry may be performed. See With/Without Carry and Carry in description of Program Status Word.

PSW Bits Affected: C, CC, IDC, OVF

Condition Code Setting:

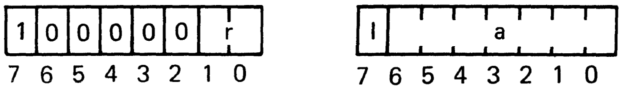
REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ADDR,r (*)a — ADD RELATIVE

Addressing: Relative

Operation Codes: 88 - 8B

Binary Coding:



Execution Time: 3 cycles (9 clock periods)

```
Operation:  (r)←(r) + (EA)          ; if (WC) = 0
            (r)←(r) + (EA) + (C)    ; if (WC) = 1
```

DESCRIPTION: This 2-byte instruction causes the contents of register r and the contents of the byte of memory pointed to by the effective address to be added together in a true binary adder. The 8-bit sum replaces the contents of register r.

Indirect addressing may be specified.

NOTE
Add with Carry may be performed. See With/Without Carry and Carry in description of Program Status Word.

PSW Bits Affected: C, CC, IDC, OVF

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ADDZ r — ADD TO REGISTER ZERO

Addressing: Register

Operation Codes: 80-83

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: $(R0) \leftarrow (R0) + (r)$; if $(WC) = 0$
$(R0) \leftarrow (R0) + (r) + (c)$; if $(WC) = 1$

Description: This 1-byte instruction causes the contents of register zero to be added together in a true binary adder. The 8-bit sum of the addition replaces the contents of register zero. The contents of register r remain unchanged.

NOTE
Add with Carry may be performed. See With/Without Carry in description of Program Status Word.

PSW Bits Affected: C, CC, IDC, OVF

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ANDA,r (*)a(,X) AND ABSOLUTE

Addressing: Absolute
Operation Codes: 4C - 4F

Binary Code:

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

7	6	5	4	3	2	1	0
I	IC		a high order				

7	6	5	4	3	2	1	0
a low order							

Execution Time: 4 cycles (12 clock periods)

Operation: (r) ← (r) AND (EA)

Description: This 3-byte instruction causes the contents of register r to be logically ANDed with the contents of the memory byte pointed to by the effective address. The result of the operation replaces the contents of register r.

The AND operation treats each bit of the argument bytes as in the truth table below:

BIT (0-7)	BIT (0-7)	AND RESULT
0	0	0
0	1	0
1	1	1
1	0	0

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER N	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ANDI,r v — AND IMMEDIATE

Addressing: Immediate

Operation Codes: 44 - 47

Binary Code:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 2 cycles (6 clock periods)

Operation: (r)←(r) AND v

Description: This two-byte instruction causes the contents of the specified register r to be logically ANDed with the contents of the second byte of this instruction. The result of this operation replaces the contents of register r.

The AND operation treats each bit of the argument bytes as in the truth table below:

BIT (0-7)	BIT (0-7)	AND RESULT
0	0	0
0	1	0
1	1	1
1	0	0

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ANDR,r (*)a AND RELATIVE

Addressing: Relative

Operation Codes: 48 - 4B

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (r) ← (r) AND (EA)

Description: This two-byte instruction causes the contents of the specified register r to be logically ANDed with the contents of the memory byte pointed to by the effective address. The result of this operation replaces the contents of register r.

The AND operation treats each bit of the argument bytes as in the truth table below:

BIT (0-7)	BIT (0-7)	AND Result
0	0	0
0	1	0
1	1	1
1	0	0

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ANDZ r AND TO REGISTER ZERO

Addressing: Register

Operation Codes: 41 -43

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (RO) ← (RO) AND (r) ; r = RO

Description: This 1-byte instruction causes the contents of the specified register, r, to be logically ANDed with the contents of register zero. The result of the operation replaces the contents of register zero. The contents of register r remain unchanged.

The AND operation treats each bit of the argument bytes as in the truth table below:

BIT (0-7)	BIT (0-7)	AND RESULT
0	0	0
0	1	0
1	1	1
1	0	0

NOTE
Register r may not be specified as zero. The operation code '01000000' is reserved for HALT.

PSW Bits Affected: CC

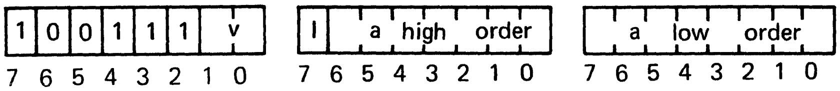
Condition Code Setting:

REGISTER ZERO	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

Addressing: Absolute

Operation Codes: 9C - 9E

Binary Code:



Execution Time: 3 cycles (9 clock periods)

Operation: (IAR)←EA ; if v=(CC), v=3

Description: This 2-byte instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the 2-bit v field does not match the 2-bit Condition Code field (CC) in the Program Status Word. If there is no match, the contents of the Instruction Address Register are replaced by the effective address.

If the v field and CC field match, the next instruction is fetched from the location following the third byte of this instruction.

Indirect addressing may be specified.

The v field may not be set to 3 as this bit combination is used for the BXA operation code.

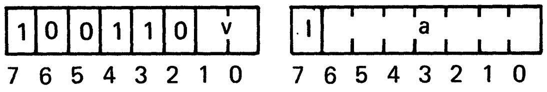
PSW Bits Affected: None

Condition Code Setting: N/A

Addressing: Relative

Operation Codes: 98 - 9A

Binary Code:



Execution Time: 3 cycles (9 clock periods)

Operation: (IAR)←EA ; if v = (CC), v = 3

Description: This 2-byte branch instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the 2-bit v field does not match the 2-bit Condition Code field (CC) in the Program Status Word. If there is no match, the contents of the Instruction Address Register are replaced by the effective address

If the v field and CC field match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

The v field may not be set to 3 as this bit combination is used for the ZBRR operation code.

PSW Bits Affected: None

Condition Code Setting: N/A

BCTA,v (*)a — BRANCH ON CONDITION TRUE, ABSOLUTE

Addressing: Absolute

Operation Codes: 1C - 1F

Binary Code:

0	0	0	1	1	1	v		I	a high order							a low order							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

```
Operation:  (IAR)←EA      ;  if v = (CC), v = 3
            (IAR)←EA      ;  if v = 3
```

Description: This 3-byte conditional branch instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the 2-bit v field matches the 2-bit Condition Code field (CC) in the Program Status Word. If the v field is set to 3, an unconditional branch is effected.

If the v field and CC field do not match, the next instruction is fetched from the location following the third byte of this instruction. Indirect addressing may be specified.

PSW Bits Affected: None

Condition Code Setting: N/A

BCTR,v (*)a — BRANCH ON CONDITION TRUE, RELATIVE

Addressing: Relative

Operation Codes: 18 - 1B

Binary Code:

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

7	6	5	4	3	2	1	0
I	a						

Execution Time: 3 cycles (9 clock periods)

Operation: (IAR) ← EA ; if v = (CC), v = 3
(IAR) ← EA ; if v = 3

Description: This 2-byte conditional branch instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the 2-bit v field matches the current Condition Code field (CC) in the Program Status Word. If the v field is set to 3, an unconditional branch is effected.

If the v field and CC field do not match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

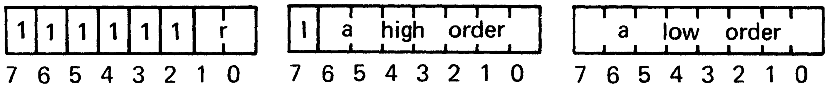
PSW Bits Affected: None

Condition Code Setting: N/A

Addressing: Absolute

Operation Codes: FC - FF

Binary Code:



Execution Time: 3 cycles (9 clock periods)

Operation: (r) ← (r) - 1, then
(IAR) ← EA ; if (r) = 0

Description: This 3-byte instruction causes the processor to decrement the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address; i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value in register r is zero, the next instruction to be executed follows the third byte of this instruction.

Indirect addressing may be specified.

PSW Bits Affected: None

Condition Code Setting: N/A

BDRR,r (*)a — BRANCH ON DECREMENTING REGISTER, RELATIVE

Addressing: Relative

Operation Codes: F8 - FB

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (r) ← (r) - 1, then
(IAR) ← EA ; if (r) = 0

Description: This 2-byte branch instruction causes the processor to decrement the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address; i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value in register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

PSW Bits Affected: None

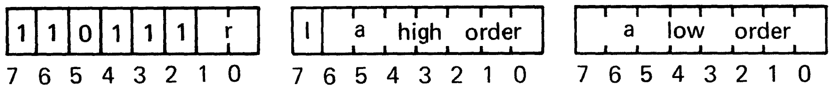
Condition Code Setting: N/A

BIRA,r (*)a — BRANCH ON INCREMENTING REGISTER, ABSOLUTE

Addressing: Absolute

Operation Codes: DC - DF

Binary Code:



Execution Time: 3 cycles (9 clock periods)

```
Operation:  (r)←(r) + 1, then
            (IAR)←EA          ;  if (r) = 0
```

Description: This 3-byte branch instruction causes the processor to increment the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address, i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value of register r is zero, the next instruction to be executed follows the third byte of this instruction.

Indirect addressing may be specified.

PSW Bits Affected: None

Condition Code Setting: N/A

BIRR,r (*)a — BRANCH ON INCREMENTING REGISTER, RELATIVE

Addressing: Relative

Operation Codes: D8 - DB

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

```
Operation:  (r)←(r) +1, then
            (IAR)←EA              ; if (r) = 0
```

Description: This 2-byte branch instruction causes the processor to increment the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address; i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value in register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

PSW Bits Affected: None

Condition Code Setting: N/A

BRNA,r (*)a — BRANCH ON REGISTER NON-ZERO, ABSOLUTE

Addressing: Absolute

Operation Codes: 5C - 5F

Binary Coding:

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

7	6	5	4	3	2	1	0
I	a high order						

7	6	5	4	3	2	1	0
a low order							

Execution Time: 3 cycles (9 clock periods)

Operation: (IAR)←EA ; if (r) = 0

DESCRIPTION: This 3-byte branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address; i.e., the effective address replaces the contents of the Instruction Address Register.

If the specified register contains a zero value, the next instruction is fetched from the location following the third byte of this instruction.

Indirect addressing may be specified.

PSW Bits Affected: None

Condition Code Setting: N/A

BRNR,r (*)a — BRANCH ON REGISTER NON-ZERO, RELATIVE

Addressing: Relative

Operation Codes: 58 - 5B

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (IAR)←EA ; if (r) = 0

Description: This 2-byte branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address; i.e., the effective address replaces the current contents of the Instruction Address Register.

If the specified register contains a zero value, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

PSW Bits Affected: None

Condition Code Setting: N/A

BSFA,v (*)a — BRANCH TO SUBROUTINE ON CONDITIOn FALSE, ABSOLUTE

Addressing: Absolute

Operation Codes: BC - BE

Binary Code:

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

7	6	5	4	3	2	1	0
I	a high order						

7	6	5	4	3	2	1	0
a low order							

Execution Time: 3 cycles (9 clock periods)

Operation: (SP) ← (SP) + 1
((SP)) ← (IAR)
(IAR) ← EA
} if v = (CC),v = 3

Description: This 3-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch only if the 2-bit v field does not match the current Condition Code (CC) in the Program Status Word. If the fields do not match, the Stack Pointer is incremented by one and the current content of the Instruction Address Register, which points to the location following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and the CC match, the next instruction is fetched from the location following this instruction and the SP is unaffected. The v field may not be coded as 3 since this combination is used for the BSXA operation code.

Indirect addressing may be specified.

PSW Bits Affected: SP

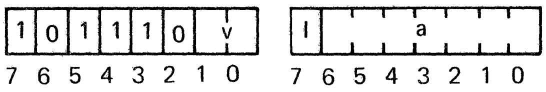
Condition Code Setting: N/A

BSFR,v (*)a BRANCH TO SUBROUTINE ON CONDITION FALSE, RELATIVE

Addressing: Relative

Operation Codes: B8 - BA

Binary Code:



Execution Time: 3 cycles (9 clock periods)

Operation: $\left.\begin{array}{l}(SP) \leftarrow (SP) + 1 \\ ((SP)) \leftarrow (IAR) \\ (IAR) \leftarrow EA\end{array}\right\}$ if $v = (CC)$, $v = 3$

Description: This 2-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch only if the 2-bit v field does not match the current Condition Code field (CC) in the Program Status Word. If the fields do not match, the Stack Pointer is incremented by one and the current content of the Instruction Address Register, which points to the location following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and the CC match, the next instruction is fetched from the location following this instruction and the SP is unaffected. The v field may not be coded as 3 because this combination is used for the ZBSR operation code.

Indirect addressing may be specified.

PSW Bits Affected: SP

Condition Code Setting: N/A

BSNA,r (*)a BRANCH TO SUBROUTINE ON NON-ZERO REGISTER, ABSOLUTE

Addressing: Absolute

Operation Codes: 7C - 7F

Binary Code:

0	1	1	1	1	1	r		I	a high order							a low order							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation:
$$\left.\begin{array}{l}(SP)\leftarrow(SP)+1\\((SP))\leftarrow(IAR)\\(IAR)\leftarrow EA\end{array}\right\}\qquad \text{if } (r) = 0$$

Description: This 3-byte subroutine branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address. Before replacing the current contents of the Instruction Address Register (IAR) with the effective address, the Stack Pointer (SP) is incremented by one and the address of the byte following the instruction is pushed into the Return Address Stack (RAS).

If the specified register contains a zero value, the next instruction is fetched from the location following the third byte of this instruction and the SP is unaffected.

Indirect addressing may be specified.

PSW Bits Affected: SP

Condition Code Setting: N/A

BSNR,r (*)a BRANCH TO SUBROUTINE ON NON-ZERO REGISTER, RELATIVE

Addressing: Relative

Operation Codes: 78 - 7B

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation:

$$\left.\begin{array}{l} (SP) \leftarrow (SP) + 1 \\ ((SP)) \leftarrow (IAR) \\ (IAR) \leftarrow EA \end{array}\right\} \quad \text{if } (r) = 0$$

Description: This 2-byte subroutine branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address. Before replacing the contents of the Instruction Register with the effective address, the Stack Pointer (SP) is incremented by one and the address of the byte following the instruction is pushed into the Return Address Stack (RAS).

If the specified register contains a zero value, the next instruction is fetched from the location following the second byte of this instruction and the SP is unaffected.

Indirect addressing may be specified.

PSW Bits Affected: SP

Condition Code Setting: N/A

BSTA,v (*)a BRANCH TO SUBROUTINE ON CONDITION TRUE, ABSOLUTE

Addressing: Absolute

Operation Codes: 3C - 3F

Binary Code:

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

I	a high order						
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation:

$$\left.\begin{array}{l}(SP) \leftarrow (SP) + 1\\ ((SP)) \leftarrow (IAR)\\ (IAR) \leftarrow EA\end{array}\right\} \quad \begin{array}{l}\text{if } v = (CC) \text{ or}\\ \text{if } v = 3\end{array}$$

Description: This 3-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch only if the 2-bit v field matches the current Condition Code Field (CC) in the Program Status Word. If the fields match, the Stack Pointer is incremented by one and the current contents of the Instruction Address Register, which points to the byte following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and the CC field do not match, the next instruction is fetched from the location following the third byte of this instruction and the Stack Pointer is unaffected. If v is set to 3, the BSTA instruction branches unconditionally.

Indirect addressing may be specified.

PSW Bits Affected: SP

Condition Code Setting: N/A

BSTR,v (*)a BRANCH TO SUBROUTINE ON CONDITION TRUE, RELATIVE

Addressing: Relative

Operation Codes: 38 - 3B

Binary Code:

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

I	a high order						
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation:

$$\left.\begin{array}{r} (SP) \leftarrow (SP) + 1 \\ ((SP)) \leftarrow (IAR) \\ (IAR) \leftarrow EA \end{array}\right\} \qquad \begin{array}{l} \text{if } v = (CC) \text{ or} \\ \text{if } v = 3 \end{array}$$

Description: This 2-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch only if the 2-bit v field matches the current Condition Code field (CC) in the Program Status Word. If the fields match, the Stack Pointer is incremented by one and the current contents of the Instruction Address Register, which points to the byte following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and CC field do not match, the next instruction is fetched from the location following the second byte of the instruction and the SP is unaffected. If v is set to 3, the BSTR instruction branches unconditionally.

Indirect addressing may be specified.

PSW Bits Affected: SP

Condition Code Setting: N/A

BSXA (*)a,x BRANCH TO SUBROUTINE, INDEXED, ABSOLUTE, UNCONDITIONAL

Addressing: Absolute

Operation Code: BF

Binary Code:

1 0 1 1 1 1 1 1	I a high order	a low order
7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0

Execution Time: 3 cycles (9 clock periods)

Operation: $(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (IAR)$
$(IAR) \leftarrow EA$

Description: This 3-byte instruction causes the processor to perform an unconditional subroutine branch. Indexing is required, and register 3 must be specified as the index register because the entire first byte of this instruction is decoded by the processor. Auto-incrementing or auto-decrementing of the index register cannot be specified.

Execution of this instruction causes the Stack Pointer (SP) to be incremented by one, the address of the byte following this instruction is pushed into the Return Address Stack (RAS), and the effective address replaces the contents of the Instruction Address Register.

If indirect addressing is specified, the value in the index register is added to the indirect address to calculate the effective address.

PSW Bits Affected: SP

Condition Code Setting: N/A

BXA (*)a,x — BRANCH INDEXED, ABSOLUTE, UNCONDITIONAL

Addressing: Absolute

Operation Code: 9F

Binary Code:

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

I	a high order						
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (IAR) ← EA

Description: This 3-byte branch instruction causes the processor to perform an unconditional branch. Indexing is required, and register 3 must be specified as the index register because the entire first byte of this instruction is decoded by the processor.

Auto-incrementing or auto-decrementing of the index register cannot be specified. When executed, the content of the Instruction Address Register (IAR) is replaced by the effective address.

If indirect addressing is specified, the value in the index register is added to the indirect address to calculate the effective address.

PSW Bits Affected: None

Condition Code Setting: N/A

COMA,r (*)a(,X) — COMPARE, ABSOLUTE

Addressing: Absolute

Operation Codes: EC - EF

Binary Code:

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

I	IC		a high order				
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time: 4 cycles (12 clock periods)

Operation: (r) : (EA)

Description: This 3-byte instruction causes the contents of register r to be compared to the contents of the memory byte pointed to by the effective address. The comparison will be performed in either the arithmetic or logical mode depending on the setting of the COM bit in the Program Status Word.

When COM = 1 (logical mode), the values will be treated as 8-bit, positive binary numbers; when COM = 0 (arithmetic mode), the values will be treated as 8-bit, two's complement numbers.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the register used in the operation implicitly becomes register zero.

PSW Bits Affected: CC

Condition Code Setting:

The execution of this instruction causes the Condition Code to be set as shown in the following table:

RESULT	CC1	CC0
Register r greater than memory byte	0	1
Register r equal to memory byte	0	0
Register r less than memory byte	1	0

COMI,r v — COMPARE, IMMEDIATE

Addressing: Register

Operation Codes: E4 - E7

Binary Code:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 2 cycles (6 clock periods)

Operation: (r) : v

Description: This 2-byte instruction causes the contents of the specified register r to be compared to the contents of the second byte of this instruction. The comparison will be performed in either the arithmethic or logical mode depending on the setting of the COM bit in the Program Status Word.

When COM = 1 (logical mode), the values will be treated as 8-bit positive binary numbers; when COM = 0, the values will be treated as 8-bit two's complement numbers.

PSW Bits Affected: CC

Condition Code Setting:

The execution of this instruction causes the Condition Code to be set as shown in the following table:

RESULT	CC1	CC0
Register r greater than v	0	1
Register r equal to v	0	0
Register r less than v	1	0

COMR,r (*)a — COMPARE, RELATIVE

Addressing: Relative

Operation Codes: E8 - EB

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (r) : (EA)

Description: This 2-byte instruction causes the contents of the specified register r to be compared to the contents of the memory byte pointed to by the effective address. The comparison will be performed in either the arithmetic or logical mode depending upon the setting of the COM bit in the Program Status Word.

When COM = 1 (logical mode), the values will be treated as 8-bit positive binary numbers; when COM = 0, the values will be treated as 8-bit, two's complement numbers.

PSW Bits Affected: CC

Condition Code Setting:

The execution of this instruction causes the Condition Code to be set as shown in the following table:

RESULT	CC1	CC0
Register r greater than memory byte	0	1
Register r equal to memory byte	0	0
Register r less than memory byte	1	0

COMZ r — COMPARE TO REGISTER ZERO

Addressing: Register

Operation Codes: E0 - E3

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (RO) : (r)

Description: This 1-byte instruction causes the contents of the specified register r to be compared to the contents of register zero. The comparison will be performed in either arithmetic or logical mode depending on the setting of the COM bit in the Program Status Word.

When COM = 1 (logical mode), the values will be interpreted as 8-bit two's complement numbers.

PSW Bits Affected: CC

Condition Code Setting:

The execution of this instruction causes the Condition Code to be set as shown in the following table:

RESULT	CC1	CC0
Register zero greater than register r	0	1
Register zero equal to register r	0	0
Register zero less than register r	1	0

CPSL v — CLEAR PROGRAM STATUS LOWER, MASKED

Addressing: Immediate

Operation Code: 75

Binary Code:

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

Execution Time: 3 cycles (9 clock periods)

Operation: (PSL) ← (PSL) AND NOT v

Description: This 2-byte instruction causes individual bits in the Lower Program Status Byte to be selectively cleared. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and, if a particular bit in the v field contains a one, the corresponding bit in the status byte is cleared to zero. Any bits in the status byte which are not selected are not modified.

PSW Bits Affected: CC, IDC, RS, WC, OVF, COM, C

Condition Code Setting: The CC bits may be cleared by execution of this instruction.

CPSU v — CLEAR PROGRAM STATUS UPPER, MASKED

Addressing: Immediate

Operation Code: 74

Binary Code:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (PSU) ← (PSU) AND NOT v

Description: This 2-byte instruction causes individual bits in the Upper Program Status Byte to be selectively cleared. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and, if a particular bit in the v field contains a one, the corresponding bit in the status byte is cleared to zero. Any bits in the status byte which are not selected are not modified.

PSW Bits Affected: F, II, SP

Condition Code Setting: N/A

Addressing: Register

Operation Codes: 94 - 97

Binary Code:

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

Execution Time: 3 cycles (9 clock periods)

Operation: See description and table below.

Description: This 1-byte instruction conditionally adds a decimal ten (two's complement negative six in a 4-bit binary number system) to either the high order 4 bits and/or the low order 4 bits of the specified register r.

The truth tables below indicate the logical operation performed. The operation proceeds based on the contents of the Carry (C) and Interdigit Carry (IDC) bits in the Program Status Word. The C and IDC bits remain unchanged by execution of this instruction.

This instruction allows BCD sign magnitude arithmetic to be performed on packed digits by the following procedures:

BCD Addition

1. Add 66_{16} to augend
2. Perform addition of addend and augend
3. Perform DAR instruction

BCD Subtraction

1. Perform subtraction (2's complement of subtrahend is added to the minuend)
2. Perform DAR instruction

Since this operation is on sign-magnitude numbers, it is necessary to establish the sign of the result prior to executing in order to properly control the definition of the subtrahend and minuend.

CARRY	INTERDIGIT CARRY	ADDED TO REGISTER r
0	0	AA_{16}
0	1	$A0_{16}$
1	1	00_{16}
1	0	$0A_{16}$

PSW Bits Affected: CC

Condition Code Setting: The condition code is set to a value reflecting the contents of the register as if it were a two's complement binary number.

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

EORA,r (*)a(,X) EXCLUSIVE-OR ABSOLUTE

Addressing: Absolute

Operation Codes: 2C - 2F

Binary Code:

0	0	1	0	1	1	r		I	IC		a high order					a low order							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time: 4 cycles (12 clock periods)

Operation: (r) ← (r) XOR (EA)

Description: This 3-byte instruction causes the contents of register r to be Exclusive-ORed with the contents of the memory byte pointed to by the effective address. The result of the operation replaces the previous contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register, and the destination of the operation implicitly becomes register zero.

The Exclusive-OR operation treats each bit of the argument bytes as shown in the truth table below:

BIT (0-7)	BIT (0-7)	EXCLUSIVE-OR RESULT
0	0	0
0	1	1
1	1	0
1	0	1

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

EORI,r v — EXCLUSIVE-OR IMMEDIATE

Addressing: Immediate

Operation Codes: 24 - 27

Binary Code:

0	0	1	0	0	1	r		v							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time: 2 cycles (6 clock periods)

Operation: (r) ← (r) XOR v

Description: This two-byte instruction causes the contents of the specified register r to be logically Exclusive-ORed with the contents of the second byte of this instruction. The result of this operation replaces the previous contents of register r.

The Exclusive OR operation treats each bit of the argument bytes as shown in the truth table below:

BIT (0-7)	BIT (0-7)	EXCLUSIVE OR RESULT
0	0	0
0	1	1
1	1	0
1	0	1

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

EORR,r (*)a EXCLUSIVE-OR RELATIVE

Addressing: Relative

Operation Codes: 28 - 2B

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (r) ← (r) XOR (EA)

Description: This 2-byte instruction causes the contents of the specified register r to be logically Exclusive-ORed with the contents of the memory byte pointed to by the effective address. The result of the operation replaces the previous contents of register r.

Indirect addressing may be specified.

The Exclusive-OR operation treats each bit of the argument bytes as shown in the truth table below:

BIT (0-7)	BIT (0-7)	AND Result
0	0	0
0	1	1
1	1	0
1	0	1

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

EORZ r — EXCLUSIVE-OR TO REGISTER ZERO

Addressing: Register

Operation Codes: 20 - 23

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (RO) ← (RO) XOR (r)

Description: This 1-byte instruction causes the contents of the specified register r to be logically Exclusive-ORed with the contents of register zero. The result of this operation replaces the contents of register zero. The contents of register r remain unchanged.

The Exclusive-OR operation treats each bit of the argument bytes as shown in the truth table below:

BIT (0-7)	BIT (0-7)	EXCLUSIVE OR RESULT
0	0	0
0	1	1
1	1	0
1	0	1

PSW Bits Affected: CC

Condition Code Setting:

REGISTER ZERO	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

HALT

HALT, ENTER WAIT STATE

Operation Code: 40

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: HALT

Description: This 1-byte instruction causes the processor to stop executing instructions and enter the Wait state. The RUN/WAIT line is set to the Wait state.

The only way to enter the Run state after a HALT has been executed is to reset the 2650 or to interrupt the processor.

PSW Bits Affected: None

Condition Code Setting: N/A

IORA,r (*)a(,X) INCLUSIVE-OR ABSOLUTE

Addressing: Absolute

Operation Codes: 6C - 6F

Binary Code:

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

7	6	5	4	3	2	1	0
I	IC		a high order				

7	6	5	4	3	2	1	0
a low order							

Execution Time: 4 cycles (12 clock periods)

Operation: (r) ← (r) OR (EA)

Description: This 3-byte instruction uses the contents of register r to be logically Inclusive-ORed with the contents of the memory byte pointed to by the effective address. The result of the operation replaces the previous contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

The Inclusive-OR operation treats each bit of the argument bytes as in the truth table below.

BIT (0-7)	BIT (0-7)	INCLUSIVE OR RESULT
0	0	0
0	1	1
1	1	1
1	0	1

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

IORI,r v — INCLUSIVE-OR IMMEDIATE

Addressing: Immediate

Operation Codes: 24 - 27

Binary Code:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 2 cycles (6 clock periods)

Operation: (r) ← (r) or v

Description: This 2-byte instruction causes the contents of the specified register r to be logically Inclusive-ORed with the contents of the second byte of this instruction. The result of this operation replaces the contents of register r.

The Inclusive-OR operation treats each bit of the argument bytes as in the truth table below:

BIT (0-7)	BIT (0-7)	INCLUSIVE OR RESULT
0	0	0
0	1	1
1	1	1
1	0	1

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

IORR,r (*)a — INCLUSIVE-OR RELATIVE

Addressing: Relative

Operation Codes: 68 - 6B

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (r) ← (r) OR (EA)

Description: This 2-byte instruction causes the contents of the specified register r to be logically Inclusive-ORed with the contents of the memory byte pointed to by the effective address. The result of this operation replaces the previous contents of register r.

Indirect addressing may be specified.

The Inclusive-OR operation treats each bit of the argument bytes as in the truth table below:

BIT (0-7)	BIT (0-7)	INCLUSIVE OR RESULT
0	0	0
0	1	1
1	1	1
1	0	1

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

IORZ r — INCLUSIVE-OR TO REGISTER ZERO

Addressing: Register

Operation Codes: 60 - 63

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (RO) ← (RO) or (r)

Description: This 1-byte instruction causes the contents of the specified register, r, to be logically Inclusive-ORed with the contents of register zero. The result of this operation replaces the contents of register zero. The contents of register r remain unchanged.

The Inclusive-OR operation treats each bit of the argument bytes as in the truth table below:

BIT (0-7)	BIT (0-7)	INCLUSIVE OR RESULT
0	0	0
0	1	1
1	1	1
1	0	1

PSW Bits Affected: CC

Condition Code Setting:

REGISTER ZERO	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

Addressing: Absolute

Operation Codes: OC - OF

Binary Code:

0	0	0	0	1	1	r or X	
7	6	5	4	3	2	1	0

I	IC		a high order				
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time: 4 cycles (12 clock periods)

Operation: (r) ← (EA)

Description: This 3-byte instruction transfers a byte of data from memory into the specified register, r. The data byte is found at the effective address. The previous contents of register r are lost.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicilty becomes register zero.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

LODI,r v — LOAD IMMEDIATE

Addressing: Immediate

Operation Codes: 04 - 07

Binary Code:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 2 cycles (6 clock periods)

Operation: (r) ← v

Description: This 2-byte instruction transfers the second byte of the instruction, v, into the specified register, r. The previous contents of r are lost.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

LODR,r (*)a — LOAD RELATIVE

Addressing: Relative

Operation Codes: 08 - 0B

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (r) ← (EA)

Description: This 2-byte instruction transfers a byte of data from memory into the specified register, r. The data byte is found at the effective address formed by the addition of the field, considered as a 7-bit two's complement number, and the address of the byte following this instruction. The previous contents of register r are lost. Indirect addressing may be specified.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

LODZ r — LOAD REGISTER ZERO

Addressing: Register

Operation Codes: 00 - 03

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (R0) ← (r)

Description: This 1-byte instruction transfers the contents of the specified register, r, into register zero. The previous contents of register zero are lost. The contents of register r remain unchanged.

When the specified register, r, equals 0, the operation code is changed to 60_{16} (IORZ) by the assembler. However, the processor will execute the instruction 00_{16} correctly.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER ZERO	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

LPSL LOAD PROGRAM STATUS, LOWER

Operation Code: 93

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (PSL) ← (RO)

Description: This 1-byte instruction causes the current contents of the Lower Program Status Byte to be replaced with the contents of register zero.

See Program Status Word description for bit assignments.

PSW Bits Affected: CC, IDC, RS, WC, OVF, COM, C

Condition Code Setting: The CC will take on the values in bits 7 and 6 of register zero.

LPSU — LOAD PROGRAM STATUS, UPPER

Operation Code: 92

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (PSU) ← (RO)

Description: This 1-byte instruction causes the current contents of the Upper Program Status Byte to be replaced with the contents of register zero.

See Program Status Word description for bit assignments. Bits 4 and 3 of the PSU are unassigned and will always be regarded as containing zeroes.

PSW Bits Affected: F, II, SP

Condition Code Setting: N/A

NOP NO OPERATION

Operation Code: CO

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: None

Description: This 1-byte instruction causes the processor to take no action upon decoding it. No registers are changed, but fetching and executing a NOP instruction requires two processor cycles.

PSW Bits Affected: None

Condition Code Setting: N/A

PPSL v — PRESET PROGRAM STATUS LOWER, MASKED

Addressing: Immediate

Operation Code: 77

Binary Code:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (PSL) ← (PSL) or v

Description: This 1-byte instruction causes individual bits in the Lower Program Status Byte to be selectively set to binary one. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and, if a particular bit in the v field contains a one, the corresponding bit in the status byte is set to one. Any bits in the status byte which are not selected are not modified.

PSW Bits Affected: CC, IDC, RS, WC, OVF, COM, C

Condition Code Setting: The CC bits may be set by the execution of this instruction.

PPSU v — PRESET PROGRAM STATUS UPPER, MASKED

Addressing: Immediate

Operation Code: 76

Binary Code:

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

			v				
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (PSU) ← (PSU) or v

Description: This 2-byte instruction causes individual bits in the Upper Program Status Byte to be selectively set to binary one. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and, if a particular bit in the v field contains a one, the corresponding bit in the ststus byte is set to one. Any bits in the status byte which are not selected are not modified.

PSW Bits Affected: F, II, SP

Condition Code Setting: N/A

REDC,r — READ CONTROL

Addressing: Register

Operation Codes: 30 - 33

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (r) ← (Port C)

Description: This 1-byte input instruction causes a byte of data to be transferred from the data bus into register r. Signals on the data bus are considered to be true signals; i.e., a high level will be set into the register as a one.

When executing this instruction, the processor raises the Operation Request (OPREQ) line, simultaneously switching the M/IO line to IO, the R/W line to R (Read), the D/C line to C (Control), and the E/NE line to NE (Non-Extended).

See Input/Output section of this chapter.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

REDD,r READ DATA

Addressing: Register

Operation Codes: 70 - 73

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (r) ← (Port D)

Description: This 1-byte input instruction causes a byte of data to be transferred from the data bus into register r. Signals on the data bus are considered to be true signals; i.e., a high level will be set into the register as a one.

When executing this instruction, the processor raises the Operation Request (OPREQ) line, simultaneously switching the M/IO line to IO and the R/W to R (Read). Also, during the OPREQ signal, the D/C line switches to D (Data) and the E/NE switches to NE (Non-Extended).

See Input/Output section of this chapter.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

REDE,r v — READ EXTENDED

Addressing: Immediate

Operation Codes: 54 - 57

Binary Code:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (r) ← (Port v)

Description: This 2-byte input instruction causes a byte of data to be transferred from the data bus into register r. During execution of this instruction, the content of the second byte of this instruction is made available on the ADR0 to ADR7 lines of the address bus.

During execution, the processor raises the Operation Request (OPREQ) line, simultaneously placing the contents of the second byte of the instruction on the address bus. During the OPREQ signal, the M/IO line is switched to IO, the R/W line to R (READ), and the E/NE line to E (Extended).

See Input/Output section of this chapter.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

RETC, v RETURN FROM SUBROUTINE, CONDITIONAL

Operation Codes: 14 -17

Binary Code:

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

Execution Time: 3 cycles (9 clock periods)

Operation:

$$\left.\begin{array}{l}(IAR) \leftarrow ((SP)) \\ (SP) \leftarrow (SP) - 1\end{array}\right\} \qquad \begin{array}{l}\text{if } v = (CC) \text{ or} \\ \text{if } v = 3\end{array}$$

Description: This 1-byte instruction is used by a subroutine to conditionally effect a return of control to the program which last issued a subroutine branch instruction.

If the 2-bit v field in the instruction matches the Condition Code field (CC) in the Program Status Word, the following action is taken: The address contained in the top of the Return Address Stack replaces the previous contents of the Instruction Address Register (IAR), and the Stack Pointer is decremented by one.

If the v field does not match CC, the return is not effected, and the next instruction to be executed is taken from the location following this instruction.

If v is specified as 3, the return is executed unconditionally.

PSW Bits Affected: SP

Condition Code Setting: N/A

RETE,v RETURN FROM SUBROUTINE AND ENABLE INTERRUPT, CONDITIONAL

Operation Codes: 34 - 37

Binary Code:

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

Execution Time: 3 cycles (9 clock periods)

Operation:

$$\left.\begin{array}{l}(IAR) \leftarrow ((SP)) \\ (SP) \leftarrow (SP) - 1 \\ (II) \leftarrow 0\end{array}\right\} \qquad \begin{array}{l}\text{if } v = (CC) \text{ or} \\ \text{if } v = 3\end{array}$$

Description: This 1-byte instruction is used by a subroutine to conditionally effect a return of control to the program which last issued a Subroutine Branch instruction. Additionally, if the return is effected, the Interrupt Inhibit (II) bit in the Program Status Word is cleared to zero, thus enabling interrupts. This instruction is mainly intended to be used by an interrupt handling routine because receipt of an interrupt causes a Subroutine Branch to be effected and the Interrrupt Inhibit bit to be set to 1. The interrupt handling routine must be able to return and enable simultaneously so that the interrupt routine cannot be interrupted unless specifically desired.

If the 2-bit v field in the instruction matches the Condition Code field (CC) in the Program Status Word, the following action is taken: The address contained in the top of the Return Address Stack (RAS) replaces the previous contents of the Instruction Address Register (IAR), the Stack Pointer is decremented by one and the II bit is cleared to zero.

If the v field does not match CC, the return is not effected and the next instruction to be executed is taken from the location following this instruction.

If v is specified as 3, the return is executed unconditionally.

PSW Bits Affected: SP, II

Condition Code Setting: N/A

Addressing: Register

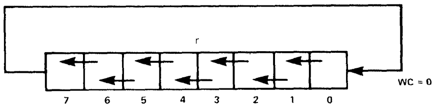
Operation Codes: DO - D3

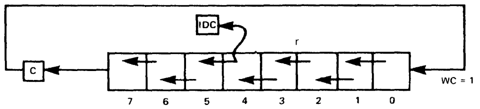
Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

```
Operation:  (r(7:1)) ← (r(6:0))
             (r(0))  ← (r(7))        ; if (WC) = 0
             (r(0))  ← (C)    )
                (C)  ← (r(7)) }      ; if (WC) = 1
              (IDC)  ← (r(4)) )
```





Description: This 1-byte instruction causes the contents of the specified register r to be shifted left one bit. If the WC bit in the Program Status Word is set to zero, bit 7 of register r flows into bit 0; if (WC) = 1, then bit 7 flows into the Carry bit and the Carry bit flows into bit 0.

Register bit 4 flows into the IDC if (WC) = 1.

PSW Bits Affected: C, CC, IDC, OVF

NOTE
If (WC) = 1, and the Rotate causes bit 7 of the specified register to change from 0 to 1, the OVF bit is set in the PSL.

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

RRR,r ROTATE REGISTER RIGHT

Addressing: Register

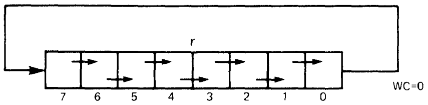
Operation Codes: 50 - 53

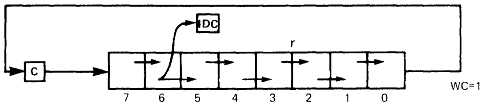
Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

```
Operation:  (r(6:0)) ← (r(7:1))
             (r(7))  ← (r(0))          ; if (WC) = 0
             (r(7))  ← (C)       \
               (C)   ← (r(7))     }    ; if (WC) = 1
             (IDC)   ← (r(6))    /
```





Description: This 1-byte instruction causes the contents of the specified register r to be shifted right 1 bit. If the WC bit in the Program Status Word is set to zero, bit 0 of register r flows into bit 7; if (WC) = 1, then bit 0 of the register r flows into the Carry bit and the Carry bit flows into bit 7.

Register bit 6 flows into the IDC if (WC) = 1.

PSW Bits Affected: C, CC, IDC, OVF

NOTE
If (WC) = 1, and the Rotate causes bit 7 of the specified register to change from 0 to 1, the OVF bit is set in the PSL.

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SPSL STORE PROGRAM STATUS, LOWER

Operation Code: 13

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (RO) ← (PSL)

Description: This 1-byte instruction causes the contents of the Lower Program Status Byte to be transferred into register zero.

See Program Status Word description for bit assignments.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER ZERO	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SPSU — STORE PROGRAM STATUS, UPPER

Operation Code: 12

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (RO) ← (PSU)

Description: This 1-byte instruction causes the contents of the Upper Program Status Byte to be transferred into register zero.

See Program Status Word description for bit assignments. Bits 4 and 3 of the PSU which are unassigned will always be stored as zeroes.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER ZERO	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

STRA,r (*)a(,X) STORE ABSOLUTE

Addressing: Absolute

Operation Codes: CC - CF

Binary Code:

1	1	0	0	1	1	r or X	
7	6	5	4	3	2	1	0

I	IC		a high order				
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time: 4 cycles (12 clock periods)

Operation: (EA) ← (r)

Description: This 3-byte instruction transfers a byte of data from the specified register, r, into the byte of memory pointed to by the effective address. The contents of register r remain unchanged and the contents of the memory byte are replaced.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the source of the operation implicitly becomes register zero.

PSW Bits Affected: None

Condition Code Setting: N/A

STRR,r (*)a STORE RELATIVE

Addressing: Relative

Operation Codes: C8 - CB

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (EA) ← (r)

Description: This 2-byte instruction transfers a byte of data from the specified register, r, into the byte of memory pointed to by the effective address. The contents of register r remain unchanged and the contents of the memory byte are replaced.

Indirect addressing may be specified.

PSW Bits Affected: None

Condition Code Setting: N/A

STRZ r STORE REGISTER ZERO

Addressing: Register

Operation Codes: C1 - C3

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (r) ← (RO) ; r ≠ RO

Description: This 1-byte instruction transfers the contents of register zero into the specified register r. The previous contents of register r are lost. The contents of register zero remain unchanged.

NOTE
Register r may not be specified as zero. This operation code, '11000000', is reserved for NOP.

PSW Bits Affected: CC

Condition Code Setting:

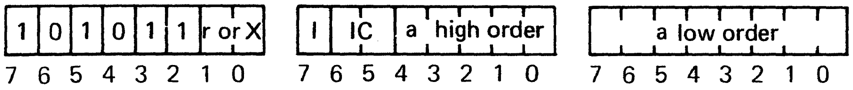
REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SUBA,r (*)a(,X) SUBTRACT ABSOLUTE

Addressing: Absolute

Operation Codes: AC - AF

Binary Code:



Execution Time: 4 cycles (12 clock periods)

Operation: (r) ← (r) - (EA) ; if (WC) = 0
(r) ← (r) - (EA) - (C) ; if (WC) = 1

Description: This 3-byte instruction causes the contents of the byte of memory pointed to by the effective address to be subtracted from the contents of register r. The result of the subtraction replaces the contents of register r.

The subtraction is performed by taking the binary two's complement of the contents of the memory byte and adding that result to the contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register, and the destination of the operation implicitly becomes register zero.

NOTE
Subtract with Borrow may be performed. See With/Without Carry and Carry in description of Program Status Word.

PSW Bits Affected: C, CC, IDC, OVF

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SUBI,r v — SUBTRACT IMMEDIATE

Addressing: Immediate

Operation Codes: A4 - A7

Binary Code:

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

7	6	5	4	3	2	1	0
v							

Execution Time: 2 cycles (6 clock periods)

Operation: $(r) \leftarrow (r) - v$; if (WC) = 0
$(r) \leftarrow (r) - v - (C)$; if (WC) = 1

Description: This 2-byte instruction causes the contents of the second byte of this instruction to be subtracted from the contents of register r. The result of the subtraction replaces the contents of register r.

The subtraction is performed by taking the binary two's complement of the contents of the second instruction byte and adding that result to the contents of register r.

NOTE
Subtraction with Borrow may be performed. See With/Without Carry and Carry in description of Program Status Word.

PSW Bits Affected: C, CC, IDC, OVF

Condition Code Setting:

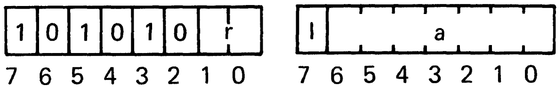
REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SUBR,r (*)a — SUBTRACT RELATIVE

Addressing: Relative

Operation Codes: A8 - AB

Binary Code:



Execution Time: 3 cycles (9 clock periods)

Operation: (r) ← (r) - (EA) ; if (WC) = 0
(r) ← (r) - (EA) - (C) ; if (WC) = 1

Description: This 2-byte instruction causes the contents of the byte of memory pointed to by the effective address to be subtracted from the contents of register r. The result of the subtraction replaces the contents of register r.

The subtraction is performed by taking the binary two's complement of the contents of the byte of memory and adding that result to the contents of register r.

Indirect addressing may be specified.

NOTE
Subtract with Borrow may be performed. See With/Without Carry and Carry in description of Program Status Word.

PSW Bits Affected: C, CC, IDC, OVF

Condition Code Setting:

REGISTER r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SUBZ r — SUBTRACT FROM REGISTER ZERO

Addressing: Register

Operation Codes: A0 - A3

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation:

$$(R0) \leftarrow (R0) - (r) \quad ; \text{ if } (WC) = 0$$
$$(R0) \leftarrow (R0) - (r) - (C) \quad ; \text{ if } (WC) = 1$$

Description: This 1-byte instruction causes the contents of the specified register r to be subtracted from the contents of register zero. The result of the subtraction replaces the contents of register zero.

The subtraction is performed by taking the binary two's complement of the contents of register r and adding that result to the contents of register zero. The contents of register r remain unchanged.

NOTE
Subtract with Borrow may be performed. See With/Without Carry and Carry in description of Program Status Word.

PSW Bits Affected: C, CC, IDC, OVF

Condition Code Setting:

REGISTER ZERO	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

Addressing: Immediate

Operation Codes: F4 - F7

Binary Code:

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

7	6	5	4	3	2	1	0
v							

Execution Time: 3 cycles (9 clock periods)

Operation: $(CC) \leftarrow 0$; if (r) OR NOT v = FF_{16}
$(CC) \leftarrow 2$; otherwise

Description: This 2-byte instruction tests individual bits in the specified register r to determine if they are set to binary one. During execution, each bit in the v field of the instruction is tested for one, and if a particular bit in the v field contains a one, the corresponding bit in register r is tested for a one or zero. The Condition Code is set to reflect the result of the operation.

If a bit in the v field is zero, the corresponding bit in register r is not tested.

PSW Bits Affected: CC

Condition Code Setting:

REGISTER r	CC1	CC0
All of the selected bits are 1s	0	0
Not all of the selected bits are 1s	1	0

Addressing: Immediate

Operation Code: B5

Binary Code:

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

Execution Time: 3 cycles (9 clock periods)

Operation: (CC) ← 0 ; if (PSL) OR NOT v = FF_{16}
(CC) ← 2 ; otherwise

Description: This 2-byte instruction tests individual bits in the Lower Program Status Byte to determine if they are set to binary one. When this instruction is executed, each bit in the v field of this instruction is tested for a one, and if a particular bit in the v field contains a one, the corresponding bit in the status byte is tested for a one or zero. The Condition Code is set to reflect the result of this operation.

If a bit in the v field is zero, the corresponding bit in register r is not tested.

PSW Bits Affected: CC

Condition Code Setting:

	CC1	CC0
All of the selected bits in PSL are 1s	0	0
Not all of the selected bits in PSL are 1s	1	0

TPSU v — TEST PROGRAM STATUS UPPER, MASKED

Addressing: Immediate

Operation Code: B4

Binary Code:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: $(CC) \leftarrow 0$; if (PSU) OR NOT v = FF_{16}
$(CC) \leftarrow 2$; otherwise

Description: This 2-byte instruction tests individual bits in the Upper Program Status byte to determine if they are set to binary one. When this instruction is executed, each bit in the v field of this instruction is tested for the presence of a one, and if a particular bit in the v field contains a one, the corresponding bit in the status byte is tested for a one or zero. The Condition Code is set to reflect the result of this operation.

If a bit in the v field is zero, the corresponding bit in the status byte is not tested.

PSW Bits Affected: CC

Condition Code Setting:

	CC1	CC0
All of the selected bits in PSU are 1s	0	0
Not all of the selected bits in PSU are 1s	1	0

WRTC,r WRITE CONTROL

Addressing: Register

Operation Codes: BO - B3

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (Port C) ← (r)

Description: This 1-byte output instruction causes a byte of data to be made available to an external device. The byte to be output is taken from register r and made available on the data bus.

When executing this instruction, the processor raises the Operation Request (OPREQ) line and simultaneously places the data on the Data Bus. Along with the OPREQ signal, the M/IO line is switched to IO, the R/W signal is switched to W (Write), the D/C line is switched to C (Control), the E/NE is switched to NE (Non-Extended), and a Write Pulse (WRP) is generated.

See the Input/Output section of this chapter.

PSW Bits Affected: None

Condition Code Setting: N/A

WRTD,r WRITE DATA

Addressing: Register

Operation Codes: F0 - F3

Binary Code:

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

Execution Time: 2 cycles (6 clock periods)

Operation: (Port D) ← (r)

Description: This 1-byte output instruction causes a byte of data to be made available to an external device. The byte to be output is taken from register r and made available on the data bus.

When executing this instruction, the processor raises the Operation Request (OPREQ) line and simultaneously places the data on the data bus. Along with the OPREQ, the M/IO line is switched to IO, the R/W signal is switched to W (Write), and a Write Pulse (WRP) is generated. Also, during the OPREQ signal, the D/C line is switched to D (Data) and the E/NE line is switched to NE (Non-Extended).

See Input/Output section of this chapter.

PSW Bits Affected: None

Condition Code Setting: N/A

WRTE,r v WRITE EXTENDED

Addressing: Immediate

Operation Codes: D4 - D7

Binary Code:

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

v							
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (Port v) ← (r)

Description: This 2-byte output instruction causes a byte of data to be made available to an external device. The byte to be output is taken from register r and is made available on the data bus. Simultaneously, the data in the second byte of this instruction is made available on the ADR0 to ADR7 lines of the address bus. The second byte, v, may be interpreted as a device address.

When executing this instruction, the processor raises the Operation Request (OPREQ) line and simultanously places the data from register r on the data bus and the data from the second byte of this instruction on the address bus. Along with OPREQ, the M/IO line is switched to IO, the R/W line is switched to W (Write), the E/NE line is switched to E (Extended), and a Write Pulse (WRP) is generated.

See the Input/Output section of this chapter.

PSW Bits Affected: None

Condition Code Setting: N/A

ZBRR (*)a — ZERO BRANCH RELATIVE

Addressing: Relative

Operation Code: 9B

Binary Code:

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

I	a						
7	6	5	4	3	2	1	0

Execution Time: 3 cycles (9 clock periods)

Operation: (IAR) ← EA

Description: This 2-byte unconditional relative branch instruction directs the processor to calculate the effective address differently than the usual calculation for the Relative Addressing mode.

The specified value, a, is interpreted as a relative displacement from page zero, byte zero. Therefore, displacement may be specified from -64 to +63 bytes. The address calculation is modulo 8192_{10}, so the negative displacement actually will develop addresses at the end of page zero. For example, ZBRR -8, will develop an effective address of 8184_{10}, and a ZBRR +52 will develop an effective address of 52_{10}.

This instruction causes the processor to clear address bits 13 and 14, the page address bits, and to replace the contents of the Instruction Address Register with the effective address of the instruction. This instruction may be executed anywhere within addressable memory.

Indirect addressing may be specified.

PSW Bits Affected: None

Condition Code Setting: N/A

ZBSR (*)a — ZERO BRANCH TO SUBROUTINE, RELATIVE

Addressing: Relative

Operation Code: BB

Binary Code:

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

7	6	5	4	3	2	1	0
I	a						

Execution Time: 3 cycles (9 clock periods)

Operation: $(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (IAR)$
$(IAR) \leftarrow EA$

Description: This 2-byte unconditional subroutine branch instruction directs the processor to calculate the effective address differently than the usual calculation for the relative addressing mode.

The specified value, a, is interpreted as a relative displacement from page zero, byte zero. Therefore, displacement may be specified from -64 to +63 bytes. The address calculation is modulo 8192_{10}, so the negative displacement will develop addresses at the end of page zero. For example, ZBSR -10, will develop an effective address of 8182_{10}, and ZBSR 31 will develop an effective address of 31_{10}.

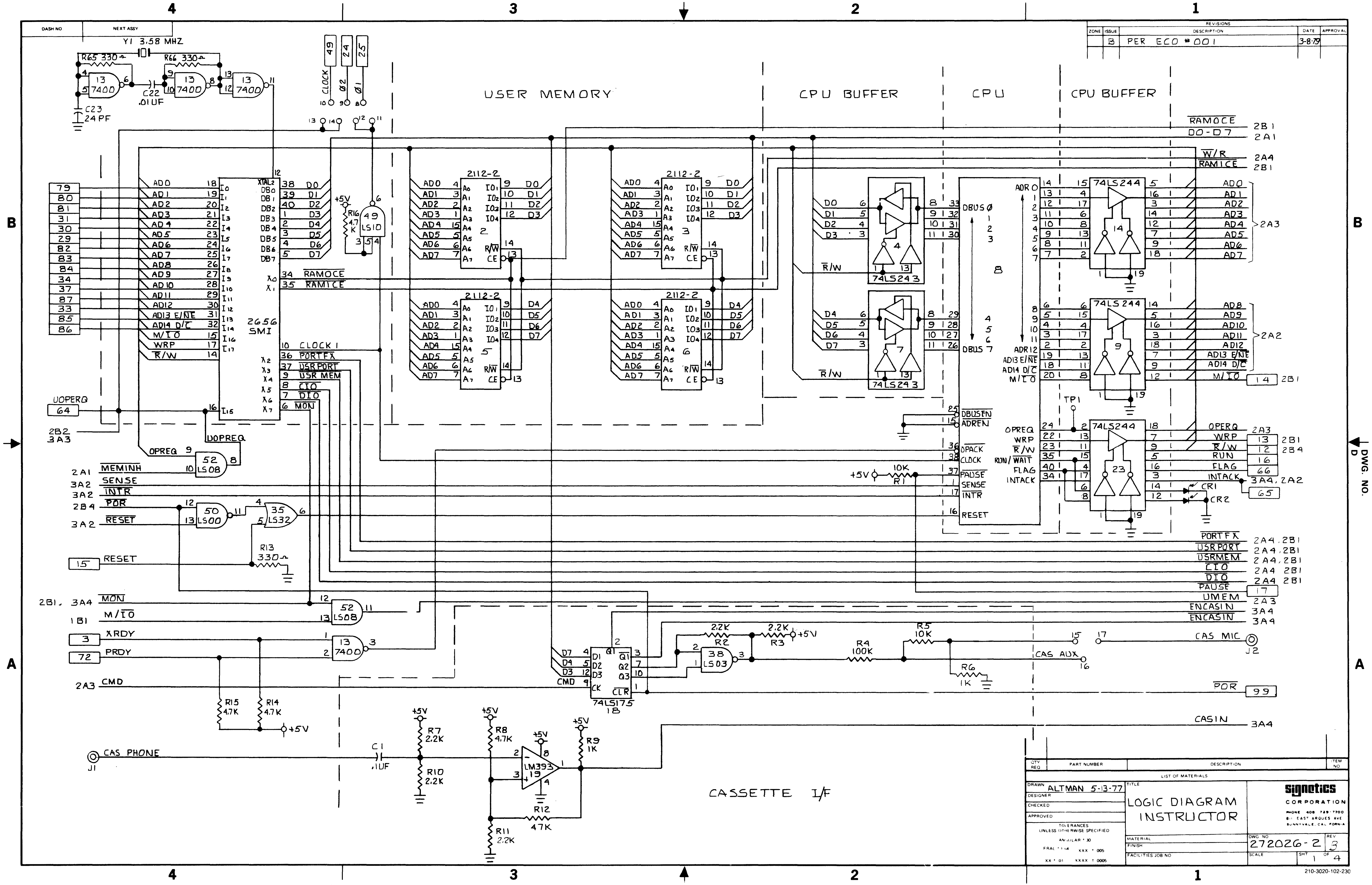
This instruction may be executed anywhere within addressable memory. Indirect addressing may be specified.

When executed, this instruction causes the Stack Pointer to be incremented by one, the address of the byte following this instruction is pushed into the Return Address Stack (RAS), and control is transferred to the effective address.

PSW Bits Affected: SP

Condition Code Setting: N/A

10. INSTRUCTOR 50 SYSTEM SCHEMATICS



ZONE	ISSUE	DESCRIPTION	DATE	APPROVAL
	B	PER ECO #001	3-8-79	

DASH NO | NEXT ASSY

QTY REQ | PART NUMBER | DESCRIPTION | ITEM NO

LIST OF MATERIALS

DRAWN ALTMAN 5-13-77
DESIGNER
CHECKED
APPROVED

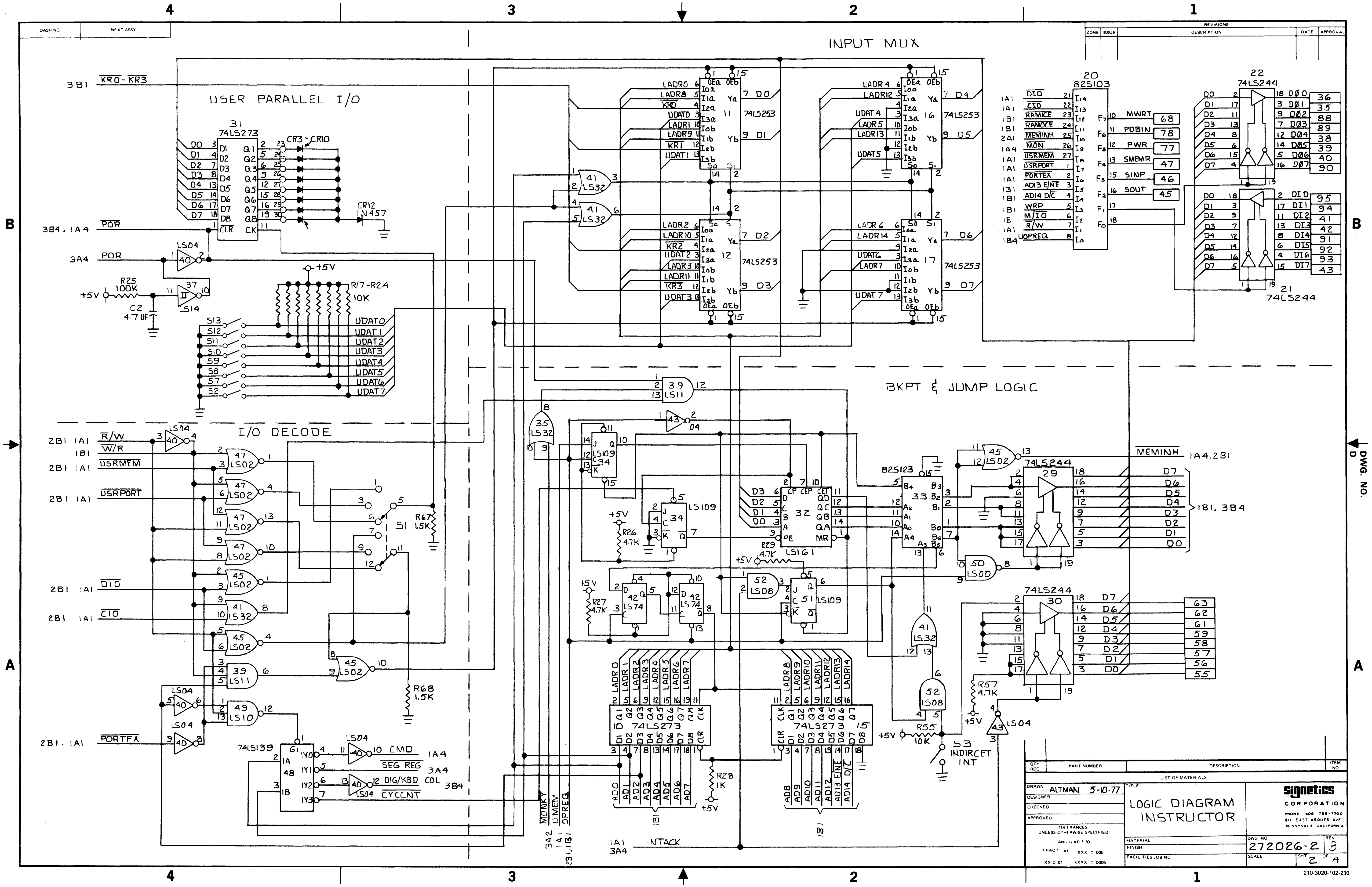
TOLERANCES UNLESS OTHERWISE SPECIFIED
ANGULAR ± 30
FRAC ± 1/64 XXX ± .005
XX ± .01 XXXX ± .0005

TITLE
LOGIC DIAGRAM
INSTRUCTOR

MATERIAL
FINISH
FACILITIES JOB NO

signetics
CORPORATION
PHONE 408 739-7700
811 EAST ARQUES AVE
SUNNYVALE, CALIFORNIA

DWG NO 272026-2 REV 3
SCALE SHT 1 OF 4



QTY REQ	PART NUMBER	DESCRIPTION	ITEM NO

LIST OF MATERIALS

DRAWN ALTMAN 5-10-77

DESIGNER

CHECKED

APPROVED

TOLERANCES UNLESS OTHERWISE SPECIFIED
ANGULAR ± 30
FRAC ± 1/64 XXX ± .005
XX ± .01 XXXX ± .0005

TITLE
LOGIC DIAGRAM
INSTRUCTOR

MATERIAL

FINISH

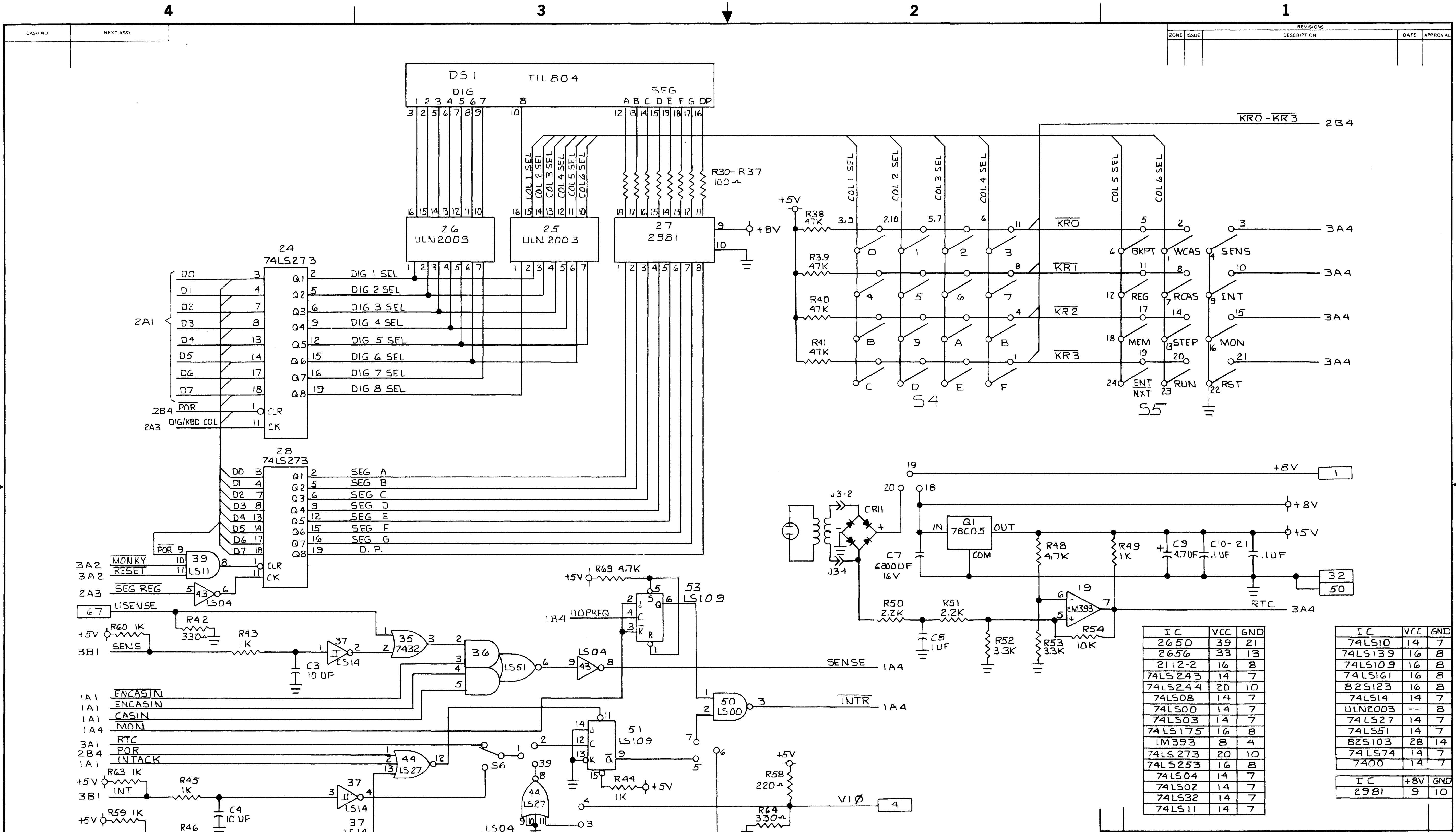
FACILITIES JOB NO

signetics
CORPORATION
PHONE 408 739-7700
811 EAST ARQUES AVE.
SUNNYVALE, CALIFORNIA

DWG NO 272026-2 REV B

SCALE SHT 2 OF 4

210-3020-102-230



I C	VCC	GND
2650	39	21
2656	33	13
2112-2	16	8
74LS243	14	7
74LS244	20	10
74LS08	14	7
74LS00	14	7
74LS03	14	7
74LS175	16	8
LM393	8	4
74LS273	20	10
74LS253	16	8
74LS04	14	7
74LS02	14	7
74LS32	14	7
74LS11	14	7

I C	VCC	GND
74LS10	14	7
74LS139	16	8
74LS109	16	8
74LS161	16	8
82S123	16	8
74LS14	14	7
ULN2003	—	8
74LS27	14	7
74LS51	14	7
82S103	28	14
74LS74	14	7
7400	14	7

I C	+8V	GND
2981	9	10

QTY REQ	PART NUMBER	DESCRIPTION	ITEM NO

LIST OF MATERIALS

DRAWN ALTMAN 6-27-77
DESIGNER
CHECKED
APPROVED

TOLERANCES
UNLESS OTHERWISE SPECIFIED
ANGULAR ± 30
FRAC ± 1/64 XXX ± 005
XX ± 01 XXXX ± 0005

TITLE
LOGIC DIAGRAM
INSTRUCTOR

MATERIAL
FINISH
FACILITIES JOB NO

signetics
CORPORATION
PHONE 408 739-7700
811 EAST ARQUES AVE
SUNNYVALE, CALIFORNIA

DWG NO 272026-2
REV B
SCALE
SHT 3 OF 4

210-3020-102-230

DASH NO	NEXT ASSY

REVISIONS				
ZONE	ISSUE	DESCRIPTION	DATE	APPROVAL

2656 SMI PROGRAM CODING

	X0	X1	X2	X3	X4	X5	X6	X7			
FUNCTION SELECT	E	E	E	E	E	E	E	E			
	0	1	2	3	4	5	6	7	8	9	10
A0	X	X	X	1	1	X	X	X	1	X	X
A1	X	X	X	1	1	X	X	X	1	X	X
A2	X	X	X	1	1	X	X	X	1	X	X
A3	X	X	1	0	1	X	X	X	1	X	X
A4	X	X	1	0	1	X	X	X	1	X	X
A5	X	X	1	0	1	X	X	X	1	X	X
A6	X	X	1	0	1	X	X	X	1	X	X
A7	X	X	1	0	1	X	X	X	1	1	X
A8	0	1	X	X	1	X	X	X	1	1	X
A9	0	0	X	X	1	X	X	X	1	1	X
A10	0	0	X	X	1	X	X	X	1	1	X
A11	0	0	X	X	1	X	X	X	0	0	1
A12	0	0	X	X	0	X	X	1	1	1	1
A13	0	0	1	1	0	0	0	0	0	0	0
A14	0	0	X	X	0	0	1	0	0	0	0
OPREQ	1	1	1	1	1	1	1	X	1	1	1
M IO	1	1	0	0	1	0	0	X	1	1	1
WRP	1	1	1	1	1	1	1	X	1	1	1
	$\overline{\text{RAM OCE}}$	$\overline{\text{RAM ICE}}$	$\overline{\text{PORT FX}}$	$\overline{\text{USR PORT}}$	$\overline{\text{USR MEM}}$	$\overline{\text{CIO}}$	$\overline{\text{DIO}}$	$\overline{\text{MON}}$	SMI PORT	SMI RAM	SMI ROM

CLOCK SOURCE XTAL
CLOCK DIVIDE BY/4
NO DISABLE

82S123 PROGRAM TABLE

ADDRESS								OUTPUT									
HEX	OCTAL	A4	A3	A2	A1	A0		B7	B6	B5	B4	B3	B2	B1	B0		
00	00	0	0	0	0	0		0	1	1	1	0	1	0	0	CØ	
01	01	0	0	0	0	1		0	1	0	1	0	0	1	1	1F	
02	02	0	0	0	1	0		0	1	0	1	0	0	1	0	18	
03	03	0	0	0	1	1		0	1	0	1	0	0	0	0	ØØ	
04	04	0	0	1	0	0		0	0	0	0	0	0	0	0	IDLE	
05	05	0	0	1	0	1		0	0	0	0	0	0	0	0	IDLE	
06	06	0	0	1	1	0		0	0	0	0	0	0	0	0		
07	07	0	0	1	1	1		0	0	0	0	0	0	0	0	IDLE	
08	10	0	1	0	0	0		0	0	0	1	0	0	0	0	SINGLE STEP	
09	11	0	1	0	0	1		0	0	0	1	0	0	0	0		
0A	12	0	1	0	1	0		0	0	0	1	0	0	0	0		
0B	13	0	1	0	1	1		0	0	0	1	0	0	0	0		
0C	14	0	1	1	0	0		0	0	0	1	0	0	0	0		
0D	15	0	1	1	0	1		0	0	0	1	0	0	0	0		
0E	16	0	1	1	1	0		0	0	0	1	0	0	0	0		
0F	17	0	1	1	1	1		0	0	0	1	0	0	0	0		
10	20	1	0	0	0	0		0	0	0	1	0	0	0	0		INTERRUPT SPACE
11	21	1	0	0	0	1		0	0	0	1	0	0	0	0	DIRECT INT VECTOR	DIRECT
12	22	1	0	0	1	0		0	1	1	1	0	0	1	1	1F	
13	23	1	0	0	1	1		0	1	0	1	0	0	1	0	18	
14	24	1	0	1	0	0		0	1	0	1	0	0	0	0	ØØ	
15	25	1	0	1	0	1		0	0	0	0	0	0	0	0		
16	26	1	0	1	1	0		0	0	0	0	0	0	0	0		
17	27	1	0	1	1	1		0	0	0	0	0	0	0	0		
18	30	1	1	0	0	0		0	0	0	0	0	0	0	0		
19	31	1	1	0	0	1		0	0	0	1	0	0	0	0	INDIRECT INT VECTOR	INDIRECT
1A	32	1	1	0	1	0		0	0	0	1	0	0	0	0	HI ADDR BYTE	
1B	33	1	1	0	1	1		0	0	0	1	0	0	0	0	LO ADDR BYTE	
1C	34	1	1	1	0	0		0	1	1	1	0	0	1	1	1F	
1D	35	1	1	1	0	1		0	1	0	1	0	0	1	0	18	
1E	36	1	1	1	1	0		0	1	0	1	0	0	0	0	ØØ	
1F	37	1	1	1	1	1		0	0	0	0	0	0	0	0		
									DATA BUS ENABLE	LAST ADDR LOAD	$\overline{\text{CNTR STOP}}$		D6, D7	D3, D4	DØ, D1, D2		

82S1Ø3 PROGRAM TABLE

OUTPUT POLARITY			INPUT VARIABLE																
			IØ	I1	I2	I3	I4	I5	I6	I7	I8	I9	IA	IB	IC	ID	IE	IF	
FØ		L	H	H	–	–	–	–	H	H	H	H	H	H	H	H	H	–	ENABLE JUTPUT DATA
F1		L	H	L	–	–	–	–	H	H	H	H	H	H	H	H	H	–	ENABLE INPUT DATA
F2		H	–	H	L	–	–	–	H	H	H	H	H	H	H	H	H	–	SOUT
F3		H	–	L	L	–	–	–	H	H	H	H	H	H	H	H	H	–	SINP
F4		H	–	L	H	–	–	–	H	H	H	H	H	H	H	H	H	–	SMEMR
F5		L	H	H	–	H	–	–	H	H	H	H	H	H	H	H	H	–	PWR
F6		H	H	L	–	–	–	–	H	H	H	H	H	H	H	H	H	–	PDBIN
F7		H	H	H	H	–	–	–	H	H	H	H	H	H	H	H	H	–	MWRT
ACTIVE HIGH = H ACTIVE LOW = L			OPREQ	$\overline{\text{R}}$/W	M/$\overline{\text{IO}}$	WRP	AD14-D/$\overline{\text{C}}$	AD13 E/$\overline{\text{NE}}$	$\overline{\text{PORTFX}}$	$\overline{\text{USRPORT}}$	$\overline{\text{USRMEM}}$	$\overline{\text{MON}}$	$\overline{\text{MEMINH}}$	$\overline{\text{RAMØCE}}$	$\overline{\text{RAMICE}}$	$\overline{\text{CIO}}$	$\overline{\text{DIO}}$		

QTY REQ	PART NUMBER	DESCRIPTION	ITEM NO

LIST OF MATERIALS

DRAWN
DESIGNER
CHECKED
APPROVED

TOLERANCES
UNLESS OTHERWISE SPECIFIED
ANGULAR ± 30
FRAC ± 1/64 XXX ± 005
XX ± 01 XXXX ± 0005

TITLE
INSTRUCTOR
PROGRAM TABLES

MATERIAL
FINISH
FACILITIES JOB NO

signetics
CORPORATION
PHONE 408 739-7700
811 EAST ARQUES AVE
SUNNYVALE, CALIFORNIA

DWG NO 272026-2
REV B
SCALE
SHT 4 OF 4

DWG. NO.

210-3020-102-230

11. USE MONITOR PROGRAM LISTINGS

```
TWIN ASSEMBLER VER 2.0    INSTRUCTOR 50 PRGM 11/1/78         PAGE 0001

LINE ADDR  OBJECT  E SOURCE

0002                 *PROGRAM WRITTEN BY DAVE WOTRING
0003                 *********************************************************************
0004                 *   EQUATE TABLES
0005                 *
0006                 *   REGISTER EQUATES
0007 0000            R0     EQU     0      REGISTER 0
0008 0001            R1     EQU     1      REGISTER 1
0009 0002            R2     EQU     2      REGISTER 2
0010 0003            R3     EQU     3      REGISTER 3
0011                 *  CONDITION CODES
0012 0001            P      EQU     1      POSITIVE RESULT
0013 0000            Z      EQU     0      ZERO RESULT
0014 0002            NG     EQU     2      NEGATIVE RESULT
0015 0002            LT     EQU     2      LESS THAN
0016 0000            EQ     EQU     0      EQUAL TO
0017 0001            GT     EQU     1      GREATER THAN
0018 0003            UN     EQU     3      UNCONDITIONAL
0019                 *  PSW LOWER EQUATES
0020 00C0            CC     EQU     H'C0'  CONDITIONAL CODES
0021 0020            IDC    EQU     H'20'  INTERDIGIT CARRY
0022 0010            RS     EQU     H'10'  REGISTER BANK
0023 0008            WC     EQU     H'08'  1=WITH 0=WITHOUT CARRY
0024 0004            OVF    EQU     H'04'  OVERFLOW
0025 0002            COM    EQU     H'02'  1=LOGIC 0=ARITHMETIC COMPARE
0026 0001            C      EQU     H'01'  CARRY/BORROW
0027                 *  PSW UPPER EQUATES
0028 0080            SENS   EQU     H'80'  SENSE BIT
0029 0040            FLAG   EQU     H'40'  FLAG BIT
0030 0020            II     EQU     H'20'  INTERRUPT INHIBIT
0031 0007            SP     EQU     H'07'  STACK POINTER
0032                 *  IO PORT DEFINITIONS
0033 0007            LEDS   EQU     H'07'  USER EXTENDED IO PORT
0034                 *  INTERUPT VECTORS
0035 0007            UINTV  EQU     H'07'  USER DIRECT INTERUPT VECTOR
0036 0087            UINTVI EQU     H'87'  USER INDIRECT INTERUPT VECTOR
0037                 * HARDWARE DEFINITIONS
0038 00FE            KBDIN  EQU     H'FE'  ADDRESS OF KBD IO PORT
0039 00F9            SEG    EQU     H'F9'  IO ADDRESS OF SEGMENT DRIVER
0040 00F9            DISP   EQU     SEG
0041 00FA            DIGIT  EQU     H'FA'  ADDRESS OF DIGIT ENABLE
0042 00F8            CTBYT  EQU     H'F8'  ADDRESS OF CONTROL BYTE
0043 00F8            CAS    EQU     CTBYT  ADDRESS OF CASSETTE INTERFACE
0044 00FB            OPRQCT EQU     H'FB'  ADDRESS OF OPREQ COUNTER
0045 00FD            LADRH  EQU     H'FD'  ADDRESS OF LAST ADDRESS REG HI BYTE
0046 00FC            LADRL  EQU     H'FC'  ADDRESS OF LAST ADDRESS REG LO BYTE
```

```
LINE ADDR  OBJECT  E SOURCE

0048                 ****************************************************
0049                 *
0050 0000                  ORG     H'1800'-64      TRAINING CARD RAM AREA
0051                 *
0052 17C0            SCTCH  RES     8       8 BYTE SCRATCH AREA
0053 17C6            TEMP   EQU     SCTCH+6 TEMP STORAGE
0054 17C8            EAD    RES     2       STOP ADDRESS FOR WCAS
0055 17CA            BAD    RES     2       BEGINNING ADDRESS FOR WCAS
0056 17CC            BPD    RES     1       DATA TO BE RESTORED IN BREAK LOC
0057 17CD            BPL    RES     2       ADDRESS OF BREAK POINT LOC
0058 17CF            BPF    RES     1       BREAK POINT SET FLAG
0059 17D0            SSF    RES     1       SINGLE STEP SET FLAG
0060 17D1            DISBUF RES     8       8 BYTE DISPLAY REGISTER
0061 17D9            SAVREG RES     4       A PLACE TO SAVE R0 THRU R3 OF ONE BANK
0062 17DD            MEM    RES     2       ADDRESS FOR ALTER OR PATCH COMMAND
0063 17DF            FID    RES     2       FILE ID FLAG AND FILE ID
0064 17E1            BCC    RES     1       BLOCK CHECK CHAR
0065 17E2            BSTT   RES     1       SAVE UNITS DIGIT
0066 17E3            T      RES     2       TEMP REGISTER
0067 17E5            T1     RES     1       TEMP REGISTER
0068 17E6            T2     RES     1       TEMP REGISTER
0069 17E7            T3     RES     1       TEMP REGISTER
0070 17E8            LADR   RES     2       COPY OF LAST ADDRESS REGISTER
0071 17EA            SLADR  RES     2       SAVE LOCATION FOR LADR
0072 17EC            KFLG   RES     2       KBD SCAN FLAGS
0073 17EE                   RES     1       KBD DEBOUNCE COUNT
0074 17EF            ALTF   RES     1       DISPLAY AND ALTER FLAG
0075 17F0            RESTF  RES     1       RESTORE REGISTERS FLAG
0076 17F1            IFLG   RES     1       INTERUPT INHIBIT FLAG
0077 17F2            UREG   RES     12      STORAGE FOR USER REGISTERS
0078 17FE            PWRON  RES     2       WHEN POWER ON THESE LOC CONTAIN H'5946'
0079                 *****************************************************
```

```
LINE ADDR  OBJECT  E SOURCE

0081                 **********************************************************
0082 1800                   ORG H'1800'     BEGINING OF TRAINING CARD ROM AREA
0083                 **********************************************************
0084                 *
0085                 *SAVE ALL REGISTERS UPON ENTRY TO PROGRAM
0086                 *
0087                 *REGISTERS USED
0088                 *
0089                 *R0 THRU R3' PSU PSL
0090                 *
0091                 *SUBROUTINES CALLED
0092                 *
0093                 *NONE
0094                 *
0095                 *RAM MEMORY USED
0096                 *
0097                 *UREG  = R0
0098                 *UREG+1        = R1
0099                 *UREG+2        = R2
0100                 *UREG+3        = R3
0101                 *UREG+4        = R1'
0102                 *UREG+5        = R2'
0103                 *UREG+6        = R3'
0104                 *UREG+7        = PSU
0105                 *UREG+8        = PSL
0106                 *UREG+9        = PPSL INSTRUCTION OPCODE
0107                 *UREG+10       = PSL
0108                 *UREG+11       = RETC,UN       INSTRUCTION OPCODE
0109                 *
0110                 **************************************************************
0111                 *
0112 1800 C870       SAVRG  STRR,R0 UREG    SAVE R0
0113 1802 13                SPSL            GET PSL
0114 1803 C875              STRR,R0 UREG+8  SAVE PSL
0115 1805 C875              STRR,R0 UREG+10 SAVE PSL         FOR RESTORE ROUTINE
0116 1807 12                SPSU            GET PSU
0117 1808 C86F              STRR,R0 UREG+7  SAVE PSU
0118 180A 7620              PPSU    II      SET INTERUPT INHIBIT
0119 180C 7510              CPSL    RS      CLEAR REGISTER SWITCH
0120 180E C963              STRR,R1 UREG+1  SAVE R1
0121 1810 CA62              STRR,R2 UREG+2  SAVE R2
0122 1812 CB61              STRR,R3 UREG+3  SAVE R3
0123 1814 7710              PPSL    RS      SET REGISTER SWITCH
0124 1816 C95E              STRR,R1 UREG+4  SAVE R1'
0125 1818 CA5D              STRR,R2 UREG+5  SAVE R2'
0126 181A CB5C              STRR,R3 UREG+6  SAVE R3'
0127 181C 75FF              CPSL    255     CLEAR PSL
0128 181E 7702              PPSL    COM     DO LOGICAL COMPARES
0129 1820 54FD              REDE,R0 LADRH   GET LAST ADDRESS HI BYTE
0130 1822 CC17E8            STRA,R0 LADR    SAVE IN MEMORY
0131 1825 54FC              REDE,R0 LADRL   GET LAST ADDRESS LO BYTE
0132 1827 CC17E9            STRA,R0 LADR+1  SAVE IT
0133 182A 20                EORZ    R0      GET A 0
0134 182B CC17F1            STRA,R0 IFLG    CLEAR INTERUPT INHIBIT FLAG
```

```
LINE ADDR  OBJECT  E SOURCE

0136                 *****************************************************************
0137                 *
0138                 *
0139                 *
0140                 *PROGRAM ENTRY ROUTINE
0141                 *
0142                 *
0143                 *DECIDES HOW REACHED ENTRY POINT OF PROGRAM
0144                 *
0145                 *1 POWER ON
0146                 *2 SINGLE STEP
0147                 *3 MONITOR PUSHBUTTON ON KEY BOARD
0148                 *4 BREAKPOINT
0149                 *
0150                 *
0151                 *
0152                 *
0153                 ******************************************************
0154                 *
0155 182E 084E       BEG    LODR,R0 PWRON   CHECK POWER ON FLAG
0156 1830 E459              COMI,R0 H'59'   AFTER POWER VALUE OF FLAG IS
0157                 *                      H'5946'
0158 1832 9813       BEG1   BCFR,EQ INIT    IF NOT CORRECT THIS IS POWER ON
0159                 *                      GO INITIALIZE THE MONITOR FLAGS
0160 1834 0849       BEG3   LODR,R0 PWRON+1 CHECK LO BYTE OF POWER ON FLAG
0161 1836 E446              COMI,R0 H'46'   IS SECOND BYTE CORRECT
0162 1838 980D              BCFR,EQ INIT IF NOT INITIALIZE THE PROGRAM
0163 183A 0C17D0            LODA,R0 SSF     CHECK THE SINGLE STEP' FLAG
0164 183D 9C19C9            BCFA,EQ SGLSTP  IF FLAG THEN GO SINGLE STEP
0165 1840 0899              LODR,R0 *MON+1     SEE IF BREAK POINT ENABLED
0166 1842 9C197A     BEG2   BCFA,EQ BRKPT  GO EXECUTE THE BREAK POINT ROUTINE
0167 1845 1B13              BCTR,UN MON     MUST BE MONITOR KEY
0168                 *
```

```
LINE ADDR  OBJECT  E SOURCE

0170                 *****************************************************
0171                 *
0172                 *
0173                 *KEY BOARD MONITOR ROUTINE
0174                 *
0175                 *
0176                 *REGISTERS USED
0177                 *
0178                 *R0 SCRATCH
0179                 *R1 SCRATCH
0180                 *R2 SCRATCH
0181                 *R3 NOT USED
0182                 *
0183                 *SUBROUTINES USED
0184                 *
0185                 *MOV MOVE DATA TO DISPLAY BUFFER
0186                 *DISPLY DISPLAY MESSAGE AND KEY BOARD SCAN
0187                 *
0188                 *
0189                 *RAM MEMORY USED
0190                 *
0191                 *PWRON POWER ON FLAG
0192                 *SSF   SINGLE STEP FLAG
0193                 *BPF   BREAK POINT FLAG
0194                 *BPL   BREAK POINT LOCATION
0195                 *
0196                 *************************************************
0197                 *
0198 1847 0459       INIT   LODI,R0 H'59'   SET THE POWER ON FLAG
0199 1849 CC17FE            STRA,R0 PWRON   TO POWER ON VALUE H'5946'
0200 184C 0446              LODI,R0 H'46'
0201 184E CC17FF            STRA,R0 PWRON+1
0202 1851 20                EORZ    R0      GET A 0
0203 1852 CC17DD            STRA,R0 MEM     PRESET INDIRECT ADDRESS MEM
0204 1855 CC17DE            STRA,R0 MEM+1
0205 1858 C881              STRR,R0 *MON+1  CLEAR BREAK POINT FLAG
0206 185A 0C17CF     MON    LODA,R0 BPF     GET BREAK POINT FLAG
0207 185D 180F              BCTR,EQ MON5    BREAK POINT NOT SET
0208 185F 0C17CC            LODA,R0 BPD     GET BREAK POINT DATA
0209 1862 CC97CD            STRA,R0 *BPL    CLEAR BREAK POINT
0210 1865 EC97CD            COMA,R0 *BPL    CHECK DATA STORED CORRECTLY
0211 1868 1804              BCTR,EQ MON5    BREAK POINT CLEARED OK
0212 186A 0701              LODI,R3 1       BREAK POINT DIDN'T CLEAR
0213 186C 9BE8              ZBRR    *ERR    GOTO ERROR
0214 186E 20         MON5   EORZ    R0      GET A 0
0215 186F D4F8              WRTE,R0 CTBYT  CLEAR CONTROL BYTE
0216 1871 CC17D0            STRA,R0 SSF     CLEAR SINGLE STEP FLAG
0217 1874 051F       MON3   LODI,R1 <HELLO-1        GET ADDRESS OF HELLO MESSAGE
0218 1876 0694              LODI,R2 >HELLO-1
0219 1878 BBFE       MON1   ZBSR *MOV     MOVE MESSAGE TO DISBUF
0220 187A 20         MON4   EORZ    R0      SET FLAG TO WAIT FOR ENTRY
0221 187B BBEC              ZBSR    *DISPLY DISPLAY MESSAGE AND SCAN KEYBOARD
0222 187D F680       MON2   TMI,R2  H'80'   CHECK COMMAND FLAG
0223 187F 9816              BCFR,EQ ERR2     IF FLAG NOT SET ERROR
0224 1881 460F              ANDI,R2 H'0F'   MASK COMMAND VALUE
```

```
LINE ADDR  OBJECT  E SOURCE

0225 1883 E607             COMI,R2 7       MAX COMMAND VALUE
0226 1885 1910             BCTR,GT ERR2     ERROR CODE VALUE TO LARGE
0227 1887 D2               RRL,R2          MULTIPLY INDEX BY 2
0228 1888 0E78A4           LODA,R0 CMD,R2  SET UP AN INDIRECT ADDRESS
0229 188B CC17E3           STRA,R0 T       TO THE FUNCTION WANTED
0230 188E 0E78A5           LODA,R0 CMD+1,R2
0231 1891 CC17E4           STRA,R0 T+1
0232 1894 1F97E3           BCTA,UN *T      EXECUTE A COMMAND
0233                *
0234                *
0235 1897 0702      ERR2   LODI,R3 2       INVALID COMMAND SEQUENCE
0236 1899 051F      ERRI   LODI,R1 <ERROR-1  GET ADDRESS OF ERROR MESSAGE
0237 189B 0684             LODI,R2 >ERROR-1
0238 189D BBFE             ZBSR    *MOV    MOVE MESSAGE TO DISBUF
0239 189F CF17D8           STRA,R3 DISBUF+7        WRITE THE ERROR NUMBER
0240 18A2 1B56             BCTR,UN MON4    GO LOOK FOR NEW COMMAND
0241                *
0242                *COMMAND ADDRESS TABLE
0243                *
0244 18A4 1C91      CMD    ACON    WCAS    WRITE CASSETTE COMMAND
0245 18A6 1D61             ACON    SCBP     BREAK POINT COMMAND
0246 18A8 1BAC             ACON    RCAS    READ CASSETTE COMMAND
0247 18AA 1A7E             ACON    REG     REGISTER DISPLAY AND ALTER COMMAND
0248 18AC 18B4             ACON    SSTEP      SINGLE STEP COMMAND
0249 18AE 1A0C             ACON    ALTER     DISPLAY AND ALTER MEMORY
0250 18B0 1E59             ACON    GO      GOTO COMMAND
0251 18B2 187A             ACON    MON4    ENTR/NEXT KEY IS NOT COMMAND
0252                *
```

```
LINE ADDR  OBJECT  E SOURCE

0254                 **********************************************************
0255                 *
0256                 *
0257                 *SINGLE STEP ROUTINES
0258                 *THIS ROUTINE WRITTEN BY BBC
0259                 *
0260                 * PROCESSOR TRANSFERS CONTROL TO USER PROGRAM
0261                 * AFTER COMPUTING THE NUMBER OF OPREQ'S TILL
0262                 * THE NEXT INSTRUCTION FETCH.
0263                 *
0264                 *
0265                 *REGISTERS USED
0266                 *
0267                 *R0 THRU R3 SCRATCH
0268                 *
0269                 *
0270                 *SUBROUTINES CALLED
0271                 *
0272                 *RLADR RESTORE LAST ADDRESS REGISTER
0273                 *
0274                 *
0275                 *RAM LOCATIONS USED
0276                 *
0277                 *LADR  LAST ADDRESS REGISTER
0278                 *T3    TEMP REGISTER
0279                 *TEMP  TEMP REGISTER
0280                 *SCTCH SCRATCH REGISTER
0281                 *
0282                 **************************************************************
0283 0000            OVHD   EQU     0      NEGATIVE NUMBER OF OPREQ'S
0284                 *
0285                 *
0286                 *CHECK IF NEXT SINGLE STEP IS IN MONITOR AREA
0287                 *
0288 18B4 0C17E8     SSTEP  LODA,R0 LADR    GET MSB OF LADR
0289 18B7 E410              COMI,R0 H'10'   IS ADDRESS LT H'1000'
0290 18B9 1A08              BCTR,LT SSTEP1  GO SINGLE STEP
0291 18BB E420              COMI,R0 H'20'   IS ADDRESS GT OR EQ H'2000'
0292 18BD 9A04              BCFR,LT SSTEP1  GO SINGLE STEP
0293 18BF 0709              LODI,R3 9       NEXT SINGLE STEP ENTERS MONITOR
0294 18C1 9BE8              ZBRR    *ERR    GOTO ERROR
0295 18C3 047F       SSTEP1 LODI,R0 127     SET SINGLE STEP FLAG
0296 18C5 CC17D0            STRA,R0 SSF     STORE IT
0297 18C8 0420              LODI,R0 H'20'   SET THE INTERUPT INHIBIT
0298 18CA CC17F1            STRA,R0 IFLG    SAVE IN INTERUPT INHIBIT FLAG
0299 18CD 7508              CPSL WC         CLEAR WITH CARRY IF SET
0300 18CF 0601       SSTEP2 LODI,R2 1       SET INDEX
0301 18D1 0EF7E8            LODA,R0 *LADR,R2        GET SECOND BYTE OF INSTRUCTION
0302 18D4 CC17E7            STRA,R0 T3      SAVE IT FOR LATER
0303 18D7 0F97E8            LODA,R3 *LADR   GET NEXT INSTRUCTION
0304 18DA 03                LODZ    R3      SAVE INSTRUCTION IN R0
0305 18DB 471C              ANDI,R3 H'1C'   EXTRACT INSTRUCTION CLASS
0306 18DD 0500              LODI,R1 OVHD    SET OVERHEAD OPREQ COUNT
0307 18DF 0605              LODI,R2 5       SHIFT OR MOVE COUNT
0308 18E1 F420              TMI,R0  H'20'   TEST FOR ODD OPCODE IN CLASS4
```

```
LINE ADDR  OBJECT  E SOURCE

0309 18E3 9F1902          BXA     CBRTB,R3       BRANCH TO CLASS PROCESSOR
0310                *
0311                * CLASS 5.     MIXED NUMBER OF OPREQ'S.
0312                *
0313 18E6 FA2F      CL5B  BDRR,R2 CL5A
0314 18E8 4707            ANDI,R3 H'07'   MASK TO OPCODE
0315 18EA 0F7967          LODA,R0 CL5TB,R3       GET NUMBER OF OPREQ'S FROM TABLE.
0316 18ED C1              STRZ    R1
0317                *
0318                * WRITE OPREQ COUNT AND EXIT TO USER
0319                *
0320 18EE 1F195E    EXIT BCTA,UN EXIT4     GOTO USER
0321                *
0322                * RETURN FROM TEST BRANCH, IF BRANCH TAKEN
0323                *
0324 18F1 3F196F    BRCH  BSTA,UN RLADR   RESTORE LAST ADDRESS REGISTER
0325 18F4 0D17C6    BRCH1 LODA,R1 TEMP    GET OPREQ COUNT BACK AFTER TEST BRANCH
0326 18F7 7508            CPSL    WC      CLEAR PSL WC BIT
0327                *
0328                * ROUTINE TO ADD 2 OPREQ'S IF INDIRECT APPLIES.
0329                *
0330 18F9 0C17E7    CIND  LODA,R0 T3      GET SECOND BYTE OF INSTRUCTION
0331 18FC F480            TMI,R0  H'80'   TEST INDIRECT BIT
0332 18FE 1806            BCTR,0  PLS2    SET, ADD 2 OPREQ'S
0333 1900 1B6C            BCTR,UN EXIT    NOT SET, DO NOT ADD
0334                *
0335                * CLASS PROCESSOR TABLE.
0336                *
0337 1902           CBRTB EQU     $
0338                *
0339 1902 A501      PLS1  SUBI,R1 1       CLASS 0.        1 OPREQ
0340 1904 1B68            BCTR,UN EXIT
0341 1906 A502      PLS2  SUBI,R1 2       CLASS 1.        2 OPREQ'S
0342 1908 1B64            BCTR,UN EXIT
0343 190A A503            SUBI,R1 3       CLASS 2.        3 OPREQ'S + INDIRECT
0344 190C 1B6B            BCTR,UN CIND
0345 190E A504            SUBI,R1 4       CLASS 3.        4 OPREQ'S + INDIRECT
0346 1910 1B67            BCTR,UN CIND
0347 1912 1872            BCTR,EQ PLS2    CLASS4          2 OPREQS IF OPCODE ODD
0348 1914 1B6C            BCTR,UN PLS1                    1 OPREQ IF OPCODE EVEN
0349 1916 C3              STRZ    R3      CLASS 5.        MIXED NUMBER OF OPREQ'S
0350 1917 53        CL5A  RRR,R3          SHIFT OPCODE TO LOW BYTE
0351 1918 1B4C            BCTR,UN CL5B    AND LOOK UP IN TABLE
0352 191A 8501            ADDI,R1 1       CLASS 6.        2 OPREQ'S + IND IF BRANCH TAKEN
0353 191C 6404            IORI,R0 H'04'   CONVERT TO CLASS 7.
0354 191E A503            SUBI,R1 3       CLASS 7.        3 OPREQ'S + IND IF BRANCH TAKEN
0355                *
0356                * CLASS 6 AND 7.
0357                * ADD 2 OPREQ'S IF INDIRECT AND BRANCH IS TAKEN.
0358                *
0359 1920 C9D3            STRR,R1 *BRCH1+1 SAVE PRESENT NUMBER OF OPREQ'S IN TEMP
0360 1922 D5FB            WRTE,R1 OPRQCT  ALSO OUTPUT TO HARDWARE
0361 1924 F440            TMI,R0  H'40'   TEST FOR REGISTER CLASS
0362 1926 1804            BCTR,0  CL67B   IF SO, DO NOT TEST FOR UNCONDITIONAL
0363 1928 F403            TMI,R0  H'03'   IS BRANCH UNCONDITIONAL
0364 192A 184D            BCTR,0  CIND    IF SO, DO NOT TEST BRANCH
```

```
LINE ADDR  OBJECT  E SOURCE

0365 192C F4E0       CL67B  TMI,R0  H'E0'   TEST FOR BDR INST
0366 192E 1802              BCTR,0  CL67C   IF SO, DO NOT REMOVE 'SUBROUTINE' BIT
0367 1930 44DF              ANDI,R0 H'DF'   REMOVE SUBROUTINE BIT FROM OPCODE.
0368 1932 CC17C0     CL67C  STRA,R0 SCTCH   STORE IN TEST AREA
0369 1935 0E7951     MVCODE LODA,R0 BRCD,R2 GET ROM CODE
0370 1938 CE77C0            STRA,R0 SCTCH,R2  STORE IN RAM
0371 193B FA78              BDRR,R2 MVCODE  DO UNTILL ALL HAS BEEN MOVED.
0372                 *
0373                 *SAVE LADR
0374                 *
0375 193D 0C17E8     SLAD1  LODA,R0 LADR    GET LAST ADDRESS REG
0376 1940 C8AE              STRR,R0 *RLADR+1   SAVE IT
0377 1942 0C17E9     SLAD2  LODA,R0 LADR+1
0378 1945 C8AE              STRR,R0 *RLADR1+1
0379 1947 0417              LODI,R0 <SCTCH  GET ADDRESS SCRATCH
0380 1949 C8F3              STRR,R0 *SLAD1+1
0381 194B 04C0              LODI,R0 >SCTCH
0382 194D CC9943            STRA,R0 *SLAD2+1
0383 1950 1B93              BCTR,UN *EXIT3+1      DO TEST BRANCH
0384                 *
0385                 *THIS IS CODE FOR TEST BRANCH
0386                 *
0387 1951            BRCD   EQU     $-1
0388 1952 18F1              ACON    BRCH    ADDRESS FOR TEST BRANCH
0389 1954 1F1957            BCTA,UN EXIT2      RETURN IF BRANCH NOT TAKEN
0390                 *
0391 1957 3B16       EXIT2  BSTR,UN RLADR   RESTORE LAST ADDRESS REG
0392 1959 0D17C6            LODA,R1 TEMP    GET OPREQ COUNT
0393 195C 7508              CPSL    WC      CLEAR WITH CARRY
0394 195E D5FB       EXIT4  WRTE,R1 OPRQCT  SET THE OPREQ COUNTER
0395 1960 20                EORZ    R0      CLEAR INTERUPT INHIBIT FLAG
0396 1961 CC17F1            STRA,R0 IFLG    SAVE IN INTERUPT INHIBIT FLAG
0397 1964 1F1E59     EXIT3  BCTA,UN GO
0398                 *
0399                 * CLASS 5 OPREQ TABLE
0400                 *
0401 1967 FF         CL5TB  DATA    OVHD-1      RETC
0402 1968 FF                DATA    OVHD-1      RETE
0403 1969 FD                DATA    OVHD-3      REDE
0404 196A FE                DATA    OVHD-2      C-P PSW
0405 196B FF                DATA    OVHD-1      DAR
0406 196C FE                DATA    OVHD-2      TPSW
0407 196D FD                DATA    OVHD-3      WRTE
0408 196E FE                DATA    OVHD-2      TMI
0409                 *
0410                 *RLADR RESTORE LAST ADDRESS REG
0411                 *
0412 196F 0C17EA     RLADR  LODA,R0 SLADR   GET SAVED LADR
0413 1972 C8CA              STRR,R0 *SLAD1+1
0414 1974 0C17EB     RLADR1 LODA,R0 SLADR+1
0415 1977 C8CA              STRR,R0 *SLAD2+1
0416 1979 17                RETC,UN
0417                 *
```

```
LINE ADDR  OBJECT  E SOURCE

0419                 ************************************************************************
0420                 *
0421                 *
0422                 *BREAK POINT AND SINGLE STEP RUN TIME CODE
0423                 *
0424                 *
0425                 *SINGLE STEP
0426                 *
0427                 *WHEN ENTERED AT SINGLE STEP. SINGLE STEP FLAG IS CLEARED
0428                 *AND DISPLAY IS ' ADDR DD'
0429                 *
0430                 *
0431                 *WHEN ENTERED AT BREAK POINT AND BREAK POINT IS SET AND MATCHES
0432                 *BREAK POINT REGISTER. THE DISPLAY IS '-ADDR DD'
0433                 *
0434                 *
0435                 *REGISTER USED
0436                 *
0437                 *R0
0438                 *
0439                 *SUBROUTINE CALLED
0440                 *
0441                 *DLSLD PREPARE BINARY DATA FOR DISPLAY
0442                 *
0443                 *
0444                 *RAM MEMORY USED
0445                 *
0446                 *DISBUF DISPLAY BUFFER
0447                 *BPF BREAK POINT FLAG
0448                 *BPL BREAK POINT LOCATION
0449                 *BPD DATA FOR BREAK POINT LOCATION
0450                 *LADR COPY OF LAST ADDDRESS REGISTER
0451                 *SSF SINGLE STEPFLAG
0452                 *
0453                 ************************************************************************
0454                 *
0455 197A 0C17E8     BRKPT  LODA,R0 LADR    GET LAST ADDRESS REGISTER
0456 197D 0D17E9     BRK3   LODA,R1 LADR+1
0457 1980 7709              PPSL    C+WC    SET CARRY AND WITH CARRY
0458 1982 A501              SUBI,R1 1       DECREMENT LAST ADDRESS REG
0459 1984 A400              SUBI,R0 0       SO CAN COMPARE TO BREAK POINT REGISTER
0460 1986 447F              ANDI,R0 H'7F    MASK OFF UNUSED BIT
0461 1988 7509              CPSL    C+WC    CLEAR CARRY AND WITH CARRY
0462 198A E8AF       BRK2   COMR,R0 *BRKPT2+1 COMPARE WITH BPL
0463 198C 9C185A     BRK1   BCFA,EQ MON    NO COMPARE
0464 198F E9AF              COMR,R1 *BRKPT1+1 COMPARE WITH BPL+1
0465 1991 98FA              BCFR,EQ *BRK1+1   NO COMPARE
0466 1993 C8E6              STRR,R0 *BRKPT+1        IF COMPARE UP DATE PC
0467 1995 C9E7              STRR,R1 *BRK3+1
0468 1997 0C17CC            LODA,R0 BPD     IF COMAPRE CLEAR BREAK POINT
0469 199A CC97CD            STRA,R0 *BPL
0470 199D EC97CD            COMA,R0 *BPL    ERROR CHECK OF DATA WRITTEN
0471 19A0 1804              BCTR,EQ BRK0    DATA STORED OK
0472 19A2 0701              LODI,R3 1       BREAK POINT NOT CLEARED OK
0473 19A4 9BE8              ZBRR    *ERR
```

```
LINE ADDR  OBJECT  E SOURCE

0474 19A6 E440        BRK0   COMI,R0 H'40'   HALT INSTRUCTION OPCODE
0475 19A8 1808               BCTR,EQ BRKPT9  IF HALT DON'T DO HIDDEN SINGLE STEP
0476 19AA 0480               LODI,R0 H'80'   SET FLAG FOR HIDDEN SINGLE STEP
0477 19AC CC17CF      BRKPT3 STRA,R0 BPF     SET FLAG IN BREAK POINT
0478 19AF 1F18B4             BCTA,UN SSTEP   EXECUTE ONE USER INSTRUCTION
0479 19B2 047F        BRKPT9 LODI,R0 127     SET BREAK POINT FLAG
0480 19B4 C8F7               STRR,R0 *BRKPT3+1
0481 19B6 0419               LODI,R0 H'19'   DASH SYMBOL FOR BREAK POINT
0482 19B8 C89A               STRR,R0 *BRKPT8+1  SET THE DASH SYMBOL IN DISBUF
0483 19BA 0C17CD      BRKPT2 LODA,R0 BPL     GET BREAK POINT ADDRESS
0484 19BD 3B32               BSTR,UN BRKPT7  SET THE DISPLAY
0485 19BF 0C17CE      BRKPT1 LODA,R0 BPL+1
0486 19C2 3B36               BSTR,UN BRKPT6
0487 19C4 0C97CD             LODA,R0 *BPL    GET INSTRUCTION OPCODE
0488 19C7 1B1A               BCTR,UN BRKPT5
0489                  *
0490                  *ENTRY POINT FOR SINGLE STEP
0491                  *
0492 19C9 20           SGLSTP EORZ    R0      GET A 0
0493 19CA CC17D0             STRA,R0 SSF     CLEAR SINGLE STEP FLAG
0494 19CD 08DE               LODR,R0 *BRKPT3+1       CHECK BREAK POINT FLAG
0495 19CF 1A61               BCTR,NG BRKPT9  DID A HIDDEN SINGLE STEP
0496                  *                      DISPLAY THE BREAK POINT
0497 19D1 0417        SGLST9 LODI,R0 H'17'   BLANK SYMBOL
0498 19D3 CC17D1      BRKPT8 STRA,R0 DISBUF  SET DISPLAY BUFFER
0499 19D6 0C17E8             LODA,R0 LADR    GET ADDRESS
0500 19D9 3B16               BSTR,UN BRKPT7  SET THE DISPLAY
0501 19DB 0C17E9             LODA,R0 LADR+1
0502 19DE 3B1A               BSTR,UN BRKPT6  SET THE DISPLAY
0503 19E0 0C97E8             LODA,R0 *LADR   GET INSTRUCTION DATA
0504 19E3 3B03        BRKPT5  BSTR,UN BRKPTI  SET UP DISPLAY
0505 19E5 1F187A             BCTA,UN MON4     GOTO MONITOR
0506                  *
0507                  *SET UP DISBUF 6&7
0508                  *
0509 19E8 BBF4        BRKPTI ZBSR    *DISLSD CONVERT TO BIN FOR DISPLAY
0510 19EA CC17D7             STRA,R0 DISBUF+6
0511 19ED CD17D8             STRA,R1 DISBUF+7
0512 19F0 17                 RETC,UN
0513                  *
0514                  *SET UP DISBUF 1&2
0515                  *
0516 19F1 BBF4        BRKPT7 ZBSR    *DISLSD CONVERT BIN TO DISPLAY
0517 19F3 CC17D2             STRA,R0 DISBUF+1
0518 19F6 CD17D3             STRA,R1 DISBUF+2
0519 19F9 17                 RETC,UN
0520                  *
0521                  *SETUP DISBUF 3&4
0522                  *
0523 19FA BBF4        BRKPT6 ZBSR    *DISLSD CONVERT BIN TO DISPLAY
0524 19FC CC17D4             STRA,R0 DISBUF+3        STORE DATA
0525 19FF CD17D5             STRA,R1 DISBUF+4
0526 1A02 0417               LODI,R0 H'17'   BLANK SYMBOL
0527 1A04 CC17D6             STRA,R0 DISBUF+5
0528 1A07 17                 RETC,UN
```

```
LINE ADDR  OBJECT  E SOURCE

0530                 *******************************************************
0531                 *
0532                 *
0533                 *DISPLAY AND ALTER MEMORY ROUTINE
0534                 *PATCH MEMORY ROUTINE
0535                 *
0536                 *
0537                 *REGISTERS USED
0538                 *
0539                 *R0 SCRATCH
0540                 *R1 SCRATCH
0541                 *R2 SCRATCH
0542                 *R3 SCRATCH
0543                 *
0544                 *SUBROUTINES CALLED
0545                 *
0546                 *GAD GET ADDRESS PARAMETER
0547                 *GNP GET NUMBER PARAMETER
0548                 *ROT ROTATE R0 1 NIBBLE LEFT
0549                 *BRKPT4        SETUP DISPLAY 6&7
0550                 *BRKPT6        SETUP DISPLAY 3&4
0551                 *BRKPT7        SETUP DISPLAY 1&2
0552                 *
0553                 *RAM MEMORY USED
0554                 *
0555                 *MEM INDIRECT ADDRESS MEMORY POINTER
0556                 *ALTF ALTER FLAG = 1 FOR DISPLAY AND ALTER
0557                 *                  3 OR 5 FOR PATCH
0558                 *
0559                 ******************************************************
0560                 *
0561                 *
0562                 *ENTRY POINT FOR PATCH COMMAND
0563                 *
0564 1A08 0403       PTCH   LODI,R0 3       SET ALTER FLAG TO PATCH
0565 1A0A 1B02              BCTR,UN ALTER5
0566                 *
0567                 *ENTRY POINT FOR DISPLAY AND ALTER COMMAND
0568                 *
0569 1A0C 0401       ALTER  LODI,R0 1       SET ALTER FLAG TO ALTER
0570 1A0E C8A4       ALTER5 STRR,R0 *ALTER1+1       STORE IN ALTF
0571 1A10 3F1B04            BSTA,UN GAD     DISPLAY AD= AND WAIT TILL DIGITS ENTERED
0572 1A13 E687              COMI,R2 H'87'   ENTR/NXT?
0573 1A15 9C187D            BCFA,EQ MON2    NEW FUNCTION ABORT ALTER COMMAND
0574 1A18 5B0E              BRNR,R3 ALTER4  NO ADDRESS ENTERED CONTINUE FROM LAST LOCATION
0575 1A1A C88D              STRR,R0 *ALTER4+1 MEM+1   SAVE ADDRESS DATA
0576 1A1C C981              STRR,R1 *AL1+1
0577 1A1E 0E17DD     AL1    LODA,R2 MEM     GET DATA
0578 1A21 0717              LODI,R3 H'17'      BLANK
0579 1A23 CF17D8            STRA,R3 DISBUF+7        CLEAR DISPLAY
0580 1A26 1B05              BCTR,UN ALTER2  SET UP DISPLAY
0581                 *
0582                 *NO ADDRESS CONTINUE FROM LAST ADDRESS
0583                 *
0584 1A28 0C17DE     ALTER4 LODA,R0 MEM+1   GET ADDRESS
```

```
LINE ADDR  OBJECT  E SOURCE

0585 1A2B 0AF2            LODR,R2 *AL1+1 MEM
0586                *
0587                *UPDATE THE DISPLAY
0588                *
0589 1A2D 3B4B      ALTER2 BSTR,UN BRKPT6  SET UP ADDRESS DISPLAY
0590 1A2F 02              LODZ    R2      GET MSD
0591 1A30 3F19F1          BSTA,UN BRKPT7  SET UP DISPLAY
0592                *
0593                *
0594 1A33 0C17EF    ALTER1 LODA,R0 ALTF    CHECK ALTER FLAG
0595 1A36 E401            COMI,R0 1       PATCH COMMAND
0596 1A38 1807            BCTR,EQ ALTER8  NOT PATCH
0597 1A3A 0417            LODI,R0 H'17'   BLANK CHAR
0598 1A3C CC17D7          STRA,R0 DISBUF+6
0599 1A3F 1B05            BCTR,UN ALTER9  PATCH COMMAND
0600                *
0601 1A41 0C97DD    ALTER8 LODA,R0 *MEM    GET THE DATA
0602 1A44 BBEA            ZBSR  *BRKPT4  SET UP DATA VALUE DISPLAY
0603 1A46 08EC      ALTER9 LODR,R0 *ALTER1+1    SET FLAG TO SINGLE BYTE DATA
0604 1A48 BBFC            ZBSR *GNPA    DISPLAY BUFFER AND WAIT FOR NEW ENTRY
0605 1A4A 5B0C            BRNR,R3 ALTER3   NO DATA
0606 1A4C CC97DD          STRA,R0 *MEM    CHANGE DATA IN LOCATION
0607 1A4F EC97DD          COMA,R0 *MEM    CHECK DATA STORED OK
0608 1A52 1804            BCTR,EQ ALTER3  DATA STORED OK
0609 1A54 0703            LODI,R3 3       ALTER OR PATCH WRITE ERROR
0610 1A56 9BE8            ZBRR    *ERR    GOTO ERROR
0611                *
0612                *EXIT FROM COMMAND
0613                *
0614 1A58 08DA      ALTER3 LODR,R0 *ALTER1+1    EXIT FROM ALTER OR PATCH
0615 1A5A E401            COMI,R0 1       IS IT PATCH
0616 1A5C 9807            BCFR,EQ ALTER6  IF YES TAKE THIS BRANCH
0617 1A5E E687            COMI,R2 H'87'   IS IT ALTER NEXT KEY FUNCTION?
0618 1A60 9C187D    AL2    BCFA,EQ MON2    GO TO MONITOR NEW COMMAND
0619 1A63 1B0B            BCTR,UN ALTER7   GO UPDATE THE DISPLAY
0620                *
0621                *EXIT FROM PATCH
0622                *
0623 1A65 E60F      ALTER6 COMI,R2 H'0F'   WAS LAST KEY FUNCTION KEY
0624 1A67 19F8            BCTR,GT *AL2+1 MON2    FUNCTION KEY WAS LAST GO TO MONITOR
0625 1A69 0405            LODI,R0 5       RETURN ON SECOND DIGIT FLAG
0626 1A6B C8C7            STRR,R0 *ALTER1+1       SAVE IN ALTF
0627 1A6D CE17D8          STRA,R2 DISBUF+7        SET DISPLAY
0628                *
0629                *INCREMENT INDIRECT ADDRESS
0630                *
0631 1A70 3F1C55    ALTER7 BSTA,UN INK     INCREMENT THE ADDRESS
0632 1A73 1F1A2D          BCTA,UN ALTER2
0633                *
0634                *PREPARE BIN DATA FOR DISPLAY
0635                *
0636 1A76 C1        DISLSI STRZ    R1      SAVE NUMBER IN R0
0637 1A77 44F0            ANDI,R0 H'F0'   MASK FOR MSD
0638 1A79 450F            ANDI,R1 H'0F'   MASK FOR LSD
0639 1A7B BBF6            ZBSR    *ROT    ROTATE A NIBBLE
0640 1A7D 17              RETC,UN
```

```
LINE ADDR  OBJECT  E SOURCE

0642                 ************************************************************
0643                 *
0644                 *
0645                 *DISPLAY AND ALTER REGISTERS COMMAND
0646                 *
0647                 *THE DISPLAY AND ALTER REGISTERS COMMAND ALLOWS
0648                 *THE USER TO EXAMINE AND ALTER R0,R1,R2,R3,R1',R2',R3',PSU,PSL,PC
0649                 *
0650                 *THIS COMMAND ALSO PROVIDES ENTRY POINT TO ALTERNATE FUNCTIONS
0651                 *REG 9 NOT DEFINED
0652                 *REG A ADJUST CASSETTE COMMAND
0653                 *REG B NOT DEFINED
0654                 *REG D NOT DEFINED
0655                 *REG E NOT DEFINED
0656                 *REG F ENTER THE FAST PATCH MODE
0657                 *
0658                 *REGISTERS USED
0659                 *
0660                 *R0 SCRATCH
0661                 *R1 SCRATCH
0662                 *R2 SCRATCH
0663                 *R3 SCRATCH
0664                 *
0665                 *SUBROUTINES CALLED
0666                 *
0667                 *MOV   MOVE DATA TO DISBUF
0668                 *GNP   GET NUMERIC PARAMETERS
0669                 *ROT   ROTATE A NIBBLE
0670                 *GNPA  DISPLAY AND GET NUMERIC PARAMETERS
0671                 *BRKPT4        SET DISPLAY 6&7
0672                 *SCBP2 SET DISPLAY 4&5
0673                 *
0674                 *RAM MEMORY USED
0675                 *
0676                 *DISBUF        DISPLAY BUFFER
0677                 *UREG  USER REGISTERS
0678                 *LADR  LAST ADDRESS REGISTER PC COUNTER
0679                 *T2  TEMP REGISTER
0680                 *
0681                 **************************************************************
0682 1A7E 051F       REG    LODI,R1 <REQ-1  GET ADDRESS OF R= DISPLAY
0683 1A80 06A4              LODI,R2 >REQ-1
0684 1A82 BBFE              ZBSR *MOV     MOVE DATA TO DISBUF
0685 1A84 20                EORZ    R0      SET FLAG TO RETURN AFTER KEY PRESSED
0686 1A85 BBEC              ZBSR    *DISPLY
0687                 *
0688 1A87 F480              TMI,R0  H'80'   SEE IF FUNCTION
0689 1A89 18B2              BCTR,EQ *REG14+1 MON2    GOTO MONITOR
0690 1A8B E409              COMI,R0 9       CHECK THE COMMAND
0691 1A8D 1E1AC2            BCTA,LT REG2    DISPLAY AND ALTER REGISTERS R0 THRU PSL
0692 1A90 E40A              COMI,R0 H'0A'   IS IT ADJUST CASSETTE COMMAND
0693 1A92 1C1F32            BCTA,EQ TCAS    TEST CASSETTE
0694 1A95 E40C              COMI,R0 H'0C'   IS IT DISPLAY AND ALTER PC
0695 1A97 1807              BCTR,EQ REG3    DISPLAY AND ALTER PC
0696 1A99 E40F              COMI,R0 H'0F'   IS IT THE PATCH COMMAND
```

```
LINE ADDR  OBJECT  E SOURCE

0697 1A9B 1C1A08            BCTA,EQ PTCH    DO THE PATCH COMMAND
0698 1A9E 1B5E              BCTR,UN REG     NOT DEFINED TRY AGAIN
0699                 *
0700                 *DISPLAY AND ALTER PROGRAM COUNTER
0701                 *
0702 1AA0 051F       REG3   LODI,R1 <PCEQ-1 GET ADDRESS OF PC EQUALS DISPLAY
0703 1AA2 06AC              LODI,R2 >PCEQ-1
0704 1AA4 BBFE              ZBSR    *MOV    MOVE DATA TO DISBUF
0705 1AA6 088E              LODR,R0 *REG4+1  GET CURRENT PC ADDRESS
0706 1AA8 BBEA              ZBSR  *BRKPT4  SET UP DISPLAY
0707 1AAA 0C17E8     REG11  LODA,R0 LADR    GET MSB OF CURRENT PC
0708 1AAD 3F1D8A            BSTA,UN SCBP2   SET UP DISPLAY
0709 1AB0 20                EORZ    R0      SET FLAG TO DOUBLE BYTE
0710 1AB1 BBFC              ZBSR *GNPA    DISPLAY ADDRESS AND WAIT FOR ENTRY
0711 1AB3 5B05              BRNR,R3 REG5    DON'T CHANGE DATA
0712 1AB5 CC17E9     REG4   STRA,R0 LADR+1  UP DATE THE PC SAVE LSB
0713 1AB8 C9F1              STRR,R1 *REG11+1 SAVE MSB OF PC
0714 1ABA E687       REG5   COMI,R2 H'87'   ENTR/NXT TERMINATION
0715 1ABC 9C187D     REG14  BCFA,EQ MON2    IF NOT NEW FUNCTION EXIT
0716 1ABF 1F1A7E            BCTA,UN REG     GO ASK FOR NEW REGISTER
0717                 *
0718                 *DISPLAY AND ALTER REGISTERS
0719                 *
0720 1AC2 CC17E6     REG2   STRA,R0 T2      SAVE IT
0721 1AC5 C3                STRZ    R3      SAVE R0 TO USE AS INDEX
0722 1AC6 0510              LODI,R1 H'10'   P CHAR
0723 1AC8 E707              COMI,R3 7       IS IT PSU
0724 1ACA 1A0A              BCTR,LT REG8    NOT PSU PSL
0725 1ACC 1904              BCTR,GT REG10   NOT PSU
0726 1ACE 0412              LODI,R0 H'12'   CHAR U
0727 1AD0 1B06              BCTR,UN REG12    GO DISPLAY
0728                 *
0729 1AD2 0411       REG10  LODI,R0 H'11'   CHAR L
0730 1AD4 1B02              BCTR,UN REG12    GO DISPLAY
0731                 *
0732 1AD6 0513       REG8   LODI,R1 H'13'      CHAR R
0733 1AD8 CD17D3     REG12  STRA,R1 DISBUF+2        SET DISPLAY RN=
0734 1ADB CC17D4     REG9   STRA,R0 DISBUF+3        SET UP DISPLAY
0735 1ADE 0F77F2            LODA,R0 UREG,R3 GET REGISTER CONTENT
0736 1AE1 BBEA              ZBSR  *BRKPT4  SET UP DISPLAY
0737 1AE3 0401              LODI,R0 1       SET FLAG TO SINGLE BYTE
0738 1AE5 BBFC              ZBSR *GNPA    DISPLAY REG CONTENT AND WAIT FOR ENTRY
0739 1AE7 5B0C              BRNR,R3 REG7    NO DATA TERMINATE
0740 1AE9 0BD8       REG6   LODR,R3 *REG2+1         GET THE INDEX VALUE
0741 1AEB CF77F2            STRA,R0 UREG,R3 PUT NEW VALUE IN REGISTER
0742 1AEE E708              COMI,R3 8       IS IT PSL?
0743 1AF0 9803              BCFR,EQ REG7    NO CHECK TERMINATION
0744 1AF2 CC17FC            STRA,R0 UREG+10 SAVE FOR RESTORE OF PSL
0745 1AF5 E687       REG7   COMI,R2 H'87'   CHECK TERMINATION
0746 1AF7 98C4              BCFR,EQ *REG14+1 MON2    NEW FUNCTION
0747 1AF9 03                LODZ    R3      INCREMENT INDEX VALUE
0748 1AFA D800              BIRR,R0 $+2     INCREMENT REGISTER COUNT
0749 1AFC E408              COMI,R0 8       ROLL OVER?
0750 1AFE 9D1AC2     REG13  BCFA,GT REG2    NO UP DATE DISPLAY
0751 1B01 20                EORZ    R0      GET A 0 GO TO R0
0752 1B02 1BFB              BCTR,UN *REG13+1    UPDATE DISPLAY
```

```
LINE ADDR  OBJECT  E SOURCE

0754                *********************************************************
0755                *
0756                *
0757                *GET NUMERIC PARAMETERS
0758                *
0759                *
0760                *THIS ROUTINE GETS EITHER 2 OR 4 DIGIT NUMERIC PARAMETERS
0761                *
0762                *INPUT PARAMETERS
0763                *
0764                *R0 CONTAINS INPUT PARAMETER
0765                *
0766                *BIT0 = 0 DOUBLE BYTE
0767                *BIT0 = 1 SINGLE BYTE DATA TO BE RETURNED
0768                *BIT1 = 0 REQUIRES FUNCTION KEY DEPRESSION TO EXIT
0769                *BIT1 = 1 WHEN SET WITH BIT0 EXIT IS AFTER ENTRY OF THIRD DIGIT
0770                *        OF SINGLE BYTE DATA
0771                *BIT2 = 1 WHEN SET WITH BIT0 EXIT IS AFTER SECOND DIGIT
0772                *        OF SINGLE BYTE DATA
0773                *
0774                *SINGLE BYTE DATA USES DISPLAY BUFFER 5 THRU 7
0775                *DOUBLE BYTE DATA USES DISPLAY BUFFER 4 THRU 7
0776                *OTHER DIGITS OF DISBUF MUST BE INITIALIZED ON ENTRY
0777                *
0778                *RETURNS WHEN FUNCTION KEY DEPRESSED
0779                *
0780                *OUTPUT PARAMETERS
0781                *
0782                *R0 = LSB OF DOUBLE BYTE DATA OR SINGLE BYTE DATA
0783                *R1 = MSB OF DOUBLE BYTE DATA OR 0 FOR SINGLE BYTE DATA
0784                *R2 = FUNCTION KEY PRESSED CODE
0785                *R3 = 0 DATA RETURNED IN R0(LSB), R1(MSB)
0786                *R3 = NOT 0 NO DATA RETURNED R0,R1 = 0
0787                *
0788                *REGISTERS USED
0789                *
0790                *R0 SCRATCH
0791                *R1 SCRATCH
0792                *R2 SCRATCH
0793                *R3 SCRATCH
0794                *
0795                *SUBROUTINES CALLED
0796                *
0797                *DISPLY       DISPLAY AND READ KEY BOARD
0798                *CLR BLANK DIGIT DISPLAY
0799                *
0800                *RAM MEMORY USED
0801                *
0802                *T1 SAVE ENTRY FLAG
0803                *DISPLY 4 THRU 7
0804                *
0805                *********************************************************
0806                *
0807                *
0808                *DISPLAY AD= AND GET DATA
```

```
LINE ADDR  OBJECT  E SOURCE

0809                 *
0810 1B04 051F       GAD    LODI,R1 <ADR-1  GET ADDRESS OF AD= DISPALY
0811 1B06 068C              LODI,R2 >ADR-1
0812 1B08 BBFE              ZBSR *MOV    MOVE DATA TO DISBUF
0813 1B0A 20                EORZ    R0      SET FLAG TO DOUBLE BYTE DATA
0814 1B0B 1B2E              BCTR,UN GNPI     GET THE ADDRESS DATA
0815                 *
0816                 *THIS ROUTINE CLEARS DIGIT DISPLAY
0817                 *
0818 1B0D 0517       CLR    LODI,R1 H'17'   BLANK SYMBOL
0819 1B0F F401              TMI,R0  1       SINGLE BYTE?
0820 1B11 1803              BCTR,EQ CLR1    ONE BYTE DATA
0821 1B13 CD17D5            STRA,R1 DISBUF+4        INITIALIZE DISPLAY TO BLANK
0822 1B16 CD17D6     CLR1   STRA,R1 DISBUF+5        GET HERE FOR ONE BYTE DATA
0823 1B19 CD17D7            STRA,R1 DISBUF+6
0824 1B1C CD17D8            STRA,R1 DISBUF+7
0825 1B1F 17                RETC,UN
0826                 *
0827                 *THIS ENTRY POINT ALLOWS DISPLAY OF DATA IN DISPLAY
0828                 *BUFFER 4 THRU 7
0829                 *
0830 1B20 C8A6       GNPAI  STRR,R0 *GNP12+1        SAVE INPUT FLAG IN T1
0831 1B22 0480              LODI,R0 H'80'   TURN ON DECIMAL POINT FOR ENTRY
0832 1B24 BBEC              ZBSR    *DISPLY DISPLAY MESSAGE AND READ KEY BOARD
0833 1B26 08A0              LODR,R0 *GNP12+1      GET INPUT PARAMETER
0834 1B28 F680              TMI,R2  H'80'   FUNCTION KEY?
0835 1B2A 9809              BCFR,EQ GNP13   FIRST CHAR IS COMMAND TERMINATE
0836 1B2C E687              COMI,R2 H'87'   ENTR/NXT?
0837 1B2E 1802              BCTR,EQ $+4
0838 1B30 3B5B              BSTR,UN CLR     CLEAR DISPLAY
0839 1B32 1F1B74            BCTA,UN GNP4
0840 1B35 F404       GNP13  TMI,R0  4       PATCH COMMAND RETURN SECOND DIGIT?
0841 1B37 3A54              BSTR,NG CLR     CLEAR DISPLAY
0842 1B39 1B0C              BCTR,UN GNP12
0843                 *
0844                 *THIS ENTRY POINT CLEARS DISPLAY AND WAITS FOR ENTRY
0845                 *
0846 1B3B C88B       GNPI   STRR,R0 *GNP12+1        SAVE INPUT FLAG IN T1
0847 1B3D 3B4E       GNP11  BSTR,UN CLR     CLEAR DISPLAY
0848 1B3F 0480       GNP2   LODI,R0 H'80'   TURN ON DECIMAL POINT FOR ENTRY
0849 1B41 BBEC              ZBSR *DISPLY  DISPLAY MESSAGE AND READ KEY BOARD
0850 1B43 F680       GNP5   TMI,R2  H'80'   FUNCTION KEY PRESSED?
0851 1B45 182D              BCTR,EQ GNP4    GO TERMINATE
0852                 *
0853                 *MOVE DISPLAY 1 DIGIT LEFT
0854                 *
0855 1B47 0C17E5     GNP12  LODA,R0 T1      GET INPUT PARAMETER
0856 1B4A F483              TMI,R0  H'83'   THIRD DIGIT *EXIT ON THIRD ENTRY*SINGLE BYTE
0857 1B4C 1826              BCTR,EQ GNP4    GO TERMINATE
0858 1B4E F425              TMI,R0  H'25'   2ND DIGIT*EXIT ON 2ND DIGIT*SINGLE BYTE
0859 1B50 1822              BCTR,EQ GNP4    GO TERMINATE
0860 1B52 F401              TMI,R0  1       SINGLE BYTE DATA?
0861 1B54 180B              BCTR,EQ GNP3    ONLY TWO DIGITS
0862 1B56 0D17D6     GN1    LODA,R1  DISBUF+5       GET DIGIT
0863 1B59 CD17D5     GN2    STRA,R1 DISBUF+4        SHIFT IT
0864 1B5C 0D17D7     GN3    LODA,R1 DISBUF+6        GET DIGIT
```

```
LINE ADDR  OBJECT  E SOURCE

0865 1B5F C9F6              STRR,R1 *GN1+1       SHIFT IT
0866 1B61 0D17D8     GNP3   LODA,R1 DISBUF+7       GET DIGIT
0867 1B64 C9F7              STRR,R1 *GN3+1       SHIFT IT
0868 1B66 CAFA              STRR,R2 *GNP3+1       ENTER NEW DIGIT
0869 1B68 08DE              LODR,R0 *GNP12+1     GET INPUT PARAMETER
0870 1B6A 7508              CPSL    WC      CLEAR WITH CARRY
0871 1B6C 8440              ADDI,R0 H'40'   SET BEEN HERE ONCE FLAG
0872 1B6E 6420              IORI,R0 H'20'   SET SECOND DIGIT FLAG
0873 1B70 C8D6       GN4    STRR,R0 *GNP12+1     RESTORE THE FLAG
0874 1B72 1B4B              BCTR,UN GNP2    GET NEXT ENTRY
0875                 *
0876                 *SET UP DATA TO BE RETURNED
0877                 *
0878 1B74 20         GNP4   EORZ    R0      GET A 0
0879 1B75 C1                STRZ    R1      CLEAR R1 DATA
0880 1B76 C3                STRZ    R3      CLEAR R3
0881 1B77 08CF              LODR,R0 *GNP12+1     GET INPUT PARAMETER
0882 1B79 F401              TMI,R0  1       CHECK FOR SINGLE BYTE
0883 1B7B 1812              BCTR,EQ GNP7    IF EQ ONLY 1 DIGIT
0884 1B7D 0C9B5A            LODA,R0 *GN2+1 DISBUF+4        GET MSD OF MSB
0885 1B80 E410              COMI,R0 H'10'   SEE IF HEX DIGIT
0886 1B82 9A03              BCFR,LT GNP6    IF NOT SKIP TO NEXT DIGIT
0887 1B84 3B1F              BSTR,UN ROTI     ROTATE NIBBLE
0888 1B86 C1                STRZ    R1      SAVE IN R1
0889 1B87 08CE       GNP6   LODR,R0 *GN1+1 DISBUF+5        GET LSD OF MSB
0890 1B89 E410              COMI,R0 H'10'   SEE IF HEX DIGIT
0891 1B8B 9A02              BCFR,LT GNP7    IF NOT SKIP TO NEXT DIGIT
0892 1B8D 61                IORZ    R1      INCLUSIVE OR MSD AND LSD OF MSB
0893 1B8E C1                STRZ    R1      SAVE IN R1
0894 1B8F 08CC       GNP7   LODR,R0 *GN3+1 DISBUF+6        GET MSD OF LSB
0895 1B91 E410              COMI,R0 H'10'   SEE IF HEX DIGIT
0896 1B93 9A03              BCFR,LT GNP8    IF NOT SKIP TO NEXT DIGIT
0897 1B95 3B0E              BSTR,UN ROTI     ROTATE THE NIBBLE
0898 1B97 C3                STRZ    R3      SAVE IN R3
0899 1B98 08C8       GNP8   LODR,R0 *GNP3+1 DISBUF+7        GET LSD OF LSB
0900 1B9A E410              COMI,R0 H'10'   SEE IF HEX DIGIT
0901 1B9C 9A04              BCFR,LT GNP9    IF NOT RETURN
0902 1B9E 63                IORZ    R3      INCLUSIVE OR MSD WITH LSD OF LSB
0903 1B9F 0700              LODI,R3 0    SET DATA IN R0,R1 FLAG
0904 1BA1 17                RETC,UN
0905 1BA2 077F       GNP9   LODI,R3 127     NO DATA
0906 1BA4 17                RETC,UN
0907                 *
0908                 *THIS ROUTINE ROTATES A NIBBLE 4 BITS LEFT
0909                 *
0910 1BA5 7508       ROTI    CPSL    WC      CLEAR WITH CARRY
0911 1BA7 D0                RRL,R0
0912 1BA8 D0                RRL,R0
0913 1BA9 D0                RRL,R0
0914 1BAA D0                RRL,R0
0915 1BAB 17                RETC,UN
```

```
LINE ADDR  OBJECT  E SOURCE

0917                 **********************************************************
0918                 *
0919                 *
0920                 *READ CASSETTE COMMAND
0921                 *
0922                 *
0923                 *
0924                 *THIS IS THE HEX OBJECT LOADER
0925                 *
0926                 *THIS ROUTINE REQUESTS A FILE ID AND THEN LOADS 2650 HEX OBJECT MODULES
0927                 *INTO MEMORY
0928                 *
0929                 *REGISTERS USED
0930                 *
0931                 *ALL
0932                 *
0933                 *SUBROUTINES CALLED
0934                 *
0935                 *IN CASSETTE INPUT ROUTINE
0936                 *MOV   MOVE DATA TO DISPLAY BUFFER
0937                 *GNP   GET NUMERIC PARAMETERS
0938                 *
0939                 ************************************************************************
0940                 *
0941                 *
0942 1BAC 051F       RCAS   LODI,R1 <FEQ-1  GET ADDRESS OF F= DISPLAY
0943 1BAE 06B4              LODI,R2 >FEQ-1
0944 1BB0 BBFE              ZBSR *MOV     MOVE DATA TO DISBUF
0945 1BB2 0401              LODI,R0 1       SET FLAG FOR SINGLE BYTE
0946 1BB4 BBFA              ZBSR   *GNP     GET THE FILE ID
0947 1BB6 180A              BCTR,EQ RCAS1   FILE ID SPECIFIED
0948 1BB8 E687              COMI,R2 H'87'   ENTR/NXT KEY?
0949 1BBA 988C              BCFR,EQ *RCAS4+1        GODO NEW FUNCTION
0950 1BBC 047F              LODI,R0 127     SET FILE ID FLAG TO FILE ID FOUND
0951 1BBE C8A4              STRR,R0 *RCAS5+1        STORE IN FILE ID FLAG
0952 1BC0 1B24              BCTR,UN LOAD
0953                 *
0954                 *
0955                 *FILE ID SPECIFIED
0956                 *
0957 1BC2 CC17E0     RCAS1  STRA,R0 FID+1   SAVE FILE ID
0958 1BC5 E687              COMI,R2 H'87'   ENTR/NXT KEY?
0959 1BC7 9C187D     RCAS4  BCFA,EQ MON2    GO DO NEW FUNCTION
0960 1BCA 20                EORZ   R0       SET FILE ID TO ID NOT FOUND
0961 1BCB C897              STRR,R0 *RCAS5+1        STORE IN FILE ID FLAG
0962 1BCD 75FD              CPSL   H'FD'      CLEAR PSL
0963 1BCF BBEE       RCAS2  ZBSR   *IN      LOOK FOR BEGINNING OF FILE
0964 1BD1 E416              COMI,R0 H'16'   BEGINNING OF FILE CHAR?
0965 1BD3 987A              BCFR,EQ RCAS2   LOOP TILL FIND BEGIN OF FILE
0966 1BD5 3F1C28            BSTA,UN BIN     GET THE FILE ID
0967 1BD8 E9E9              COMR,R1 *RCAS1+1   CHECK FILE ID FOR MATCH
0968 1BDA 1805              BCTR,EQ RCAS3   FOUND A MATCH
0969 1BDC 20                EORZ   R0       GET A 0
0970 1BDD C885              STRR,R0 *RCAS5+1 NO MATCH SAVE IN FID FLAG
0971 1BDF 1B05              BCTR,UN LOAD
```

```
LINE ADDR  OBJECT  E SOURCE

0972 1BE1 047F       RCAS3  LODI,R0 127     SET FLAG TO FILE IS MATCH
0973 1BE3 CC17DF     RCAS5  STRA,R0 FID     FILE ID FOUND
0974 1BE6 75FD       LOAD   CPSL    H'FD'     CLEAR PSL
0975 1BE8 BBEE              ZBSR    *IN     GET A CHAR
0976 1BEA E43A              COMI,R0 A':'    START OF LINE CHAR?
0977 1BEC 9878              BCFR,EQ LOAD    LOOP TILL FIND START FO RECORD
0978 1BEE 20                EORZ    R0      GET A 0
0979 1BEF CC17E1            STRA,R0 BCC     PRESET BCC
0980 1BF2 3B34              BSTR,UN BIN     INPUT A BYTE OF DATA
0981 1BF4 CD17DD            STRA,R1 MEM HI ADDR
0982 1BF7 3B2F              BSTR,UN BIN INPUT A BYTE OF DATA
0983 1BF9 CD17DE            STRA,R1 MEM+1 LO ADDR
0984 1BFC 3B2A              BSTR,UN BIN INPUT A BYTE OF DATA
0985 1BFE 01                LODZ    R1
0986 1BFF 1C1C42            BCTA,EQ LOAD1  GO TO START OF PROGRAM IF BYTE COUNT 0
0987 1C02 C3                STRZ    R3      SAVE BYTE COUNT
0988 1C03 08DF              LODR,R0 *LOAD-2 GET FILE ID FLAG
0989 1C05 185F              BCTR,EQ LOAD    FILE ID NOT FOUND SKIP TO END OF FILE
0990 1C07 3B1F              BSTR,UN BIN INPUT A BYTE OF DATA
0991 1C09 1804              BCTR,EQ BLOA    BCC OK READ THE RECORD
0992 1C0B 0704       BLOA1  LODI,R3 4       BCC ERROR
0993 1C0D 9BE8              ZBRR    *ERR    GOTO ERROR
0994 1C0F 3B17       BLOA   BSTR,UN BIN INPUT A BYTE OF DATA
0995 1C11 CD97DD            STRA,R1 *MEM STORE DATA IN MEMORY
0996 1C14 ED97DD            COMA,R1 *MEM    DO THE ERROR CHECK
0997 1C17 1804              BCTR,EQ BLOA2   DATA STORED OK
0998 1C19 0705              LODI,R3 5       READ CASSETTE MEMORY WRITE ERROR
0999 1C1B 9BE8              ZBRR    *ERR    GOTO ERROR
1000 1C1D 3B36       BLOA2  BSTR,UN INK INCREMENT POINTER MEM
1001 1C1F FB6E              BDRR,R3 BLOA LOOP TILL DONE
1002 1C21 3B05              BSTR,UN BIN INPUT A BYTE OF DATA
1003 1C23 9866              BCFR,EQ BLOA1 BCC ERROR
1004 1C25 1F1BE6            BCTA,UN LOAD
1005                 *
1006                 * INPUT A PAIR OF HEX ASCII CHAR
1007                 *CONVERT TO BINARY
1008                 *OUTPUT IS IN R1
1009                 *CALCULATE BCC ON DATA
1010                 *
1011 1C28 BBEE       BIN    ZBSR    *IN INPUT A CHAR
1012 1C2A 7509              CPSL    C+WC    CLEAR CARRY AND WITH CARRY
1013 1C2C 3B36       BIN1   BSTR,UN AHO3    LOOK UP VALUE
1014 1C2E 02                LODZ    R2      PUT VALUE IN R0
1015 1C2F BBF6              ZBSR    *ROT    ROTATE VALUE
1016 1C31 C1                STRZ    R1      SAVE VALUE IN R1
1017 1C32 BBEE              ZBSR    *IN     GET A CHAR
1018 1C34 7509              CPSL    C+WC    CLEAR CARRY AND WITH CARRY
1019 1C36 3B2C              BSTR,UN AHO3    LOOK UP VALUE
1020 1C38 01                LODZ    R1      GET SAVED VALUE
1021 1C39 62                IORZ    R2      MAKE THE BINARY BYTE
1022                 *
1023                 *CALCULATE BCC
1024                 *
1025 1C3A C1         CBCC   STRZ    R1      SAVE VALUE
1026 1C3B 2C17E1            EORA,R0 BCC     XOR WITH CURRENT BCC
1027 1C3E D0                RRL,R0          ROTATE LEFT
```

```
LINE ADDR  OBJECT  E SOURCE

1028 1C3F C8FB            STRR,R0 *CBCC+2 UPDATE THE BCC
1029 1C41 17              RETC,UN
1030                *
1031                *
1032                *FINISHED READING FILE
1033                *
1034 1C42 0C17DF    LOAD1 LODA,R0 FID     CHECK FILE ID FLAG FOR FILE ID FOUND
1035 1C45 1C1BCF          BCTA,EQ RCAS2   NO LOOK FOR START OF NEXT FILE
1036 1C48 088F            LODR,R0 *INK2+1 GET VALUE FROM MEM     PLACE START ADDRESS IN PC
1037 1C4A CC17E8          STRA,R0 LADR
1038 1C4D 0887            LODR,R0 *INK+1  GET VALUE FROM MEM+1
1039 1C4F CC17E9          STRA,R0 LADR+1
1040 1C52 1F1874          BCTA,UN MON3     GO TO THE MONITOR
1041                *
1042                *INCREMENT ADDRESS MEM
1043                *
1044 1C55 0C17DE    INK   LODA,R0 MEM+1   GET ADDRESS
1045 1C58 0E17DD    INK2  LODA,R2 MEM
1046 1C5B D802            BIRR,R0 INK1    INCREMENT IT
1047 1C5D DA00            BIRR,R2 INK1
1048 1C5F C8F5      INK1  STRR,R0 *INK+1   SAVE  IN MEM+1
1049 1C61 CAF6            STRR,R2 *INK2+1 SAVE IN MEM
1050 1C63 17              RETC,UN
1051                *
1052                *LOOK UP ASCII HEX TO CONVERT TO BINARY
1053                *
1054 1C64 06FF      AHO3  LODI,R2 255     PRESET INDEX
1055 1C66 EE3FC5          COMA,R0 ASCII,R2,+      CHECK THE VALUE
1056 1C69 14              RETC,EQ RETURN IF EQUAL
1057 1C6A E610            COMI,R2 H'10'   CHECK FOR MAX COUNT
1058 1C6C 9878            BCFR,EQ AHO3+2  LOOP
1059 1C6E 0706            LODI,R3 6       CHAR NOT ASCII HEX
1060 1C70 9BE8            ZBRR    *ERR    GOTO ERROR
1061                *
1062                *CARRAGE RETURN AND LINE FEED
1063                *
1064 1C72 040D      CRLFF LODI,R0 13      CARRAGE RETURN
1065 1C74 BBF0            ZBSR *OUT       PRINT
1066 1C76 040A            LODI,R0 10      LINE FEED
1067 1C78 BBF0            ZBSR *OUT       PRINT
1068 1C7A 17              RETC,UN
1069                *
1070                *CONVERT BINARY TO ASCII HEX AND PRINT
1071                *
1072 1C7B 7508      HOUTT  CPSL    WC
1073 1C7D BBF4            ZBSR    *DISLSD CONVERT BIN TO NIBBLE
1074 1C7F C2              STRZ    R2      SAVE IN R2
1075 1C80 0E7FC5          LODA,R0 ASCII,R2        TENS DIGIT
1076 1C83 BBF0            ZBSR    *OUT    PRINT TENS DIGIT
1077 1C85 0D7FC5          LODA,R0 ASCII,R1        GET UNITS DIGIT
1078 1C88 BBF0            ZBSR *OUT       PRINT UNITS DIGIT
1079 1C8A 17              RETC,UN
```

```
LINE ADDR  OBJECT  E SOURCE

1081                 ***********************************************************
1082                 *
1083                 *
1084                 *WRITE CASSETTE COMMAND
1085                 *
1086                 *THIS ROUTINE WRITES 2650 HEX FORMAT TO CASSETTE TAPE
1087                 *
1088                 *REGISTERS USED
1089                 *
1090                 *R0 SCRATCH
1091                 *R1 SCRATCH
1092                 *R2 SCRATCH
1093                 *R3 SCRATCH
1094                 *
1095                 *SUBROUTINES CALLED
1096                 *
1097                 *OUT WRITE CHAR TO TAPE
1098                 *HOUT CONVERT BINARY TO ASCII HEX AND WRITE TO TAPE
1099                 *INK INCREMENT POINTER MEM
1100                 *
1101                 *RAM USED
1102                 *
1103                 *BCC   BLOCK CHECK CHAR
1104                 *MEM   POINTER
1105                 *BAD   PROGRAM START ADDRESS
1106                 *SAD   DUMP STOP ADDRESS
1107                 *FID   FILE ID FLAG AND STORAGE
1108                 *
1109                 *THIS ROUTINE PUNCHES A HEX FORMAT TAPE
1110                 *
1111                 *
1112                 *   LEADER16ID:ADDRCTBCAADDCCRR.......BC
1113                 *
1114                 *********************************************************************
1115                 *
1116                 *
1117 1C8B 20         WCAS4  EORZ    R0      GET A 0
1118 1C8C BBFA              ZBSR    *GNP    GET NUMBER
1119 1C8E E687              COMI,R2 H'87'   ENTR/NXT KEY
1120 1C90 17                RETC,UN
1121                 *
1122                 *
1123 1C91 051F       WCAS   LODI,R1 <LADEQ-1        GET ADDRESS OF LAD= DISPLAY
1124 1C93 06BC              LODI,R2 >LADEQ-1
1125 1C95 BBFE              ZBSR    *MOV    MOVE TO DISPLAY BUFFER
1126 1C97 3B72              BSTR,UN WCAS4   GET ADDRESS DATA
1127 1C99 988E              BCFR,EQ *WCAS6+1 MON2    IF NOT EXIT
1128 1C9B CD17DD            STRA,R1 MEM     SAVE START ADDRESS
1129 1C9E CC17DE            STRA,R0 MEM+1
1130 1CA1 0412              LODI,R0 H'12'   CHANGE DISPLAY
1131 1CA3 CC17D1            STRA,R0 DISBUF  DISPLAY 'UAD=    '
1132 1CA6 3B63              BSTR,UN WCAS4   GET ADDRESS DATA
1133 1CA8 9C187D     WCAS6  BCFA,EQ MON2    NOT ENTR/NXT MUST BE NEW COMMAND
1134                 *
1135                 *CHECK FOR START ADDRESS GT THAN STOP
```

```
LINE ADDR  OBJECT  E SOURCE

1136                 *
1137 1CAB ED17DD            COMA,R1 MEM     CHECK HI BYTE
1138 1CAE 1806              BCTR,EQ WCAS7
1139 1CB0 1909              BCTR,GT WCAS9
1140 1CB2 0707       WCAS8  LODI,R3 7       SET THE ERROR NUMBER
1141 1CB4 9BE8              ZBRR    *ERR    GOTO ERROR
1142 1CB6 EC17DE     WCAS7  COMA,R0 MEM+1   CHECK LO BYTE
1143 1CB9 1A77              BCTR,LT WCAS8
1144                 *
1145 1CBB D802       WCAS9  BIRR,R0 WCASA   INCREMENT STOP ADDRESS
1146 1CBD D900              BIRR,R1 WCASA   SO DUMP IS INCLUSIVE
1147 1CBF CD17C8     WCASA  STRA,R1 EAD     SAVE END ADDRESS
1148 1CC2 CC17C9            STRA,R0 EAD+1
1149 1CC5 0405              LODI,R0 H'05'   CHANGE DISPLAY
1150 1CC7 CC17D1            STRA,R0 DISBUF  DISPLAY 'SAD=    '
1151 1CCA 3F1C8B            BSTA,UN WCAS4   GET PROGRAM START ADDRESS
1152 1CCD 9C9CA9     WCAS3  BCFA,EQ *WCAS6+1 MON2   GOTO MONITOR NEW FUNCTION
1153 1CD0 CD17CA            STRA,R1 BAD     SAVE START ADDRESS
1154 1CD3 CC17CB            STRA,R0 BAD+1
1155 1CD6 051F              LODI,R1 <FEQ-1  GET ADDRESS OF F= DISPLAY
1156 1CD8 06B4              LODI,R2 >FEQ-1
1157 1CDA BBFE              ZBSR *MOV     MOVE DATA TO DISBUF
1158 1CDC 0401              LODI,R0 1       SET FLAG TO SINGLE BYTE
1159 1CDE BBFA              ZBSR *GNP     GET THE FILE ID
1160 1CE0 E687              COMI,R2 H'87'   ENTR/NXT KEY
1161 1CE2 98C5              BCFR,EQ *WCAS6+1 MON2    EXIT NEW COMMAND
1162 1CE4 C894              STRR,R0 *WCAS5+1   SAVE FILE ID
1163 1CE6 060A              LODI,R2 10      SET THE DELAY
1164 1CE8 0719       PUN10  LODI,R3 25
1165 1CEA 20                 EORZ    R0      GET A 0
1166 1CEB BBF0              ZBSR    *OUT    OUTPUT A LEADER
1167 1CED FB7B              BDRR,R3 PUN10+2
1168 1CEF 12                SPSU        GET FLAG
1169 1CF0 2440              EORI,R0 H'40'   COMPLEMENT IT
1170 1CF2 92                LPSU        RESTORE IT
1171 1CF3 FA73              BDRR,R2 PUN10    DECREASE THE COUNT
1172 1CF5 0416              LODI,R0 H'16' START OF FILE CHAR
1173 1CF7 BBF0              ZBSR *OUT       PRINT
1174 1CF9 0C17E0     WCAS5  LODA,R0 FID+1   GET FILE ID
1175 1CFC BBF2              ZBSR *HOUT      CONVERT TO ASCII HEX AND PRINT
1176 1CFE BBF8       PUN2   ZBSR    *CRLF   OUTPUT CARRAGE RETURN AND LINE FEED
1177 1D00 043A              LODI,R0 A':'    START OF BLOCK CHAR
1178 1D02 BBF0              ZBSR    *OUT    PRINT
1179 1D04 20                EORZ    R0      GET A 0
1180 1D05 C8A3              STRR,R0 *PUN3+1     PRESET BCC
1181 1D07 0C17C8            LODA,R0 EAD CALCULATE NO OF BYTES TO OUTPUT
1182 1D0A 7709              PPSL    WC+C    SET CARRY AND WITH CARRY
1183 1D0C 0F17C9            LODA,R3 EAD+1   GET END ADDRESS
1184 1D0F ABAC              SUBR,R3 *BDUM1+1 MEM+1  SUBTRACT START ADDRESS FROM STOP ADDRESS
1185 1D11 A8A5              SUBR,R0 *BDUM+1 MEM
1186 1D13 7508              CPSL    WC      CLEAR WITH CARRY
1187 1D15 1E1CB2            BCTA,NG WCAS8   START > STOP
1188                 *
1189                 *
1190 1D18 581B       PUN4   BRNR,R0 ADUM    START ADDRESS GT THAN 256 AWAY FROM STOP
1191 1D1A 5B15              BRNR,R3 GDUM    START ADDRESS LT 256 AWAY FROM STOP
```

```
LINE ADDR  OBJECT  E SOURCE

1192 1D1C 0C17CA           LODA,R0 BAD THIS IS END OF FILE BLOCK
1193 1D1F 3B39             BSTR,UN EDUM SO OUTPUT START ADDRESS OF PROGRAM
1194 1D21 0C17CB           LODA,R0 BAD+1
1195 1D24 3B34             BSTR,UN EDUM    OUTPUT A BYTE AS 2 ASCII HEX CHARS
1196 1D26 20               EORZ    R0      END OF FILE BLOCK
1197 1D27 3B31             BSTR,UN EDUM OUTPUT BYTE COUNT
1198 1D29 0C17E1     PUN3  LODA,R0 BCC     GET BCC
1199 1D2C 3B2C             BSTR,UN EDUM OUTPUT BCC
1200 1D2E 1F1874           BCTA,UN       MON3      GOTO MONITOR
1201                 *
1202                 *
1203 1D31 E71E       GDUM  COMI,R3 H'1E'   IS START LT 30 AWAY FROM STOP
1204 1D33 1A02             BCTR,LT BDUM    OUTPUT LAST BYTES
1205 1D35 071E       ADUM  LODI,R3 H'1E'   NO OF BYTES THIS RECORD IS 30
1206 1D37 0C17DD     BDUM  LODA,R0 MEM     OUT ADDR HI
1207 1D3A 3B1E             BSTR,UN EDUM    OUTPUT BYTE AS 2 ASCII HEX CHARS
1208 1D3C 0C17DE     BDUM1 LODA,R0 MEM+1   OUT ADDR LO
1209 1D3F 3B19             BSTR,UN EDUM    OUTPUT BYTE AS 2 ASCII HEX CHARS
1210 1D41 03               LODZ    R3      OUT BYTE COUNT
1211 1D42 3B16             BSTR,UN EDUM    OUTPUT BYTE AS 2 ASCII HEX CHARS
1212 1D44 08E4             LODR,R0 *PUN3+1     OUT BCC FOR ADDR AND BYTE COUNT
1213 1D46 3B12             BSTR,UN EDUM    OUTPUT BYTE AS 2 ASCII HEX CHARS
1214 1D48 0C97DD     DDUM  LODA,R0 *MEM    OUTPUT DATA FROM MEM
1215 1D4B 3B0D             BSTR,UN EDUM    OUTPUT BYTE AS 2 ASCII HEX CHARS
1216 1D4D 3F1C55           BSTA,UN INK     INCREMEMT POINTER MEM
1217 1D50 FB76             BDRR,R3 DDUM    LOOP TILL DONE
1218 1D52 0C17E1           LODA,R0 BCC     GET BCC
1219 1D55 3B03             BSTR,UN EDUM    OUTPUT BCC FOR DATA
1220 1D57 1F1CFE           BCTA,UN PUN2
1221                 *
1222 1D5A 3F1C3A     EDUM  BSTA,UN CBCC    CALCULATE BCC
1223 1D5D 01               LODZ    R1      GET VALUE TO OUTPUT
1224 1D5E BBF2             ZBSR    *HOUT   PRINT AS 2 ASCII HEX CHARS
1225 1D60 17               RETC,UN
```

```
LINE ADDR  OBJECT  E SOURCE

1227                 *******************************************************
1228                 *
1229                 *
1230                 *SET OR CLEAR BREAK POINT
1231                 *
1232                 *
1233                 *TO SET BREAK POINT ENTR ADDRESS AND DEPRESS FUNCTION KEY
1234                 *TO CLEAR BREAK POINT DEPRESS FUNCTION KEY
1235                 *
1236                 *SUBROUTINES CALLED
1237                 *
1238                 *MOV MOVE DATA TO DISBUF
1239                 *GNPA DISPLAY AND GET ADDRESS DATA
1240                 *ROT ROTATE A NIBBLE
1241                 *SCBP2 SET DISBUF 4&5
1242                 *BRKPT4 SET DISBUF 6&7
1243                 *DSLSD CONVERT TO BINARY FOR DISPLAY
1244                 *
1245                 *RAM MEMORY USED
1246                 *
1247                 *BPF BREAK POINT FLAG
1248                 *BPL LOCATION OF BREAK POINT
1249                 *BPD DATA TO BE RESTORED IN BREAK POINT LOCATION
1250                 *
1251                 *
1252                 *
1253                 ****************************************************************
1254                 *
1255 1D61 051F       SCBP   LODI,R1 <BPEQ-1  GET ADDRESS OF BP= DISPALY
1256 1D63 069C              LODI,R2 >BPEQ-1
1257 1D65 BBFE              ZBSR *MOV     MOVE DATA TO DISBUF
1258 1D67 0C17CF            LODA,R0 BPF     BREAK POINT SET?
1259 1D6A 180A              BCTR,EQ SCBP1   NOT SET GET ADDRESS
1260                 *
1261                 *BREAK POINT SET SET UP ADDRESS DISPLAY
1262                 *
1263 1D6C 0C17CE            LODA,R0 BPL+1   PREPARE THE ADDRESS
1264 1D6F BBEA              ZBSR  *BRKPT4  SET UP DISPLAY
1265 1D71 0C17CD            LODA,R0 BPL     GET MSB
1266 1D74 3B14              BSTR,UN SCBP2   SETUP DISPLAY
1267 1D76 20         SCBP1  EORZ    R0      SET UP GET NUMBER PARAMETER TO 4 DIGIT
1268 1D77 BBFC              ZBSR *GNPA    GET THE ADDESS IF ANY
1269 1D79 1818              BCTR,EQ SCBP4   SET THE BREAK POINT
1270                 *
1271                 *THIS SECTION CLEARS THE BREAK POINT
1272                 *
1273 1D7B 0B88              LODR,R3 *SCBP6+1     CHECK BREAK POINT FLAG
1274 1D7D 1989              BCTR,EQ *SCBP5+1    BREAK POINT NOT SET GO TO MONITOR
1275 1D7F E681              COMI,R2 H'81'   IS TERMINATION BKP?
1276 1D81 9885              BCFR,EQ *SCBP5+1    NO LEAVE BREAK POINT SET GO TO MONITOR
1277 1D83 20                EORZ    R0      GET A 0
1278 1D84 CC17CF     SCBP6  STRA,R0 BPF     CLEAR BREAK POINT FLAG
1279 1D87 1F167D     SCBP5  BCTA,UN MON2    GO TO MONITOR
1280                 *
1281 1D8A BBF4       SCBP2  ZBSR    *DISLSD CONVERT TO BIN FOR DISPLAY
```

```
LINE ADDR  OBJECT  E SOURCE

1282 1D8C CD17D6            STRA,R1 DISBUF+5
1283 1D8F CC17D5            STRA,R0 DISBUF+4
1284 1D92 17                RETC,UN
1285                 *
1286                 *THIS SECTION SETS THE BREAK POINT
1287                 *
1288 1D93 CC17CE     SCBP4  STRA,R0 BPL+1   SET BREAK POINT ADDRESS
1289 1D96 CD17CD            STRA,R1 BPL
1290 1D99 20                EORZ    R0      CLEAR BREAK POINT FLAG
1291 1D9A C8E9              STRR,R0 *SCBP6+1        CHECK THE BREAK POINT CAN BE SET
1292 1D9C 0C97CD            LODA,R0 *BPL    GET DATA FROM BREAK POINT LOCATION
1293 1D9F 05B0              LODI,R1 H'B0'   BREAK POINT INSTRUCTION...WRTC,R0
1294 1DA1 CD97CD            STRA,R1 *BPL    TRY TO SET BREAK POINT
1295 1DA4 ED97CD            COMA,R1 *BPL    DID IT SET OK?
1296 1DA7 1804              BCTR,EQ SCBP7   BREAK POINT CAN BE SET
1297 1DA9 0701              LODI,R3 1       CANT SET BREAK POINT ERROR
1298 1DAB 9BE8              ZBRR    *ERR    GOTO ERROR
1299 1DAD CC97CD     SCBP7  STRA,R0 *BPL    RESTORE USER DATA
1300 1DB0 047F              LODI,R0 127     SET THE BREAK POINT FLAG
1301 1DB2 C8D1              STRR,R0 *SCBP6+1        SET IT
1302 1DB4 1BD2              BCTR,UN *SCBP5+1    GOTO MONITOR
```

```
LINE ADDR  OBJECT  E SOURCE

                *******************************************************
                *
                *
                *MOVE 8 BYTES OF DATA POINTED TO IN R1 AND R2 TO DISBUF
                *
                *
                *REGISTERS USED
                *
                *R0 SCRATCH
                *R1 HI ADDRESS BYTE OF DATA ADDRESS-1
                *R2 LO ADDRESS BYTE OF DATA ADDRESS-1
                *R3 NOT USED
                *
                *SUBROUTINES CALLED
                *
                *NONE
                *
                *RAM MEMORY USED
                *
                *T TEMP INDIRECT ADDRESS
                *
                **********************************************************
                *
1DB6 CD17E3     MOVI   STRA,R1 T       SET INDIRECT ADDRESS
1DB9 CE17E4            STRA,R2 T+1
1DBC 0608              LODI,R2 8       SET INDEX TO MOVE 8 BYTES
1DBE 0EF7E3     MOV1   LODA,R0 *T,R2   GET A BYTE
1DC1 CE77D0            STRA,R0 DISBUF-1,R2       MOVE TO BUFFER
1DC4 FA78              BDRR,R2 MOV1
1DC6 17                RETC,UN
```

```
LINE ADDR  OBJECT  E SOURCE

             ************************************************************
             *
             *
             *KEY BOARD SCAN AND DISPLAY ROUTINE
             *
             *THIS ROUTINE WRITTEN BY ALEX GOLDBURGER
             *
             *
             *TO USE THIS ROUTINE PLACE DATA TO BE DISPLAYED
             *IN DISBUF (SEE CODES AT BEGINNING OF PROGRAM)
             *
             *ON ENTRY R0 CONTAINS A FLAG
             *
             *R0 = 0 NORMAL OPERATION
             *      ON EXIT R0 = KEY PRESSED CODE
             *R0 = 1-127 GO THRU SCAN ONCE AND EXIT
             *          ON EXIT R0 = KEY PRESSED CODE
             *R0 = H'80' TURN ON DECIMAL POINT FOR ENTRY MODE
             *          ON EXIT R0 = KEY PRESSED CODE
             *
             *SEE KEY PRESSED CODES AT BEGINNING OF PROGRAM
             *
             *REGISTERS USED IN BANK ON ENTRY
             *
             *R0   SCRATCH
             *R1   KEYBOARD FLAGS
             *R2   DIGIT SELECT
             *R3   DIGIT POINTER
             *
             *SUBROUTINES CALLED
             *
             *NONE
             *
             *RAM MEMORY USED
             *
             *DISBUF       DISPLAY BUFFER
             *KFLG  KEY BOARD FLAG
             *
             ************************************************************
             *
1DC7 0406    DLOOP  LODI,R0 6       DELAY TO MAKE LOOPS EQUAL
1DC9 F87E           BDRR,R0 $
1DCB 52      DLOOP1 RRR,R2  ROTATE DIGIT SELECT
1DCC 0F77D0         LODA,R0 DISBUF-1,R3       GET DATA TO BE DISPLAYED
1DCF C1             STRZ    R1      SAVE DISPLAY CODE
1DD0 4580           ANDI,R1 H'80'   MASK FOR DECIMAL POINT
1DD2 447F           ANDI,R0 H'7F'   MASK OFF DECIMAL POINT
1DD4 0C7F68         LODA,R0 SEGTBL,R0       CONVERT TO SEGMENT DATA
1DD7 61             IORZ    R1      SET THE DECIMAL POINT IF NEEDED
1DD8 F601           TMI,R2  H'01'   COL 7?
1DDA 1A08           BCTR,NG DLOOP3  DONT PUT DECIMAL POINT HERE
1DDC 0D17ED         LODA,R1 KFLG+1  GET FLAG
1DDF 9A03           BCFR,NG DLOOP3  IF FLAG NOT NEG NO DECIMAL POINT
1DE1 4580           ANDI,R1 H'80'   MASK DECIMAL POINT
1DE3 61             IORZ    R1      SET DECIMAL POINT
```

```
LINE ADDR  OBJECT E SOURCE

1390 1DE4 0500       DLOOP3 LODI,R1 0       GET A 0
1391 1DE6 D5F9              WRTE,R1 SEG     TURN OFF SEGMENTS
1392 1DE8 D6FA              WRTE,R2 DIGIT   ENABLE NEXT DIGIT
1393 1DEA D4F9              WRTE,R0 SEG     AND DISPLAY IT
1394 1DEC 0C17EC            LODA,R0 KFLG    SEE IF KEY IS DOWN?
1395 1DEF 980B              BCFR,EQ DLOOP4  KEY UP DEBOUNCE
1396 1DF1 1B1A              BCTR,UN DLOOP5  IS KEY DOWN?
1397                 *
1398 1DF3 FB52       DLOOP2 BDRR,R3 DLOOP   DECREMENT DIGIT PTR
1399                 *                      TEST IF ONE SCAN IS DONE.
1400                 *                      IF ONE SCAN DONE INITIALIZE SCAN
1401                 *                      PARAMETERS AND KEY FLAGS
1402 1DF5 0C17ED            LODA,R0  KFLG+1 CHECK FOR ONE PASS THEN EXIT MODE
1403 1DF8 1933              BCTR,GT DISP3   IF ONE PASS EXIT
1404 1DFA 1B23              BCTR,UN DISP4   RESET THE FLAGS
1405                 *
1406                 *
1407 1DFC 3B28       DLOOP4 BSTR,UN GETKEY  GET A KEY
1408 1DFE 9806              BCFR,EQ DLP0    KEY IS DOWN RESET DEBOUNCE
1409 1E00 0887              LODR,R0 *DLP1+1 KFLG+2  GET COUNTER VALUE
1410 1E02 F804              BDRR,R0 DLP1
1411 1E04 1B14              BCTR,UN DISP1   SET FLAG TO ACCEPT KEY
1412 1E06 0460       DLP0   LODI,R0 H'60'   SET THE DELAY COUNT
1413 1E08 CC17EE     DLP1   STRA,R0 KFLG+2  SAVE DELAY COUNT
1414 1E0B 1B66              BCTR,UN DLOOP2  DO THE NEXT SCAN
1415                 *
1416                 *
1417 1E0D 3B17       DLOOP5 BSTR,UN GETKEY  IS A KEY DOWN?
1418 1E0F 1862              BCTR,EQ DLOOP2    NO
1419 1E11 1B24              BCTR,UN CODE
1420                 *
1421                 *ENTRY TO DISPLAY ROUTINE HERE
1422                 *
1423 1E13 CC17ED     DISPLI STRA,R0 KFLG+1  SAVE INPUT PARAMETER
1424 1E16 0460       DISP2  LODI,R0 H'60'   KEY WAS DOWN - SET KFLG
1425                 *                      NOT TO ACCEPT KEY NEXT SCAN
1426 1E18 C8EF              STRR,R0 *DLP1+1 KFLG+2  SET KEY DEBOUNCE DELAY
1427 1E1A CC17EC     DISP1  STRA,R0 KFLG    SAVE KFLG
1428 1E1D 7509              CPSL    C+WC    CLEAR CARRY AND WITH CARRY
1429 1E1F 0708       DISP4  LODI,R3 H'08'   INITIALIZE DIGIT POINTER
1430 1E21 0601              LODI,R2 H'01'   AND DIGIT SELECT
1431 1E23 1F1DCB            BCTA,UN DLOOP1  GO DISPLAY
1432                 *
1433                 *GET KEY CODE
1434                 *
1435 1E26 55FE       GETKEY REDE,R1 KBDIN   READ KEYBOARD
1436 1E28 450F              ANDI,R1 H'0F'   MASK OFF UNUSED BITS
1437 1E2A 250F              EORI,R1 H'0F'   INVERT THE INPUT
1438 1E2C 17                RETC,UN
1439                 *
1440                 *SINGLE PASS EXIT
1441                 *
1442 1E2D 040A       DISP3  LODI,R0 10
1443 1E2F F87E              BDRR,R0 $       DELAY
1444 1E31 D4F9              WRTE,R0 SEG     TURN OFF SEGMENTS
1445 1E33 0488              LODI,R0 H'88'   NO KEY PRESSED CODE
```

```
LINE ADDR  OBJECT  E SOURCE

1446 1E35 C2              STRZ    R2      SAVE IN R2
1447 1E36 17              RETC,UN
1448               *
1449               *CONVERT KEY LINE DATA TO KEY CODE
1450               *
1451 1E37 20       CODE   EORZ    R0      GET A 0
1452 1E38 D4F9            WRTE,R0 SEG     TURN OFF SEGMENTS
1453 1E3A A701            SUBI,R3 1       DECREMENT COLUMN COUNTER
1454 1E3C D4FA            WRTE,R0 DIGIT   TURN OFF COLUMNS
1455 1E3E 0604     CODE1  LODI,R2 4       LOOP COUNT
1456 1E40 51       CODE4  RRR,R1          GET WEIGHT OF KEY LINE
1457 1E41 E580            COMI,R1 H'80'   CHECK FOR 1 KEY DOWN
1458 1E43 1808            BCTR,EQ CODE2   R0 = 0,4,8, OR H'C'
1459 1E45 8404            ADDI,R0 H'04'
1460 1E47 FA77            BDRR,R2 CODE4   CHECK FOR ONLY 1 KEY
1461 1E49 0708            LODI,R3 8       MORE THAN 1 KEY DOWN OR NO KEY DOWN
1462 1E4B 9BE8            ZBRR    *ERR    GOTO ERROR
1463 1E4D E704     CODE2   COMI,R3 H'04'   NUMBER OR FUNCTION KEY?
1464 1E4F 1A05            BCTR,LT CODE3   # KEY
1465 1E51 50              RRR,R0          DIVIDE KEYLINE WEIGHT BY 2
1466 1E52 6480            IORI,R0 H'80'   FUNCTION KEY DESIGNATOR
1467 1E54 4701            ANDI,R3 H'01'   RETAIN LSB ONLY
1468 1E56 83       CODE3  ADDZ    R3      TO GET WHOLE KEYCODE
1469 1E57 C2              STRZ    R2      SAVE KEY CODE IN R2
1470 1E58 17              RETC,UN
```

```
LINE ADDR  OBJECT  E SOURCE

1472                 **************************************************************
1473                 *
1474                 *
1475                 *GOTO ROUTINE
1476                 *
1477                 *
1478                 *REGISTERS USED
1479                 *
1480                 *R0 SCRATCH
1481                 *R1 SCRATCH
1482                 *R2 SCRATCH
1483                 *R3 SCRATCH
1484                 *R1' RESTORED
1485                 *R2' RESTORED
1486                 *R3' RESTORED
1487                 *PSU RESTORED
1488                 *PSL RESTORED
1489                 *
1490                 *SUBROUTINES USED
1491                 *
1492                 *NONE
1493                 *
1494                 *RAM MEMORY USED
1495                 *
1496                 *SSF   SINGLE STEP FLAG
1497                 *BPF   BREAK POINT FLAG
1498                 *BPL   BREAK POINT LOCATION
1499                 *BPD   BREAK POINT DATA
1500                 *LADR INDIRECT ADDRESS TO JUMP THRU
1501                 *
1502                 **********************************************************
1503                 *
1504 1E59 0C17D0     GO     LODA,R0 SSF     GET SINGLE STEP FLAG
1505 1E5C 9819              BCFR,EQ GO1     NO SINGLE STEP GOTO USER
1506 1E5E 0C17CF            LODA,R0 BPF     GET BREAK POINT FLAG
1507 1E61 1814              BCTR,EQ GO1     BREAK POINT GO TO USER NO BREAK POINT
1508 1E63 0C97CD            LODA,R0 *BPL    GET USER DATA
1509 1E66 CC17CC            STRA,R0 BPD     SAVE USER DATA
1510 1E69 04B0              LODI,R0 H'B0'   WRTC,R0 BREAK POINT INSTRUCTION
1511 1E6B CC97CD            STRA,R0 *BPL    SET THE BREAK POINT
1512 1E6E EC97CD            COMA,R0 *BPL    CHECK BREAK POINT SET OK
1513 1E71 1804              BCTR,EQ GO1     GOTO USER
1514 1E73 0701              LODI,R3 1       ERROR BREAK POINT NOT SET OK
1515 1E75 9BE8              ZBRR    *ERR    GOTO ERROR
1516 1E77            GO1    EQU     $
```

```
LINE ADDR  OBJECT  E SOURCE

1518                ************************************************************
1519                *
1520                *
1521                *RESTORE REGISTERS BEFORE GOING TO USER PROGRAM
1522                *
1523                *
1524                *
1525                *
1526                *REGISTERS USED
1527                *
1528                *R0 THRU R3' PSU PSL
1529                *
1530                *SUBROUTINES CALLED
1531                *
1532                *UREG+9        RESTORE PSL
1533                *
1534                *RAM MEMORY USED
1535                *
1536                *UREG          = R0
1537                *UREG+1        = R1
1538                *UREG+2        = R2
1539                *UREG+3        = R3
1540                *UREG+4        = R1'
1541                *UREG+5        = R2'
1542                *UREG+6        = R3'
1543                *UREG+7        = PSU
1544                *UREG+8        = PSL
1545                *UREG+9        = PPSL INSTRUCTION OPCODE
1546                *UREG+10       = PSL
1547                *UREG+11       = RETC,UN       INSTRUCTION OPCODE
1548                *
1549                ****************************************************************
1550 1E77 0577      RESTRG LODI,R1 H'77'   PPSL INSTRUCTION OPCODE
1551 1E79 CD17FB           STRA,R1 UREG+9  CREATE A SUBROUTINE TO RESTORE PSL
1552 1E7C 0517             LODI,R1 H'17'   RETC,UN INSTRUCTION OPCODE
1553 1E7E CD17FD           STRA,R1 UREG+11
1554 1E81 7510             CPSL    RS      CLEAR REGISTER SWITCH
1555 1E83 0D17F3           LODA,R1 UREG+1  RESTORE R1
1556 1E86 0E17F4           LODA,R2 UREG+2  RESTORE R2
1557 1E89 0F17F5           LODA,R3 UREG+3  RESTORE R3
1558 1E8C 7710             PPSL    RS      SET THE REGISTER SWITCH
1559 1E8E 0D17F6           LODA,R1 UREG+4  RESTORE R1'
1560 1E91 0E17F7           LODA,R2 UREG+5  RESTORE R2'
1561 1E94 0F17F8           LODA,R3 UREG+6  RESTORE R3'
1562 1E97 0C17F9    RESTR1 LODA,R0 UREG+7  GET PSU DATA
1563 1E9A 6C17F1           IORA,R0 IFLG    SET INTERUPT INHIBIT IF REQUIRED
1564 1E9D 92               LPSU             RESTORE PSU
1565 1E9E 0C17F2           LODA,R0 UREG    RESTORE R0
1566 1EA1 75FF             CPSL    255      CLEAR PSL
1567 1EA3 3F17FB           BSTA,UN UREG+9  RESTORE PSL
1568 1EA6 1F97E8           BCTA,UN *LADR   GOTO USER
1569                *
```

```
LINE ADDR  OBJECT  E SOURCE

1571                 **********************************************************
1572                 *
1573                 *
1574                 *SUBROUTINE TO SAVE R1,R2,R3
1575                 *
1576                 *REGISTERS USED IN BANK ON ENTRY
1577                 *
1578                 *R1 SAVED IN SAVREG+1
1579                 *R2 SAVED IN SAVREG+2
1580                 *R3 SAVED IN SAVREG+3
1581                 *
1582                 *SUBROUTINES CALLED
1583                 *
1584                 *NONE
1585                 *
1586                 *RAM MEMORY USED
1587                 *
1588                 *SAVREG+1
1589                 *SAVREG+2
1590                 *SAVREG+3
1591                 ****************************************************
1592 1EA9 CD17DA     SAVR0  STRA,R1 SAVREG+1
1593 1EAC CE17DB     SAVR01 STRA,R2 SAVREG+2
1594 1EAF CF17DC     SAVR02 STRA,R3 SAVREG+3
1595 1EB2 17                RETC,UN
1596                 *************************************************************
1597                 *
1598                 *
1599                 *SUBROUTINE TO RESTORE R1,R2,R3
1600                 *
1601                 *
1602                 *REGISTERS USED IN BANK ON ENTRY
1603                 *
1604                 *R1 RESTORED TO VALUE IN SAVREG+1
1605                 *R2 RESTORED TO VALUE IN SAVREG+2
1606                 *R3 RESTORED TO VALUE IN SAVERG+3
1607                 *
1608                 *SUBROUTINES CALLED
1609                 *
1610                 *NONE
1611                 *
1612                 *RAM MEMORY USED
1613                 *
1614                 *SAVREG+1
1615                 *SAVREG+2
1616                 *SAVREG+3
1617                 *****************************************************
1618 1EB3 09F5       RESTR0 LODR,R1 *SAVR0+1
1619 1EB5 0AF6              LODR,R2 *SAVR01+1
1620 1EB7 0BF7              LODR,R3 *SAVR02+1
1621 1EB9 17                RETC,UN
```

```
LINE ADDR  OBJECT  E SOURCE

1623                 *******************************************************
1624                 *
1625                 *
1626                 *CASSETTE IO ROUTINES
1627                 *PROGRAM WRITTEN BY BBC
1628                 *
1629                 * 04-27-77
1630                 *
1631                 * THESE ROUTINES WRITES OR READS ONE BYTE TO OR FROM
1632                 * THE CASSETTE IN SIMCA FORMAT.
1633                 *
1634                 * THE FREQUENCY IS DETERMINED BY FREQ
1635                 * (CYCLE TIME IS 3.333 MICRO-SEC.)
1636                 *
1637                 *
1638                 *ROUTINES SAVE AND REESTORE R1,R2,R3 OF CURRENT BANK
1639                 *
1640                 *IN RETURNS WITH DATA BYTE IN R0
1641                 *OUT REQUIRES BYTE TO BE OUTPUT TO BE IN R0
1642                 *
1643                 *TCAS IS THE CASSETTE READ TEST USED TO SET LEVELS ON PLAY BACK
1644                 *
1645                 *SEE FRONT OF PROGRAM FOR DISPLAYS AND INSTRUCTIONS
1646                 *
1647                 *
1648                 *REGISTERS USED
1649                 *
1650                 *R0,R1,R2,R3 ARE SCRATCH
1651                 *
1652                 *SUBROUTINES CALLED
1653                 *
1654                 *SAVR0 SAVES R1,R2,R3
1655                 *RESTR0 RESTORES R1,R2,R3
1656                 *
1657                 *RAM MEMORY USED
1658                 *
1659                 *TEMP  TEMPORARY STORAGE
1660                 *
1661                 ***********************************************************************
1662 0011            FREQ   EQU     17      PULSE TIME ( 0.2 MSEC. )
1663 0088            SPDLY  EQU     8*FREQ  INTER-BIT SPACE
1664 0013            TMDLY  EQU     19      TIME-OUT FOR INTER-BIT DETECTION
1665 0003            PULS1  EQU     3       NUMBER OF PULSES FOR A ONE
1666 0006            PULS0  EQU     2*PULS1 NUMBER OF PULSES FOR A ZERO
1667 0009            THRES  EQU     3*PULS1 TRANSITION THRESHOLD FOR DETECTION
1668 000F            EBIT   EQU     5*PULS1 TRANSITION THRESHOLD FOR END BIT
1669                 *
1670                 *
1671                 * SUBROUTINE OUT
1672                 * WRITES ONE BYTE FROM R0 TO CASSETTE
1673                 *
1674 1EBA 3B6D       OUTT   BSTR,UN SAVR0   SAVE R1-R3
1675 1EBC D407              WRTE,R0 LEDS    WRITE BYTE TO LEDS FOR DISPLAY
1676 1EBE 0708              LODI,R3 8       BIT COUNT
1677 1EC0 C8A8       OUT1   STRR,R0 *OUT5+1 TEMP    SAVE BYTE IN TEMP
```

```
LINE ADDR  OBJECT  E SOURCE

1678 1EC2 CBAA              STRR,R3 *OUT6+1 TEMP+1  SAVE BIT COUNT IN TEMP+1
1679 1EC4 0506              LODI,R1 PULS0   GET NUMBER OF PULSES FOR A ZERO
1680 1EC6 F401              TMI,R0  H'01'   TEST FOR A ONE
1681 1EC8 9801              BCFR,0  OUT2
1682 1ECA 51                RRR,R1          DIVIDE COUNT IF A ONE
1683 1ECB FB02       OUT2   BDRR,R3 OUT3    CHECK FOR LAST BIT
1684 1ECD 8506              ADDI,R1 PULS0   YES, ADD LAST BIT PULSES
1685 1ECF 0611       OUT3   LODI,R2 FREQ    LENGTH OF PULSE
1686 1ED1 0718              LODI,R3 H'18'   SET ENV AND FREQ
1687 1ED3 D7F8              WRTE,R3 CAS
1688 1ED5 FA7E              BDRR,R2 $       DELAY 10 MICRO-SEC PER ITERATION
1689 1ED7 0611              LODI,R2 FREQ    LENGTH OF PULSE
1690 1ED9 0710              LODI,R3 H'10'   RESET FREQ
1691 1EDB D7F8              WRTE,R3 CAS
1692 1EDD FA7E              BDRR,R2 $       DELAY 10 MICRO-SEC PER ITERATION
1693 1EDF F96E              BDRR,R1 OUT3    DO NEXT PULSE
1694 1EE1 0688              LODI,R2 SPDLY   INTER-BIT SPACE
1695 1EE3 0700              LODI,R3 H'00'   TURN OFF ENV AND FREQ
1696 1EE5 D7F8              WRTE,R3 CAS
1697 1EE7 FA7E              BDRR,R2 $       DELAY 10 MICRO-SEC PER ITERATION
1698                 *
1699 1EE9 0C17C6     OUT5   LODA,R0 TEMP    GET CHARACTER BACK
1700 1EEC 50                RRR,R0          ROTATE RIGHT ONE PLACE
1701 1EED 0F17C7     OUT6   LODA,R3 TEMP+1  GET BIT COUNT
1702 1EF0 FB4E              BDRR,R3 OUT1    CONTINUE IF COUNT NON-ZERO
1703 1EF2 3F1EB3     OUT4   BSTA,UN RESTR0  RESTORE R1-R3
1704 1EF5 17                RETC,UN         ELSE, RETURN
1705                 *
1706                 * SUBROUTINE IN
1707                 * READS ONE BYTE FROM CASSETTE TO R0
1708                 *
1709 1EF6 3F1EA9     INN    BSTA,UN SAVR0   SAVE R1-R3
1710 1EF9 20                EORZ    R0      SET R0 TO ZERO
1711 1EFA 44FE       IN1    ANDI,R0 H'FE'   MASK OUT LOW BIT
1712 1EFC C8EC              STRR,R0 *OUT5+1 TEMP    SAVE PARTIAL BYTE
1713 1EFE 3B0A              BSTR,UN GBIT    GET NEXT BIT
1714 1F00 88E8              ADDR,R0 *OUT5+1 TEMP    ADD IN PARTIAL BYTE
1715 1F02 50                RRR,R0          MOVE NEW BIT TO HIGH POSITION
1716 1F03 5975              BRNR,R1 IN1     TEST LAST BIT FLAG
1717 1F05 3BEC              BSTR,UN *OUT4+1  YES, RESTORE R1-R3
1718 1F07 D407              WRTE,R0 LEDS    WRITE BYTE TO LEDS FOR DISPLAY
1719 1F09 17                RETC,UN         RETURN
1720                 *
1721                 * SUBROUTINE TO GET THE NEXT BIT FROM CASSETTE
1722                 * BIT IS RETURNED AS LEAST SIGNIFICANT BIT OF R0
1723                 *
1724 1F0A 0580       GBIT   LODI,R1 H'80'
1725 1F0C D5F8              WRTE,R1 CAS     SET SENSE TO CASSETTE
1726 1F0E 12                SPSU            GET PSU
1727 1F0F 07FF              LODI,R3 -1      SET TRANSITION COUNT TO -1
1728 1F11 06FF              LODI,R2 H'FF'   SET TIME-OUT TO MAX FOR FIRST TRANSITION
1729 1F13 1B02              BCTR,UN GBT3
1730 1F15 0613       GBT2   LODI,R2 TMDLY   SET END-OF-BIT DETECTION DELAY
1731 1F17 C1         GBT3   STRZ    R1      SAVE LAST COPY OF PSU IN R1
1732 1F18 8701              ADDI,R3 1       INCREMENT TRANSITION COUNTER
1733 1F1A 12         GBT4   SPSU            LOOK FOR TRANSITION
```

```
LINE ADDR  OBJECT E SOURCE

1734 1F1B E1               COMZ    R1
1735 1F1C 9877             BCFR,EQ GBT2    IF NOT EQUAL NEW TRANSITION
1736 1F1E FA7A             BDRR,R2 GBT4    IF EQUAL, TEST TIME-OUT
1737 1F20 20               EORZ    R0      SET R0 TO ZERO
1738 1F21 D4F8             WRTE,R0 CAS     SET SENSE BACK TO USER
1739 1F23 0501             LODI,R1 1       PRESET END FLAG TO 1
1740 1F25 E70F             COMI,R3 EBIT    ENDBIT THRESHOLD
1741 1F27 9903             BCFR,GT GBT5
1742 1F29 A70C             SUBI,R3 2*PULS0 LAST BIT, SUB ENDBIT PULSES
1743 1F2B C1               STRZ    R1      AND SET END FLAG
1744 1F2C E709      GBT5   COMI,R3 THRES   IS COUNT GREATER THAN THRESHOLD
1745 1F2E 15               RETC,GT         RETURN IF TRUE
1746 1F2F 0401             LODI,R0 1       NO, SET BIT TO ONE
1747 1F31 17               RETC,UN RETURN
1748                *
1749                *
1750                * SUBROUTINE TEST CASSETTE READS
1751                *
1752 1F32 0580      TCAS   LODI,R1 H'80'   SELECT LEAST SIGNIFICANT DIGIT
1753 1F34 D5FA             WRTE,R1 DIGIT
1754 1F36 0740      TCS0   LODI,R3 H'40'   OUTPUT '-' TO DISPLAY
1755 1F38 D407      TCS1   WRTE,R0 LEDS    OUTPUT VALUE TO LED'S
1756 1F3A D7F9             WRTE,R3 DISP    OUTPUT TO DISPLAY
1757 1F3C CF17C7    TCS10  STRA,R3 TEMP+1  SAVE UD CONDITION
1758 1F3F 060A             LODI,R2 10      RETURN AFTER 10 EXACT READS
1759 1F41 CE17C6    TCS2   STRA,R2 TEMP    SAVE R2
1760 1F44 3B44             BSTR,UN GBIT    GET A BIT
1761 1F46 0AFA             LODR,R2 *TCS2+1 TEMP    RESTORE R2
1762 1F48 050C             LODI,R1 2*PULS0 NUMBER OF TRANSITIONS FOR A ZERO
1763 1F4A 60               IORZ    R0      GET CONDITION CODE FOR R0
1764 1F4B 1801             BCTR,EQ TCS3    BRANCH IF A ZERO
1765 1F4D 51               RRR,R1          DIVIDE NOMINAL TRANSITION COUNT BY 2
1766 1F4E 03        TCS3   LODZ    R3      GET COUNT IN R0
1767 1F4F 1867             BCTR,EQ TCS1    BLANK DISPLAY IF 0
1768 1F51 A1               SUBZ    R1      TEST COUNT
1769 1F52 9804             BCFR,EQ TCS4    IF NOT EQUAL, RETURN
1770 1F54 FA6B      TCS35  BDRR,R2 TCS2    IF EQUAL AND COUNT NOT UP, GET NEW BIT
1771 1F56 1B5E             BCTR,UN TCS0
1772 1F58 190A      TCS4   BCTR,GT TCS5    DETERMINE POLARITY
1773 1F5A 08E1             LODR,R0 *TCS10+1        TEMP+1 GET UD CONDITION
1774 1F5C E4DE             COMI,R0 H'DE'   DOWN CONDITION
1775 1F5E 1874             BCTR,EQ TCS35    CANT GO DIRECT FROM DOWN TO UP
1776 1F60 073E             LODI,R3 H'3E'   OUTPUT 'U' TO DISPLAY
1777 1F62 1B54             BCTR,UN TCS1
1778 1F64 07DE      TCS5   LODI,R3 H'DE'   OUTPUT 'D' TO DISPLAY
1779 1F66 1B50             BCTR,UN TCS1
```

```
LINE ADDR  OBJECT  E SOURCE

1781                 *********************************************************************
1782                 *
1783                 *HEXTAB LOOKUP TABLE FOR HEX TO SEVEN SEGMENT
1784                 *
1785                 *THIS TABLE CONTAINS THE VALUES FOR LIGHTING THE
1786                 *SEGMENTS FOR THE DIGITS 0 THRU 9 AND LETTERS A TO F
1787                 *
1788 1F68 3F065B4F   SEGTBL DATA    H'3F,06,5B,4F,66,6D,7D,07,7F,67,77,FC,39,DE,79,71'
     1F6C 666D7D07
     1F70 7F6777FC
     1F74 39DE7971
1789                 *
1790                 *SEGMENT DATA FOR SYMBOLS P L U R H O = BLANK J - . Y N
1791                 *
1792 1F78 73383E50          DATA    H'73,38,3E,50,76,5C,48,00,0E,40,80,6E,54'
     1F7C 765C4800
     1F80 0E40806E
     1F84 54
1793                 *
1794                 *THIS TABLE CONTAINS THE DISPLAY ERROR
1795                 *
1796 1F85 170E1313   ERROR  DATA    H'17,0E,13,13,15,13,17,17'
     1F89 15131717
1797                 *
1798                 *THIS TABLE CONTAINS THE DISPLAY AD=
1799                 *
1800 1F8D 170A0D16   ADR    DATA    H'17,0A,0D,16,17,17,17,17'
     1F91 17171717
1801                 *
1802                 *THIS TABLE CONTAINS THE DISPLAY HELLO
1803                 *
1804 1F95 17140E11   HELLO  DATA    H'17,14,0E,11,11,00,17,17'
     1F99 11001717
1805                 *
1806                 *THIS TABLE CONTAINS THE DISPLAY BP=
1807                 *
1808 1F9D 170B1016   BPEQ   DATA    H'17,0B,10,16,17,17,17,17'
     1FA1 17171717
1809                 *
1810                 *THIS TABLE CONTAINS THE DISPLAY R=
1811                 *
1812 1FA5 17171317   REQ    DATA    H'17,17,13,17,16,17,17,17'
     1FA9 16171717
1813                 *
1814                 *THIS TABLE CONTAINS THE DISPLAY PC=
1815                 *
1816 1FAD 17100C16   PCEQ   DATA    H'17,10,0C,16,17,17,17,17'
     1FB1 17171717
1817                 *
1818                 *THIS TABLE CONTAINS THE DISPLAY F=
1819                 *
1820 1FB5 17170F16   FEQ    DATA    H'17,17,0F,16,17,17,17,17'
     1FB9 17171717
1821                 *
1822                 *THIS TABLE CONTAINS THE DISPLAY LAD=
LINE ADDR  OBJECT  E SOURCE

1823                 *
1824 1FBD 110A0D16   LADEQ  DATA    H'11,0A,0D,16,17,17,17,17'
     1FC1 17171717
1825                 *
1826                 *THIS TABLE IS THE ASCII LOOK UP TABLE
1827                 *
1828 1FC5 30313233   ASCII  DATA    A'0123456789ABCDEF'
     1FC9 34353637
     1FCD 38394142
     1FD1 43444546
1829                 *
```

```
LINE ADDR  OBJECT  E SOURCE

1831                 ***********************************************************
1832                 *
1833                 *
1834                 *USER ENTRY TO DISPLAY ROUTINES
1835                 *
1836                 *
1837 1FD5 BBFE       USRDSI ZBSR    *MOV    SET UP DISPLAY
1838 1FD7 03                LODZ    R3      GET DISPLAY FLAG
1839 1FD8 BBEC              ZBSR    *DISPLY GO TO DISPLAY ROUTINE
1840 1FDA 17                RETC,UN
```

```
LINE ADDR  OBJECT  E SOURCE

1842                 **********************************************************
1843 1FD8                   ORG 8192-26 THE ZBSR OR ZBRR VECTORS ARE HERE
1844                 ***********************************************************
1845 1FE6 1FD5       USRDSP ACON    USRDSI  USER ENTRY TO DISPLAY ROUTINES
1846 1FE8 1899       ERR    ACON    ERRI    ERROR MESSAGE
1847 1FEA 19E8       BRKPT4  ACON    BRKPTI SET DISBUF6,7 WITH CONTENTS OF R0
1848 1FEC 1E13       DISPLY ACON    DISPLI DISPLAY AND KEYBOARD ROUTINE
1849 1FEE 1EF6       IN     ACON    INN     CASSETTE INPUT ROUTINE
1850 1FF0 1EBA       OUT    ACON    OUTT    CASETTE OUT PUT
1851 1FF2 1C7B       HOUT   ACON    HOUTT   CASSETTE BINARY TO ASCII HEX OUTPUT
1852 1FF4 1A76       DISLSD ACON    DISLSI  CONVERT BYTE TO NIBBLE
1853 1FF6 1BA5       ROT    ACON    ROTI    ROTATE A  NIBBLE
1854 1FF8 1C72       CRLF   ACON    CRLFF   CARRAGE RETURN AND LINE FEED
1855 1FFA 1B3B       GNP    ACON    GNPI    GET NUMBERS
1856 1FFC 1B20       GNPA   ACON    GNPAI   GET NUMBERS AND DISPLAY
1857 1FFE 1DB6       MOV     ACON    MOVI    MOVE DATA TO DISBUF
1858                 ************************************************************
1859 1800                   END     SAVRG

 TOTAL ASSEMBLY ERRORS = 0000
```

12. CONVERSION TABLES

ASCII CONVERSION TABLE

ASCII CHARACTER SET (7-BIT CODE)									
M.S. CHAR L.S. CHAR		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P		p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	(	k	{
C	1100	FF	FS	.	<	L		l	
D	1101	CR	GS	–	=	M	)	m	}
E	1110	SO	RS	•	>	N	↑	n	~
F	1111	SI	US	/	?	O	← or _	o	DEL

DECIMAL TO HEX CONVERSION TABLE

HEXADECIMAL COLUMNS											
6		5		4		3		2		1	
HEX=DEC		HEX=DEC		HEX=DEC		HEX=DEC		HEX=DEC		HEX=DEC	
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

13. GLOSSARY

This chapter contains definitions of terms unique to microprocessors and microcomputers. Definitions used are common to the industry where such definitions exist. Other definitions were specially written for this glossary.

ABSOLUTE ADDRESS

The actual address of a memory location, as opposed to a relative address which may not be determined until it is used during the execution of an instruction.

ACCESS TIME

The time interval between the instant that data is called from or delivered to a storage device and the instant the requested retrieval or storage is complete.

Time between the instant that an address is sent to a memory and the instant that data returns. Since the access time to different locations (addresses) of the memory may be different, the access time specified in a memory device is the path that takes the longest time.

ACCUMULATOR

A register of the Arithmetic Logic Unit (ALU) of a central processor used as intermediate storage during the formation of algebraic sums, or for other intermediate logical and arithmetic operations.

Register and related circuitry that holds one operand for arithmetic and logical operations.

ADDRESS

A unique lable, name, or number that identifies a memory location or a device register for access by a computer.

A number used by the CPU to specify a location in memory.

ADDRESS FIELD

That portion of a computer word containing either the address of the operand or the information necessary to derive that address.

ALGORITHM

A prescribed set of well-defined rules or processes for the solution of a problem. Algorithms are implemented on a computer by a programmed sequence of instructions.

ADDRESSING MODES

See MEMORY ADDRESSING MODES.

ARRAY

A named group of related variables or constants. Also referred to as a table or list. An index is used in combination with the array name to access individual variables or constants in the array. Items in the array may be located in consecutive memory locations or they may be linked.

ARCHITECTURE

Organizational structure of a computing system mainly referring to the CPU or microprocessor.

ARGUMENT

The independent variable of a function. Arguments can be passed as part of a subroutine call where they would be used in that subroutine.

ASCII CODE

The acronym for American Standard Code for Information Interchange. This standardized code is used extensively in data transmission. The code includes 128 upper and lower case letters, numerals, and special-purpose symbols each encoded by a unique 7-bit binary number.

ASSEMBLER

A computer program which converts a symbolic assembly language program into an executable object (binary-coded) program. Depending on the assembler, the machine language program produced can be structured to occupy a set of locations in system memory by adding a given value (offset) to each assembled address.

ASSEMBLY LANGUAGE

A human oriented symbolic-mnemonic source language which is used by the programmer to encode programs and associated data bases. Assembly language programs are read by the assembler and converted to executable machine language programs during the assembly processes. Assembly language is easier to remember and manipulate than machine language.

BAUD RATE

Synonymous with signal events (bits)-per-second and used as a measure of serial data flow between a computer and/or communication devices.

BIDIRECTIONAL

Refers to a type of bus structure where a single conductor is used to transmit data or signals in either direction between a peripheral device and a central processor or memory.

BINARY NUMBER SYSTEM

A number system having 2 as its base and expressing all quantities by the numerals 0 and 1. As in the decimal system, the value of binary digits is positionally weighted from right to left by ascending powers of the base. The two-state character of the binary number system makes it especially suitable for the digital computer which operates most conveniently on a bistable basis.

BISTABLE LATCH

A rudimentary flip-flop which can be enabled to store a logical one or a logical zero. One bistable latch device is commonly used in memory and register circuits for the storage of each bit.

BIT

A minimum logic element. A binary number of either 0 or 1. A bit is the smallest unit of information in a binary system of notation. It is the choice between two possible states, usually designated one (1) and zero(0).

BIT PARALLEL

A method of simultaneously moving or transferring all bits in a contiguous set of bits over seperate wires, one wire for each bit in the set.

BIT SERIAL

Refers to a method of sequentially moving or transferring a contiguous set of bits one at a time over a single wire, according to a fixed sequence.

BLOCK

A group of consecutive words, characters, or bits which are handled as a single unit, particularly with respect to input-output operations.

BLOCK DIAGRAM

A chart which graphically depicts the functional relationships of hardware making up a system. The block diagram serves to indicate the various data and control signal paths between functional units of the system hardware.

BOOTSTRAP

A short sequence of instructions used for loading system memory with a larger, more sophisticated loader of system programs. Bootstrap programs are often hardwired in a reserved section of a computer system memory which, when entered, will operate a device such as a paper tape reader to load the system.

BRANCH

An instruction which, when executed, can cause the computer's arithmetic and control unit to obtain the next instruction to be executed from a location other than the next sequential location. A branch can be unconditional or conditional based on the magnitude or state of some value. Branch is synonymous with jump.

BREAKPOINT

A location in a program at which execution of that program can be halted to permit visual check, printing out, or other performance analysis.

BUS

A circuit or group of circuits which provide a communication path between two or more devices, such as between a central processor, memory, and peripherals.

BUS DRIVER

A circuit which amplifies a bus data or control signal sufficiently to assure valid receipt of that signal at the destination.

BYTE

A set of contiguous binary bits, usually eight, which are operated on as a unit. A byte can also be a subset of a computer word.

CARRY

In arithmetic operations, the transfer of a value from a lower order position to the next higher position as a result of the lower order digit having equaled or exceeded the base of the number system involved.

CENTRAL PROCESSING UNIT

A typical microprocessor incorporates a central processing unit (CPU), a memory and Input/Output ports. The CPU reads instructions from the program

memory in a logical determined sequence and uses them to initiate processor actions. To perform these functions, a CPU must incorporate registers, an arithmetic/logic unit (ALU) and central circuitry. Registers, temporary storage units inside the CPU, differ in their functional capabilities. Some, like the program counter and instruction registers, serve dedicated applications; others, like the accumulator, serve general-purpose functions.
An accumulator usually stores one of the operands manipulated by the ALU. A typical instruction could direct the ALU to add another register's contents to those of the accumulator and store the results in the accumulator. In general, the accumulator serves both as a source and destination register. In addition to the accumulator, the CPU may contain additional general-purpose registers that programmers can use to store source or intermediate "scratchpad" data.

CHARACTER
A letter, digit, or other symbol that is used as part of the organization, control, or representation of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes.

CLOCK
A pulse generator which generates basic timing signals to which all system operations are synchronized.

CONDITIONAL JUMP
An instruction causing a program transfer to an instruction location other than the next sequential instruction only if a specific condition tested by the instruction is satisfied. If the condition is not satisfied, the next sequential instruction in the program line is executed.

CONSTANT
A data item which does not change during program execution. Program constants can be symbolically defined in 2650 assembly language with the EQU, DATA and ACON statements.

CONTROL CHARACTER
A character whose occurrence in a particular context initiates, modifies, or halts operation.

CONTROL STATEMENT

A program statement (instruction) which is used to direct the flow of the program either causing an unconditional transfer or making a transfer dependent upon meeting a certain specified condition. Branch instructions in the 2650 are control statements.

CYCLE STEALING

A characteristic of Direct Memory Access (DMA) channels. An I/O device can delay CPU use of the I/O bus for one or more bus cycles while it accesses system memory.

CYCLE TIME

The time required by a computer to read from or write into the system memory. If system memory is core, the read cycle time includes a write after read (restore) subcycle. Cycle time is often used as a measure of computer performance, since this is a measure of the time required to fetch an instruction.

DATA BUFFER REGISTER

A temporary storage register in a CPU or peripheral device capable of receiving or transmitting data at different I/O rates. Data buffer registers are generally positioned between the computer and slower system components, allowing data to flow at the computer's I/O rate.

DEBUG PROGRAMS

Debug programs help the programmer to find errors in his programs while they are running on the computer, and allow him to replace or patch instructions into (or out of) his program.

DECODER

A logic device which converts data from one number system to another; e.g., an octal-to-decimal decoder. Decoders are also used to recognize unique addresses, such as a device address and bit patterns.

DECREMENT

To reduce the numerical contents of a counter. A decrement of one is usually assumed unless specified otherwise.

DEFERRED ADDRESSING

An indirect addressing mode in which the directly addressed location contains the address of the operand, rather than the operand itself.

DIAGNOSTIC PROGRAM(S)

A troubleshooting aid for locating hardware malfunctions in a system or a program to aid in locating coding errors in newly developed programs. These programs check the various hardware parts of a system for proper operation; CPU diagnostics check the CPU, memory diagnositcs check the memory, and so forth.

DIRECT ADDRESSING

1) An addressing mode in which the contents of the addressed location is the operand. 2) The address of an instruction or operand is completely specified in an instruction without reference to a base register or index register.

DIRECT MEMORY ACCESS (DMA)

A method of transferring blocks of data directly between an external device and system memory without the need for CPU intervention. This method significantly increases the data transfer rate, hence system efficiency. (see Cycle Stealing.)

DOUBLE OPERAND

An instruction type containing two address fields, source operand address field, and destination operand address field.

DOUBLE-PRECISION ARITHMETIC

Refers to a method of performing arithmetic operations in which two computer words are used to represent a single number, effectively doubling the data word size.

EDITOR

1) A program which permits a user to create new files in symbolic form or modify existing files. 2) As an aid in preparing source programs, certain programs have been developed that manipulate text material. These programs, called editors, text editors, or paper tape editors, make it possible to compose assembly language programs on-line, or on a stand-alone system.

EMULATOR

A program or a hardware device which duplicates the instruction set of one computer on a different computer, allowing program development for the emulated computer without that computer being available.

EXECUTE

To perform a specified computer instruction. To run a program.

FETCH

1) The action of obtaining an instruction from a stored program and decoding that instruction. Also refers to that portion of a computer's instruction cycle when that action is performed. 2) A process of addressing the memory and reading into the CPU the information word, or byte, stored at the addressed location. Most often, fetch refers to the reading out of an instruction from the memory.

FIFO

First In, First Out method of storing and retrieving items from a stack, table or list.

FILE

A collection of related records treated as a unit. In a computer system, a file can exist on cassette tape, disk, punched paper tape, punched cards, or as an accumulation of information in a system memory. A file can contain data, programs, or both.

FIRMWARE

See SOFTWARE.

FIXED-INSTRUCTION COMPUTER

(Stored-Instruction Computer): The instruction set of a computer is fixed by the manufacturer. The users will design application programs using this instruction set (in contrast to the micro-programmable computer for which the users may design their own instruction set and thus customize the computer for their needs.)

FIXED-POINT ARITHMETIC

Arithmetic in which the binary point that separates the integer and fractional portions of numerical expressions is either explicitly stated for all expressions or is fixed with respect to the first or last digit of each expression.

FLAG

An indicator, usually a single binary bit, whose state is used to inform a later section of a program that a condition, identified with the flag and designated by the state of the flag, has occurred. A flag can be both software and hardware implemented.

FLAG LINES

Inputs to a microprocessor controlled by I/O devices and tested by branch instructions.

FLOATING-POINT ARITHMETIC

Arithmetic in which the location of the decimal point for each number in an arithmetic operation is defined as a power of ten and all exponents are equalized prior to the operation. The major advantages of floating point arithmetic are that it extends the calculation capability of a computer beyond the limit imposed by the fixed word length and that it contributes to ease of programming.

FLOWCHART

A graphical representation of the processing steps performed by a computer program or of the sequence of logic operations implemented in hardware.

GENERAL REGISTER

One of a specified number of internal addressable registers in a CPU which can be used for temporary storage, as an accumulator, an index register, a stack pointer or for any other general-purpose function.

HANDSHAKING

Refers to the required sequence of signals for communication between system functions. The I/O bus protocol for a system defines its handshaking requirements. (This is especially true for asynchronous I/O systems in which each signal requires a response (reply) to complete an I/O operation).

HARD-WIRED LOGIC

A group of logic circuits permanently interconnected to perform a specific function--permanently assigned device address, memory bank assignments, and interrupt vector addresses.

HEXADECIMAL

A number system using 0, 1, ..., A, B, C, D, E, F to represent all the possible values of a 4-bit digit. The decimal equivalent is 0 to 15. Two hexadecimal digits can be used to specify a byte.

HIGH-LEVEL LANGUAGE

Programming language that generates machine codes from problem or function-oriented statements. FORTRAN, COBOL, and BASIC are three commonly used high-level languages. A single functional statement may translate into a series of instructions or subroutines in machine language, in contrast to a low-level(assembly) language in which statements translate on a one-for-one basis.

I/O HANDLERS

Input/Output handlers, sometimes called device drivers, are subroutines that service specific peripheral devices such as teletypewriters and card readers. They help prevent "reinvention of the wheel" every time a programmer wants to use a standard peripheral.

I/O PORT

A connection to a CPU that is configured (or programmed) to provide a data path between the CPU and external devices, such as a keyboard, display panel, audio cassette recorder, etc. An I/O port of a microprocessor may be an input or an output port, or it may be bidirectional.

IMMEDIATE ADDRESSING

The method of addressing an instruction in which the operand is located in the instruction itself or in the memory location immediately following the instruction.

IMMEDIATE DATA

Data that immediately follows an instruction in memory and is used as an operand by that instruction.

INDEX REGISTER

A register that contains a quantity which may be used to modify memory address.

INDEXED ADDRESSING

An addressing mode in which the address part of an instruction is modified by the contents in an auxiliary (index) register during the execution of that instruction.

INDIRECT ADDRESSING

A means of addressing in which the address of the operand is specified by an auxiliary register or memory location specified by the instruction rather than by bits in the instruction itself.

INPUT-OUTPUT (I/O)

General term for the equipment used to communicate with a computer CPU, or the data involved in that communication.

INSTRUCTION

A set of bits that defines a computer operation, and is a basic command understood by the CPU. It may move data, do arithmetic and logic functions, control I/O devices, or make decisions as to which instruction to execute next.

INSTRUCTION CYCLE

The process of fetching an instruction from memory and executing it.

INSTRUCTION LENGTH

The number of words needed to store an instruction. It is one word in most computers, but will use multiple words to form one instruction. Multiple-word instructions have different instruction execution times depending on the length of the instruction.

INSTRUCTION REPERTOIRE

See INSTRUCTION SET.

INSTRUCTION SET

The set of general-purpose instructions available with a given computer. In general, different machines have different instruction sets. The number of instructions only partially indicates the quality of an instruction set. Some instructions may only be slightly different from one another; others rarely may be used.

INSTRUCTION TIME

The time required to fetch an instruction from memory and then execute it.

INTERPRETER

A program that fetches and executes "instructions" (pseudo instructions) written in a higher-level language. The higher-level language program is a pseudo program. Contrast with compiler.

INTERRUPT

An interrupt involves the suspension of the normal programming routine of a microprocessor in order to handle a sudden request for service. The importance of the interrupt capability of a microprocessor depends on the kind of applications to which it will be exposed. When a number of peripheral devices interface the microprocessor, one or several simultaneous interrupts may occur on a frequent basis. Multiple interrupt requests require the processor to be able to accomplish the following: to delay or prevent further interrupts; to break into an interrupt in order to handle a more urgent interrupt; to establish a method of interrupt priorities; and, after completion of interrupt service, to resume the interrupted program from the point where it was interrupted.

INTERRUPT MASK

(Interrupt Enable): A mechanism that allows the program to specify whether or not interrupt requests will be accepted.

INTERRUPT REQUEST

A signal to the computer that temporarily suspends the normal sequence of a routine and transfers control to a special routine. Operation can be resumed from this point later. Ability to handle interrupts is very useful in communication applications where it allows the microprocessor to service many channels.

INTERRUPT SERVICE ROUTINE

A routine (program) to properly store away to the stack the present status of the machine in order to respond to an interrupt request; perform the "real work" required by the interrupt; restore the saved status of the machine; and then resume the operation of the interrupted program.

INTERRUPT VECTOR

Typically, two memory locations assigned to an interrupting device and containing the starting address and processor status word for its service routine.

JUMP

1) An instruction which, when executed, can cause the computer to fetch the next instruction to be executed from a location other than the next sequential location. Synonymous with branch. 2) A departure from the normal one-step incrementing of the program counter. By forcing a new value (address) into the program counter, the next instruction can be fetched from an arbitrary location (either further ahead or back). For example, a program jump can be used to go from the main program to a subroutine, from a subroutine back to the main program, or from the end of a short routine back to the beginning of the same routine to form a loop. See also BRANCH INSTRUCTION. If you reached this point from branch, you have executed a jump. Now return.

LIFO

Last-In, First-Out method of storing and retrieving data in a stack, table or list.

LOADERS

The various applications (user written) programs must be placed in the proper locations of the system memory. The Programs that do this job are called loaders. Loader programs range from simple ones that load absolute binary object code with no error detection, to sophisticated loaders that load relocatable binary object code, resolve global (between Program) symbolic label linkages, perform error detection, and execute various commands, including starting the program just loaded.

LOOP

A self-contained series of instructions in which the last instruction can cause repetition of the series until a terminal condition is reached. Branch instructions are used to test the conditions in the loop to determine if the loop should be continued or terminated.

MACHINE CYCLE

The basic CPU cycle. In one machine cycle an address may be sent to memory and one word (data or instruction) read or written, or, in one machine cycle a fetched instruction can be executed.

MACHINE LANGUAGE

The numeric form of specifying instructions ready for loading into memory and execution by the machine. This is the lowest level language in which to write programs. The value of every bit in every instruction in the program must be specified (e.g., by giving a string of binary, octal, or hexadecimal digits for the contents of each word in memory).

MEMORY

A general term which refers to any storage media for binary data. Basic memory functional types include read/write and read-only.

That part of a computer that holds data and instructions. Each instruction or datum is assigned a unique address that is used by the CPU when fetching or storing the information.

MEMORY ADDRESS REGISTER

The CPU register that holds the address of the memory location being accessed.

MEMORY ADDRESSING MODES

The method of specifying the memory location of an operand. Common addressing modes are: direct, immediate, relative, indexed, and indirect. These modes are important factors in program efficiency.

MEMORY CYCLE

The operations required for addressing, reading, writing, and/or reading and writing data in memory.

MEMORY MAP

A listing of addresses or symbolic representations of addresses which define the boundaries of the memory address space occupied by a program or a series of programs. Memory maps can be produced by a high-level language such as FORTRAN.

MICROCODE

A set of control functions performed by the instruction decoding and execution logic of a computer which defines the instruction repertoire of that computer. Microcode is not generally accessible by the programmer.

MICROCOMPUTER

A class of computer having all major central processor functions contained on a single printed circuit board constituting a stand-alone module. Microcomputers are typically implemented by a small number of LSI circuits and are characterized by a word size not exceeding 16 bits, and very low cost, usually under $1,000. A computer whose CPU is a microprocessor. A microcomputer is an entire system with microprocessor, memory, and input-output controllers.

MICROPROGRAM

A combination of primitive computer operations which are executed in parallel and/or serial and which accomplish the execution of one programming level instruction. The instruction sets of computers may be hardwired or they may be microprogrammed. Microprogramming takes place at a level of abstraction below the programming level. The essential difference between the two levels is that at the programming level computers are represented as sequential devices whereas at the microprogramming level operations take place in parallel and many components are active simultaneously.

MICROPROCESSOR

A single LSI circuit which performs the functions of a CPU. Some characteristics of a microprocessor include small size, inclusion in a single integrated circuit or a set of integrated circuits, and low cost.
Frequently called "a computer on a chip," the microprocessor is, in reality, a set of one, or a few, LSI circuits capable of performing the essential functions of a computer CPU.

MNEMONIC CODE

Computer instructions written in brief, easy-to-learn, symbolic or abbreviated form. Mnemonic code is also recognizable by the assembly program. For example, ADD, SUB, CLR, and MOV are mnemonic codes for instructions which will be executed as machine code.

MONITOR PROGRAMS

Monitor programs (also called supervisors, executives, and operating systems) enable you to communicate with all of the system hardware and software. They allocate available resources as efficiently as possible, and range from simple microcomputer monitors to complex time-sharing systems.

NESTING

A programming technique in which a segment of a larger program is executed iteratively (looping) until a specific data condition is detected, or until a predetermined number of interactions has been performed. The nesting technique allows a program segment to be nested within a larger segment and that segment to be nested within an even larger segment.

NIBBLE

A sequence of 4 bits operated upon as a unit. Also see byte.

NOP

Contraction of No Operation. An instruction which specifically instructs the computer to do nothing for one cycle, and then to get the next instruction.

OBJECT PROGRAM

The binary form of a source program produced by an assembler or a compiler. The object program is composed of machine-coded instructions that the computer can execute.

OPERAND

Any of the quantities arising out of or resulting from the execution of a computer instruction. An operand can be an argument, a result of computation, a constant, a parameter, the address of any of these quantities, or the next instruction to be executed. The field of an assembly language or machine instruction which specifies the data to be operated on.

OPERATION CODE (OP-CODE)

That part of a computer instruction word which designates the function performed by a given instruction. For example, the op-codes for arithmetic instructions include ADD, SUB, DIV, and MUL.

OVERFLOW

A condition occurring in a computer when the results of a mathematical operation produces a result which has a magnitude exceeding the capacity of the computer's data word size.

PAGE

A natural grouping of memory locations by higher-order address bit. In an 8-bit microprocessor, 256 consecutive bytes often may constitute a page. Words on the same page only differ in the lower-order 8 address bits.

PAGE ZERO

The memory page that includes the lowest numbered memory addresses.

PARITY CHECK

A method of checking the correctness of binary data after that data has been transferred from or to storage. An additional bit, called the parity bit, is appended to the binary word or character to be transferred. The parity bit is the single-digit sum of all the binary digits in the word or character and its logical state can be assigned to represent either an even or an odd number of 1s making up the binary word. Parity is checked in the same manner in which it is generated.

PERIPHERAL DEVICE

A general term designating various kinds of machines which operate in combination or conjunction with a computer but are not physically part of the computer. Peripheral devices typically display computer data, store data from the computer and return the data to the computer on demand, prepare data for human use, or acquire data from a source and convert it to a form usable by a computer. Peripheral devices include printers, keyboards, graphic display terminals, paper tape reader/punches, analog-to-digital converters, audio cassette tape recorders, etc.

POINTER

Registers in the CPU that contain memory addresses. See PROGRAM COUNTER.

POINTER ADDRESS MODE

The pointer consists of one or two internal registers that must be set with the desired address before an instruction referring to memory is called. This mode is not unlike indirect addressing except that it uses registers rather than main memory, and it requires more time because the pointer must be set with separate instructions, such as load immediate or increment.

PROGRAM

A complete sequence of computer instructions necessary to solve a specific problem, perform a specific action, or respond to external stimuli in a prescribed manner. As a verb, it means to develop a program.

PROGRAM COUNTER

A CPU register that specifies the address of the next instruction to be fetched and executed. Normally it is incremented automatically each time an instruction is fetched.

A register that holds the identification of the next instruction.

PROGRAM STATUS WORD (PSW)

A special-purpose register within the 2650 processor that contains status and control bits. It is 16 bits long and is divided into two bytes called Program Status Upper (PSU) and the Program Status Lower (PSL).

PSEUDO INSTRUCTION

A symbolic statement meaningful only to the program containing it, rather than to the computer as a machine instruction.

PUSH-DOWN STACK

Dedicated consecutive temporary storage registers in a computer, sometimes part of system memory, structured so that the data items retrieved are the most recent items stored in the stack.

RANDOM ACCESS

Accessibility of data is effectively independent of the location of the data.

READ

The Process of transferring information from an input device into the computer. Also, the process of taking information out of the computer's memory.

READ-WRITE CYCLE

The sequence of operations required to read and write (restore) memory data.

REGISTER

1) A temporary storage unit which can be implemented as a hardware device or as a software structure and used to store data for manipulation and/or processing reference. Typically, a register consists of a single computer word or a portion of a word. 2) A fast-access circuit used to store bits or words in a CPU. Registers play a key role in CPU operations. In most applications, the efficiency of programs is related to the number of registers.

REGISTER ADDRESS MODE

Access to a general-purpose register can be achieved by citing the name of the register in the instruction.

REGISTERS

An array of hardware binary circuits (flip-flops, toggles) for temporary storage of information. Registers can be wired for operation to allow flexible control of the contained information; that is, for arithmetic operations, shifts, transfers. The nature of the data determines whether the register is an index, pointer, program counter, flag, or temporary register.

RELATIVE ADDRESS

1) An address of a machine instruction which is referred to an origin address. For example, consider the relative address 15 which is translated into the absolute address origin R + 15, where R is, typically, the contents of the PC register. Relative addressing allows the generation of position-independent code. 2) The number that specifies the difference between the actual address and a base address.

RELATIVE ADDRESSING

The address of the data referred to is the address given in the instruction plus some other number. The "other number" can be the address of the instruction, the address of the first location of the current memory page, or a number stored in a register. Relative addresssng permits the machine to relocate a program or block of data by changing only one number.

RELOCATABLE

Object programs that can reside in any part of system memory. The actual starting address is established at load time by adding a relocation offset to the starting address. Relocatable code is typically composed of position-independent code.

RELOCATABLE PROGRAMS

Programs having symbolic rather than absolute addresses.

ROUTINE

A program or program segment designed to accomplish a single function.

SCRATCHPAD MEMORY

Scratchpad memory usually designates an area of memory used for many quick data transfers. It is the most frequently used memory segment. Some microprocessors have simplified instructions that can only be used in a certain small part of the memory (say, the first 256 bytes), where the most-significant byte of the address is zero. The scratchpad is usually placed in such a location.

SERIAL I/O

A method of data transfer between a computer and a peripheral device in which data is transmitted for input to the computer (or output to the device) bit by bit over a single circuit.

SERVICE ROUTINE

A set of instructions to perform a programmed operation, typically in response to an interrupt

SET

A signal condition representing a binary "one."

SHIFT REGISTER

A register in which binary data bits are moved as a contiguous group a prescribed number of positions to the right or to the left.

SIMULATOR, SOFTWARE

Program written to run on computer 'A' but which simulates the execution of instructions of computer 'B'. Allows debbugging and verification of computer 'B' software.

Software simulators are sometimes used in the debug process to simulate the execution of machine-language programs using another computer (ofter a time-sharing system). These simulators are especially useful if the actual computer is not available. They may facilitate the debugging by providing access to internal registers of the CPU which are not brought out to external pins in the hardware.

SINGLE LEVEL INTERRUPT

The interrupt signal causes transfer of control to a pre-assigned memory location at which the interrupt processing routine starts. The program must poll all possible sources of interrupt to determine which one requires service.

SINGLE LEVEL VECTOR INTERRUPT

The interrupt signal causes the microprocessor to interrogate the vector (V2, V1, V0), which specifies an address to which the program jumps to find the appropriate service subroutine. Each possible source of interrupt can be assigned a different service subroutine. Vector interrupt eliminates the need for polling.

SINGLE-OPERAND INSTRUCTION

An instruction containing a reference to one register, memory location, or device.

SKIP

An instruction which causes the computer to omit the instruction in the immediately following location.

SOFTWARE/FIRMWARE

The microprocessor is generally a stored program computer, with its collection of programs and instructional procedures referred to as Software. Software, by directing the hardware, enables the microprocessor to perform a functional system related task. In a fixed instruction microprocessor, a set number of instructions of operations are defined with fixed word lengths, and these exercise the CPU independent of the data. Software is alterable and accessible by the user.

Firmware can be considered an extension to a computer's basic instruction repertoire that creates microprograms for a software instruction set. This extension to the basic instruction set is often permanently burned into Read Only Memory (ROM), rather than being implemented in software. Firmware programs may be composed of instructions of variable width; the number of instructions in a Firmware program is generally smaller than in a Software program, although the instructions are usually much wider. A Firmware program can be used to implement a Software instruction set; this occurs in the emulation of larger minicomputers by bit slice microprocessors.

SORT

A function performed by a program, usually part of a utility package; items in a data file are arranged or rearranged in a logical sequence designated by a key word or field in each item in the file.

SOURCE ADDRESS

In computer systems having a source-destination architecture, the source address is the address of the device address or memory location from which data is being transferred.

SOURCE PROGRAM

A program coded in other than machine language (in assembly or compiler language) that must be translated into machine language for use. Assembly and compiler language programs are human readable whereas object programs are machine readable.

STACK

1) A dynamic, sequential data list, usually contained in system memory, having special provisions for program access from one end or the other. Storage and retrieval of data from the stack is generally performed by the processor automatically. 2) A sequence of registers and/or memory locations used in Last In, First Out (LIFO) fashion. A stack pointer specifies the last-in entry (or where the next-in entry will go).

STACK POINTER

The stack pointer is coordinated with the storing and retrieval of information in the stack. The stack pointer is decremented by one immediately following the storage in the stack of each byte of information. Conversely, the stack pointer is incremented by one immediately before retrieving each byte of information from the stack. The stack pointer may be manipulated for transferring its contents to the index register or vice versa.

STACK, SUBROUTINE LINKAGE
One-dimensional array or registers used specifically for storing subroutine return addresses.

STARTING ADDRESS
The address of a memory location in which is stored the first instruction of a given program.

STATEMENT
An instruction in any computer-related language.

STATUS CODES (CONDITION CODES)
Indicators used to record the resulting condition of data in the accumulator. Four control bits are set as a result of each arithmetic and logical operation: carry flip-flop (C), sign flip-flop(s), and parity flip-flop (P). The carry bit provides a means of performing multiple precision binary arithmetic.

STRING
A connected sequence of entities.

SUBROUTINE
1) A short program segment which performs a specific function and is available for general use by other programs and routines. 2) A subprogram (group of instructions) reached from more than one place in a main program. The process of passing control from the main program to a subroutine is a subroutine call, and the mechanism is a subroutine linkage. Often data or data addresses are made available by the main program to the subroutine. The process of returning control from subroutine to main program is subroutine return. The linkage automatically returns control to the original position in the main program or to another subroutine. 3) Programming technique that allows the same instruction sequence or subprogram to be given control and used repeatedly by other sections of the program.

SYMBOL TABLE

A table in which symbols and their corresponding values are recorded.

SYMBOLIC ADDRESS

A label assigned instead of absolute numeric addresses, usually for purposes of relocation.

TRAP

A CPU-initiated interrupt which is automatically generated when a predetermined condition, such as an illegal instruction, a breakpoint, a specified error, or a power failure is detected. Two vector locations are dedicated for each trap type. The vector locations contain the PC and PS for the service routine.

TEMPORARY STORAGE

Memory locations or registers reserved for immediate and partial results obtained during the execution of a program.

TWO'S COMPLEMENT

A two's complement number is obtained electronically by inverting the states of all bits in the number and adding one (complement and increment). Two's complement arithmetic is widely used in microprocessors.

UTILITY ROUTINE

A standard routine, usually part of a larger software package, which performs a service and/or program maintenance function, such as file maintenance, file storage and retrieval, media conversions, and production of memory and file printouts.

VARIABLE

A named memory (RAM) location which is given some consistent and meaningful interpretation by the programmer and which will contain different data values during the execution of the program. A variable may contain a boolean, integer, floating point, ASCII, etc. value. The type of data stored in each variable should remain consistent throughout the execution of the program. RAM memory locations can be reserved for variables using the 2650 language RES statement.

VECTORED INTERRUPT

This term is used to describe a microprocessor system in which each interrupt, both internal and external, have their own uniquely recognizable address. This enables the microprocessor to perform a set of specified operations which are preprogrammed by the user to handle each interrupt in a distinctively different manner.

WORD

A set of binary bits handled by the computer as the primary unit of information. The length of a computer word is determined by the hardware design. Typically, each system memory location contains one word.

WRITE

The process of storing data in memory.

signetics

a subsidiary of U.S. Philips Corporation

Signetics Corporation
P.O. Box 9052
811 East Arques Avenue
Sunnyvale, California 94086
Telephone 408/739-7700

Printed in USA AUG 1979