COMPUTER AUTOMATION

ALPHA 16 & NAKED MINI™ 16 COMPUTER REFERENCE MANUAL

INDEX

$20.00

ALPHA 16 and NAKED MINI™ 16 COMPUTER

REFERENCE MANUAL

JANUARY 1972
(REVISED EDITION)

COMPUTER AUTOMATION, INC. 895 W. 16th ST., NEWPORT BEACH, CALIF. 92660

COPYRIGHT, 1971 COMPUTER AUTOMATION, INC.

00-9701900-A0

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

SECTION 1

GENERAL DESCRIPTION

1.1 INTRODUCTION

1.1.1 General

The ALPHA 16 and NAKED MINI 16 are general purpose, stored program digital computers. They are extensions of the successful and proven 16-bit computer family from Computer Automation, and are effectively repackaged and improved versions of the Model 116 computer.

1.1.1.1 Upward Compatibility. Both the ALPHA 16 and NAKED MINI 16 are upward software and I/O compatible with earlier 16-bit computers from Computer Automation. Figure 1-1 illustrates the evolution of these computers. Upward software compatibility means that programs written for the earlier 16-bit computers will run without change on the ALPHA 16 or NAKED MINI 16. However, due to the expanded and improved instruction set of the ALPHA 16 and NAKED MINI 16, programs written for these computers may not run on the earlier computers.

1.1.1.2 General Features. All of the 16-bit computers from Computer Automation feature a 16-bit word format and a very powerful and efficient instruction set of over 145 basic instructions. The ALPHA 16 and NAKED MINI 16 incorporates all of the power and flexibility of the earlier computers plus some new instructions and features that make these computers a major advance in the mini computer field. Perhaps the most significant advance is the incorporation of byte processing and byte addressing as well as full 16-bit word processing and 16-bit word addressing. Since most peripheral devices are byte oriented, this feature alone improves software efficiency and memory efficiency tremendously. Software packing and unpacking of bytes is virtually eliminated. Data may be packed two bytes to each word automatically by the computer hardware even when performing block transfers of data between the computer and high speed peripheral devices such as magnetic tape or disks.

In addition to byte processing instructions, additional instructions have been incorporated in the ALPHA 16 and NAKED MINI 16 to improve I/O operations, interrupt control, and processor control.

1.1.2 The NAKED MINI Concept

Within the 16-bit computer family from Computer Automation, the NAKED MINI 16 is the most revolutionary. Conventional mini computers have followed the design concepts of larger computers in that they have been designed to work as stand-alone processors with some peripheral devices attached. Figure 1-2 illustrates a conventional mini computer in a typical application. This figure shows that the conventional mini computer is effectively a separate entity from the system in which it is used. It has its own power supply and control panel separate from the power supply and control panel used by the remainder of the system. It treats the remainder of the system as peripherals to the mini computer.

1.1.2.1 System Component. The NAKED MINI 16 is designed to be a component of a system rather than a separate entity that is connected to the system in which it is used. Figure 1-3 illustrates a typical NAKED MINI 16 application. The NAKED MINI 16 is designed to be used as a system component along with other system components. It depends on the system power supply for a source of power. It depends on the system control panel for controlling signals that may be needed. It is truly a modular component of the system in which it is used.

1.1.2.2 System Advantages. Elimination of a separate computer power supply and control panel reduces the cost of the computer component in the system. Elimination of

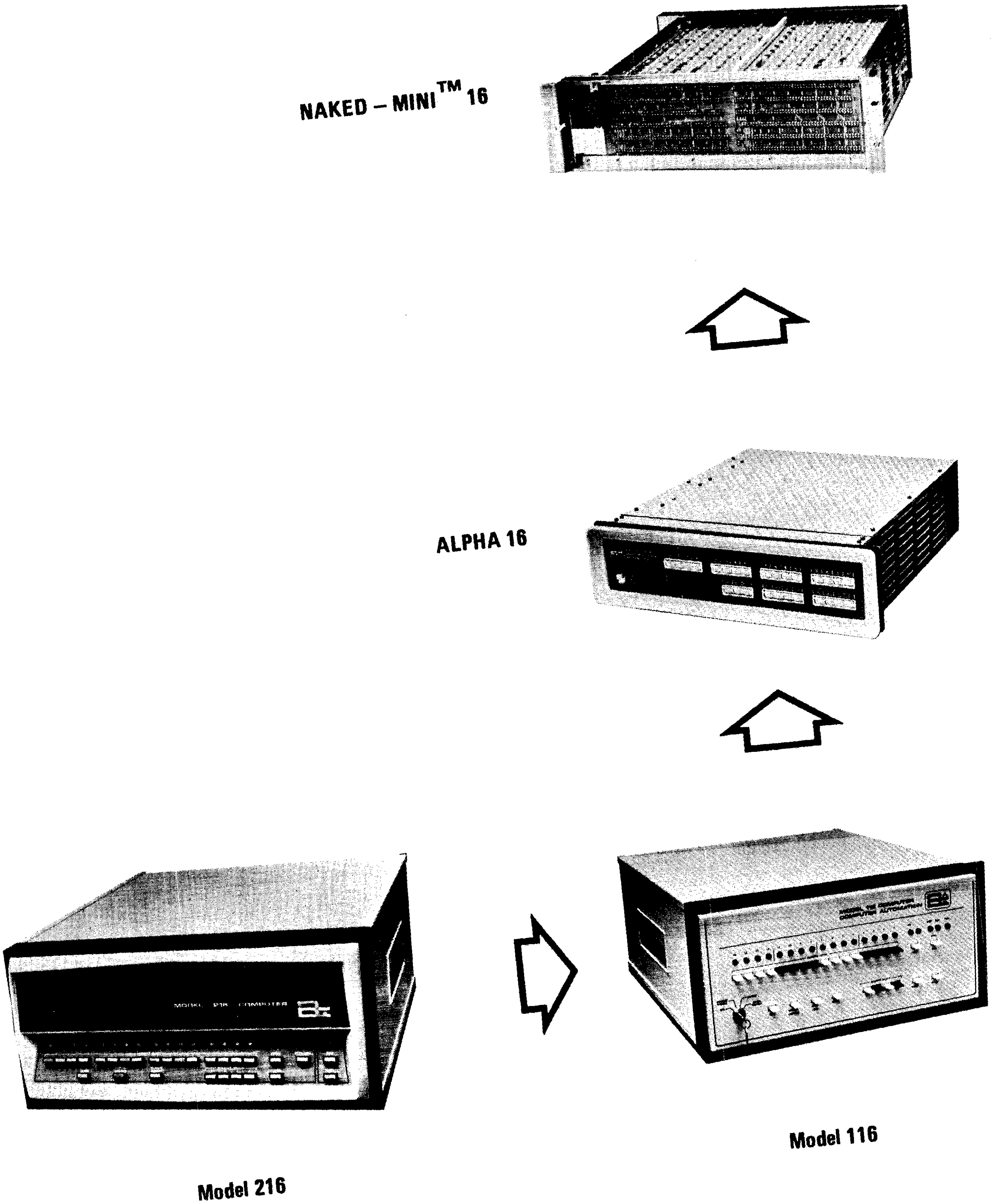


Figure 1-1. Evolution of Compatible 16-Bit Computers

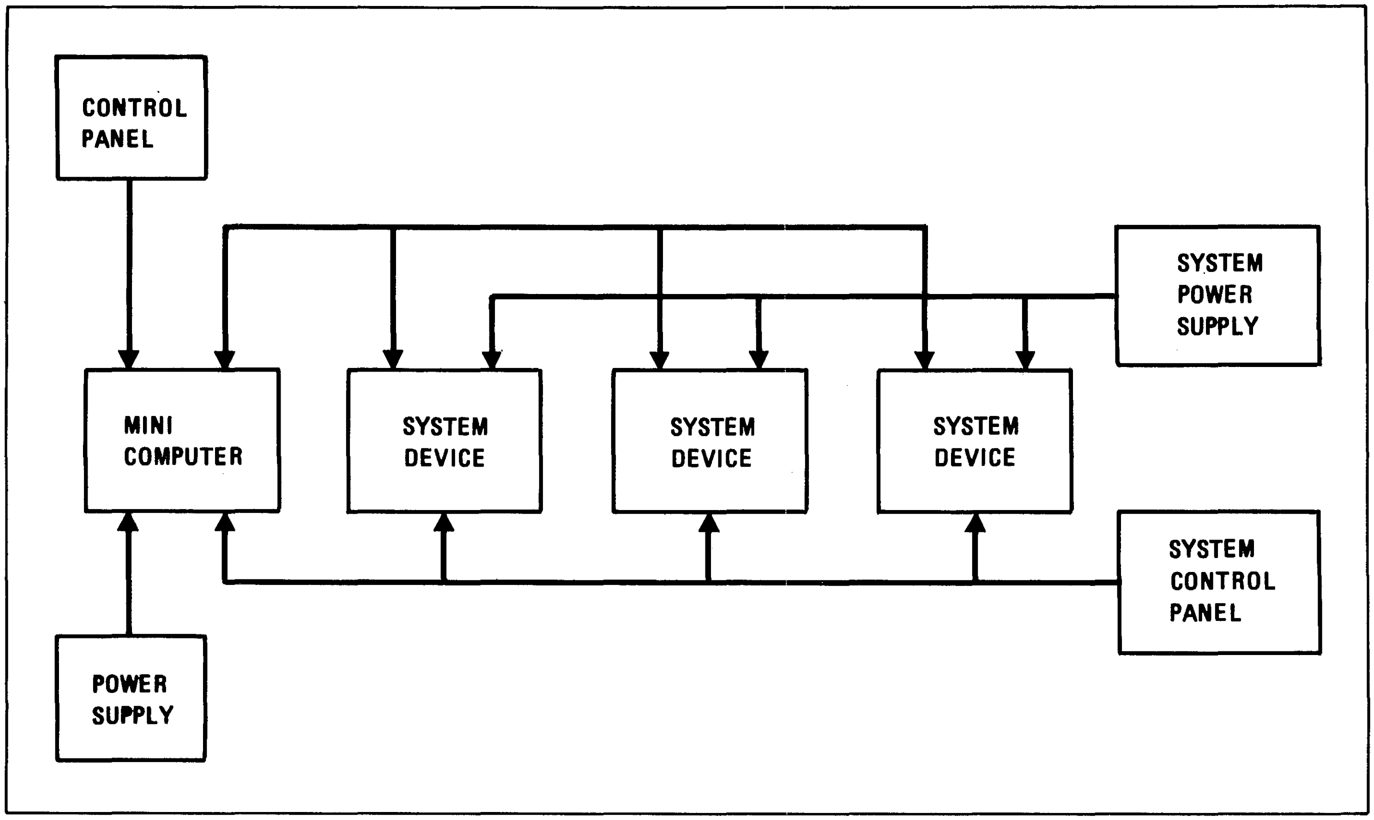


Figure 1-2. Conventional Mini Computer Application

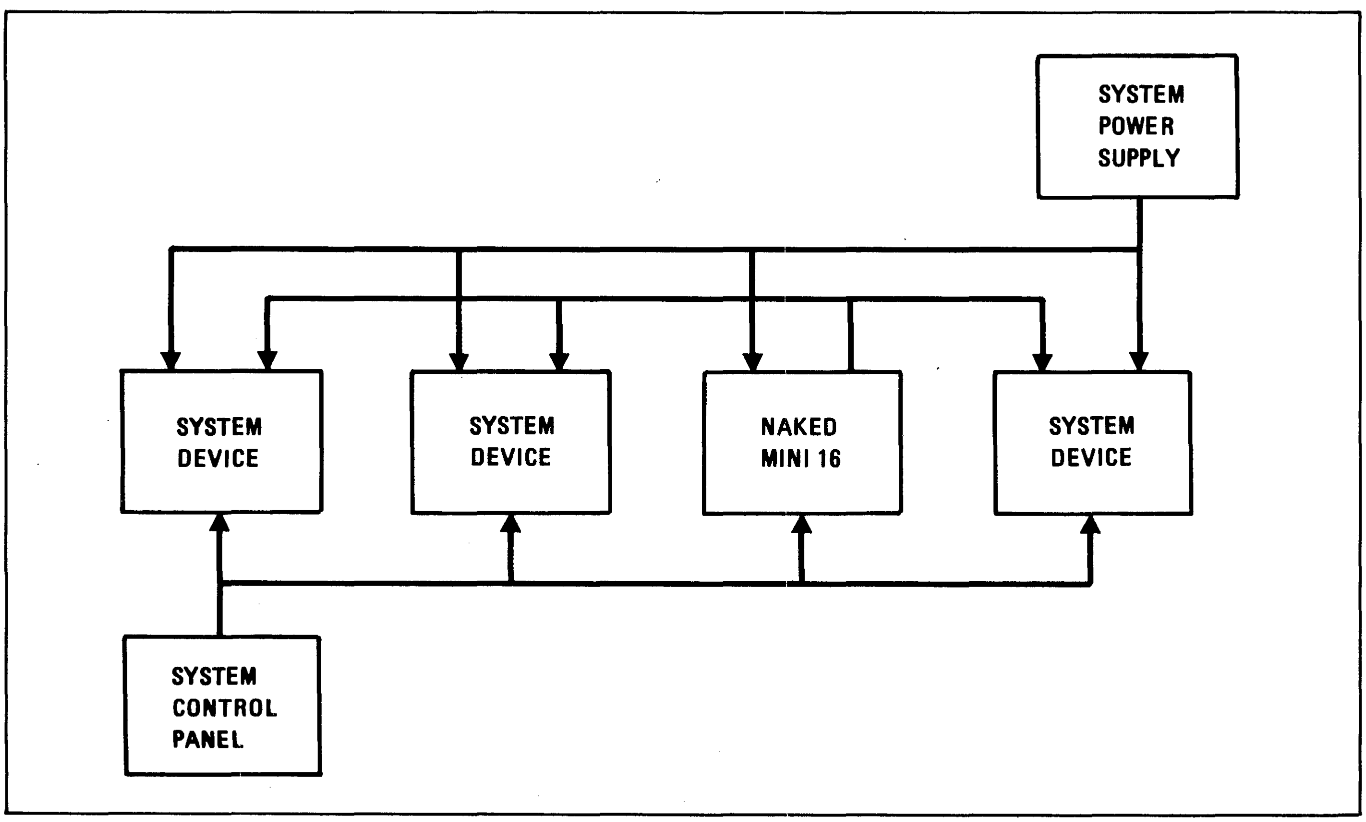


Figure 1-3. NAKED MINI 16 Application

the control panel also reduces the possibility of inexperienced operators interfering with system operation by misuse of the computer control panel. Since the computer control panel is incorporated in the system control panel, the need for the computer to be "front and center" is eliminated, thus enhancing design and packaging flexibility for the total system in which the NAKED MINI 16 is used.

1.1.2.3 Module Concept. The name "NAKED MINI 16" was chosen to emphasize the concept of a computer as a component or module which is a fully tested operational unit. Apply power and the NAKED MINI 16 runs without a control panel. If external control is needed, a console connector is available for connecting the NAKED MINI 16 to a system control panel.

1.1.3 The ALPHA 16

The powerful instruction set and I/O structure of the NAKED MINI 16 can be very useful in a stand-alone processor in the conventional sense. The ALPHA 16 is a conventional mini computer with all of the power of the NAKED MINI 16. It is effectively a "dressed" NAKED MINI 16. It has a dedicated control panel and its own power supply. In addition, it is mounted in an air cooled chassis.

The ALPHA 16 processor is identical to the NAKED MINI 16 processor. Printed circuit boards are interchangeable between the two machines. The two machines are identical in every respect, except for the packaging and the inclusion of a power supply and a control panel with the ALPHA 16.

1.1.4 Applications

These computers are designed for commercial, industrial control, and monitoring applications where emphasis is on reliability, flexibility, and economy. Extensive experience has shown that there is no limit to the applications of this 16-bit computer family. Some current applications include:

- Production test and automation
- EDP source data entry
- Point-of-sale systems
- Scientific and medical instrumentation

1.2 CHARACTERISTICS

1.2.1 General

Detailed characteristics of the ALPHA 16 and NAKED MINI 16 are explained in subsequent sections of this manual. The following is an overview of the characteristics of these computer.

1.2.2 Processor

Some of the significant characteristics of the computer processor are:

- Parallel processing of full 16-bit words and 8-bit bytes
- Seven 16-bit hardware registers
- Memory word size of 16 bits, with each word addressable as a full 16-bit word or as two separate 8-bit bytes
- Memory capacity is 2,048 words minimum, expandable to 32,768 words maximum, with 4,096 words standard
- Computer cycle time is 1.6 microseconds with memory cycle time included within the computer cycle time.
- Binary 2's complement arithmetic processing
- Automatic memory scan (standard)
- Hardware Multiply and Divide (standard)

1.2.3 Instruction Set

These computers have a very powerful instruction set consisting of 145 basic instructions divided into seven classes. The instruction classes are:

- Memory Reference — These instructions access memory in either full word or byte mode and perform logical and arithmetic operations involving data in memory and data in hardware registers.
- Immediate — These instructions are similar to memory reference in that they perform logical and arithmetic operations involving memory data and data in hardware registers. The memory data, however, is contained within the instruction word so that it is immediately available for processing without requiring an operand cycle to fetch it from memory.
- Conditional Jump — These instructions test conditions within the processor and perform conditional branches depending on the results of the tests performed. Jumps may be as much as ±64 locations from the location of the conditional jump instruction.
- Shift — These instructions include single-register logical shifts, single-register arithmetic shifts, single-register rotate shifts, double-register logical shifts, and double-register rotate shifts. The hardware multiply and divide instructions are part of this class.
- Register Change — These instructions provide logical manipulation of data within hardware registers.
- Control — These instructions are used to enable and disable interrupts. suppress status, control word or byte mode data processing and perform other general control functions.
- Input/Output — These are the instructions that provide communications between the computer and external devices. They include conventional I/O instruction plus Block Transfer and Automatic Input/Output instructions.

1.2.4 Memory Addressing

An important feature of these machines is the ability to access full 16-bit words and 8-bit bytes (half words) in core memory. Core memory may be as small as 2K 16-bit words, and as large as 32K 16-bit words. Since memory may contain 32K words, and since each word contains two bytes, provisions are made for addressing up to 64K bytes.

Instructions which access memory may operate in either word or byte mode. Memory reference instructions are sixteen bits in length (one-word instructions), with the eight least-significant bits plus three control bits dedicated to memory addressing. The eight least significant bits address 256 words or bytes. The ALPHA 16 and NAKED MINI 16 computers use the three control bits to specify several addressing modes. These addressing modes are discussed briefly in the following paragraphs, and are explained in detail in Section 2. The addressing modes used are Scratchpad, Relative Forward, Relative backward, Indexed, and Indirect.

- Scratchpad — Scratchpad addressing uses the 8-bit address field of the memory reference instruction as the effective memory address. Scratchpad addressing accesses the first 256 words in memory in Word Mode, or the first 256 bytes in Byte Mode. The first 256 words in memory are referred to as "Scratchpad" memory, because these are common words which can be addressed directly by instructions located anywhere in memory.

- Relative — Relative addressing uses the location of the instruction which is addressing memory as a reference point, and address memory relative to that instruction. In Word Mode, relative addressing can address an area of memory extending from the instruction address forward 256 words (+256) or backward 255 words (-255). In Byte Mode, the range is forward 512 bytes. Bytes cannot be directly addressed relative backward.

- Indexed — There is a register in the processor which can be added to the address field of memory reference instructions to form an effective memory address. This register is the Index, or X, register. The Index register is a 16-bit register which can be set by software to any desired value. The address of any specific word (in Word Mode) or byte (in Byte Mode) may be formed by adding the address field of the instruction to the value in the Index register and using the result to address memory.

- Indirect — Indirect addressing uses scratchpad or relative addressing to access a word in memory which contains the address of a memory operand. The word that contains a memory address rather than an operand is called an Address Pointer. In Word Mode multi-level indirect addressing is possible; i.e., one Address Pointer may contain the address of another address pointer rather than the address of an operand. In Byte Mode, only one level of indirect addressing is possible.

  Indirect addressing may also be used in conjunction with indexing. When indexed indirect addressing is specified, the indirect operation is performed first and then the contents of the X Register are added to the contents of the Address Pointer. This process is called Post Indexing.

1.2.5 I/O Structure

The ALPHA 16 and NAKED MINI 16 have a parallel I/O structure that provides both ease of interfacing and powerful peripheral control. Some special features of the I/O Structure are:

- **Vectored Interrupts** — These machines feature vectored hardware priority interrupts. There are three standard interrupt lines. The third, with control lines, can accommodate a virtually unlimited number of vectored interrupts.
- **Direct Memory Channels** — Direct memory channels (DMC) provide data transfers between the computer and peripheral components without affecting the operating registers of the computer. DMC's are a standard feature of these computers. The maximum data transfer rate using DMC's under interrupt control is 238,000 bytes/sec.
- **Block Input/Output** — The Block I/O feature of these computers dedicates the computer to I/O data transfer at the maximum possible transfer rate. The maximum transfer rate using Block I/O is 1,000,000 bytes/sec. Block I/O is a standard feature of these computers.
- **Parallel Busses** — Separate busses providing device address selection, data transfer, and control signals are used for ease of interfacing. Busses are not time shared for I/O functions. This feature alone simplifies interface design considerably.

1.2.6 Processor Mounted Options

Processor Mounted Options are those optional features which are mounted directly on basic processor printed circuit boards. Since these options are mounted on basic processor boards, they do not occupy plug-in interface/option slots within the computer chassis. The processor mounted options are:

- **Teletype Interface** — Interfaces a modified ASR-33 or ASR-35 Teletype to the computer. This is a fully-buffered interface that includes remote Teletype power on/off control.
- **Power Fail Restart** — This option includes the hardware necessary to detect low input power conditions, and bring the computer to an orderly halt until normal input power is restored. When normal power is restored this option will generate an orderly restart. The Power Fail Restart option allows completely unattended operation of the computer at locations where power conditions are unreliable.
- **Real Time Clock** — The Real Time Clock option features a crystal controlled internal clock which may be wired to produce clock rates of 100 microseconds, 1 millisecond, or 10 milliseconds. The 10 millisecond rate is standard. An external clock

source, such as AC line frequency, may also be used. The Real Time Clock provides time-of-day information to the computer and may be used to time periodic events that must be controlled by the computer.

- Memory Protect — Memory Protect provides a means for protecting selected sections of core memory from destruction by program-generated or I/O-generated write commands. The segment to be protected is selected by jumper wiring. Protect mode may optionally be enabled and disabled by software.

- Multi-Device Autoload — The Multi-Device Autoload option consists of a Read-Only Memory (ROM) programmed with a complete binary loader which is capable of loading binary programs from any one of several input devices. The Autoload hardware consists of the ROM and the necessary logic to cause the computer to execute the program in ROM when the Autoload switch is activated.

1.2.7 Processor Plug-In Options

Locations are provided within the computer chassis for the installation of processor options, peripheral interfaces, and memory modules. The options are mounted on printed circuit boards which plug into the locations within the computer chassis. Some of the available plug-in processor options are:

- DTL I/O buffers, up to 64 bits
- Relay I/O buffers, up to 32 isolated relays
- Modem interfaces: non-synchronous, synchronous, parallel, and autodial; multiplexed up to 16 channels
- Direct Memory Access, allowing peripheral access to memory on a cycle-steal basis at data transfer rates of 1,250,000 bytes/sec.
- Read Only Memory (ROM)

1.2.8 Peripheral Equipment

The following is a partial list of the various types of peripheral equipment for which interfaces to the ALPHA 16 and NAKED MINI 16 have been developed. This list does not imply that these are the only devices for which interfaces can be developed. The interface structure of these computers is such that virtually any peripheral device can be interfaced to the computer.

- ASR-33 and ASR-35 Teletypewriters
- High speed paper tape readers and punches
- Line printers
- Card readers
- Open reel and cassette magnetic tape units
- Magnetic disks
- A/D and D/A converters
- CRT terminals

1.2.9 Standard Software

The following is a brief description of the standard software packages provided with the ALPHA 16 and NAKED MINI 16 computers. Detailed operating procedures and descriptions of each program are provided separately.

- BETA — BETA is a symbolic assembler for translating free-form source (symbolic code) tapes into Object Language tapes which can be loaded into the computer and executed. In addition to recognizing symbolic instruction codes, BETA recognizes a full set of pseudo-operation codes. The symbolic instruction codes recognized by BETA are those codes listed in the definitions of the ALPHA 16 and NAKED MINI 16 instructions in subsequent sections of this manual.

- STP — Source Tape Preparation. STP provides a means for preparing and/or editing symbolic source tapes for input to BETA. STP is used with an ALPHA 16, a teletype keyboard, and a paper tape punch. Source lines are entered through the keyboard and are stored temporarily in the computer memory where they may be edited before being punched on paper tape. Source code may be edited in memory, or previously prepared source tapes may be read into memory through a paper tape reader and edited to produce a corrected source tape. Source listings are also produced by STP.

- OMEGA — OMEGA is a conversational assembler that includes the features of BETA and STP in one program. Source code may be typed in, edited, and assembled using this one program. Source tapes, source listings, Object (assembly) tapes, and assembly listings are produced by OMEGA.

- ROLL — Relocatable Object Language Loader. BETA and OMEGA generate Object Language tapes. These tapes are not binary images of programs as they appear in core memory when the programs are executed. Object Language tapes are relocatable; i.e., they may be loaded anywhere in memory by an Object Language Loader. ROLL is a sophisticated loader capable of reading Object language tapes, assigning memory locations, linking separate Object language tapes together into one program, and relocating programs in memory. Object language program tapes produced by BETA or OMEGA must be loaded into the ALPHA 16 or NAKED MINI 16 by ROLL.

- BLD/BDP — Binary Load/Binary Dump. This program provides a means for loading and dumping programs in absolute binary format. The Binary Dump portion of the program is normally used to dump binary images of memory in a format that may be loaded using the Binary Load portion of the program. Object Language programs that have been loaded into the computer memory using ROLL may be dumped onto a binary tape using BDP. Binary tapes may then be loaded into the computer memory in binary format using BLD. BLD/BDP is a much shorter program than ROLL, therefore much longer programs can be loaded with BLD than with ROLL. Also, ROLL is often used to link main programs on one tape with subroutines on another tape. The total program, including main program and subroutines, may be dumped by BDP and subsequently loaded using BLD. This procedure incorporates object language programs on several tapes into a single binary image tape.

- DBUG

Debug Package. DBUG is an interactive program which aids the user in debugging his programs on the ALPHA 16 or NAKED MINI 16. An ASR-33 or ASR-35 Teletype is required by DBUG. DBUG functions include: transfer control, fill memory, copy memory, search memory, breakpoint, inspect and/or change memory, and modify memory. Register save/change features assist debugging operations, and 16 relocation pseudo registers are included for accessing subroutines.

- MATH 1

Fixed Point Arithmetic Package. This package consists of twelve Object language programs which perform single- and double-precision arithmetic functions.

- MATH 2

Fixed Point Elementary Functions Package. This package is composed of the twelve most frequently used mathematical functions, organized into six convenient Object language programs on one tape. The six programs are:

1. Square Root: SQRT
2. Exponential: EXP2, EXPE, EXP1
3. Logarithmic: LOG2, LOGE, LOG1
4. Trignometric: SIN, COS, TAN
5. Arctangent: ATAN
6. Hyperbolic Tangent: TANH

- TUP

Teletype Utility Package. TUP consists of 15 object programs which perform the most common teletype input/output functions. The basic routines input or output a single character, right justified in the A Register of the computer. Conversion routines input and output single- and double-precision decimal, hexadecimal, and octal values.

- IDP — Instruction Diagnostic Program. This diagnostic program tests all memory reference and register change instructions for all possible results, and tests enough conditional jump instructions to test the skip logic. All types of addressing are checked on three of the memory reference instructions. If any test on any instruction fails, the processor will halt.

- CMD — Core Memory Diagnostic. CMD tests every core of memory to ensure that no bits are 'picked' or 'dropped.' Address logic is checked by storing the address of each memory word within the word it addresses. All words are read twice to check the read and restore logic. Error messages are typed on the teletype printer.

- WPMD — Worst Pattern Memory Diagnostic. WPMD occupies the first 32 (:20) words of memory and fills the remainder of core, to a preset limit, with the worst case pattern of zeroes and ones. This pattern is then read back and verified under the worst case noise level of memory. WPMD is preset to protect the Binary Loader during testing to facilitate reloading programs.

- TDP — Teletype Diagnostic Program. TDP tests all I/O logic that is used by the teletype interface. It tests the teletype reader, punch, and printer for every character code. It tests input and output under program control, interrupt control, and block input and output.

1.2.10 Optional Software

Software packages which are available but not included in the standard software package are briefly described below. These packages include higher-level language compilers, executives, and symbolic assemblers which may be run on machines other than Computer Automation's 16-bit computers.

- FORTRAN — Complies with ANSI (ASA) Basic FORTRAN. In addition it provides such features as N Dimensional Subscripts and Free Field Data Input. It accepts source statements and operates in 4K words of core. It operates as a one-pass compiler and provides a source listing and a relocatable object tape.

- Advanced BASIC — This package includes all the Elementary BASIC and Advanced BASIC statements defined by Kemeny and Kurtz in their book BASIC Programming, published by

John Wiley & Sons. Some additional features of this package are: unlimited depth of expression in equations, a business arithmetic package which includes picture formatting, and an immediate execute mode. This program will operate in 4K words of memory.

- Extended BASIC — Includes all the features of Advanced BASIC, plus text variables (string manipulation) and Matrix instructions. Requires 8K words of memory.

- Extended Time-Sharing BASIC — This package provides all the features of Extended BASIC to up to 16 users simultaneously. A system with eight users requires 8K words of core. A system with sixteen users requires 12K words of core.

- Sigma Cross Assemblers (CROSS) — These are assembly programs for assembling ALPHA 16 and NAKED MINI 16 source statements on XDS Sigma series computers. CROSS performs the same functions as BETA, except that CROSS runs on the Sigma machines. For the Sigma 2 and 3, CROSS is written in Sigma 3 Basic FORTRAN, and operates under the Sigma 3 Real Time Batch Monitor. For the Sigma 5 and 7, CROSS is written in FORTRAN IV and operates under the Batch Time Sharing Monitor as a terminal job, and under the Batch Processing Monitor as a batch job.

1.2.11 Processor Physical Characteristics

Physical characteristics of the ALPHA 16 and NAKED MINI 16 are summarized below. Refer to the ALPHA 16 and NAKED MINI 16 MAINTENANCE MANUAL for more detailed information concerning the physical characteristics of these machines.

- Operating Temperature — -5° C to $+55^{\circ}$ C
- Operating Humidity — 5% to 90% relative, non-condensing
- Dimensions, ALPHA 16 — 5-1/4 in. high, 19 in. wide, 19-1/2 in. deep; power supply is 3-1/2 in. high and 19 in. wide
- Dimensions, NAKED MINI 16 — 5-1/4 in. high, 19 in. wide, 18-1/4 in. deep
- AC Power Requirements ALPHA 16 — 6A at 115 VAC, 3A at 220 VAC, 47-63 Hz
- Weight, ALPHA 16 — 75 lb, including power supply and operators panel
- Weight, NAKED MINI 16 — 8.6 lb.

1.3 PROCESSOR CONFIGURATION

1.3.1 General

The ALPHA 16 and NAKED MINI 16 contain seven hardware registers, an Adder unit, a Control section, and the necessary busses to transfer data and control signals between the various units within the computer. Figure 1-4 is a block diagram of the ALPHA 16 and NAKED MINI 16 processor. Note that the Console applies to the ALPHA 16 only.

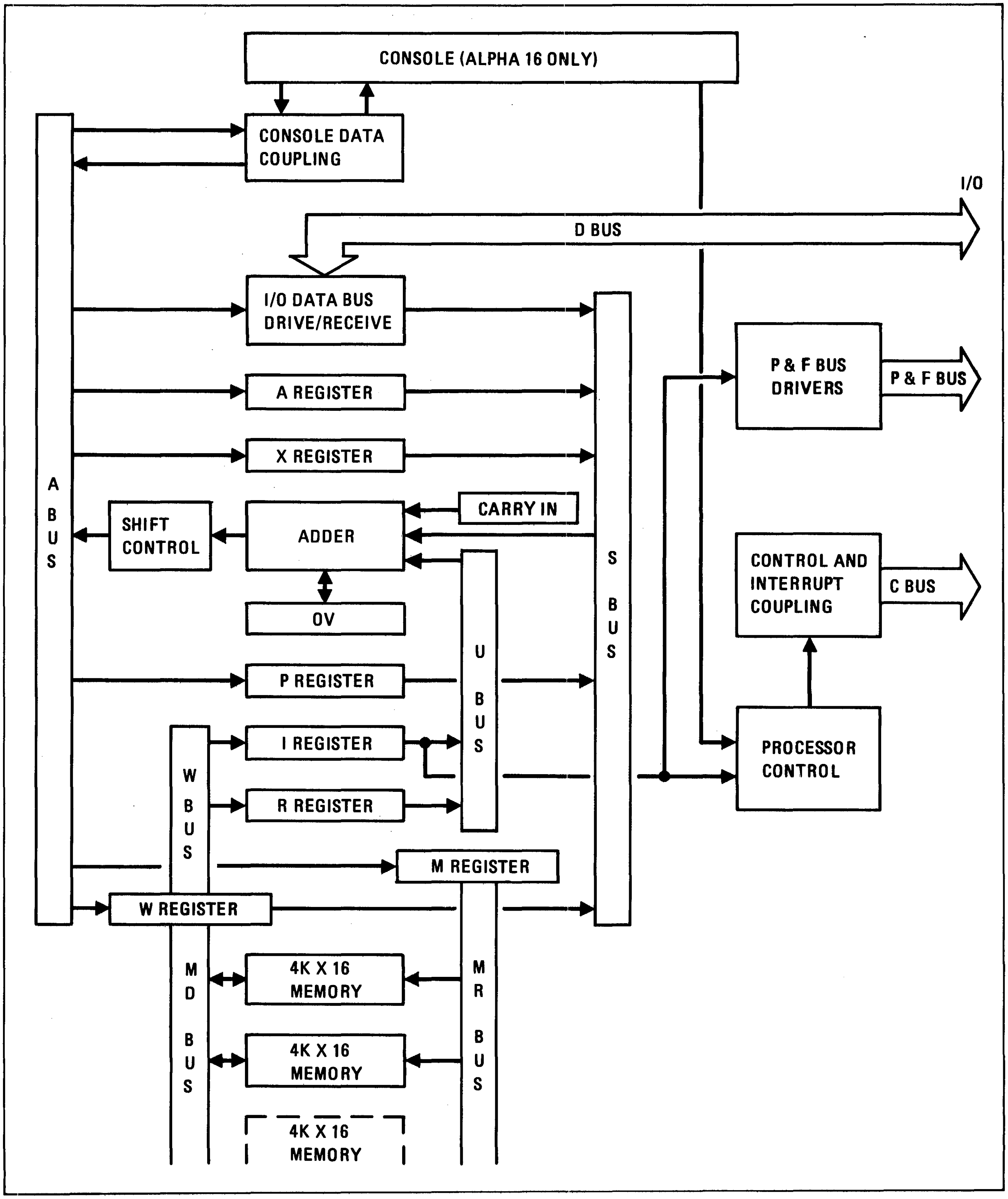


Figure 1-4. ALPHA 16 and NAKED MINI 16 Block Diagram

1.3.2 Adder

The adder is a 16-bit parallel adder which produces the sum of a 16-bit input from the S Bus, another 16-bit input from the U Bus, and a 1-bit input from the Carry-In input. The sum of these three inputs is applied to the A Bus via the Shift Control section.

The adder is a completely passive device that always presents the sum of its three inputs to the computer shift logic as long as power is applied to the computer. It has no storage capability and no control over the inputs which it receives.

1.3.3 Hardware Registers

There are seven hardware registers in the ALPHA 16 and NAKED MINI 16. The functions of the registers are described in the following paragraphs.

1.3.3.1 W Register. The W Register is a 16-bit register that interfaces the processor to the computer memory. Data read from memory is stored in the W Register after the memory read cycle is completed. Data to be written into memory is palced in the W Register prior to the start of the memory write cycle.

1.3.3.2 M Register. The M Register is a 16-bit register that interfaces the processor to the address decoding circuits of the memory. Address information is stored in the M Register at the beginning of a memory cycle and is held there until the memory cycle is completed.

1.3.3.3 P Register. The P Register is a 16-bit register that serves as the program counter. It addresses each instruction that is executed, and is incremented automatically as instructions are executed. When Skip or Jump instructions that modify the normal sequence of program execution are executed, the program branch is performed by loading the P Register with the address of the next instruction to be executed.

1.3.3.4 A Register. The A Register is a 16-bit register that is used as an accumulator for arithmetic operations. It is a general purpose register that is available to the programmer for arithmetic operations, logical functions, and I/O control.

1.3.3.5 X Register. The X Register is a 16-bit register that is used as an index register for memory address modification, and as a general purpose register for use by the programmer. It may be used for I/O control, and serves as an extension of the A Register for long shifts, hardware multiply, and hardware divide.

1.3.3.6 I Register. The I Register is the computer instruction register. It holds the instruction that is currently being executed by the computer. It is a 16-bit register.

1.3.3.7 R Register. The R Register is the computer operand register. It is a 16-bit register which holds the memory operand for memory reference instructions. It is used to hold the multiplicand for hardware multiply instructions, and the divisor for hardware divide instructions.

1.3.3.8 OV Register. The OV Register is a 1-bit register that flags arithmetic operations that exceed the capacity of the adder. It is also used in various shift, rotate, and control instructions. It may be tested and conditioned by software.

1.3.4 Processor Data Paths

Computer memory modules, registers, and control circuitry are connected by data and control busses. Busses within the ALPHA 16 and NAKED MINI 16 are parallel transmission busses. Data busses are normally 16 parallel data lines, and control busses contain the number of lines required to perform the required control functions. Figure 1-4 illustrates the bus structure of the ALPHA 16 and NAKED MINI 16 processor.

1.3.4.1 A Bus. The A Bus is one of the two principle data paths within the computer processor. It receives data from shift control and from the Console Data Coupling logic. It is the only source of data for the A, X, P, and

M Registers. It is also a source of data for the W Register. Data to be transmitted on the I/O Data Bus (D Bus) must first be placed on the A Bus.

1.3.4.2 S Bus. The S Bus is the second of the two principle data paths within the computer processor. The S Bus receives the output of the A, X, P, W, and M Registers. It also is the internal bus for data received from the D Bus, via the I/O Data Bus Receivers. The S Bus transmits data received from any of these sources to the Adder.

1.3.4.3 MD Bus. The MD Bus is a bi-directional data bus that connects the W Register with the computer memory modules. Data to be written into memory is first placed in the W Register via the A Bus. It is then carried to the memory modules via the MD Bus. Data read from memory is placed on the MD Bus for transmission to the W Register.

1.3.4.4 MR Bus. The MR Bus carries addressing information from the M Register to the memory modules. All memory addresses, whether for data or instructions, must first be placed in the M Register and carried to memory via the MR Bus.

1.3.4.5 W Bus. The W Bus connects the W Register with the R Register and the I Register. Words read from memory are usually computer instructions or data to be processed (operands). Instructions are loaded into the I Register for execution, and operands are loaded into the R Register for processing. The W Bus is the path for carrying instructions from the W Register to the I Register, and operands from the W Register to the R Register.

1.3.4.6 U Bus. The U Bus provides the second input to the Adder. It receives data from the R Register and the I Register, and transmits that data to the Adder for processing.

1.3.5 Shift Control

As shown in Figure 1-4, data passing from the Adder to the A Bus must pass through the processor Shift Control. Shift Control has the ability to pass data unchanged, shift data left, shift data right, and rotate data left or right. Specific shift instructions and timing considerations are discussed in Section 2 of this manual. The following paragraphs briefly describe the control functions involved.

1.3.5.1 Shift Gates. The shift gates for each bit position of the sum produced by the Adder have the capability of shifting data one bit left, one bit right, or passing data without being shifted. If data is to be shifted more than one bit position, it must be passed through the adder and shift gates once for each bit position that it is to be shifted.

1.3.5.2 Shift Timing. Computer instructions allow shifts of up to eight bit positions for single-register shifts, and up to sixteen bit positions for double-register shifts. Since the shift gates can handle shifts of only one bit position each time data is passed through them, the processor must pass data through them once for each bit position to be shifted. The processor must "stretch" the computer execution cycle to accommodate the extra shifts. For single-register shifts, the cycle must be stretched by 1/4-cycle for each additional bit position that is to be shifted. For example, a shift of one bit position requires one cycle. A shift of two bit positions requires 1-1/4 cycles, and a shift of three bit positions requires 1-1/2 cycles.

Double-register shifts require that data from two registers be passed through the Adder and shift gates sequentially, therefore additional stretching is required. An additional 1/4 cycle stretch is required for each bit position shifted for double-register shifts. For example, a shift of one bit position requires 1-1/4 cycles. A shift of two bit positions requires 1-3/4 cycles, and a shift of three bit positions requires 2-1/4 cycles.

Shift timing is discussed in more detail in Section 2 of this manual.

1.3.6 I/O Control and Data Paths

A mini computer is of little or no use unless it can communicate with those who use it. Communication and control functions are accomplished through peripheral devices of some sort. Devices such as Teletypewriters provide a means for entering information into and receiving information from the computer. Devices such as Analog-to-Digital (A/D) and Digital-to-Analog (D/A) converters provide a means for the mini computer to monitor or control external functions such as measuring devices or assembly lines.

Peripheral devices generally bear little resemblance to the computer with which they must communicate. They differ in speed of operation, mode of data transmission, and the language or codes used to represent data. For example, the ALPHA 16 can move a 16-bit data word from the A Register to the X Register in the computer in 1.6 microseconds. An ASR-33 Teletypewriter requires 200 milliseconds to move two bytes (total of 16 data bits) from the teletype to a receiving device. The computer is 125,000 times faster than the teletype in this data move. The ALPHA 16 transmits data in a parallel mode; i.e., it has separate data lines in each data bus for each bit of the data word that is to be moved. The ASR-33 transmits data in a bit-serial mode; i.e., it has only one data line, and transmits each data bit on the same line, one bit following the other, in a serial fashion. The ALPHA 16 uses a voltage level of 0 volts to represent a one-bit on the I/O Data Bus, and a +5 volt level to represent a zero-bit on the bus. The ASR-33 transmits a one-bit as the presence of current flow, and a zero-bit as the absence of current flow.

It is obvious that the ALPHA 16 cannot communicate directly with the ASR-33 Teletypewriter. There appears to be little or no similarity between the two devices. Therefore, for the two to communicate some means must be found to match these two dissimilar devices. The matching is accomplished by an Interface.

1.3.6.1 Interface Control. Figure 1-5 illustrates the relationship between the mini computer, the peripheral interface, and some peripheral device. A peripheral interface is especially designed to match a specific peripheral device to a specific mini computer. The busses which connect the mini computer to the interface must provide sufficient general control signals to permit the matching of the computer to a wide variety of peripheral devices. There are four specific functions which must be accomplished by these control lines:

1. Device Selection — Since a mini computer may be controlling several peripheral devices, some means must be provided to select, or address, a specific device.

2. Function Command — A peripheral device may be capable of performing many different functions. The mini computer must have some means for specifying which function it wants the device to perform.

3. Sense Status — A peripheral device may require a relatively long period of time (as the computer measures time) to complete a function. The computer must be able to determine the availability or functional status of the peripheral device to determine whether or not the device is ready to accept another command.

4. Data Transfer — The ultimate objective of the computer/device hook-up is the transfer of data between the computer and the peripheral device. Data transfer paths must be established between the two devices and the speed of transfer must be controlled so that no data is lost.

The ALPHA 16 and NAKED MINI 16 have a number of control and data transfer modes available for optimum

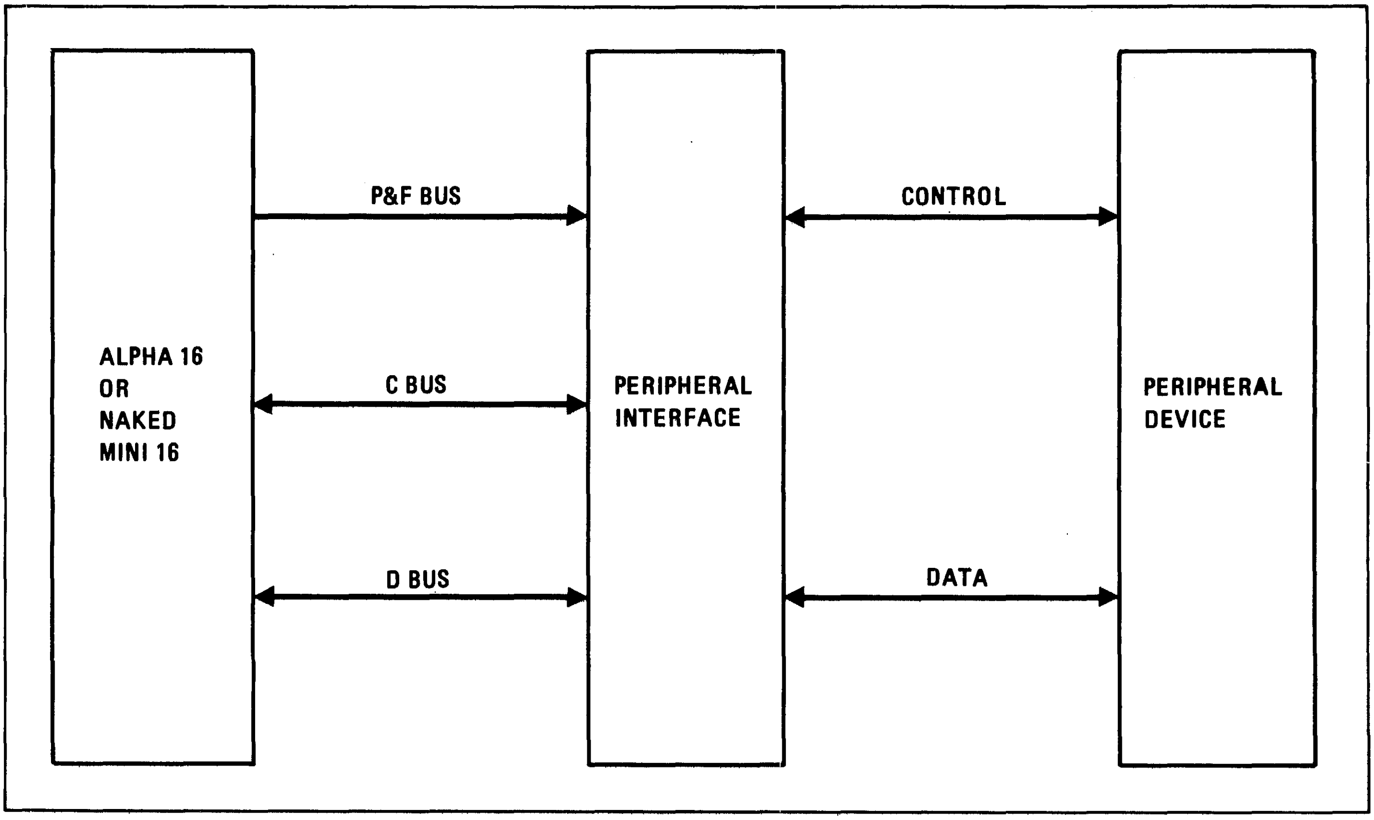


Figure 1-5. I/O Control and Data Paths

data transfer and control functions involving peripheral devices. Computer instructions and general timing considerations are discussed in Section 3 of this manual. Detailed interfacing considerations are discussed in the ALPHA 16 and NAKED MINI 16 INTERFACE MANUAL. The following paragraphs briefly describe the busses which connect the mini computer to the peripheral interface.

1.3.6.2 P&F Busses. The eight least significant bits of the I Register drive the P and F busses. These bits are used as a device address and a function code for Input/Output instructions. These eight bits are arbitrarily divided into two busses. The P Bus contains five bits and is the Device Address bus. Five bits give the computer the ability to address up to 32 different devices. The F bus contains three bits and is the Function Code bus. Three bits give the computer the ability to specify any one of eight functions for the selected device to perform.

The separation of the eight bits into two busses is purely an arbitrary separation. There is really no difference between the lines that drive the P Bus and those that dirve the F Bus. They may be considered to be a single eight-bit Device Address bus capable of addressing up to 256 devices, where each function within a device is assigned a different address.

1.3.6.3 C Bus. The C Bus contains all the control lines connecting the computer and the peripheral interface. These are individual control lines such as interrupt lines, sense response lines, and timing lines.

1.3.6.4 D Bus. The D Bus is a 16-bit data bus used for the parallel transfer of data between the peripheral interface and the mini computer. Data transmission may be in either 16-bit words or 8-bit bytes. The D Bus is a bi-directional bus; i.e., it is used for data transmission from the computer to the peripheral interface, and from the peripheral interface to the computer.

1.3.7 Instruction Execution Sequences

Figure 1-4 illustrates the internal configuration of the ALPHA 16 and NAKED MINI 16 processor. Registers and data paths are shown, but the use of these registers and data paths may not be readily apparent. The purpose of the following paragraphs is to present several examples of internal computer operations so that the functions of the internal computer registers and busses may be more easily understood.

1.3.7.1 Instruction Cycle. Before any computer instruction can be executed, it must first be read from memory and then loaded into the computer Instruction Register (I Register) for decoding and execution. In order to get the instruction from the computer memory, the location of the instruction must be known. The Program Counter (P Register) contains the address of the next instruction to be executed. The sequence of events which must occur during the instruction cycle are:

1. (P) → M — The contents of the P Register, written (P), are gated onto the S Bus and applied to the Adder inputs. (P), via the S Bus, are passed through the Adder and Shift Control unchanged, and are placed on the A Bus. A signal is generated to load the contents of the A Bus, containing (P), into the M Register for use as an address to memory.

2. Start Read Cycle — Processor control logic generates a signal to Memory to read the location addressed by (M).

3. (P) + 1 → P — The P register must be incremented to point to the next instruction to be executed. While the memory is performing the read cycle, (P) are gated onto the S Bus and applied to one input to the Adder. The U Bus is forced to all zeros, and a Carry In is generated to provide a +1 to the Carry In input to the Adder. The sum at the output of the Adder is then (P) + 1. The sum is passed through Shift Control without change and is applied to the A Bus. Processor control then generates a signal to store (A Bus) into the P Register, completing the incrementing of the P Register.

4. Instruction → W — When the memory read cycle is completed, the word read is placed on the MD Bus, and is then loaded into the W Register. At this point, the W Register contains the instruction to be executed.

5. Instruction → I — (W) are placed on the W Bus and then applied to the inputs to the I Register. Processor control logic generates a signal to load (W Bus) into I where it can be decoded and executed as a computer instruction.

The only distinction between instructions and data in memory is that instructions are addressed by the P Register and are placed into the I Register for decoding and execution. If the P Register should contain the address of a data word rather than an instruction, the computer would attempt to recognize that data word as an instruction and would attempt to execute it as an instruction.

1.3.7.2 Register Load. A common function within the computer is the loading of either the A or X register with a data word from memory. The instruction to load the A Register (or X Register) must first be read and decoded. The load sequence, after the instruction is decoded, is accomplished as follows:

1. Address → M	An effective data address must be formed using the eight least significant bits of the instruction word and possibly some other information. The exact manner of address formation is discussed in Section 2 of this manual. The address appears on the A Bus and is stored in the M Register for use as an address to memory.
2. Start Read cycle	Processor control generates a signal to memory to read the memory cell addressed by the M Register.
3. Data → W	When the read cycle is completed, the data word is placed on the MD Bus and loaded into the W Register.
4. Data → R	The W Register places the data on the W Bus where it is applied to the inputs to the R Register. A signal from Processor Control causes the (W Bus) to be stored in the R Register.
5. Data → A or X	(R) are placed on the U Bus and applied to the inputs to the Adder. (U Bus) are passed unchanged through the Adder and Shift Control, and are applied to the A Bus. A signal from Processor Control causes (A Bus) to be stored in the A or X register, completing the load operation.

1.3.7.3 Add. A common arithmetic function in the computer is the addition of a word in memory to (A), with the results stored in the A Register. This is the addition of two values, where one value is in the A Register and the other value is in some word in memory. The two values are added together and the sum is stored in the A Register. The instruction to perform the add operation must first be read and decoded. The data word must then be addressed and read into the W Register. The following sequence of events describes the operation after the data word has been stored in the W Register:

1. (W) → R	The data word is palced on the W Bus and stored in the R Register by a signal from Processor Control.
2. (R) → Adder	The data word from memory, now in the R Register, is placed on the U Bus and applied to one set of Adder inputs.
3. (A) → Adder	The second value to be added is in the A Register. (A) are gated to the S Bus and applied to a second set of Adder inputs.
4. Sum → A	The Adder is a passive device which always produces the sum of all of its inputs. (S Bus) are added to (U Bus) in the Adder and the

result is applied to the Shift Control where it is passed unchanged to the A Bus. (A Bus) are then applied to the inputs to the A Register where a signal from Processor Control stores the sum in A.

1.3.7.4 Shift. A simple single-register shift involves the movement of all 16 bits of a data word either left or right one or more bit positions. For purposes of this example, it is assumed that the word to be shifted is in the A Register, and that the shift instruction is in the I Register and has been decoded. The sequence of events is as follows:

1. (A) → Adder	The word to be shifted is gated from the A Register onto the S Bus and is applied as an input to the Adder. The word is passed unchanged through the Adder.
2. Shift one bit position	A control signal from Processor Control causes the word to be shifted one bit position in the direction specified by the shift instruction. Shift Control accomplishes the shift. The shifted data word is then applied to the A Bus.
3. (A Bus) → A	The shifted data word on the A Bus is then stored in the A Register by a signal from Processor Control.
4. Check Shift Count	The shift count is then checked for the completion of the shift instruction. If all shifts have been completed, the instruction is terminated. If more shifts must be performed, the sequence is repeated until all shifts have been completed.

1.3.7.5 Register Change. Register change instructions perform logical operations or simple moves between registers. The logical operation that will be illustrated is the logical product, or AND, of the contents of the A and X registers, written symbolically as

$$(A) \wedge (X) \rightarrow A$$

where each bit of the A Register is logically ANDed with the corresponding bit of the X Register, and the result is stored in the A Register.

The logical product is formed on the S Bus. The S Bus is a positive true bus, with zero levels predominating. That is, if a logical one and a logical zero are simultaneously gated onto the S Bus in the same bit position, the logical zero will predominate and the S Bus will contain a logical zero in that bit position. If two logical ones are gated onto the S Bus in the same bit position, a logical one will appear on the S Bus in that bit position. Therefore, the S Bus may be used to perform an AND of two registers which are gated onto the bus at the same time.

The sequence for performing the logical product of the A and X registers is:

1. (A), (X) → S Bus	Processor Control gates the A Register and the X Register onto the S Bus simultaneously, and the logical product of the two registers is formed, bit by bit.
2. (A) ∧ (X) → A	(S Bus) is applied to one input to the Adder, and is passed through the Adder and Shift Control unchanged onto the A Bus. Processor Control then generates a

signal to store (A Bus) in the A Register, completing the operation.

1.3.8 Data Word Format

Processor registers and memory word locations are capable of storing data words consisting of 16 binary digits, or "bits." A word may be handled as a single 16-bit field, or as two 8-bit bytes. The following paragraphs describe the word format of the computers. Byte format is described later in this section.

1.3.8.1 Bit Identification. A data word may contain a single number, or it may contain a string of individual binary bits, with each bit having a unique meaning. For purposes of explanation and identification, each bit within a word is uniquely identified. The identification is accomplished by numbering each bit within a word from right to left. The bit on the extreme right of the word is bit 0, and the bit on the extreme left is bit 15. Figure 1-6 illustrates the format of a 16-bit data word with the bit number shown above the bit position.

1.3.8.2 Bit Values. The ALPHA 16 and NAKED MINI 16 are binary computers, therefore numeric information stored in the computer and processed by the computer must be in binary format. Figure 1-6 illustrates the binary value of a one-bit in each bit position of the 16-bit data word. These values are expressed as powers of two. For example, a one-bit in bit position 3 has the value of 2^3, or 8. Note that the bit position identification number is the same as the exponent of 2 for the value of a one-bit in that bit position. The single exception to this rule is bit position 15.

1.3.8.3 Signed Numbers. The ALPHA 16 and NAKED MINI 16 are capable of performing arithmetic operations with signed numbers. Binary two's complement notation is used to represent and process numeric information. Bit 15 of a data word indicates the algebraic sign of the number contained within that word.

1.3.8.4 Positive Numbers. A positive number is identified by a 0 in bit 15, and the binary equivalent of the magnitude of the positive number is stored in bits 0 - 14. For example:

Digital Number	Binary Signed Word
	S Magnitude
+5	0 000 0000 0000 0101
+32	0 000 0000 0010 0000
+585	0 000 0010 0100 1001

In the examples above, the decimal value of the binary number is obtained by adding the values of each bit position containing a one-bit. For example:

$$(+585)_{10} = (0000001001001001)_2$$

The binary number contains one-bits in positions 0, 3, 6, and 9. Therefore:

$$2^0 = 1$$
$$2^3 = 8$$
$$2^6 = 64$$
$$2^9 = \underline{512}$$
$$\text{Total} = 585$$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Figure 1-6. Data Word Bit Identification

The largest positive signed number which can be stored in a 16-bit word is +32,767. The binary equivalent of this number is: 0111 1111 1111 1111.

Note that positive numbers contain a 0-bit in the sign bit position, and generally have 0-bits preceding the most significant 1-bit.

1.3.8.5 Negative Numbers. A negative number is identified by a 1 in bit 15 of the data word. A negative number is represented by the binary two's complement of the equivalent positive number. A negative number must follow the mathematical rule where:

$$0-(+n)=-n$$

For example:

$$0-(+5)=-5$$

Negative numbers must also be constructed such that:

$$(+n)+(-n)=0$$

The binary two's complement of some numeric value may be constructed by subtracting the binary representation of the absolute magnitude of that value from 0. For example:

+5 = 0000 0000 0000 0101

Subtracting from 0:

```
  0000 0000 0000 0000
- 0000 0000 0000 0101
  -------------------
  1111 1111 1111 1011  = -5
```

To satisfy the condition that (+n) + (–n) = 0:

```
  0000 0000 0000 0101  = +5
+ 1111 1111 1111 1011  = -5
  -------------------
  0000 0000 0000 0000  = 0
```

Note that the formation of a binary two's complement negative number from the equivalent positive number automatically sets the sign bit to a one. Binary two's complement negative numbers generally have 1-bits preceding the most significant 0-bit.

It was shown above that binary two's complement numbers may be formed by subtracting the corresponding positive number from a binary zero. Since the computer does not have the ability to subtract, other than through the addition of a binary two's complement number to a positive number, some other method must be used to form two's complements. A characteristic of binary numbers is that the one's complement of a binary number can be formed by substituting 0-bits for all 1-bits in the number, and substituting 1-bits for all 0-bits in the number. For example:

+ 5 = 0000 0000 0000 0101

One's complement:

$\overline{+5}$ = 1111 1111 1111 1010

The two's complement is then formed by adding +1 to the one's complement:

```
+5(bar) = 1111 1111 1111 1010
          +                 1
          -------------------
     -5 = 1111 1111 1111 1011
```

1.3.8.6 Arithmetic Operations. When a negative number, represented by a binary two's complement, is added to a positive number, the sum is the actual difference between the two numbers. For example:

```
   +6 = 0000 0000 0000 0110
+(-4) = 1111 1111 1111 1100
        -------------------
  Sum = 0000 0000 0000 0010 = +2
```

In the above example the sum of the positive and negative numbers is positive because the absolute magnitude of the positive number is greater than the absolute magnitude of the negative number. The following example illustrates the results where the negative number is greater in absolute magnitude than the positive number:

```
   +4 = 0000 0000 0000 0100
+(-6) = 1111 1111 1111 1010
        -------------------
  Sum = 1111 1111 1111 1110 = -2
```

When two negative numbers are added, or when a positive number is added to a negative number which has a larger absolute magnitude, the sum is a binary two's complement number.

1.3.8.7 Word Processing. The ALPHA 16 and NAKED MINI 16 computers have the ability to place the one's complement of certain registers on the output busses connected to those registers. The one's complement of the A or X registers can be placed on the S Bus. The one's complement of the R Register can be placed on the U Bus. Refer to figure 1-4. The three inputs to the Adder are (1) the S Bus, (2) the U Bus, and (3) the Carry In.

Adds and subtracts in the processor are accomplished by controlling these three inputs to the Adder. For example, one number may be subtracted from another by adding its two's complement to the number from which it is to be subtracted. The SUB (subtract) instruction in the computer subtracts a value that is stored in memory from a value in the A Register. The difference is stored in A. For normal subtract operations, the value in memory is usually represented as a positive value. This is not a requirement, however, because the subtract may be used to subtract positive or negative numbers from positive or negative numbers. The result will be algebraically correct. The computer operations are as follows:

1. (Memory) → R — The number to be subtracted from (A) is stored in the R Register during the computer operand cycle.

2. ($\overline{R}$) → U Bus — The one's complement of the R Register is gated to the U Bus. For example, if R contains +5, then:

   R = +5 = 0000 0000 0000 0101

   U Bus = $\overline{+5}$ = 1111 1111 1111 1010

3. Carry In → Adder — An initial Carry In is generated and added to the other inputs to the Adder. The carry in, added to the ($\overline{R}$) negates (R). The inputs to the Adder at this point are:

   U Bus = ($\overline{R}$) = 1111 1111 1111 1010

   Carry In = 1 = 1

   Sum = -5 = 1111 1111 1111 1011

4. (A) → S Bus — The absolute binary value of the A Register is gated to the S Bus and applied as an input to the Adder. If the A Register contains +10, the three inputs to the Adder are:

   U Bus = ($\overline{R}$) = 1111 1111 1111 1010

   Carry in = 1 = 1

   S Bus = +10 = 0000 0000 0000 1010

   Sum = +5 = 0000 0000 0000 0101

5. Sum → A — The sum at the output from the Adder is the sum of the three inputs. The sum is passed through Shift Control unchanged and is stored in the A Register.

1.3.9 Data Byte Format

A 16-bit data word is capable of storing two 8-bit bytes. Since most data transfers between mini computers and peripheral devices are in the form of bytes rather than words, the ALPHA 16 and NAKED MINI 16 computers provide the capability of addressing individual bytes as well as full data words. Figure 1-7 illustrates the storage of two bytes within one computer word.

Bit positions within bytes are identified much the same as in 16-bit words. Figure 1-7 also illustrates the numbering of data bits within a byte. The bits are numbered 0 through 7, where bit 0 is the least-significant bit (LSB), and bit 7 is the most-significant bit (MSB) of the byte.

1.3.9.1 Byte Mode Processing. There are two control instructions in the computer which control Word Mode processing and Byte Mode processing. One of the

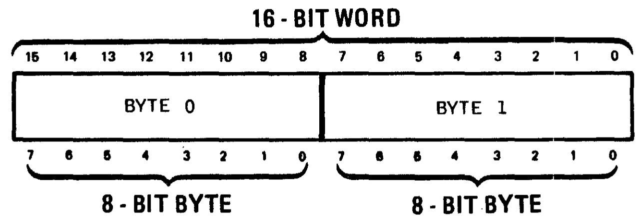


Figure 1-7. Byte Storage, Two Bytes Per Word

instructions causes the computer to enter Byte Mode processing, and the other causes the computer to enter Word Mode.

In Word Mode all memory reference instructions access full words in memory. In Byte Mode all memory reference instructions (except IMS, SCN, JMP, and JST) access one byte within a word. The method of addressing individual bytes is discussed in a subsequent part of this Section. The present discussion is concerned with computer operations while in Byte Mode as contrasted with computer operations in Word Mode.

Byte Mode affects the operand cycle of the computer only. All other computer functions operate the same as in Word Mode. In Byte Mode the computer operand cycle reads a single byte from memory instead of a full word. The following paragraphs illustrate Byte Mode operations for memory reference instructions.

1.3.9.2 Register Load. In Word Mode, a register load instruction causes a full 16-bit word in memory to be read and stored in a 16-bit register in the computer. In Byte Mode one byte within a word in memory is read and stored in the lower eight bits of the computer register. The upper eight bits are set to zeros. For example:

	Byte 0	Byte 1
Memory Word =	1001 0110	1111 0000

Load A with Memory Word:	A = 1001 0110 1111 0000
Load A with Byte 0:	A = 0000 0000 1001 0110
Load A with Byte 1:	A = 0000 0000 1111 0000

In Word Mode the full word is loaded into the selected register. In Byte Mode the selected byte is loaded into the lower eight bits of the selected register, and the upper eight bits are cleared. Note that the location of the byte within the memory word does not determine the location the byte will occupy in the register being loaded.

1.3.9.3 Arithmetic Operations. For arithmetic purposes, bytes are handled as positive numbers only. The reason is that a byte occupies the lower eight bits of a register or a data bus, and the upper eight bits are logical zeros. Consider an ADD operation as an example. In Byte Mode, the selected byte is added to the contents of the A Register. The byte occupies the lower eight bit positions of the U Bus during the addition, and (A) occupies the full S Bus:

	Byte 0	Byte 1
Memory Word:	1001 0110	1111 0000

A Register: 0000 1100 0111 1100

Add Byte 0 to A Register:

S Bus =	0000 1100 0111 1100
U Bus =	0000 0000 1001 0110
Sum =	0000 1101 0001 0010

The addition is handled in the computer as the addition of two 16-bit words, with the word from memory containing significant data in the eight least-significant bit positions only.

A subtract operation subtracts the absolute magnitude of the selected byte from the value in the A Register. To understand the functions of the subtract in Byte Mode it must be remembered that the operand goes from memory to the R Register and is then placed on the U Bus in one's complement form:

Memory Word: 1001 0110 1111 0000
Byte 0 Byte 1

A Register: 0000 1100 0111 1100

Subtract Byte 0 From A Register:

R Register = 0000 0000 1001 0110

The one's complement of the R Register is gated to the U Bus:

U Bus =	1111 1111 0110 1001
Carry In =	1
S Bus = (A) =	0000 1100 0111 1100
Sum =	0000 1011 1110 0110

The subtract is performed as an operation involving two 16-bit numbers. The byte being subtracted occupies the lower eight bits of the word being subtracted from the contents of the A Register.

1.3.9.4 Data Packing. One of the most useful features of byte mode processing is in the packing and unpacking of data in memory. Since most of the peripheral devices used with mini computers are byte oriented, high-speed data transfers between the computer and the peripheral device generally require data to be packed one byte per word. Such an arrangement is illustrated in Figure 1-8. In this illustration, the upper eight bits of each data word to be transmitted to a peripheral device contain zeros. A full 16-bit word is transmitted to the device, but the device discards the upper eight bits and accepts only the lower eight bits. Data received from a byte oriented peripheral device during high-speed data transfers is packed in memory one byte per word in the format shown in Figure 1-8. If a software subroutine were required to pack the data two bytes per word, in the format illustrated in Figure 1-9, it would waste memory and time in performing the formatting required for high-speed data transfers.

The capability of the ALPHA 16 and NAKED MINI 16 computers to address individual bytes in memory allows high speed data transfers using the memory format shown in Figure 1-9 for both transmission and reception of data. Bytes may be addressed sequentially and transmitted or received sequentially, just as words are transmitted or received sequentially in conventional unpacked data transfers. This arrangement saves memory space since none of the memory word is wasted, and it saves time since no software routines are required to pack and unpack data for internal processing.

1.3.10 Memory Address Formats

Maximum memory capacity in the ALPHA 16 and NAKED MINI 16 computers is 32,768 words, which means a byte capacity of 65,536 bytes. A fifteen bit address is required to address 32,768 words, and a sixteen bit address is required to address 65,536 bytes. The following paragraphs discuss the formats of the addresses that must be presented to memory for addressing both words and bytes. This discussion is concerned only with address formats. Section 2 of this manual discusses the memory address modes which form these addresses.

1.3.10.1 Word Addressing. Figure 1-10 illustrates the format of an address presented to memory to address a full word. This is the format that is used to address instructions or full data words. The address is contained in bits 0 – 14, and bit 15 contains a zero.

1.3.10.2 Byte Addressing. Figure 1-11 illustrates the format used to address a byte within a data word. Bits 1 – 15 contain the address of the memory word, and bit 0 specifies which byte within the word is to be addressed.

Bit 0 = 0 specifies Byte 0 (Most Significant Byte).

Bit 0 = 1 specifies Byte 1 (Least Significant Byte).

If the computer is set for Byte Mode, all operand addresses presented to memory are assumed to be byte addresses.

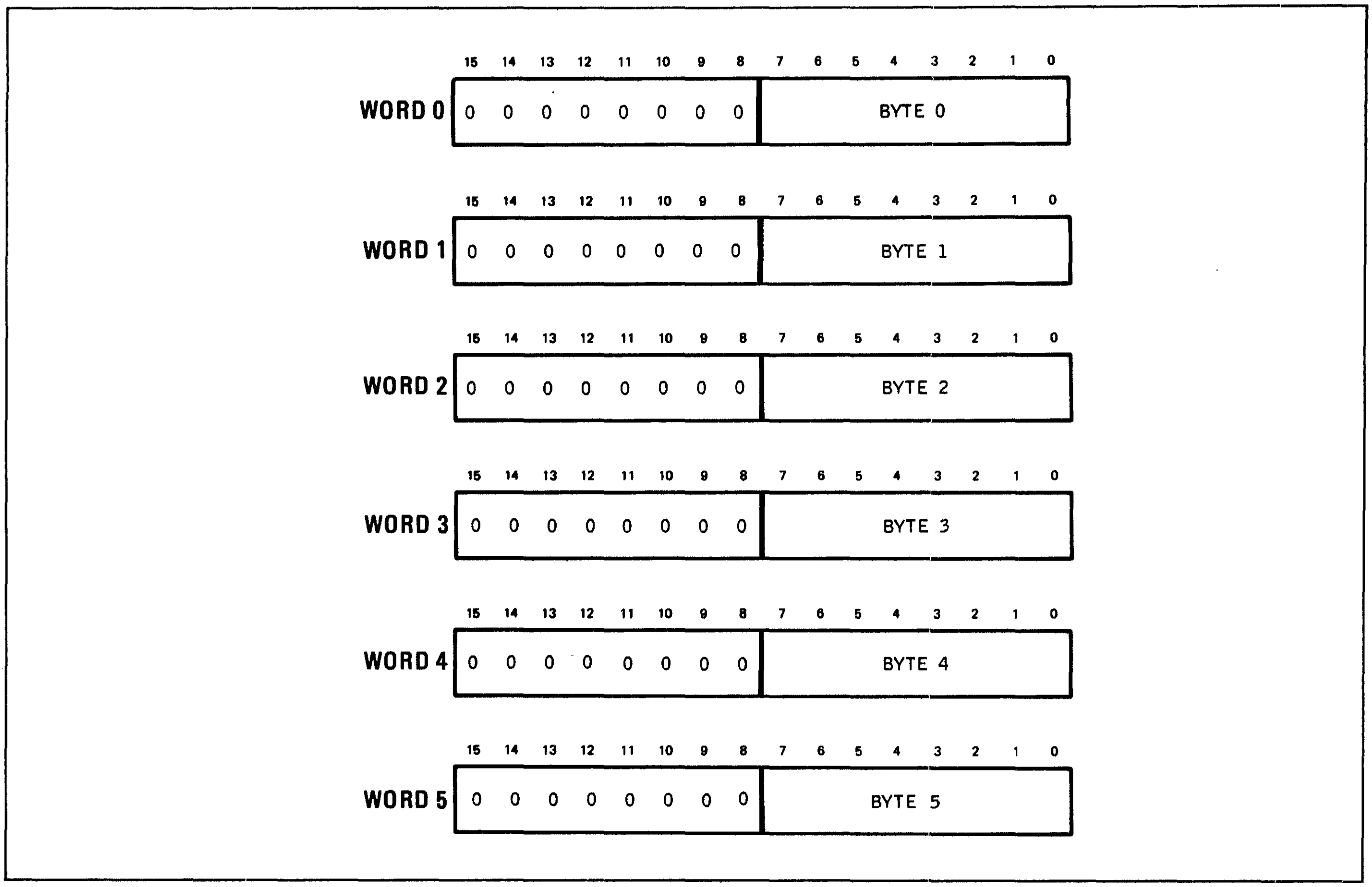


Figure 1-8. Data in Memory, One Byte Per Word

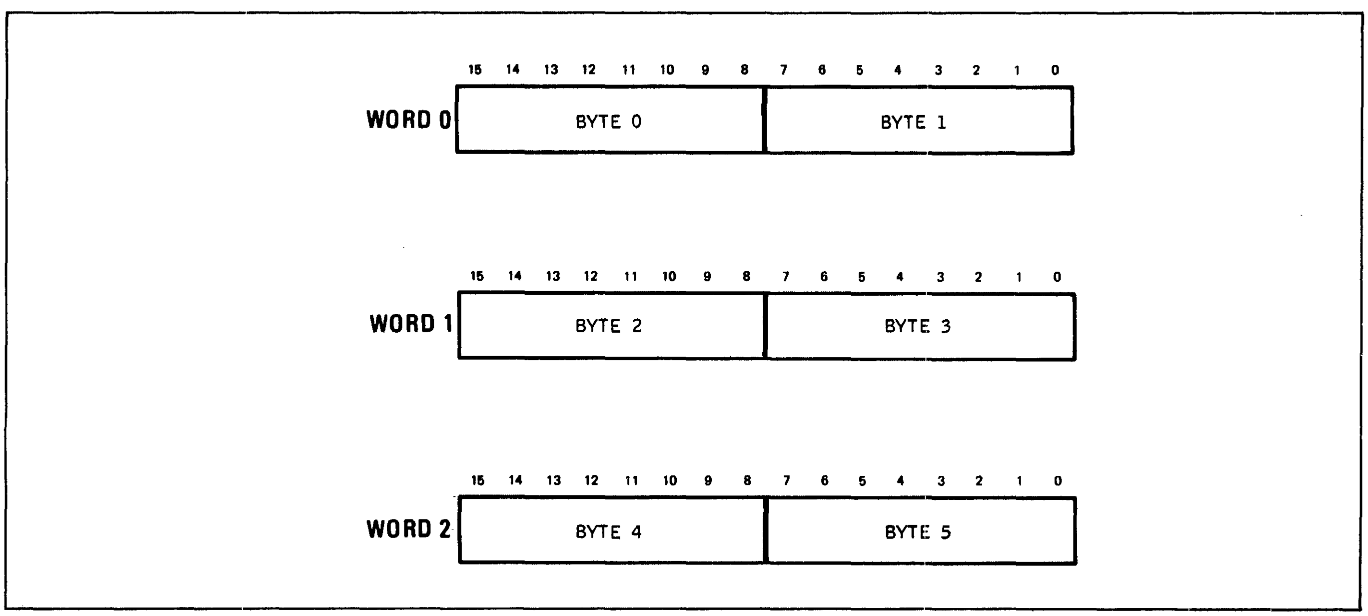


Figure 1-9. Data in Memory, Two Bytes Per Word

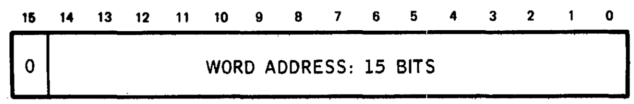


Figure 1-10. Basic Word Address Format

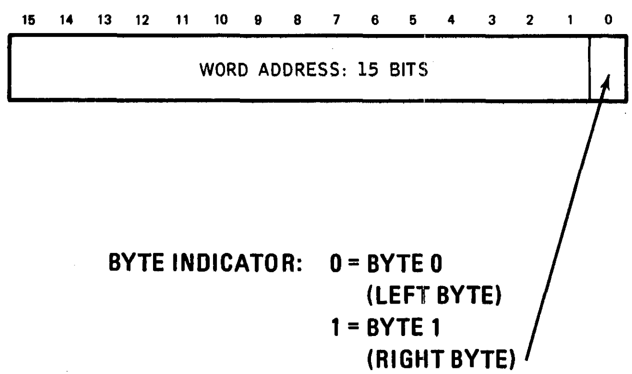


Figure 1-11. Byte Address Format

The computer assumes that the address is in the format shown in Figure 1-11. If the computer is set for word mode processing, all addresses presented to memory are assumed to be word addresses in the format shown in Figure 1-10. These assumptions apply to operand cycles only. They do not apply to instruction cycles or indirect addressing cycles.

1.3.10.3 Indirect Addressing. The ALPHA 16 and NAKED MINI 16 computers are capable of performing single level indirect addressing for addressing bytes, and multi-level indirect addressing for addressing words. Indirect addressing uses direct addressing to read a word in memory, called an Address Pointer, which contains the address of another word. In Byte Mode the Address

Pointer contains the address of the byte to be addressed. The format of the address in the Address Pointer is the same as that shown in Figure 1-11.

In Word Mode the format of the address in the Address Pointer is that shown in Figure 1-12. Bits 0 – 14 contain the address of another word in memory. Bit 15 is a multi-level indicator. If bit 15 contains a 0, the address in bits 0 – 14 is the address of an operand. If bit 15 contains a 1, the address in bits 0 – 14 is the address of another indirect Address Pointer. The number of levels of indirect addressing which may be used is limited only by the size of memory.

1.3.11 Control Console

Figure 1-13 illustrates the ALPHA 16 Control Console. The NAKED MINI 16 does not have a console, so the description that follows applies to the ALPHA 16 only.

The Control Console contains register display lights, data entry switches, register select switches, and various control switches and indicators. Functions of the switches and indicators are explained in the following paragraphs. Refer to Figure 1-13 for the location of each component.

1.3.11.1 Register Display. The Register Display lights are 16 light emitting diodes which display the contents of a selected register when the computer is halted. The A, X, I, and P registers may be displayed. When the computer is running the contents of the A Bus are displayed.

1.3.11.2 Data Entry Switches. Sixteen latching switches are provided for entering data into computer registers. Data is entered into the selected register by entering the data in the entry switches and depressing the ENTRY switch. A 1-bit is entered when a data switch is down, and a 0-bit is entered when a data switch is up. Data can be entered only when the computer is halted and the STOP switch is down (STOP position).

The four least significant data switches, switches 0 thru 3, may be examined by software and may be used as sense switches for operator interface to the operating program. These four switches may be read by a computer instruction and their settings stored in either the A or X register for software examination.

1.3.11.3 Register Select Switches. Four Register Select switches are provided to select the A, X, I, and P registers for data entry or display. A register is selected for data

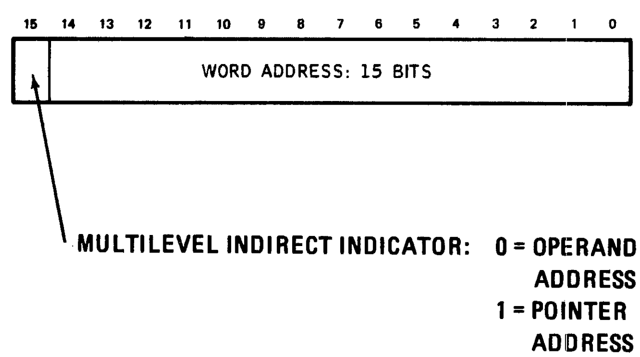


Figure 1-12. Indirect Address Pointer Format

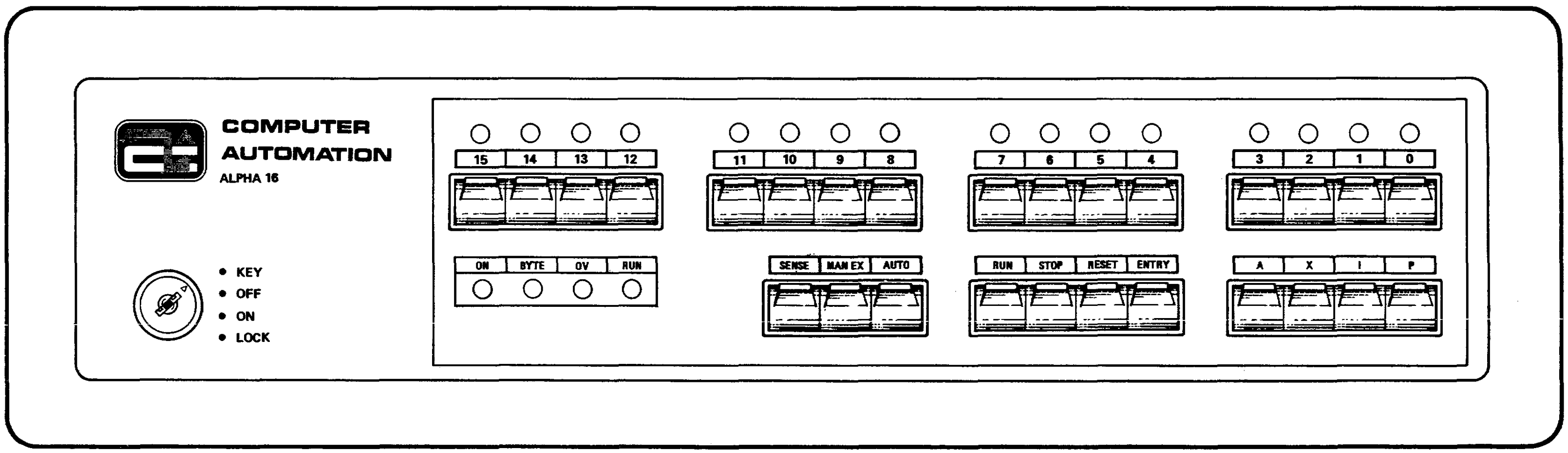


Figure 1-13. ALPHA 16 Control Panel

entry or display when the associated switch is down, and no switch of higher priority is down.

The switches are wired in a priority series with each switch having priority over the Register Select switches to its right. Switch A has priority over the X, I, and P switches. Switch X has priority over the I and P switches. Switch I has priority over the P switch, and switch P has no priority.

These switches are not automatic return switches. When one switch is depressed, it remains depressed until lifted by the operator. If two switches are down simultaneously, the register selected by the higher priority switch is displayed and the register selected by the lower priority switch is ignored. In practice, the P switch may be left down at all times, since the P register can then be selected by lifting higher priority switches.

1.3.11.4 <u>ENTRY Switch.</u> The ENTRY switch is a momentary switch used to load the contents of the data entry switches into the selected register. The switch is activated when depressed, and automatically returns to the inactive position when released. This switch is totally disabled when the STOP switch is in the RUN position (STOP switch up).

1.3.11.5 <u>RUN Switch.</u> The RUN switch is a momentary switch which causes the computer to execute one instruction if the STOP switch is down (STOP position), or enter the RUN mode if the STOP switch is up (RUN position).

1.3.11.6 <u>AUTO LD Switch.</u> The AUTO LD switch is a momentary switch which initiates the Autoload sequence (if the Autoload option is included in the system). The Autoload sequence can be entered only if the STOP switch is up (Run position) and the computer is not in the RUN mode; i.e., the STOP switch must be in the run position and the RUN Mode indicator must be off.

If the computer is in the Run mode, the AUTO LD switch has a different function. If the computer is running and the AUTO LD switch is depressed, a console interrupt is generated. The computer is interrupted to location: 1E in memory, where it will execute the instruction at the interrupt location. (Refer to Section 3 for interrupt processing.)

1.3.11.7 <u>SENSE Switch.</u> The SENSE switch is a latching switch which provides operator interface to an operating program. This switch differs from the four data entry switches which may be examined by software in that the SENSE switch may be tested directly for conditional program branches according to the setting of the switch.

1.3.11.8 <u>MAN EX Switch.</u> The MAN EX (Manual Execute) switch is a latching switch which, when down, locks an instruction in the I Register, provided the STOP switch is also down. The instruction in the I Register is then executed once each time the RUN switch is depressed. The MAN EX switch is used primarily for displaying data in memory and entering data into memory from the control console. Refer to the Console Display Procedure (Paragraph 1.3.12.6) and the Console Load Procedure (Paragraph 1.3.12.5) for Manual Execute switch functions.

1.3.11.9 <u>RESET Switch.</u> The RESET switch is a momentary action switch which, when depressed, initializes the processor control flip-flops, resets the OV (Overflow) indicator, resets the BYTE (Byte Mode) indicator, and sends an initialize pulse to all peripheral interfaces. This switch does not affect processor registers.

1.3.11.10 <u>ON Indicator.</u> The On indicator is a light emitting diode which is illuminated when power is applied to the computer.

1.3.11.11 <u>RUN Indicator.</u> The RUN indicator is a light emitting diode which is illuminated when the processor is in the Run mode; i.e., when the processor is executing computer instructions. It is turned off when the processor executes a Halt instruction. If the processor is in the Run mode and the STOP switch is depressed, the processor will leave the Run mode and the RUN indicator will be turned off at the end of the instruction being executed at the time the STOP switch was depressed.

1.3.11.12 STOP Switch. The STOP switch is a latching switch which puts the computer in the Stop mode when depressed. With the STOP switch depressed, the computer will execute one instruction each time the RUN switch is depressed. When the STOP switch is depressed, the MAN EX and ENTRY switches are enabled. When the STOP switch is up, the computer Run mode is enabled but is not entered until the RUN switch is depressed. The MAN EX and ENTRY switches are disabled when the STOP switch is up.

1.3.11.13 BYTE Indicator. The BYTE indicator is a light emitting diode which is illuminated when the computer is set for Byte Mode processing. When the computer is set for Word Mode processing, the BYTE indicator is off.

1.3.11.14 OV Indicator. The OV (Overflow) indicator is a light emitting diode which is illuminated when the Overflow flip-flop within the computer is set. The OV indicator is off when the Overflow flip-flop is reset.

1.3.11.15 Key Lock. The Key Lock is a four-position switch which is activated by a key. Two of the four positions are key removal positions (KEY and LOCK). The four positions are:

1. KEY	Power OFF, key removal position.
2. OFF	Power OFF.
3. ON	Power ON, control console enabled.
4. LOCK	Power ON, control console switches disabled except for SENSE switch, the AUTO LD console interrupt, and the four low order data entry switches (data sense switches). Key removal position.

1.3.12 Console Operation

The ALPHA 16 Control Console is used for initial start-up, program debug, and troubleshooting. The primary functions executed at the console are register display and register change, and the display and entry of memory data. The following paragraphs discuss detailed procedures for performing these operations.

1.3.12.1 Hexadecimal Notation. Memory addresses, data patterns, and instruction codes are difficult to work with when expressed in binary machine language. For this reason hexadecimal notation is used as a shorthand notation for binary bit patterns. Table 1-1 is a conversion table for Binary and Hexadecimal numbers, along with their decimal equivalents. Appendix A discusses the hexadecimal number system in detail and provides numerous tables of hexadecimal arithmetic operations. The remainder of this manual makes extensive use of hexadecimal notation. For purposes of clarity, hexadecimal numbers are distinguished from other numbers by a colon (:) preceding the hexadecimal number. For example, :FA35 is a hexadecimal number representing the binary number 1111 1010 0011 0101. Note that the binary number is separated into groups of four binary bits. This facilitates conversion to or from hexadecimal notation.

Table 1-1. Binary, Hexadecimal, and Decimal Conversion

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

1.3.12.2 Console Preparation. There are several common steps that must be performed before any console operations may be attempted. These steps prepare the console and the computer for console operations. The initial steps are:

1. Power ON — Insert the key into the key lock and turn the key lock switch to the ON position. This applies power to the computer and enables the console.

2. Depress STOP — The computer may come up in the Run mode because of a previously loaded program. Depressing STOP causes the computer to leave the Run mode.

   NOTE: In some cases the RUN indicator may remain on after the STOP switch is depressed. This condition may exist when the computer is attempting to execute certain I/O instructions. This does not indicate a malfunction of the computer. When this occurs, step 3 of this procedure will correct the condition.

3. Depress RESET — Depressing RESET puts the computer in word mode and initializes the computer and peripheral interfaces. It forces the termination of any incomplete instructions.

1.3.12.3 Register Display. After the console has been prepared as described in the preceding paragraph, the contents of the A, X, I, and P registers may be displayed in the Register Display indicators. Display procedure is as follows:

1. Select Register — The register to be displayed is selected by depressing the appropriate Register Select switch with all higher priority Register Select switches up. Paragraph 1.3.11.3 discusses the priority of these switches.

2. Read Contents — The contents of the selected register are automatically displayed in the Register Display indicators in binary format.

1.3.12.4 Register Entry. The following procedure is used to enter data into the A, X, I, and P registers from the Control Console.

1. Select Register — Depress the Register Select switch corresponding to the register in which data is to be entered. Be sure all higher priority switches are up.

2. Enter Data in Data Switches — The data to be stored in the selected register is entered in the Data Entry switches.

3. Depress ENTRY — The ENTRY switch loads the contents of the Data Entry switches into the selected register.

1.3.12.5 Console Load Procedure. The Console Load Procedure is used to store data into selected memory locations from the ALPHA 16 Control Console. The general procedure is to address the desired memory location, enter the data that is to be stored in memory into the A Register, and execute an instruction that stores the contents of the A Register into the addressed memory location. The detailed procedure is as follows:

1. Ready Console — Prepare the console and the computer for console operations as described in paragraph 1.3.12.2.

2. Depress MAN EX	When the MAN EX switch is depressed, an instruction can be entered in the I Register from the console for direct execution without performing a memory instruction cycle.
3. :9E00 → I	Enter :9E00 in the I Register. :9E00 is a memory reference instruction that stores the contents of the A Register in the memory location addressed by the P Register.
4. Memory Address → P	The address of the memory location where data is to be stored is entered in the P Register.
5. Data → A	Select the A Register and enter the data word that is to be stored in memory.
6. Depress RUN	When the RUN switch is depressed, the instruction in the I Register is executed. The instruction stores the data word in the A Register into the memory location addressed by the P Register.
7. Sequential Stores	The P Register is automatically incremented each time RUN is depressed. To store data in sequential locations, go back to step 5 for each succeeding word. To store data in a new location, go back to step 4.

1.3.12.6 Console Display Procedure. The Console Display Procedure is used to display words stored in memory. The general procedure is to address the memory location that is to be displayed, and read the contents of that location into the A Register. The contents of the A Register are then displayed in the Register Display indicators. The detailed procedure is as follows:

1. Ready Console	Prepare the console and the computer for console operations as described in paragraph 1.3.12.2.
2. Depress MAN EX	When the MAN EX switch is depressed, an instruction can be entered in the I Register from the console for direct execution without performing a memory instruction cycle.
3. :B600 → I	Enter :B600 in the I Register. :B600 is a memory reference instruction that reads the memory location addressed by the P Register and loads the contents of that location into the A Register.
4. Memory Address → P	The address of the memory location to be displayed is entered in the P Register.
5. Depress RUN	When the RUN switch is depressed, the instruction in the I Register is executed. The instruction reads the addressed memory location and copies it in the A Register. Memory is unchanged.
6. Display A	Display the A Register. The A Register contains a copy of the data stored in the addressed memory location.
7. Sequential Displays	The P Counter is incremented each time RUN is depressed.

Therefore, to display data in sequential locations in memory, go back to step 5 for each subsequent word to be displayed. To display data from another location, go back to step 4.

1.3.12.7 Program Execution. Programs to be executed may be entered into memory by a number of different means. Short programs may be entered using the Console Load Procedure described in paragraph 1.3.12.5. Longer programs may be entered using the Autoload feature or various Loader programs. Regardless of the means used to get a program into memory, the method used to execute that program is generally the same. The Program Counter (P Register) must be set to the starting address of the program, and the computer Run mode must be entered. The following steps are used to start program execution from the Control Console:

1. Ready Console	Prepare the console and the computer for console operations as described in paragraph 1.3.12.2.
2. Start Address →P	Enter the starting address of the program to be executed in the P Register.
3. Release STOP	Lift the STOP switch. This enables Run mode, but does not cause the computer to enter Run mode. NOTE: Releasing STOP also disables the MAN EX switch and the ENTRY switch.
4. Depress RUN	Depress the RUN switch to cause the computer to enter the Run mode. The computer will continue to run until it executes a Halt instruction, or until the STOP switch is depressed.

1.3.12.8 BOOTSTRAP Program. If a machine is not equipped with Autoload, a simple loader program must be entered into memory from the computer console in order to load more complex programs. The standard BOOTSTRAP program is listed below. This program is normally used to load the Binary Loader, BLD/BDP. BOOTSTRAP is a simple program consisting of only eight words. It is designed to read a program from a Teletype paper tape reader, and store the program into memory at a starting location specified by the X Register.

BOOTSTRAP is loaded into memory from the ALPHA 16 console using the Console Load Procedure (paragraph 1.3.12.5). Each instruction is loaded into sequential locations starting at :0FF8 for machines with 4K words of memory, and location :1FF8 for machines with 8K words of memory.

Detailed procedures for using BOOTSTRAP are contained in the BOOTSTRAP program description.

SECTION 2

ALPHA 16 AND NAKED MINI 16 INSTRUCTIONS

2.1 INTRODUCTION

2.1.1 General

The instruction set of the ALPHA 16 and NAKED MINI 16 computers is divided into seven classes:

1. Memory Reference
2. Immediate
3. Conditional Jump
4. Shift
5. Register Change
6. Control
7. Input/Output

The Memory Reference class processes data in either Word Mode or Byte Mode. Because of the significance of these two modes of processing, the Memory Reference class is divided into two subclasses. The subclasses are Memory Reference: Word Mode, and Memory Reference Byte Mode.

Each instruction class and subclass is treated separately in this Section. The general format of each class is explained along with any special considerations. The format and binary bit pattern of each instruction are covered in detail.

2.1.2 Symbolic Notation

Some standard abbreviations and symbols are used in explaining the functions of instructions and the formation of memory addresses. These symbols and abbreviations are listed in Table 2-1.

2.1.2.1 Symbol Usage. It is very important that the usage of the standard symbols in Table 2-1 be thoroughly understood. Most of the symbols are self explanatory, but there are some subtle differences between symbols that should be noted. The following examples illustrate these differences.

Table 2-1. Standard Symbols

+	Addition
-	Subtraction
$\wedge$	Logical AND
$\vee$	Inclusive OR
$\veebar$	Exclusive OR
=	Equals
$\rightarrow$	Transfer
	(a $\rightarrow$ b, a is transferred to b)
—	One's Complement ($\bar{a}$)
-	Two's Complement (-a)
( )	"Contents of" or "the number in"
>	Greater Than
<	Less Than
≥	Greater than or equal to
≤	Less than or equal to
D	Address portion of Memory Reference Instructions
Y	Any Effective Memory Location
A	Accumulator Register
X	Index Register
P	Program Counter (Register)
OV	Overflow Flip-Flop
IOB	Input/Output Bus
AP	Address Pointer
BA	Base Address
BIS	Bit Store
WC	Word Count

1. A — The symbol A refers to the A Register in the computer.

2. (A) — The symbol (A) stands for contents of A; i.e., the value

that is stored in A rather than the A Register itself.

3. Y — Y is an effective memory address. Y represents the value which will be placed in the M register to be used as an address to memory.

5. (Y) — (Y) stands for the contents of the memory location addressed by the value Y.

6. = — This is the conventional mathematical Equality sign. It is generally used to show how some value is derived; e.g., Y = (P) - (D) means that the value Y is derived by subtracting the contents of the D field of an instruction from the contents of the P Register.

7. → — The transfer symbol should not be confused with the equality symbol. The transfer symbol is often used to show where some value is to be stored; e.g., (Y) + (A) → A means that the value derived by adding (Y) to (A) is stored in the A Register after the operation is performed.

8. - (minus) — The two's complement symbol, when applied to some value, means that the two's complement of that value is used in the operation that is being performed; e.g., - (Y) + (A) → A means that the two's complement of (Y) is added to (A) and the result is stored in A.

2.2 ARITHMETIC OVERFLOW

2.2.1 General

Arithmetic Overflow is a condition which occurs when the result of an arithmetic operation exceeds the capacity of the computer Adder. When arithmetic overflow occurs, an indicator called the Overflow (OV) indicator is set. OV is a testable indicator which may be tested by software to determine whether or not arithmetic operations have exceeded the capacity of the Adder.

Part 1.3.8 of Section 1 describes the word format of the computer and binary two's complement arithmetic. The binary two's complement format used by the ALPHA 16 and NAKED MINI 16 dictates the conditions which will cause arithmetic overflow.

2.2.1.1 Sign Bit. In binary two's complement format, the arithmetic sign of a number contained in a word or register is specified by the most significant bit, bit 15, of that memory word or register. The following conditions are true by definition of binary two's complement notation:

1. Bit 15 = 0; the number contained in the word or register is positive.

2. Bit 15 = 1: the number contained in the word or register is negative.

In general, any arithmetic operation which causes the sign of the result to be incorrect will cause OV to be set. The conditions which will set OV are discussed in detail in the following paragraphs.

2.2.1.2 Maximum and Minimum Numbers. For purposes of this discussion, it is assumed that all numbers are integers; i.e., the binary point of all numbers is immediately to the right of bit 0. For integer arithmetic, the following is true.

1. The maximum positive number that can be stored in a word or register is:

    S
    +32,767 = 0 111 1111 1111 1111

2. The smallest number (most negative number) that can be stored is:

    S
    −32,768 = 1 000 0000 0000 0000

   In this case, the sign bit is also a magnitude bit. Note that the binary number is the two's complement of +32,768.

2.2.2 Overflow Conditions

An Overflow occurs when the result of an arithmetic operation exceeds the maximum or minimum number which can be stored in a computer word or register. An Overflow occurs when:

1. Result > +32,767 (0 111 1111 1111 1111)
2. Result < −32,768 (1 000 0000 0000 0000)

2.2.2.1 Addition of Positive Numbers. When the addition of two positive numbers produces a result that is greater than +32,767, OV will be set:

    S
    +32,767 = 0 111 1111 1111 1111
    + (+1) = 0 000 0000 0000 0001
    Sum +32,768 ≠ 1 000 0000 0000 0000
    (Sum = −32,768)

In this case the addition produced a result which is, by definition of two's complement numbers, a negative result (bit 15 = 1). A negative result is impossible when two positive numbers are added, therefore OV is set.

2.2.2.2 Addition of Negative Numbers. When the addition of two negative numbers produces a result that is less than −32,768, OV will be set:

    S
    −32,768 = 1 000 0000 0000 0000
    + (−1) = 1 111 1111 1111 1111
    Sum = −32,769 ≠ 0 111 1111 1111 1111
    (Sum = +32,767)

The addition produced a result which is, by definition, a positive number (bit 15 = 0). A positive result is impossible when two negative numbers are added, therefore OV is set.

2.2.2.3 Subtraction of Negative from Positive. When the subtraction of a negative number from a positive number produces a result that is greater than +32,767, OV will be set:

    S
    +32,767 = 0 111 1111 1111 1111
    − (−1) = 0 000 0000 0000 0001 (Two's complement of −1)
    Sum = +32,768 ≠ 1 000 0000 0000 0000
    (Sum = −32,768)

Subtracting a negative number from a positive number is the same as adding two positive numbers. A negative result is impossible. The result in this case is, by definition, a negative number. Therefore OV is set.

2.2.2.4 Subtraction of Positive from Negative. When the subtraction of a positive number from a negative number produces a result which is less than −32,768, OV will be set:

    S
    −32,768 = 1 000 0000 0000 0000
    − (+1) = 1 111 1111 1111 1111 (Two's complement of +1)
    Sum = −32,769 ≠ 0 111 1111 1111 1111
    (Sum = +32,767)

Subtracting a positive number from a negative number is the same as adding two negative numbers, which cannot produce a positive result. In this case the result is, by definition, positive. Therefore, OV is set.

2.2.3 Computer Determination of Overflow Condition

The ALPHA 16 and NAKED MINI 16 computers examine the carry into bit 15 of the Adder and the carry out of bit 15 of the Adder to determine whether or not OV should be set. If the carries match (carry in and carry out, or no carry in and no carry out), the result is within the capacity of the Adder and OV will not be set. If the carries are different (carry in and no carry out, or carry out and no carry in), the result exceeds the capacity of the Adder and OV will be set.

2.2.3.1 Carry In and Carry Out Condition. This condition results from the addition of two negative numbers, the sum of which is greater than −32,768:

```
             S
            11 111 1111 1111   11   ← Carries
   -5    =   1 111 1111 1111 1011

 +(-5)   =   1 111 1111 1111 1011
            ---------------------
Sum =-10 =   1 111 1111 1111 0110
```

This condition also exists when a positive number is subtracted from a negative number.

2.2.3.2 No Carry In and No Carry Out. This condition results from the addition of two positive numbers, the sum of which is less than +32,767:

```
             S                1 1  ← Carries
   +5    =   0 000 0000 0000 0101

 +(+5)   =   0 000 0000 0000 0101
            ---------------------
Sum =+10 =   0 000 0000 0000 1010
```

This condition also exists when a negative number is subtracted from a positive number.

2.2.3.3 Carry In and No Carry Out. This condition exists when the sum of two positive numbers exceeds +32,767: (OV set)

```
             S
             1 111 1111 1111 111   ← Carries
+32,767  =   0 111 1111 1111 1111
+  (+1)  =   0 000 0000 0000 0001
            ---------------------
             1 000 0000 0000 0000
```

This condition may also occur when a negative number is subtracted from a positive number.

2.2.3.4 Carry Out and No Carry In. This condition occurs when the sum of two negative numbers is less than −32,768: (OV set)

```
             S
             1                     ← Carry
-32,768  =   1 000 0000 0000 0000
+  (-1)  =   1 111 1111 1111 1111
            ---------------------
             0 111 1111 1111 1111
```

This condition may also occur when a positive number is subtracted from a negative number.

2.3 MEMORY REFERENCE INSTRUCTIONS: WORD MODE

2.3.1 General

Memory Reference instructions are those computer instructions which perform arithmetic and logical operations involving data stored in memory and data stored in the operating registers of the computer. The following paragraphs describe memory addressing and the functions of memory reference instructions when the computer is set for word mode processing; i.e., when the memory operand is a full 16-bit word rather than an 8-bit byte.

2.3.2 Memory Addressing: Word Mode

Memory address formats are discussed in Part 1.3.10 of Section 1. Figure 1-10 illustrates a basic word address. This is the format that is used to address computer instructions, Address Pointers for indirect addressing, and full-word operands. The purpose of the present discussion is to describe the various methods used in the ALPHA 16 and NAKED MINI 16 computers to form full-word operand addresses. The addressing modes used are described briefly in Section 1 (Part 1.2.4). They are described in detail in the following paragraphs.

2.3.2.1 Memory Reference Instruction Format. Figure 2-1 illustrates the format for Memory Reference instructions. The Mode Code (M Field), Indirect Tag (I Bit), and D Field (Base address) are used to define the address of an Operand or an indirect Address Pointer.

2.3.2.2 Direct Addressing. If I = 0 (Bit 8 = 0), the addressing mode specified by the M Field defines the address of a memory operand. The operand address may be formed in the following ways (refer to Table 2-1 for symbol definitions): (Figure 2-2 illustrates the memory areas accessed by these addressing modes.)

M = 00 Operand in Scratchpad. The D Field of the instruction contains the operand address:

$$Y = (D)$$

Since the D Field contains only eight bits, the address will have the form:

Y = 0000 0000 xxxx xxxx

(D) are right-justified in the effective address and the upper eight bits of the address word are set to zeros. An eight-bit address field has the capability of addressing 256 words (locations 0 through 255 decimal, or :00 through :FF). This is the only area in memory that can be addressed directly by an instruction located anywhere in memory.

M = 01 Operand Relative to P, Forward. The operand address is formed by adding the value in the D Field of the instruction to the value in the P Register. The addition is performed after the P Register has been incremented during the instruction cycle, so the effective address is defined as:

$$Y = (D) + (P) + 1$$

The address generated has the form:

(D)	=	0000 0000 xxxx xxxx
(P) + 1	=	0xxx xxxx xxxx xxxx
Y		0xxx xxxx xxxx xxxx

This form of addressing accesses memory locations up to 256 locations forward from the memory reference instruction itself. The locations that can be addressed directly by this mode are (P) + 1 through (P) + 256.

M = 11 Operand Relative to P, Backward. The effective address is formed by subtracting the value in the D Field from the value in the P Register. This mode can access the location of the instruction itself, and 255 locations backward from that location:

$$Y = (P) - (D)$$

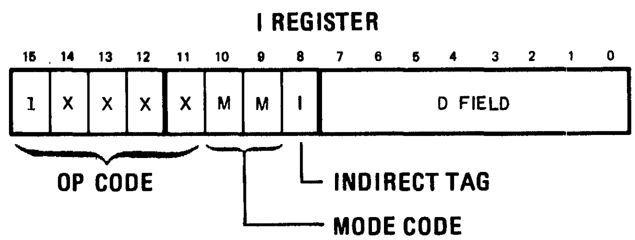
Since the P counter is incremented before the operand address is formed, the address is generated as follows:

$(\overline{D})$	=	1111 1111 xxxx xxxx (One's complement)
(P) + 1	=	0xxx xxxx xxxx xxxx
Y	=	0xxx xxxx xxxx xxxx

Since the one's complement of (D) is added to (P) + 1, the result is (P) – (D). The locations that may be addressed using this mode are (P) through (P) –255.

M = 10 Indexed. The operand address is formed by adding the value in the D Field to the value in the X register:

$$Y = (X) + (D)$$



Field	Bits	Definition
Op Code	11-15	Operation Code. Defines the specific instructions to be executed.
Mode Code (M Field)	9, 10	Used in conjunction with the Indirect Tag to define the memory addressing mode to be used: M = 00 Scratchpad M = 01 Relative to P, Forward M = 10 Indexed M = 11 Relative to P, Backward
Indirect Tag (I)	8	Specifies Direct or Indirect addressing. I = 0 Direct. M specifies operand address. I = 1 Indirect. M specifies address pointer address. Address Pointer specifies operand address, or the address of another Address Pointer.
D Field	0-7	Base value used to form operand or address pointer address.

Figure 2-1. Memory Reference Instruction Format: Word Mode

The address generated has the form:

(D) = 0000 0000 xxxx xxxx

(X) = 0xxx xxxx xxxx xxxx

Y = 0xxx xxxx xxxx xxxx

This mode of addressing forms a 15-bit address which uses the D Field of the instruction as modified by the X register. Since the X register may be easily incremented or decremented, this mode is especially useful for stepping through tables in memory. The locations which may be

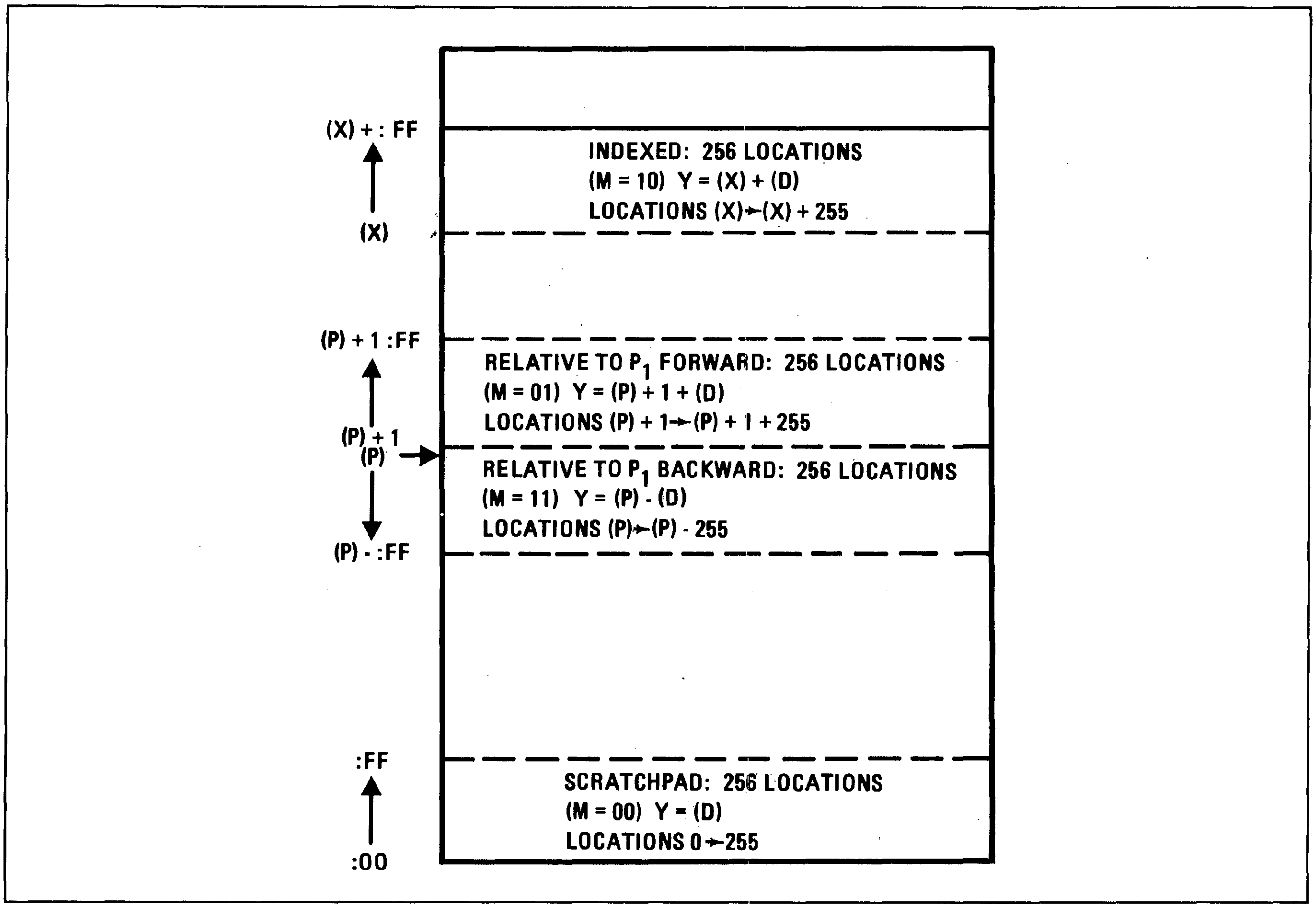


Figure 2-2. Direct Memory Addressing: Word Mode

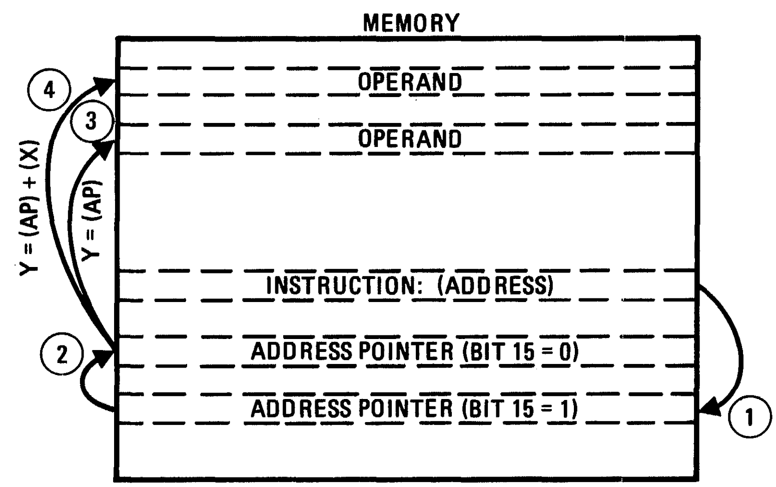
addressed using this mode are (X) through (X) +255.

2.3.2.3 Indirect Addressing. If I=1 (Bit 8 = 1), Indirect addressing is used to address a memory operand. In general, the address mode specified by M is used to address an indirect Address Pointer, AP, in memory. The Address Pointer contains the address of the operand. If multi-level indirect addressing is used, the Address Pointer may contain the address of another Address Pointer. The final Address Pointer contains the address of the memory operand.

Figure 1-12 illustrates the format of the Address Pointer. Each Address Pointer is examined by the computer after it is read from memory. If Bit 15 of the pointer contains a 0, the pointer contains the address of the memory operand. If Bit 15 of the Address Pointer contains a 1, the pointer contains the address of another Address Pointer. The number of indirect Address Pointers that may be accessed before the memory operand is accessed is limited only by memory capacity. Each Address Pointer is examined independently to determine whether or not another indirect level is required.

Figure 2-3 illustrates indirect addressing. Important points to note in the illustration are these:

1. A memory reference instruction uses either Scratchpad (M=00) or Relative to P (M=01 or M=11) addressing to access an indirect address pointer in memory. Indexed addressing is not used to address the address Pointer.
2. The Address Pointer contains a memory address and an indicator bit. The memory



(1) SCRATCH PAD ADDRESSING OR RELATIVE TO P ADDRESSING IS USED TO ADDRESS AN ADDRESS POINTER

(2) BITS 0 - 14 OF THE ADDRESS POINTER CONTAIN A MEMORY ADDRESS. IF BIT 15 OF THE ADDRESS POINTER CONTAINS A 1-BIT, THE MEMORY ADDRESS IN BITS 0 - 14 IS THE ADDRESS OF ANOTHER ADDRESS POINTER.

(3) IF BIT 15 OF THE ADDRESS POINTER CONTAINS A 0-BIT, THE ADDRESS IN BITS 0 - 14 IS THE ADDRESS OF THE MEMORY OPERAND.

(4) IF INDEXING IS SPECIFIED BY THE INSTRUCTION, THE ADDRESS IN BITS 0 - 14 IS ADDED TO THE CONTENTS OF THE X REGISTER TO FORM THE EFFECTIVE OPERAND ADDRESS.

Figure 2-3. Indirect Addressing: Word Mode

address is contained in bits 0-14 (word address: not byte address). Bit 15 of the Address Pointer is an indicator which tells what the memory address in bits 0-14 is addressing. If bit 15 contains a 1-bit (Bit 15=1), then the address in bits 0-14 is the address of another Address Pointer. The computer uses the address in bits 0-14 to read another word from memory. If bit 15 of the Address Pointer contains a 1 bit, the computer will treat the word addressed by the Address Pointer as another Address Pointer.

3. If bit 15 of an Address Pointer contains a 0-bit, then the address contained in bits 0-14 of the Address Pointer is the address of the memory operand that the memory reference instruction is looking for.

4. If indexing is specified by the memory reference instruction, the contents of the index register (X Register) are added to the address in the Address Pointer to form the effective operand address. This addition is performed with the Address Pointer that addresses the operand only. Indexing is not used to address the Address Pointer. Also, if indexing is specified, the first Address Pointer must be in the Scratchpad area of memory. Relative addressing cannot be used to address the Address Pointer if

indexing is to be used. The process of indexing the operand address rather than the Address Pointer address is called post indexing.

The following is a detailed description of indirect addressing using various addressing modes:

M = 00 Address Pointer in Scratchpad. The value in the D Field of the Memory Reference instruction is used as the address of the Address Pointer.

AP=(D)

If multi level indirect addressing is required, the value in the Address Pointer is the address of another Address Pointer:

AP=(AP)

The final Address Pointer contains the address of the memory operand:

Y=(AP)

M = 01 Address Pointer Relative to P, Forward. The value in the D Field is added to the contents of P+1 to form the address of the first Address Pointer:

AP=(D)+(P)+1

The remainder of the addressing steps are the same as for Address Pointer in Scratchpad:

AP=(AP), Y=(AP)

M = 11 Address Pointer Relative to P, Backward. The value in the D Field is subtracted from the value in P to form the address of the first Address Pointer:

AP=(P)-(D)

The remainder of the address steps are the same as for Address Pointer in Scratchpad:

AP=(AP), Y=(AP)

M = 10 Address Pointer in Scratchpad, Indexed. The value in the D Field is used to address the first Address Pointer.

AP=(D)

If multi level indirect addressing is required, the value in the Address Pointer is the address of another Address Pointer:

AP=(AP)

The value in the final Address Pointer is added to the value in the X Register to form the effective operand address:

Y=(AP)+(X)

2.3.3 Instruction Description Format

The instruction descriptions which follow completely describe each Memory Reference instruction used for Word Mode processing. Each instruction description follows the same format. The ADD instruction description, Paragraph 2.3.4.1, is used to explain the format. Refer to that description while reading the following explanation of the format.

2.3.3.1 Title. Each instruction description contains a title made up of the symbolic assembler code and short instruction description:

ADD ADD TO (A)

"ADD" is the symbolic code recognized by the ABLE and OMEGA assemblers for operation code assembly. "ADD TO (A)" is the short description of the instruction.

2.3.3.2 Format. Immediately following the title is a drawing of the instruction bit pattern showing fixed bits and variable fields. In the case of the ADD instruction, bits 11 through 15 contain the fixed operation code that uniquely defines the instruction. Bits 8, 9, and 10 define addressing modes, and bits 0 through 7 contain the D Field of the instruction.

2.3.3.3 Description. The bit format is followed by a detailed description of the functions of the instruction. The description contains both a word description stating what the instruction does, and a symbolic description of the functions of the instruction.

2.3.3.4 Machine Codes. There are eight different memory addressing modes which may be used with most memory reference instructions. The Machine Codes section contains the hexadecimal code defining the Operation Code and address mode for each memory addressing mode that may be used with the instruction:

:88nn	Direct, Scratchpad	Y=(D)

The ":88nn" is the hexadecimal code that defines an ADD instruction when Direct, Scratchpad memory addressing is used. The "nn" defines the variable D Field of the instruction. "Direct, Scratchpad" describes the memory addressing mode associated with the hexadecimal code. "Y=(D)" is a symbolic representation of the method used to form the effective address, Y, when that particular code is used.

2.3.3.5 Registers Affected. In this part of the instruction description, registers that may be changed by the execution of the instruction are listed along with the changes that may take place. For purposes of this explanation, the memory location that takes part in the operation is considered to be a register and is listed if memory is changed.

Since the P Register is normally incremented during the execution of an instruction, the P Register is not listed as a register affected unless some condition may cause the P Register to be modified in some way other than:

$$(P)+1 \rightarrow P$$

2.3.3.6 Timing. The number of computer cycles required to execute the instruction is defined by the timing description. For example, the ADD instruction requires two machine cycles (one cycle to fetch the instruction and one to fetch the operand) if no indirect addressing is used. A cycle is defined as 1.6 microseconds. Therefore, the time required to execute an ADD instruction if no indirect cycles are involved is:

$$(1.6)(2)=3.2 \text{ microseconds}$$

Add 1.6 microseconds for each level of indirect addressing.

2.3.4 Memory Reference Instruction Descriptions

Detailed descriptions of ALPHA 16 and NAKED MINI 16 Memory Reference (Word Mode) instructions are contained in the following paragraphs.

2.3.4.1

ADD ADD TO A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	M	M	I	D FIELD							

Adds contents of effective memory location to contents of A Register. Results stored in A:

$$(Y) + (A) \rightarrow A$$

Memory is unchanged. Previous contents of A are lost.

Machine Codes:

:88nn	Direct, Scratchpad	Y=(D)
:8Ann	Direct, Relative to P, Forward	Y=(P)+1+(D)
:8Enn	Direct, Relative to P, Backward	Y=(P)-(D)
:8Cnn	Indexed	Y=(D)+(X)
:89nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:8Bnn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:8Fnn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:8Dnn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP) +(X)

Registers Affected:

A	Previous contents replaced by sum.
OV	Set if arithmetic overflow occurs.

Timing: 2 + 1 for each indirect level.

2.3.4.2

SUB SUBTRACT FROM A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	M	M	I	D FIELD							

Subtracts the contents of effective memory location from contents of A Register. Results stored in A:

$$(A)\text{-}(Y) \rightarrow A$$

Memory is unchanged. Previous contents of A are lost.

Machine Codes:

:90nn	Direct, Scratchpad	Y=(D)
:92nn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:96nn	Direct, Relative to P, Backward	Y=(P)-(D)
:94nn	Indexed	Y=(D)+(X)
:91nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:93nn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:97nn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:95nn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

A	Previous contents replaced by difference
OV	Set if arithmetic overflow occurs.

Timing: 2+1 for each indirect level.

2.3.4.3

IMS INCREMENT MEMORY AND SKIP ON ZERO RESULT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	M	M	I	D FIELD							

The contents of effective memory location are incremented by one count and replaced. If the incrementing causes the result to become zero, a one place skip occurs:

$$(Y)+1 \rightarrow Y$$

If $(Y)+1 \neq 0$, then $(P)+1 \rightarrow P$

If $(Y)+1 = 0$, then $(P)+2 \rightarrow P$

Overflow is set if (Y)+1 = :8000.

Note: IMS is often used as an interrupt instruction. When IMS is used as an interrupt instruction, the skip will not occur when (Y)+1 = 0, and OV will not be set when (Y)+1 = :8000. An Echo is generated to the device requesting the interrupt when (Y)+1 = 0.

Machine Codes:

:D8nn	Direct, Scratchpad	Y=(D)
:DAnn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:DEnn	Direct, Relative to P, Backward	Y=(P)-(D)
:DCnn	Indexed	Y=(D)+(X)
:D9nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:DBnn	Indirect, Pointer relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:DFnn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:DDnn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

Memory	The contents of Y are incremented and replaced.
P	P is incremented twice if a skip condition occurs.
OV	OV is set if an arithmetic overflow occurs.

Timing: 2+1 for each indirect level.

Note: When executed as an interrupt instruction, execution time is 2-1/2 cycles. (All interrupt instructions are stretched 1/4 cycle, and IMS is stretched an additional 1/4 cycle.)

2.3.4.4

LDA LOAD A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	M	M	I	D FIELD							

Loads the contents of the effective memory location into the A Register:

(Y) → A

Memory is unchanged. Previous contents of A are lost.

Machine Codes:

:B0nn	Direct, Scratchpad	Y=(D)
:B2nn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:B6nn	Direct, Relative to P, Backward	Y=(P)-(D)
:B4nn	Indexed	Y=(D)+(X)
:B1nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:B3nn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:B7nn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:B5nn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

A Previous contents replaced by (Y).

Timing: 2+1 for each indirect level.

2.3.4.5

LDX LOAD X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	M	M	I	D FIELD							

Loads the contents of the effective memory location into the X Register:

(Y) → X

Memory is unchanged. Previous contents of X are lost.

Machine Codes:

:E0nn	Direct, Scratchpad	Y=(D)
:E2nn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:E6nn	Direct, Relative to P, Backward	Y=(P)–(D)
:E4nn	Indexed	Y=(D)+(X)
:E1nn	Indirect, Pointer in Scratchpad	AP=(D),Y=(AP)
:E3nn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:E7nn	Indirect, Pointer Relative to P, Backward	AP=(P)–(D), Y=(AP)
:E5nn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

X Previous contents replaced by (Y).

Timing: 2+1 for each indirect level.

2.3.4.6

STA STORE A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	M	M	I	D FIELD							

Stores contents of the A Register into the effective memory location:

$$(A) \rightarrow Y$$

A is unchanged. Previous contents of memory are lost.

Machine Codes:

:98nn	Direct, Scratchpad	Y=(D)
:9Ann	Direct, Relative to P, Forward	Y=(P)+1+(D)
:9Enn	Direct, Relative to P, Backward	Y=(P)-(D)
:9Cnn	Indexed	Y=(D)+(X)
:99nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:9Bnn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:9Fnn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:9Dnn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

Memory Previous contents of Y replaced by (A).

Timing: 2+1 for each indirect level.

2.3.4.7

STX STORE X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	M	M	I	D FIELD							

Stores contents of the X Register into the effective memory location:

$$(X) \rightarrow Y$$

X is unchanged. Previous contents of memory are lost.

Machine Codes:

:E8nn	Direct, Scratchpad	Y=(D)
:EAnn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:EEnn	Direct, Relative to P, Backward	Y=(P)-(D)
:ECnn	Indexed	Y=(D)+(X)
:E9nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:EBnn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:EFnn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:EDnn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

Memory Previous contents of Y replaced by (X).

Timing: 2+1 for each indirect cycle.

2.3.4.8

EMA EXCHANGE MEMORY AND A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	M	M	I	D FIELD							

Simultaneously stores contents of A Register in the effective memory location and loads contents of effective memory location into the A Register:

$$(Y) \rightarrow A$$

$$(A) \rightarrow Y$$

No data is lost when this instruction is executed.

Machine Codes:

:B8nn	Direct, Scratchpad	Y=(D)
:BAnn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:BEnn	Direct, Relative to P, Backward	Y=(P)-(D)
:BCnn	Indexed	Y=(D)+(X)
:B9nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:BBnn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:BFnn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:BDnn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

A	Previous contents of A are replaced by previous contents of Y.
Memory	Previous contents of Y are replaced by previous contents of A.

Timing: 2+1 for each indirect level.

2.3.4.9

AND AND TO A

I REGISTER

15	14	13	12	11	10	9	8	7–0
1	0	0	0	0	M	M	I	D FIELD

Performs the AND (logical product) of the contents of the effective memory location and the contents of the A Register. Results stored in A:

$$(Y) \wedge (A) \rightarrow A$$

Memory is unchanged. Previous contents of A are lost.

Machine Codes:

:80nn	Direct, Scratchpad	Y=(D)
:82nn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:86nn	Direct, Relative to P, Backward	Y=(P)-(D)
:84nn	Indexed	Y=(D)+(X)
:81nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:83nn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:87nn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:85nn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

A	Previous contents replaced by logical product.

Timing: 2+1 for each indirect level.

2.3.4.10

IOR INCLUSIVE OR TO A

I REGISTER

15	14	13	12	11	10	9	8	7–0
1	0	1	0	0	M	M	I	D FIELD

Inclusively OR's the contents of the effective memory location with contents of the A Register. The result is stored in A:

$$(Y) \vee (A) \rightarrow A$$

Memory is unchanged. The previous contents of A are lost.

Machine Codes:

:A0nn	Direct, Scratchpad	Y=(D)
:A2nn	Direct, Relative to P, Forward	Y=(P)+1+(D)

:A6nn	Direct, Relative to P, Backward	Y=(P)-(D)
:A4nn	Indexed	Y=(D)+(X)
:A1nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:A3nn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:A7nn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:A5nn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

A — Previous contents of A are replaced by the result of the Inclusive OR.

Timing: 2+1 for each indirect level.

2.3.4.11

XOR EXCLUSIVE OR TO A

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	0	1	0	1	M	M	I	D FIELD

Performs the Exclusive OR of the contents of the effective memory location and the A Register. The result is stored in A:

$$(Y) \vee (A) \rightarrow A$$

Memory is unchanged. The previous contents of A are lost.

Machine Codes:

:A8nn	Direct, Scratchpad	Y=(D)
:AAnn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:AEnn	Direct, Relative to P, Backward	Y=(P)-(D)
:ACnn	Indexed	Y=(D)+(X)
:A9nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:ABnn	Indirect, Pointer Relative to P, Forward	AP=(P)-(D), Y=(AP)
:AFnn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:ADnn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

A — Previous contents of A are replaced by the result of the Exclusive OR.

Timing: 2+1 for each indirect level.

2.3.4.12

CMS COMPARE AND SKIP IF HIGH OR EQUAL

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	1	0	M	M	I	D FIELD

Compares contents of effective memory location with contents of A Register and tests for A equal to, less than or greater than memory.

- If A less than memory, next instruction in sequence is executed (no skip). $(A) < (Y)$, then $(P)+1 \rightarrow P$
- If A greater than memory, a one-place skip occurs. $(A) > (Y)$, then $(P)+2 \rightarrow P$
- If A equal to memory, a two-place skip occurs. $(A) = (Y)$, then $(P)+3 \rightarrow P$

CMS is not interruptable if the skip is executed. (A) and (Y) are unchanged.

Machine Codes:

:D0nn	Direct, Scratchpad	Y=(D)
:D2nn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:D6nn	Direct, Relative to P, Backward	Y=(P)–(D)

:D4nn	Indexed	Y=(D)+(X)
:D1nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:D3nn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:D7nn	Indirect, Pointer Relative to P, Backward	AP=(P)–(D), Y=(AP)
:D5nn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

P Incremented normally if (A) < (Y).
Incremented twice if (A) > (Y).
Incremented a third time if (A) = (Y).

Timing: 2+1 for each indirect level.

2.3.4.13

SCN SCAN MEMORY

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	0	1	D FIELD							

SCAN causes a specified area in memory to be read and compared with the contents of the A Register. If a match is found within the area being scanned, the computer terminates the scan and skips one instruction. If no match is found, the computer terminates the instruction after all words have been compared and executes the next sequential instruction. The Scan instruction compares each full memory word with (A) if OV is reset (OV=0), and compares bits 8-15 of the memory word with bits 8-15 of (A) if OV is set (OV=1). The number of words to be scanned is specified by a word count in the X Register, and the base address (minus one) of the area to be scanned is contained in an indirect Address Pointer in Scratchpad. The D field of the Scan instruction contains the address of the Address Pointer. Therefore, the value assignments are:

A = Compare value
X = Word Count
(D) = Address of Address Pointer in Scratchpad: AP
(AP) = Base Address-1
(OV) = Compare indicator: 0 = compare full word.
1 = compare bits 8-15 only.

The X Register is decremented for each word scanned. The first word to be scanned is addressed through Indirect, Indexed addressing. The scan is performed from the highest address to the lowest. The Scan is terminated when one of two conditions occurs:

1. A match is found

2. All words scanned and no match found

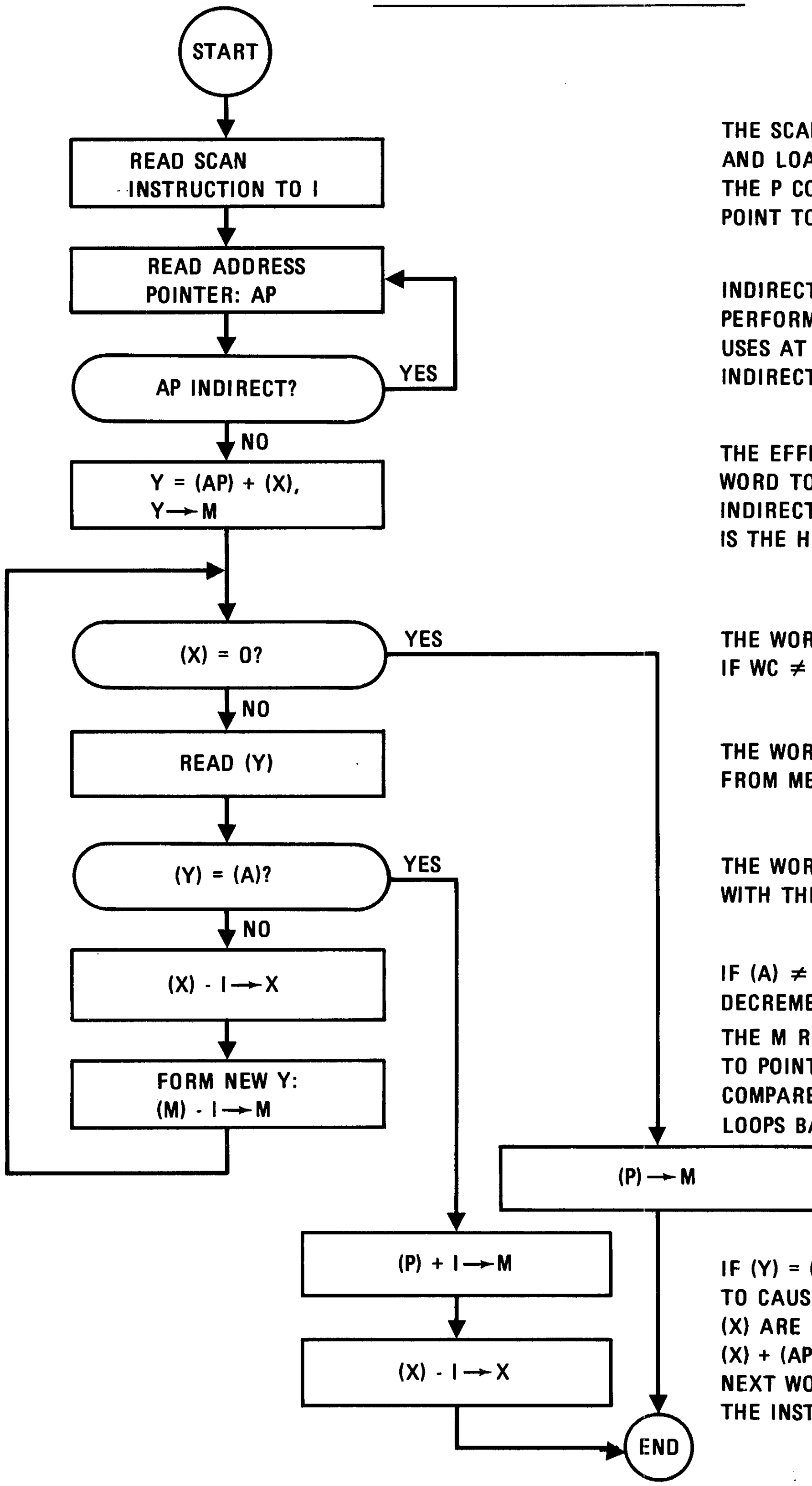
If no match is found, the computer executes the next sequential instruction. If a match is found, the computer skips one instruction

If a match is found, the X Register contains tne number of words remaining to be scanned, therefore the remainder of the table can be scanned simply by executing the SCN instruction again. The location of the word where the match was found is:

$$Y = (AP) + (X) + 1$$

The following is a flow chart of the SCN instruction.

SCN INSTRUCTION FLOW CHART



THE SCAN INSTRUCTION IS READ AND LOADED INTO THE I REGISTER. THE P COUNTER IS INCREMENTED TO POINT TO THE NEXT INSTRUCTION.

INDIRECT ADDRESS CYCLES ARE PERFORMED. THE SCAN INSTRUCTION USES AT LEAST ONE LEVEL OF INDIRECT ADDRESSING.

THE EFFECTIVE ADDRESS OF THE FIRST WORD TO BE SCANNED, Y, IS FORMED BY INDIRECT, INDEXED ADDRESSING. THIS IS THE HIGHEST ADDRESS TO BE SCANNED.

THE WORD COUNT IS TESTED FOR ZERO. IF WC ≠ 0, THE SCAN PROCEEDS.

THE WORD TO BE COMPARED IS READ. FROM MEMORY.

THE WORD FROM MEMORY IS COMPARED WITH THE CONTENTS OF THE A REGISTER.

IF (A) ≠ (Y), THE X REGISTER IS DECREMENTED.

THE M REGISTER IS DECREMENTED TO POINT TO THE NEXT WORD TO BE COMPARED, AND THE INSTRUCTION LOOPS BACK TO THE (X) = 0 TEST.

IF (X) = 0, THE CONTENTS OF P ARE TRANSFERED TO M, AND THE INSTRUCTION TERMINATES.

IF (Y) = (A), (P) ARE INCREMENTED TO CAUSE A ONE-PLACE SKIP, AND (X) ARE DECREMENTED SO THAT (X) + (AP) WILL POINT TO THE NEXT WORD TO BE SCANNED. THE INSTRUCTION TERMINATES.

Machine Codes:

:DCnn	Indirect, Indexed	AP=(D), Y=(AP)+(X)

Register Status:

A	Contains compare key
X	Contains number of words remaining to be scanned.
OV	Compare indicator: full word (OV=0) or upper byte (OV=1)
P	Incremented once (next instruction) if no compare found: (P)+1 P Incremented twice (skip one instruction) if compare found: (P)+2 P

Timing: 2 + 1 for each word scanned

2.3.4.14

JMP JUMP UNCONDITIONAL

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	M	M	I	D FIELD							

The P counter is loaded with the value of the effective memory address, causing an unconditional branch to that address:

$$Y \rightarrow P$$

The previous contents of P are lost, therefore there is no return linkage to the point from which the JMP occurred.

Machine Codes:

:F0nn	Direct, Scratchpad	Y=(D)
:F2nn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:F6nn	Direct, Relative to P, Backward	Y=(P)-(D)
:F4nn	Indexed	Y=(D)+(X)
:F1nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:F3nn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:F7nn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)
:F5nn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

P The previous contents of P are replaced by Y.

Timing: 1+1 for each indirect level

2.3.4.15

JST JUMP AND STORE

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	M	M	I	D FIELD							

The contents of the P counter (P + 1) are stored in the effective memory address. The P counter is changed after the store to contain the effective memory address plus one:

$$(P) + 1 \rightarrow Y$$
$$Y+1 \rightarrow P$$

This instruction provides an unconditioned jump to another location in memory, and stores a pointer to provide a return to the location following the JST instruction.

Machine Codes:

:F8nn	Direct, Scratchpad	Y=(D)
:FAnn	Direct, Relative to P, Forward	Y=(P)+1+(D)
:FEnn	Direct, Relative to P, Backward	Y=(P)-(D)
:FCnn	Indexed	Y=(D)+(X)
:F9nn	Indirect, Pointer in Scratchpad	AP=(D), Y=(AP)
:FBnn	Indirect, Pointer Relative to P, Forward	AP=(P)+1+(D), Y=(AP)
:FFnn	Indirect, Pointer Relative to P, Backward	AP=(P)-(D), Y=(AP)

:FDnn	Indirect, Indexed, Pointer in Scratchpad	AP=(D), Y=(AP)+(X)

Registers Affected:

Memory	Previous contents of Y replaced by (P)+1.
P	Previous contents of P replaced by Y+1.

Timing: 2+1 for each indirect level.

2.4 MEMORY REFERENCE INSTRUCTIONS: BYTE MODE

2.4.1 General

When the ALPHA 16 or NAKED MINI 16 computer is set for Byte Mode processing, Memory Reference instructions perform their logical or airthmetic operations using byte operands instead of word operands. When in Byte Mode, all of the memory reference instructions use byte operands with four exceptions: JMP, JST, IMS, and SCN. Even in Byte Mode, these instructions use word operands.

2.4.1.1 Byte Operands. General concepts of byte mode processing are discussed in Part 1.3.9 of Section 1. Several important points are illustrated in that discussion:

1. When a byte operand is read from memory, the byte is right justified within the operand word and the upper eight bits of the operand word are set to zeros.
2. Byte Mode affects the operand cycle only. Once the operand is read from memory, all other operations within the computer are performed the same as for full 16-bit words. In the case of byte operands, only the eight least significant bits of the operand word contain significant information.
3. A byte operand is an unsigned, absolute magnitude value for arithmetic operations; i.e., byte operands are always handled as positive values. This is true because the upper eight bits of a byte operand word always contain all zeros.
4. For arithmetic operations, carries are handled as if both values involved are full 16-bit words. Overflow will be set only if an arithmetic operation causes a full word arithmetic overflow; i.e., a carry from bit 7 to bit 8 of the Adder will not set the Overflow indicator. Overflow is set by the same conditions as for Word Mode.
5. Register store operations store the lower byte (eight least significant bits) of the register in the effective byte address.

2.4.1.2 Excluded Instructions. There are four memory reference instructions which are not affected by Byte Mode. These instructions always use a full word operand regardless of whether or not the computer is set for Byte Mode. The four excluded instructions are:

1.	IMS	IMS is normally used to increment counters for loops and timers, or indirect addresses for stepping through tables. Byte operands for IMS would be a limitation rather than an asset.
2.	SCN	Scan Memory is normally used for full word searches. It is used extensively in program debug operations when searching for program branches, etc.
3.	JMP	The unconditional Jump instruction generates an instruction address rather than an operand address. Instruction addresses are always full word addresses.
4.	JST	The Jump and Store instruction performs an operand cycle when it stores the contents of the P Register in the effective address. The full value of the

P Register must be stored for the instruction to be meaningful, therefore this instruction is excluded from Byte Mode.

2.4.2 Byte Addressing

The ALPHA 16 and NAKED MINI 16 computers have a maximum memory capacity of 64K bytes (32K words). A 16-bit address is required to address the maximum memory capacity in Byte Mode. When the computer is set for Byte Mode processing, the computer assumes that all operand addresses presented to memory by byte processing instructions are byte addresses. The computer assumes that the address is in the format shown in Figure 1-11.

2.4.2.1 Memory Reference Instruction Format. Figure 2-4 illustrates the format of memory reference instructions. The format for Byte Mode instructions is the same as for word mode instructions, except for the interpretation of the M Field of the instruction. There is nothing in the format of the instruction that distinguishes a memory reference instruction executed in Byte Mode from a memory reference instruction executed in Word Mode. The parameter that causes the computer to address a byte operand rather than a word operand is the Byte Mode indicator. If the Byte Mode indicator is set, the computer addresses byte operands. If the Byte Mode indicator is reset, the computer addresses word operands.

2.4.2.2 Direct Byte Addressing. Direct memory addressing in Byte Mode is not the same as for Word Mode addressing. The interpretation of the M Field is handled differently. Direct addressing is specified when I=0 (Bit 8=0). Direct memory addressing modes are explained below and are summarized in Figure 2-4: (Figure 2-5 illustrates the memory areas covered by each addressing mode.)

M=00 Byte Operand in Scratchpad. The D Field of the instruction contains the address of the byte operand in the scratchpad area of memory:

Y(byte)=(D)

Since the D Field contains only eight bits, the address will have the form:

Y(byte)=0000 0000 xxxx xxxx

(D) are right justified in the address word, and the total word is used as a byte address to memory. Since an 8-bit address can address up to 256 byte locations, direct Scratchpad addressing can address the first 256 bytes in memory (contained in the first 128 words in memory).

M=01, M=11 Byte Operand Relative to P, Forward. Relative addressing uses a word address in the P Register along with a word address in the D Field of the instruction to form the address of the word containing the byte to be addressed:

Y(word)=(P)+1+(D)

The address thus formed addresses the word, and the M Field of the instruction specified which byte in the word is to be used:

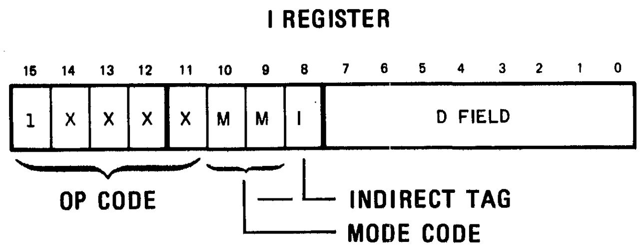
M=01	Byte 0 (left Byte)
M=11	Byte 1 (right Byte)

It is important to note that the address generated by relative addressing is a word address rather than a byte address. The M Field of the instruction specifies which byte of the word is being addressed. The address generated has the form:

(D)word	= 0000 0000 xxxx xxxx
(P)+1 word	= 0xxx xxxx xxxx xxxx
Y(word)	= 0xxx xxxx xxxx xxxx

Note. Byte addressing does not permit direct addressing relative to P, backward.

M=10 Indexed. The byte operand address is formed by adding the byte address value in the D Field to the byte address value in the X Register.



Field	Bits	Definition
OpCode	11-15	Operation Code. Defines the specific instruction.
M Field (Mode)	9, 10	Mode Code. Used in conjunction with the Indirect Tag to define the memory addressing mode to be used.
I Bit	8	Indirect Tag. Specifies direct or indirect addressing.
D Field	0-7	Address Field. Base address used to form byte operand address or address pointer address.

Addressing Modes:

MM	I		
Direct Addressing:			
00	0	Scratchpad	Y(byte)=(D)
01	0	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
11	0	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
10	0	Indexed	Y(byte)=(D)+(X)
Indirect Addressing:			
00	1	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
01	1	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
11	1	AP Relative to P, Backward	AP(word)=(P)-(D), Y(byte)=(AP)
10	1	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Figure 2-4. Memory Reference Instruction Format: Byte Mode

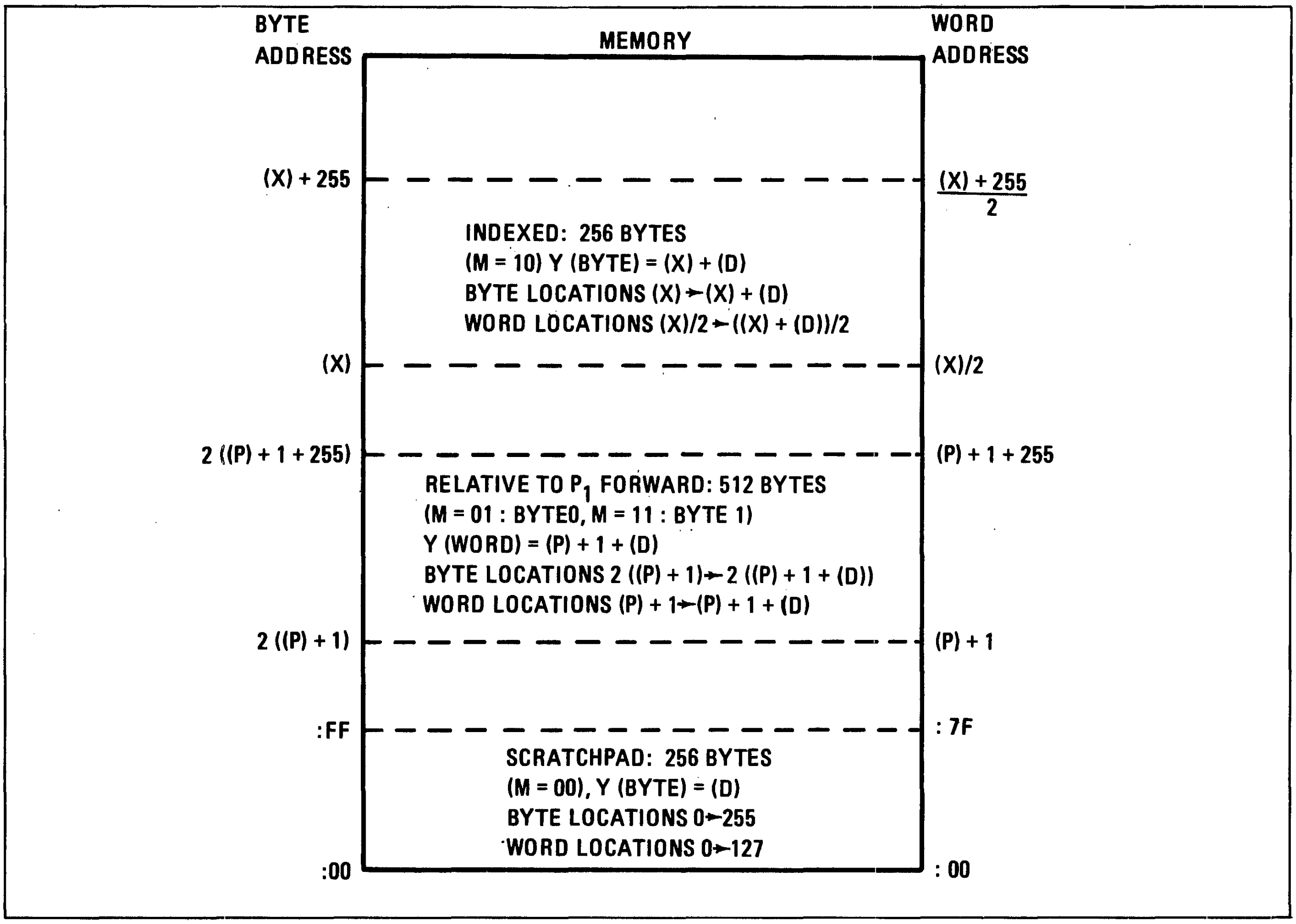


Figure 2-5. Direct Memory Addressing: Byte Mode

Y(byte)=(D)+(X)

The address generated has the form:

(D)byte	=	0000 0000 xxxx xxxx
(X)byte	=	xxxx xxxx xxxx xxxx
Y(byte)	=	xxxx xxxx xxxx xxxx

This mode of addressing forms a 16-bit address capable of addressing any byte in memory. Since the X Register may be easily incremented or decremented, this mode is especially useful for stepping through segments of memory where data is packed two bytes per word.

2.4.2.3 Indirect Byte Addressing. If I=1 (Bit 8=1), Indirect addressing is used to address a byte operand. Indirect byte addressing is limited to single level indirect addressing; i.e., there is only one indirect Address Pointer between the instruction and the byte operand.

For indirect byte addressing, the M Field of the memory reference instruction is interpreted the same as for indirect word addressing. The addressing mode specified by the M Field is used to form the address of an Address Pointer, AP, in memory. The address of the Address Pointer is a full word address, since the Address Pointer must have a full 16-bit capacity. The Address Pointer contains a btye address in byte address format. The byte address in the Address Pointer may be used directly as an effective memory address, or it may be modified by the contents of the X Register. Figure 2-6 illustrates Indirect Byte addressing.

The addressing modes used for Indirect Byte addressing are as follows:

M=00 Address Pointer in Scratchpad. The D Field of the Memory Reference instruction contains the word address of an Address Pointer in Scratchpad:

AP(word)=(D)

The Address Pointer contains the byte address of the byte operand:

Y(byte)=(AP)

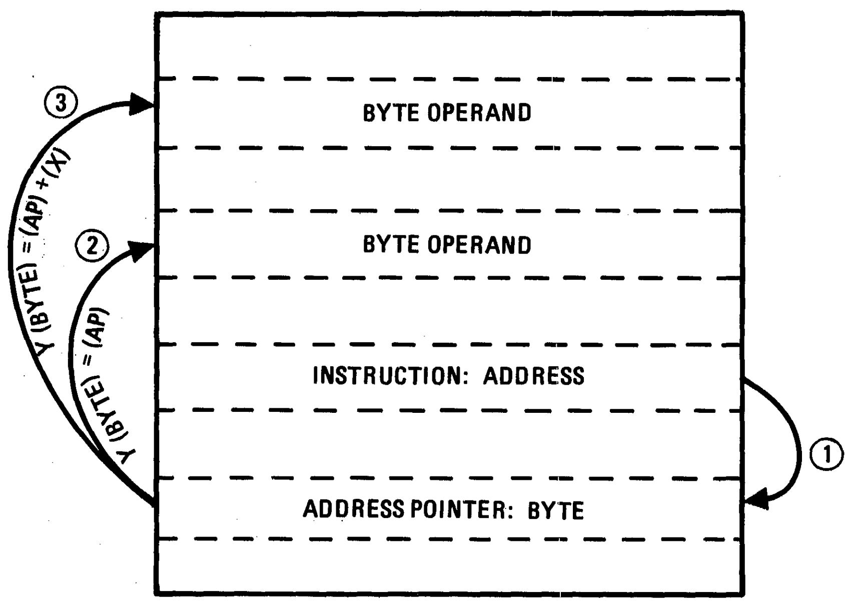
M=01 Address Pointer Relative to P, Forward. The value in the D Field of the Memory Reference instruction is added to the contents of P, +1, to form the address of the Address Pointer:

AP(word)=(P)+1+(D)

The Address Pointer contains the byte address of the byte operand:

Y(byte)=(AP)

M=11 Address Pointer Relative to P, Backward. The value in the D Field of the Memory Reference instruction is subtracted from the value in the P Register



① SCRATCHPAD OR RELATIVE ADDRESSING IS USED TO ADDRESS A FULL WORD ADDRESS POINTER.

② IF INDEXING IS NOT REQUIRED, THE ADDRESS POINTER CONTAINS THE EFFECTIVE 16-BIT BYTE ADDRESS.

③ IF INDEXING IS REQUIRED, THE BYTE ADDRESS IN THE ADDRESS POINTER IS ADDED TO THE VALUE IN THE X REGISTER TO FORM THE EFFECTIVE BYTE ADDRESS.

Figure 2-6. Indirect Addressing: Byte Mode to form the word address of the Address Pointer:

AP(word)=(P)-(D)

The Address Pointer contains the byte address of the byte operand.

M=10 Address Pointer in Scratchpad, Indexed. The D Field of the Memory Reference instruction contains the word address of the Address Pointer in Scratchpad:

AP(word)=(D)

The contents of the Address Pointer are added to the contents of the X Register to form the effective byte operand address:

Y(byte)=(AP)+(X)

2.4.3 Instruction Descriptions

Memory reference instruction functions, when executed in Byte Mode, are explained in the following paragraphs. The description format is the same as for the instruction descriptions explained in Part 2.3.3.

2.4.3.1

ADDB ADD BYTE TO (A)

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	M	M	I	D FIELD							

Adds the absolute magnitude of effective byte to contents of A Register. Results stored in A:

(Y)byte+(A) → A

The byte is right justified in the operand word and added to the contents of the A register. The addition is a full 16 bit add. OV is set if arithmetic overflow occurs.

Machine Codes:

Direct Addressing:

:88nn	Scratchpad	Y(byte)=(D)
:8Ann	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:8Enn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:8Cnn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:89nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:8Bnn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
:8Fnn	AP Relative to P, Backward	AP(word)=(P)-(D), Y(byte)=(AP)
:8Dnn	AP in Scratchpad, Indexed	AP(word)=D, Y(byte)=(AP)+(X)

Registers Affected:

A Previous contents replaced by sum.
OV Set if arithmetic overflow occurs.

Timing: 2+1 if indirect.

2.4.3.2

SUBB SUBTRACT BYTE FROM (A)

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	M	M	I	D FIELD							

Subtracts the absolute magnitude of effective byte from contents of A Register. Results stored in A:

(A)word-(Y)byte → A

The byte operand is right justified in the operand word and subtracted from the contents of the A Register. OV is set if airthmetic overflow occurs.

Machine Codes:

Direct Addressing:

:90nn	Scratchpad	Y(byte)=(D)
:92nn	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:96nn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:94nn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:91nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:93nn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
:97nn	AP Relative to P, Backward	AP(word)=(P)-(D), Y(byte)=(AP)
:95nn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

A Previous contents replaced by difference.

OV Set if arithmetic overflow occurs.

Timing: 2+1 if indirect.

2.4.3.3

LDAB LOAD A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	M	M	I	D FIELD							

Loads the contents of the effective byte into the lower half of the A Register. The upper half of the A Register is set to zeros:

$$A = 0000\ 0000\ \underbrace{\text{xxxx xxxx}}_{\text{Byte}}$$

The operation is:

$$(Y)\text{byte} \rightarrow A$$

Memory is unchanged. The previous contents of A are lost.

Machine Codes:

Direct Addressing:

:B0nn	Scratchpad	Y(byte)=(D)
:B2nn	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:B6nn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:B4nn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:B1nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:B3nn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
:B7nn	AP Relative to P, Backward	AP(word)=(P)-(D), Y(byte)=(AP)
:B5nn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

A Previous contents replaced by (Y)byte

Timing: 2+1 if indirect.

2.4.3.4

LDXB LOAD X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	M	M	I	D FIELD							

Loads the contents of the effective byte into the lower half of the X Register. The upper half of the X Register is set to zeros:

$$X = 0000\ 0000\ \underset{\text{Byte}}{\underline{xxxx\ xxxx}}$$

The operation is:

$$(Y)\text{byte} \rightarrow X$$

Memory is unchanged. The previous contents of X are lost.

Machine Codes:

Direct Addressing:

:E0nn	Scratchpad	Y(byte)=(D)
:E2nn	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:E6nn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:E4nn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:E1nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:E3nn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
:E7nn	AP Relative to P, Backward	AP(word)=(P)–(D), Y(byte)=(AP)
:E5nn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

X Previous contents replaced by (Y) byte.

Timing: 2+1 if indirect.

2.4.3.5

STAB STORE A

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	0	0	1	1	M	M	I	D FIELD

Stores contents of the lower half of the A Register into the effective byte location:

$$(A)_{0\text{-}7} \rightarrow Y(\text{byte})$$

A is unchanged. The previous contents of the effective byte location are lost.

Machine Codes:

Direct Addressing:

:98nn	Scratchpad	Y(byte)=(D)
:9Ann	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:9Enn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:9Cnn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:99nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:9Bnn	AP Relative to P, Forward	AP(word=(P)+1+(D), Y(byte)=(AP)
:9Fnn	AP Relative to P, Backward	AP(word)=(P)–(D), Y(byte)=(AP)
:9Dnn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

Memory Previous contents of effective byte location replaced by contents of A Register, bits 0-7.

Timing: 2+1 if indirect.

2.4.3.6

STXB STORE X

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	1	0	1	M	M	I	D FIELD

Stores contents of the lower half of the X Register into the effective byte location.

$$(X)_{0\text{-}7} \rightarrow Y(\text{byte})$$

X is unchanged. The previous contents of the effective byte location are lost.

Machine Codes:

Direct Addressing:

:E8nn	Scratchpad	Y(byte)=(D)
:EAnn	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:EEnn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:ECnn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:E9nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:EBnn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
:EFnn	AP Relative to P, Backward	AP(word)=(P)–(D), Y(byte)=(AP)
:EDnn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

Memory — Previous contents of effective byte location replaced by contents of X Register, bits 0-7.

Timing: 2+1 if indirect.

2.4.3.7

EMAB EXCHANGE MEMORY AND A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	M	M	I	D FIELD							

Simultaneously stores contents of the lower half of the A Register in the effective byte location and loads contents of effective byte location into the lower half of the A Register is unconditionally set to zeros:

$$(A)_{0\text{-}7} \rightarrow Y(\text{byte})$$

$$(Y)\text{byte} \rightarrow A_{0\text{-}7}$$

$$0 \rightarrow A_{8\text{-}15}$$

The previous contents of the upper half of the A Register (bits 8-15) are lost.

Machine Codes:

Direct Addressing:

:B8nn	Scratchpad	Y(byte)=(D)
:BAnn	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:BEnn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:BCnn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:B9nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:BBnn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
:BFnn	AP Relative to P, Backward	AP (word)=(P)–(D), Y(byte)=(AP)
:BDnn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

A — Previous contents of A, bits 0-7, replaced by (Y) byte.
Previous contents of A, bits 8-15, replaced by 0's.

Memory — Previous contents of Y(byte) replaced by $(A)_{0\text{-}7}$.

Timing: 2+1 if indirect.

2.4.3.8

ANDB AND TO A

I REGISTER

15	14	13	12	11	10	9	8	7-0
1	0	0	0	0	M	M	I	D FIELD

Performs the AND (logical product) of the contents of the effective byte location and the contents of the A Register. Results stored in A:

$$(Y)\text{byte} \wedge (A) \rightarrow A$$

Since the byte operand occupies bits 0-7 of the operand word and bits 8-15 of the operand word contains zeros, bits 8-15 of the A Register are unconditionally set to zeros as a result of this operation. Memory is unchanged.

Machine Codes:

Direct Addressing:

:80nn	Scratchpad	Y(byte)=(D)
:82nn	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:86nn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:84nn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:81nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:83nn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
:87nn	AP Relative to P, Backward	AP(word)=(P)–(D), Y(byte)=(AP)
:85nn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

A Previous contents replaced by logical product of $(A)_{0\text{-}7}$ and (Y)byte. $(A)_{8\text{-}15}$ set to zeros.

Timing: 2+1 if indirect.

2.4.3.9

IORB INCLUSIVE OR

I REGISTER

15	14	13	12	11	10	9	8	7-0
1	0	1	0	0	M	M	I	D FIELD

Inclusively OR's the contents of the effective byte location with contents of the A Register:

$$(Y)\text{byte} \vee (A) \rightarrow A$$

This instruction effectively performs the inclusive OR of $(A)_{0\text{-}7}$ with (Y)byte. Since the upper half of the operand word contains all zeros, the upper half of A, $(A)_{8\text{-}15}$, is unchanged by this instruction. Memory is unchanged.

Machine Codes:

Direct Addressing:

:A0nn	Scratchpad	Y(byte)=(D)
:A2nn	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:A6nn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:A4nn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:A1nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:A3nn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)

:A7nn	AP Relative to P, Backward	AP(word)=(P)–(D), Y(byte)=(AP)
:A5nn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

A Previous contents of A, bits 0-7, replaced by inclusive OR of $(A)_{0-7}$ and (Y)byte. $(A)_{8-15}$ unchanged.

Timing: 2+1 if indirect.

2.4.3.10

XORB EXCLUSIVE OR TO A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	M	M	I	D FIELD							

Performs the Exclusive OR of the contents of the effective byte location and the A Register. The result is stored in A:

$$(Y)\text{byte} \veebar (A) \rightarrow A$$

This instruction effectively performs the exclusive OR of $(A)_{0-7}$ with (Y)byte. Since the upper half of the operand word contains all zeros, the upper half of A, $(A)_{8-15}$, is not changed by this instruction. Memory is unchanged.

Machine Codes:

Direct Addressing:

:A8nn	Scratchpad	Y(byte)=(D)
:AAnn	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:AEnn	Relative to P, Forward, Byte 1	Y(word)=(P)+1+(D)
:ACnn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:A9nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:ABnn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
:AFnn	AP Relative to P, Backward	AP(word)=(P)–(D), Y(byte)=(AP)
:ADnn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

A Previous contents of A, bits 0-7, replaced by exclusive OR of $(A)_{0-7}$ and (Y)byte. $(A)_{8-15}$ unchanged.

Timing: 2+1 if indirect.

2.4.3.11

CMSB COMPARE AND SKIP IF HIGH OR EQUAL

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	M	M	I	D FIELD							

Compares contents of effective byte location with contents of A Register and tests for A equal to, less than or greater than memory.

- If A less than memory, next instruction in sequence is executed (no skip). (A) < (Y)byte, then (P)+1 → P
- If A greater than memory, a one-place skip occurs. (A) > (Y) byte, then (P)+1 → P
- If A equal to memory, a two-place skip occurs. (A)=(Y)byte, then (P)+3 → P

The compare is a full word (16 bit) compare. For the instruction to be meaningful when executed with a byte operand, the upper half of A, A_{8-15}, should contain all zeros.

(A) and (Y)byte are unchanged by this instruction. CMSB is not interruptable if a skip is executed.

Machine Codes:

Direct Addressing:

:D0nn	Scratchpad	Y(byte)=(D)
:D2nn	Relative to P, Forward; Byte 0	Y(word)=(P)+1+(D)
:D6nn	Relative to P, Forward, Byte 1	Y(Word)=(P)+1+(D)
:D4nn	Indexed	Y(byte)=(D)+(X)

Indirect Addressing:

:D1nn	AP in Scratchpad	AP(word)=(D), Y(byte)=(AP)
:D3nn	AP Relative to P, Forward	AP(word)=(P)+1+(D), Y(byte)=(AP)
:D7nn	AP Relative to P, Backward	AP(word)=(P)–(D), Y(byte)=(AP)
:D5nn	AP in Scratchpad, Indexed	AP(word)=(D), Y(byte)=(AP)+(X)

Registers Affected:

P Incremented normally if (A) < (Y)byte.
Incremented twice if (A) > (Y)byte.
Incremented by 3 if (A)=(Y) byte.

Timing: 2+1 if indirect.

2.5 IMMEDIATE INSTRUCTIONS

2.5.1 General

Immediate instructions are similar to Memory Reference instructions in that they perform logical and arithmetic operations involving memory data and operating registers. The memory data, however, is stored within the Immediate instruction itself rather than in a separate operand word or operand byte.

2.5.2 Immediate Instruction Format

Figure 2-7 illustrates the general format used by Immediate instructions. The format is divided into three fields.

2.5.2.1 Class. The Immediate instruction class is defined by the bit pattern in bits 11 – 15 of the instruction. By class definition, Immediate instructions are a subclass of Memory Reference instructions because Bit 15 of the Immediate class contains a 1-bit. Because of the difference in function, however, Immediate instructions are treated as a separate class.

2.5.2.2 Op Code. Bits 8 – 10 define the specific Immediate instruction to be executed once the class is decoded. Since there are three bits in this field, there are eight possible Immediate instructions.

2.5.2.3 D Field. The D Field of an Immediate instruction contains the operand used by the instruction; i.e., the value in the D Field is the actual value used by the instruction rather than an address parameter.

Immediate instruction operands are similar to byte operands in that the lower eight bits of the instruction are right justified in the operand word and the upper eight bits of the operand word are set to zeros. The operand is then handled as a full 16-bit word with significant data in the eight least significant bits only.

2.5.3 Immediate Instruction Functions

There are two distinct advantages to using Immediate instructions instead of Memory Reference instructions:

1. Speed — Immediate instructions require only one cycle since no operand cycle is required.
2. Memory — Since the operand is stored in the instruction word, no additional memory space is required to store the operand.

These instructions are especially useful for storing constants for comparisons, iteration counts, etc.

2.5.4 Instruction Descriptions

Detailed descriptions of the Immediate instructions are contained in the following paragraphs. The descriptions

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	0	0				D FIELD (OPERAND)

IMMEDIATE CLASS OP CODE

Bits	Field	Definition
11-15	Class	Specifies the Immediate Instruction class.
8-10	OpCode	Defines the specific Immediate instruction.
0-7	D Field	Contains the Operand of the Immediate instruction.

Figure 2-7. Immediate Instruction Format

follow the same format as the Memory Reference instruction descriptions. The format is described in Part 2.3.3.

2.5.4.1

AXI ADD TO X IMMEDIATE

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	0	0	0	1	0	D FIELD (OPERAND)

The operand (D Field of the instruction) is added to the contents of the X Register:

(D) + (X) → X

The Add is in the form:

(X) =	xxxx xxxx xxxx xxxx
(D) =	0000 0000 xxxx xxxx
Sum =	xxxx xxxx xxxx xxxx

The upper half of X is changed if there are carries from the add operation in the lower half. OV is set if arithmetic overflow occurs. Previous contents of X are lost.

Machine Code:

:C2nn

Registers Affected:

X Previous contents replaced by sum.
OV Set if arithmetic overflow occurs.

Timing: 1

2.5.4.2

SXI SUBTRACT FROM X IMMEDIATE

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	0	0	0	1	1	D FIELD (OPERAND)

The operand (D Field of instruction) is subtracted from the contents of X Register:

(X) – (D) → X

The subtract is in the form:

(X) =	xxxx xxxx xxxx xxxx	
+ (-D) =	1111 1111 xxxx xxxx	= 2's Complement of (D)
Result =	xxxx xxxx xxxx xxxx	

The value in the D Field is treated as an absolute magnitude, positive value. The two's complement of the full 16-bit operand, with (D) right justified, is added to (X). The result is (X) – (D). The result is stored in X, and the previous contents of X are lost. OV is set if arithmetic overflow occurs.

Machine Code:

:C3nn

Registers Affected:

X Previous contents replaced by (X) – (D).
OV Set if arithmetic overflow occurs.

Timing: 1

2.5.4.3

LAP LOAD A POSITIVE IMMEDIATE

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	0	0	1	1	0	D FIELD (OPERAND)

The operand (D Field of instruction) is loaded into lower half of A Register. The upper half of A is set to zero:

$$(D) \rightarrow A_{0\text{-}7}$$

$$0 \rightarrow A_{8\text{-}15}$$

Previous contents of A are lost.

Machine Code:

:C6nn

Registers Affected:

A Previous contents replaced by (D), right justified.

Timing: 1

2.5.4.4

LXP LOAD X POSITIVE IMMEDIATE

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	0	0	1	0	0	D FIELD (OPERAND)

The operand (D Field of instruction) is loaded into the lower half of the X Register. The upper half is set to zero:

$$(D) \rightarrow X_{0\text{-}7}$$

$$0 \rightarrow X_{8\text{-}15}$$

Previous contents of X are lost.

Machine Codes:

:C4nn

Registers Affected:

X Previous contents replaced by (D), right justified.

Timing: 1

2.5.4.5

LAM LOAD A MINUS IMMEDIATE

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	0	0	1	1	1	D FIELD (OPERAND)

The operand (D Field of instruction) is negated (two's complemented) and loaded into the A Register:

$$-(D) \rightarrow A$$

The value stored in A has the form:

(A) = 1111 1111 xxxx xxxx

Previous contents of A are lost.

Machine Code:

:C7nn

Registers Affected:

A Previous contents replaced by –(D).

Timing: 1

2.5.4.6

LXM LOAD X MINUS IMMEDIATE

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	0	0	1	0	1	D FIELD (OPERAND)

The operand (D Field of instruction) is negated (two's complemented) and loaded into the X Register:

$$-(D) \rightarrow X$$

The value stored in X has the form:

(X) = 1111 1111 xxxx xxxx

Previous contents of X are lost.

Machine Code:

:C5nn

Registers Affected:

X Previous contents replaced by –(D).

Timing: 1

2.5.4.7

CAI COMPARE TO A IMMEDIATE

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	0	0	0	0	0	D FIELD (OPERAND)

The operand (D Field of instruction) is compared to lower half of A Register. If unequal a skip of one place occurs. If equal, the next instruction in sequence is executed. The contents of A are not disturbed:

If $(D) = (A)_{0\text{-}7}$ then $(P)+1 \rightarrow P$
If $(D) \neq (A)_{0\text{-}7}$ then $(P)+2 \rightarrow P$

The upper half of A, $(A)_{8\text{-}15}$, does not take part in the comparison.

Machine Code:

:C0nn

Registers Affected:

P Incremented normally if $(A)_{0\text{-}7} = (D)$.
Incremented twice if $(A)_{0\text{-}7} \neq (D)$.

Timing: 1

2.5.4.8

CXI COMPARE TO X IMMEDIATE

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
1	1	0	0	0	0	0	1	D FIELD (OPERAND)

The operand (D Field of instruction) is compared to lower half of X register. If unequal, a skip of one place occurs. If equal, the next instruction in sequence is executed. The contents of X are not disturbed:

If $(D) = (X)_{0\text{-}7}$ then $(P)+1 \rightarrow P$
If $(D) \neq (X)_{0\text{-}7}$ then $(P)+2 \rightarrow P$

The upper half of X, $(X)_{8\text{-}15}$, does not take part in the comparison.

Machine Code:

:C1nn

Registers Affected:

P Incremented normally if $(X)_{0\text{-}7} = (D)$
Incremented twice if $(X)_{0\text{-}7} \neq (D)$

Timing: 1

2.6 CONDITIONAL JUMP INSTRUCTIONS

2.6.1 General

Conditional Jump instructions are those instructions which test conditions within the ALPHA 16 or NAKED MINI 16 computers and perform program branches depending on the results of the test. A Jump occurs if the test condition is satisfied, and the next sequential instruction is executed if the test condition is not satisfied. All branches are relative to the contents of the P Register (location of the Conditional Jump instruction). Jumps may be relative to P forward, from 1 to 64 locations, or relative to P backward, from 0 to 63 locations:

Forward Jumps:	P+1 through P+64
Backward Jumps:	P–0 through P–63

Figure 2-8 illustrates the general format for Conditional Jump instruction.

2.6.2 Testable Conditions

There are five different conditions within the computer which may be tested by Conditional Jump instructions. These conditions are:

1. Sign of A (positive or negative)
2. Contents of A (zero or not zero)
3. Contents of X (zero or not zero)
4. Overflow Indicator (set or reset)
5. Sense Switch (on or off)

The testable conditions may be tested individually or in combination. Test instructions may be coded so that all conditions specified must be met for a jump to occur (AND test group), or they may be coded so that only one of the selected conditions must be met for a jump to occur (OR test group). There are limits to the conditions that can be tested in each group. For example, the AND test group can test the A Register for a response if A is positive, but cannot test for a response if A is negative. The OR test group can test for a response if A is negative, but not for a response if A is positive.

2.6.2.1 AND Test Group. The AND test group is identified by a 1-bit in the G Field (Bit 12) of a Conditional Jump instruction. Bits 7 through 11 of the instruction identify the conditions to be tested. A 1-bit indicates that the test is to be performed, and a 0-bit indicates that the test is not to be performed. For example, a 1-bit in bit 7 specifies that the A Register is to be tested for a positive condition; i.e., the sign bit (Bit 15) of A is positive. If bit 8 is on, the A Register is tested for a non-zero condition. If bits 7 and 8 are both on, the A Register is tested for both positive and non-zero. Both conditions must be met for the test to be satisfied.

The test conditions in the AND test group are:

Bit	Test	Description
7	A Positive	The test is satisfied if the sign bit of A is positive (A_{15}=0).
8	A≠0	The test is satisfied if the A Register contains at least one 1-bit.
9	OV Reset	The test is satisfied if the Overflow indicator is reset (OV=0).
10	SS On	The test is satisfied if the Sense Switch on the console is On (down).
11	X≠0	The test is satisfied if the X Register contains at least one 1-bit.

In the AND test group, all of the conditions specified by the instruction must be satisfied for the branch to occur.

2.6.2.2 OR Test Group. The OR test group is identified by a 0-bit in the G Field (Bit 12) of a Conditional Jump instruction. The OR group differs from the AND group in that only one of the conditions specified by the instruction must be satisfied for the branch to occur. Also, the OR group tests for opposite states than the AND group.

Test conditions in the OR test group are:

Bit	Test	Description
7	A Negative	The test is satisfied if the sign bit of A is negative (A_{15}=1).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	G						R	D FIELD					

Bits	Field	Definition
13-15	Class	Identifies the Conditional Jump Instruction Class
12	G	Test Group Indicator: G=1 for AND Group G=0 for OR Group
7-11	Conditions	Microcode of Test Conditions:
6	R	Jump Direction: R=0 for Forward Jump R=1 for Backward Jump
0-5	D Field	Jump Distance (–63 to +64)

Microcode of Test Conditions (bits 7-11):

Bit	AND Group	OR Group
7	A Positive	A Negative
8	A≠0	A=0
9	OV Reset	OV Set (Resets OV)
10	SS On	SS Off
11	X≠0	X-0

Figure 2-8. Conditional Jump Format

Bit	Test	Description
8	A=0	*The test is satisfied if the A Register contains all zeros.
9	OV Set	The test is satisfied if the Overflow indicator is set (OV=1). NOTE: The Overflow indicator is conditionally reset when this test is executed.
10	SS Off	The test is satisfied if the Sense Switch on the console is Off (up).

*NOTE: All of the OR tests can be used in combination except for A=0 and X=0. These two cannot be used in the same text. For example, the test:

(A=0) OR (X=0)

cannot be used. The reason is that a condition such as the following would satisfy the test:

A = 0101 0101 0101 0101

X = 1010 1010 1010 1010

If the two registers taken together have a 0-bit in each bit position, the test is satisfied. Therefore, this combination is excluded as a legitimate test.

Bit	Test	Description
11	X=0	*The test is satisfied if the X Register contains all zeros.

2.6.3 Instruction Descriptions

Conditional Jump instructions for which symbolic codes have been derived are explained in the following paragraphs. A general code, JOC, for Jump on Condition, is provided so that the programmer may microcode jump conditions for which symbolic codes are not provided.

The format of the instructions described is similar to the Memory Reference description format with the exception of the Machine Codes section. The hexadecimal codes listed show the range of each instruction for both forward and backward jumps.

2.6.3.1

JOC JUMP ON CONDITIONS

Assembler Format:

JOC XX, ADR

JOC is a general symbolic operation code recognized by the 16-bit machine language assemblers. It allows the programmer to microcode specific Conditional Jump instructions for which symbolic codes are not provided. The Assembler Format is as follows:

1. JOC — The general symbolic Op Code.
2. XX — The hexadecimal code for the bit pattern in bits 7-12 (condition bits).
3. ADR — Jump direction and distance or symbolic address to which jump is to be made if jump condition(s) is met.

Example: The JAL instruction could be microcoded using JOC in this manner:

JOC :03, Loop (LOOP must be within ±64 locations)

2.6.3.2

JAM JUMP IF A MINUS

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0	1	R	D FIELD					

A jump occurs if the A Register is less than zero (A15 = 1). Otherwise the next instruction in sequence is executed.

If (A) < 0, then Jump
If (A) ≥ 0, then (P)+1 → P

(A) are unchanged.

Machine Codes:

:2080 – :20BF for forward jumps (+1 thru +64)
:20C0 – :20FF for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if test conditions not met. Loaded with jump address if test condition met.

Timing: 1

2.6.3.3

JAP JUMP IF A POSITIVE

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	1	R	D FIELD					

A jump occurs if the A Register is positive (A15 = 0). Otherwise the next instruction in sequence is executed:

If (A) ≥ 0, then Jump
If (A) < 0, then (P)+1 → P

(A) are unchanged.

Machine Codes:

:3080 – :30BF for forward jumps (+1 thru +64)
:30C0 – :30FF for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if test condition not satisfied.
Offset by (D) if test condition satisfied.

Timing: 1

2.6.3.4

JAZ JUMP IF A ZERO

I REGISTER

15	14	13	12	11	10	9	8	7	6	5 4 3 2 1 0
0	0	1	0	0	0	0	1	0	R	D FIELD

A jump occurs if the A Register is zero. Otherwise the next instruction in sequence is executed.

If (A) = 0, then Jump
If (A)≠0, then (P)+1 → P

(A) unchanged.

Machine Codes:

:2100 – :213F for forward jumps (+1 thru +64)
:2140 – :217F for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if test condition not satisfied.
Offset by (D) if test condition satisfied.

2.6.3.5

JAN JUMP IF A NOT ZERO

I REGISTER

15	14	13	12	11	10	9	8	7	6	5 4 3 2 1 0
0	0	1	1	0	0	0	1	0	R	D FIELD

A jump occurs if the A Register is not zero. Otherwise the next instruction in sequence is executed:

If (A)≠0, then Jump
If (A)=0, then (P)+1 → P

(A) are unchanged.

Machine Codes:

:3100 – :313F for forward jumps (+1 thru +64)
:3140 – :317F for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if test condition not satisfied.
Offset by (D) if test condition satisfied.

Timing: 1

2.6.3.6

JAG JUMP IF A GREATER THAN ZERO

I REGISTER

15	14	13	12	11	10	9	8	7	6	5 4 3 2 1 0
0	0	1	1	0	0	0	1	1	R	D FIELD

A jump occurs if the A Register is greater than zero. Otherwise the next instruction in sequence is executed.

If (A) > 0, then Jump
If (A) ≤ 0, then (P)+1 → P

(A) are unchanged.

Note: The test conditions are:

(A) Positive AND (A)≠0

Machine Codes:

:3180 – :31BF for forward jumps (+1 thru +64)
:31C0 – :31FF for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if test conditions are not satisfied.

Offset by (D) if all test conditions are satisfied.

Timing: 1

2.6.3.7

JAL JUMP IF A LESS THAN OR EQUAL TO ZERO

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	1	R	D FIELD					

A jump occurs if the A Register is less than or equal to zero. Otherwise the next instruction in sequence is executed.

If (A) < 0, then Jump
If (A) = 0, then Jump
If (A) > 0, then (P)+1 → P

(A) are unchanged.

Machine Codes:

:2180 – :21BF for forward jumps (+1 thru +64)
:21C0 – :21FF for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if neither test condition satisifed.
Offset by (D) if either test condition satisfied.

Timing: 1

2.6.3.8

JXZ JUMP IF X ZERO

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	0	0	0	R	D FIELD					

A jump occurs if the X Register is zero. Otherwise the next instruction in sequence is executed:

If (X) = 0, then Jump
If (X)≠0, then (P)+1 → P

(X) are unchanged.

Machine Codes:

:2800 – :283F for forward jumps (+1 thru +64)
:2840 – :287F for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if test condition not satisfied.
Offset by (D) if test condition satisfied.

Timing: 1

2.6.3.9

JXN JUMP IF X NOT ZERO

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	0	0	R	D FIELD					

A jump occurs if the X Register is not zero. Otherwise the next instruction in sequence is executed:

If (X)≠0, then Jump
If (X)=0, then (P)+1 → P

(X) are unchanged.

Machine Codes:

:3800 – :383F for forward jumps (+1 thru +64)
:3840 – :387F for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if test conditions not satisfied.
Offset by (D) if test condition satisfied.

Timing: 1

2.6.3.10

JOR JUMP IF OVERFLOW RESET

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	1	0	0	R	D FIELD					

A jump occurs if the overflow bit is reset (0). Otherwise the next instruction in sequence is executed.

If OV = 0, then Jump
If OV = 1, then (P)+1 → P

OV is unchanged.

Machine Codes:

:3200 – :323F for forward jumps (+1 thru +64)
:3240 – :327F for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if test condition satisfied.
Offset by (D) if test condition satisfied.

Timing: 1

2.6.3.11

JOS JUMP IF OVERFLOW SET

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	1	0	0	R	D FIELD					

A jump occurs if the overflow bit is set (1). Otherwise the next instruction in sequence is executed:

If OV = 1, then Jump and reset OV.
If OV = 0, then (P)+1 → P

OV is unconditionally reset by this instruction.

Machine Codes:

:2200 – :223F for forward jumps (+1 thru +64)
:2240 – :227F for backward jumps (0 thru –63)

Registers Affected:

OV Unconditionally reset.
P Incremented normally if test condition not satisfied.
Offset by (D) if test condition satisfied.

Timing: 1

2.6.3.12

JSS JUMP IF SENSE SWITCH SET

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	0	0	R	D FIELD					

A jump occurs if the sense switch is set down. Otherwise the next instruction in sequence is executed.

If SS ON, the Jump
If SS Off, then (P)+1 → P

Machine Codes:

:3400 – :343F for forward jumps (+1 thru +64)
:3440 – :347F for backward jumps (0 thru –63)

Registers Affected:

P Incremented normally if test condition not satisfied.
Offset by (D) if test condition satisfied.

Timing: 1

2.6.3.13

JSR JUMP IF SENSE SWITCH RESET

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	0	0	0	R	D FIELD					

A jump occurs if the SENSE SWITCH is reset up. Otherwise the next instruction in sequence is executed:

If SS Off, then Jump
If SS On, then (P)+1 → P

Machine Codes:

:2400 – :243F for forward jumps (+1 thru +64)
:2440 – :247F for backward jumps (0 thru –63)

Registers Affected:

P   Incremented normally if test condition not satisfied
     Offset by (D) if test condition satisfied.

Timing: 1

2.7 SHIFT INSTRUCTIONS

2.7.1 General

Shift instructions move bit patterns in the computer registers either right or left. Shifts may involve a single register (A or X), a single register and the Overflow (OV) indicator, or both the A and X registers and the OV indicator.

Shift instructions have a variety of uses in a computer. They may be used to pack and unpack data for Input/Output operations; they may be used to move specific data bits into the OV indicator for testing; they may be used for code conversions; they may be used for arithmetic operations. The ALPHA 16 and NAKED MINI 16 computers provide logical, arithmetic, and rotate shifts for these functions.

2.7.2 Single Register Shifts

Three types of single register shifts are available in the ALPHA 16 and NAKED MINI 16 computers: logical, arithmetic, and rotate. The general features and bit paths are described in the following paragraphs.

2.7.2.1 Logical Shifts. Logical single register shifts couple a computer register and the Overflow indicator together to form a 17-bit register. Since individual bits within the A and X registers cannot be tested directly, logical shifts are often used to move bits into OV for testing. Logical shifts may couple either the A or X register with OV for shifting.

Figure 2-9 illustrates a Logical Right shift. When a Logical Right shift is executed the entire 16-bit word within the specified register is shifted right. Bits shifted out of bit 0 of the register are shifted into the OV indicator. As each bit is shifted into OV, the last bit that occupied OV is lost. Bit positions vacated on the left are filled with zeros. The end result is that zeros are shifted into the register on the left (into bit position 15), and data is shifted right within the register. Bits shifted out of bit 0 of the register are shifted into OV. Bits shifted out of OV are lost.

Figure 2-10 illustrates a Logical Left shift. The operation is the same as for the right shift, except that the direction is left instead of right. As the data in the register is shifted left, zeros are shifted into bit 0 of the register. Bits shifted out of bit 15 of the register are shifted into the OV indicator. Bits shifted out of OV are lost.

The following examples illustrate logical single register shifts:

Right Shift	Register	OV
Original contents: A Reg.	0000 1010 0011 1111	0
Shift Right 1 place:	0000 0101 0001 1111	1
Shift Right 4 more places:	0000 0000 0101 0001	1

Left Shift	OV	Register
Original Contents: A Reg.	0	0000 1010 0011 1111
Shift Left 1 place:	0	0001 0100 0111 1110
Shift left 4 more places	1	0100 0111 1110 0000

2.7.2.2 Rotate Shifts. Rotate shifts operate in the same manner as logical shifts, except that no data is lost. Data

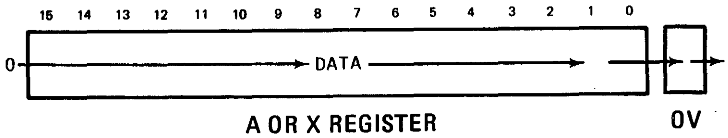


Figure 2-9. Logical Right Shift

shifted out of one end of the combined register is shifted into the other end.

Figure 2-11 illustrates the data path followed when a Rotate Right instruction is executed. The data bits within the register are shifted right. Bits shifted out of bit 0 are shifted into OV. Bits shifted out of OV are shifted into bit 15.

Figure 2-12 illustrates the data path followed when a Rotate Left instruction is executed. The data bits within the register are shifted left. Bits shifted out of bit 15 are shifted into OV. Bits shifted out of OV are shifted into bit 0.

The following examples illustrate Rotate shifts:

Rotate Left:	OV	Register
Original Contents:	1	1010 1111 0000 1100
Rotate Left 2 places:	0	1011 1100 0011 0011
Rotate Left 4 more places:	1	1100 0011 0011 0101

Rotate Right:	OV	Register
Original Contents:	1	1010 1111 0000 1100
Rotate Right 2 places:	0	0110 1011 1100 0011
Rotate Right 4 more places:	0	0110 0110 1011 1100

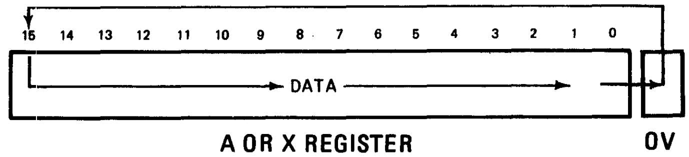


Figure 2-11. Rotate Right

2.7.2.3 Arithmetic Shifts. In general, logical shifts and rotate shifts are used to process data words which contain something other than numeric information. Arithmetic shifts are used to process numeric data.

A characteristic of numbers, regardless of the number base used, involves the shifting of numbers right or left one or more digit positions. For examples, if the decimal number

+150.

is shifted left one digit position, the following number is obtained:

+1500.

Shifting the number left one digit position causes the number to be multipled by 10. If the number +150 is shifted right one digit position, the following number is obtained:

+15.

The right shift causes the number to be divided by 10.

If octal numbers are shifted right or left in a like manner, the numbers are multiplied or divided by 8. Whenever any number in any base is shifted right or left, the number is

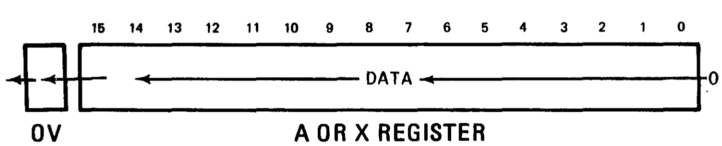


Figure 2-10. Logical Left Shift

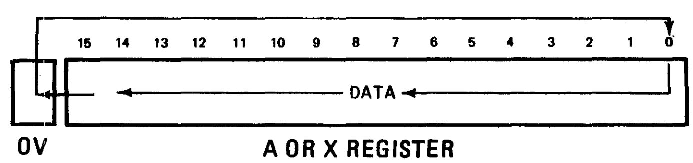


Figure 2-12. Rotate Left

divided or multiplied by the base of the number system. This characteristic holds true for negative numbers as well as positive numbers. If the number

–150.

is shifted left one digit position, the result is

–1500.

If the number is shifted right one digit position, the result is

–15.

Since this characteristic is true regardless of the base of the number system, it is true for binary numbers. If the binary number

0000 0000 0000 0110

is shifted left one bit position, the result is

0000 0000 0000 1100

which is the same number, multiplied by the base of the number system. The original number is equivalent to a decimal +6, and the second is equivalent to a decimal +12. If the original number is shifted right one bit position, the result is:

0000 0000 0000 0011

which is equivalent to the decimal number +3.

The ALPHA 16 and NAKED MINI 16 computers use binary two's complement numbers to represent negative numbers in memory and in computer registers. A characteristic of a two's complement number is that it has leading 1's instead of leading 0's. However, an arithmetic shift of a binary two's complement number must maintain the integrity of the number. A left shift must multiply the number by two for each bit postion shifted, and a right shift must divide the number by two for each bit position shifted. If the two's number complement

1111 1111 1111 1000

is shifted left one bit position, the result is

1111 1111 1111 0000

which is the correct result.

However, if the same original number is shifted right one bit position by a Logical Right shift the result is

0111 1111 1111 1100

which is not the correct result. A zero is shifted into the sign bit position, changing the number from negative to positive. The result is not a division by two.

To correct this condition, Arithmetic shift instructions divide the register being shifted into two parts: the sign, and the numeric value. Arithmetic shifts do not shift the sign bit. The sign bit remains unchanged, regardless of the number of bit positions shifted. In the case of left shifts, data bits are shifted to the left into bit 14 and out of bit 14. Bit 15 is not changed. Zeros are shifted into bit 0. Figure 2-13 illustrates the data path used for Arithmetic left shifts.

For right shifts, the sign in bit 15 is duplicated in bit 14 for each bit position shifted. The sign bit again remains unchanged. Bits shifted out of bit 0 are lost. Figure 2-14 illustrates the path followed by Arithmetic right shifts.

Note that the OV indicator is not used for Arithmetic shifts, and that only fifteen bits are shifted. The sign bit does not get shifted, but instead is duplicated in bit position 14 during right shifts.

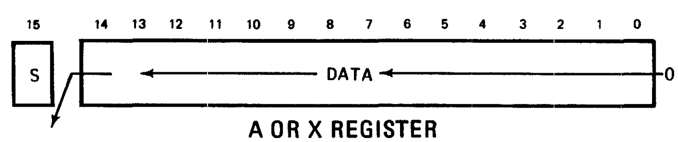


Figure 2-13. Arithmetic Left Shift

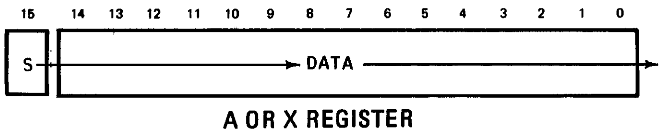


Figure 2-14. Arithmetic Right Shift

2.7.3 Double Register Shifts

Double register shifts couple the A Register, X Register, and OV indicator together for shifting operations. The two registers and the OV indicator act as a 33-bit register. There are two types of long shifts: Long Logical shifts, and Long Rotate shifts. Two long shifts, Multiply Step and Divide Step, are special cases of the Long Logical shift group.

2.7.3.1 Long Logical Shifts. Figure 2-15 illustrates the data path used for Long Logical right shifts. Zeros are shifted into bit 15 of the A Register, bits are shifted from bit 0 of A into bit 15 of X, bits are shifted from bit 0 of X into OV, and bits shifted out of OV are lost.

Figure 2-16 illustrates the data path used for Long Logical Left shifts. Zeros are shifted into bit 0 of X, bits are shifted from bit 15 of X into bit 0 of A, bits are shifted from bit 15 of A into OV, and bits shifted out of OV are lost.

2.7.3.2 Long Rotate Shifts. Figure 2-17 illustrates the data path used for long Rotate right shifts, and Figure 18 illustrates the data paths used for Long-Rotate left shifts. The Long Rotate shifts are similar to the Single Register Rotate shifts except that both the A and X registers are involved in the shifts.

2.7.4 Shift Instruction Formats

Shift instructions are a special case of the Register Change class of instructions. Figure 2-19 illustrates the format for

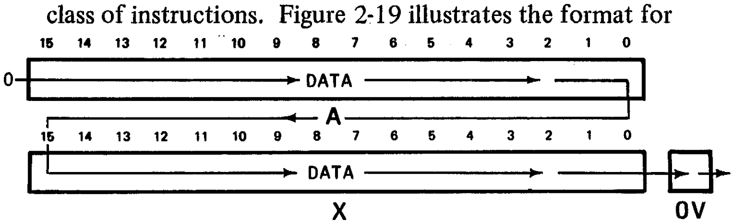


Figure 2-15. Long Right Shift

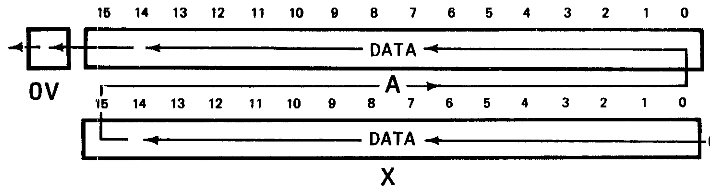


Figure 2-16. Long Left Shift

Single Register shifts, and Figure 2-20 illustrates the format for Long shifts. Bits 12-15 identify the Shift class, and bit 11 specifies Single Register or Long shift.

2.7.4.1 Single Register Format. A zero in bit 11 of a shift instruction identifies a Single Register shift. The shift code is contained in bits 3-10. The shift code identifies the type of shift to be performed (shift Op Code). The K Field, bits 0-2, specify the number of bit positions to be shifted. The formula for determining the number of bits to be shifted is 1+K. The maximum shift distance is 8 bit positions, since the maximum value which can be contained in K is 7. If K contains a value of 0, the shift instruction will shift one bit position (1+0=1). If K contains a value of 5, the shift instruction will shift 6 bit positions (1+5=6).

2.7.4.2 Long Shift Format. A one in bit 11 of a shift instruction identifies a Long shift. The shift code, identifying the type of shift to be performed, is contained in bits 4-10. The K Field, in bits 0-3, specifies the number of bit positions to be shifted. Note that the K Field of the Long Shift format contains four bit positions instead of three. Therefore the maximum number of bit positions that can be shifted by a Long shift is 16 instead of 8. The formula for calculating the number of bit positions to be shifted is again 1+K, where K has a maximum value of 15 (:F).

2.7.5 Shift Timing

The ALPHA 16 and NAKE MINI 16 have the capability of shifting one bit position each time data is passed through

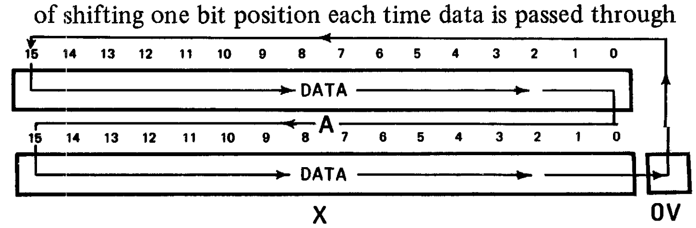


Figure 2-17. Long Rotate Right

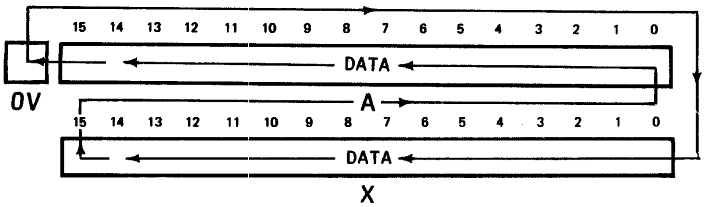


Figure 2-18. Long Rotate Left

the Adder and Shift Control logic of the computer. In order to shift more than one bit position, the computer execution cycle must be "stretched" to pass the data to be shifted through the Adder and Shift Control once for each bit position to be shifted. For long shifts, data from two registers must be passed through the Adder and Shift Control logic. This requires additional time. However, it is not necessary to repeat the entire computer cycle for each bit position to be shifted. It is necessary to repeat only a portion of the cycle.

2.7.5.1 Single Register Shift Timing. For single register shifts, the cycle must be stretched by 1/4 cycle for each bit position to be shifted beyond the first bit position. If data is to be shifted only one bit position, the shift can be completed in a single cycle. If data is to be shifted two bit positions, one cycle is required for the first bit position and an additional 1/4 cycle is required for the next. For a three position shift, one cycle is required for the first position, 1/4 cycle for the second, and 1/4 cycle for the third, for a total of 1-1/2 cycles. The formula for calculating the number of cycles required for a Single Register shift is:

$$1 + (1/4)K$$

where K is the value in the K Field of the shift instruction.

2.7.5.2 Long Shift Timing. For Long shifts, the cycle must be stretched by 1/4 cycle for the first bit position to be shifted and by 1/2 cycle for each additional bit position to be shifted. The additional 1/4 cycle for each bit position is

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	SHIFT CODE								K		

Figure 2-19. Single Register Shift Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	SHIFT CODE							K			

Figure 2-20. Long Shift Format

required because the contents of two registers must be passed through the Adder and Shift Control logic of the computer. The formula for calculating the number of cycles required for Long shifts is:

$$1\text{-}1/4 + (1/2)K$$

2.7.6 Instruction Descriptions

The Shift instruction descriptions follow the same general format as the Memory Reference instruction with these exceptions:

1. A shift path diagram is shown for each instruction.
2. The Machine Codes portion of the description includes the hexadecimal code for a minimum (1 place) shift and a maximum (8 or 16 place) shift.

The Multiply Step and Divide Step instructions are handled as special cases. In addition to the instruction description, programming examples are included to clarify the use of these very powerful instructions.

2.7.6.1

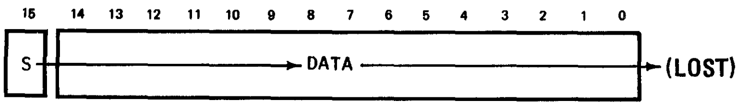
ARA ARITHMETIC SHIFT A RIGHT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	1	1	0	1	0	K		

The contents of the A Register are shifted right 1+K places. The sign bit (bit 15) is unchanged and is shifted into and propagated through bit 14. Bits shifted out of bit 0 are lost.

Shift Path: A Register.



Machine Codes:

:10D0 for 1 place shift
thru
:10D7 for 8 place shift

Registers Affected:

A Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.2

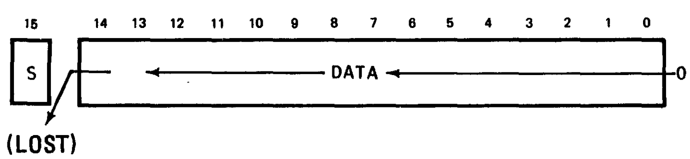
ALA ARITHMETIC SHIFT A LEFT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2 1 0
0	0	0	1	0	0	1	0	0	1	0	1	0	K

The contents of the A Register (bits 0-14) are shifted left 1+K places. The sign bit (bit 15) is unchanged. Zeros are shifted into bit 0, and bits shifted out of bit 14 are lost.

Shift Path: A Register



Machine Codes:

:1050 for one place shift
thru
:1057 for eight place shift

Registers Affected:

A Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.3

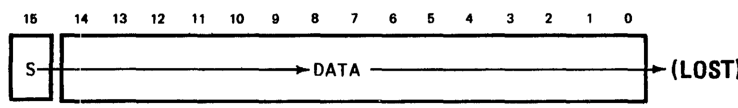
ARX ARITHMETIC SHIFT X RIGHT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2 1 0
0	0	0	1	0	0	0	0	1	0	1	0	1	K

The contents of the X Register are shifted right 1+K places. The sign bit (bit 15) is unchanged and is shifted into and propagated through bit 14. Bits shifted out of bit 0 are lost.

Shift Path: X Register



Machine Codes:

:10A8 for 1 place shift
thru
:10AF for 8 place shift

Registers Affected:

X Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.4

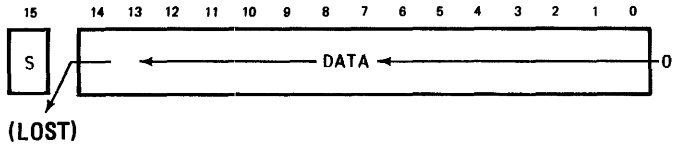
ALX ARITHMETIC SHIFT X LEFT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2 1 0
0	0	0	1	0	0	0	0	0	0	1	0	1	K

The contents of the X Register (bits 0-14) are shifted left 1+K places. The sign bit (bit 15) is unchanged. Zeros are shifted into bit 0, and bits shifted out of bit 14 are lost.

Shift Path: X Register



Machine Codes:

:1028 for 1 place shift
thru
:102F for 8 place shift

Registers Affected:

X Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.5

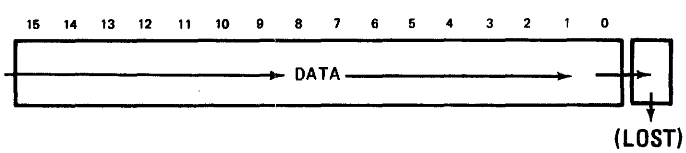
LRA LOGICAL SHIFT A RIGHT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	1	1	1	0	1	0	K		

The contents of the A Register are shifted right 1+K places through OV. Zeros are shifted into bit 15. Bits are shifted from bit 0 of A into OV. Bits shifted out of OV are lost. A and OV set as a 17-bit register.

Shift Path: A Register and OV



Machine Codes:

:13D0 for a 1 place shift
thru
:13D7 for 8 place shift

Registers Affected:

A,OV Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.6

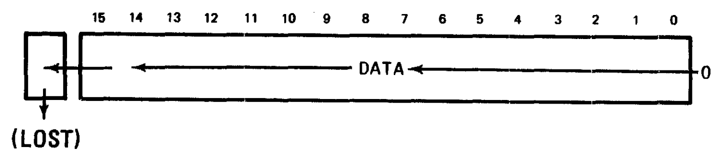
LLA LOGICAL SHIFT A LEFT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	1	0	1	0	1	0	K		

The contents of the A Register are shifted left 1+K places through OV. Zeros are shifted into bit 0. Bits are shifted from bit 15 of A into OV. Bits shifted out of OV are lost. A and OV act as a 17 bit register.

Shift Path: A Register and OV



Machine Codes:

:1350 for 1 place shift
thru
:1357 for 8 place shift

Registers Affected:

A,OV Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.7

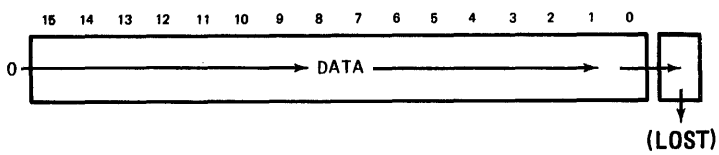
LRX LOGICAL SHIFT X RIGHT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	1	1	0	1	0	1	K		

The contents of the X Register are shifted right 1+K places through OV. Zeros are shifted into A_{15}, bits are shifted from A_{00} into OV, and bits shifted out of OV are lost. X and OV act as a 17 bit register.

Shift Path: X Register and OV



Machine Codes:

:13A8 for 1 place shift
thru
:13AF for 8 place shift

Registers Affected:

X,OV Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.8

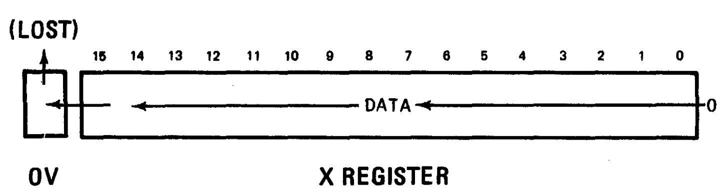
LLX LOGICAL SHIFT X LEFT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	1	0	0	1	0	1	K		

The contents of the X Register are shifted left 1+K places through OV. Zeros are shifted into bit 0, bits are shifted from X_{15} to OV, and bits shifted out of OV are lost. X and OV act as a 17-bit register.

Shift Path: X Register, OV



Machine Codes:

1328 for 1 place shift
thru
:132F for 8 place shift

Registers Affected:

X,OV Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.9

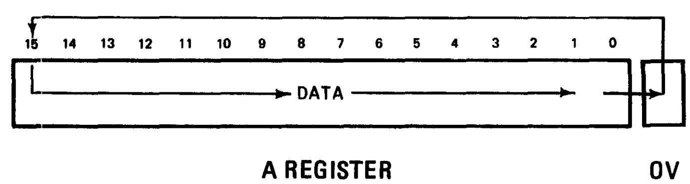
RRA ROTATE A RIGHT WITH OV

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1	1	1	0	1	0	K		

The Contents of the A Register are shifted right 1+K places through the OV flip-flop. OV is shifted into bit 15, and bit 0 of A is shifted into OV. No bits are lost when this shift is executed. A and OV act as a 17-bit register.

Shift Path: A Register and OV



Machine Codes:

:11D0 for 1 place shift
thru
:11D7 for 8 place shift

Registers Affected:

A,OV Previous contents replaced by results of shift.

Timing: 1 + 1/4K

2.7.6.10

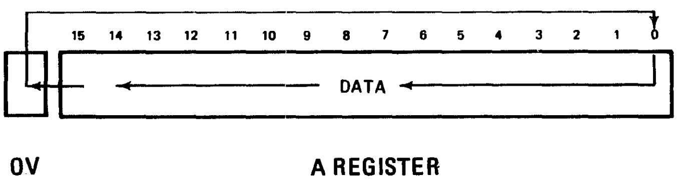
RLA ROTATE A LEFT WITH OV

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1	0	1	0	1	0	K		

The contents of the A Register are shifted left 1+K places through the OV fip-flop. OV is shifted into bit 0, and bit 15 is shifted into OV. No bits are lost when this shift is executed. A and OV act as a 17-bit register.

Shift Path: A Register and OV



Machine Codes:

:1150 for 1 place shift
thru
:1157 for 8 place shift

Registers Affected:

A,OV Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.11

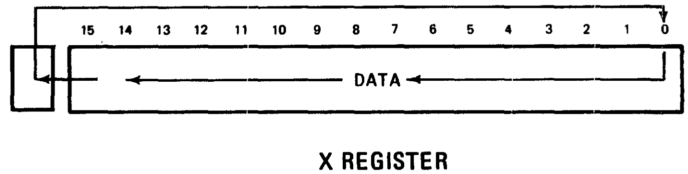
RLX ROTATE X LEFT WITH OV

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1	0	0	1	0	1	K		

The contents of the X Register are shifted left 1+K places through the OV flip-flop. OV is shifted into bit 0, and bit 15 is shifted into OV. No bits are lost when this shift is executed. X and OV act as a 17-bit register.

Shift Path: X Register and OV



Machine Codes:

:1128 for 1 place shift
thru
:112F for 8 place shift

Registers Affected:

A,OV Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.12

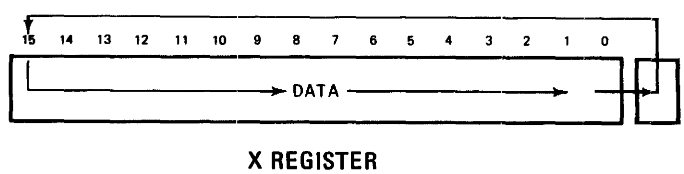
RRX ROTATE X RIGHT WITH OV

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1	1	0	1	0	1	K		

The contents of the X Register are shifted right 1+K places through the OV flip-flop. OV is shifted into bit 15, and bit 0 is shifted into OV. No bits are lost when this shift is executed. X and OV act as a 17-bit register.

Shift Path: X Register and OV



Machine Codes:

:11A8 for 1 place shift
thru
:11AF for 8 place shift

Registers Affected:

X,OV Previous contents replaced by result of shift.

Timing: 1 + 1/4K

2.7.6.13

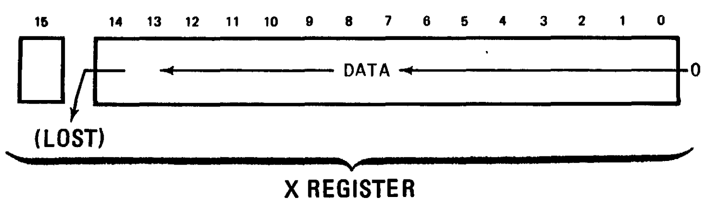
NOR NORMALIZE X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	0	0	1	0	1	K		

The contents of the X Register are arithmetically shifted left 1+K places or until X_{15} is not equal to X_{14}, whichever occurs first. Zero is shifted into X00. When $X_{15} \neq X_{14}$, the remaining shifts are inhibited and OV will be set to indicate the contents of X are normalized. Bits shifted out of bit 14 are lost.

Shift Path: X Register



Machine Codes:

:1228 for 1 place shift
thru
:122F for 8 place shift

Registers Affected:

$X_{0\text{-}14}$ Previous contents replaced by result of shift.
OV Set if $X_{15} \neq X_{14}$.
Unchanged if all shifts executed and $X_{15} = X_{14}$

Timing: 1 + 1/4K

2.7.6.14

SAO SIGN OF A TO OV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0

Copy the sign of the A Register in the Overflow indicator:

$$(A_{15}) \rightarrow OV$$

Machine Code:

:1340

Registers Affected:

OV Previous contents replaced by sign of A.

Timing: 1

2.7.6.15

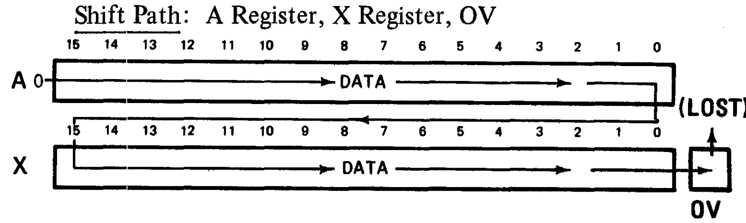
LLR LONG LOGICAL SHIFT RIGHT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	1	1	0	0	0	K			

The contents of the A and X Registers are logically shifted right through OV 1+K places. For each bit position shifted, zero is shifted into A_{15}, A_{00} is shifted into X_{15}, and X_{00} is shifted into OV. The previous contents of OV are lost. A, X and OV act as a 33-bit register.

Shift Path: A Register, X Register, OV



Machine Codes:

:1B80 for 1 place shift
thru
:1B8F for 8 place shift

Registers Affected:

A,X,OV Previous contents replaced by result of shift.

Timing: 1-1/4 x 1/2K

2.7.6.16

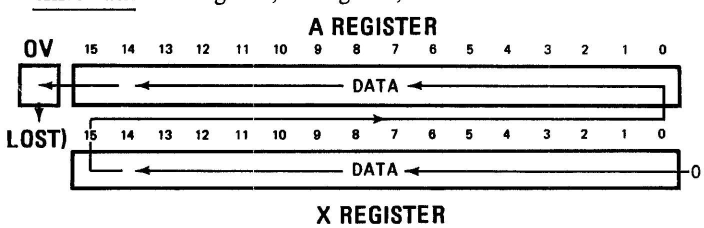
LLL LONG LOGICAL SHIFT LEFT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	1	0	0	0	0	K			

The contents of the A and X Registers are logically shifted left through OV 1+K places. For each bit position shifted, zero is shifted into X_{00}, X_{15} is shifted into A_{00}, and A_{15} is shifted into OV. The previous contents of OV are lost. A, X and OV act as a 33-bit register.

Shift Path: A Register, X Register, OV



Machine Codes:

:1B00 for 1 place shift
thru
:1B0F for 16 place shift

Registers Affected:

A,X,OV Previous contents replaced by result of shift.

Timing: 1-1/4 x 1/2K

2.7.6.17

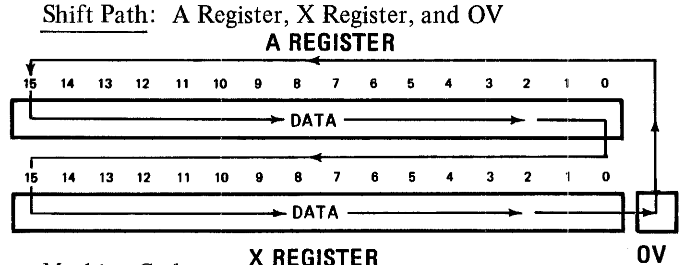
LRR LONG ROTATE RIGHT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	1	1	0	0	0	K			

Contents of A and X Registers are shifted right through OV 1+K places. OV is shifted into A_{15}, X_{00} is shifted into OV, and A_{00} is shifted into X_{15}. A, X, and OV act as a 33-bit register. No bits are lost when this shift is executed.

Shift Path: A Register, X Register, and OV



Machine Codes:

:1980 for 1 place shifts
thru
:198F for 16 place shifts

Registers Affected:

A,X,OV Previous contents replaced by result of shift.

Timing: 1-1/4 + 1/2K

2.7.6.18

LRL LONG ROTATE LEFT

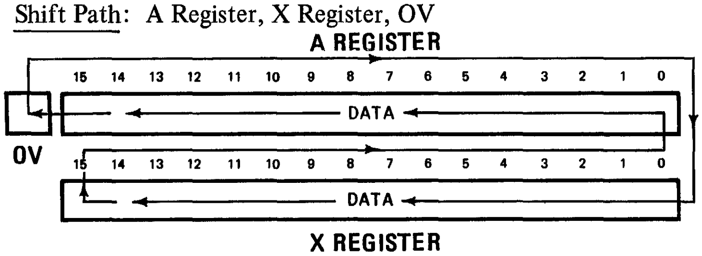
I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	1	0	0	0	0	K			

Contents of A and X Registers are shifted left through OV 1+K places. OV is shifted into X_{00}, A_{15} is shifted into OV.

X_{15} is shifted into A_{00}. A, X, and OV act as a 33-bit register.

Shift Path: A Register, X Register, OV



Machine Codes:

:1900 for 1 place shift
thru
:190F for 16 place shift

Registers Affected:

A,X,OV   Previous contents replaced by result of shift.

Timing: 1-1/4 x 1/2K

2.7.6.19

MPS   MULTIPLY STEP

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3 2 1 0
0	0	0	1	1	0	0	1	1	0	1	0	K

The Multiply Step instruction is used to code fast multiply routines. It performs all of the shifts and conditional adds necessary to multiply two 15-bit numbers and produce a 30-bit product. MPS is not a complete multiply instruction, but it does perform the time consuming shift/test/add loop that is the heart of all software multiply routines.

MPS multiplies a signed 15-bit number in the R Register by a signed 15-bit number in the X Register. It produces a 30-bit product in the A and X registers. The product produced by MPS is not a standard double-precision format, and the upper 15 bits of the product (in the A Register) will require software correction if the Multiplier is negative.

The following is a typical software multiply routine using MPS. This routine will multiply two 15-bit numbers to produce a 30-bit product. The product will be in standard double-precision format, and the upper 15 bits of the product will be corrected if the Multiplicand was negative:

LDX	MPLR	Place the Multiplier in the X Register.
RRX	1	Pre-shift X. Place the LSB of the Multiplier in OV, and save previous contents of OV in X_{15}.
SIN	3	Suppress Interrupts. The Multiplicand will be loaded in the R Register, which cannot be saved by software. Interrupts must be disabled until MPS is executed.
LDA	MCND	This instruction loads the Multiplicand into R and A simultaneously.
ZAR		Clear the A Register, but don't change R Register.
MPS	15	Do a 15-bit multiply. Form the product in A and X.
JOR	$+2	OV will be set if the Multiplier was negative (sign bit = 1). Skip the next instruction if sign is +.
SUB	MCND	Subtract the Multiplicand from the upper 15 bits of the product if the Multiplier was negative. This corrects the product.
LRX	1	Shift the X Register right one place. This restores the original contents of OV, and separates the product into standard double-precision format.

Several points should be noted in this Multiply routine.

1. The second instruction (RRX 1) shifts the Multiplier right one place and saves the contents of OV. If this is not done, OV must first be cleared, and the MPS instruction must be MPS 16 rather than MPS 15. The method used in this example is shorter and also saves OV.

2. The third instruction (SIN 3) suppresses interrupts for three instructions. The R Register may be loaded by the programmer, but it can't be saved by the programmer. Any interrupt will destroy the contents of R, therefore interrupts must be suspended until the contents of R are no longer needed.

3. The fourth and fifth instructions (LDA MCND, ZAR) load the multiplicand into the R Register and clear the A Register. The MPS instruction forms the partial product in the A Register by conditionally adding the contents of R to the contents of A and then shifting the partial product into X as each bit of the Multiplier is shifted into OV for testing. If the A Register contains some prior value, the product generated will be the product of the Multiplier and the Multiplicand, plus the value in A.

   NOTE: An alternate method of loading R is:

CMS	MCND	Load Multiplicand into R, but don't disburb A or X
JMP	$+2	Jump to MPS if MCND greater than (A)
NOP		Filler if MCND less than (A)

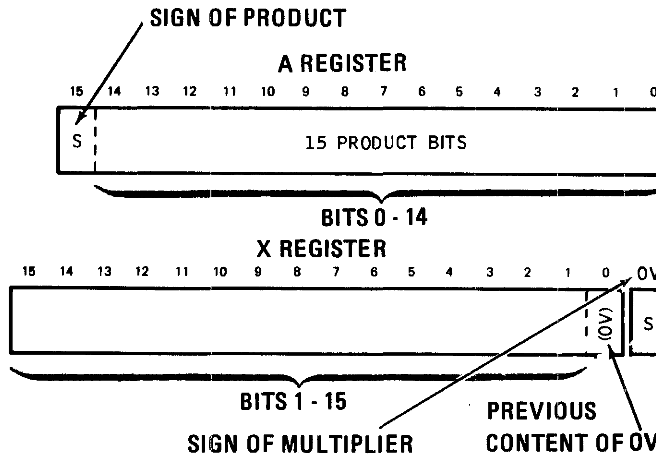
4. The sixth instruction (MPS 15) does the actual multiply. The algorithm used is:

   a. Test OV. If OV=1, add (R)+(A) and store result in A. If OV=0, do not add.

   b. Shift A and X right one place. This is a Long Logical Right shift. A_0 goes to X_{15} and X_0 goes to OV.

   c. Test for expiration of shift count. If all shifts are done, exit. If more shifts to be done, go back to step a.

This algorithm forms the partial product in A and X. The sign of the product is in A_{15}; The fifteen most significant bits of the product are in bits 0 – 14 of A. The fifteen least significant bits of the product are in bits 1 – 15 of X. Bit 0 of X contains the original contents of OV, and OV contains the sign of the Multiplier. The register conditions at the end of MPS are:



The product extends from A_{14} through X_1. If the sign of the multiplier is negative (OV=1), then the product bits in A will require correction. The product at this point is not in standard double-precision format. The remaining instructions in this routine correct the upper 15 bits of the product, restore the original contents of OV, and put the product in standard double-precision format.

5. The two instructions following MPS(JOR $+2, SUB MCND) test the sign of the Multiplier and correct the product if the Multiplier was negative. The correction requirement is the result of the multiplication of two's complement numbers. If the Multiplicand is negative in this routine, the sign of the product indicates that the product is negative. If the Multiplier is positive,

then the sign of the product is correct and the product requires no further correction. The same holds true if the Multiplier and Multiplicand are both positive.

However, if the Multiplier is negative with either a positive or negative Multiplicand, the sign of the product is wrong, and the leading bits require correction. The correction may be performed very easily. The SUB MCND instruction performs the total product correction that may be required.

6. The last instruction (LRX 1) places the product in standard double-precision format. Standard double precision format is as follows:

15	14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
S	15 PRODUCT BITS

A REGISTER

15	14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0	15 PRODUCT BITS

X REGISTER

In standard double precision format the sign of the number is in bit 15 of the word containing the 15 most significant bits of the number. Bit 15 of the word containing the 15 least significant bits of the number always contains a 0.

Machine Codes:

:19A0 for 1 bit multiply
thru
:19AF for 16 bit multiply

Registers Affected:

A,X Previous contents replaced by product.

Timing: (MPS instructions only) 1-1/4 + (1/2)K

2.7.6.20

DVS DIVIDE STEP

15	14	13	12	11	10	9	8	7	6	5	4	3 2 1 0
0	0	0	1	1	0	0	1	0	1	0	0	K

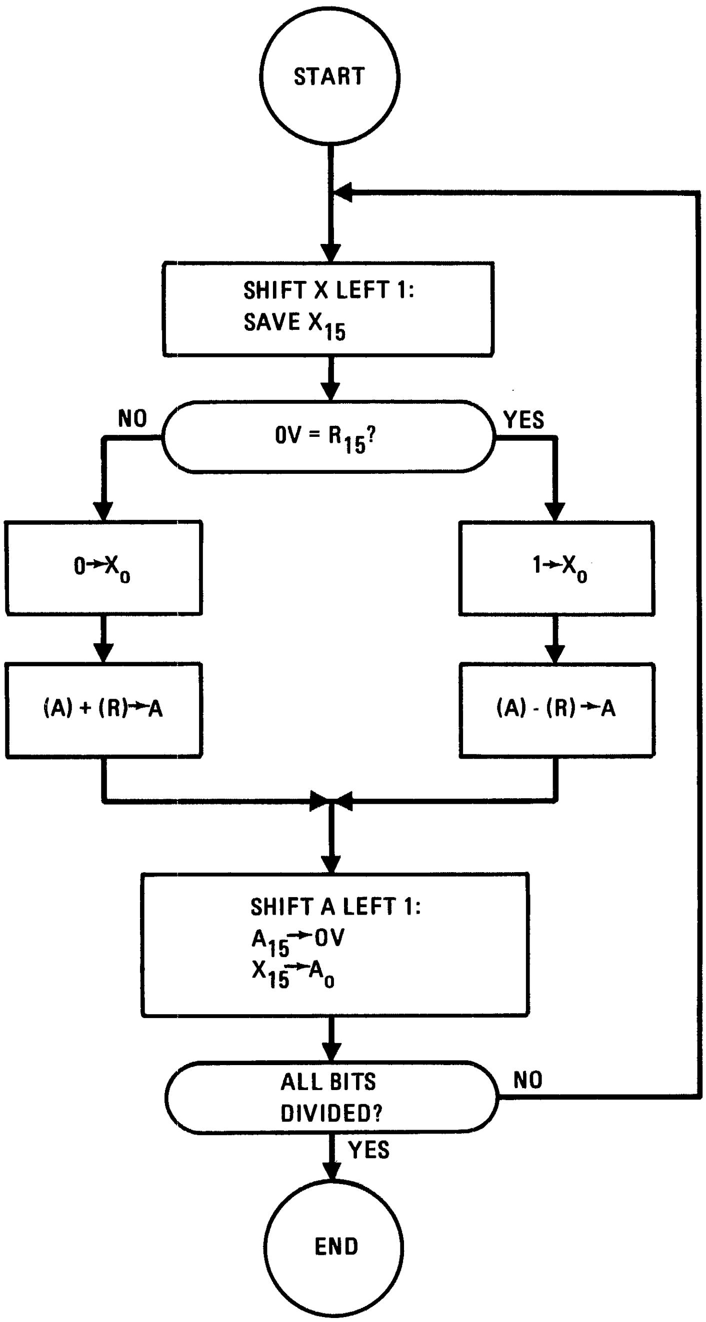
The Divide Step (DVS) instruction is used to code fast software divide routines. It performs the shifts and adds or subtracts necessary to divide a 30-bit dividend in the A and X Registers by a 15-bit divisor in the R Register. When the DVS instruction is completed, the quotient is in the X Register and the remainder is in the A Register.

The DVS instruction is not a complete divide instruction, but instead performs the repetitive shift/test/add or subtract loop of a non-restoring divide algorithm. DVS does not test for divide faults (quotient too large for a single-precision register). The quotient is not rounded by DVS, and the remainder may require correction. However, the DVS instruction requires only 9-1/4 cycles to complete the shift/test/add or subtract loop for a 16-bit divide as compared to over 60 cycles if DVS is not used.

Page 2-54 contains a flow chart of the functional operation of the DVS instruction.

Fractional Divide Example. The following is an example of fractional divide where the numerator must be less than the denominator in absolute magnitude. This routine assumes that the remainder is insignificant, and that the quotient need not be rounded (least significant bit may be off by 1). This routine does check for divide faults, and does save the remainder. Page 2-55 contains a flow chart of a fractional divide. The flow chart is followed by a sample program which implements the flow chart.

DIVIDE STEP: FUNCTIONAL FLOW CHART



START CONDITIONS DEPEND ON SOFTWARE ROUTINE BEING USED, NORMALLY $0V = A_{15}$.

ALL BITS OF X ARE SHIFTED LEFT 1 PLACE. X_{15} IS SAVED IN A TEMPORARY STORE. THE NEXT QUOTIENT BIT IS PLACE IN X_0 ACCORDING TO THE FOLLOWING TEST.

IF $0V = A_{15}$, STORE A 1 IS THE QUOTIENT, X_0, AND SUBTRACT THE DIVISOR FROM THE UPPER 15 BITS OF THE DIVIDEND.

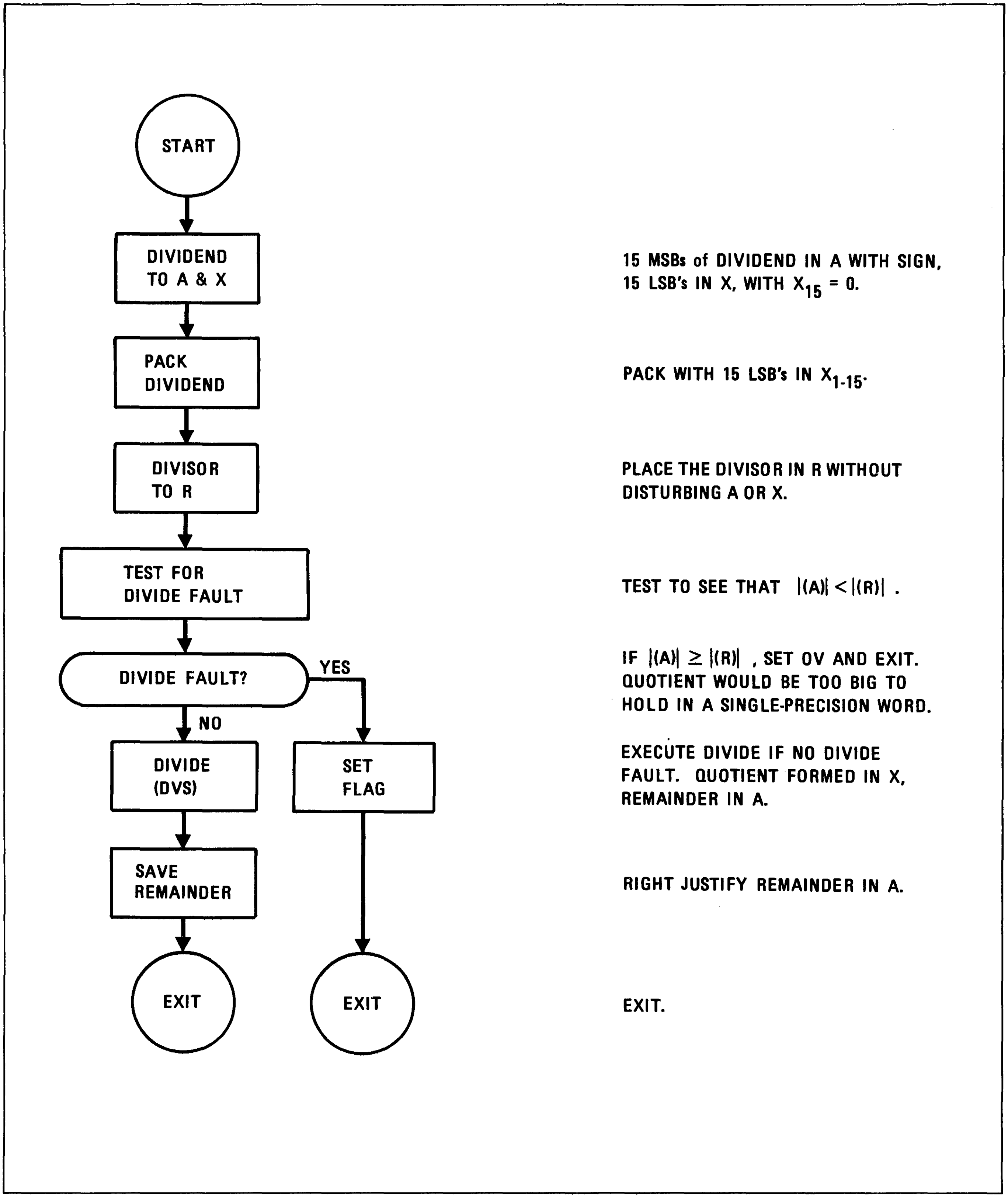
IF $0V \neq A_{15}$, STORE A 0 IN THE QUOTIENT AND ADD THE DIVISOR TO THE UPPER 15 BITS OF THE DIVIDEND.

A IS SHIFTED LEFT 1 PLACE. THE SIGN OF A, A_{15}, IS SHIFTED INTO 0V. THE SAVED DIVIDEND BIT FROM X_{15} (SEE FIRST BLOCK) IS SHIFTED INTO A0.

THE SHIFT COUNT IS TESTED TO SEE IF ALL BITS HAVE BEEN DIVIDED. IF NOT, THE LOOP IS REPEATED.

AT THE END OF A 16-BIT DIVIDE, THE QUOTIENT IS IN X AND THE REMAINDER IN A, SHIFTED LEFT 1 PLACE. THE SIGN OF THE REMAINDER IS IN 0V.

FRACTIONAL DIVIDE FLOW CHART



Fractional Divide Program

Place double-precision dividend in the A and X Registers. Dividend is assumed to be in standard double precision format.

1.	LDA HDND	15 MSB's to A. A_{15} = sign.
2.	LDX LDND	15 LSB's to X.

The dividend is packed so that it extends from A_{15} to X_1. This removes the insignificant O-bit from X_{15}.

3.	LLX 1	Pack dividend.

For purposes of testing, the sign of the dividend is copied in the OV indicator. This is necessary because the test for a Divide Fault for a negative dividend is not the same as for a positive dividend.

4.	SAO	Copy dividend sign in OV

Since the divisor will be placed in the R Register, and since (R) cannot be saved by software, interrupts must be inhibited so long as the R Register contains significant information.

5.	SIN 7	Suppress interrupts for 7 instructions.

The R Register must be loaded with the divisor without changing the contents of A or X. The only Memory Reference instruction which can accomplish this task is CMS.

6.	CMS DVSR	Load R with divisor.
7.	JMP $+2	Filler from CSM. The JMP or the NOP may be executed, but not both.
8.	NOP	

The next step is to test for a potential Divide Fault. The first step is to see if the dividend is positive or negative and go to the appropriate test.

9.	JOR	DIV1	If the dividend is positive, go to DIV1 to test a positive dividend. Otherwise, continue for negative test.

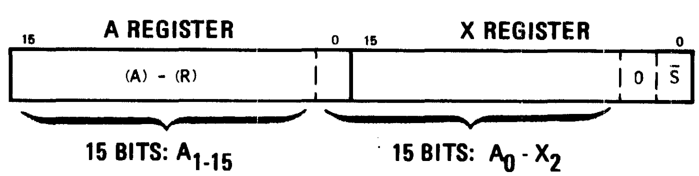
The Divide Fault test is made by attempting a 1-bit divide and then testing the OV indicator. For a negative dividend, OV must be reset. If OV is set, a Divide Fault condition exists.

10.	DVS	1	Perform 1-bit divide.
11.	JOR	DIV2	If OV is reset after the 1-bit divide, there is no Divide Fault. Jump to completion of Divide.
12.	JMP	DIV4	If OV is set, there is a Divide Fault. Jump to EXIT routine.

The Divide Fault test for a positive dividend is the same as for a negative dividend except for the test following the 1-bit divide. If OV is set, there is no Divide Fault. If OV is reset, there is a Divide Fault.

13.	DIV1	DVS 1	Perform 1-bit divide.
		JOR DIV3	If OV is reset after the 1-bit divide, a Divide Fault exists. Jump to the EXIT routine. If OV is set, continue with the completion of the divide.

If no Divide Fault condition exists, the next step is to perform the divide. The condition that exists in the A and X registers and the OV indicator at this point are:



- A Register, bits 1-15, contains the difference between the 15 MSB's of the dividend and the divisor. This is because the DVS 1 instruction subtracted (R) from (A) and shifted the result left one place. Bit 0 of A contains the previous contents of bit 15 of X due to the long left shift.

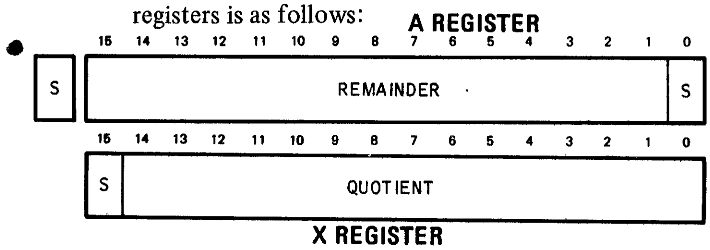
- X Register, bits 2-15, contains the 14 LSB's of the dividend due to the left shift. Bit 1 of X contains a 0-bit, and bit 0 of X contains the complement of the eventual sign of the quotient.

- OV contains the sign of the remainder.

The divide is performed by executing the DVS instruction for a 16-bit divide.

14. DIV2 DVS 16 Do the divide.

When the divide is completed, the status of the registers is as follows:



- X Register contains the quotient and its proper sign. The quotient is unrounded, therefore X_0 may be incorrect.

- A Register contains the remainder in bits 1-15. Bit 0 contains an insignificant sign bit.

- OV contains the sign of the remainder.

The next step is to right justify the remainder and insert its proper sign bit.

15. RRA 1 Rotate A right 1 place. (OV) → A_{15}. Right justify A_{1-15} to A_{0-14}.

The divide is now complete. The remaining steps are housekeeping. The EXIT routine must be included, and a flag must indicate whether or not a Divide Fault occurred. The conventions used by this routine at the EXIT are:

OV=0 No Divide Fault. A contains uncorrected remainder if A is negative. X contains unrounded quotient.

OV=1 Divide Fault detected. A and X contain insignificant data.

If there is no Divide Fault and the divide was executed, OV must be conditioned prior to going to the EXIT routine.

16. SOV The EXIT routine will complement OV, so it is set before going to EXIT. The EXIT routine, for Divide Fault and normal divide is as follows:

17. DIV3 COV Complement OV if the step is entered from the normal divide or from positive Divide Fault test.

18. DIV4 (EXIT) End of program for all conditions. This is normally an indirect JMP back to a main program.

Machine Codes:

:1940 for 1 place divide
thru
:194F for 16 place divide

Registers Affected (DVS instruction only):

A	Contains remainder (uncorrected) in bits 1-15 for 16 place divide.
X	Contains signed quotient for 16 place divide.
OV	Contains sign of remainder.

Timing: 1-1/4 + 1/2K

2.8 REGISTER CHANGE INSTRUCTIONS

2.8.1 General

Register Change instructions are those instructions which perform arithmetic and logical operations involving the A and X registers without requiring data from memory. Operations are performed using the contents of A and X only. The overflow (OV) indicator may be affected as a result of the operations performed.

2.8.2 Instruction Format

Figure 2-21 illustrates the format of the Register Change instructions. Bits 11-15 define the Register Change instruction class. The operation code defining specific instructions is contained in bits 3-10.

Two instructions use a special format. These instructions are the Input Data Switches to A (ISA) and Input Data Switches to X (ISX). These instructions are special cases of the Input/Output instruction class. They are actually coded as Unconditional Input instructions with Device Address 0 and Function Code 1. The use of I/O instructions for special computer functions is discussed in Part 3.1.4 of Section 3.

2.8.3 Instruction Descriptions

The following paragraphs describe the Register Change instructions. The descriptions follow the same format as that used for Memory Reference instructions.

2.8.3.1

ZAR ZERO A REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0

Sets contents of A Register to Zero.

:0000 → A

Previous contents of A are lost.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	OP CODE								0	0	0

Figure 2-21. Register Change Format

Machine Code:

:0110

Registers Affected:

A Previous contents replaced by :0000.

Timing: 1

2.8.3.2

ZXR ZERO X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0

Sets contents of X Register to Zero (:0000).

:0000 → X

Previous contents of X are lost.

Machine Code:

:0108

Registers Affected:

X Previous contents replaced by :0000

Timing: 1

2.8.3.3

ZAX ZERO A AND X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0

Sets contents of A and X Registers to Zero.

:0000 → A

:0000 → X

Previous contents of A and X are lost.

Machine Codes:

:0118

Registers Affected:

A Previous contents replaced by :0000

X Previous contents replaced by :0000

Timing: 1

2.8.3.4

ARM SET A REGISTER TO MINUS 1

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Sets contents of A Register to −1 (:FFFF).

−1 → A

Previous contents of A are lost.

Machine Code:

:0010

Registers Affected:

A Previous contents replaced by :FFFF.

Timing: 1

2.8.3.5

XRM SET X REGISTER TO MINUS 1

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Sets contents of X Register to −1 (:FFFF).

−1 → X

Previous contents of X are lost.

Machine Code:

:0008

Registers Affected:

X Previous contents replaced by :FFFF.

Timing: 1

2.8.3.6

AXM SET A AND X REGISTER TO MINUS 1

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0

Sets contents of A and X Registers to −1 (:FFFF).

−1 → A

−1 → X

Previous contents of A and X are lost.

Machine Code:

:0018

Registers Affected:

A Previous contents replaced by :FFFF.

X Previous contents replaced by :FFFF.

Timing: 1

2.8.3.7

ARP SET A REGISTER TO PLUS 1

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0

Sets contents of A Register to plus 1 (:0001).

:0001 → A

Previous contents of A are lost.

Machine Codes:

:0350

Registers Affected:

A Previous contents replaced by +1 (:0001).

Timing: 1

2.8.3.8

XRP SET X REGISTER TO PLUS 1

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0

Sets contents of X Register to plus 1 (:0001).

:0001 → X

Previous contents of X are lost.

Machine Code:

:0528

Registers Affected:

X Previous contents replaced by :0001.

Timing: 1

2.8.3.9

AXP SET A AND X REGISTERS TO PLUS 1

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0

Sets contents of A and X Registers to plus 1 (:0001).

:0001 → A

:0001 → X

Previous contents of A and X are lost.

Machine Code:

:0358

Registers Affected:

A Previous contents replaced by :0001.

X Previous contents replaced by :0001.

Timing: 1

2.8.3.10

DAR DECREMENT A REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0

Subtracts one from the contents of A Register and places results in A.

(A) –1 → A

OV set if previous (A) = :8000.

Machine Code:

:00D0

Registers Affected:

A Contents decremented

OV Set if previous (A) = $-32{,}768_{10}$ (:8000).

Timing: 1

2.8.3.11

DXR DECREMENT X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0

Subtracts one from the contents of X Register and places result in X.

$$(X) -1 \rightarrow X$$

OV set if previous (X) = :8000.

Machine Code:

:00A8

Registers Affected:

X Contents decremented.

OV Set if previous (X) = $-32{,}768_{10}$ (:8000).

Timing: 1

2.8.3.12

IAR INCREMENT A REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0

Adds one to contents of A Register and places results in A.

$$(A) +1 \rightarrow A$$

OV set if (A) = :7FFF.

Machine Code:

:0150

Registers Affected:

A Contents incremented.

OV Set if previous (A) = $32{,}767_{10}$ (:7FFF)

Timing: 1

2.8.3.13

IXR INCREMENT X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0

Adds one to the contents of the X Register and places result in X.

$$(X) +1 \rightarrow X$$

OV set if previous (X) = :7FFF.

Machine Code:

:0128

Registers Affected:

X Contents incremented.

OV Set if previous (X) = 32,767 (:7FFF).

Timing: 1

2.8.3.14

NAR NEGATE A REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0

Performs 2's complement of contents of A Register and places result in A.

$$-(A) \rightarrow A$$

OV set if previous (A) = :8000.

Machine Codes:

:0310

Registers Affected:

A Contents negated.

OV Set if (A) = –32,768 (:8000).

Timing: 1

2.8.3.15

NXR NEGATE X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0

Performs 2's complement of contents of X Register and places result in X.

$$-(X) \rightarrow X$$

OV set if (X) = :8000.

Machine Code:

:0508

Registers Affected:

X Contents negated.
OV Set if (X) = –32,768 (:8000).

Timing: 1

2.8.3.16

CAR COMPLEMENT A REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0

Performs 1's complement of contents of A Register and places result in A.

$$(\overline{A}) \rightarrow A$$

Machine Code:

:0210

Registers Affected:

A Contents complemented.

Timing: 1

2.8.3.17

CXR COMPLEMENT X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0

Performs 1's complement of contents of X Register and places result in X.

$$(\overline{X}) \rightarrow X$$

Machine Codes:

:0408

Registers Affected:

X Contents complemented.

Timing: 1

2.8.3.18

TAX TRANSFER A TO X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0

Transfers contents of A Register to the X Register. A is unchanged.

$$(A) \rightarrow X$$

Previous contents of X are lost.

Machine Codes:

:0048

Registers Affected:

X Previous contents replaced by (A).

Timing: 1

2.8.3.19

TXA TRANSFER X TO A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0

Transfers contents of X Register to A Register. X is unchanged.

$$(X) \rightarrow A$$

Previous contents of A are lost.

Machine Codes:

:0030

Registers Affected:

A Previous contents replaced by (X).

Timing: 1

2.8.3.20

NAX NEGATE A AND PUT IN X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0

Places the 2's complement of contents of A into X. A is unchanged.

$$-(A) \rightarrow X$$

OV set if (A) = :8000. Previous contents of X are lost.

Machine Codes:

:0308

Registers Affected:

X Previous contents replaced by −(A).
OV Set if (A) = −32,768 (:8000)

Timing: 1

2.8.3.21

NXA NEGATE X AND PUT IN A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0

Places the 2's complement of contents of X into A. X is unchanged.

$$-(X) \rightarrow A$$

OV is set if (X) = :8000. Previous contents of A are lost.

Machine Code:

:0510

Registers Affected:

A Previous contents replaced by −(X).
OV Set if (X) = $-32{,}768_{10}$(:8000)

Timing: 1

2.8.3.22

CAX COMPLEMENT A AND PUT IN X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0

Places the 1's complement of contents of A Register into X. A is unchanged.

$$(\overline{A}) \rightarrow X$$

Previous contents of X are lost.

Machine Codes:

:0208

Registers Affected:

X Previous contents replaced by (A).

Timing: 1

2.8.3.23

CXA COMPLEMENT X AND PUT IN A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0

Places the 1's complement of contents of X Register into A. X is unchanged.

$(\overline{X}) \rightarrow A$

Previous contents of A are lost.

Machine Code:

:0410

Registers Affected:

A Previous contents replaced by $(\overline{X})$.

Timing: 1

2.8.3.24

IAX INCREMENT A AND PUT IN X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0

Adds one to contents of A Register and puts results in X. A is unchanged.

$(A) +1 \rightarrow X$

OV set if (A) = :7FFF. Previous contents of X are lost.

Machine Codes:

:0148

Registers Affected:

X Previous contents replaced by (A)+1.

OV Set if (A) = $32{,}767_{10}$ (:7FFF)

2.8.3.25

IXA INCREMENT X AND PUT IN A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0

Adds one to contents of X register and puts results in A. X is unchanged.

$(X) +1 \rightarrow A$

OV is set if (X) = :7FFF. Previous contents of A are lost.

Machine Codes:

:0130

Registers Affected:

A Previous contents replaced by (X) +1.

OV Set if (X) = $32{,}767_{10}$ (:7FFF)

Timing: 1

2.8.3.26

DAX DECREMENT A AND PUT IN X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0

Subtracts one from contents of A Register and places results in X. A is unchanged.

$(A) -1 \rightarrow X$

OV set if (A) = :8000. Previous contents of X are lost.

Machine Codes:

:00C8

Registers Affected:

X Previous contents replaced by (A) −1.

OV Set if (A) = −32,768 (:8000)

Timing: 1

2.8.3.27

DXA DECREMENT X AND PUT IN A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0

Subtracts one from contents of X Register and places results in A. X is unchanged.

$$(X) - 1 \rightarrow A$$

OV set if (X) = :8000. Previous contents of A are lost.

Machine Codes:

:00B0

Registers Affected:

A Previous contents replaced by (X) −1.
OV Set if (X) = $-32{,}768_{10}$ (:8000).

Timing: 1

2.8.3.28

NRX NOR OF A AND X TO X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0

Performs NOR $\overline{(A) \vee (X)}$ of contents of A and X Registers and places results in X. A is unchanged.

$$\overline{(A) \vee (X)} \rightarrow X$$

Previous contents of X are lost.

Machine Codes:

:0608

Registers Affected:

X Previous contents replaced by $\overline{(A) \vee (X)}$

Timing: 1

2.8.3.29

NRA NOR OF A AND X TO A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0

Performs NOR $\overline{(A) \vee (X)}$ of contents of A and X Registers and places results in A. X is unchanged.

$$\overline{(A) \vee (X)} \rightarrow A$$

Previous contents of A are lost.

Machine Codes:

:0610

Registers Affected:

A Previous contents replaced by $\overline{(A) \vee (X)}$.

Timing: 1

2.8.3.30

ANX AND OF A AND X TO X

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0

AND's contents of A and X Registers and places result in X. A is unchanged.

$$(A) \wedge (X) \rightarrow X$$

Previous contents of X are lost.

Machine Codes:

:0068

Registers Affected:

X Previous contents replaced by $(A) \wedge (X)$.

Timing: 1

2.8.3.31

ANA AND OF A AND X TO A

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0

AND's contents of A and X Registers and places result in A. X is unchanged.

$$(S) \wedge (X) \rightarrow A$$

Previous contents of A are lost.

Machine Codes:

:0070

Registers Affected:

A Previous contents replaced by (A) ∧ (X).

Timing: 1

2.8.3.32

ISX INPUT DATA SWITCHES TO X

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	1

Read 4 low order data switches (bits 0-3) into 4 low order bit positions of X Register (bits 0-3). Bits 4-15 of X are set to zeros. Previous contents of X are lost.

NOTE: This is a special case of the I/O INA instruction. See Part 3.1.4 of Section 3

Machine Code:

:5B01

Registers Affected:

X Previous contents replaced by contents of data switches 0-3.

Timing: 1-1/4

2.8.3.33

ISA INPUT SWITCHES TO A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1

Read 4 low order data switches (bits 0-3) into 4 low order bit positions of A Register (bits 0-3). Bits 4-15 of A are set to zeros. Previous contents of A are lost.

NOTE: This is a special case of the I/O INA instruction. See Part 3.1.4 of Section 3.

Machine Code:

:5801

Registers Affected:

A Previous contents replaced by contents of data switches 0-3.

Timing: 1-1/4

2.9 CONTROL INSTRUCTIONS

2.9.1 General

Control instructions are those instructions which are used for general status manipulation in the computer. Interrupts are enabled and disabled by Control instructions. The computer status word is saved and restored using Control instructions. Miscellaneous instructions such as Halt, No Operation, and OV status change are part of the Control class.

2.9.2 Format

There is no fixed format for the Control class. The formats used by this class technically fall into the Register Change class, Shift class, and I/O class. Those instructions which fall in the I/O class are pointed out in the instruction descriptions, because these instructions place data and control signals on the I/O busses. However, all of these instructions are discussed as Control instructions because their functions are control functions rather than I/O, Register Change, or Shift.

2.9.3 Instruction Descriptions

The Control class instruction descriptions follow the same general format as the Memory Reference instruction descriptions. The primary difference is in the Registers Affected portion of the description. This has been expanded to read Registers and Status Affected in most cases. Control instructions are concerned with much more than general registers. They have influence throughout the computer, therefore their total range must be described.

2.9.3.1

HLT HALT

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Halts the computer. Resets the Run mode indicator.

Machine Code:

:0800

Registers and Status Affected:

Run Resets Run mode

Timing: 1

2.9.3.2

NOP NO OPERATION

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This instruction causes a 1-cycle pause in the program.

Machine Code:

:0000

Registers and Status Affected:

None

Timing: 1

2.9.3.3

ROV RESET OVERFLOW

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0

Resets the Overflow indicator

$0 \rightarrow OV$

Machine Code:

:1200

Registers Affected:

OV Unconditionally reset.

Timing: 1

2.9.3.4

SOV SET OVERFLOW

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0

Sets the Overflow indicator.

$1 \rightarrow OV$

Machine Code:

:1400

Registers Affected:

OV Unconditionally set.

Timing: 1

2.9.3.5

COV COMPLEMENT OVERFLOW

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0

Complements the Overflow indicator.

$(\overline{OV}) \rightarrow OV$

Machine Code:

:1600

Registers Affected:

OV Complemented.

Timing: 1

2.9.3.6

SBM SET BYTE MODE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0

The Set Byte Mode (SBM) instruction conditions the computer to address byte operands rather than word operands when executing Memory Reference instructions. (See Memory Reference instruction descriptions for those instructions affected by Byte Mode.)

Machine Code:

:0E00

Registers and Status Affected:

Byte Mode Conditions the computer for Byte Mode addressing.

Timing: 1

2.9.4.7

SWM SET WORD MODE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

The Set Word Mode (SWM) instruction conditions the computer to address word operands rather than byte operands when executing Memory Reference instructions.

Machine Code:

:0F00

Registers and Status Affected:

Word Mode Conditions the computer for Word Mode addressing.

Timing: 1

2.9.3.8

EIN ENABLE INTERRUPTS

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0

Sets the Enable Interrupt flip-flop in the processor. Enables the recognition of external interrupts by the computer. (See Part 3.2 of Section 3.)

Machine Code:

:0A00

Registers and Status Affected:

Interrupts — Enables recognition of external interrupts.

2.9.3.9

DIN DISABLE INTERRUPTS

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

Resets the Enable Interrupt flip-flop in the processor. Prevents processor from responding to any interrupts (except Power Fail and Console. See PFE and CIE instructions).

Machine Code:

:0C00

Registers and Status Affected:

Interrupt — Prevents recognition of all interrupt which are under EIN/DIN control.

Timing: 1

2.9.3.10

SIN STATUS INHIBIT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	0	0	0	0	0	0	ID		

This instruction suspends Enable Interrupts status and Byte Mode status for the number of computer instructions specified by the ID Field of the instruction. When this instruction is executed, interrupts are inhibited and the computer is placed in Word Mode until the computer has executed a specified number of computer instructions. The number of instructions is one less than the number specified by the ID (Inhibit Duration) field of the Status Inhibit instruction. When the computer executes the specified number of instructions, Interrupt status and Byte Mode status are returned to the status they were in prior to the execution of the Status Inhibit instruction. This instruction is especially useful when writing subroutines which are entered from random locations in a main program. Computer status may be inhibited for up to 6 instructions. A count of 0 in the ID field does not inhibit computer status.

NOTE: This instruction is a special case of the I/O OTZ instruction. See Part 3.1.4 of Section 3.

Machine Code:

:680Z for 1 instruction inhibit duration.
thru
:6807 for 6 instruction inhibit duration.

Registers and Status Affected:

Byte Mode — Unconditionally set to Word Mode for one less than the number of instructions specified by ID Field, then returned to previous status.

Interrupts — Unconditionally inhibited for one less than the number of instructions specified by ID Field, then returned to previous status.

Timing: 1-1/4

2.9.3.11

TRP TRAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Causes the computer to be interrupted to the console interrupt location. Several conditions govern the execution of this instruction:

1. If Status Inhibit (SIN) is not in effect, the Trap will be recognized immediately. The Trap will be processed the same as any other interrupt.

2. If a SIN instruction has been executed and has not expired (the number of instructions specified by the SIN instruction have not been executed), the recognition of the Trap will be delayed until the required number of instructions have been executed.

3. Power Fail interrupts have priority over Trap interrupts. If a Power Fail interrupt is generated before a Trap interrupt is recognized, the Power Fail interrupt will be processed first.

Interrupt Location:	Normal	:001E
	Displaced	:011E

NOTE: This instruction is a special case of the I/O SEL instruction. See Part 3.1.4 of Section 3.

Machine Code:

:4007

Registers and Status Affected:

Interrupt	Generates an interrupt to location :001E (or :011E if displaced).

Timing: 1-1/4

2.9.3.12

SIX STATUS INPUT TO X

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0

Reads the status of the OV indicator, Byte Mode indicator, and Enable Interrupts flip flop into bits 0, 1, and 2 of the X Register. Unconditionally resets the OV indicator and the Byte Mode indicator after status is read. Does not reset the Enable Interrupts flip flop. The format of the status in the X Register is:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													EIN	BYTE	OV

Bits 3-15 of X are set to zeros. A 1-bit in the X Register indicates that the corresponding status indicator was set when read. A 0-bit indicates that the indicator was reset. The previous contents of X are lost.

NOTE: This instruction is a special case of the I/O INA instruction. See Part 3.1.4 of Section 3.

Machine Code:

:5A00

Registers and Status Affected:

X	Previous contents replaced by computer status.
OV	Unconditionally reset after status read.
Byte Mode	Unconditionally reset after status read.

Timing: 1-1/4

2.9.3.13

SIA STATUS INPUT TO A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0

Reads the status of the OV indicator, Byte Mode indicator, and Enable Interrupts flip flop into bits 0, 1, and 2 of the A Register. Unconditionally resets OV indicator and Byte Mode indicator after status is read. Does not reset the Enable Interrupts flip flop. The format in the A Register is:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													EIN	BYTE	OV

Bits 3-15 of A are set to zeros. A 1-bit in the A Register indicates that the corresponding status indicator was set when read. A 0-bit indicates that the indicator was reset. Previous contents of A are lost.

NOTE: This instruction is a special case of the I/O INA instruction. See Part 3.1.4 of Section 3.

Machine Code:

:5800

Registers and Status Affected:

A	Previous contents replaced by computer status.
OV	Unconditionally reset after status is read.
Byte Mode	Unconditionally reset after status is read.

Timing: 1-1/4

2.9.3.14

SOX STATUS OUTPUT FROM X

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0

Sets the OV indicator to the status of bit 0 of the X Register, and sets the Byte Mode indicator to the status of bit 1 of the X Register. This instruction does not restore the status of the Enable Interrupts flip flop.

NOTE: This instruction is a special case of the I/O OTA instruction. See Part 3.1.4 of Section 3.

Machine Code:

:6E00

Registers and Status Affected:

OV	Set to condition of bit 0 of the X Register. (1=Set, 0=Reset)
Byte Mode	Set to condition of bit 1 of the X Register. (1=Set, 0=Reset.

Timing: 1-1/4

2.9.3.15

SOA STATUS OUTPUT FROM A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0

Sets the OV indicator to the status of bit 0 of the A Register, and sets the Byte Mode indicator to the status of bit 1 of the A Register. This instruction does not restore the status of the Enable Interrupts flip-flop.

NOTE: This instruction is a special case of the I/O OTA instruction. See Part 3.1.4 of Section 3.

Machine Code:

:6C00

Registers and Status Affected:

OV	Set to condition of bit 0 of the A Register. (1=Set, 0=Reset)
Byte Mode	Set to condition of bit 1 of the A Register. (1=Set, 0=Reset)

Timing: 1-1/4

2.9.3.16

CIE CONSOLE INTERRUPT ENABLE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1

This instruction enables console interrupts. Console interrupts are generated when the AUTO LD siwtch is depressed and the computer is in the Run mode. Console interrupts are also under the control of the Enable Interrupts (EIN) or

Power Fail Enable (PFE) instructions, depending on the computer configuration selected. If Power Fail interrupt enable is placed under the control of the EIN instruction, then console interrupts are also under EIN control (both EIN and CIE instructions must be executed for console interrupts to be recognized), but if Power Fail interrupt enable is placed outside EIN control then console interrupts are also outside EIN control and under PFE control (both PFE and CIE instructions must be executed for console interrupts to be recognized).

NOTE: This instruction is a special case of the I/O SEL instruction. See Part 3.1.4 of Section 3.

Machine Code:

:4005

Registers and Status Affected:

Console Interrupts — Sets the Console Interrupt Enable flip flop.

Timing: 1-1/4

2.9.3.17

CID — CONSOLE INTERRUPT DISABLE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0

This instruction unconditionally disables console interrupts, regardless of the Enable Interrupts flip flop or Power Fail Enable flip flop (see Console Interrupt Enable instruction).

NOTE: This instruction is a special case of the I/O SEL instruction. See Part 3.1.4 of Section 3.

Machine Code:

:4006

Registers and Status Affected:

Console Interrupts — Unconditionally disabled.

Timing: 1-1/4

2.9.3.18

PFE — POWER FAIL INTERRUPT ENABLE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0

A special computer option allows Power Fail Interrupt Enable and Power Fail Interrupt Disable to be placed outside EIN and DIN control. When this option is exercised, the Power Fail Enable (PFE) instruction is effective. This instruction enables Power Fail interrupts. (See Part 4.3 of Section 4 for a description of the Power Fail option.) When power fail interrupts are enabled, low power conditions will be recognized by the computer and will generate a Power Fail Interrupt to location :001C (location :011C if displaced).

NOTE: This instruction is a special case of the I/O SEL instruction. See Part 3.1.4 of Section 3.

Machine Code:

:4002

Registers and Status Affected:

Power Fail — Enables power fail interrupts.

Timing: 1-1/4

2.9.3.19

PFD — POWER FAIL INTERRUPT DISABLE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1

A special computer option allows Power Fail Interrupt Enable and Power Fail Interrupt Disable to be placed outside EIN and DIN control. When this option is exercised, the Power Fail Interrupt Disable (PFD) instruction is effective. This instruction disables Power Fail interrupts. (See Part 4.3 of Section 4 for a description of the Power Fail option.)

CAUTION

WHEN THIS INSTRUCTION IS EXECUTED, LOW POWER CONDITIONS CANNOT BE RECOGNIZED BY THE COMPUTER UNTIL POWER FAIL INTERRUPTS ARE AGAIN ENABLED.

NOTE: This instruction is a special case of the I/O SEL instruction. See Part 3.1.4 of Section 3.

Machine Code:

:4003

Registers and Status Affected:

Power Fail — Inhibits recognition of power fail interrupts.

Timing: 1-1/4

2.9.3.20

MPE — MEMORY PROTECT ENABLE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

When the Memory Protect option is installed in the computer, a special option allows protection to be enabled or disabled by computer software. The Memory Protect Enable (MPE) instruction enables the memory protect feature and prevents the modification (writing) of data in the protected area of memory. When this instruction is executed the computer may read instructions and data from the protected area, but may not write into the protected area. (See Part 4.6 of Section 4 for a description of the Memory Protect option.)

NOTE: This instruction is a special case of the I/O SEL instruction. See Part 3.1.4 of Section 3.

Machine Code:

:4000

Registers and Status Affected:

Memory Protect — Enables memory protect feature.

Timing: 1-1/4

2.9.3.21

MPD — MEMORY PROTECT DISABLE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

When the Memory Protect option is installed in the computer, a special option allows protection to be enabled or disabled by computer software. The Memory Protect Disable (MPD) instruction disables the memory protect feature and allows the modification (writing) of data in the protected area. (See Part 4.6 of Section 4 for a description of the Memory Protect option.)

CAUTION

WHEN THIS INSTRUCTION IS EXECUTED THE CONTENTS OF THE PROTECTED AREA OF MEMORY MAY BE MODIFIED BY COMPUTER SOFTWARE. THE AREA REMAINS UNPROTECTED UNTIL THE MPE INSTRUCTION IS EXECUTED.

NOTE: This instruction is a special case of the I/O SEL instruction. See Part 3.1.4 of Section 3.

Machine Code:

:4001

Registers and Status Affected:

Memory Protect — Disables memory protect feature.

Timing: 1-1/4

2.9.3.22

RAM SET RANDOM ACCESS MODE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	1

A special option allows the installation of up to 4K words of Read Only Memory in the computer. The Read Only Memory may be paralleled in the lower 256 words by Random Access Memory. When this option is installed, special control instructions are provided to select the random access memory or the read only memory. The Set Random Access Mode (RAM) instruction conditions the computer to address the random access memory rather than the read only memory when addressing the lower 256 words of the Read Only Memory option.

NOTE: This instruction is a special case of the I/O SEL instruction. See Part 3.1.4 of Section 3.

Machine Code:

:4045

Registers and Status Affected:

Read Only Memory	Computer addresses Random Access Memory rather than Read Only Memory when addressing Read Only Memory option.

Timing: 1-1/4

2.9.3.23

ROM SET READ ONLY MODE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	1	0	0	0	1	1	0

A special option allows the installation of up to 4K words of Read Only Memory in the computer. The Read Only Memory may be paralleled in the lower 256 words by Random Access Memory. When this option is installed, special control instructions are provided to select the random access memory or the read only memory. The Set Read Only Mode (ROM) instruction conditions the computer to address the read only memory rather than the random access memory when addressing the lower 256 words of the Read Only Memory option.

NOTE: This instruction is a special case of the I/O SEL instruction. See Part 3.1.4 of Section 3.

Machine Code:

:4046

Registers and Status Affected:

Read Only Memory	Computer addresses Read Only Memory rather than Random Access Memory when addressing Read Only Memory option.

Timing: 1-1/4

SECTION 3
INPUT/OUTPUT

3.1 INTRODUCTION

3.1.1 General

A computer, if it is to perform any useful function, must have the ability to receive information from external sources and to send information to external devices. A characteristic of the ALPHA 16 and NAKED MINI 16 computers is a very powerful input/output configuration which allows very efficient communication between the computer and external devices.

Part 1.3.6 of Section 1 is a general discussion of the I/O configuration used by these computers. The purpose of Section 3 is to describe in detail the I/O functions of the ALPHA 16 and NAKED MINI 16 computers. The introduction to this section describes control requirements, I/O organization, and general data movement. The second part of this section describes the priority interrupt system. The third part describes general I/O instructions. The fourth part describes block transfer instructions. The final part describes direct memory channels (automatic I/O instructions).

3.1.2 Control Requirements

Part 1.3.6 of Section 1 points out that there are four general functions which a computer must perform in order to effectively control peripheral devices. These functions are required to provide effective data transfer in both directions, and to properly monitor peripheral activities.

3.1.2.1 Device Selection. A computer is capable of controlling more than one peripheral device. For this reason, there must be some means for selecting or addressing each device with which the computer must communicate. Each peripheral device is assigned some unique device address which the computer may use to select the device for control or data transfer.

3.1.2.2 Function Command. Most peripheral devices are capable of performing several different functions. For example, a magnetic tape unit is capable of writing data, reading data, rewinding, backspacing, etc. The computer must be able to tell the device exactly which function it is to perform. Therefore, the computer must not only be able to select a device, but it must also be able to command the selected device to perform certain functions.

3.1.2.3 Sense Status. Because of the vastly different operating speed of computers and peripheral devices, there may be a relatively long period of time between the issuing of a command and the completion of that command. Also, because peripheral devices often must execute some mechanical action to perform the commands that are given, the chances for errors to occur in peripheral devices are much greater than in the computer. Peripheral devices often have elaborate error checking schemes. The computer must be able to interrogate the peripheral device to see whether or not commands have been completed, and whether or not error conditions have been detected. In order to do this, the computer has the ability to address a device interface and sense certain status conditions in the device or interface.

3.1.2.4 Data Transfer. There are many types of peripheral devices. Many of them are extensions of the computer memory (disks, magnetic tape units, etc.). Others convert information from computer codes to a form that can be read by humans (printers, CRTs, plotters, etc.). Others allow the computer to monitor and control physical events and actions (analog to digital and digital to analog converters). The types of peripheral devices are virtually unlimited. But a common characteristic of all is that they send data to and/or receive data from the computer. The ultimate objective of all peripheral devices is the transfer of data in one form or another.

3.1.3 Organization

Control of peripheral devices and data transmission between the devices and the computer are accomplished by the use of a peripheral interface and computer instructions to condition and interrogate the interface. Figure 1-5 illustrates the relationship between the computer, the interface and the peripheral device. Figure 3-1 is a more detailed block diagram of this relationship. The following paragraphs discuss the four general functions which the computer must perform, using Figure 3-1 to illustrate how these functions are performed.

3.1.3.1 Device Selection. Device selection starts with the decoding of an I/O instruction in the I Register of the computer. Each I/O instruction contains an operation code specifying the type of instruction that is being executed, the address of the device which is to respond to the instruction, and a function code specifying the action that the device is to take.

The device address is gated directly from the I Register of the computer to the P Bus. The P Bus is applied directly to an Address Decode section in all peripheral interfaces which are connected to the computer. Only one interface will be able to decode the address and respond to it. Those interfaces which are not selected will not be able to decode the address. The interface which can decode the address generates a signal which enables the Function Decode and Control Logic of the interface. Those interfaces which cannot decode the address have their Function Decode and Control Logic sections disabled by the lack of an address code.

3.1.3.2 Function Command. The function that is to be performed by the peripheral device is determined by two things: (1) the type of I/O instruction that is being executed, and (2) the instruction function code. The operation code in the I Register determines the type of instruction that is being executed. Processor Control decodes the operation code and sends signals via the C Bus to the interface. These signals tell the interface whether the instruction is a Select, Sense, Input, or Output instruction. These signals are applied to the interface Control Logic.

The I/O instruction function code is passed from the I Register to the interface Function Decode section via the F Bus. The function code is decoded in Function Decode and is applied to the interface Control Logic. The Control Logic examines the function code from Function Decode and the instruction type from Processor Control to determine what control signals must be generated in the interface and what control signals must be applied to the peripheral device via the Device Control lines. Note that the Control Logic section in the interface cannot perform any of these functions unless it is enabled by the Address Decode section.

3.1.3.3. Sense Status. The peripheral interface receives status information from the peripheral device via the Device Control lines. The computer may interrogate the interface to determine the status of the peripheral device. The computer uses a Sense instruction to perform the interrogation.

When a Sense instruction is decoded in the I Register of the computer, the Process Control section sends control information to the peripheral interface. The device address portion of the Sense instruction specifies which device is being sensed, and the function code specifies what status in the device is being sensed. The Control Logic of the interface examines the enable signal from Address Decode, the fact that a Sense instruction is being executed from Processor Control, and the function code from Function Decode. The Control Logic then sends the status of the sensed function to Processor Control via the C Bus. Processor Control examines the status of the sensed function and takes appropriate action in the computer.

3.1.3.4 Data Transfer. Data transfer between the computer and the peripheral device is accomplished by executing appropriate control instructions in the computer and having the peripheral interface control the data transfer operations

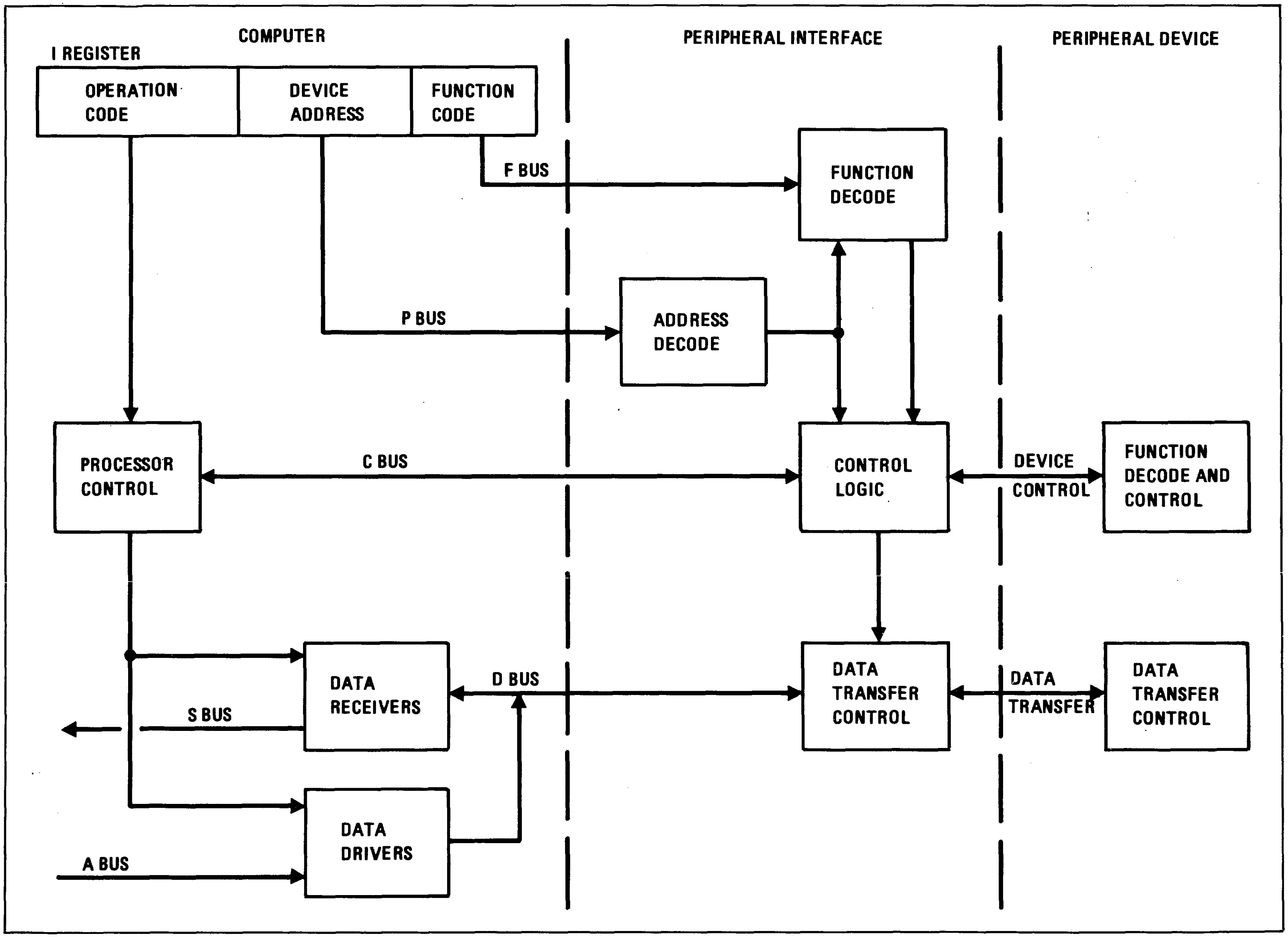


Figure 3-1. Computer/Interface/Device Relationships

of the peripheral device. The peripheral interface often acts as an intermediate storage location for data being transferred between the computer and the peripheral device.

For example, many peripheral devices transmit and receive data in a serial mode; i.e., one bit at a time rather than all bits making up a word or byte in parallel. Since the computer moves data in parallel, the interface must act as a data assembler when reading data from the peripheral device, and as a disassembler when writing data to the peripheral device. Also, because the computer moves data very rapidly and peripheral devices often move data relatively slowly, the interface is used as an intermediate buffer for the speed transition.

Data transfers may often be accomplished in several steps. The computer may execute some function command type instructions (Select instructions) to set up the conditions in the peripheral device to allow the movement of data. In the case of a magnetic tape unit, for example, the tape unit must be commanded to start tape motion before the transfer of data can be started. The tape must then be allowed to accelerate to its operating speed before data transfer can begin. The computer may then be required to sense the status of the device to see if it is ready to begin data transfer. When the device is ready, the computer may execute Input or Output instructions to cause the transfer of data. The operation code of the I/O instruction tells whether the transfer is an input or an output, and the device address and function code of the instruction specify the device that is to take part.

Data is transferred between the Data Transfer Control section of the peripheral device and the Data Transfer Control section of the interface. The computer either reads the data from the Data Transfer Control section of the interface, or sends data to the Data Transfer Control section of the interface. When data is read from the interface, it is placed on the S Bus of the computer. When data is sent to the interface, it is sent from the A Bus of the computer.

3.1.3.5 Party Line I/O Structure. Figure 3-1 illustrates the relationship between the computer and one peripheral interface. Figure 3-2 illustrates the party line I/O structure used by the ALPHA 16 and NAKED MINI 16 computers. The I/O busses apply their signals to all peripheral interfaces in parallel. However, only one interface will be able to decode the device address that is placed on the P Bus. The other interfaces will not be able to decode the address, and thus will be unable to respond to the other signals on the other busses.

3.1.4 Reserved Device Addresses

Because of the flexibility of the I/O structure of the ALPHA 16 and NAKED MINI 16 computers, I/O instructions can be used to control functions and operations other than peripheral devices. Two device addresses are reserved for internal computer use for control of processor options and special functions. The reserved device addresses are Device Address 0 and Device Address 8.

3.1.4.1 Device Address 0. Device Address 0 is used for control of processor options and implementation of certain internal processor instructions. Functions controlled by Device Address 0 are:

1. Option Control. The Memory Protect and Power Fail/Restart options are controlled by use of special I/O instructions using Device Address 0. Refer to the MPE, MPD, PFE, and PFD Control instructions in Section 2.

2. Option Sense. Special Sense instructions, using Device Address 0 along with function codes, are used by diagnostic programs and executive programs to sense the presence of processor options. These instructions can determine which options are installed in the computer which is executing the program. The options which can be sensed using these instructions are the Autoload, Power Fail/Restart, Real Time Clock, Memory Protect,

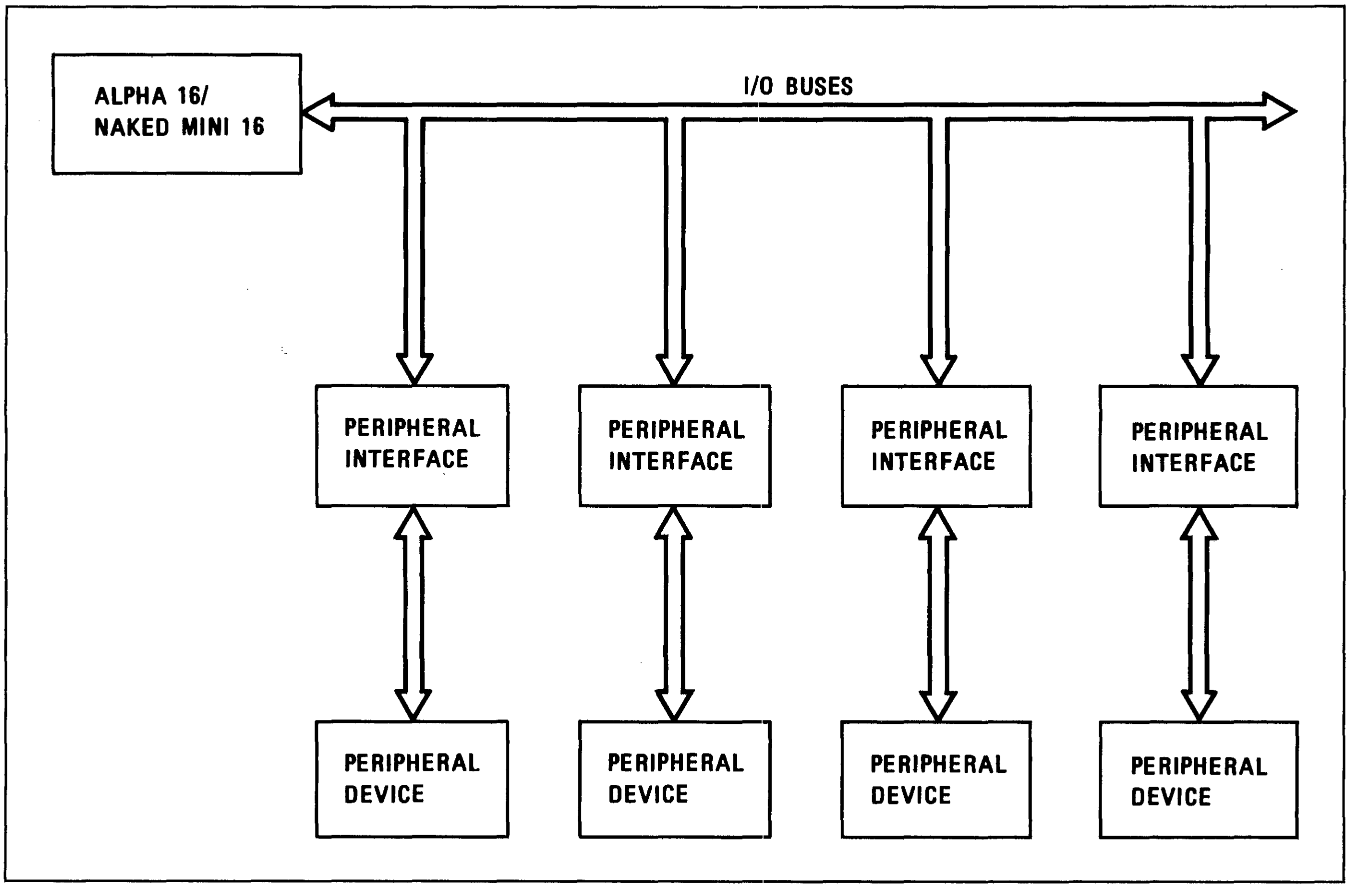


Figure 3-2. Party Line I/O Structure

and TTY options. (Refer to the descriptions of the individual options for the specific sense instructions used.)

3. Processor Instructions. Several processor instructions are special cases of I/O instructions using Device Address 0. These instructions are: CIE, CID, TRP, SOA, SOX, SIN, SIA, SIX, ISA, and ISX. Refer to the instruction descriptions in Section 2 for details concerning these instructions.

3.1.4.2 Device Address 8. Device Address 8 is used to control the Real Time Clock option and to implement two control instructions. The two control instructions are RAM and ROM.

3.2 PRIORITY INTERRUPT SYSTEM

3.2.1 General

Interrupts allow a computer to respond to external stimuli. A mini computer may be used in a wide variety of applications where it must communicate with many different types of devices. The devices with which the computer must communicate often operate at widely varying speeds. Often the events to which the computer must respond occur randomly rather than at evenly spaced time intervals. And if the events do occur at evenly spaced time intervals, these intervals may be relatively far apart.

If a computer does not have a priority interrupt system, the computer must poll all of the external devices which may require service. The polling must be at frequent enough intervals so that events are serviced within a reasonable time after they occur. Polling consumes considerable time, and may not allow much processing time between the handling of external events.

A priority interrupt system relieves the computer of the polling responsibility. The computer may continue processing data between external events, and may take time out from main program processing to handle external events as they occur.

3.2.2 Basic Concepts

An external interrupt causes the computer to execute one instruction outside of the main program. If that one instruction does not modify the P Register (Program Counter), the computer continues with the main program after executing the interrupt instruction. If the interrupt instruction modifies the Program Counter, the computer continues processing at the location specified by the new value in the Program Counter.

3.2.2.1 Interrupt Location. If an external device is to operate under interrupt control, reserved locations in memory are assigned to the device. Interrupt lines are configured to cause the computer to execute the instruction at the reserved location when the external device generates an interrupt to the computer. Each device may be assigned one or more reserved locations. For example, if a device must move a block of data from the device to the computer, it may generate one interrupt for each word to be moved and another interrupt when the whole block has been moved. The interrupt for each word would require one location, and the interrupt indicating the end of the block of words would require another location.

Interrupt locations are fixed by hardware. The design of the computer and the design of peripheral interfaces determine the reserved interrupt locations associated with each peripheral device.

3.2.2.2 Interrupt Lines. Interrupts are transmitted from a peripheral device to the computer via interrupt lines. The ALPHA 16 and NAKED MINI 16 interrupt lines are configured in such a manner that large numbers of devices can be handled under interrupt control. There are three standard interrupt lines. Two of these lines are assigned fixed interrupt locations, and the third can be used to vector interrupts from a virtually unlimited number of peripheral devices. The standard interrupt lines are:

1. IL1 — Interrupt Line 1. This is a standard interrupt line which is assigned memory location :0002 as its interrupt location.

2. IL2 — Interrupt Line 2. This is a standard interrupt line which is assigned memory location :0006 as its interrupt location.

3. IUR — Interrupt Request Line. This is a standard interrupt line which transmits interrupts from more than one peripheral device to the computer. When a peripheral device requests an interrupt via IUR, the peripheral interface also provides the computer with the address of the interrupt location. Therefore, the reserved interrupt location is a function of the peripheral interface design.

In addition to the three standard interrupt lines, certain options have special interrupt lines. Because of the special nature of the options, special lines are generated so that the option need not compete with other peripheral devices for interrupt recognition.

3.2.3 Interrupt Processing

An interrupt is a signal from some peripheral device requesting computer action. The interrupt generally means that some external event has occurred which requires computer recognition or some positive action on the part of the computer.

> Example 1. Conside a computer which uses a Teletypewriter as a peripheral device. Messages are printed on the Teletype printer one character at a time. Since the transfer rate from the computer to the printer is very slow, the computer can continue processing data between characters. The Teletype interface can be programmed to interrupt the computer after each character has been printed. The computer responds to the interrupt by sending another character to the interface for printing.

> Example 2. Consider a computer which is being used in a highway traffic monitoring system. One purpose of the system is to count the traffic in each lane of the highway and store that information for further processing. Sensors are placed in each lane of the highway, and each sensor generates an interrupt to the computer each time an automobile crosses the sensor. The computer response to each interrupt is to increment a counter.

3.2.3.1 Interrupt Recognition. Before any interrupt can be recognized by the computer, several conditions must be met:

1. Interrupts Must Be Enabled. The programmer has absolute control over the recognition of interrupts. If interrupts are to be recognized, the Enable Interrupts (EIN) instruction must be executed. This instruction enables interrupts until some condition occurs to disable interrupts. Paragraph 3.2.4 discusses the conditions which will disable interrupts.
2. The Interrupt Mask Must Be Set. The EIN instruction enables interrupts in general. Specific interrupts are enabled by setting an interrupt mask in the peripheral interface. Masks are generally set by executing a Select (SEL) instruction with a device address and function code specifying which interrupt is to be enabled. By using interrupt masks, the programmer can selectively enable and disable interrupts.
3. The Interrupt Condition Must Exist. The EIN instruction and setting the interrupt mask allow an interrupt to occur. For the interrupt to actually occur, the event which has been enabled must occur. In the case of Example 1, the Teletype interface must complete the transmission of a character to the Teletype for an interrupt to be generated. In the case of Example 2, an automobile must cross a sensor.
4. No Higher Priority Interrupt Must Be Waiting. Each peripheral interface or computer option has a definite priority assignment. Each interrupt must wait its turn. Interrupts are processed by the computer in the order received, or according to priority if more than one interrupt is pending. (Priorities are discussed in Part 3.2.5.)
5. In Run Mode. Interrupts cannot be recognized if the computer is halted or if the STOP switch is down.

Once these conditions have been met, the computer can recognize and process the interrupt. The computer completes the instruction that it is currently executing and then recognizes the highest priority interrupt that is waiting.

3.2.3.2 Interrupt Instructions. When an interrupt is recognized, the computer executes one instruction at the interrupt location. If that instruction does not modify the Program Counter the computer then continues with its main program. If the interrupt instruction modifies the Program Counter, the computer resumes processing at the location specified by the new value in the Program Counter. Almost any computer instruction can be used as an interrupt instruction, but some lend themselves more readily to this function than others. The instructions which are most commonly used as interrupt instructions are:

1. IMS — The Increment Memory and Skip on Zero instruction is normally used when the computer is counting external events. When IMS is used for this purpose, it does not cause a skip when the memory location being incremented goes to zero. Instead, it generates a signal (called an Echo) to the peripheral interface which generated the interrupt. (See Part 4.4, Real Time Clock programming example, for an example of IMS used as an interrupt instruction.)

2. JST — When an interrupt cannot be processed by a single instruction, a subroutine must be entered. But there must be some way to get back to the main program after the interrupt has been processed. The JST instruction is the only unconditional jump instruction which fills this need. It stores the address of the next instruction to be executed in the main program. This provides a return to the main program. It then sets the Program Counter to the start of the interrupt processing subroutine.

   NOTE: When executed as an interrupt instruction, the JST instruction also disables interrupts. The programmer must re-enable interrupts before leaving the interrupt subroutine if he wants subsequent interrupts to be recognized.

3. Auto I/O:
   AIN
   AOT
   AIB
   AOB

   The Automatic Input/Output instructions are designed specifically as interrupt instructions. Each Auto I/O instruction is effectively a complete interrupt subroutine in one instruction. These instructions contain their own word or byte count and their own memory addresses. They can be used to transfer large blocks of data between the computer memory and peripheral devices. Since they do not affect the A Register, X Register, OV indicator, or the Program Counter when transferring data, they are ideal as interrupt processing instructions. (See Part 3.5 for a complete description of the Auto I/O instructions.)

3.2.3.3 Single Instruction Interrupt Processing. If an interrupt can be processed by a single instruction, such as an IMS or an Auto I/O instruction, the computer executes the interrupt instruction in response to the interrupt and continues with the main program. Figure 3-3 illustrates the sequence of events involved in the processing of an interrupt using a single interrupt instruction. The events shown in this figure are:

1. An interrupt will usually be received by the computer while the computer is busy executing an instruction. The computer must complete the execution of that instruction before it can recognize the interrupt. If all other necessary conditions have been met (see Paragraph 3.2.3.2), the computer will recognize the interrupt when it completes its current instruction. If the computer receives an interrupt while executing the instruction at P-1, it completes that instruction before recognizing the interrupt.

2. When the computer recognizes the interrupt, it executes the instruction at the interrupt location. The Program Counter is not incremented by virtue of the execution of the interrupt instruction. It is assumed that in this case the interrupt instruction does not modify the program counter.

3. Once the interrupt instruction has been executed, the computer resumes the execution of the main program by executing the next sequential instruction following the last one completed. The computer had finished the instruction at P-1, so it resumes with the instruction at P.

The end result of single instruction interrupt processing is that the computer executes one instruction outside of the main program, and then continues with the main program.

NOTE: When a Memory Reference instruction is used as an interrupt instruction, all memory addressing modes are valid. If relative addressing is used to fetch an operand or address pointer, however, the fetch will be relative to the interrupt location rather than relative to the Program Counter.

3.2.3.4 Subroutine Interrupt Processing. If an interrupt cannot be processed by a single instruction, a subroutine must be used to process the interrupt. Figure 3-4 illustrates the general sequence of events involved in the execution of interrupt subroutines:

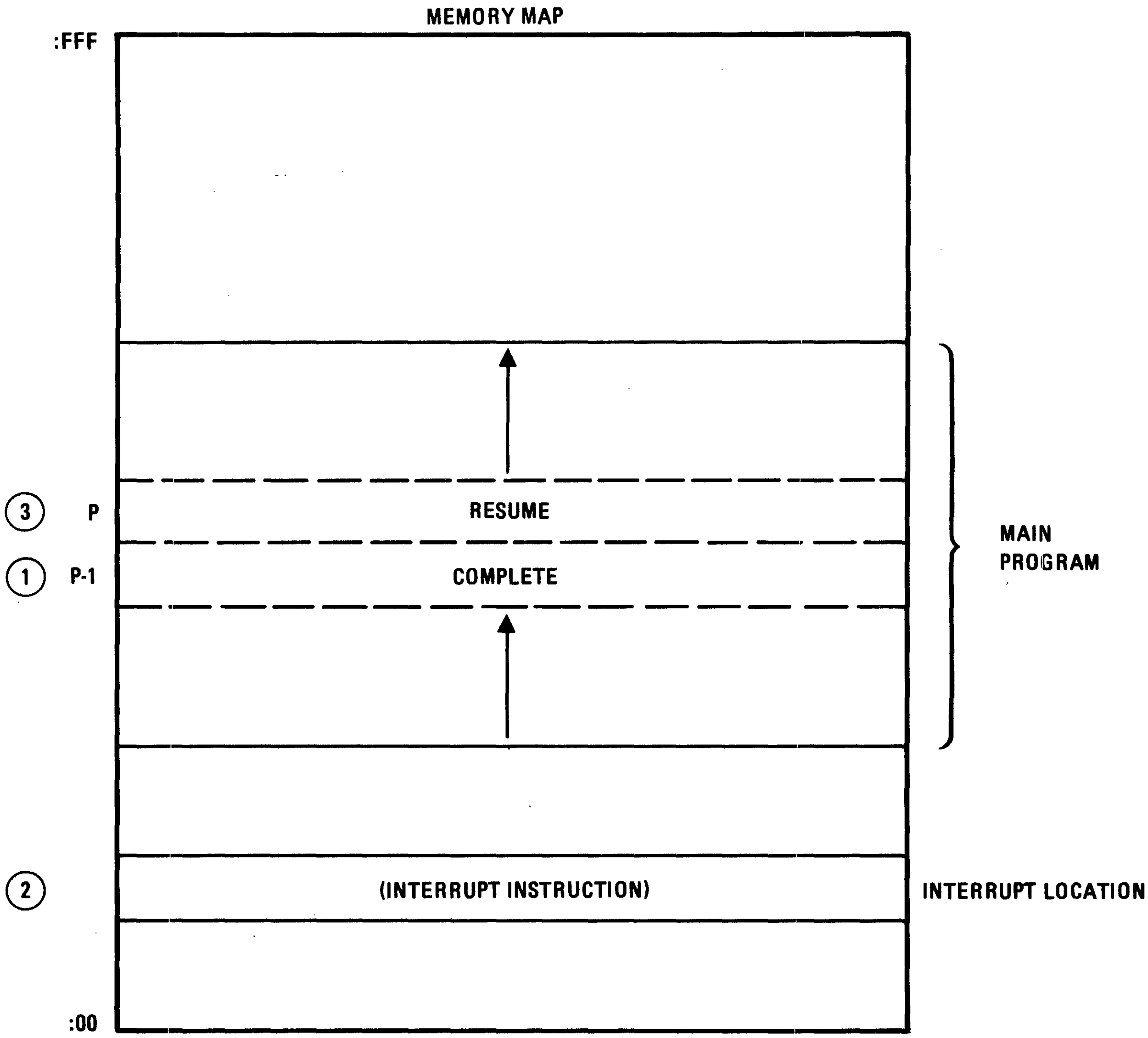
1. Assume that the computer is executing the instruction at P-1 when the interrupt is received. The computer first completes its current instruction and then recognizes the interrupt.

2. When the interrupt is recognized the computer executes the instruction at the interrupt location. In this case the instruction at the interrupt location is a Jump and Store. A Jump and Store is an unconditional jump instruction that modifies the Program Counter.

3. The Jump and Store instruction causes the value in the Program Counter to be stored at the jump address. Since the value in the Program Counter is the address of the next instruction in the main program, this provides return linkage for the subroutine.

4. The Jump and Store instruction causes the jump address plus 1 to be placed in the Program Counter (in this case, SUB+1 goes to P). The computer then begins the execution of the interrupt subroutine.

5. The computer continues the execution of the interrupt subroutine until it is completed. Completion is signaled by the execution of an unconditional jump back to the main program. In this case the instruction is JMP *SUB. This instruction causes an indirect jump using the value stored in SUB as an address pointer. Since the value in SUB is the address of the instruction at P, the computer will transfer back to the main program and continue execution beginning with the instruction at P.

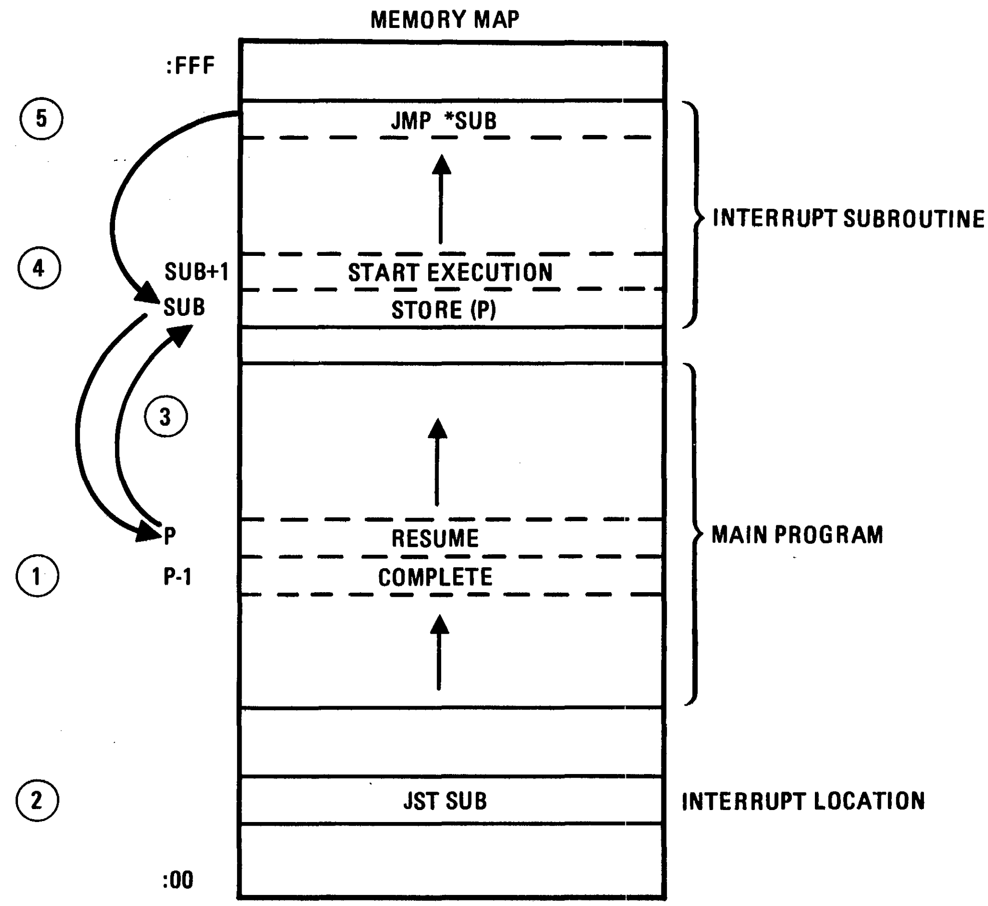


(1) AN INTERRUPT IS RECEIVED WHILE THE COMPUTER IS EXECUTING THE INSTRUCTION AT P-1. THE COMPUTER COMPLETES THAT INSTRUCTION BEFORE RECOGNIZING THE INTERRUPT.

(2) THE COMPUTER RECOGNIZES THE INTERRUPT AND EXECUTES THE INSTRUCTION AT THE INTERRUPT LOCATION. IN THIS CASE THE INTERRUPT INSTRUCTION DOES NOT MODIFY THE PROGRAM COUNTER.

(3) AFTER COMPLETING THE INTERRUPT INSTRUCTION THE COMPUTER RESUMES MAIN PROGRAM EXECUTION WITH THE INSTRUCTION AT P.

Figure 3-3. Single Instruction Interrupt Processing



(1) INTERRUPT IS RECEIVED WHILE THE INSTRUCTION AT P-1 IS BEING EXECUTED. THE COMPUTER COMPLETES THE INSTRUCTION AT P-1.

(2) THE COMPUTER RECOGNIZES THE INTERRUPT AND EXECUTES THE INSTRUCTION AT THE INTERRUPT LOCATION.

(3) THE JST INSTRUCTION CAUSES THE CONTENTS OF THE PROGRAM COUNT TO BE STORED AT SUB. THE VALUE STORED IN SUB IS THE ADDRESS OF THE INSTRUCTION AT P.

(4) THE JST INSTRUCTION CAUSES THE ADDRESS OF THE INSTRUCTION AT SUB + 1 TO BE PLACED IN THE PROGRAM COUNTER. THE COMPUTER BEGINS EXECUTING THE SUBROUTINE.

(5) WHEN THE SUBROUTINE IS COMPLETED, A JMP INDIRECT IS EXECUTED USING THE VALUE AT SUB AS AN INDIRECT ADDRESS POINTER. THIS CAUSES THE COMPUTER TO RESUME EXECUTION OF THE MAIN PROGRAM BEGINNING WITH THE INSTRUCTION AT P.

Figure 3-4. Interrupt Subroutine Processing

CAUTION

WHEN A JUMP AND STORE (JST) INSTRUCTION IS EXECUTED AS AN INTERRUPT INSTRUCTION IT AUTOMATICALLY DISABLES INTERRUPTS. INTERRUPTS MUST BE RE-ENABLED IN THE INTERRUPT SUBROUTINE IF SUBSEQUENT INTERRUPTS ARE TO BE RECOGNIZED.

3.2.4 Interrupt Latency

Interrupt latency may be defined as the conditions which may delay the recognition of an interrupt. The general rule is that the highest priority interrupt that is waiting will be recognized at the end of the instruction that the computer is currently executing. The time required to execute an instruction becomes a determining factor in interrupt recognition. Certain instructions can cause unusual interrupt delays in addition to the execution time of the instruction itself. The conditions which can delay the recognition of interrupts are discussed in the following paragraphs.

3.2.4.1 Instruction Completion. When an interrupt request is generated during the execution of an instruction, that instruction must be completed before the request is recognized and processed. The maximum delay which may be encountered can be computed by computing the maximum time required to execute an instruction. For example, Memory Reference instructions require a minimum of two cycles to complete execution (one cycle to get the instruction and one cycle to get the operand and perform the necessary logical operations). Memory reference instruction execution times are extended if indirect addressing is used. One additional cycle is required for each level of indirect addressing. Therefore, if a Memory Reference uses two levels of indirect addressing to fetch an operand, the total number of cycles required to execute the instruction is four. Since each cycle is 1.6 microseconds in length, total execution time would be (4)(1.6)=6.4 microseconds. If an interrupt request were generated at the beginning of such an instruction, the recognition of the request would be delayed for a maximum of 6.4 microseconds.

Most instructions are executed in fewer than four cycles. But there are some instructions which may require more cycles and may cause unusually long delays. These instructions are:

1. Scan. The time required to execute a Scan instruction is a function of the number of words being scanned. The minimum execution time is 1 cycle for the instruction, 1 cycle for the first indirect level (there is always at least one indirect address level), and 1 for the first word that is scanned. If it is assumed that there is only one indirect addressing level, then the timing is 2 cycles plus 1 cycle for each word scanned. If 100 words are scanned, the timing is 2+100=102 cycles. The time required to execute the Scan is (102)(1.6) = 163.2 microseconds. If 4000 words are being scanned, the timing is 4000+2=4002 cycles. The time required to execute the scan is (4002)(1.6)=6403.2 microseconds, or approximately 6.4 milliseconds. A delay of this sort may be insignificant for some peripheral devices, but it may be unbearable for others. Therefore, the Scan instruction should be used with extreme caution when interrupts may occur while the scan is in process.
2. Block I/O. Block I/O instructions are similar to the Scan instruction in that a large number of words may be handled before the instruction terminates. The timing is computed in a manner similar to that used for the Scan instruction. (See the Block I/O instruction descriptions for the timing formula.) These instructions must also be used with caution when interrupts may occur during the execution of the instruction.
3. Shift Instructions. The maximum time which may be required to complete a shift instruction is 1-1/4+8=9-1/4 cycles. This is for a Long shift of 16 places. The time required to complete that number of cycles is (9-1/4)(1.6) = 14.8 microseconds. If high speed peripheral devices are operating under interrupt control, a delay of 14.8 microseconds

in recognition of an interrupt may be excessive.

3.2.4.2 Interrupt Control Instructions. Several instructions are used to control the times during which interrupts may be recognized. There are special situations which must be considered when using these instructions:

1. Enable Interrupts (EIN). When the EIN instruction is executed, the computer guarantees that the next instruction following the EIN instruction will be executed before the first interrupt is recoginized. (The primary reason for this is in the use of EIN at the end of an interrupt subroutine. This allows EIN to be executed and a Jump back to the main program to be executed before another interrupt is recognized.) Therefore the earliest that an interrupt can be recognized following an EIN is the time required to execute the EIN plus the time required to execute the next instruction in sequence.
2. Power Fail Enable (PFE). When Power Fail interrupt control is outside EIN/DIN control, the PFE instruction has the same timing considerations as EIN.
3. Disable Interrupts (DIN). When the DIN instruction is executed, interrupts are disabled during the instruction execution, therefore no interrupts can be recognized following the DIN until an EIN instruction is executed.
4. Status Inhibit (SIN). The SIN instruction inhibits interrupts for the number of instructions specified by the SIN instruction. Note that the inhibit time is for a specified number of instructions and not for a specified number of cycles. The total time that interrupts will be inhibited when the SIN instruction is executed is the total time required to execute the instructions for which interrupts have been inhibited, plus the time required to execute the SIN instruction.

3.2.4.3 Interrupt Instructions. Special timing considerations are involved when an instruction is executed as an interrupt instruction. These special considerations are:

1. Interrupt Delay. When any interrupt instruction is executed, the computer guarantees that at least one instruction will be executed following the interrupt instruction before another interrupt will be recognized. For example, if a peripheral device sends a constant interrupt to the computer, and the computer services the interrupt with a single instruction, the computer will recognize the interrupt and execute the interrupt instruction. It will then execute one instruction in the main program before recognizing the interrupt again. If the interrupt never goes away, the computer will continue to execute the interrupt instruction, then one instruction from the main program, then the interrupt instruction, then the next instruction from the main program, etc.
2. Jump and Store. The Jump and Store instruction is a special case. When this instruction is executed as an interrupt instruction, it unconditionally disables interrupts. The programmer must execute an EIN instruction to re-enable interrupts before another interrupt can be recognized.
3. Auto I/O Instructions. The Automatic I/O instructions are single instructions which require 4-1/2 cycles to execute. Since the computer must execute one instruction following the interrupt instruction, the total delay required before another interrupt can be recognized is 4-1/2 cycles plus the number of cycles required to execute the next instruction. The safest practice would be to compute the maximum delay as 4-1/2 cycles plus the number of cycles of the longest instruction in the main program sequence where interrupts may be generated.
4. Instruction "Stretch", Every instruction which is executed as an interrupt instruction is stretched by 1/4 cycle. This is to allow the generation of the interrupt address. This additional 1/4 cycle must be added to the execution time of any instruction used as an interrupt instruction. (The IMS instruction is also stretched an additional 1/4 cycle. See IMS instruction description.)

3.2.5 Interrupt Priorities

When more than one peripheral device or computer option is operating under interrupt control, a priority scheme must be established to determine which interrupt will be processed first if more than one interrupt is waiting to be processed.

In general, the highest priority interrupt waiting to be processed will be processed first, regardless of the sequence in which interrupt requests are generated. This means that if a lower priority device generates an interrupt request first, and then a higher priority device generates a request before the lower priority request is recognized by the computer, the higher priority request will be recognized and processed first by the computer.

3.2.5.1 Standard Priorities. The standard priorities which have been established for the ALPHA 16 and NAKED MINI 16 computers are as follows:

1. Power Fail Option. The Power Fail/restart option has the highest priority for interrupt processing in the computer, provided the option is installed in the computer. Power Fail is on a separate interrupt line.
2. Console Interrupt and Trap. The Console interrupt and the Trap instruction share the second highest priority. Console and Trap interrupts take priority over all interrupts except power fail. Console and Trap interrupts are on a separate interrupt line from all other interrupts.
3. Interrupt Line 1 (IL1). IL1 has the third level of priority. The peripheral device assigned to IL1 will have the highest priority of all peripheral devices or options except Power Fail or the Console/Trap interrupt.
4. Interrupt Line 2 (IL2). Interrupt Line 2 has the next priority level.
5. Memory Protect Option. The Memory Protect option generates an interrupt when a write operation is attempted in the protected area of memory with Memory Protect enabled.
6. Real Time Clock (RTC). The RTC option generates two interrupts which share equal priority. These two interrupts are on special interrupt lines.
7. Teletype Interface (TTY). The TTY interface requests interrupts on the Interrupt Request (IUR) line, but has the highest priority on that line.
8. Interrupt Request Line (IUR). All remaining interrupts are vectored on the IUR line. The priority of the devices using this line is determined by the physical location of the devices interface in the ALPHA 16 or the NAKED MINI 16 chassis. In the basic chassis, the priority sequence is:

   a. Slot E200

   b. Slot E100

   c. Slot F100

   d. Slot F200

3.2.6 Reserved Interrupt Locations in Memory

The standard interrupt locations which are assigned to computer functions and common options are summarized in Table 3-1. Note that several functions and options have two interrupt locations. Refer to the description of the function or option for the use of each interrupt location.

3.2.6.1 Interrupt Offset. All of the standard interrupt locations are in the Scratchpad area of memory. Since Scratchpad is the only area of memory that can be addressed directly by an instruction located anywhere in memory, it may prove useful to move interrupt locations outside of Scratchpad. A computer option allows all standard interrupt locations (except Power Up, which is not really an interrupt) to be displaced by :100 locations. In Table 3-1, the Offset column gives the interrupt location if the offset option is exercised.

Table 3-1. Standard Interrupt Locations

Function (By Priority)	Standard Location	Offset Location
Power Fail/Restart:		
Power Down Interrupt	:001C	:011C
Power Up Restart location	:0000	:0000
Console Interrupt and Trap	:001E	:011E
Interrupt Line 1 (IL1)	:0002	:0102
Interrupt Line 2 (IL2)	:0006	:0106
Memory Protect	:0014	:0114
Real Time Clock:		
Clock Interrupt	:0018	:0118
Sync Interrupt	:001A	:011A
Teletype (TTY):		
Standard		
End of Word Interrupt	:0002	:0102
End of Block Interrupt	:0006	:0106
Optional		
End of Word Interrupt	:0022	:0122
End of Block Interrupt	:0026	:0126
Interrupt Request Line (IUR)	*	*

*Interrupt locations are determined by interface design.

3.2.6.2 IUR Interrupt Locations. Peripheral devices which generate interrupt requests on the IUR line are not assigned standard interrupt locations by the computer. Instead, the interrupt address is assigned by the interface designer. Interrupt locations may be anywhere in memory. They are not limited to Scratchpad, or even to the lower 4K words of memory. They can be assigned anywhere. When an interface is being designed, the design engineer and the programmer should work together to determine the optimum interrupt location address.

3.3 GENERAL INPUT/OUTPUT INSTRUCTIONS

3.3.1 General

The General I/O instructions are those instructions which are used for single word or single byte data moves, and for general conditioning and interrogation of peripheral interfaces. These instructions may be used to load or read data buffers in an interface, trigger control flip flops or relays, sense the state of a flip flop or incoming line, and other similar functions.

3.3.1.1 Instruction Types. The instructions in the General I/O group are these:

1. Sense. Sense instructions are used to test certain conditions in the peripheral interfaces and perform conditional branches on the results of the tests.

2. Select. Select instructions are used to condition peripheral interfaces to perform certain functions other than data transfer. These instructions may be used to set control flip flops, set interrupt masks, reset the interface, etc. The functions that are performed by the Select instructions are determined by the design of the individual peripheral interfaces.

3. Input. There are several types of input instructions in this group. However, all of the input instructions in this group read data from the peripheral interface to either the A or X register in the computer. Data may be read either as full 16-bit words or as 8-bit bytes. Inputs may be masked so that only certain bits are recognized. Inputs may be unconditional, or they may be combined with Sense functions to read data upon a sense response.

4. Output. Output instructions move data from either the A or X register in the computer to the peripheral interface. Outputs may be unconditional, or they may be combined with sense functions to output data only upon a sense response.

3.3.1.2 Instruction Format. Figure 3-5 illustrates the format of the General I/O instruction group. Bits 14 and 15 identify the instruction as being part of the I/O class. Bits 8-13 define the specific instruction within the class. Bits 0-7 are arbitrarily divided into a Device Address in bits 3-7, and a Function Code in bits 0-2.

If the Device Address is considered to be contained in bits 3-7 and the Function Code in bits 0-2, each instruction may address up to 32 different devices and have up to 8 different functions specified with each address. When an instruction is executed, a signal is sent from the computer to the peripheral interface to tell the interface what type of instruction is being executed. The functions that will be performed by the peripheral interface are determined by both the function code and the type of instruction that is being executed. For example, a function code of "4", when used with a Select instruction, may cause the interface to turn on a particular control flip flop. The same function code, when used with a Sense instruction may test to see if that control flip flop is turned on. The same function code, when used with an Input instruction may gate a certain set of lines to the Data Bus. Therefore, each function code may not have just one meaning. It may have a different meaning for each type of instruction with which it is used.

The division of bits 0-7 into a Device Address and Function Code is purely an arbitrary division. The user may wish to consider all eight bits as a single Device Address field, with each function within a device having a separate address. If this convention is used, the computer may be considered to have the capability of addressing up to 256 different devices.

3.3.1.3 Description Format. The instruction descriptions which follow use the same general format as that used for Memory Reference instructions. Variations in the description format are:

1. Instruction Diagram. Bit 9 of the Input and Output instructions contains the letter R. Since data transfers are made between the peripheral interface and either the A or X register for the General I/O instructions, bit 9 specifies which computer register takes part in the transfer. The identification is:

   R=0, A Register

   R=1, X Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	OP CODE						DEVICE ADDRESS					FUNCTION CODE		

Field	Bits	Description
Class	14, 15	These bits define the I/O instruction class.
Operation Code	8-13	These bits identify the specific I/O instruction that is being executed.
Device Address	*3-7	Used to select the specific peripheral device which is to respond to the I/O instruction.
Function Code	*0-2	Specified the function which the peripheral device is to perform. May also be used to identify a data source within the interface, or a status that is being sensed.

*The Device Address and Function Code fields may be combined into a single Device Address field, where each function within a device has a separate address.

Figure 3-5. General Input/Output Instruction Format

2. Machine Codes. The Machine Codes section shows the possible hexadecimal codes that may be used in the two upper positions, and the letters "nn" in the two lower positions. The letters "nn" stand for the variable Device Address and Function Code fields which are determined by the device with which the instruction is used.

3.3.2 Sense Instructions

The Sense and Skip instructions allow the ALPHA 16 and NAKED MINI 16 computers to sense the state of a specified function in a peripheral interface and execute a conditional skip depending on the result of the test. There are two instructions in this group. One causes a skip on a true response, and the other causes a skip on a false response.

3.3.2.1

SEN SENSE AND SKIP ON RESPONSE

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	1	DEVICE ADDRESS					FUNCTION CODE		

Tests the specified function in the specified device. If a true response is obtained, a one-place skip is executed. If a false response is obtained, the next instruction in sequence is executed.

Machine Codes:

:49nn

Registers Affected:

P Incremented normally if a false response obtained: (P)+1→P.

Incremented twice if a true response obtained: (P)+2→P.

Timing: 1 1/4

3.3.2.2

SSN SENSE AND SKIP ON NO RESPONSE

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	DEVICE ADDRESS					FUNCTION CODE		

Tests the specified function in the addressed device. If a false response is obtained, a one-place skip is executed. If a true response is obtained, the next instruction in sequence is executed.

Machine Codes:

:48nn

Registers Affected:

P Incremented normally if a true response is obtained: (P)+1→P.

Incremented twice if a false response is obtained: (P)+2→P.

Timing: 1 1/4

3.3.3 Select Instructions

Select instructions are used to set up conditions in a peripheral interface which will cause the peripheral device to perform some specified function. Select instructions are sometimes called "External Control" instructions because they are used primarily for control functions rather than data transfer functions.

There are two basic instructions in the Select group. One instruction presents a Device Address and Function Code to the peripheral interface along with a control signal stating that the instruction being executed is a Select instruction. The peripheral interface examines the Device Address, Function Code, and control signal to determine what function is to be performed.

The other instruction in this group does exactly the same thing, but in addition it places the contents of either the A or X register on the Data Bus. The peripheral interface then examines the Device Address, Function Code, control signals, and Data Bus to determine what functions are to be performed.

3.3.3.1

SEL SELECT FUNCTION

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	0	0	0	0	0	0	DEVICE ADDRESS	FUNCTION CODE

The Function Code is transmitted to the addressed device along with a Select Control signal. The actual function performed within the device is a function of interface design.

Machine Codes:

:40nn

Registers Affected:

None in the computer.

Timing: 1 1/4

3.3.3.2

SEA SELECT AND PRESENT A

SEX SELECT AND PRESENT X

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	0	0	0	1	R	0	DEVICE ADDRESS	FUNCTION CODE

The Function Code is transmitted to the addressed device interface with control signals. In addition, the contents of either the A or X register are placed on the I/O data bus.

Machine Codes:

:44nn SEA
:46nn SEX

Registers Affected:

None in the computer.

Timing: 1 1/4

3.3.4 Input to Register Instructions

The Input to Register group of instructions cause data to be moved from a peripheral interface or device to either the A or X register of the computer. Input instructions may input either full 16-bit words or 8-bit bytes. If a byte input instruction is used, the byte is read into the lower half of the receiving register without affecting the upper half of the register.

Inputs may be unconditional, or may be conditioned on sense response. Unconditional inputs read the specified data source within the peripheral device regardless of the conditions existing in the device. Inputs conditioned on sense response sense a specified condition in the peripheral device and input on a true response. If a true response is not received, the computer repeats the input instruction. The computer effectively "hangs" on the input instruction until a true response is received. Input instructions which are conditioned by a sense response are interruptable; i.e., if an external interrupt is received while the computer is executing an instruction which inputs on a true response, the computer will recognize the interrupt at the end of the test and, if the input was not accomplished (a true response was not received), the computer will return to the execution of the input instruction after the interrupt is processed.

Inputs may be made directly to the receiving register, or may be ANDed with the contents of the receiving register with the results of the AND operation replacing the original contents of the register. ANDing the input data with the contents of the receiving register is called a Masked input. For masked word inputs, the input data is ANDed with the full 16 bits of the receiving register. For masked byte inputs, the input data is ANDed with the lower half of the receiving register and the upper half of the receiving register is unchanged.

3.3.4.1

INA INPUT TO A REGISTER (UNCONDITIONALLY)

INX INPUT TO X REGISTER (UNCONDITIONALLY)

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	R	0	DEVICE ADDRESS					FUNCTION CODE		

Unconditionally inputs a full 16-bit word from the addressed device to the A or X register. The previous contents of the selected receiving register are lost. (The source of data in the addressed device may be specified by the function code.)

Machine Codes:

:58nn INA
:5Ann INX

Registers Affected:

A or X Previous contents replaced by input word.

Timing: 1 1/4

3.3.4.2

IBA INPUT BYTE TO A REGISTER (UNCONDITIONALLY)

IBX INPUT BYTE TO X REGISTER (UNCONDITIONALLY)

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	R	0	DEVICE ADDRESS					FUNCTION CODE		

Unconditionally inputs an 8-bit byte from the addressed device to the lower half of the selected receiving register. The upper half of the receiving register is unchanged. (The source of the data in the addressed device may be specified by the function code.)

Machine Codes:

:78nn IBA
:7Ann IBX

Registers Affected:

A or X Previous contents of lower half replaced by input byte.

Timing: 1 1/4

3.3.4.3

INAM MASKED INPUT TO A REGISTER (UNCONDITIONALLY)

INXM MASKED INPUT TO X REGISTER (UNCONDITIONALLY)

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	0	1	1	1	R	0	DEVICE ADDRESS	FUNCTION CODE

Inputs a full 16-bit word from the addressed device. The incoming word is logically ANDed with the contents of the selected receiving register, and the results are stored in the selected receiving register:

$$(\text{Input word}) \wedge (A) \rightarrow A$$

or

$$(\text{Input word}) \wedge (X) \rightarrow X$$

This instruction is normally used to mask off unwanted bits or fields from the incoming word.

Machine Codes:

:5Cnn INAM
:5Enn INXM

Registers Affected:

A or X Previous contents replaced by masked input.

Timing: 1 1/4

3.3.4.4

IBAM INPUT BYTE TO A REGISTER MASKED (UNCONDITIONALLY)

IBXM INPUT BYTE TO X REGISTER MASKED (UNCONDITIONALLY)

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	1	1	1	1	R	0	DEVICE ADDRESS	FUNCTION CODE

Inputs a byte from the addressed device. The incoming data is logically ANDed with the lower 8 bits of the receiving register, and the results are placed in the lower 8 bits of the receiving register:

$$(\text{Input Byte}) \wedge (A)_{0\text{-}7} \rightarrow A_{0\text{-}7}$$

or

$$(\text{Input Byte}) \wedge (X)_{0\text{-}7} \rightarrow X_{0\text{-}7}$$

This instruction is normally used to mask off unwanted data bits from the incoming byte and retain only those bits which are wanted.

Machine Codes:

:7Cnn IBAM
:7Enn IBXM

Registers Affected:

A or X Previous contents of lower half replaced by masked input.

Timing: 1 1/4

3.3.4.5

RDA READ WORD TO A REGISTER

RDX READ WORD TO X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	0	1	1	0	R	1	DEVICE ADDRESS	FUNCTION CODE

Senses the specified data source in the addressed device. If a true response is received, a word is input from the device to the selected register. If a false response is received, the instruction is repeated. The instruction continues to repeat itself until a true response is received.

NOTE: This instruction is interruptable.

Machine Codes:

:59nn RDA
:5Bnn RDX

Registers Affected:

A or X Previous contents replaced by input word.

Timing: 1 1/4

3.3.4.6

RBA READ BYTE TO A REGISTER

RBX READ BYTE TO X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	1	1	1	0	R	1	DEVICE ADDRESS	FUNCTION CODE

Senses the specified data source in the addressed device. If a true response is received, a byte is input from the device to the lower half of the selected register. If a false response is received, the instruction is repeated. The instruction continues to repeat itself until a true response is received.

Note: This instruction is interruptable.

Machine Codes:

:79nn RBA
:7Bnn RBX

Registers Affected:

A or X Previous contents of lower half replaced by input byte. Upper half unchanged.

Timing: 1 1/4

3.3.4.7

RDAM READ WORD TO A REGISTER MASKED

RDXM READ WORD TO X REGISTER MASKED

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	0	1	1	1	R	1	DEVICE ADDRESS	FUNCTION CODE

This instruction is a combination of the Read Word instruction and the Input Word Masked instruction. The specified data source in the addressed device is sensed. If a true response is obtained, a word is read from the device. The input word is ANDed with the contents of the selected register and the result is stored in the selected register. If a false response is obtained, the instruction repeats itself until a true response is obtained.

Note: This instruction is interruptable.

Machine Codes:

:5Dnn RDAM
:5Fnn RDXM

Registers Affected:

A or X Previous contents replaced by masked input.

Timing: 1 1/4 minimum

3.3.4.8

RBAM READ BYTE TO A REGISTER MASKED

RBXM READ BYTE TO X REGISTER MASKED

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	1	1	1	1	R	1	DEVICE ADDRESS	FUNCTION CODE

This instruction is a combination of the Read Byte instruction and the Input Byte Masked instruction. The specified data source in the addressed device is sensed. If a true response is obtained, a byte is read from the device. The input byte is ANDed with the lower half of the selected register, and the results of the AND are stored in the lower half of the register. If a false response is obtained, the instruction is repeated until a true response is obtained.

Note: This instruction is interruptable.

Machine Codes:

:7Dnn RBAM
:7Fnn RBXM

Registers Affected:

A or X — Previous contents of lower half replaced by masked input. Upper half unchanged.

Timing: 1 1/4

3.3.5 Output from Register Instructions

The Output from Register instructions transfer data from either the A or X register to the addressed device. The function code of the instruction may be used to specify the destination of the data in the addressed device.

Outputs may be unconditional, or they may be conditioned on a sense response. If conditional output instructions are used, the instruction function code is normally used to specify the condition being sensed within the addressed device. Conditional output instructions effectively "hang" until a true response is received from the addressed device. Unconditional output instructions transfer data to the device regardless of the conditions existing in the device. Unconditional output instructions are normally used in conjunction with Sense instructions to see if a device is ready to accept data.

Output instructions transfer a full 16-bit word to the addressed device. If the device is byte oriented, it is normally designed to accept only the lower 8 bits of the word. The registers in the computer which are used as sources of output data are not changed by output instructions. Thus, for byte oriented peripheral devices, a register can be loaded with a full word and the word can be output to the device. The device accepts only the lower 8 bits. The word in the register can then be shifted and the other 8 bits transferred to the device.

Since the ALPHA 16 and NAKED MINI 16 computers may address byte operands, an alternate byte transfer mode is to load the output register one byte at a time. A single output instruction in an output loop is then sufficient to output bytes to the peripheral device.

3.3.5.1

OTA — OUTPUT A REGISTER (UNCONDITIONALLY)

OTX — OUTPUT X REGISTER (UNCONDITIONALLY)

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	1	0	1	1	R	0	DEVICE ADDRESS	FUNCTION CODE

Unconditionally outputs the contents of the A or X register to the addressed device. (The function code may be used to specify the destination of the data within the selected device.)

Machine Codes:

:6Cnn OTA
:6Enn OTX

Registers Affected:

None in the processor.

Timing: 1 1/4

3.3.5.2

OTZ — OUTPUT ZERO (UNCONDITIONALLY)

I REGISTER

15	14	13	12	11	10	9	8	7 6 5 4 3	2 1 0
0	1	1	0	1	0	R	0	DEVICE ADDRESS	FUNCTION CODE

This is an unconditional output instruction which places an all-zero word on the I/O Data Bus, along with output control signals to the addressed device.

Machine Codes:

:68nn

or

:6Ann

Registers Affected:

No processor registers.

Timing: 1 1/4

3.3.5.3

WRA WRITE FROM A REGISTER

WRX WRITE FROM X REGISTER

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	R	1	DEVICE ADDRESS					FUNCTION CODE		

The specified condition in the addressed device is sensed. If a true response is received, the contents of the selected register are transferred to the device. If a false response is received, the instruction is repeated until a true response is received.

Note: This instruction is interruptable.

Machine Codes:

:6Dnn WRA
:6Fnn WRX

Registers Affected:

None in the processor.

Timing: 1 1/4 minimum

3.3.5.4

WRZ WRITE ZEROS

I REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	R	1	DEVICE ADDRESS					FUNCTION CODE		

The specified condition in the addressed device is sensed. If a true response is obtained an all-zero word is transferred to the device. If a false response is obtained the instruction is repeated until a true response is obtained.

Note: This instruction is interruptable.

Machine Codes:

:69 nn

or

:6Bnn

Registers Affected:

None in the processor.

Timing: 1 1/4 minimum

3.4 BLOCK TRANSFER INSTRUCTIONS

3.4.1 General

To allow very high speed input and output from memory the ALPHA 16 and NAKED MINI 16 computers incorporate a very powerful pair of instructions called BLOCK IN and BLOCK OUT. A block of any length (limited only by memory size) may be transferred into or out of memory at a maximum rate of 500,000 16-bit words per second. If the peripheral interface is capable of unpacking bytes, the transfer rate may be thought of as 1,000,000 bytes per second. However, the Block Transfer instructions operate only with full 16-bit words.

The processor is totally devoted to the Block Transfer instruction until the entire block has been transferred. Data does not go through the A or X register during the move, but instead goes directly between the peripheral interface and memory.

3.4.2 Block Transfer Operation

Block Transfer instructions are double word instructions. The first word contains the instruction operation code, device address, and function code as in general I/O instructions. The second word contains the base address, minus one, of the block of data to be moved.

3.4.2.1 Word Count. Whenever a block of data is to be moved by a single instruction between memory and a peripheral device, the number of words that the instruction is to move must be known. In order to increase the speed of the Block Transfer instructions, the X Register in the computer is used to hold the word count. As each word is moved, the word count in the X Register is decremented. When the word count is decremented to zero, the instruction terminates. The programmer must load the X Register with the word count before executing a Block Transfer instruction. (If the word count were stored in memory, an extra cycle would be required for each word transferred to decrement and test the word count.)

3.4.2.2 Base Address. Whenever data is to be moved between memory and a peripheral device, the address of that data in memory must be known. In the case of an output instruction, the address is the location of the data that must be transferred to the peripheral device. In the case of an input instruction, the address is the location where the data must be stored after it is received from the peripheral device. For the Block Transfer instructions, the base address, minus one, is contained in the memory location immediately following the Block Transfer instruction:

P = BLOCK TRANSFER INSTRUCTION

P+1 = BASE ADDRESS – 1

The Block Transfer instructions are effectively double word instructions. The first word is the instruction, and the second word is the base address of the data block in memory.

3.4.2.3 Data Movement. When a Block Transfer instruction is executed, the address of the first word to be moved is formed by adding the word count in the X Register to the value in location P+1, which contains the Base Address less one. Therefore, the address of the first word to be moved is effectively formed by indirect indexed addressing. The address thus formed is stored in the memory address, M, register. Each subsequent address is obtained by decrementing the M register. The sequence of movement of data is the word at the highest address first, and the word at the base address last:

First Word Moved:	(P+1)+(X)
Second Word Moved:	(P+1)+(X)–1
*	
*	
*	
Last Word Moved:	(P+1)+1

The instruction terminates when the X Register is decremented to zero. A word is transferred when (X)=1, but a word is not transferred when (X)=0.

3.4.2.4 Sense Response. Before a word is transferred, the peripheral device is sensed. If a true response is received, a word is transferred and the M and X registers are decremented. If a false response is received, the M and X registers are not decremented and the device is sensed again. (This requires another instruction cycle.) Thus the speed of the data transfer is a function of the speed of the peripheral device. The processor tests the peripheral device once during each computer cycle until a true response is obtained. Data transfers are made only after a true response is received from the peripheral device. The maximum data transfer rate is 500,000 16-bit words per second, or 2.0 microseconds per word.

3.4.2.5 Interrupt Considerations. There are two factors concerned with interrupt operation which must be considered:

1. Not Interruptable. Block transfer instructions cannot be interrupted. The ALPHA 16 and NAKED MINI 16 computers can be interrupted only at the end of the instruction being executed when the interrupt is received. Block transfer instructions do not end until the last word of the block has

been transferred, therefore they cannot be interrupted until the last word has been transferred. The computer is totally dedicated to the data move.

2. Not Interrupt Instructions. Block transfer instructions cannot be used as normal interrupt instructions since the X Register must be loaded with the word count prior to executing the instruction. This means that Block Transfer instructions cannot be used as single-instruction interrupt processing instructions. It does not mean that these instructions cannot be used as part of an interrupt subroutine.

3.4.2.5 Program Counter. Since Block Transfer instructions occupy two sequential words of memory, the P Register (Program Counter) must be incremented twice to point to the next instruction to be executed. Thus, if a Block Transfer instruction is located at location P in memory, the Program Counter will point to location P+2 after the instruction is executed:

P = BLOCK TRANSFER INSTRUCTION

P+1 = BASE ADDRESS – 1

P+2 = NEXT INSTRUCTION

3.4.2.6 Programming Example. Use of Block Transfer instructions is relatively simple. The data buffers must be set up as for any other movement of data to or from memory. The only thing to remember is that data will be moved using the highest address in the buffer and then using sequentially lower addresses. In coding the instruction itself, a location must be reserved for the Base Address word immediately following the Block Transfer instruction, and the X Register must be loaded prior to executing the instruction. The coding for a Block In may be as follows:

LDX	COUNT	Load the X Register with the Word Count.
BIN	DA/FC	Block Input instruction with Device Address and Function Code.
DATA	BA	Put the Base Address, less 1, at the location immediately following the BIN instruction.
(Next instruction)		The next location continues the program. When BIN is executed, P is incremented twice instead of only once. This causes the computer to skip the Base Address word.

3.4.2.7

BIN BLOCK IN

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P	0	1	1	1	0	0	0	1	DEVICE ADDRESS					FUNCTION CODE		

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P+1	0	BASE ADDRESS -1														

This instruction inputs a block of data from the addressed device. Each word is input upon receipt of a true response when the condition specified by the function code is sensed. The X Register must contain the Word Count, and location P+1 must contain the Base Address, less 1, of the data buffer in memory.

Machine Codes:

:71nn

Registers Affected:

X	Contains all zeros after the instruction is completed.
Memory	Previous contents of the data buffer in memory replaced by input data.
P	Incremented two times instead of one. P=(P)+2

Timing: 2 + ½ for each word transferred

3.4.2.8

BOT BLOCK OUT

	15	14	13	12	11	10	9	8	7–3	2–0
P	0	1	1	1	0	1	0	1	DEVICE ADDRESS	FUNCTION CODE

	15	14–0
P+1	0	BASE ADDRESS -1

This instruction outputs a block of data from memory to the addressed device. Each word is output upon receipt of a true response when the condition specified by the function code of the instruction is sensed. The X Register must contain the Word Count, and location P+1 must contain the Base Address, less 1, of the location of the data in memory.

Machine Code:

:75nn

Registers Affected:

X Contains all zeros when the instruction is completed.

P Incremented two times instead of one. P=(P)+2

Timing: 2+1/2 for each word transferred

3.5 AUTOMATIC INPUT/OUTPUT INSTRUCTIONS

3.5.1 General

Automatic Input/Output instructions are powerful I/O instructions which provide data transfers directly between memory and peripheral devices. An Automatic I/O instruction is effectively a complete I/O subroutine in a single instruction. When a block of data is to be moved between the computer memory and a peripheral device using a subroutine, the subroutine must perform these functions:

1. Decrement a word count so that the subroutine will know when the last word has been moved.
2. Increment an address pointer to point to the next data location in memory for reading or storing data.
3. Transfer a data word between memory and the peripheral device.

An Automatic I/O instruction performs all of these functions in a single instruction.

There are four Automatic I/O instructions which the ALPHA 16 and NAKED MINI 16 computers are capable of executing: Automatic Input Word, Automatic Input Byte, Automatic Output Word, and Automatic Output Byte. The Automatic Word instructions transfer full 16-bit words between the computer memory and peripheral devices. The Automatic Byte instructions transfer 8-bit bytes, packed two bytes per word, between memory and peripheral devices.

Automatic I/O instructions may be used for in-line programming, or as interrupt instructions. They are primarily designed to be used as interrupt instructions. When used as interrupt instructions, they provide a virtually unlimited number of Direct Memory Channels in the computer for fast transfers of data directly between memory and peripheral devices under interrupt control.

3.5.2 Format

Each Automatic I/O instruction occupies three words in memory. The first word contains the instruction, the second word contains the two's complement of the word count, and the third word contains an address pointer:

P	AUTOMATIC I/O INSTRUCTION
P+1	WORD COUNTER
P+2	ADDRESS POINTER

3.5.2.1 Instruction. The instruction word has the same general format as all other I/O instructions. The operation code is contained in bits 8-15, and the Device Address and Function Code are contained in bits 0-7.

3.5.2.2 Word Count. The word immediately following the instruction word must contain the negative of the word count; i.e., if n words are to be moved, the Word Counter must contain –n (two's complement of the word count). Each time the instruction is executed, the Word Counter is incremented and tested for zero. Therefore, the Word Counter counts from –n to 0.

3.5.2.3 Address Pointer. The Address Pointer must contain an address which is one less than the address of the first word to be moved; i.e., if the first word to be moved is at location m, then the Address Pointer must contain the value m–1. The reason is that the Address Pointer is incremented before it is used as an address to memory.

3.5.3 Operation

Automatic I/O instructions are unconditional transfer instructions. When the instruction is executed, a word or byte is unconditionally moved between the peripheral device and memory. Each time the instruction is executed one word or byte is moved. Four memory cycles are required for each word or byte transferred. The memory cycles are:

1. Read and decode the Automatic I/O instruction.
2. Read the Word Counter, increment it, restore it, and test it for zero.
3. Read the Address Pointer, increment it, restore it, and use it as an address to memory.
4. Transfer the data: read it from memory for an output or store it in memory for an input.

3.5.3.1 Timing. Since the Automatic I/O instructions are I/O instructions, the first memory access will require a total of 1 1/4 computer cycles. Each subsequent memory cycle will require 1 computer cycle. Therefore, each word or byte transferred requires a total of 4 1/4 computer cycles to finish the transfer.

3.5.3.2 Word Transfers. Automatic I/O Word instructions transfer full 16-bit words between the computer memory and the peripheral device. Figure 3-6 illustrates the sequence of words moved, and the addressing of those words. The memory location for the first word (source if an output; destination if an input) is one greater than the address in the Address Pointer of the Auto I/O instruction. If the address in the address pointer is identified by the symbol BA, then the location of the first data word is BA+1. If n words are moved, the location of the last word moved in BA+n. The words moved, then, are located at BA+1 through BA+n. Words are transferred using the lowest address in the memory buffer first through the highest address in the buffer.

3.5.3.3 Byte Transfers. The only difference between Automatic I/O Byte instructions and Automatic I/O Word instructions is that the byte instructions move 8-bit bytes rather than full words. The byte instructions address sequential bytes as illustrated in Figure 3-7. For the byte instructions, the word immediately following the Auto I/O Byte instruction is a Byte Count rather than a word count. The Address Pointer contains a Byte Address rather than a Word Address. The Auto I/O Byte instruction handles the packing and unpacking of bytes which are stored two bytes per word. The starting byte address may be odd or even (left right byte within a word), and the number of bytes moved is limited only by memory size.

3.5.4 Direct Memory Channels

When an Automatic Input/Output instruction is used as an interrupt instruction, a Direct Memory Channel is formed. Since these instructions are single instructions which do the job of a complete I/O subroutine, they are ideal as interrupt instructions.

3.5.4.1 Word Transfers. When an interrupt is recognized by the ALPHA 16 or NAKED MINI computer, the computer executes one instruction at the interrupt location. If the instruction at the interrupt location

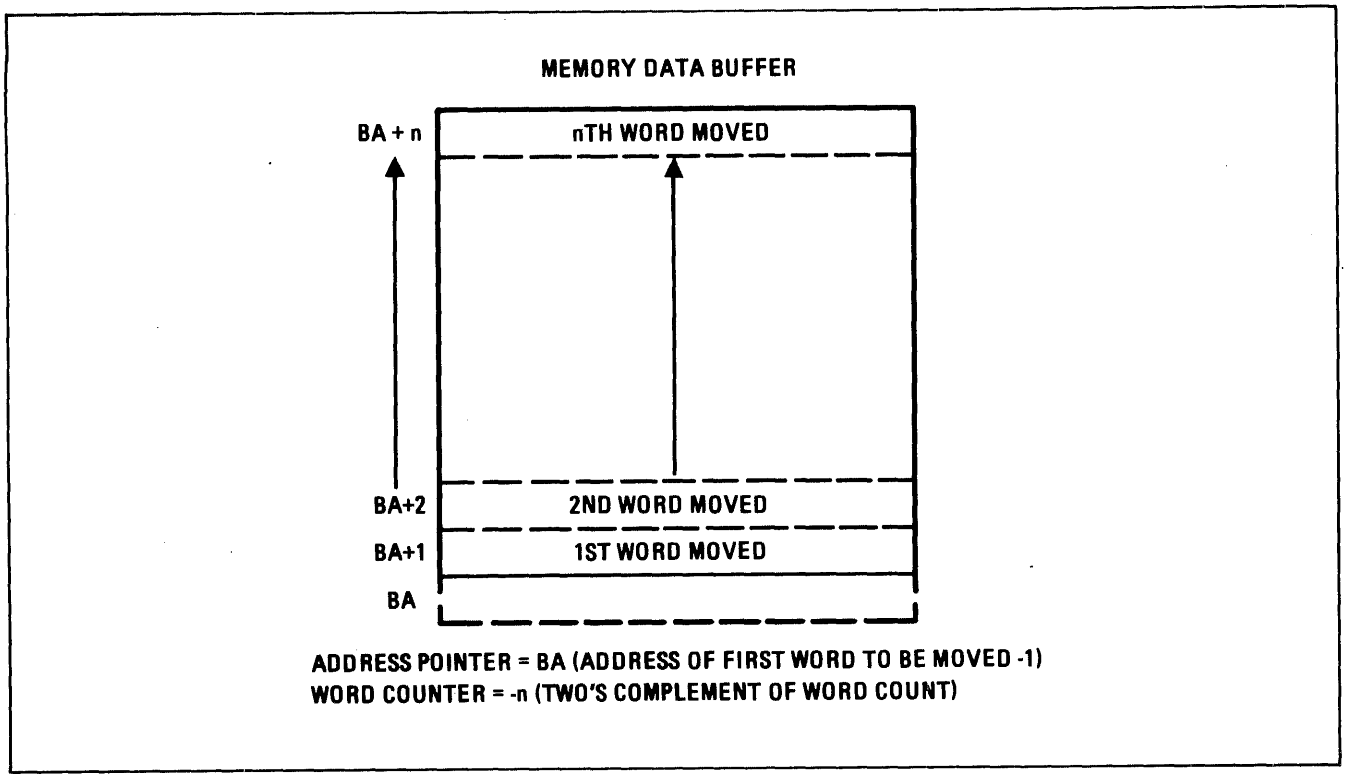


Figure 3-6. Word Movement Sequence

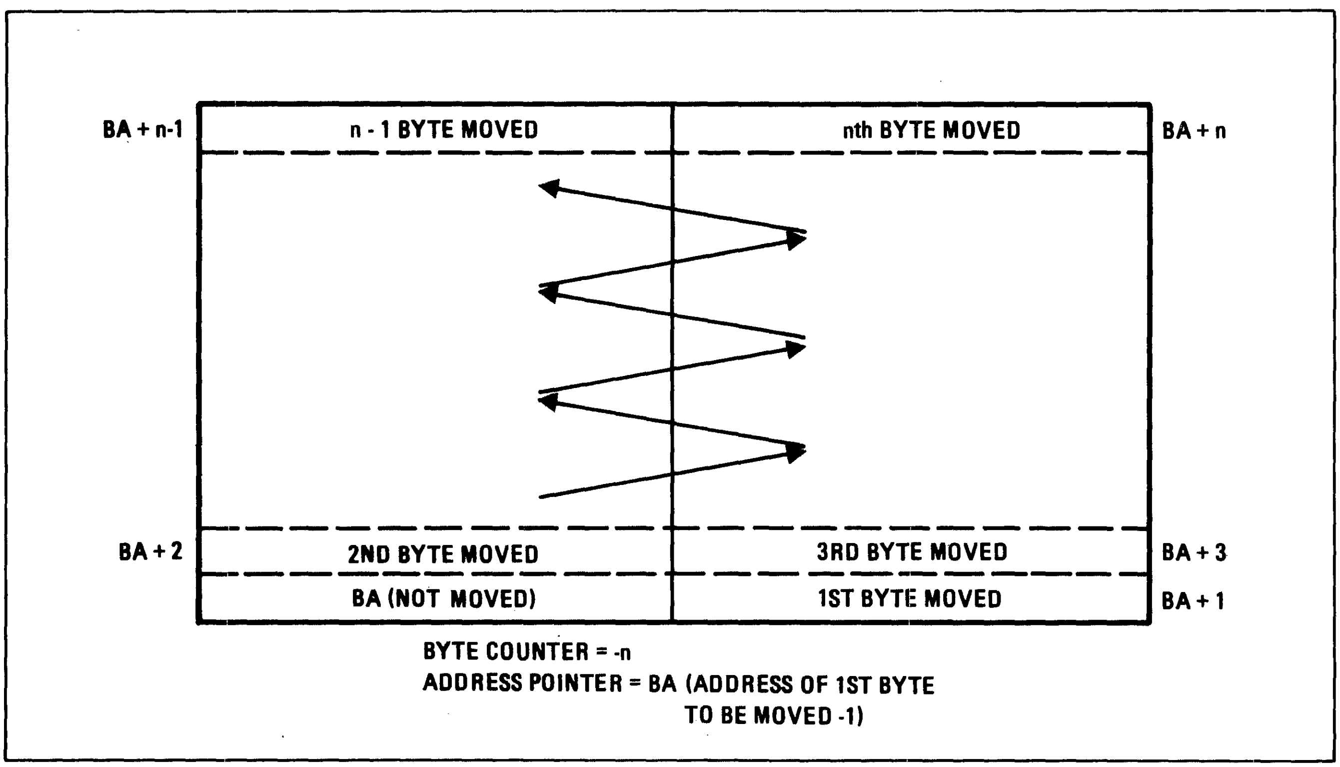


Figure 3-7. Byte Movement Sequence

is an Automatic I/O instruction, the following sequence of events occurs:

1. The Automatic I/O instruction is read from memory and decoded. The instruction addresses the interrupting device and selects the data source/destination in the device.

2. The Word/Byte Count is read from the location immediately following the Auto I/O instruction. The word or byte count is incremented and tested for zero, and the updated count is written back into memory.

3. The Address Pointer is read from the second location following the Auto I/O instruction. The word or byte address is incremented and restored to memory. The incremented address is also used as an address to memory for reading or storing data.

4. A word or byte is transferred between the computer memory and the peripheral device. The data is stored in memory for an input, or read from memory for an output. This completes the transfer of a single word or byte.

Each time the peripheral device is ready to transfer a word or byte, it interrupts the computer and the above sequence is repeated. This continues until all data has been moved.

3.5.4.2 End of Block. When the Word/Byte count reaches zero, the computer sends an Echo signal to the peripheral device to indicate that all data has been moved. The action at that point is determined by the design of the peripheral interface. The normal action is that the peripheral interface will generate an End of Block interrupt to a different interrupt location to signal that the peripheral device has completed its job. It is then up to the programmer to do the necessary housekeeping associated with the end of a data transfer.

3.5.5 In-Line Programming

Although Automatic I/O instructions are designed to be used as interrupt instructions, they can also be used as in-line instructions. The format is expanded somewhat when this application is used.

3.5.5.1 Format. When an Automatic I/O instruction is executed as an in-line instruction, a single word or byte is unconditionally transferred each time the instruction is executed. If device sensing is required to determine whether or not the peripheral device is ready for a data transfer, the sensing must be accomplished before the Automatic I/O instruction is executed. When the Automatic I/O instruction is executed the location of the next instruction to be executed is a function of the word count in the location following the Auto I/O instruction. The format of the program, including the Auto I/O instruction and the locations which immediately follow, is as follows:

P	AUTOMATIC I/O INSTRUCTION
P+1	WORD/BYTE COUNTER
P+2	ADDRESS POINTER
P+3	END OF BLOCK EXIT (WORD COUNT = 0)
P+4	NEXT INSTRUCTION (WORD COUNT ≠ 0)

3.5.5.2 Operation. When an Auto I/O instruction is executed in-line, it functions exactly the same as when it is executed as an interrupt instruction in that the Word/Byte counter and the Address Pointer are incremented and restored and a single word or byte is transferred between memory and the peripheral device. However, the instruction which is executed after the Auto I/O instruction depends on whether or not the word/byte count is equal to zero. If the Auto I/O instruction is at P, and the count at P+1 is not equal to zero after it is incremented, the instruction at P+4 will be executed following the Auto I/O instruction. However, if the count at P+1 is equal to zero after it is incremented, the instruction at P+3 will be executed following the Auto I/O instruction. The instruction at P+3 is referred to as the "End of Block Exit", since it will normally be an unconditional jump to get out of the data transfer loop.

3.5.6 Instruction Descriptions

Descriptions of the individual Automatic I/O instructions follow. The description format follows the same general format as that used for Memory Reference instructions, except that the instruction diagrams show the three words required by the instructions. The descriptions also assume an understanding of the preceding general description of the functions of these instructions.

3.5.6.1

AIN AUTOMATIC INPUT TO MEMORY: WORD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	0	DEVICE ADDRESS					FUNCTION CODE		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WORD COUNTER (TWO'S COMPLEMENT OF WORD COUNT)															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	ADDRESS POINTER (DATA LOCATION - 1)														

This instruction increments the Word Counter and the Address Pointer and inputs one word from the addressed device to the updated address in memory specified by the Address Pointer. When the word count is incremented to zero, the computer sends an Echo signal to the peripheral interface if the instruction is executed as an interrupt instruction, or executes the End of Block Exit instruction if the instruction is an in-line instruction.

Machine Codes:

:50nn

Registers Affected:

Memory — The two locations immediately following the instruction are incremented each time the instruction is executed, and the previous contents of the location addressed by the updated Address Pointer are replaced by the input word.

P — If the instruction is executed in-line, then P is incremented according to the contents of the Word Counter:

If Word Counter $\neq$ 0, then P+4 $\rightarrow$ P

If Word Counter = 0, then P+3 $\rightarrow$ P

Timing: 4 1/4

3.5.6.2

AOT AUTOMATIC OUTPUT FROM MEMORY: WORD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	DEVICE ADDRESS					FUNCTION CODE		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WORD COUNTER (TWO'S COMPLEMENT OF WORD COUNT)															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	ADDRESS POINTER (DATA LOCATION - 1)														

This instruction increments the Word Counter and the Address Pointer. It reads the word from the location addressed by the updated address in the Address Pointer and unconditionally outputs that word to the addressed peripheral device. When the word count is incremented to zero, the computer sends an Echo signal to the peripheral interface if the instruction is executed as an interrupt instruction, or executes the End of Block instruction if the instruction is executed in-line.

Machine Codes:

:60nn

Registers Affected:

Memory — The two locations immediately following the instruction are incremented each time the instruction is executed. The data locations in memory are unchanged.

P — If the instruction is executed in-line, then P is incremented according to the contents of the Word Counter:

If Word Counter ≠ 0, then P+4 → P

If Word Counter = 0, then P+3 → P

Timing: 4 1/4

3.5.6.3

AIB AUTOMATIC INPUT TO MEMORY: BYTE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	DEVICE ADDRESS					FUNCTION CODE		

15 — 0
BYTE COUNTER (TWO'S COMPLEMENT OF BYTE COUNT)

15 — 0
ADDRESS POINTER (BYTE LOCATION - 1)

This instruction increments the Byte Counter and the Address Pointer, and inputs one byte from the addressed device to the updated byte location in memory addressed by the Address Pointer. When the Byte Count is incremented to zero, the computer sends an Echo signal to the peripheral interface if the instruction is executed as an interrupt instruction, or executes the End of Block Exit instruction if the instruction is an in-line instruction.

Machine Codes:

:54nn

Registers Affected:

Memory — The two word locations immediately following the instruction are incremented each time the instruction is executed, and the previous contents of the byte location addressed by the updated Address Pointer are replaced by the input byte.

P — If the instruction is executed in-line, then P is incremented according to the contents of the Byte Counter:

If Byte Counter ≠ 0, then P+4 → P

If Byte Counter = 0, then P+3 → P

Timing: 4 1/4

3.5.6.4

AOB AUTOMATIC OUTPUT FROM MEMORY: BYTE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	0	DEVICE ADDRESS					FUNCTION CODE		

15 — 0
BYTE COUNTER (TWO'S COMPLEMENT OF BYTE COUNT)

15 — 0
ADDRESS POINTER (BYTE LOCATION -1)

This instruction increments the Byte Counter and Address Pointer. It reads the byte from the location addressed by the updated address in the Address Pointer and unconditionally outputs that byte to the addressed peripheral device. When the Byte Count is incremented to zero, the computer sends an Echo signal to the peripheral interface if the instruction is executed as an interrupt instruction, or executes the End of Block Exit instruction if the instruction is executed in-line.

Machine Code:

:64nn

Registers Affected:

Memory The two word locations immediately following the instruction are incremented each time the instruction is executed. The data locations in memory are unchanged.

P If the instruction is executed in-line, then P is incremented according to the contents of the Byte Counter:

If Byte Counter $\neq$ 0, then P+4 $\rightarrow$ P

If Byte Counter = 0, then P+3 $\rightarrow$ P

Timing: 4 1/4

SECTION 4

PROCESSOR OPTIONS

4.1 INTRODUCTION

4.1.1 General

There are five computer options which are mounted on basic processor boards and do not require separate I/O connector slots in the computer chassis. The options are:

- Teletypewriter (TTY) Interface
- Power Fail/Restart (PFR)
- Autoload (AL)
- Memory Protect (MP)
- Real Time Clock (RTC)

4.1.2 Standard Configurations

Most of the processor options may be installed individually in the processor. However, the Autoload option uses some of the circuitry used by the Power Fail/Restart option. Autoload cannot be installed without Power Fail/Restart, but Power Fail/Restart can be installed without Autoload. These options are manufactured in certain "off-the-shelf" configurations. Any configuration other than these configurations must be handled as a special case. The standard configurations are:

1. Power Fail/Restart and TTY Interface
2. Power Fail/Restart, TTY Interface, and Real time clock
3. Power Fail/Restart, TTY Interface, and Autoload
4. Power Fail/Restart, TTY Interface, Real Time Clock, and Autoload
5. Power Fail/Restart, TTY Interface, Real Time Clock, Autoload, and Memory Protect

4.2 TTY INTERFACE

4.2.1 General

The Teletypewriter (TTY) Interface option interfaces a modified ASR-33* or ASR-35* to the ALPHA 16 or NAKED MINI 16 computer. It performs all of the data and control signal conversion required for the computer to control the TTY. An ASR-33 or ASR-35 Teletypewriter provides four I/O features in one package: keyboard input, page printer, paper tape reader, and paper tape punch.

4.2.2 Operation

The interface contains a data buffer register which performs parallel-to-serial data conversion for outputing data from the computer to the Teletype, and serial-to-parallel conversion when inputing data from the Teletype to the computer. In addition the interface has provisions for interrupt generation for both word interrupts and end of block interrupts.

4.2.2.1 Device Address. The standard device address for the TTY Interface option is Device Address 7. Since the device address field of an I/O instruction spans two hexadecimal characters, the device address 7 with a function code of 0 would be written :38 in hexadecimal code.

4.2.2.2 Output. All outputs from the computer are printed on the page printer. If the punch is turned on at the teletype, outputs are also punched. The punch and the page printer cannot be controlled separately by the computer. Selecting punch outputs is an operator function.

4.2.2.3 Input. Inputs may be made from either the keyboard or the paper tape reader. The paper tape reader may be programmed to read a single byte from paper tape, or to continuously read tape. When the TTY Interface is conditioned to input data from the teletype, it will accept

*The ASR-33 and ASR-35 Teletypewriters are manufactured by Teletype Corporation.

data from any source. The only difference between selecting the keyboard and selecting the paper tape reader is that the instructions that select the paper tape reader turn the reader on. The instruction that selects the keyboard does not turn the reader on. A special feature allows inputs from the keyboard or tape reader to be automatically echoed back to the TTY for printing. This feature is called Automatic Echo.

CAUTION

WHEN THE PAPER TAPE READER IS READING TAPE, DO NOT DEPRESS KEYS ON THE KEYBOARD. IF KEYS ON THE KEYBOARD ARE DEPRESSED, INCORRECT DATA WILL BE SENT TO THE COMPUTER.

4.2.2.4 Control. Since the TTY operates at a much slower rate than the computer, there must be some method for sensing whether or not the interface is ready to accept data for output, or has assembled a complete byte for input. A Buffer Ready flip flop is used in the interface to indicate the condition of the interface data buffer. Generally when the computer is placed in a read mode, the Buffer Ready flip flop is turned off until a complete character has been assembled in the data buffer. When a complete character has been read from the teletype to the buffer, the Buffer Ready flip flop is turned on automatically by the TTY Interface. The Buffer Ready flip flow is then turned off again when the computer reads the byte from the interface buffer into the computer.

When the computer is sending data to the TTY Interface, the Buffer Ready flip flop is turned off when the computer loads the interface buffer with a byte. The interface turns the Buffer Ready flip flop on automatically after the interface has finished sending the byte to the TTY for printing.

4.2.3 Data Transfer Rates

The teletypes have a maximum data transfer rate of ten (10) bytes per second. Outputs to the printer and/or punch may be made at the maximum rate. Inputs from the paper tape reader are at the maximum rate when the reader is selected for continuous read mode. When inputing through the keyboard, care must be taken to insure that typing speeds do not exceed the maximum data rate (it is possible, even for a relatively slow typist, to exceed the maximum permissible rate for two consecutive characters.)

4.2.4 Programming

The teletype interface can be controlled using all types of I/O instructions. General I/O instructions must be used to condition the interface for data transfer. Once the interface has been conditioned, any type of instruction may be used for the actual transfer of data. Teletype speed does not require the speed associated with Block Transfer instructions, but these instructions may be used with the teletype.

4.2.4.1 General Instructions. The following is a list of general I/O instructions used with the teletype option. These instructions assume Device Address 7 is used to address the teletype. Table 4-1 lists the function codes associated with the TTY Interface.

SEL	:38		ENABLE AUTO ECHO. This instruction causes all inputs from the TTY keyboard or paper tape reader to be echoed back to the TTY for printing.
SEL	:39	(:4039)	SELECT Keyboard. This instruction resets the Buffer Ready flip-flop and puts the teletype interface in the read mode.
SEL	:3A	(:403A)	STEP Read. This command causes the character under the read station on the paper tape reader to be read and the tape advanced one character. The reader switch on the teletype must be in the RUN position. The Buffer Ready flip-flop is reset.
SEL	:3B	(:403B)	SELECT Continuous Read. This command causes the

			paper tape reader to continuously read at a rate of 10 char/sec until the reader is stopped or the tape runs out. The reader switch must be in the RUN position. The Buffer Ready flip-flop is reset.
SEL	:3C	(:403C)	Initialize the teletype interface. This command resets the control flip-flops, stops the oscillator and puts the interface in a static marking condition. The Buffer Ready flip-flop is set.
SEL	:3D	(:403D)	SET Word Xfer Mask. This command sets a mask flip-flop in the interface to enable an interrupt to be generated by Buffer Ready flip-flop. (The interrupt line is wired according to system requirements.)
SEL	:3E	(:403E)	SET Block Xfer Mask. This command sets a mask flip-flop in the interface to allow an interrupt to be generated when the Word Xfer Mask is in the off state. The interrupt can be used to indicate "End of Block."
SEL	:3F	(:403F)	RESET Masks. This instruction disables both interrupt lines in the teletype interface by resetting the mask flip-flops.
SEN	:39	(:4939)	Sense Buffer Ready. This instruction senses the On state of the Buffer Ready flip-flop, i.e., a true response will occur if the flip-flop is set.
SEN	:3A	(:493A)	Sense Word Xfer Mask Off. This instruction senses the Word Xfer Mask flip-flop and generates a true response if the flip-flop is in the off state.
SEN	:3B	(:493B)	Sense TTY not busy. This instruction senses the state of the TTY controller and generates a true response if the TTY is not printing or reading a character.
OTA OTX	:38 :38	(:6C38) (:6E38)	Output A or X Register to teletype. This instruction transfers the contents of the Register to the teletype interface and causes the character to be printed. If the punch is on, the character will also be punched.
IBA IBX	:38 :38	(:7838) (:7A38)	Input byte from teletype to the A or X Register. The character in the teletype interface buffer is transferred to the A or X Register.
RBA RBX	:39 :39	(:7939) (:7B39)	Read Byte from teletype to the A or X Register on sense response. This instruction senses the Buffer Ready flip flop in the TTY Interface and inputs a bypte on a true response.

Table 4-1. TTY Interface Function Codes

Select Instructions:	
0	Enable Auto Echo
1	Keyboard
2	Step Read
3	Continuous Read
4	Initialize the Interface
5	Set Word Transfer Mask
6	Set Block Transfer Mask
7	Reset Masks (Word Transfer and Block Transfer)
Sense Instructions:	
1	Buffer Ready
2	Word Transfer Mask Off
3	TTY Not Busy

4.2.4.2 Automatic Instructions. Since the teletype is a byte oriented device, the Auto I/O Byte instructions will normally be most useful for automatic data transfers. These instructions automatically pack data two bytes per word in memory. The Automatic I/O Byte instructions associated with the teletype option are:

AIB	:38	(:5438)	Input Byte to Memory from TTY Interface. This instruction uncontionally reads the contents of the TTY Interface data buffer and stores the contents in the byte location in memory specified by the AIB instruction Address Pointer. This will normally be executed as an interrupt instruction.
AOB	:38	(:6438)	Output Byte From Memory to TTY Interface. This instruction unconditionally reads the byte location in memory specified by the AOB Address Pointer and outputs the contents to the TTY Interface data buffer. This instruction will normally be executed as an interrupt instruction.

4.2.4.3 Programming Examples. The following are typical examples of teletype subroutines. These examples are illustrations only. They are not necessarily the most efficient methods which may be used to handle a specific problem.

1. Data output under program control. The following is a portion of a routine to output data from a data buffer in memory to the TTY interface. This routine does not use interrupts.

	SBM		Set Byte Mode so that data will be automatically unpacked by the computer hardware.
	SEL	:3C	Initialize the TTY Interface.
LOOP	LDA	*DATA	Read a byte from the data buffer in memory and hold in the A Register.
	IMS	DATA	Increment the data buffer address pointer.
	SEN	:39	Sense the Buffer Ready flip flop.
	JMP	$-1	Jump hack to the Sense instruction if the interface is not ready to accept data.
	OTA	:38	Output the byte from the A Register to the interface data register when the interface data buffer is ready.

IMS	COUNT	The location COUNT contains the negative of the number of bytes to be removed. When COUNT goes to zero, the computer will skip the next instruction and exit from the subroutine.
JMP	LOOP	If COUNT ≠ 0, go back to the LDA instruction and repeat the loop.
(Next Instruction)		When COUNT goes to zero, the computer skips the JMP instruction and executes the instruction following JMP.

2. Data input under program control. The following is a portion of a routine to input data from the data buffer in the TTY Interface to the A Register and then to memory. Note the similarity between this routine and the output routine above.

	SBM		Set Byte Mode. The computer will pack data in memory.
	SEL	:3B	Start the paper tape reader in a continuous read mode and reset the Buffer Ready flip flop.
LOOP	SEN	:39	Sense the Buffer Ready flip flop. It will be set when a byte has been read from paper tape.
	JMP	$-1	Sense again if not set.
	INA	:38	Input the byte from the interface data buffer to the A Register and reset Buffer Ready.
	STA	*DATA	Store the data in the data buffer in memory.
	IMS	DATA	Increment the memory byte address so that the next byte will be stored at the next sequential location.
	IMS	COUNT	Increment COUNT. When COUNT goes to zero, the computer will exit from this routine.
	JMP	LOOP	If COUNT ≠ 0, go back to the SEN instruction.
	SEL	:3C	Initialize the TTY Interface to stop the paper tape reader once all data has been read.

4.2.5 Reserved Memory Locations

Two memory locations are reserved as interrupt locations to be used with the TTY Interface option. The locations used are determined by other option selections.

4.2.5.1 Standard Interrupt Locations. The following are the standard interrupt location assignments for the TTY Interface option:

:0002 Word Interrupt. TTY Interface interrupts to this location when the Word Transfer Mask is set, interrupts are enabled, and the Buffer Ready flip-flop is set.

:0006 End of Block Interrupt. TTY Interface interrupts to this location when the Block Transfer Mask is set, interrupts are enabled, and an Echo signal is received from the computer.

4.2.5.2 Alternate Interrupt Locations. A jumper option provides two alternate interrupt locations for use by the TTY Interface. Locations :0002 and :0006 are used by Interrupt Line 1 and Interrupt Line 2, therefore it may be desired to cause TTY Interrupts to go to these alternate locations:

:0022 Word Interrupt.

:0026 End of Block Interrupt.

4.2.5.3 Offset Interrupt Locations. A jumper option allows all interrupts to be moved out of Scratchpad and offset by :0100 locations. When this is done, the standard interrupt locations are offset to these locations:

:0102 Word Interrupt

:0106 End of Block Interrupt

The alternate interrupt locations are offset to these locations:

:0122 Word Interrupt.

:0126 End of Block Interrupt.

4.3 POWER FAIL/RESTART

4.3.1 General

The Power Fail/Restart (PFR) option allows the ALPHA 16 and NAKED MINI 16 computers to be operated from unreliable AC power sources. A low power condition or a temporary power outage will be detected in time to allow the operating program to prepare for the power loss. When power returns to normal, the computer is automatically restarted without loss of data or operating position. Thus, unattended operation is possible.

4.3.2 Operation

The PFR logic monitors the unregulated DC power supply voltages to detect low power conditions for its power down sequence, and to determine when power has been restored to an acceptable level for its power up sequence. The DC power supply must guarantee that the regulated DC supply voltages will remain within operating tolerances for a minimum of 2.0 milliseconds following the detection of a low power condition.

4.3.2.1 Power Down Sequence. When an imminent power failure is detected, a power fail interrupt is generated to the processor and a 0.9 millisecond Down Sequence is started. If the Power Fail Interrupt is enabled, the processor is interrupted to a reserved location in memory (location :001C). The processor will execute the instruction at that location. The interrupt instruction will normally be a Jump and Store (JST) to a power-down software routine. The software routine should be written to store all volatile registers and indicators in core memory so that information will not be lost when power is lost. The processor has 0.9 millisecond to complete the power-down routine once the PFR Down Sequence is started.

The power-down software routine will normally execute a Halt once all volatile data has been stored in core memory. The processor then waits for the PFR hardware to complete the Down Sequence. 0.9 millisecond after the Down Sequence is started, PFR disables memory by removing read/write current from the memory. This is done so that data in memory cannot be inadvertently destroyed when power is completely lost. PFR and the computer then wait for power to be restored.

4.3.2.2 Power Up Sequence. When PFR detects that power has been restored to an acceptable level, a Power Up sequence is started. PFR waits 100 milliseconds to insure that power is stabilized, and then re-enables memory. PFR then sets the Program Counter (P Register) in the computer to :0000, and generates a run signal to the computer. The computer then executes the instruction at location :0000.

The instruction at location :0000 will normally be a Jump (JMP) to a software routine to restore the contents of the volatile registers and indicators that were saved during the Down Sequence, and restart the program at the point where it was interrupted by the power failure.

Once an Up Sequence is started by PFR, power fail interrupts are disabled for 0.9 millisecond so that the Up

Sequence can be completed before another Down Sequence can be initiated. Therefore the power supply must guarantee reserve power sufficient to complete an Up Sequence followed by a Down Sequence. Another 200 microseconds are reserved for possible electronic component timing variations, thus the power supply must guarantee 2 milliseconds of operating power once a power failure is detected.

4.3.3 Interrupt Control

The enable and disable of power fail interrupts may be handled in one of two ways. They may be placed under the control of the normal EIN and DIN instructions, or they may be separated from these instructions and placed under separate enable and disable instructions. A hardware wiring option makes the selection.

4.3.3.1 EIN/DIN Control. When power fail interrupts are under EIN/DIN control the execution of the EIN instruction enables power fail interrupts, and the execution of the DIN instruction disables power fail interrupts. It is not necessary to execute any masking instructions, since the PFR option is not designed for interrupt masking.

4.3.3.2 PFE/PFD Control. When it is desired to separate power fail interrupts from EIN/DIN control, two new instructions are generated:

PFE	Power Fail Enable
PFD	Power Fail Disable

These instructions enable and disable power fail interrupts independently of the EIN and DIN instructions.

4.3.3.3 Enable Timing. Power fail interrupts are enabled 10 microseconds after the execution of the PFE instruction or the EIN instruction. This allows a power-up subroutine to enable all interrupts and exit before another power fail interrupt can be generated.

4.3.4 Programming Examples

The following is an example of a simple power fail subroutine. It saves program status and volatile registers when a power failure is detected. It restores the status and registers when power is restored and continues the interrupted program at the point where it was interrupted. More sophisticated routines which print out power fail messages upon restoration of power may be used in actual practice.

1. Interrupt locations contain the following:

:0000	JMP	UP	This is the Power Up restart location. It contains an unconditional Jump to the Power Up subroutine.
:001C	JST	DOWN	This is the Power Down interrupt location. It contains a Jump and Store to the Power Down subroutine. Using a JST automatically saves the contents of the Program Counter.

2. Subroutines for Power Down and Power Up may be written as follows:

DOWN	RES	1	Reserved location for storage of P Counter when JST instruction at the power fail interrupt location is executed.
	SIN	1	Inhibit Byte Mode if set.
	STA	ASAVE	Save the A Register contents.
	SIA		Read the computer status word to the A Register and turn off Byte Mode and OV.
	STA	STAT	Save the computer status word.
	STX	XSAVE	Save the X Register contents.
	HLT		Halt the computer and wait for power to be restored.

UP	LDX	XSAVE	The JMP instruction at the Power Up restart location enters here. This instruction restores the contents of the Register.
	LDA	STAT	Read the computer status word into the A Register from its temporary storage location.
	SOA		Restore the computer status. Restore OV status and Byte Mode status.
	SIN	1	Inhibit Byte Mode if it is set.
	LDA	ASAVE	Restore the contents of the A Register.
	PFE		Enable power fail interrupts (if they are outside EIN/DIN control).
	EIN		Enable all other interrupts.
	JMP	*DOWN	Restart the main program by doing an indirect Jump to the location specified by the saved contents of the P Counter.
ASAVE	RES	1	A Register save location.
XSAVE	RES	1	X Register save location.
STAT	RES	1	Status word save location.

4.3.5 Reserved Memory Locations

The Power Fail/Restart option requires two reserved memory locations: one for the power fail interrupt, and one for the power up restart. The power fail interrupt is a true interrupt. The power up restart, however, is not a true interrupt. It is a direct hardware reset of the P Register in the computer and does not use the interrupt structure of the computer.

4.3.5.1 Power Fail Interrupt Location. Since the power fail interrupt is a true interrupt, it has a standard interrupt location which may be offset by the jumper option which offsets all standard interrupt locations. The standard interrupt location and offset location are:

Standard location: :001C

Offset location: :011C

4.3.5.2 Power Up Restart Location. Since the power up restart operation does not operate through the interrupt structure of the computer, the restart location cannot be offset by the interrupt offset jumper option. The restart location is:

Standard location: :0000

Offset location: :0000

4.4 REAL-TIME CLOCK

4.4.1 General

The Real-Time Clock (RTC) option provides a means for determining elapsed time and/or creating a time-of-day clock with software. The RTC keeps time by counting electrical pulses of known frequency, such as the output of a crystal oscillator or the input frequency of an AC power source.

4.4.2 Clock Sources

A number of different sources are available for use as RTC timing pulses. The standard configuration uses a 1 MHz crystal oscillator as the basic timing source. The 1 MHz clock is applied to a decade counter to produce 10 KHz, 1 KHz, and 100 Hz clock sources. These sources produce timing increments of 100 microseconds, 1 millisecond, and 10 milliseconds. The desired clock source to be used with the RTC option is selected by a jumper wire.

An external timing source may be applied to the RTC option if some source other than the crystal oscillator is desired. This allows the use of almost any timing period that may be desired.

4.4.3 Operation

The RTC provides timing signals to the computer each time a timing pulse from the clock source is detected, and a sync pulse when a specified elapsed time has expired. The RTC uses two interrupts to perform its functions.

4.4.3.1 Time Interrupt. If RTC interrupts are enabled, the RTC generates a time interrupt to the computer each time a clock pulse is detected from the clock source. This interrupt is usually serviced by an IMS instruction at the interrupt location. The interrupt instruction in this case is

```
IMS        COUNT
```

where COUNT is a memory word used to maintain a count of the number of Time interrupts received from the RTC. If COUNT goes to zero when it is incremented, an Echo is sent to the RTC (whenever IMS is used as an interrupt instruction an Echo is sent to the interrupting device when the location being incremented goes to zero.) This operation allows COUNT to be set to some negative value so that an Echo will be generated to the RTC after some specific period of time has elapsed.

4.4.3.2 Sync Interrupt. If Sync interrupts are enabled, the RTC generates a Sync interrupt to the computer whenever an Echo is received from the computer. Since an Echo is sent to the RTC when COUNT goes to zero, the Sync interrupt normally signals that some specified time interval has elapsed. The Sync interrupt is normally serviced by an interrupt subroutine.

4.4.3.3 Timekeeping Example. The Time interrupt is normally used to increment a memory location which is being used as a computer clock. The Sync interrupt is used to flag the main program when some specified time period has elapsed. For example, assume that the 10 millisecond clock is being used as a clock source. Assume also that some external device must be sampled by the main program once each second. The main program could set COUNT to -100 and enable Time and Sync interrupts. The Time interrupt could then be serviced by

```
IMS        COUNT
```

so that COUNT would go to zero after being incremented 100 times. A Sync interrupt would be generated when COUNT goes to zero, telling the main program that one second has elapsed. The main program could service the Sync interrupt by jumping to a subroutine that resets COUNT to -100 and samples the external device.

4.4.4 Control Instructions

The RTC is controlled through the I/O structure of the computer. It is assigned Device Address 8, and is controlled by I/O instructions. The control instructions used are:

SEL	:40	(:4040)	Enable RTC. Sets a mask flip flop in the RTC allowing Time and Sync interrupts to be generated (if Sync is armed).
SEL	:42	(:4042)	Arm Sync. Allows Sync interrupts to be generated if the RTC is enabled and an Echo is received.
SEL	:43	(:4043)	Clear RTC interrupts. Resets both Time and Sync interrupt requests. Does not disable or disarm interrupts, but instead removes interrupt request history from the RTC.
SEL	:44	(:4044)	Initialize RTC. Disarms, disables, and clears interrupt requests, preventing RTC interrupts and removing history.

SEL	:47	(:4047)	Disarm Sync. Prevents Sync interrupts from being generated without disabling Time interrupts.

4.4.5 Interrupt Locations

Since there are two interrupts associated with the RTC, two interrupt locations are required. Since they are true interrupts, they may be offset by the interrupt offset option.

4.4.5.1 Standard Locations. The standard interrupt locations for the Time and Sync interrupts are:

Time interrupt location: :0018

Sync interrupt location: :001A

4.4.5.2 Offset Locations. If the interrupt offset jumper option is used, the offset interrupt locations are

Time interrupt offset location: :0118

Sync interrupt offset location: :011A

4.5 AUTOLOAD

4.5.1 General

The Autoload (AL) option consists of a read only memory (ROM) preprogrammed with a binary loader and the necessary logic to cause that loader to be executed. Autoload uses the Power Up sequence logic of the Power Fail/ Restart option to initialize the computer and start the autoload sequence. Therefore, the PFR option is a prerequisite for the Autoload option.

The Autoload option is a multi-device loader which reads programs in standard binary format and stores them in the computer memory. Autoload may read in programs from a TTY paper tape reader, high speed paper tape reader, magnetic tape unit, cassette tape unit, or disk.

4.5.2 Operating Procedures

The Autoload sequence may be entered by depressing the AUTO LD switch on the operators panel with the STOP switch up and the machine not in RUN mode. (RUN indicator must be off. If the machine is running, the AUTO LD switch generates a console interrupt.) The device from which the load is to be performed is selected with the Data Entry switches. Detailed operating procedures are as follows:

1. Depress STOP — Depressing the STOP switch halts the computer.

2. RESET — Depress the RESET switch to initialize the computer logic.

3. Ready Device — Ready the input device from which the binary program is to be loaded. Ready the program in the device and place the device on line.

4. Select Device — Select the input device in the Data Switches on the operator's panel. Switch selections are:

TTY	All switches up.
High Speed Paper Tape	Switch 0 Down
Magnetic Tape	Switch 1 Down
Cassette Tape	Switch 2 Down
Disk	Switch 3 Down

5. STOP Up — Put the STOP switch up to take the machine out of Step mode and enable Run mode.

6. Depress AUTO LD — Depress the AUTO LD switch to start the Autoload sequence.

4.5.3 Operation

When the Autoload sequence is entered the Power Up sequence of the PFR option is entered to renerate a general reset to the computer, force the Program Counter to :0000, and generate a start pulse to the computer. This puts the computer in Run mode and causes it to address location :0000 for its first instruction.

4.5.3.1 Loader Execution. The ROM program parallels locations :0000 thru :007F. The Autoload logic causes all instruction cycles to fetch instructions from the ROM, and all data cycles to access core memory. Thus the load program in the ROM is executed, and the program being read from the peripheral device is treated as data which is stored in core memory.

4.5.3.2 Autoload Termination. In the standard Autoload option, the autoload sequence is terminated when the computer executes the instruction at location :005F in the ROM. The computer action at that point is a function of the program which was loaded. A special option allows the loader in the ROM to use all 128 words of ROM. In that case, the autoload sequence is terminated when the instruction at location :007F.

4.5.3.3 ROM Diagnostic. In machines with less than 32K words of core memory the program in the ROM may be read by reading memory locations :7000 and above, modulo 128. These locations must be read in Word mode rather than Byte mode. This allows diagnostic programs to read ROM and verify the program. It also allows special ROM's other than loader programs to be used. If the computer has 32K words of core memory, the software access of ROM must be disabled.

4.5.4 Reserved Memory Locations

There are no reserved memory locations associated with the Autoload option. During the execution of the autoload sequence the load program appears to reside in locations :0000 through :005F for instruction cycles. However, all of core memory, including locations :0000 through :005F, are available for data cycles.

4.6 MEMORY PROTECT

4.6.1 General

The Memory Protect (MP) option allows the user to protect the contents of a selected segment of core memory by preventing memory write operations in that segment. Any 2K, 4K, 8K, 16K, or all 32K, may be protected.

4.6.2 Operation

Segments of memory are selected for protection by removing jumper wires to decode the addresses which are to be protected. If all jumper wires are removed, all 32K words of memory are protected. The jumpers are part of a plug which connects to the back of one of the computer control boards. If the plug is inadvertently removed, all of the memory is protected. (Refer to the INSTALLATION PROCEDURES Appendix for jumper connections.)

4.6.2.1 Memory Protect Enable. The normal Memory Protect configuration is for Memory Protect to be operable at all times. An optional feature allows Memory Protect to be enabled and disabled by software.

4.6.2.2 Protect Operation. When Memory Protect is enabled the MP logic decodes all memory reference addresses and compares them with the protected addresses. If the memory location being accessed is outside the protected segment, or if the memory reference is a memory read operation rather than a write operation, MP does not interfere. However, if the memory location being accessed is within the protected segment, and the program is attempting to write in that location, the MP option sets a Write Disable latch which prevents altering the contents of memory. At the same time an interrupt is generated to flag the attempted violation of the protected area. At the end of the memory cycle the Write Disable latch is reset and the computer is returned to normal operation.

4.6.2.3 Interrupt Operation. MP interrupts are under the control of the EIN/DIN instructions. If interrupts are enabled, and MP has priority, the computer is interrupted to flag the attempted modification of protected memory. If interrupts are not enabled the protected segment will still be protected, but the attempted write violation flag may be lost.

4.6.3 Control Instructions

An optional feature allows memory protection to be placed under software control. If this feature is included, two control instructions become effective:

MPE	Memory Protect enable
MPD	Memory Protect disable.

When Memory Protect is enabled, the generation of MP interrupts is also enabled. The recognition of memory protect interrupts is under control of the EIN/DIN instructions. When Memory Protect is disabled, the normally protected areas of memory may be accessed just like the unprotected areas of memory.

4.6.4 Reserved Memory Locations

A single interrupt is associated with the Memory Protect option, thus one memory location must be reserved as an interrupt location. The location may be offset by the interrupt offset option.

Standard interrupt location:	:0014
Offset interrupt location:	:0114

Appendix A HEXIDECIMAL ARITHMETIC

NUMBERING SYSTEMS

Efficient and accurate communications with numbering systems between a computer console and operator, written page and reader, I/O devices and a computer are a necessity in computer systems.

Each of the many numbering systems in use today has its specific set of characters. The name of each numbering system describes the quantity of symbols used to define each discrete power (base) of the basic set of characters. As an example:

a. The binary numbering system is a base 2 system so that each character is 0 or 1.

b. The quinary numbering system is a base 5 system so that each character is 0, 1, 2, 3 or 4.

c. The octal numbering system is a base 8 system so that each character is 0, 1, 2, 3, 4, 5, 6 or 7.

d. The decimal numbering system is a base 10 system so that each character is 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9.

e. The hexidecimal numbering system is a base 16 system so that each character is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E or F.

A binary number with 24 characters is almost impossible to remember, or communicate in its original form. It is common practice therefore in the computer environment to convert such a binary number to an octal or hexidecimal numbering system when it is printed, spoken, or entered on a control panel. The purpose of the conversion is only to ease the communication problem between the binary displays or controls designed into the computers, and the people who must analyze and control the machines.

An example of a 24 bit binary number is given below with its octal and hexidecimal equivalents.

<u>Example</u>

```
0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 binary

011 101 011 001 101 111 011 101
 3   5   3   1   5   7   3   5      octal

0111 0101 1001 1011 1101 1101
  7    5    9    B    D    D        hexadecimal
```

It is exident in the example above that it takes 24 characters to represent the binary number, eight characters to represent the binary number in octal, and six characters to represent the same binary number in hexidecimal. Table A-1 identifies the binary, octal, and hexidecimal representation of a single character and shows the relationship between each.

Table A1. Binary, Octal, Hexidecimal Characters

Binary	Octal Equivalent	Binary	Hexidecimal Equivalent
000	0	0000	0
001	1	0001	1
010	2	0010	2
011	3	0011	3
100	4	0100	4
101	5	0101	5
110	6	0110	6
111	7	0111	7
		1000	8
		1001	9
		1010	A
		1011	B
		1100	C
		1101	D
		1110	E
		1111	F

HEXADECIMAL ARITHMETIC

Table A-2. ADDITION TABLE

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Table A-3. MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2B	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Table A-4. HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE

The table below provides for direct conversions between hexadecimal integers in the range 0–FFF and decimal integers in the range 0–4095. For conversion of larger integers, the table values may be added to the following figures:

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Hexadecimal fractions may be converted to decimal fractions as follows:

1. Express the hexadecimal fraction as an integer times 16^{-n}, where n is the number of significant hexadecimal places to the right of the hexadecimal point.

$$0.CA9BF3_{16} = CA9\ BF3_{16} \times 16^{-6}$$

2. Find the decimal equivalent of the hexadecimal integer

$$CA9\ BF3_{16} = 13\ 278\ 195_{10}$$

3. Multiply the decimal equivalent by 16^{-n}

$$\begin{array}{r} 13\ 278\ 195 \\ \times\ 596\ 046\ 448 \times 10^{-16} \\ \hline 0.791\ 442\ 096_{10} \end{array}$$

Decimal fractions may be converted to hexadecimal fractions by successively multiplying the decimal fraction by 16_{10}. After each multiplication, the integer portion is removed to form a hexadecimal fraction by building to the right of the hexadecimal point. However, since decimal arithmetic is used in this conversion, the integer portion of each product must be converted to hexadecimal numbers.

Example: Convert 0.895_{10} to its hexadecimal equivalent

```
           0.895
              16
      (14).320
              16
       (5).120
              16
       (1).920
              16
0.E51E₁₆ ← (14).720
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

Table A-4. HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

Table A-4. HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

Table A-4. HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

Table A-4. HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

Table A-4. HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

Table A-4. HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.40 00 00 00	.25000 00000	.80 00 00 00	.50000 00000	.C0 00 00 00	.75000 00000
.01 00 00 00	.00390 62500	.41 00 00 00	.25390 62500	.81 00 00 00	.50390 62500	.C1 00 00 00	.75390 62500
.02 00 00 00	.00781 25000	.42 00 00 00	.25781 25000	.82 00 00 00	.50781 25000	.C2 00 00 00	.75781 25000
.03 00 00 00	.01171 87500	.43 00 00 00	.26171 87500	.83 00 00 00	.51171 87500	.C3 00 00 00	.76171 87500
.04 00 00 00	.01562 50000	.44 00 00 00	.26562 50000	.84 00 00 00	.51562 50000	.C4 00 00 00	.76562 50000
.05 00 00 00	.01953 12500	.45 00 00 00	.26953 12500	.85 00 00 00	.51953 12500	.C5 00 00 00	.76953 12500
.06 00 00 00	.02343 75000	.46 00 00 00	.27343 75000	.86 00 00 00	.52343 75000	.C6 00 00 00	.77343 75000
.07 00 00 00	.02734 37500	.47 00 00 00	.27734 37500	.87 00 00 00	.52734 37500	.C7 00 00 00	.77734 37500
.08 00 00 00	.03125 00000	.48 00 00 00	.28125 00000	.88 00 00 00	.53125 00000	.C8 00 00 00	.78125 00000
.09 00 00 00	.03515 62500	.49 00 00 00	.28515 62500	.89 00 00 00	.53515 62500	.C9 00 00 00	.78515 62500
.0A 00 00 00	.03906 25000	.4A 00 00 00	.28906 25000	.8A 00 00 00	.53906 25000	.CA 00 00 00	.78906 25000
.0B 00 00 00	.04296 87500	.4B 00 00 00	.29296 87500	.8B 00 00 00	.54296 87500	.CB 00 00 00	.79296 87500
.0C 00 00 00	.04687 50000	.4C 00 00 00	.29687 50000	.8C 00 00 00	.54687 50000	.CC 00 00 00	.79687 50000
.0D 00 00 00	.05078 12500	.4D 00 00 00	.30078 12500	.8D 00 00 00	.55078 12500	.CD 00 00 00	.80078 12500
.0E 00 00 00	.05468 75000	.4E 00 00 00	.30468 75000	.8E 00 00 00	.55468 75000	.CE 00 00 00	.80468 75000
.0F 00 00 00	.05859 37500	.4F 00 00 00	.30859 37500	.8F 00 00 00	.55859 37500	.CF 00 00 00	.80859 37500
.10 00 00 00	.06250 00000	.50 00 00 00	.31250 00000	.90 00 00 00	.56250 00000	.D0 00 00 00	.81250 00000
.11 00 00 00	.06640 62500	.51 00 00 00	.31640 62500	.91 00 00 00	.56640 62500	.D1 00 00 00	.81640 62500
.12 00 00 00	.07031 25000	.52 00 00 00	.32031 25000	.92 00 00 00	.57031 25000	.D2 00 00 00	.82031 25000
.13 00 00 00	.07421 87500	.53 00 00 00	.32421 87500	.93 00 00 00	.57421 87500	.D3 00 00 00	.82421 87500
.14 00 00 00	.07812 50000	.54 00 00 00	.32812 50000	.94 00 00 00	.57812 50000	.D4 00 00 00	.82812 50000
.15 00 00 00	.08203 12500	.55 00 00 00	.33203 12500	.95 00 00 00	.58203 12500	.D5 00 00 00	.83203 12500
.16 00 00 00	.08593 75000	.56 00 00 00	.33593 75000	.96 00 00 00	.58593 75000	.D6 00 00 00	.83593 75000
.17 00 00 00	.08984 37500	.57 00 00 00	.33984 37500	.97 00 00 00	.58984 37500	.D7 00 00 00	.83984 37500
.18 00 00 00	.09375 00000	.58 00 00 00	.34375 00000	.98 00 00 00	.59375 00000	.D8 00 00 00	.84375 00000
.19 00 00 00	.09765 62500	.59 00 00 00	.34765 62500	.99 00 00 00	.59765 62500	.D9 00 00 00	.84765 62500
.1A 00 00 00	.10156 25000	.5A 00 00 00	.35156 25000	.9A 00 00 00	.60156 25000	.DA 00 00 00	.85156 25000
.1B 00 00 00	.10546 87500	.5B 00 00 00	.35546 87500	.9B 00 00 00	.60546 87500	.DB 00 00 00	.85546 87500
.1C 00 00 00	.10937 50000	.5C 00 00 00	.35937 50000	.9C 00 00 00	.60937 50000	.DC 00 00 00	.85937 50000
.1D 00 00 00	.11328 12500	.5D 00 00 00	.36328 12500	.9D 00 00 00	.61328 12500	.DD 00 00 00	.86328 12500
.1E 00 00 00	.11718 75000	.5E 00 00 00	.36718 75000	.9E 00 00 00	.61718 75000	.DE 00 00 00	.86718 75000
.1F 00 00 00	.12109 37500	.5F 00 00 00	.37109 37500	.9F 00 00 00	.62109 37500	.DF 00 00 00	.87109 37500
.20 00 00 00	.12500 00000	.60 00 00 00	.37500 00000	.A0 00 00 00	.62500 00000	.E0 00 00 00	.87500 00000
.21 00 00 00	.12890 62500	.61 00 00 00	.37890 62500	.A1 00 00 00	.62890 62500	.E1 00 00 00	.87890 62500
.22 00 00 00	.13281 25000	.62 00 00 00	.38281 25000	.A2 00 00 00	.63281 25000	.E2 00 00 00	.88281 25000
.23 00 00 00	.13671 87500	.63 00 00 00	.38671 87500	.A3 00 00 00	.63671 87500	.E3 00 00 00	.88671 87500
.24 00 00 00	.14062 50000	.64 00 00 00	.39062 50000	.A4 00 00 00	.64062 50000	.E4 00 00 00	.89062 50000
.25 00 00 00	.14453 12500	.65 00 00 00	.39453 12500	.A5 00 00 00	.64453 12500	.E5 00 00 00	.89453 12500
.26 00 00 00	.14843 75000	.66 00 00 00	.39843 75000	.A6 00 00 00	.64843 75000	.E6 00 00 00	.89843 75000
.27 00 00 00	.15234 37500	.67 00 00 00	.40234 37500	.A7 00 00 00	.65234 37500	.E7 00 00 00	.90234 37500
.28 00 00 00	.15625 00000	.68 00 00 00	.40625 00000	.A8 00 00 00	.65625 00000	.E8 00 00 00	.90625 00000
.29 00 00 00	.16015 62500	.69 00 00 00	.41015 62500	.A9 00 00 00	.66015 62500	.E9 00 00 00	.91015 62500
.2A 00 00 00	.16406 25000	.6A 00 00 00	.41406 25000	.AA 00 00 00	.66406 25000	.EA 00 00 00	.91406 25000
.2B 00 00 00	.16796 87500	.6B 00 00 00	.41796 87500	.AB 00 00 00	.66796 87500	.EB 00 00 00	.91796 87500
.2C 00 00 00	.17187 50000	.6C 00 00 00	.42187 50000	.AC 00 00 00	.67187 50000	.EC 00 00 00	.92187 50000
.2D 00 00 00	.17578 12500	.6D 00 00 00	.42578 12500	.AD 00 00 00	.67578 12500	.ED 00 00 00	.92578 12500
.2E 00 00 00	.17968 75000	.6E 00 00 00	.42968 75000	.AE 00 00 00	.67968 75000	.EE 00 00 00	.92968 75000
.2F 00 00 00	.18359 37500	.6F 00 00 00	.43359 37500	.AF 00 00 00	.68359 37500	.EF 00 00 00	.93359 37500
.30 00 00 00	.18750 00000	.70 00 00 00	.43750 00000	.B0 00 00 00	.68750 00000	.F0 00 00 00	.93750 00000
.31 00 00 00	.19140 62500	.71 00 00 00	.44140 62500	.B1 00 00 00	.69140 62500	.F1 00 00 00	.94140 62500
.32 00 00 00	.19531 25000	.72 00 00 00	.44531 25000	.B2 00 00 00	.69531 25000	.F2 00 00 00	.94531 25000
.33 00 00 00	.19921 87500	.73 00 00 00	.44921 87500	.B3 00 00 00	.69921 87500	.F3 00 00 00	.94921 87500
.34 00 00 00	.20312 50000	.74 00 00 00	.45312 50000	.B4 00 00 00	.70312 50000	.F4 00 00 00	.95312 50000
.35 00 00 00	.20703 12500	.75 00 00 00	.45703 12500	.B5 00 00 00	.70703 12500	.F5 00 00 00	.95703 12500
.36 00 00 00	.21093 75000	.76 00 00 00	.46093 75000	.B6 00 00 00	.71093 75000	.F6 00 00 00	.96093 75000
.37 00 00 00	.21484 37500	.77 00 00 00	.46484 37500	.B7 00 00 00	.71484 37500	.F7 00 00 00	.96484 37500
.38 00 00 00	.21875 00000	.78 00 00 00	.46875 00000	.B8 00 00 00	.71875 00000	.F8 00 00 00	.96875 00000
.39 00 00 00	.22265 62500	.79 00 00 00	.47265 62500	.B9 00 00 00	.72265 62500	.F9 00 00 00	.97265 62500
.3A 00 00 00	.22656 25000	.7A 00 00 00	.47656 25000	.BA 00 00 00	.72656 25000	.FA 00 00 00	.97656 25000
.3B 00 00 00	.23046 87500	.7B 00 00 00	.48046 87500	.BB 00 00 00	.73046 87500	.FB 00 00 00	.98046 87500
.3C 00 00 00	.23437 50000	.7C 00 00 00	.48437 50000	.BC 00 00 00	.73437 50000	.FC 00 00 00	.98437 50000
.3D 00 00 00	.23828 12500	.7D 00 00 00	.48828 12500	.BD 00 00 00	.73828 12500	.FD 00 00 00	.98828 12500
.3E 00 00 00	.24218 75000	.7E 00 00 00	.49218 75000	.BE 00 00 00	.74218 75000	.FE 00 00 00	.99218 75000
3F 00 00 00	.24609 37500	.7F 00 00 00	.49609 37500	.BF 00 00 00	.74609 37500	.FF 00 00 00	.99609 37500

Table A-4. HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 40 00 00	.00097 65625	.00 80 00 00	.00195 31250	.00 C0 00 00	.00292 96875
.00 01 00 00	.00001 52587	.00 41 00 00	.00099 18212	.00 81 00 00	.00196 83837	.00 C1 00 00	.00294 49462
.00 02 00 00	.00003 05175	.00 42 00 00	.00100 70800	.00 82 00 00	.00198 36425	.00 C2 00 00	.00296 02050
.00 03 00 00	.00004 57763	.00 43 00 00	.00102 23388	.00 83 00 00	.00199 89013	.00 C3 00 00	.00297 54638
.00 04 00 00	.00006 10351	.00 44 00 00	.00103 75976	.00 84 00 00	.00201 41601	.00 C4 00 00	.00299 07226
.00 05 00 00	.00007 62939	.00 45 00 00	.00105 28564	.00 85 00 00	.00202 94189	.00 C5 00 00	.00300 59814
.00 06 00 00	.00009 15527	.00 46 00 00	.00106 81152	.00 86 00 00	.00204 46777	.00 C6 00 00	.00302 12402
.00 07 00 00	.00010 68115	.00 47 00 00	.00108 33740	.00 87 00 00	.00205 99365	.00 C7 00 00	.00303 64990
.00 08 00 00	.00012 20703	.00 48 00 00	.00109 86328	.00 88 00 00	.00207 51953	.00 C8 00 00	.00305 17578
.00 09 00 00	.00013 73291	.00 49 00 00	.00111 38916	.00 89 00 00	.00209 04541	.00 C9 00 00	.00306 70166
.00 0A 00 00	.00015 25878	.00 4A 00 00	.00112 91503	.00 8A 00 00	.00210 57128	.00 CA 00 00	.00308 22753
.00 0B 00 00	.00016 78466	.00 4B 00 00	.00114 44091	.00 8B 00 00	.00212 09716	.00 CB 00 00	.00309 75341
.00 0C 00 00	.00018 31054	.00 4C 00 00	.00115 96679	.00 8C 00 00	.00213 62304	.00 CC 00 00	.00311 27929
.00 0D 00 00	.00019 83642	.00 4D 00 00	.00117 49267	.00 8D 00 00	.00215 14892	.00 CD 00 00	.00312 80517
.00 0E 00 00	.00021 36230	.00 4E 00 00	.00119 01855	.00 8E 00 00	.00216 67480	.00 CE 00 00	.00314 33105
.00 0F 00 00	.00022 88818	.00 4F 00 00	.00120 54443	.00 8F 00 00	.00218 20068	.00 CF 00 00	.00315 85693
.00 10 00 00	.00024 41406	.00 50 00 00	.00122 07031	.00 90 00 00	.00219 72656	.00 D0 00 00	.00317 38281
.00 11 00 00	.00025 93994	.00 51 00 00	.00123 59619	.00 91 00 00	.00221 25244	.00 D1 00 00	.00318 90869
.00 12 00 00	.00027 46582	.00 52 00 00	.00125 12207	.00 92 00 00	.00222 77832	.00 D2 00 00	.00320 43457
.00 13 00 00	.00028 99169	.00 53 00 00	.00126 64794	.00 93 00 00	.00224 30419	.00 D3 00 00	.00321 96044
.00 14 00 00	.00030 51757	.00 54 00 00	.00128 17382	.00 94 00 00	.00225 83007	.00 D4 00 00	.00323 48632
.00 15 00 00	.00032 04345	.00 55 00 00	.00129 69970	.00 95 00 00	.00227 35595	.00 D5 00 00	.00325 01220
.00 16 00 00	.00033 56933	.00 56 00 00	.00131 22558	.00 96 00 00	.00228 88183	.00 D6 00 00	.00326 53808
.00 17 00 00	.00035 09521	.00 57 00 00	.00132 75146	.00 97 00 00	.00230 40771	.00 D7 00 00	.00328 06396
.00 18 00 00	.00036 62109	.00 58 00 00	.00134 27734	.00 98 00 00	.00231 93359	.00 D8 00 00	.00329 58984
.00 19 00 00	.00038 14697	.00 59 00 00	.00135 80322	.00 99 00 00	.00233 45947	.00 D9 00 00	.00331 11572
.00 1A 00 00	.00039 67285	.00 5A 00 00	.00137 32910	.00 9A 00 00	.00234 98535	.00 DA 00 00	.00332 64160
.00 1B 00 00	.00041 19873	.00 5B 00 00	.00138 85498	.00 9B 00 00	.00236 51123	.00 DB 00 00	.00334 16748
.00 1C 00 00	.00042 72460	.00 5C 00 00	.00140 38085	.00 9C 00 00	.00238 03710	.00 DC 00 00	.00335 69335
.00 1D 00 00	.00044 25048	.00 5D 00 00	.00141 90673	.00 9D 00 00	.00239 56298	.00 DD 00 00	.00337 21923
.00 1E 00 00	.00045 77636	.00 5E 00 00	.00143 43261	.00 9E 00 00	.00241 08886	.00 DE 00 00	.00338 74511
.00 1F 00 00	.00047 30224	.00 5F 00 00	.00144 95849	.00 9F 00 00	.00242 61474	.00 DF 00 00	.00340 27099
.00 20 00 00	.00048 82812	.00 60 00 00	.00146 48437	.00 A0 00 00	.00244 14062	.00 E0 00 00	.00341 79687
.00 21 00 00	.00050 35400	.00 61 00 00	.00148 01025	.00 A1 00 00	.00245 66650	.00 E1 00 00	.00343 32275
.00 22 00 00	.00051 87988	.00 62 00 00	.00149 53613	.00 A2 00 00	.00247 19238	.00 E2 00 00	.00344 84863
.00 23 00 00	.00053 40576	.00 63 00 00	.00151 06201	.00 A3 00 00	.00248 71826	.00 E3 00 00	.00346 37451
.00 24 00 00	.00054 93164	.00 64 00 00	.00152 58789	.00 A4 00 00	.00250 24414	.00 E4 00 00	.00347 90039
.00 25 00 00	.00056 45751	.00 65 00 00	.00154 11376	.00 A5 00 00	.00251 77001	.00 E5 00 00	.00349 42626
.00 26 00 00	.00057 98339	.00 66 00 00	.00155 63964	.00 A6 00 00	.00253 29589	.00 E6 00 00	.00350 95214
.00 27 00 00	.00059 50927	.00 67 00 00	.00157 16552	.00 A7 00 00	.00254 82177	.00 E7 00 00	.00352 47802
.00 28 00 00	.00061 03515	.00 68 00 00	.00158 69140	.00 A8 00 00	.00256 34765	.00 E8 00 00	.00354 00390
.00 29 00 00	.00062 56103	.00 69 00 00	.00160 21728	.00 A9 00 00	.00257 87353	.00 E9 00 00	.00355 52978
.00 2A 00 00	.00064 08691	.00 6A 00 00	.00161 74316	.00 AA 00 00	.00259 39941	.00 EA 00 00	.00357 05566
.00 2B 00 00	.00065 61279	.00 6B 00 00	.00163 26904	.00 AB 00 00	.00260 92529	.00 EB 00 00	.00358 58154
.00 2C 00 00	.00067 13867	.00 6C 00 00	.00164 79492	.00 AC 00 00	.00262 45117	.00 EC 00 00	.00360 10742
.00 2D 00 00	.00068 66455	.00 6D 00 00	.00166 32080	.00 AD 00 00	.00263 97705	.00 ED 00 00	.00361 63330
.00 2E 00 00	.00070 19042	.00 6E 00 00	.00167 84667	.00 AE 00 00	.00265 50292	.00 EE 00 00	.00363 15917
.00 2F 00 00	.00071 71630	.00 6F 00 00	.00169 37255	.00 AF 00 00	.00267 02880	.00 EF 00 00	.00364 68505
.00 30 00 00	.00073 24218	.00 70 00 00	.00170 89843	.00 B0 00 00	.00268 55468	.00 F0 00 00	.00366 21093
.00 31 00 00	.00074 76806	.00 71 00 00	.00172 42431	.00 B1 00 00	.00270 08056	.00 F1 00 00	.00367 73681
.00 32 00 00	.00076 29394	.00 72 00 00	.00173 95019	.00 B2 00 00	.00271 60644	.00 F2 00 00	.00369 26269
.00 33 00 00	.00077 81982	.00 73 00 00	.00175 47607	.00 B3 00 00	.00273 13232	.00 F3 00 00	.00370 78857
.00 34 00 00	.00079 34570	.00 74 00 00	.00177 00195	.00 B4 00 00	.00274 65820	.00 F4 00 00	.00372 31445
.00 35 00 00	.00080 87158	.00 75 00 00	.00178 52783	.00 B5 00 00	.00276 18408	.00 F5 00 00	.00373 84033
.00 36 00 00	.00082 39746	.00 76 00 00	.00180 05371	.00 B6 00 00	.00277 70996	.00 F6 00 00	.00375 36621
.00 37 00 00	.00083 92333	.00 77 00 00	.00181 57958	.00 B7 00 00	.00279 23583	.00 F7 00 00	.00376 89208
.00 38 00 00	.00085 44921	.00 78 00 00	.00183 10546	.00 B8 00 00	.00280 76171	.00 F8 00 00	.00378 41796
.00 39 00 00	.00086 97509	.00 79 00 00	.00184 63134	.00 B9 00 00	.00282 28759	.00 F9 00 00	.00379 94384
.00 3A 00 00	.00088 50097	.00 7A 00 00	.00186 15722	.00 BA 00 00	.00283 81347	.00 FA 00 00	.00381 46972
.00 3B 00 00	.00090 02685	.00 7B 00 00	.00187 68310	.00 BB 00 00	.00285 33935	.00 FB 00 00	.00382 99560
.00 3C 00 00	.00091 55273	.00 7C 00 00	.00189 20898	.00 BC 00 00	.00286 86523	.00 FC 00 00	.00384 52148
.00 3D 00 00	.00093 07861	.00 7D 00 00	.00190 73486	.00 BD 00 00	.00288 39111	.00 FD 00 00	.00386 04736
.00 3E 00 00	.00094 60449	.00 7E 00 00	.00192 26074	.00 BE 00 00	.00289 91699	.00 FE 00 00	.00387 57324
.00 3F 00 00	.00096 13037	.00 7F 00 00	.00193 78662	.00 BF 00 00	.00291 44287	.00 FF 00 00	.00389 09912

Table A-5. HEXADECIMAL-DECIMAL FRACTION CONVERSION TABLE

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 40 00	.00000 38146	.00 00 80 00	.00000 76293	.00 00 C0 00	.00001 14440
.00 00 01 00	.00000 00596	.00 00 41 00	.00000 38743	.00 00 81 00	.00000 76889	.00 00 C1 00	.00001 15036
.00 00 02 00	.00000 01192	.00 00 42 00	.00000 39339	.00 00 82 00	.00000 77486	.00 00 C2 00	.00001 15633
.00 00 03 00	.00000 01788	.00 00 43 00	.00000 39935	.00 00 83 00	.00000 78082	.00 00 C3 00	.00001 16229
.00 00 04 00	.00000 02384	.00 00 44 00	.00000 40531	.00 00 84 00	.00000 78678	.00 00 C4 00	.00001 16825
.00 00 05 00	.00000 02980	.00 00 45 00	.00000 41127	.00 00 85 00	.00000 79274	.00 00 C5 00	.00001 17421
.00 00 06 00	.00000 03576	.00 00 46 00	.00000 41723	.00 00 86 00	.00000 79870	.00 00 C6 00	.00001 18017
.00 00 07 00	.00000 04172	.00 00 47 00	.00000 42319	.00 00 87 00	.00000 80466	.00 00 C7 00	.00001 18613
.00 00 08 00	.00000 04768	.00 00 48 00	.00000 42915	.00 00 88 00	.00000 81062	.00 00 C8 00	.00001 19209
.00 00 09 00	.00000 05364	.00 00 49 00	.00000 43511	.00 00 89 00	.00000 81658	.00 00 C9 00	.00001 19805
.00 00 0A 00	.00000 05960	.00 00 4A 00	.00000 44107	.00 00 8A 00	.00000 82254	.00 00 CA 00	.00001 20401
.00 00 0B 00	.00000 06556	.00 00 4B 00	.00000 44703	.00 00 8B 00	.00000 82850	.00 00 CB 00	.00001 20997
.00 00 0C 00	.00000 07152	.00 00 4C 00	.00000 45299	.00 00 8C 00	.00000 83446	.00 00 CC 00	.00001 21593
.00 00 0D 00	.00000 07748	.00 00 4D 00	.00000 45895	.00 00 8D 00	.00000 84042	.00 00 CD 00	.00001 22189
.00 00 0E 00	.00000 08344	.00 00 4E 00	.00000 46491	.00 00 8E 00	.00000 84638	.00 00 CE 00	.00001 22785
.00 00 0F 00	.00000 08940	.00 00 4F 00	.00000 47087	.00 00 8F 00	.00000 85234	.00 00 CF 00	.00001 23381
.00 00 10 00	.00000 09536	.00 00 50 00	.00000 47683	.00 00 90 00	.00000 85830	.00 00 D0 00	.00001 23977
.00 00 11 00	.00000 10132	.00 00 51 00	.00000 48279	.00 00 91 00	.00000 86426	.00 00 D1 00	.00001 24573
.00 00 12 00	.00000 10728	.00 00 52 00	.00000 48875	.00 00 92 00	.00000 87022	.00 00 D2 00	.00001 25169
.00 00 13 00	.00000 11324	.00 00 53 00	.00000 49471	.00 00 93 00	.00000 87618	.00 00 D3 00	.00001 25765
.00 00 14 00	.00000 11920	.00 00 54 00	.00000 50067	.00 00 94 00	.00000 88214	.00 00 D4 00	.00001 26361
.00 00 15 00	.00000 12516	.00 00 55 00	.00000 50663	.00 00 95 00	.00000 88810	.00 00 D5 00	.00001 26957
.00 00 16 00	.00000 13113	.00 00 56 00	.00000 51259	.00 00 96 00	.00000 89406	.00 00 D6 00	.00001 27553
.00 00 17 00	.00000 13709	.00 00 57 00	.00000 51856	.00 00 97 00	.00000 90003	.00 00 D7 00	.00001 28149
.00 00 18 00	.00000 14305	.00 00 58 00	.00000 52452	.00 00 98 00	.00000 90599	.00 00 D8 00	.00001 28746
.00 00 19 00	.00000 14901	.00 00 59 00	.00000 53048	.00 00 99 00	.00000 91195	.00 00 D9 00	.00001 29342
.00 00 1A 00	.00000 15497	.00 00 5A 00	.00000 53644	.00 00 9A 00	.00000 91791	.00 00 DA 00	.00001 29938
.00 00 1B 00	.00000 16093	.00 00 5B 00	.00000 54240	.00 00 9B 00	.00000 92387	.00 00 DB 00	.00001 30534
.00 00 1C 00	.00000 16689	.00 00 5C 00	.00000 54836	.00 00 9C 00	.00000 92983	.00 00 DC 00	.00001 31130
.00 00 1D 00	.00000 17285	.00 00 5D 00	.00000 55432	.00 00 9D 00	.00000 93579	.00 00 DD 00	.00001 31726
.00 00 1E 00	.00000 17881	.00 00 5E 00	.00000 56028	.00 00 9E 00	.00000 94175	.00 00 DE 00	.00001 32322
.00 00 1F 00	.00000 18477	.00 00 5F 00	.00000 56624	.00 00 9F 00	.00000 94771	.00 00 DF 00	.00001 32918
.00 00 20 00	.00000 19073	.00 00 60 00	.00000 57220	.00 00 A0 00	.00000 95367	.00 00 E0 00	.00001 33514
.00 00 21 00	.00000 19669	.00 00 61 00	.00000 57816	.00 00 A1 00	.00000 95963	.00 00 E1 00	.00001 34110
.00 00 22 00	.00000 20265	.00 00 62 00	.00000 58412	.00 00 A2 00	.00000 96559	.00 00 E2 00	.00001 34706
.00 00 23 00	.00000 20861	.00 00 63 00	.00000 59008	.00 00 A3 00	.00000 97155	.00 00 E3 00	.00001 35302
.00 00 24 00	.00000 21457	.00 00 64 00	.00000 59604	.00 00 A4 00	00000 97751	.00 00 E4 00	.00001 35898
.00 00 25 00	.00000 22053	.00 00 65 00	.00000 60200	.00 00 A5 00	.00000 98347	.00 00 E5 00	.00001 36494
.00 00 26 00	.00000 22649	.00 00 66 00	.00000 60796	.00 00 A6 00	.00000 98943	.00 00 E6 00	.00001 37090
.00 00 27 00	.00000 23245	.00 00 67 00	.00000 61392	.00 00 A7 00	.00000 99539	.00 00 E7 00	.00001 37686
.00 00 28 00	.00000 23841	.00 00 68 00	.00000 61988	.00 00 A8 00	.00001 00135	.00 00 E8 00	.00001 38282
.00 00 29 00	.00000 24437	.00 00 69 00	.00000 62584	.00 00 A9 00	.00001 00731	.00 00 E9 00	.00001 38878
.00 00 2A 00	.00000 25033	.00 00 6A 00	.00000 63180	.00 00 AA 00	.00001 01327	.00 00 EA 00	.00001 39474
.00 00 2B 00	.00000 25629	.00 00 6B 00	.00000 63776	.00 00 AB 00	.00001 01923	.00 00 EB 00	.00001 40070
.00 00 2C 00	.00000 26226	.00 00 6C 00	.00000 64373	.00 00 AC 00	.00001 02519	.00 00 EC 00	.00001 40666
.00 00 2D 00	.00000 26822	.00 00 6D 00	.00000 64969	.00 00 AD 00	.00001 03116	.00 00 ED 00	.00001 41263
.00 00 2E 00	.00000 27418	.00 00 6E 00	.00000 65565	.00 00 AE 00	.00001 03712	.00 00 EE 00	.00001 41859
.00 00 2F 00	.00000 28014	.00 00 6F 00	.00000 66161	.00 00 AF 00	.00001 04308	.00 00 EF 00	.00001 42455
.00 00 30 00	.00000 28610	.00 00 70 00	.00000 66757	.00 00 B0 00	.00001 04904	.00 00 F0 00	.00001 43051
.00 00 31 00	.00000 29206	.00 00 71 00	.00000 67353	.00 00 B1 00	.00001 05500	.00 00 F1 00	.00001 43647
.00 00 32 00	.00000 29802	.00 00 72 00	.00000 67949	.00 00 B2 00	.00001 06096	.00 00 F2 00	.00001 44243
.00 00 33 00	.00000 30398	.00 00 73 00	.00000 68545	.00 00 B3 00	.00001 06692	.00 00 F3 00	.00001 44839
.00 00 34 00	.00000 30994	.00 00 74 00	.00000 69141	.00 00 B4 00	.00001 07288	.00 00 F4 00	.00001 45435
.00 00 35 00	.00000 31590	.00 00 75 00	.00000 69737	.00 00 B5 00	.00001 07884	.00 00 F5 00	.00001 46031
.00 00 36 00	.00000 32186	.00 00 76 00	.00000 70333	.00 00 B6 00	.00001 08480	.00 00 F6 00	.00001 46627
.00 00 37 00	.00000 32782	.00 00 77 00	.00000 70929	.00 00 B7 00	.00001 09076	.00 00 F7 00	.00001 47223
.00 00 38 00	.00000 33378	.00 00 78 00	.00000 71525	.00 00 B8 00	.00001 09672	.00 00 F8 00	.00001 47819
.00 00 39 00	.00000 33974	.00 00 79 00	.00000 72121	.00 00 B9 00	.00001 10268	.00 00 F9 00	.00001 48415
.00 00 3A 00	.00000 34570	.00 00 7A 00	.00000 72717	.00 00 BA 00	.00001 10864	.00 00 FA 00	.00001 49011
.00 00 3B 00	.00000 35166	.00 00 7B 00	.00000 73313	.00 00 BB 00	.00001 11460	.00 00 FB 00	.00001 49607
.00 00 3C 00	.00000 35762	.00 00 7C 00	.00000 73909	.00 00 BC 00	.00001 12056	.00 00 FC 00	.00001 50203
.00 00 3D 00	.00000 36358	.00 00 7D 00	.00000 74505	.00 00 BD 00	.00001 12652	.00 00 FD 00	.00001 50799
.00 00 3E 00	.00000 36954	.00 00 7E 00	.00000 75101	.00 00 BE 00	.00001 13248	.00 00 FE 00	.00001 51395
.00 00 3F 00	.00000 37550	.00 00 7F 00	.00000 75697	.00 00 BF 00	.00001 13844	.00 00 FF 00	.00001 51991

Table A-5. HEXADECIMAL-DECIMAL FRACTION CONVERSION TABLE

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 00 40	.00000 00149	.00 00 00 80	.00000 00298	.00 00 00 C0	.00000 00447
.00 00 00 01	.00000 00002	.00 00 00 41	.00000 00151	.00 00 00 81	.00000 00300	.00 00 00 C1	.00000 00449
.00 00 00 02	.00000 00004	.00 00 00 42	.00000 00153	.00 00 00 82	.00000 00302	.00 00 00 C2	.00000 00451
.00 00 00 03	.00000 00006	.00 00 00 43	.00000 00155	.00 00 00 83	.00000 00305	.00 00 00 C3	.00000 00454
.00 00 00 04	.00000 00009	.00 00 00 44	.00000 00158	.00 00 00 84	.00000 00307	.00 00 00 C4	.00000 00456
.00 00 00 05	.00000 00011	.00 00 00 45	.00000 00160	.00 00 00 85	.00000 00309	.00 00 00 C5	.00000 00458
.00 00 00 06	.00000 00013	.00 00 00 46	.00000 00162	.00 00 00 86	.00000 00311	.00 00 00 C6	.00000 00461
.00 00 00 07	.00000 00016	.00 00 00 47	.00000 00165	.00 00 00 87	.00000 00314	.00 00 00 C7	.00000 00463
.00 00 00 08	.00000 00018	.00 00 00 48	.00000 00167	.00 00 00 88	.00000 00316	.00 00 00 C8	.00000 00465
.00 00 00 09	.00000 00020	.00 00 00 49	.00000 00169	.00 00 00 89	.00000 00318	.00 00 00 C9	.00000 00467
.00 00 00 0A	.00000 00023	.00 00 00 4A	.00000 00172	.00 00 00 8A	.00000 00321	.00 00 00 CA	.00000 00470
.00 00 00 0B	.00000 00025	.00 00 00 4B	.00000 00174	.00 00 00 8B	.00000 00323	.00 00 00 CB	.00000 00472
.00 00 00 0C	.00000 00027	.00 00 00 4C	.00000 00176	.00 00 00 8C	.00000 00325	.00 00 00 CC	.00000 00474
.00 00 00 0D	.00000 00030	.00 00 00 4D	.00000 00179	.00 00 00 8D	.00000 00328	.00 00 00 CD	.00000 00477
.00 00 00 0E	.00000 00032	.00 00 00 4E	.00000 00181	.00 00 00 8E	.00000 00330	.00 00 00 CE	.00000 00479
.00 00 00 0F	.00000 00034	.00 00 00 4F	.00000 00183	.00 00 00 8F	.00000 00332	.00 00 00 CF	.00000 00481
.00 00 00 10	.00000 00037	.00 00 00 50	.00000 00186	.00 00 00 90	.00000 00335	.00 00 00 D0	.00000 00484
.00 00 00 11	.00000 00039	.00 00 00 51	.00000 00188	.00 00 00 91	.00000 00337	.00 00 00 D1	.00000 00486
.00 00 00 12	.00000 00041	.00 00 00 52	.00000 00190	.00 00 00 92	.00000 00339	.00 00 00 D2	.00000 00488
.00 00 00 13	.00000 00044	.00 00 00 53	.00000 00193	.00 00 00 93	.00000 00342	.00 00 00 D3	.00000 00491
.00 00 00 14	.00000 00046	.00 00 00 54	.00000 00195	.00 00 00 94	.00000 00344	.00 00 00 D4	.00000 00493
.00 00 00 15	.00000 00048	.00 00 00 55	.00000 00197	.00 00 00 95	.00000 00346	.00 00 00 D5	.00000 00495
.00 00 00 16	.00000 00051	.00 00 00 56	.00000 00200	.00 00 00 96	.00000 00349	.00 00 00 D6	.00000 00498
.00 00 00 17	.00000 00053	.00 00 00 57	.00000 00202	.00 00 00 97	.00000 00351	.00 00 00 D7	.00000 00500
.00 00 00 18	.00000 00055	.00 00 00 58	.00000 00204	.00 00 00 98	.00000 00353	.00 00 00 D8	.00000 00502
.00 00 00 19	.00000 00058	.00 00 00 59	.00000 00207	.00 00 00 99	.00000 00356	.00 00 00 D9	.00000 00505
.00 00 00 1A	.00000 00060	.00 00 00 5A	.00000 00209	.00 00 00 9A	.00000 00358	.00 00 00 DA	.00000 00507
.00 00 00 1B	.00000 00062	.00 00 00 5B	.00000 00211	.00 00 00 9B	.00000 00360	.00 00 00 DB	.00000 00509
.00 00 00 1C	.00000 00065	.00 00 00 5C	.00000 00214	.00 00 00 9C	.00000 00363	.00 00 00 DC	.00000 00512
.00 00 00 1D	.00000 00067	.00 00 00 5D	.00000 00216	.00 00 00 9D	.00000 00365	.00 00 00 DD	.00000 00514
.00 00 00 1E	.00000 00069	.00 00 00 5E	.00000 00218	.00 00 00 9E	.00000 00367	.00 00 00 DE	.00000 00516
.00 00 00 1F	.00000 00072	.00 00 00 5F	.00000 00221	.00 00 00 9F	.00000 00370	.00 00 00 DF	.00000 00519
.00 00 00 20	.00000 00074	.00 00 00 60	.00000 00223	.00 00 00 A0	.00000 00372	.00 00 00 E0	.00000 00521
.00 00 00 21	.00000 00076	.00 00 00 61	.00000 00225	.00 00 00 A1	.00000 00374	.00 00 00 E1	.00000 00523
.00 00 00 22	.00000 00079	.00 00 00 62	.00000 00228	.00 00 00 A2	.00000 00377	.00 00 00 E2	.00000 00526
.00 00 00 23	.00000 00081	.00 00 00 63	.00000 00230	.00 00 00 A3	.00000 00379	.00 00 00 E3	.00000 00528
.00 00 00 24	.00000 00083	.00 00 00 64	.00000 00232	.00 00 00 A4	.00000 00381	.00 00 00 E4	.00000 00530
.00 00 00 25	.00000 00086	.00 00 00 65	.00000 00235	.00 00 00 A5	.00000 00384	.00 00 00 E5	.00000 00533
.00 00 00 26	.00000 00088	.00 00 00 66	.00000 00237	.00 00 00 A6	.00000 00386	.00 00 00 E6	.00000 00535
.00 00 00 27	.00000 00090	.00 00 00 67	.00000 00239	.00 00 00 A7	.00000 00388	.00 00 00 E7	.00000 00537
.00 00 00 28	.00000 00093	.00 00 00 68	.00000 00242	.00 00 00 A8	.00000 00391	.00 00 00 E8	.00000 00540
.00 00 00 29	.00000 00095	.00 00 00 69	.00000 00244	.00 00 00 A9	.00000 00393	.00 00 00 E9	.00000 00542
.00 00 00 2A	.00000 00097	.00 00 00 6A	.00000 00246	.00 00 00 AA	.00000 00395	.00 00 00 EA	.00000 00544
.00 00 00 2B	.00000 00100	.00 00 00 6B	.00000 00249	.00 00 00 AB	.00000 00398	.00 00 00 EB	.00000 00547
.00 00 00 2C	.00000 00102	.00 00 00 6C	.00000 00251	.00 00 00 AC	.00000 00400	.00 00 00 EC	.00000 00549
.00 00 00 2D	.00000 00104	.00 00 00 6D	.00000 00253	.00 00 00 AD	.00000 00402	.00 00 00 ED	.00000 00551
.00 00 00 2E	.00000 00107	.00 00 00 6E	.00000 00256	.00 00 00 AE	.00000 00405	.00 00 00 EE	.00000 00554
.00 00 00 2F	.00000 00109	.00 00 00 6F	.00000 00258	.00 00 00 AF	.00000 00407	.00 00 00 EF	.00000 00556
.00 00 00 30	.00000 00111	.00 00 00 70	.00000 00260	.00 00 00 B0	.00000 00409	.00 00 00 F0	.00000 00558
.00 00 00 31	.00000 00114	.00 00 00 71	.00000 00263	.00 00 00 B1	.00000 00412	.00 00 00 F1	.00000 00561
.00 00 00 32	.00000 00116	.00 00 00 72	.00000 00265	.00 00 00 B2	.00000 00414	.00 00 00 F2	.00000 00563
.00 00 00 33	.00000 00118	.00 00 00 73	.00000 00267	.00 00 00 B3	.00000 00416	.00 00 00 F3	.00000 00565
.00 00 00 34	.00000 00121	.00 00 00 74	.00000 00270	.00 00 00 B4	.00000 00419	.00 00 00 F4	.00000 00568
.00 00 00 35	.00000 00123	.00 00 00 75	.00000 00272	.00 00 00 B5	.00000 00421	.00 00 00 F5	.00000 00570
.00 00 00 36	.00000 00125	.00 00 00 76	.00000 00274	.00 00 00 B6	.00000 00423	.00 00 00 F6	.00000 00572
.00 00 00 37	.00000 00128	.00 00 00 77	.00000 00277	.00 00 00 B7	.00000 00426	.00 00 00 F7	.00000 00575
.00 00 00 38	.00000 00130	.00 00 00 78	.00000 00279	.00 00 00 B8	.00000 00428	.00 00 00 F8	.00000 00577
.00 00 00 39	.00000 00132	.00 00 00 79	.00000 00281	.00 00 00 B9	.00000 00430	.00 00 00 F9	.00000 00579
.00 00 00 3A	.00000 00135	.00 00 00 7A	.00000 00284	.00 00 00 BA	.00000 00433	.00 00 00 FA	.00000 00582
.00 00 00 3B	.00000 00137	.00 00 00 7B	.00000 00286	.00 00 00 BB	.00000 00435	.00 00 00 FB	.00000 00584
.00 00 00 3C	.00000 00139	.00 00 00 7C	.00000 00288	.00 00 00 BC	.00000 00437	.00 00 00 FC	.00000 00586
.00 00 00 3D	.00000 00142	.00 00 00 7D	.00000 00291	.00 00 00 BD	.00000 00440	.00 00 00 FD	.00000 00589
.00 00 00 3E	.00000 00144	.00 00 00 7E	.00000 00293	.00 00 00 BE	.00000 00442	.00 00 00 FE	.00000 00591
.00 00 00 3F	.00000 00146	.00 00 00 7F	.00000 00295	.00 00 00 BF	.00000 00444	.00 00 00 FF	.00000 00593

Table A-6. MATHEMATICAL CONSTANTS

Constant	Decimal Value	Hexadecimal Value
π	3.14159 26535 89793	3.243F 6A89
π^{-1}	0.31830 98861 83790	0.517C C1B7
$\sqrt{\pi}$	1.77245 38509 05516	1.C5BF 891C
$\ln \pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.B7E1 5163
e^{-1}	0.36787 94411 71442	0.5E2D 58D9
$\sqrt{e}$	1.64872 12707 00128	1.A612 98E2
$\log_{10} e$	0.43429 44819 03252	0.6F2D EC55
$\log_2 e$	1.44269 50408 88963	1.7154 7653
γ	0.57721 56649 01533	0.93C4 67E4
$\ln \gamma$	-0.54953 93129 81645	-0.8CAE 9BC1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
$\ln 2$	0.69314 71805 59945	0.B172 17F8
$\log_{10} 2$	0.30102 99956 63981	0.4D10 4D42
$\sqrt{10}$	3.16227 76601 68379	3.298B 075C
$\ln 10$	2.30258 50929 94046	2.4D76 3777

Table A-7. ASC II TELETYPE CODES

Symbol	Hexadecimal Code	Symbol	Hexadecimal Code
@	C0	ƀ	A0
A	C1	!	A1
B	C2	"	A2
C	C3	#	A3
D	C4	$	A4
E	C5	%	A5
F	C6	&	A6
G	C7	'	A7
H	C8	(	A8
I	C9	)	A9
J	CA	*	AA
K	CB	+	AB
L	CC	,	AC
M	CD	-	AD
N	CE	.	AE
O	CF	/	AF
P	D0	0	B0
Q	D1	1	B1
R	D2	2	B2
S	D3	3	B3
T	D4	4	B4
U	D5	5	B5
V	D6	6	B6
W	D7	7	B7
X	D8	8	B8
Y	D9	9	B9
Z	DA	:	BA
[	DB	;	BB
\	DC	<	BC
]	DD	=	BD
↑	DE	>	BE
←	DF	?	BF
NULL	00	CR	8D
BELL	87	LF	8A
		RUBOUT	FF

Table A-8. TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625

APPENDIX B
INSTRUCTION SET BY CLASS

	Class	Number of Mnemonics	Number of Instructions	Format
1.	Memory Reference	15	15	15: 1 | 14–11: OP CODE | 10–9: M | 8: I | 7–0: D
2.	Memory Reference Immediate	8	8	15–11: 1 1 0 0 0 | 10–8: OP CODE | 7–0: D
3.	Conditional Jump	13	63	15–12: 0 0 1 0 | 11–7: MICROCODE | 6: ± | 5–0: D
4.	Shift	20	86	15–11: 0 0 0 1 0 | 10–3: MICROCODE | 2–0: K
5.	Register Change	33	186	15–11: 0 0 0 0 0 | 10–3: MICROCODE | 2–0: 0 0 0
6.	Control Instructions	23	23	15–0: VARIABLE FORMAT
7.	Input/Output	33	33	15–14: 0 1 | 13–11: OP CODE | 10–8: MICRO-CODE | 7–3: DEVICE ADDRESS | 2–0: FUNCTION CODE
		145	414	

Definition of Symbols

+	Addition
−	Subtraction
∧	Logical AND
∨	Inclusive OR
⊻	Exclusive OR
=	Equals
→	Transfer (a → b, a is transferred to b)
−	One's Complement: ($\bar{a}$)
-	Two's Complement: −(a)
( )	"Contents of" or "the number in"
>	Greater Than
<	Less Than
≥	Greater Than or Equal
≤	Less Than or Equal

D	Address portion of Memory Reference Instructions
Y	Any Effective Address
A	Accumulator Register
X	Index Register
P	Program Counter (Register)
OV	Overflow Flip-Flop
IOB	Input/Output Bus
AP	Address Pointer
BA	Base Address
BIS	Bit Store
WC	Word Count

1. MEMORY REFERENCE INSTRUCTIONS

STRUCTURE

D = Address Field (0 to 255)$_{10}$

I = Direct/Indirect Address Bit

M = Address Mode Code

15	14 13 12 11	10 9	8	7 6 5 4 3 2 1 0
1	OP CODE	M	I	D

ADDRESSING

M	I	Word Mode (Word Operand)	Byte Mode (Byte Operand)
00	0	Y = (D), Words :00 -:FF	Y = (D), Bytes :00-:FF
01	0	Y = (D) + (P) + 1	Y = (D) + (P) 1, Byte 0
10	0	Y = (D) + (X)	Y = (D) + (X)
11	0	Y = (P) - (D)	Y = (D) + (P) + 1, Byte 1
00	1	AP = (D), [AP = (AP)], Y = (AP)	AP = (D), Y = (AP)
01	1	AP = (D) + (P) + 1, [AP = (AP)], Y = (AP)	AP = (D) + (P) + 1, Y = (AP)
10	1	AP = (D), [AP = (AP)], Y = (AP) + (X)	AP = (D), Y = (AP) + (X)
11	1	AP = (P) - (D), [AP = (AP)], Y = (AP)	AP = (P) - (D), Y = (AP)

INSTRUCTIONS

Instruction codes are shown with:

M = 00

I = 0

D = :00

Hexa Code	Mnemonic	Function	Description	Cycles*
8800	ADD	(A) + (Y) → A	Add to A	2
9000	SUB	(A) – (Y) → A	Subtract from A	2
8000	AND	(A) ∧ (Y) → A	Logical and with A	2
9800	STA	(A) → Y	Store A	2
E800	STX	(X) → Y	Store X	2
B000	LDA	(Y) → A	Load A	2
E000	LDX	(Y) → X	Load X	2
A000	IOR	(Y) ∨ (A) → A	Inclusive or with A	2
A800	XOR	(Y) ⊻ (A) → A	Exclusive or with A	2
B800	EMA	(Y) → A, (A) → Y	Exchange Memory and A	2

D800	IMS	(Y) + 1 → Y
		If (Y) + 1≠0, (P) + 1 → P
		If (Y) + 1≠0, (P) + 2 → P
			Increment and Skip if Zero	2

*Each level of indirect addressing adds 1 cycle to total time

Hexa Code	Mnemonic	Function	Description	Cycles*
F000	JMP	Y → P	Jump Unconditional	1
F800	JST	(P) + 1 → Y	Jump and Store P	
		Y + 1 → P		
D000	CMS	If (A) < (Y), (P) + 1 → P	Compare and Skip	2
		If (A) > (Y), (P) + 2 → P		
		If (A) = (Y), (P) + 3 → P		
CD00	SCN		Scan Memory	2+(X)

If (A) = Any of List ((D) + (X)) to ((D) + 1)), then (P) + 2 → P

If (A) ≠ Any of List ((D) + (X)) to ((D) + 1)), then (P) + 1 → P

where (D) is table address - 1

also, if OV=1, upper 8 bits of A is compared (8 bits)

OV=0, full contents of A is compared (16 bits)

If (A) = Any of List then (X) Reg = Compared Address - Table Address

---

*Each level of indirect addressing adds 1 cycle to total time

2. MEMORY REFERENCE IMMEDIATES

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	OP CODE			D							

STRUCTURE

D = The Immediate Operand with Eight-Bit Precision

Opcodes, Eight As Follows:

INSTRUCTIONS

Code	Mnemonic	Function	Description	Cycles
C000	CAI	If D ≠ (A0-7), (P) + 2 → P If D = (A0-7), (P) + 1 → P	Compare A Immed.	1
C100	CXI	If D ≠ (X0-7), (P) + 2 → P If D = (X0-7), (P) + 1 → P	Compare X Immed.	1
C200	AXI	(X) + D → X	Add to X Immed.	1
C300	SXI	(X) – D → X	Subtract from X Immed.	1
C400	LXP	0 + D → X	Load X Positive Immed.	1
C500	LXM	0 – D → X	Load X Minus Immed.	1
C600	LAP	0 + D → A	Load A Positive Immed.	1
C700	LAM	0 – D → A	Load A Minus Immed.	1

3. CONDITIONAL JUMP INSTRUCTIONS

15	14	13	12	11 10 9 8 7	6	5 4 3 2 1 0
0	0	1	0	MICROCODE	±	D

STRUCTURE

D = Displacement Field (0 to 63)

R = Sign of Displacement (I)

μcode = Five Bits each of which may select a Jump Condition

G = Group Select Bit. If G = 0 the μcode is interpreted as an "OR" group condition. If G = 1 the μcode is interpreted as an "And" Group

ADDRESSING

If Selected Conditions = Machine Status

Then $[(P) + 1)] + (R,D) \rightarrow P$

If Selected Conditions $\neq$ Machine Status

Then $(P) + 1 \rightarrow P$

SINGLE μCODE "OR" (G=0) INSTRUCTIONS

Code	Mnemonic	Function	Description	Cycles
2080	JAM	If $(A) < 0$	Jump If A Minus	1
2100	JAZ	If $(A) = 0$,	Jump If A Zero	1
2180	JAL	If $(A) \leq 0$	Jump If A Less Than or Equal to Zero	1
2200	JOS	If $(OV) = 1, 0 \rightarrow OV$	Jump If Overflow Set	1
2400	JSR	If Sense Switch = 0	Jump If Sense Switch Reset	1
2800	JXZ	If $(X) = 0$	Jump If X Zero	1

SINGLE μCODE "AND" (G=1) INSTRUCTIONS

Code	Mnemonic	Function	Description	Cycles
3080	JAP	If $(A15) \geq 0$	Jump If A Positive	1
3100	JAN	If $(A) \neq 0$	Jump If A Not Zero	1
3180	JAG	If $(A) > 0$	Jump If A Greater Than Zero	1
3200	JOR	If $(OV) = 0$	Jump If Overflow Reset	1
3400	JSS	If Sense Switch = 1	Jump If Sense Switch Set	1
3800	JXN	If $(X) \neq 0$	Jump If X Not Zero	1

UNIVERSAL JUMP ON CONDITION

Mnemonic	Function	Description	Cycles
JOC	Specified by given μcode	Jump on Condition Specified	1

4. SHIFT INSTRUCTIONS

15	14	13	12	11	10–3	2–0
0	0	0	1	0	MICROCODE	K

SINGLE REGISTER SHIFTS

K = Shift Count. Shift Will Move 1 + K Bit Positions.

μcode = Shift Control Code Which Selects Source, Type of Shift, and Location of Results

INSTRUCTIONS

Code	Mnemonic	Function	Description	Cycles
1050	ALA	A15 Unchanged 0 → A0	Arithmetic Shift A Left	1 + 1/4 K
1028	ALX	X15 Unchanges 0 → X0	Arithmetic Shift X Left	1 + 1/4 K
10D0	ARA	A15 Unchanged A15 → A14	Arithmetic Shift A Right	1 + 1/4 K
10A8	ARX	X15 Unchanged X15 → X14	Arithmetic Shift X Right	1 + 1/4K
11D0	RRA	A0 → OV OV → A15	Rotate A Right With OV	1 + 1/4 K
11A8	RRX	X0 → OV OV → X15	Rotate X Right With OV	1 + 1/4 K
1150	RLA	A15 → OV OV → A0	Rotate A Left With OV	1 + 1/4 K
1128	RLX	X15 → OV OV → X0	Rotate X Left With OV	1 + 1/4 K
13D0	LRA	0 → A15	Logical Shift A Right	1 + 1/4 K
13A8	LRX	0 → X15	Logical Shift X Right	1 + 1/4 K
1350	LLA	0 → A0	Logical Shift A Left	1 + 1/4 K
1328	LLX	0 → X0	Logical Shift X Left	1 + 1/4 K
1228	NOR	Left Arithmetic Shift X Until X14 ≠ X15 Then 1 → OV and Stop Shift	Normalize X Register	1 + 1/4 K
1340	SAO	A_{15} → OV	Sign of A to OV	1

LONG SHIFTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	MICROCODE							K			

μcode = Shift Control Code Which Selects The Type of Long Shift to be Executed

K = Shift Count. Shift Will Move 1 + K Bit Positions.

Code	Mnemonic	Function	Description	Cycles
1980	LRR	A0 → X15	Long Rotate Right	1-1/4 + 1/2 K
		X0 → OV		
		OV → A15		
1900	LRL	A15 → OV	Long Rotate Left	1-1/4 + 1/2 K
		OV → X0		
		X15 → A0		
1B00	LLL	A15 → OV	Long Logical Shift Left	1-1/4 + 1/2 K
		X15 → A0		
		0 → X0		
1B80	LLR	0 → A15	Long Logical Shift Right	1-1/4 + 1/2 K
		A0 → X15		
		X00 → OV		
19A0	MPS		Multiply Step	1-1/4 + 1/2 K
1940	DVS		Divide Step	1-1/4 + 1/2 K

19A0 MPS — Multiply Step

Multiply Step Instruction Performs a Conditional (on OV = 1) Add with the A Register and the R Register and a Long Right Shift.

For Each Shift of 1 + K Shifts

1. If OV = 1; 0 → OV, (R) + (A) → A
2. And then Long Logical Shift Right One, (A15) $\veebar$ (OV) → A15

1940 DVS — Divide Step

Divide Step Instructional Performs A Special Long Left Shift With A Bit Store Flip-Flop (BIS) and OV

In Three Steps per Shift Count as follows:

1. X Logical Left Shift 1, X15 → BIS, $\overline{(OV) \veebar (R15)}$ → X0
2. If X0 = 1, (A) – (R) → A; If X0 = 0, (A) + (R) → A
3. A Logical Left Shift 1, BIS → A0, A15 → OV

5. REGISTER CHANGE INSTRUCTIONS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	MICROCODE								0	0	0

STRUCTURE

μcode = The Register Change Control Code which specifies the Source, Operation, and Location of Results

INSTRUCTIONS

Code	Mnemonic	Function	Description	Cycles
0008	XRM	-1 → X	Set X Register to -1	1
0010	ARM	-1 → A	Set A Register to -1	1
0018	AXM	-1 → X, -1 → A	Set A and X Registers to -1	1
0108	ZXR	0 → X	Zero X Register	1
0110	ZAR	0 → A	Zero A Register	1
0118	ZAX	0 → X, 0 → A	Zero A and X Registers	1
0528	XRP	+1 → X	Set X Register to +1	1
0350	ARP	+1 → A	Set A Register to +1	1
0358	AXP	+1 → A, +1 → X	Set A and X Registers to +1	1
00A8	DXR	(X) -1 → X	Decrement X	1
00D0	DAR	(A) -1 → A	Decrement A	1
0128	IXR	(A) +1 → X	Increment X	1
0150	IAR	(A) +1 → A	Increment A	1
0408	CXR	$\overline{(X)} \rightarrow X$	Complement X	1
0210	CAR	$\overline{(A)} \rightarrow A$	Complement A	1
0508	NXR	-(X) → X	Negate X	1
0310	NAR	-(A) → A	Negate A	1
0030	TXA	(X) → A	Transfer X to A	1
0048	TAX	(A) → X	Transfer A to X	1
0070	ANA	$(A) \wedge (X) \rightarrow A$	And of A And X to A	1
0068	ANX	$(A) \wedge (X) \rightarrow X$	And of A And X to X	1
0610	NRA	$[\overline{(A) \vee (X)}] \rightarrow A$	NOR of A And X to A	1
0608	NRX	$[\overline{(A) \vee (X)}] \rightarrow X$	NOR of A And X to X	1
00C8	DAX	(A) -1 → X	Decrement A And Put In X	1
00B0	DXA	(A) -1 → A	Decrement X And Put In A	1
0148	IAX	(A) +1 → X	Increment A And Put In X	1
0130	IXA	(X) +1 → A	Increment X And Put In A	1
0208	CAX	$\overline{(A)} \rightarrow X$	Complement A And Put In X	1

Code	Mnemonic	Function	Description	Cycles
0410	CXA	$(\overline{X}) \rightarrow A$	Complement X And Put In A	1
0308	NAX	$-(A) \rightarrow X$	Negate A And Put In X	1
0510	NXA	$-(X) \rightarrow A$	Negate X And Put In A	1
5801*	ISA	$DS_{0\text{-}3} \rightarrow A_{0\text{-}3}$	Data Switches 0-3 to A 0-3	1-1/4
5B01*	ISX	$DS_{0\text{-}3} \rightarrow X_{0\text{-}3}$	Data Switches 0-3 to X 0-3	1-1/4

*Special I/O codes used for register change.

6. CONTROL INSTRUCTIONS

STRUCTURE

No fixed format

INSTRUCTIONS

Code	Mnemonic	Function	Description	Cycles
0000	NOP	One Memory Cycle Pause	No Operation	1
1400	SOV	$1 \rightarrow OV$	Set Overflow	1
1200	ROV	$O \rightarrow OV$	Reset Overflow	1
1600	COV	$\overline{OV} \rightarrow OV$	Complement Overflow	1
0A00	EIN	Enable Interrupt Response	Enable Interrupts	1
0C00	DIN	Disable Interrupt Response	Disable Interrupts	1
0800	HLT	Halts Controller	Halt	1
0E00	SBM	Sets Byte Mode F/F	Enters Byte Operand Mode	1
0F00	SWM	Resets Byte Mode F/F	Enters Word Operand Mode	1
6800	SIN	Inhibit Byte Mode and Interrupt Enable	Status Inhibit	1-1/4
4007	TRP	Execute Interrupt	TRAP	1-1/4
5A00	SIX	Read Status Word to X	Status Input to X	1-1/4
5800	SIA	Read Status Word to A	Status Input to A	1-1/4
6E00	SOX	Restore Status From X	Status Output From X	1-1/4
6C00	SOA	Restore Status From A	Status Output From A	1-1/4
4005	CIE	Enable Console Interrupts	Console Interrupt Enable	1-1/4
4006	CID	Disable Console Interrupts	Console Interrupt Disable	1-1/4
4002	PFE	Set Power Fail Interrupt Mask	Power Fail Interrupt Enable	1-1/4
4003	PFD	Reset Power Fail Interrupt Mask	Power Fail Interrupt Disable	1-1/4
4000	MPE	Protect Memory Contents	Memory Protect Enable	1-1/4
4001	MPD	Remove Memory Protection	Memory Protect Disable	1-1/4
4045	RAM	Select Read/Write Memory	Set Random Access Mode	1-1/4
4046	ROM	Select Read Only Memory	Set Read Only Mode	1-1/4

7. INPUT/OUTPUT INSTRUCTIONS

15	14	13 12 11	10 9 8	7 6 5 4 3	2 1 0
0	1	OP CODE	MICROCODE	DEVICE ADDRESS	FUNCTION CODE

ONE WORD INSTRUCTIONS

Function Code = The Order to the Selected Device

Device Address = The Number to Which the Device Will Respond

Opcode = Operation Code Specifying One of the Instructions Listed

Code	Mnemonic	Function	Description	Cycles
4900	SEN	If Sense Response, (P) + 2 → P If No Sense Response, (P) + 1 → P	Sense And Skip On Response	1-1/4
4800	SSN	If Sense Response, (P) + 1 → P If No Sense Response, (P) + 2 → P	Sense And Skip On No Response	1-1/4
4000	SEL	Device Address And Function Code to Control Bus	Select Function	1-1/4
4400	SEA	Same as SEL, With (A) to Data Bus	Select And Present A	1-1/4
4600	SEX	Same as SEL, With (X) to Data Bus	Select And Present X	1-1/4
5800	INA	(IOB) → A	Input To A Register	1-1/4
5A00	INX	(IOB) → X	Input To X Register	1-1/4
5C00	INAM	(IOB) ∧ (A) → A	Masked Input To A Register	1-1/4
5E00	INXM	(IOB) ∧ (X) → X	Masked Input To X Register	1-1/4
5900	RDA	If Sense Response, (IOB) → A, (P) + 1 → P If No Sense Response, (P) → P	Read Word To A Register	1-1/4
5B00	RDX	If Sense Response, (IOB) → X, (P) + 1 → P If No Sense Response, (P) → P	Read Word To X Register	1-1/4
5D00	RDAM	If Sense Response, (IOB) ∧ (A) → A (P) + 1 → P	Read Word To A Register Masked	1-1/4
5F00	RDXM	If Sense Response, (IOB ∧ (X) → X, (P) + 1 → P If No Sense Response, (P) → P	Read Word To X Register Masked	1-1/4
7800	IBA	(IOB0-7) → A0-7	Input Byte To A Register	1-1/4
7A00	IBX	(IOB0-7) → X0-7	Input Byte To X Register	1-1/4
7C00	IBAM	(IOB0-7) ∧ (A0-7) → A0-7	Input Byte To A Register Masked	1-1/4
7E00	IBXM	(IOB0-7) ∧ (X0-7) → X0-7	Input Byte To X Register Masked	1-1/4

Code	Mnemonic	Function	Description	Cycles
7900	RBA	If Sense Response, (IOB0-7) → A, (P) + 1 → P If No Sense Response, (P) → P	Read Byte To A Register	1-1/4
7B00	RBX	If Sense Response, (IOB0-7) → X, (P) + 1 → P If No Sense Response, (P) → P	Read Byte To X Register	1-1/4
7D00	RBAM	If Sense Response, (IOB0-7) ∧ (A0-7) → A, (P) + 1 → 1 If No Sense Response (P) → P	Read Byte To A Register Masked	1-1/4
7F00	RBXM	If Sense Response, (IOB0-7) ∧ (X0-7) → X, (P) + 1 → P If No Sense Response (P) → P	Read Byte to X Register Masked	1-1/4
6C00	OTA	(A) → IOB	Output A Register	1-1/4
6E00	OTX	(X) → IOB	Output X Register	1-1/4
6800	OTZ	O → IOB	Output Zero	1-1/4
6D00	WRA	If Sense Response, (A) → IOB, (P) + 1 → P If No Sense Response (P) → P	Write From A Register	1-1/4
6F00	WRX	If Sense Response, (X) → IOB, (P) + 1 → P If No Sense Response (P) → P	Write From X Register	1-1/4
6900	WRZ	If Sense Response, O → IOB, (P) + 1 → P If No Sense Response, (P) → P	Write Zeros	1-1/4

AUTOMATIC I/O INSTRUCTIONS

<table>
<tr><td></td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>P</td><td>0</td><td>1</td><td colspan="2">OP CODE</td><td>0</td><td>B</td><td>0</td><td>0</td><td colspan="5">DEVICE ADDRESS</td><td colspan="3">FUNCTION CODE</td></tr>
<tr><td>P+1</td><td>1</td><td colspan="15">BYTE/WORD COUNTER, WC (2'S COMPLEMENT)</td></tr>
<tr><td>P+2</td><td>0</td><td colspan="15">ADDRESS POINTER, AP (START LOCATION -1)</td></tr>
</table>

Opcode; 01 = Input, 10 = Output

Byte/Word Counter = Number of Executions Until Skip or Echo

Address Pointer = Memory Location of I/O Transaction

B = 0: Word Transfer

B = 1: Byte Transfer

These Instructions may be executed in In-Line Code or as single execute interrupt instructions. In either case, the execution takes three cycles in addition to the basic instruction cycle.

Hexa Code	Mnemonic	Function	Description	Cycles
	AIN, AIB		Automatic Input	4-1/4

5000 5400	A.	In Line Code
		1. (WC) + 1 → WC
		If (WC) + 1 = O, (P) + 3 → P
		If (WC) + 1 ≠ O, (P) + 4 → P
		2. (AP) + 1 → AP, Memory Address Register
		3. (IOB) → (Y) = Where Y = Memory Address of (2) above.		

	B.	Single Execute Interrupt Instruction 1. (WC) + 1 → WC If (WC) + 1 = O, Echo 2. (AP) + 1 → AP, Memory Address Register 3. (IOB) → (Y) = Where Y = Memory Address of (2) above.		4-1/4
	AOT, AOB		Automatic Output	4-1/4

6000, 6400	A.	In Line Code
		1. (WC) + 1 → WC
		If (WC) + 1 = O, (P) + 3 → P
		If (WC) + 1 ≠ O, (P) + 4 → P
		2. (A) + 1 → AP, Memory Address Register
		3. (Y) → IOB Where Y = Memory Address of (2)		

	B.	Single Execute Interrupt Instruction 1. (WC) + 1 → WC If (WC) + 1 = O, Echo 2. (A) + 1 → AP, Memory Address Register 3. (Y) → IOB Where Y = Memory Address of (2)		4-1/4

BLOCK I/O INSTRUCTIONS

	15	14	13 12 11 10 9 8	7 6 5 4 3	2 1 0
P	0	1	OP CODE	DEVICE ADDRESS	FUNCTION CODE
P+1	BASE ADDRESS				

Opcode, Two Codes For Input or Output

Hexa Code	Mnemonic		Description
7100	BIN		Input Block

A. The X Register Must Be Preset With the Number of Words to be Transferred

B. The Instruction is Fetched from Memory Location Y = (P)

C. The Base Address is Fetched from Memory
Location Y = (P) + 1

D. The Following Word Transfer Cycle is
Executed (X) Times

If Sense Response, (IOB) → Y = BA + (X)
(X) – 1 → X
If (X) – 1 = 0, (P) + 2 → P
If (X) – 1 ≠ 0, Get Next Word
If No Sense Response (X) → X, Repeat Sense

7500 BOT Output Block

A. Same as for BIN

B. Same as for BIN

C. Same as for BIN

D. Same as for BIN Except
If Sense Response, (Y) → IOB

APPENDIX C

LOGICAL FUNCTION DESCRIPTIONS

The following examples while only 4 bits in length are applicable to any length of binary bit strings.

(A) = A Register
(M) = Memory Word Content
(X) = X Register

(IOR) Inclusive OR

(A)	0101
(M)	0110
Result in (A)	0111

(XOR) Exclusive OR

(A)	0011
(M)	0101
Result in (A)	0110

(AND)

(A)	0101
(M)	0011
Result in (A)	0001

(NRA) Nor of A and X to A register

(A)	0101
(X)	0100
Result in (A)	1010

MUL*

	1010 $(10)_{10}$
	0111 $(7)_{10}$
Sum	1010
Shift & Add	1010
Sum	11110
Shift & Add	1010
Sum	1000110
Shift & Add	0000
Product	1000110 $(70)_{10}$

(SUB)

(A)	0101
(M)	0011
Result in (A)	0010

(ADD)

(A)	0001
(M)	0011
Result in A	0100

(CAR) (1's) Compliment A Register.

(A)	0010
Result in (A)	1101

(NAR) (2's) Negative A Register.

(A)	0010
Result in (A)	1100

DIV.*

	1010 $(10)_{10}$
0111	1000110 $(70)_{10}$
Try Yes Subtract	0111
Result & Next	00011
Try No Subtract	0000
Result & Next	0111
Try Yes Subtact	0111
Result & Next	00000
Try No Subtract	0000

*The MUL & DIV. do not represent actual machine mechanization.

APPENDIX D

INSTRUCTION SET, ALPHABETICAL ORDER

Instructions are listed in alphabetical order by instruction mnemonic in this section.

*Instructions having an asterisk (*) following the instruction hexadecimal code are shown with variable fields containing all zeros. Refer to the instruction descriptions in Section 2 and Section 3 for the definitions of the variable fields. Instructions which do not have an asterisk following the instruction code do not have variable fields and the code listed is the only code that defines the instruction.

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
ADD	8800*	Add to A direct, scratchpad	2	2-10
ADD	8900*	Add to A indirect, AP in scratchpad	2+1n	2-10
ADD	8A00*	Add to A relative to P forward, direct	2	2-10
ADD	8B00*	Add to A relative to P forward, indirect	2+1n	2-10
ADD	8C00*	Add to A indexed, direct	2	2-10
ADD	8D00*	Add to A indexed, indirect	2+1n	2-10
ADD	8E00*	Add to A relative to P backward, direct	2	2-10
ADD	8F00*	Add to A relative to P backward, indirect	2+1n	2-10
ADDB	:8800*	Add Byte, direct, scratchpad	2	2-24
ADDB	:8900*	Add Byte, indirect, AP in scratchpad	3	2-24
ADDB	:8A00*	Add Byte 0, relative to P forward, direct	2	2-24
ADDB	:8B00*	Add Byte, indirect, AP relative to P, forward	3	2-24
ADDB	:8C00*	Add Byte, direct, indexed	2	2-24
ADDB	:8D00*	Add Byte, indirect, indexed, AP in scratchpad	3	2-24
ADDB	:8E00*	Add Byte 1, relative to P forward, direct	2	2-24
ADDB	:8F00*	Add Byte, indirect, relative to P, backward	3	2-24
AIB	5400*	Automatic Input: Byte	4-1/4	3-31
AIN	5000*	Automatic Input: Word	4-1/4	3-30
ALA	1050*	Arithmetic shift A left	1+1/4K	2-45
ALX	1028*	Arithmetic shift X left	1+1/4K	2-45
ANA	0070	AND of A and X to A	1	2-65
AND	8000*	AND to A direct, scratchpad	2	2-14
AND	8100*	AND to A indirect, AP in scratchpad	2+1n	2-14
AND	8200*	AND to A relative to P forward, direct	2	2-14
AND	8300*	AND to A relative to P forward, indirect	2+1n	2-14
AND	8400*	AND to A indexed, direct	2	2-14
AND	8500*	AND to A indexed, indirect	2+1n	2-14
AND	8600*	AND to A relative to P backward, direct	2	2-14
AND	8700*	AND to A relative to P backward, indirect	2+1n	2-14

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
ANDB	:8000*	AND to A Byte, direct, scratchpad	2	2-28
ANDB	:8100*	AND to A Byte, indirect, AP in scratchpad	3	2-28
ANDB	:8200*	AND to A Byte 0, direct, relative to P forward	2	2-28
ANDB	:8300*	AND to A Byte, indirect, AP relative to P forward	3	2-28
ANDB	:8400*	AND to A Byte, indexed, direct	2	2-28
ANDB	:8500*	AND to A Byte, indexed, indirect, AP in scratchpad	3	2-28
ANDB	:8600*	AND to A Byte 1, direct, relative to P forward	2	2-28
ANDB	:8700*	AND to A Byte, indirect, AP relative to P backward	3	2-28
ANX	0068	AND of A and X to X	1	2-65
AOB	6400*	Automatic Output: Byte	4-1/4	3-31
AOT	6000*	Automatic Output: Word	4-1/4	3-30
ARA	10D0*	Arithmetic shift A right	1+1/4K	2-44
ARM	0010	Set A to minus 1	1	2-59
ARP	0350	Set A to plus 1	1	2-59
ARX	10A8*	Arithmetic shift X right	1+1/4K	2-45
AXI	C200*	Add to X immediate	1	2-31
AXM	0018	Set A and X to minus 1	1	2-59
AXP	0358	Set A and X to plus 1	1	2-60
BIN	7100*	Block input to memory	2+1-1/2w	3-25
BOT	7500*	Block output from memory	2+1-1/2w	3-26
CAI	C000*	Compare to A immediate	1	2-33
CAR	0210	Complement A	1	2-62
CAX	0208	Complement A and X	1	2-63
CID	4006	Console interrupt disable	1-1/4	2-72
CIE	4005	Console interrupt enable	1-1/4	2-71
CMS	D000*	Compare memory to A and skip if high or equal; direct, scratchpad	2	2-15
CMS	D100*	Compare memory to A and skip if high or equal; indirect, AP in scratchpad	2+1n	2-15
CMS	D200*	Compare memory to A and skip if high or equal; relative to P forward, direct	2	2-15
CMS	D300*	Compare memory to A and skip if high or equal; relative to P forward, indirect	2+1n	2-15
CMS	D400*	Compare memory to A and skip if high or equal; indexed, direct	2	2-15
CMS	D500*	Compare memory to A and skip if high or equal; indexed, indirect	2+1n	2-15
CMS	D600*	Compare memory to A and skip if high or equal; relative to P backward, direct	2	2-15

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
CMS	D700*	Compare memory to A and skip if high or equal; relative to P backward, indirect	2+1n	2-15
CMSB	:D000*	Compare Byte and skip if high or equal, direct, scratchpad	2	2-29
CMSB	:D100*	Compare Byte and skip if high or equal, indirect, AP in scratchpad	3	2-29
CMSB	:D200*	Compare Byte 0 and skip if high or equal, direct, relative to P forward	2	2-29
CMSB	:D300*	Compare Byte and skip if high or equal, indirect, AP relative to P forward	3	2-29
CMSB	:D400*	Compare Byte and skip if high or equal, indexed, direct	2	2-29
CMSB	:D500*	Compare Byte and skip if high or equal, indexed, indirect, AP in scratchpad	3	2-29
CMSB	:D600*	Compare Byte 1 and skip if high or equal, direct, relative to P forward	2	2-29
CMSB	:D700*	Compare Byte and skip if high or equal, indirect, AP relative to P backward	3	2-29
COV	1600	Complement overflow	1	2-68
CXA	0410	Complement X and put in A	1	2-63
CXI	C100*	Compare to X immediate	1	2-33
CXR	0408	Complement X	1	2-62
DAR	00D0	Decrement A	1	2-60
DAX	00C8	Decrement A and put in X	1	2-64
DIN	0C00	Disable interrupts	1	2-69
DVS	1940*	Divide step	1-1/4 + 1/2K	2-53
DXA	00B0	Decrement X and put in A	1	2-64
DXR	00A8	Decrement X	1	2-60
EIN	0A00	Enable interrupts	1	2-68
EMA	B800*	Exchange memory and A; direct, scratchpad	2	2-13
EMA	B900*	Exchange memory and A; indirect, AP in scratchpad	2+1n	2-13
EMA	BA00*	Exchange memory and A; relative to P forward, direct	2	2-13
EMA	BB00*	Exchange memory and A; relative to P forward, indirect	2+1n	2-13
EMA	BC00*	Exchange memory and A; indexed, direct	2	2-13
EMA	BD00*	Exchange memory and A; indexed, indirect	2+1n	2-13
EMA	BE00*	Exchange memory and A; relative to P backward, direct	2	2-13
EMA	BF00*	Exchange memory and A; relative to P backward, indirect	2+1n	2-13

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
EMAB	:B800*	Exchange Memory and A Byte, direct, scratchpad	2	2-27
EMAB	:B900*	Exchange Memory and A Byte, indirect, AP in scratchpad	3	2-27
EMAB	:BA00*	Exchange Memory and A Byte 0, direct, relative to P forward	2	2-27
EMAB	:BB00*	Exchange Memory and A Byte, indirect, AP relative to P forward	3	2-27
EMAB	:BC00*	Exchange Memory and A Byte, indexed, direct	2	2-27
EMAB	:BD00*	Exchange Memory and A Byte, indexed, indirect, AP in scratchpad	3	2-27
EMAB	:BE00*	Exchange Memory and A Byte 1, direct, relative to P forward	2	2-27
EMAB	:BF00*	Exchange Memory and A, indirect, AP relative to P backward	3	2-27
HLT	0800	Halt	1	2-67
IAR	0150	Increment A	1	2-61
IAX	0148	Increment A and put in X	1	2-64
IBA	7800*	Input byte to A (unconditionally)	1-1/4	3-19
IBAM	7C00*	Input byte to A, masked (unconditionally)	1-1/4	3-20
IBX	7A00*	Input byte to X (unconditionally)	1-1/4	3-19
IBXM	7E00*	Input byte to X, masked (unconditionally)	1-1/4	3-20
IMS	D800*	Increment memory and skip on zero result; direct, scratchpad	2	2-11
IMS	D900*	Increment memory and skip on zero result; indirect, AP in scratchpad	2+1n	2-11
IMS	DA00*	Increment memory and skip on zero result; relative to P forward, direct	2	2-11
IMS	DB00*	Increment memory and skip on zero result; relative to P forward, indirect	2+1n	2-11
IMS	DC00*	Increment memory and skip on zero result; indexed, direct	2	2-11
IMS	DD00*	Increment memory and skip on zero result; indexed, indirect	2+1n	2-11
IMS	DE00*	Increment memory and skip on zero result; relative to P backward, direct	2	2-11
IMS	DF00*	Increment memory and skip on zero result; relative to P backward, indirect	2+1n	2-11
INA	5800*	Input word to A (unconditionally)	1-1/4	3-19
INAM	5C00*	Input word to A, masked (unconditionally)	1-1/4	3-20

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
INX	5A00*	Input word to X (unconditionally)	1-1/4	3-19
INXM	5E00*	Input word to X, masked (unconditionally)	1-1/4	3-20
IOR	A000*	Inclusive OR to A; direct, scratchpad	2	2-14
IOR	A100*	Inclusive OR to A; indirect, AP in scratchpad	2+1n	2-14
IOR	A200*	Inclusive OR to A; relative to P forward, direct	2	2-14
IOR	A300*	Inclusive OR to A; relative to P forward, indirect	2+1n	2-14
IOR	A400*	Inclusive OR to A; indexed, direct	2	2-14
IOR	A500*	Inclusive OR to A; indexed, indirect	2+1n	2-14
IOR	A600*	Inclusive OR to A; relative to P backward, direct	2	2-14
IOR	A700*	Inclusive OR to A; relative to P backward, indirect	2+1n	2-14
IORB	:A000*	Inclusive OR Byte, direct, scratchpad	2	2-28
IORB	:A100*	Inclusive OR Byte, indirect, AP in scratchpad	3	2-28
IORB	:A200*	Inclusive OR Byte 0, direct, relative to P forward	2	2-28
IORB	:A300*	Inclusive OR Byte, indirect, AP relative to P forward	3	2-28
IORB	:A400*	Inclusive OR Byte, indexed, direct	2	2-28
IORB	:A500*	Inclusive OR Byte, indexed, indirect, AP in scratchpad	3	2-28
IORB	:A600*	Inclusive OR Byte 0, direct, relative to P forward	2	2-28
IORB	:A700*	Inclusive OR Btye, indirect, AP relative to P backward	3	2-28
ISA	5801	Input data switches to A	1-1/4	2-66
ISX	5B01	Input data switches to X	1-1/4	2-66
IXA	0130	Increment X and put in A	1	2-64
IXR	0128	Increment X	1	2-61
JAG (JOC		Jump if A positive and not equal to zero: (A) > 0	1	2-37
:23, ADR)	3180*	Forward jump		
	31C0*	Backward jump		
JAL (JOC		Jump if A negative or equal to zero: (A) ≤ 0	1	2-38
:03, ADR)	2180*	Forward jump		
	21C0*	Backward jump		
JAM (JOC		Jump if A negative: (A) < 0	1	2-36
:01, ADR)	2080*	Forward jump		
	20C0*	Backward jump		
JAN (JOC		Jump if A not zero: (A) ≠ 0	1	2-37
:22, ADR)	3100*	Forward jump		
	3140*	Backward jump		
JAP (JOC		Jump if A positive or equal to zero: (A) ≥ 0	1	2-36
:21, ADR)	3080*	Forward jump		
	30C0*	Backward jump		

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JAZ (JOC		Jump if A zero: (A) = 0	1	2-37
:02, ADR)	2100*	Forward jump		
	2140*	Backward jump		
JMP	F000*	Jump unconditionally; direct, scratchpad	1	2-18
JMP	F100*	Jump unconditionally; indirect, AP in scratchpad	2	2-18
JMP	F200*	Jump unconditionally; relative to P forward, direct	1	2-18
JMP	F300*	Jump unconditionally; relative to P forward, indirect	2	2-18
JMP	F400*	Jump unconditionally; indexed, direct	1	2-18
JMP	F500*	Jump unconditionally; indexed, indirect	2	2-18
JMP	F600*	Jump unconditionally; relative to P backward, direct	1	2-18
JMP	F700*	Jump unconditionally; relative to P backward, indirect	2	2-18
JOC :01,		Jump if A negative: (A) < 0	1	2-36
ADR (JAM)	2080*	Forward jump		
	20C0*	Backward jump		
JOC :02,		Jump if A zero: (A) = 0	1	2-36
ADR (JAZ)	2100*	Forward jump		
	2140*	Backward jump		
JOC :03,		Jump if A negative or equal to zero: (A) ≤ 0	1	2-36
ADR (JAL)	2180*	Forward jump		
	21C0*	Backward jump		
JOC :04,		Jump if overflow set: OV = 1	1	2-36
ADR (JOS)	2200*	Forward jump		
	2240*	Backward jump		
JOC :05,		Jump if overflow set or A negative: OV = 1 ∨ (A) < 0	1	2-36
ADR	2280*	Forward jump		
	22C0*	Backward jump		
JOC :06,		Jump if overflow set or A equals zero: OV = 1 ∨ (A) = 0	1	2-36
ADR	2300*	Forward jump		
	2340*	Backward jump		
JOC :07,		Jump if overflow set or A less than or equal to zero:	1	2-36
ADR		OV = 1 ∨ (A) ≤ 0		
	2380*	Forward jump		
	23C0*	Backward jump		
JOC :08,		Jump if Sense Switch off: SS = 0	1	2-36
ADR (JSR)	2400*	Forward jump		
	2440*	Backward jump		
JOC :09,		Jump if Sense Switch off or A negative: SS = 0 ∨ (A) < 0	1	2-36
ADR	2480*	Forward jump		
	24C0*	Backward jump		

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOC :0A,		Jump if Sense Switch off or A equal to zero;	1	2-36
ADR		SS = 0 $\vee$ (A) = 0		
	2500*	Forward jump		
	2540*	Backward jump		
JOC :0B,		Jump if Sense Switch off or A less than or equal	1	2-36
ADR		to zero: SS = 0 $\vee$ (A) $\leq$ 0		
	2580*	Forward jump		
	25C0*	Backward jump		
JOC :0C,		Jump if Sense Switch off or overflow set: SS = 0$\vee$OV=1	1	2-36
ADR	2600*	Forward jump		
	2640*	Backward jump		
JOC :0D,		Jump if Sense Switch off or overflow set or A negative:	1	2-36
ADR		SS = 0 $\vee$ OV = 1 $\vee$ (A) < 0		
	2680*	Forward jump		
	26D0*	Backward jump		
JOC :0E,		Jump if Sense Switch off or overflow set or A equals	1	2-36
ADR		zero: SS = 0 $\vee$ OV = 1 $\vee$ (A) = 0		
	2700*	Forward jump		
	2740*	Backward jump		
JOC :0F,		Jump if Sense Switch off or overflow set or A less than	1	2-36
ADR		or equal to zero: SS = 0 $\vee$ OV = 1 $\vee$ (A) $\leq$ 0		
	2780*	Forward jump		
	27C0*	Backward jump		
JOC :10,		Jump if X equals zero: (X) = 0	1	2-36
ADR (JXZ)	2800*	Forward jump		
	2840*	Backward jump		
JOC :11,		Jump if X equals zero or A negative: (X) = 0 $\vee$ (A) < 0	1	2-36
ADR	2880*	Forward jump		
	28C0*	Backward jump		
JOC :12,	2900*	Illegal combination (includes (X) = 0 $\vee$ (A) = 0)		2-36
ADR	2940*			
JOC :13	2980*	Illegal combination (includes (X) = 0 $\vee$ (A) = 0)		2-36
ADR	29C0*			
JOC :14,		Jump if X equals zero or overflow set: (X) = 0 $\vee$ OV = 1	1	2-36
ADR	2A00*	Forward jump		
	2A40*	Backward jump		

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOC :15		Jump if X equals zero or overflow set or A negative:	1	2-36
ADR		(X)= 0 ∨ OV = 1 ∨ (A) < 0		
	2A80*	Forward jump		
	2AC0*	Backward jump		
JOC :16,	2B00*	Illegal combinations (include (X) = 0 ∨ (A) = 0)		2-36
ADR	2B40*			
JOC :17	2B80*	Illegal combinations (include (X) = 0 ∨ (A) = 0)		2-36
	2BC0*			
JOC :18,		Jump if X equals zero or Sense Switch off:	1	2-36
ADR		(X) = 0 ∨ SS = 0		
	2C00*	Forward jump		
	2C40*	Backward jump		
JOC :19,		Jump if X equals zero or Sense Switch off or A is	1	2-36
ADR		(X) = 0 ∨ SS = 0 ∨ (A) < 0		
	2C80*	Forward jump		
	2CC0*	Backward jump		
JOC :1A,	2D00*	Illegal combination (includes (X) = 0 ∨ (A) = 0)		2-36
ADR	2D40*			
JOC :1B,	2D80*	Illegal combination (includes (X) = 0 ∨ (A) = 0)		2-36
ADR	2DC0*			
JOC :1C,		Jump if X equals zero or Sense Switch off or over-	1	2-36
ADR		flow set: (X) = 0 ∨ SS-0 ∨ OV = 1		
	2E00*	Forward jump		
	2E40*	Backward jump		
JOC :1D,		Jump if X equals zero or Sense Switch off or overflow	1	2-36
ADR		set or A negative: (X) = 0 ∨ SS = 0 ∨ OV = 1 ∨ (A) < 0		
	2E80*	Forward jump		
	2EC0*	Backward jump		
JOC :1E,	2F00*	Illegal combination (includes (X) = 0 ∨ (A) = 0)		2-36
ADR	2F40*			
JOC :1F,	2F80*	Illegal combination (includes (X) = 0 ∨ (A) = 0)		2-36
ADR	2FC0*			
JOC :21,		Jump if A positive or equal to zero: (A) ∨ 0	1	2-36
ADR	3080*	Forward jump		
(JAP)	30C0*	Backward jump		
JOC :22,		Jump if A not zero: (A) = 0	1	2-36
ADR	3100*	Forward jump		
(JAN)	3140*	Backward jump		

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOC :23,		Jump if A greater than zero: (A) > 0	1	2-36
ADR	3180*	Forward jump		
(JAG)	31C0*	Backward jump		
JOC :24,		Jump if overflow is reset: OV = 0	1	2-36
ADR	3200*	Forward jump		
(JOR)	3240*	Backward jump		
JOC :25,		Jump if A positive and overflow is reset:	1	2-36
ADR		$(A) \geq 0 \wedge OV = 0$		
	3280*	Forward jump		
	32C0*	Backward jump		
JOC :26,		Jump if a non-zero and overflow reset:	1	2-36
ADR		$(A) \neq 0 \wedge OV = 0$		
	3300*	Forward jump		
	3340*	Backward jump		
JOC :27,		Jump if a non-zero and overflow reset:	1	2-36
ADR		$(A) > 0 \wedge OV = 0$		
	3380*	Forward jump		
	33C0*	Backward jump		
JOC :28,		Jump if Sense Switch on. SS = 1	1	2-36
ADR	3400*	Forward jump		
(JSS)	3440*	Backward jump		
JOC :29,		Jump if Sense Switch on and A positive:	1	3-26
ADR		$SS = 1 \wedge (A) > 0$		
	3480*	Forward jump		
	34C0*	Backward jump		
JOC :2A,		Jump if Sense Switch on and A non-zero:	1	2-36
ADR		$SS = 1 \wedge (A) \neq 0$		
	3500*	Forward jump		
	3540*	Backward jump		
JOC :2B,		Jump if Sense Switch on and A greater than zero:	1	2-36
ADR		$SS = 1 \wedge (A) > 0$		
	3580*	Forward jump		
	35C0*	Backward jump		
JOC :2C,		Jump if Sense Switch on and overflow reset:	1	2-36
ADR		$SS = 1 \wedge OV = 0$		
	3600*	Forward jump		
	3640*	Backward jump		

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOC :2D,		Jump if Sense Switch on and A positive and overflow	1	2-36
ADR		reset: SS = 1 ∧ (A) > 0 ∧ OV = 0		
	3680*	Forward jump		
	36C0*	Backward jump		
JOC :2E,		Jump if Sense Switch on and A non-zero and overflow	1	2-36
ADR		reset: SS = 1 ∧ (A) ≠ 0 ∧ OV = 0		
	3700*	Forward jump		
	3740*	Backward jump		
JOC :2F,		Jump if A greater than zero and Sense Switch on and	1	2-36
ADR		overflow reset (A)≠0 ∧ SS = 1 ∧ OV = 0		
	3780*	Forward jump		
	37C0*	Backward jump		
JOC :30,		Jump if X non-zero (X)≠0	1	2-36
ADR(JXN)	3800*	Jump forward		
	3840*	Jump backward		
JOC :31,		Jump if X non-zero and A positive: (X)≠0 ∧ (A) ≥ 0	1	2-36
ADR	3880*	Forward jump		
	38C0*	Backward jump		
JOC :32,		Jump if X non-zero and A non-zero. (X)≠0 ∧ (A)≠ 0	1	2-36
ADR	3900*	Forward jump		
	3940*	Backward jump		
JOC :33,		Jump if X non-zero and A greater than zero	1	2-36
ADR		(X) ≠ 0 ∧ (A) > 0		
	3980*	Forward jump		
	39C0*	Backward jump		
JOC :34,		Jump if X non-zero and overflow reset	1	2-36
ADR		(X) ≠ 0 ∧ OV = 0		
	3A00*	Forward jump		
	3A40*	Backward jump		
JOC :35,		Jump if X non-zero and A positive and overflow reset:	1	2-36
ADR		(X) ≠ 0 ∧ (A) ≥ 0 ∧ OV = 0		
	3A80*	Forward jump		
	3AC0*	Backward jump		
JOC :36,		Jump if X non-zero and A non-zero and overflow reset:	1	2-36
ADR		(X) ≠ 0 ∧ (A) ≠ 0 ∧ OV = 0		
	3B00*	Forward jump		
	3B40*	Backward jump		

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOC :37, ADR		Jump if X non-zero and A greater than zero and overflow reset. (X) ≠ 0 ∧ (A) > 0 ∧ OV = 0	1	2-36
	3B80*	Forward jump		
	3BC0*	Backward jump		
JOC :38, ADR		Jump if X non-zero and Sense Switch on: (X) ≠ 0 ∧ SS=1	1	2-36
	3C00*	Forward jump		
	3C40*	Backward jump		
JOC :39, ADR		Jump if X non-zero and A positive and Sense Switch on: (X) ≠ 0 ∧ (A) ≠ 0 ∧ SS-1	1	2-36
	3C80*	Forward jump		
	3CC0*	Backward jump		
JOC :3A, ADR		Jump if X non-zero and A non-zero and Sense Switch on: (X) ≠ 0 ∧ (A) > 0 ∧ SS = 1	1	2-36
	3D00*	Forward jump		
	3D40*	Backward jump		
JOC :3B, ADR		Jump if X non-zero and A greater than zero and Sense Switch on: (X) ≠ 0 ∧ (A) > 0 ∧ SS = 1	1	2-36
	3B80*	Forward jump		
	3DC0*	Backward jump		
JOC :3C, ADR		Jump if X non-zero and Sense Switch on and overflow reset: (X) ≠ 0 ∧ SS = 1 ∧ OV = 0	1	2-36
	3E00*	Forward jump		
	3E40*	Backward jump		
JOC :3D, ADR		Jump if X non-zero and A positive and Sense Switch on and overflow reset: (X) ≠ 0 ∧ (A) ≥ 0 ∧ SS=1 ∧ OV=0	1	2-36
	3E80*	Forward jump		
	3EC0*	Backward jump		
JOC :3E, ADR		Jump if X non-zero and A non-zero and Sense Switch on and overflow reset: (X) ≠ 0 ∧ (A) ≠ 0 ∧ SS = 1 ∧ OV = 0	1	2-36
	3F00*	Forward jump		
	3F40*	Backward jump		
JOC :3F, ADR		Jump if X non-zero and A greater than zero and Sense Switch on and overflow reset: (X) ≠ 0 ∧ (A) > 0 ∧ SS = 1 ∧ OV = 0	1	2-36
	3F80*	Forward jump		
	3FC0*	Backward jump		
JOR (JOC :24, ADR		Jump if overflow reset: OV = 0	1	2-36
	3200*	Forward jump		
	3240*	Backward jump		

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOS (JOC :04,		Jump if overflow set: OV = 1	1	2-39
ADR	2200*	Forward jump		
	2240*	Backward jump		
JSR (JOS :08,		Jump if Sense Switch off: SS = 0	1	2-39
ADR)	2400*	Forward jump		
	2440*	Backward jump		
JSS (JOC :28,		Jump if Sense Switch on: SS = 1	1	2-39
ADR)	3400*	Forward jump		
	3440*	Backward jump		
JST	F800*	Jump and Store; direct, scratchpad	2	2-18
JST	F900*	Jump and Store; indirect, AP in scratchpad	3	2-18
JST	FA00*	Jump and Store; relative to P forward, direct	2	2-18
JST	FB00*	Jump and Store; relative to P forward, indirect	3	2-18
JST	FC00*	Jump and Store; indexed, direct	2	2-18
JST	FD00*	Jump and Store; indexed, indirect	3	2-18
JST	FE00*	Jump and Store; relative to P backward, direct	2	2-18
JST	FF00*	Jump and Store; relative to P backward, indirect	3	2-18
JXN (JOC :30,		Jump if X non-zero: (X) ≠ 0	1	2-38
ADR)	3800*	Forward jump		
	3840*	Backward jump		
JXZ (JOC :10,		Jump if X equal to zero: (X) = 0	1	2-38
ADR)	2800*	Forward jump		
	2840*	Backward jump		
LAM	C700*	Load A minus immediate	1	2-32
LAP	C600*	Load A positive immediate	1	2-32
LDA	B000*	Load A; direct, scratchpad	2	2-12
LDA	B100*	Load A; indirect, AP in scratchpad	2+1n	2-12
LDA	B200*	Load A; relative to P forward, direct	2	2-12
LDA	B300*	Load A; relative to P forward, indirect	2+1n	2-12
LDA	B400*	Load A; indexed, direct	2	2-12
LDA	B500*	Load A; indexed, indirect	2+1n	2-12
LDA	B600*	Load A; relative to P backward, direct	2	2-12
LDA	B700*	Load A; relative to P backward, indirect	2+1n	2-12
LDAB	:B000*	Load A Byte, direct, scratchpad	2	2-25
LDAB	:B100*	Load A Byte, indirect, AP in scratchpad	3	2-25
LDAB	:B200*	Load A Byte 0, direct, relative to P forward	2	2-25
LDAB	:B300*	Load A Byte, indirect, AP relative to P forward	3	2-25
LDAB	:B400*	Load A Byte, indexed, direct	2	2-25
LDAB	:B500*	Load A Byte, indexed, indirect, AP in scratchpad	3	2-25

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
LDAB	:B600*	Load A Byte 1, direct, relative to P forward	2	2-25
LDAB	:B700*	Load A Byte, indirect, AP relative to P backward	3	2-25
LDX	E000*	Load X; direct, scratchpad	2	2-12
LDX	E100*	Load X; indirect, AP in scratchpad	2+1n	2-12
LDX	E200*	Load X; relative to P forward, direct	2	2-12
LDX	E300*	Load X; relative to P forward, indirect	2+1n	2-12
LDX	E400*	Load X; indexed, direct	2	2-12
LDX	E500*	Load X; indexed, indirect	2+1n	2-12
LDX	E600*	Load X; relative to P backward, direct	2	2-12
LDX	E700*	Load X; relative to P backward, indirect	2+1n	2-12
LDXB	:E000*	Load X Byte, direct, scratchpad	2	2-25
LDXB	:E100*	Load X Byte, indirect, AP in scratchpad	3	2-25
LDXB	:E200*	Load X Byte 0, direct, relative to P forward	2	2-25
LDXB	:E300*	Load X Byte, indirect, relative to P forward	3	2-25
LDXB	:E400*	Load X Byte, indexed, direct	2	2-25
LDXB	:E500*	Load X, indexed, indirect, AP in scratchpad	3	2-25
LDXB	:E600*	Load X Byte 1, direct, relative to P forward	2	2-25
LDXB	:E700*	Load X Byte, indirect, relative to P backward	3	2-25
LLA	1350*	Logical shift A left	1+1/4K	2-46
LLL	1B00*	Long logical left shift	1-1/4+1/2K	2-50
LLR	1B80*	Long logical right shift	1-1/4+1/2K	2-49
LLX	1328*	Logical shift X left	1+1/4K	2-47
LRA	13D0*	Logical shift A right	1+1/4K	2-47
LRL	1900*	Long rotate left	1-1/4+1/2K	2-50
LRR	1980*	Long rotate right	1-1/4+1/2K	2-50
LRX	13A8*	Logical shift X right	1+1/4K	2-46
LXM	C500*	Load X minus immediate	1	2-33
LXP	C400*	Load X positive immediate	1	2-32
MPD	4001	Memory Protect disable	1-1/4	2-73
MPE	4000	Memory Protect enable	1-1/4	2-73
MPS	19A0*	Multiply step	1-1/4+1/2K	2-51
NAR	0310	Negate A register	1	2-61
NAX	0308	Negate A and put in X	1	2-63
NOP	0000	No operation	1	2-67
NOR	1228*	Normalize	1/4K	2-49
NRA	0610	NOR if (A and X) to A: $\overline{(A) \vee (X)} \rightarrow A$	1	2-65
NRX	0608	NOR of (A and X) to X: $\overline{(A) \vee (X)} \rightarrow A$	1	2-65
NXA	0510	Negate X and put in A	1	2-63
NXR	0508	Negate X register	1	2-61

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
OTA	6C00*	Output A Register (unconditionally)	1-1/4	3-22
OTX	6E00*	Output X register (unconditionally)	1-1/4	3-22
OTZ	6800*	Output zero (unconditionally)	1-1/4	3-22
PFD	4003	Power Fail interrupt disable	1-1/4	2-72
PFE	4002	Power Fail interrupt enable	1-1/4	2-72
RAM	4045	Set Random Access mode	1-1/4	2-74
RBA	7900*	Read byte to A	1-1/4	3-21
RBAM	7D00*	Read byte to A, masked	1-1/4	3-21
RBX	7B00*	Read byte to X	1-1/4	3-21
RBXM	7F00*	Read byte to X, masked	1-1/4	3-21
RDA	5900*	Read word to A	1-1/4	3-20
RDAM	5D00*	Read word to A, masked	1-1/4	3-21
RDX	5B00*	Read word to X	1-1/4	3-20
RDXM	5F00*	Read word to X, masked	1-1/4	3-21
RLA	1150*	Rotate A left with OV	1+1/4K	2-48
RLX	1128*	Rotate X left with OV	1+1/4K	2-48
ROM	4046	Set Read Only mode	1-1/4	2-74
ROV	1200	Reset overflow	1	2-67
RRA	11D0*	Rotate A right with OV	1+1/4K	2-47
RRX	11A8*	Rotate X right with OV	1+1/4K	2-48
SAO	1340	Sign of A to OV	1	2-49
SBM	0E00	Set byte mode	1	2-68
SCN	CD00*	Scan memory, indexed, indirect	2+1w	2-16
SEA	4400*	Select and present A	1-1/4	3-18
SEL	4000*	Select function	1-1/4	3-18
SEN	4900*	Sense and skip on response	1-1/4	3-17
SEX	4600*	Select and present X	1-1/4	3-18
SIA	5800	Status input to A	1-1/4	2-70
SIN	6800	Status inhibit	1-1/4	2-69
SIX	5A00	Status input to X	1-1/4	2-70
SOA	6C00	Status output from A	1-1/4	2-71
SOX	6E00	Status output from X	1-1/4	2-71
SOV	1400	Set overflow	1	2-67
SSN	4800*	Sense and skip on no response	1-1/4	3-17
STA	9800*	Store A; direct, scratchpad	2	2-13
STA	9900*	Store A; indirect, AP in scratchpad	2+1n	2-13
STA	9A00*	Store A; relative to P forward, direct	2	2-13
STA	9B00*	Store A; relative to P forward, indirect	2+1n	2-13
STA	9C00*	Store A; indexed, direct	2	2-13

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
STA	9D00*	Store A; indexed, indirect	2+1n	2-13
STA	9E00*	Store A; relative to P backward, direct	2	2-13
STA	9F00*	Store A; relative to P backward, indirect	2+1n	2-13
STAB	:9800*	Store A Byte, direct, scratchpad	2	2-26
STAB	:9900*	Store A Byte, indirect, AP in scratchpad	3	2-26
STAB	:9A00*	Store A Byte 0, direct, relative to P forward	2	2-26
STAB	:9B00*	Store A Byte, indirect, AP relative to P forward	3	2-26
STAB	:9C00*	Store A Byte, indexed, direct	2	2-26
STAB	:9D00*	Store A Byte, indexed, indirect, AP in scratchpad	3	2-26
STAB	:9E00*	Store A Byte 1, direct, relative to P forward	2	2-26
STAB	:9F00*	Store A Byte, indirect, AP relative to P backward	3	2-26
STX	E800*	Store X; direct, scratchpad	2	2-13
STX	E900*	Store X; indirect, AP in scratchpad	2+1n	2-13
STX	EA00*	Store X; relative to P forward, direct	2	2-13
STX	EB00*	Store X; relative to P forward, indirect	2+1n	2-13
STX	EC00*	Store X; indexed, direct	2	2-13
STX	ED00*	Store X; indexed, indirect	2+1n	2-13
STX	EE00*	Store X; relative to P backward, direct	2	2-13
STX	EF00*	Store X; relative to P backward, indirect	2+1n	2-13
STXB	:E800*	Store X Byte, direct, scratchpad	2	2-26
STXB	:E900*	Store X Byte, indirect, AP in scratchpad	3	2-26
STXB	:EA00*	Store X Byte 0, direct, relative to P forward	2	2-26
STXB	:EB00*	Store X Byte, indirect, relative to P forward	3	2-26
STXB	:EC00*	Store X Byte, indexed, direct	2	2-26
STXB	:ED00*	Store X Byte, indexed, indirect, AP	3	2-26
STXB	:EE00*	Store X Byte 1, direct, relative to P forward	2	2-26
STXB	:EF00*	Store X Byte, indirect, relative to P backward	3	2-26
SUB	9000*	Subtract from A; direct, scratchpad	2	2-11
SUB	9100*	Subtract from A; indirect, AP in scratchpad	2+1n	2-11
SUB	9200*	Subtract from A; relative to P forward, direct	2	2-11
SUB	9300*	Subtract from A; relative to P forward, indirect	2+1n	2-11
SUB	9400*	Subtract from A; indexed, direct	2	2-11
SUB	9500*	Subtract from A; indexed, indirect	2+1n	2-11
SUB	9600*	Subtract from A; relative to P backward, direct	2	2-11
SUB	9700*	Subtract from A; relative to P backward, indirect	2+1n	2-11
SUBB	9000*	Subtract Byte, direct, scratchpad	2	2-24
SUBB	:9100*	Subtract Byte, indirect, AP in scratchpad	3	2-24
SUBB	:9200*	Subtract Byte 0, direct, relative to P forward	2	2-24
SUBB	:9300*	Subtract Byte, indirect, AP relative to P forward	3	2-24

INSTRUCTION SET, ALPHABETICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
SUBB	:9400*	Subtract Byte, indexed, direct	2	2-24
SUBB	:9500*	Subtract Byte, indirect, indexed, AP in scratchpad	3	2-24
SUBB	:9600*	Subtract Byte 1, direct, relative to P forward	2	2-24
SUBB	:9700*	Subtract Byte, indirect, relative to P backward	3	2-24
SWM	0F00	Set word mode	1	2-68
SXI	C300*	Subtract from X immediate	1	2-31
TAX	0048	Transfer A to X	1	2-62
TRP	4007	Trap	1-1/4	2-69
TXA	0030	Transfer X to A	1	2-62
WRA	6D00*	Write from A	1-1/4	3-23
WRX	6F00*	Write from X	1-1/4	3-23
WRZ	6900*	Write zeros	1-1/4	3-23
XOR	A800*	Exclusive OR to A; direct scratchpad	2	2-15
XOR	A900*	Exclusive OR to A; indirect, AP in scratchpad	2+1n	2-15
XOR	AA00*	Exclusive OR to A; relative to P forward, direct	2	2-15
XOR	AB00*	Exclusive OR to A; relative to P forward, indirect	2+1n	2-15
XOR	AC00*	Exclusive OR to A; indexed, direct	2	2-15
XOR	AD00*	Exclusive OR to A; indexed, indirect	2+1n	2-15
XOR	AE00*	Exclusive OR to A; relative to P backward, direct	2	2-15
XOR	AF00*	Exclusive OR to A; relative to P backward, indirect	2+1n	2-15
XORB	:A800*	Exclusive OR Byte, direct, scratchpad	2	2-29
XORB	:A900*	Exclusive OR Byte, indirect, AP in scratchpad	3	2-29
XORB	:AA00*	Exclusive OR Byte 0, direct, relative to P forward	2	2-29
XORB	:AB00*	Exclusive OR Byte, indirect, AP relative to P forward	3	2-29
XORB	:AC00*	Exclusive OR Byte, indexed, direct	2	2-29
XORB	:AD00*	Exclusive OR Byte, indexed, indirect, AP in scratchpad	3	2-29
XORB	:AE00*	Exclusive OR Byte 1, direct, relative to P forward	2	2-29
XORB	:AF00*	Exclusive OR Byte, indirect, AP relative to P backward	3	2-29
XRM	0008	Set X to minus 1	1	2-59
XRP	0528	Set X to plus 1	1	2-60
ZAR	0110	Zero A register	1	2-58
ZAX	0118	Zero A and X registers	1	2-58
ZXR	0108	Zero X register	1	2-58

APPENDIX E

INSTRUCTION SET, NUMERICAL ORDER

Instructions are listed in numerical order by hexadecimal code in the Appendix.

*Instruction codes followed by an asterisk (*) are shown with variable fields containing all zeros (address fields, jump distances, shift counts, device addresses, etc.). Instruction codes not followed by an asterisk do not have variable fields and the code shown is the only code that defines the instruction.

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
NOP	0000	No operation	1	2-67
XRM	000B	Set X to minus 1	1	2-59
ARM	0010	Set A to minus 1	1	2-59
AXM	0018	Set A and X to minus 1	1	2-59
TXA	0030	Transfer X to A	1	2-62
TAX	0048	Transfer A to X	1	2-62
ANX	0068	AND of A and X to X	1	3-31
ANA	0070	AND of A and X to A	1	2-65
DXR	00A8	Decrement X	1	2-60
DXA	00B0	Decrement X and put in A	1	2-64
DAX	00C8	Decrement A and put in X	1	2-64
DAR	00D0	Decrement A	1	2-60
ZXR	0108	Zero X register	1	2-58
ZAR	0110	Zero A register	1	2-58
ZAX	0118	Zero A and X registers	1	2-58
IXR	0128	Increment X	1	2-61
IXA	0130	Increment X and put in A	1	2-64
IAX	0148	Increment A and put in X	1	2-64
IAR	0150	Increment A	1	2-61
CAX	0208	Complement A and put in X	1	2-63
CAR	0210	Complement A	1	2-62
NAX	0308	Negate A and put in X	1	2-63
NAR	0310	Negate A register	1	2-61
ARP	0350	Set A to plus 1	1	2-59
AXP	0358	Set A and X to plus 1	1	2-60
CXR	0408	Complement X	1	2-62
CXA	0410	Complement X and put in A	1	2-63
NXR	0508	Negate X register	1	2-61
NXA	0510	Negate X and put in A	1	2-63

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
XRP	0528	Set X to plus 1	1	2-60
NRX	0608	NCP of (A and X) to X: $\overline{(A) \vee (X)} \rightarrow X$	1	2-65
NRA	0610	NOP of (A and X) to A: $\overline{(A) \vee (X)} \rightarrow A$	1	2-65
HLT	0800	Halt	1	2-67
EIN	0A00	Enable interrupts	1	2-68
DIN	0C00	Disable interrupts	1	2-69
SBM	0E00	Set byte mode	1	2-68
SWM	0F00	Set word mode	1	2-68
ALX	1028*	Arithmetic shift X left	1+1/4K	2-45
ALA	1050*	Arithmetic shift A left	1+1/4K	2-45
ARX	10A8*	Arithmetic shift X right	1+1/4K	2-45
ARA	10D0*	Arithmetic shift A right	1+1/4K	2-44
RLX	1128*	Rotate X left with OV	1+1/4K	2-48
RLA	1150*	Rotate A left with OV	1+1/4K	2-48
RRX	11A8*	Rotate X right with OV	1+1/4K	2-48
RRA	11D0*	Rotate A right with OV	1+1/4K	2-47
ROV	1200	Reset overflow	1	2-67
NOR	1228*	Normalize	1/4K	2-49
LLX	1328*	Logical shift X left	1+1/4K	2-47
SAO	1340	Sign of A to OV	1	2-49
LLA	1350*	Logical shift A left	1+1/4K	2-46
LRX	13A8*	Logical shift X right	1+1/4K	2-46
LRA	13D0*	Logical shift A right	1+1/4K	2-46
SOV	1400	Set overflow	1	2-67
COV	1600	Complement overflow	1	2-68
LRL	1900*	Long rotate left	1-1/4+1/2K	2-50
DVS	1940*	Divide step	1-1/4+1/2K	2-53
LRR	1980*	Long rotate right	1-1/4+1/2K	2-50
MPS	19A0*	Multiply step	1-1/4+1/2K	2-51
LLL	1B00*	Long logical left	1-1/4+1/2K	2-50
LLR	1B80*	Long logical right	1-1/4+1/2K	2-49
JAM (JOC :01,		Jump if A negative: (A)< 0	1	2-36
ADR)	2080*	Forward jump		
	20C0*	Backward jump		
JAZ (JOC :02,		Jump if A zero: (A)=0	1	2-37
ADR)	2100*	Forward jump		
	2140*	Backward jump		

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JAL (JOC :03,		Jump if A negative or equal to zero: (A)≤ 0	1	2-38
ADR)	2180*	Forward jump		
	21C0*	Backward jump		
JOS (JOC :04,		Jump if overflow set: OV=1	1	2-39
ADR)	2200*	Forward jump		
	2240*	Backward jump		
JOC :05,ADR		Jump if overflow set or A negative: OV=1 ∨ (A) < 0	1	2-36
	2280*	Forward jump		
	22C0*	Backward jump		
JOC :06,ADR		Jump if overflow set or A equals zero: OV=1 ∨ (A)=0	1	2-36
	2300*	Forward jump		
	2340*	Backward jump		
JOC :07,ADR		Jump if overflow set or A less than or equal to zero: OV=1 ∨ (A) < 0	1	2-36
	2380*	Forward jump		
	23C0*	Backward jump		
JSR (JOS :08,		Jump if Sense Switch off: SS=0	1	2-39
ADR)	2400*	Forward jump		
	2440*	Backward jump		
JOC :09,ADR		Jump if Sense Switch off or A negative: SS=0 ∨ (A) < 0	1	2-36
	2480*	Forward jump		
	24C0*	Backward jump		
JOC :0A,ADR		Jump if Sense Switch off or A equal to zero: SS=0 ∨ (A)=0	1	2-36
	2500*	Forward jump		
	2540*	Backward jump		
JOC :0B,ADR		Jump if Sense Switch off or A less than or equal to zero: SS=0 ∨ (A) ≤ 0	1	2-36
	2580*	Forward jump		
	25C0*	Backward jump		
JOC :0C,ADP		Jump if Sense Switch off or overflow set: SS=0 ∨ OV=1	1	2-36
	2600*	Forward jump		
	2640*	Backward jump		
JOC :0D,ADR		Jump if Sense Switch off or overflow set or A negative: SS=0 ∨ OV=1 ∨ (A) < 0	1	2-36
	2680*	Forward jump		
	26D0*	Backward jump		

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOC :0E,ADR		Jump if Sense Switch off or overflow set or A equals zero: SS=0 ∨ OV=1 ∨ (A)=0	1	2-36
	2700*	Forward jump		
	2740*	Backward jump		
JOC :0F,ADR		Jump if Sense Switch off or overflow set or A less than or equal to zero SS=0 ∨ OV=1 ∨ (A) ≤ 0	1	2-36
	2780*	Forward jump		
	27C0*	Backward jump		
JXZ (JOC :10,		Jump if X equal to zero: (X)=0	1	2-38
ADR)	2800*	Forward jump		
	2840*	Backward jump		
JOC :11,ADR		Jump if X equals zero or A negative (X)=0 ∨ (A) < 0	1	2-36
	2880*	Forward jump		
	28C0*	Backward jump		
JOC :12,ADR	2900*	Illegal combination (includes (X)=0 ∨ (A)=0)		2-36
	2940*			
JOC :13,ADR	2980*	Illegal combination (includes (X)=0 ∨ (A)=0)		2-36
	29C0*			
JOC :14,ADP		Jump if X equals zero or overflow set: (X)=0 ∨ OV=1	1	2-36
	2A00*	Forward jump		
	2A40*	Backward jump		
JOC :15,ADR		Jump if X equals zero or overflow set or A negative (X)=0 ∨ OV=1 ∨ (A) < 0	1	2-36
	2A80*	Forward jump		
	2AC0*	Backward jump		
JOC :16,ADR	2B00*	Illegal combinations (include (X)=0 ∨ (A)=0)		2-36
	2B40*			
JOC :17	2B80*	Illegal combinations (include (X)=0 ∨ (A)=0)		2-36
	2BC0*			
JOC :18,ADR		Jump if X equals zero or Sense Switch off: (X)=0 ∨ SS=0	1	2-36
	2C00*	Forward jump		
	2C40*	Backward jump		
JOC :19,ADR		Jump if X equals zero or Sense Switch off or A is negative: (X)=0 ∨ SS=0 ∨ (A) < 0	1	2-36
	2C80*	Forward jump		
	2CC0*	Backward jump		
JOC :1A,ADR	2D00*	Illegal combination (includes (X)=0 ∨ (A)=0)		2-36
	2D40*			

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOC :1B,ADR	2D80*	Illegal combination (includes $(X)=0 \vee (A)=0$)		2-36
	2DC0*			
JOC :1C,ADR		Jump if X equals zero or Sense Switch off or overflow set: $(X)=0 \vee SS=0 \vee OV=1$	1	2-36
	2E00*	Forward jump		
	2E40*	Backward jump		
JOC :1D,ADR		Jump if X equals zero or Sense Switch off or overflow set or A negative: $(X)=0 \vee SS=0 \vee OV=1 \vee (A) < 1$	1	2-36
	2E80*	Forward jump		
	2EC0*	Backward jump		
JOC :1E,ADR	2F00*	Illegal combination (includes $(X)=0 \vee (A)=0$)		2-36
	2F40*			
JOC :1F,ADR	2F80*	Illegal combination (includes $(X)=0 \vee (A)=0$)		2-36
	2FC0*			
JAP (JOC :21,		Jump if A positive or equal to zero: $(A) \geq 0$	1	2-36
ADR)	3080*	Forward jump		
	30C0*	Backward jump		
JAN (JOC :22,		Jump if A not zero: $(A) \neq 0$	1	2-37
ADR)	3100*	Forward jump		
	3140*	Backward jump		
JAG (JOC :23,		Jump if A positive and not equal to zero: $(A)>0$	1	2-37
ADR)	3180*	Forward jump		
	31C0*	Backward jump		
JOR (JOC :24,		Jump if overflow reset: $OV=0$	1	2-39
ADR)	3200*	Forward jump		
	3240*	Backward jump		
JOC :25,ADR		Jump if A positive and overflow is reset: $(A) \geq 0 \wedge OV=0$	1	2-36
	3280*	Forward jump		
	32C0*	Backward jump		
JOC :26,ADR		Jump if A non-zero and overflow reset: $(A) \neq 0 \wedge OV=0$	1	2-36
	3300*	Forward jump		
	3340*	Backward jump		
JOC :27,ADR		Jump if A greater than zero and overflow reset: $(A)>0 \wedge OV=0$	1	2-36
	3380*	Forward jump		
	33C0*	Backward jump		

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JSS (JOC :28,		Jump if Sense Switch on: SS=1	1	2-39
ADR)	3400*	Forward jump		
	3440*	Backward jump		
JOC :29,ADR		Jump if Sense Switch on and A positive: $SS=1 \wedge (A) \geq 0$	1	2-36
	3480*	Forward jump		
	34C0*	Backward jump		
JOC :2A,ADR		Jump if Sense Switch on and A non-zero: $SS=1 \wedge (A) \neq 0$	1	2-36
	3500*	Forward jump		
	3540*	Backward jump		
JOC :2B,ADR		Jump if Sense Switch on and A greater than zero: $SS=1 \wedge (A) > 0$	1	2-36
	3580*	Forward jump		
	35C0*	Backward jump		
JOC :2C,ADP		Jump if Sense Switch on and overflow reset: $SS=1 \wedge OV=0$	1	2-36
	3600*	Forward jump		
	3640*	Backward jump		
JOC :2D,ADR		Jump if Sense Switch on and A positive and overflow reset: $SS=1 \wedge (A) \geq 0 \wedge OV=0$	1	2-36
	3680*	Forward jump		
	36C0	Backward jump		
JOC :2E,ADR		Jump if Sense Switch on and A non-zero and overflow reset: $SS=1 \wedge (A) \neq 0 \wedge OV=0$	1	2-36
	3700*	Forward jump		
	3740*	Backward jump		
JOC :2F,ADR		Jump if A greater than zero and Sense Switch on and overflow reset $(A) \neq 0 \wedge SS \neq 1 \wedge OV=0$	1	2-36
	3780*	Forward jump		
	37C0*	Backward jump		
JXN (JOC :30,		Jump if X non-zero: $(X) \neq 0$	1	2-38
ADR)	3800*	Forward jump		
	3840*	Backward jump		
JOC :31,ADR		Jump if X non-zero and A positive: $(X) \neq 0 \wedge (A) \geq 0$	1	2-36
	3880*	Forward jump		
	38C0*	Backward jump		
JOC :32,ADR		Jump if X non-zero and A non-zero: $(X) \neq 0 \wedge (A) \neq 0$	1	2-36
	3900*	Forward jump		
	3940*	Backward jump		

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOC :33,ADR		Jump if X non-zero and A greater than zero: $(X)\neq 0 \wedge (A)>0$	1	2-36
	3980*	Forward jump		
	39C0*	Backward jump		
JOC :34,ADR		Jump if X non-zero and overflow reset: $(X)\neq 0 \wedge OV=0$	1	2-36
	3A00*	Forward jump		
	3A40*	Backward jump		
JOC :35,ADR		Jump if X non-zero and A positive and overflow reset: $(X)\neq 0 \wedge (A)\geq 0 \wedge OV=0$	1	2-36
	3A80*	Forward jump		
	3AC0*	Backward jump		
JOC :36,ADR		Jump if X non-zero and A non-zero and overflow reset: $(X)\neq 0 \wedge (A)\neq 0 \wedge OV=0$	1	2-36
	3B00*	Forward jump		
	3B40*	Backward jump		
JOC :37,ADR		Jump if X non-zero and A greater than zero and overflow reset: $(X)\neq 0 \wedge (A)>0 \wedge OV=0$	1	2-36
	3B80*	Forward jump		
	3BC0*	Backward jump		
JOC :38,ADR		Jump if X non-zero and Sense Switch on: $(X)\neq 0 \wedge SS=1$	1	2-36
	3C00*	Forward jump		
	3C40*	Backward jump		
JOC :39,ADR		Jump if X non-zero and A positive and Sense Switch on: $(X)\neq 0 \wedge (A)\geq 0 \wedge SS=1$	1	2-36
	3C80*	Forward jump		
	3CC0*	Backward jump		
JOC :3A,ADR		Jump if X non-zero and A non-zero and Sense Switch on: $(X)\neq 0 \wedge (A)\neq 0 \wedge SS=1$	1	2-36
	3D00*	Forward jump		
	3D40*	Backward jump		
JOC :3B,ADR		Jump if X non-zero and A greater than zero and Sense Switch on: $(X)\neq 0 \wedge (A)>0 \wedge SS=1$	1	2-36
	3D80*	Forward jump		
	3DC0*	Backward jump		
JOC :3C,ADR		Jump if X non-zero and Sense Switch on and overflow reset: $(X)\neq 0 \wedge SS=1 \wedge OV=0$	1	2-36
	3E00*	Forward jump		
	3E40*	Backward jump		

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
JOC :3D,ADR		Jump if X non-zero and A positive and Sense Switch on and overflow reset: $(X) \neq 0 \wedge (A) \geq 0 \wedge SS=1 \quad OV=0$	1	2-36
	3E80*	Forward jump		
	3EC0*	Backward jump		
JOC :3E,ADR		Jump if X non-zero and A non-zero and Sense Switch on and overflow reset: $(X) \neq 0 \wedge (A) \neq 0 \wedge SS=1 \wedge OV=0$	1	2-36
	3F00*	Forward jump		
	3F40*	Backward jump		
JOC :3F,ADR		Jump if X non-zero and A greater than zero and Sense Switch on and overflow reset: $(X) \neq 0 \wedge (A) > 0 \wedge SS=1 \wedge OV=0$	1	2-36
	3F80*	Forward jump		
	3FC0*	Backward jump		
SEL	4000*	Select function	1-1/4	3-18
MPE	4000	Memory Protect enable	1-1/4	2-73
MPD	4001	Memory Protect disable	1-1/4	2-73
PFE	4002	Power Fail interrupt enable	1-1/4	2-72
PFD	4003	Power Fail interrupt disable	1-1/4	2-72
CIE	4005	Console interrupt enable	1-1/4	2-71
CID	4006	Console interrupt disable	1-1/4	2-72
TRP	4007	Trap	1-1/4	2-69
RAM	4045	Set Random Access mode	1-1/4	2-74
ROM	4046	Set Read Only mode	1-1/4	2-74
SEA	4400*	Select and present A	1-1/4	3-18
SEX	4600*	Select and present X	1-1/4	3-18
SSN	4800*	Sense and skip on no response	1-1/4	3-17
SEN	4900*	Sense and skip on response	1-1/4	3-17
AIN	5000*	Automatic Input: Word	4-1/4	3-30
AIB	5400*	Automatic Input: Byte	4-1/2	3-31
INA	5800*	Input word to A (unconditionally)	1-1/4	3-19
SIA	5800	Status input to A	1-1/4	2-70
ISA	5801	Input data switches to A	1-1/4	2-66
RDA	5900*	Read word to A	1-1/4	3-20
INX	5A00*	Input word to X (unconditionally)	1-1/4	3-19
SIX	5A00	Status input to X	1-1/4	2-70
RDX	5B00*	Read word to X	1-1/4	3-20
ISX	5B01	Input data switches to X	1-1/4	2-66
INAM	5C00*	Input word to A, masked (unconditionally)	1-1/4	3-20

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
RDAM	5D00*	Read word to A, masked	1-1/4	3-21
INXM	5E00*	Input word to X, masked (unconditionally)	1-1/4	3-20
RDXM	5F00*	Read word to X, masked	1-1/4	3-21
AOT	6000*	Automatic Output: Word	4-1/4	3-30
AOB	6400*	Automatic Output: Byte	4-1/4	3-31
SIN	6800	Status Inhibit	1-1/4	2-69
OTZ	6800*	Output zero (unconditionally)	1-1/4	3-22
WRZ	6900*	Write zeros	1-1/4	3-23
OTA	6C00*	Output A Register (unconditionally)	1-1/4	3-22
SOA	6C00	Status output from A	1-1/4	2-71
WRA	6D00*	Write from A	1-1/4	3-23
OTX	6E00*	Output X register (unconditionally)	1-1/4	3-22
SOX	6E00	Status output from X	1-1/4	2-71
WRX	6F00*	Write from X	1-1/4	3-23
BIN	7100*	Block input to memory	2+1-1/2w	3-25
BOT	7500*	Block output from memory	2+1-1/2w	3-26
IBA	7800*	Input byte to A (unconditionally)	1-1/4	3-19
RBA	7900*	Read byte to A	1-1/4	3-21
IBX	7A00*	Input byte to X (unconditionally)	1-1/4	3-19
RBX	7B00*	Read byte to X	1-1/4	3-21
IBAM	7C00*	Input byte to A, masked (unconditionally)	1-1/4	3-20
RBAM	7D00*	Read byte to A, masked	1-1/4	3-21
IBXM	7E00*	Input byte to X, masked (unconditionally)	1-1/4	3-20
RBXM	7F00*	Read byte to X, masked	1-1/4	3-21
AND	8000*	AND to A direct, scratchpad	2	2-14
AND	8100*	AND to A indirect, AP in scratchpad	2+1n	2-14
AND	8200*	AND to A relative to P forward, direct	2	2-14
AND	8300*	AND to A relative to P forward, indirect	2+1n	2-14
AND	8400*	AND to A indexed, direct	2	2-14
AND	8500*	AND to A indexed, indirect	2+1n	2-14
AND	8600*	AND to A relative to P backward, direct	2	2-14
AND	8700*	AND to A relative to P backward, indirect	2+1n	2-14
ANDB	:8000*	AND to A byte, direct, scratchpad	2	2-28
ANDB	:8100*	AND to A byte, indirect, AP in scratchpad	3	2-28
ANDB	:8200*	AND to A byte 0, direct, relative to P forward	2	2-28
ANDB	:8300*	AND to A byte, indirect, AP relative to P forward	3	2-28
ANDB	:8400*	AND to A byte, indexed, direct	2	2-28
ANDB	:8500*	AND to A byte, indexed, indirect, AP in scratchpad	3	2-28

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
ANDB	:8600*	AND to A byte 1, direct, relative to P forward	2	2-28
ANDB	:8700*	AND to A byte, indirect, AP relative to P backward	3	2-28
ADD	8800*	Add to A direct, scratchpad	2	2-10
ADD	8900*	Add to A indirect, AP in scratchpad	2+1n	2-10
ADD	8A00*	Add to A relative to P forward, direct	2	2-10
ADD	8B00*	Add to A relative to P forward, indirect	2+1n	2-10
ADD	8C00*	Add to A indexed, direct	2	2-10
ADD	8D00*	Add to A indexed, indirect	2+1n	2-10
ADD	8E00*	Add to A relative to P backward, direct	2	2-10
ADD	8F00*	Add to A relative to P backward, indirect	2+1n	2-10
ADDB	:8800*	Add byte, direct, scratchpad	2	2-24
ADDB	:8900*	Add byte, indirect, AP in scratchpad	3	2-24
ADDB	:8A00*	Add byte 0, relative to P forward, direct	2	2-24
ADDB	:8B00*	Add byte, indirect, AP relative to P, forward	3	2-24
ADDB	:8C00*	Add byte, direct, indexed	2	2-24
ADDB	:8D00*	Add byte, indirect, indexed, AP in scratchpad	3	2-24
ADDB	:8E00*	Add byte 1, relative to P forward, direct	2	2-24
ADDB	:8F00*	Add byte, indirect, relative to P, backward	3	2-24
SUB	9000*	Subtract from A; direct, scratchpad	2	2-11
SUB	9100*	Subtract from A; indirect, AP in scratchpad	2+1n	2-11
SUB	9200*	Subtract from A; relative to P forward, direct	2	2-11
SUB	9300*	Subtract from A; relative to P forward, indirect	2+1n	2-11
SUB	9400*	Subtract from A; indexed, direct	2	2-11
SUB	9500*	Subtract from A; indexed, indirect	2+1n	2-11
SUB	9600*	Subtract from A; relative to P backward direct	2	2-11
SUB	9700*	Subtract from A; relative to P backward, indirect	2+1n	2-11
SUBB	:9000*	Subtract byte, direct, scratchpad	2	2-24
SUBB	:9100*	Subtract byte, indirect, AP in scratchpad	3	2-24
SUBB	:9200*	Subtract byte 0, direct, relative to P forward	2	2-24
SUBB	:9300*	Subtract byte, indirect, AP relative to P forward	3	2-24
SUBB	:9400*	Subtract byte, indexed, direct	2	2-24
SUBB	:9500*	Subtract byte, indirect, indexed, AP in scratchpad	3	2-24
SUBB	:9600*	Subtract byte 1, direct, relative to P forward	2	2-24
SUBB	:9700*	Subtract byte, indirect, relative to P backward	3	2-24
STA	9800*	Store A, direct, scratchpad	2	2-13
STA	9900*	Store A; indirect, AP in scratchpad	2+1n	2-13
STA	9A00*	Store A; relative to P forward, direct	2	2-13
STA	9B00*	Store A; relative to P forward, indirect	2+1n	2-13

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
STA	9C00*	Store A; indexed, direct	2	2-13
STA	9D00*	Store A; indexed, indirect	2+1n	2-13
STA	9E00*	Store A; relative to P backward, direct	2	2-13
STA	9F00*	Store A; relative to P backward, indirect	2+1n	2-13
STAB	:9800*	Store A byte, direct, scratchpad	2	2-26
STAB	:9900*	Store A byte, indirect, AP in scratchpad	3	2-26
STAB	:9A00*	Store A byte 0, direct, relative to P forward	2	2-26
STAB	:9B00*	Store A byte, indirect, AP relative to P forward	3	2-26
STAB	:9C00*	Store A byte, indexed, direct	2	2-26
STAB	:9D00*	Store A byte, indexed, indirect, AP in scratchpad	3	2-26
STAB	:9E00*	Store A byte 1, direct, relative to P forward	2	2-26
STAB	:9F00*	Store A byte, indirect, AP relative to P backward	3	2-26
IOR	A000*	Inclusive OR to A; direct, scratchpad	2	2-14
IOR	A100*	Inclusive OR to A; indirect, AP in scratchpad	2+1n	2-14
IOR	A200*	Inclusive OR to A; relative to P forward, direct	2	2-14
IOR	A300*	Inclusive OR to A; relative to P forward, indirect	2+1n	2-14
IOR	A400*	Inclusive OR to A; indexed, direct	2	2-14
IOR	A500*	Inclusive OR to A; indexed, indirect	2+1n	2-14
IOR	A600*	Inclusive OR to A; relative to P backward, direct	2	2-14
IOR	A700*	Inclusive OR to A; relative to P backward, indirect	2+1n	2-14
IORB	:A000*	Inclusive OR byte, direct, scratchpad	2	2-28
IORB	:A100*	Inclusive OR byte, indirect, AP in scratchpad	3	2-28
IORB	:A200*	Inclusive OR byte 0, direct, relative to P forward	2	2-28
IORB	:A300*	Inclusive OR byte, indirect, AP relative to P forward	3	2-28
IORB	:A400*	Inclusive OR byte, indexed, direct	2	2-28
IORB	:A500*	Inclusive OR byte, indexed, indirect, AP in scratchpad	3	2-28
IORB	:A600*	Inclusive OR byte 0, direct, relative to P forward	2	2-28
IORB	:A700*	Inclusive OR byte, indirect, AP relative to P backward	3	2-28
XOR	A800*	Exclusive OR to A; direct, scratchpad	2	2-15
XOR	A900*	Exclusive OR to A; indirect, AP in scratchpad	2+1n	2-15
XOR	AA00*	Exclusive OR to A; relative to P forward, direct	2	2-15
XOR	AB00*	Exclusive OR to A; relative to P forward, indirect	2+1n	2-15
XOR	AC00*	Exclusive OR to A; indexed, direct	2	2-15
XOR	AD00*	Exclusive OR to A; indexed, indirect	2+1n	2-15
XOR	AE00*	Exclusive OR to A; relative to P backward, direct	2	2-15
XOR	AF00*	Exclusive OR to A; relative to P backward, indirect	2+1n	2-15
XORB	:A800*	Exclusive OR byte, direct, scratchpad	2	2-29

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
XORB	:A900*	Exclusive OR byte, indirect, AP in scratchpad	3	2-29
XORB	:AA00*	Exclusive OR byte 0, direct, relative to P forward	2	2-29
XORB	:AB00*	Exclusive OR byte, indirect, AP relative to P forward	3	2-29
XORB	:AC00*	Exclusive OR byte, indexed, direct	2	2-29
XORB	:AD00*	Exclusive OR byte, indexed, indirect, AP in scratchpad	3	2-29
XORB	:AE00*	Exclusive OR byte 1, direct, relative to P forward	2	2-29
XORB	:AF00*	Exclusive OR byte, indirect, AP relative to P backward	3	2-29
LDA	B000*	Load A; direct, scratchpad	2	2-12
LDA	B100*	Load A; indirect, AP in scratchpad	2+1n	2-12
LDA	B200*	Load A; relative to P forward, direct	2	2-12
LDA	B300*	Load A; relative to P forward, indirect	2+1n	2-12
LDA	B400*	Load A indexed, direct	2	2-12
LDA	B500*	Load A; indexed, indirect	2+1n	2-12
LDA	B600*	Load A; relative to P backward, direct	2	2-12
LDA	B700*	Load A; relative to P backward, indirect	2+1n	2-12
LDAB	:B000*	Load A byte, direct, scratchpad	2	2-25
LDAB	:B100*	Load A byte, indirect, AP in scratchpad	3	2-25
LDAB	:B200*	Load A byte 0, direct, relative to P forward	2	2-25
LDAB	:B300*	Load A byte, indirect, AP relative to P forward	3	2-25
LDAB	:B400*	Load A byte, indexed, direct	2	2-25
LDAB	:B500*	Load A byte, indexed, indirect, AP in scratchpad	3	2-25
LDAB	:B600*	Load A byte 1, direct, relative to P forward	2	2-25
LDAB	:B700*	Load A byte, indirect, AP relative to P backward	3	2-25
EMA	B800*	Exchange memory and A; direct, scratchpad	2	2-13
EMA	B900*	Exchange memory and A; indirect, AP in scratchpad	2+1n	2-13
EMA	BA00*	Exchange memory and A; relative to P forward, direct	2	2-13
EMA	BB00*	Exchange memory and A; relative to P forward, indirect	2+1n	2-13
EMA	BC00*	Exchange memory and A; indexed, direct	2	2-13
EMA	BD00*	Exchange memory and A; indexed, indirect	2+1n	2-13
EMA	BE00*	Exchange memory and A; relative to P backward, direct	2	2-13
EMA	BF00*	Exchange memory and A;relative to P backward,indirect	2+1n	2-13
EMAB	:B800*	Exchange Memory and A byte, direct, scratchpad	2	2-27
EMAB	:B900*	Exchange Memory and A byte, indirect, AP in scratchpad	3	2-27
EMAB	:BA00*	Exchange Memory and A byte 0, direct, relative to P forward	2	2-27
EMAB	:BB00*	Exchange Memory and A byte, indirect, AP relative to P forward	3	2-27

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
EMAB	:BC00*	Exchange Memory and A byte, indexed, direct	2	2-27
EMAB	:BD00*	Exchange Memory and A byte, indexed, indirect, AP in scratchpad	3	2-27
EMAB	:BE00*	Exchange Memory and A byte 1, direct, relative to P forward	2	2-27
EMAB	:BF00*	Exchange Memory and A, indirect, AP relative to P backward	3	2-27
CAI	C000*	Compare to A immediate	1	2-33
CXI	C100*	Compare to X immediate	1	2-33
AXI	C200*	Add to X immediate	1	2-31
SXI	C300*	Subtract from X immediate	1	2-31
LXP	C400*	Lead X positive immediate	1	2-31
LXM	C500*	Lead X minus immediate	1	2-33
LAP	C600*	Load A positive immediate	1	2-32
LAM	C700*	Load A minus immediate	1	2-32
SCN	CD00*	Scan memory, indexed, indirect	2+1w	2-16
CMS	D000*	Compare memory to A and skip if high or equal; direct, scratchpad	2	2-15
CMS	D100*	Compare memory to A and skip if high or equal; indirect, AP in scratchpad	2+1n	2-15
CMS	D200*	Compare memory to A and skip if high or equal; relative to P forward, direct	2	2-15
CMS	D300*	Compare memory to A and skip if high or equal; relative to P forward, indirect	2+1n	2-15
CMS	D400*	Compare memory to A and skip if high or equal; indexed, direct	2	2-15
CMS	D500*	Compare memory to A and skip if high or equal; indexed, indirect	2+1n	2-15
CMS	D600*	Compare memory to A and skip if high or equal; relative to P backward, direct	2	2-15
CMS	D700*	Compare memory to A and skip if high or equal; relative to P backward, indirect	2+1n	2-15
CMSB	:D000*	Compare byte and skip if high or equal; direct, scratchpad	2	2-29
CMSB	:D100*	Compare byte and skip if high or equal, indirect, AP in scratchpad	3	2-29
CMSB	:D200*	Compare byte 0 and skip if high or equal, direct, relative to P forward	2	2-29

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
CMSB	:D300*	Compare byte and skip if high or equal, indirect, AP relative to P forward	3	2-29
CMSB	:D400*	Compare byte and skip if high or equal, indexed, direct	2	2-29
CMSB	:D500*	Compare byte and skip if high or equal, indexed, indirect, AP in scratchpad	3	2-29
CMSB	:D600*	Compare byte 1 and skip if high or equal, direct, relative to P forward	2	2-29
CMSB	:D700*	Compare byte and skip if high or equal, indirect, AP relative to P backward	3	2-29
IMS	D800*	Increment memory and skip on zero result; direct, scratchpad	2	2-11
IMS	D900*	Increment memory and skip on zero result; indirect, AP in scratchpad	2+1n	2-11
IMS	DA00*	Increment memory and skip on zero result; relative to P forward, direct	2	2-11
IMS	DB00*	Increment memory and skip on zero result; relative to P forward, indirect	2+1n	2-11
IMS	DC00*	Increment memory and skip on zero result; indexed, direct	2	2-11
IMS	DD00*	Increment memory on skip on zero result; indexed, indirect	2+1n	2-11
IMS	DE00*	Increment memory and skip on zero result; relative to P backward, direct	2	2-11
IMS	DF00*	Increment memory and skip on zero result, relative to P backward, indirect	2+1n	2-11
LDX	E000*	Load X; direct, scratchpad	2	2-12
LDX	E100*	Load X; indirect, AP in scratchpad	2+1n	2-12
LDX	E200*	Load X; relative to P forward, direct	2	2-12
LDX	E300*	Load X; relative to P forward, indirect	2+1n	2-12
LDX	E400*	Load X; indexed, direct	2	2-12
LDX	E500*	Load X; indexed, indirect	2+1n	2-12
LDX	E600*	Load X; relative to P backward, direct	2	2-12
LDX	E700*	Load X; relative to P backward, indirect	2+1n	2-12
LDXB	:E000*	Load X byte, direct, scratchpad	2	2-25
LDXB	:E100*	Load X byte, indirect, AP in scratchpad	3	2-25
LDXB	:E200*	Load X byte 0, direct, relative to P forward	2	2-25
LDXB	:E300*	Load X byte, indirect, relative to P forward	3	2-25
LDXB	:E400*	Load X byte, indexed, direct	2	2-25

INSTRUCTION SET, NUMERICAL ORDER

Instruction Mnemonic	Instruction Code in Hex	Description	Cycles	Page
LDXB	:E500*	Load X, indexed, indirect, AP in scratchpad	3	2-25
LDXB	:E600*	Load X byte 1, direct, relative to P forward	2	2-25
LDXB	:E700*	Load X byte, indirect, relative to P backward	3	2-25
STX	E800*	Store X; direct, scratchpad	2	2-13
STX	E900*	Store X; indirect, AP in scratchpad	2+1n	2-13
STX	EA00*	Store X; relative to P forward, direct	2	2-13
STX	EB00*	Store X; relative to P forward, indirect	2+1n	2-13
STX	EC00*	Store X; indexed, direct	2	2-13
STX	ED00*	Store X; indexed, indirect	2+1n	2-13
STX	EE00*	Store X; relative to P backward, direct	2	2-13
STX	EF00*	Store X; relative to P backward, indirect	2+1n	2-13
STXB	:E800*	Store X byte, direct, scratchpad	2	2-26
STXB	:E900*	Store X byte, indirect, AP in scratchpad	3	2-26
STXB	:EA00*	Store X byte 0, direct, relative to P forward	2	2-26
STXB	:EB00*	Store X byte, indirect, relative to P forward	3	2-26
STXB	:EC00*	Store X byte, indexed, direct	2	2-26
STXB	:ED00*	Store X byte, indexed, indirect, AP	3	2-26
STXB	:EE00*	Store X byte 1, direct, relative to P forward	2	2-26
STXB	:EF00*	Store X byte, indirect, relative to P backward	3	2-26
JMP	F000*	Jump unconditionally; direct, scratchpad	1	2-18
JMP	F100*	Jump unconditionally; indirect, AP in scratchpad	2	2-18
JMP	F200*	Jump unconditionally; relative to P forward, direct	1	2-18
JMP	F300*	Jump unconditionally; relative to P forward, indirect	2	2-18
JMP	F400*	Jump unconditionally; indexed, direct	1	2-18
JMP	F500*	Jump unconditionally; indexed, indirect	2	2-18
JMP	F600*	Jump unconditionally; relative to P backward, direct	1	2-18
JMP	F700*	Jump unconditionally; relative to P backward, indirect	2	2-18
JST	F800*	Jump and Store; direct, scratchpad	2	2-18
JST	F900*	Jump and Store; indirect, AP in scratchpad	3	2-18
JST	FA00*	Jump and Store; relative to P forward, direct	2	2-18
JST	FB00*	Jump and Store; relative to P forward, indirect	3	2-18
JST	FC00*	Jump and Store; indexed, direct	2	2-18
JST	FD00*	Jump and Store; indexed, indirect	3	2-18
JST	FE00*	Jump and Store; relative to P backward, direct	2	2-18
JST	FF00*	Jump and Store; relative to P backward, indirect	3	2-18