PROGRAMMER'S REFERENCE MANUAL

483

BIT INCORPORATED, 5 STRATHMORE ROAD, NATICK, MASSACHUSETTS

The BIT 483 Computer Programmer's Reference Manual
was compiled and written by the Programming staff
of BIT, Incorporated.

First Printing:
January, 1970

Printed in U.S.A.

Document No. 48306

contents

FIGURES

TABLES

section one

introduction

SCOPE

This manual covers the programming operation and capabilities of the BIT 483 Computer. This is a compact, high-speed, solid-state, variable word-length computer, featuring an expandable memory, simultaneous input/output, and an extensive system of interrupts. It is constructed on the total modularity principle.

A 350 nanosecond (ns) memory access time, coupled with a cycle-stealing capability, is standard. Multilevel priority interrupts permit relatively simple handling of multiple terminal devices. Combined with a single peripheral transfer instruction these features make the BIT 483 a singularly efficient communicator.

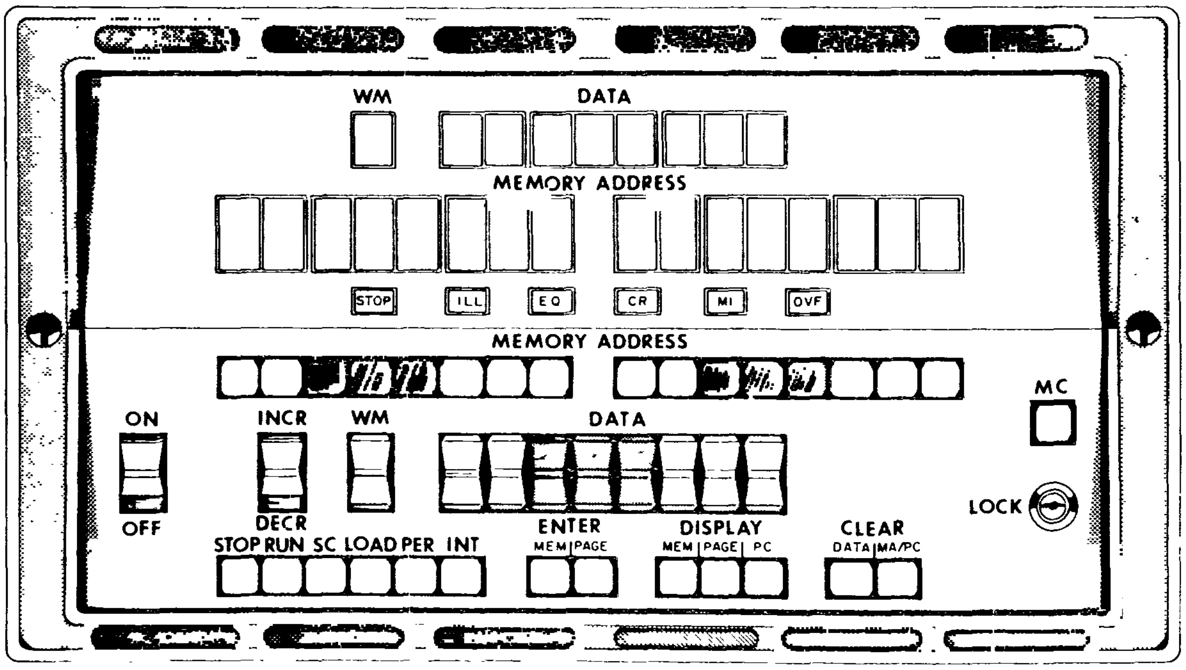


Figure 1-1. BIT 483 Computer, Front Panel.

SPECIFICATIONS

The specifications and important characteristics of the BIT 483 are listed below.

A.	Memory capacity	Variable from 1024 bytes to 65,536 bytes	
B.	Memory cycle time	980 ns	
C.	Word size	Variable, multiples of 8-bit bytes	
D.	Page size	256 bytes	
E.	Arithmetic	Binary (signed) Binary (unsigned) Decimal (signed)	
F.	Standard data channels (buffered)	C G	1 teletypewriter 32 peripheral devices (if no A channel)
	Optional data channel (buffered)	A	16 peripheral devices
G.	Maximum output data rate	8.16 million bits/sec	
H.	Maximum input data rate	6 million bits/sec	
I.	Channel modes (under program control)	Overlapped (cycle stealing), non-overlapped	
J.	Teletype modes (under program control)	Buffered, non-interrupt mode; buffered, interrupt mode	
K.	Priority interrupts	Eight basic (can be increased in groups of 8 up to 32)	
L.	Sense switches	Nine sense switches	
M.	Memory protection	Contents of memory protected during power interruption	
N.	Primary power	115/230V, ±10%, 50/60 Hz	
O.	Power consumption	500 watts	
P.	Weight	Approximately 50 pounds	
Q.	Dimensions	10½ inches high X 19½ inches wide X 23 inches in length	
R.	Environmental temperature	0 to 50°C	
S.	Relative humidity	95% maximum	
T.	Mounting capabilities	Standard 19-inch rack, pedestal, or table top	
U.	Circuit technology	Medium scale integration	

ORGANIZATION OF EQUIPMENT

The use of standard modular elements in a general-purpose, solid-state, stored-program computer such as the BIT 483 permits flexible organization and expansion of equipment to satisfy a wide variety of computer-installation requirements. This is shown in Figure 1-2. Any number of modules may be used in a 483 system. Memory expansion does not affect existing programs because system operation is controlled by an internally stored program.

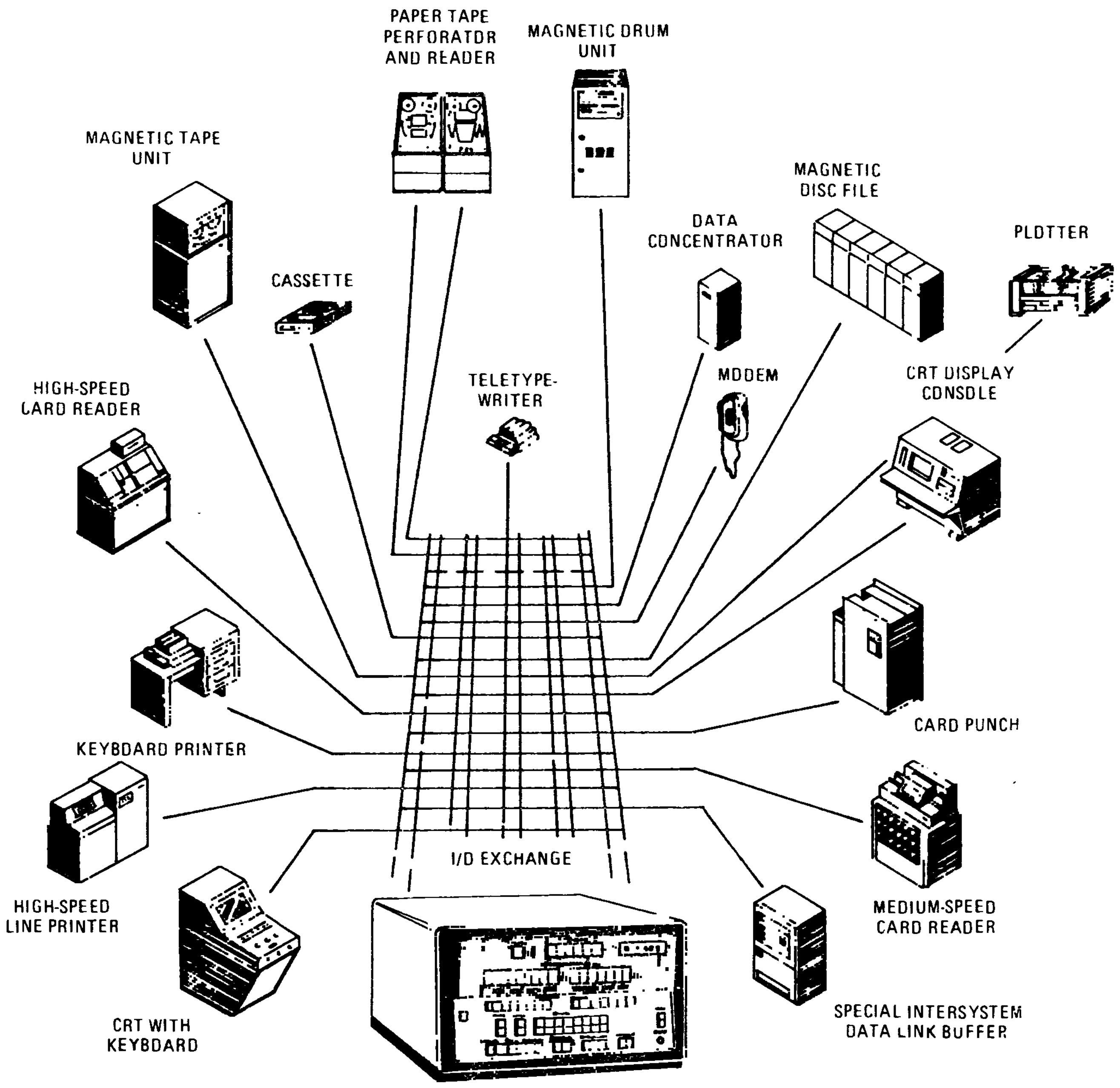


Figure 1-2. Equipment Configurations Possible with BIT 483 Computer

FUNCTIONAL CHARACTERISTICS

The major functional characteristics of the BIT 483 Computer, briefly described, are as follows:

Internally Stored Program

Operation of the 483 is controlled by an internally stored program located in memory. A program consists of the series of instructions and related operational data and constraints required for performing a particular job.

Fully-Shared Memory

The central processor and I/O network have direct access to the totally-shared core memory, permitting the computing and I/O elements to communicate directly with all data. Partially-shared memory schemes may also be implemented to satisfy the requirements of diverse system applications.

Instruction Repertoire

The instruction repertoire provided for the BIT 483 includes instructions for signed decimal and signed or unsigned binary arithmetic, character and field operations, logical and control functions, varied data manipulations, conditional jump operations, and subroutine control.

Variable Word Length

The variable word length capability in the BIT 483 enables the user to tailor the word length to the accuracy requirements of a problem. Multiple precision arithmetic, involving subroutines and additional memory space, is never required. The 483 processor performs 8-bit, 16-bit, or even 2048-bit arithmetic with equal facility, requiring only a single instruction.

In addition to possessing variable word length, each binary arithmetic or logical instruction is bidirectional. Thus, either of the possibilities

$$\text{(Storage)} + \text{(A)} \rightarrow \text{A}$$
$$\text{(Storage)} + \text{(A)} \rightarrow \text{Storage}$$

is selectable by means of a control bit in the instructions. Four indicators: EQ (equal to zero), GR (greater than zero), MI (minus indicator), and OVF (overflow), are set during every binary arithmetic and logical instruction. These indicators may be interrogated as a condition state for subsequent jump instructions. Figure 1-3 illustrates some of the possibilities inherent in the 483 variable word-length capability. In each example, only a single instruction is required.

Word Size

A word in the BIT 483 processor is any integral number of bytes as shown below.

1. Unsigned Binary Subtract two 24 bit numbers (result in accumulator)
Instruction: BSA X

INDICATORS

				EQ	GR	MI	OVF
ACC	1 0 0 1 1 1 0 0	0 1 1 1 1 0 0 1	0 0 0 0 1 0 1 0				
X	0 1 1 1 1 1 1 0	1 0 0 0 0 1 0 0	0 1 0 1 0 0 1 1				
ACC RESULT	0 0 0 1 1 1 0 1	1 1 1 1 0 1 0 0	1 0 1 1 0 1 1 1	$\boxed{0}$	$\boxed{1}$	$\boxed{0}$	$\boxed{0}$

2. Signed Binary Subtract two 16 bit numbers (result in accumulator)
Instruction: BSA Y

			EQ	GR	MI	OVF
ACC	1 0 0 1 1 1 0 0	0 1 1 1 1 0 0 1				
Y	0 0 0 0 1 1 1 0	1 0 0 0 0 1 0 0				
ACC RESULT	1 0 0 0 1 1 0 1	1 1 1 1 0 1 0 1	$\boxed{0}$	$\boxed{1}$	$\boxed{1}$	$\boxed{0}$

3. Signed Binary Add two 24 bit numbers (result in memory)
Instruction: BAM X

				EQ	GR	MI	OVF
	1 1 0 0 1 1 0 0	1 1 1 1 0 0 0 0	1 0 1 0 1 0 1 0				
ACC	1 1 1 1 0 0 0 0	1 1 0 0 1 1 0 0	0 1 0 1 0 1 0 1				
X RESULT	1 0 1 1 1 1 0 1	1 0 1 1 1 1 0 0	1 1 1 1 1 1 1 1	$\boxed{0}$	$\boxed{1}$	$\boxed{1}$	$\boxed{0}$

4. Decimal Add two 6 digit decimal numbers (result in accumulator)
Instruction: DAA X

							EQ	GR	MI	OVF
ACC	6	1	9	.	3	2				
X	2	3	1	.	7	8				
ACC RESULT	8	5	1	.	1	0	$\boxed{0}$	$\boxed{1}$	$\boxed{*}$	$\boxed{*}$

5. Decimal Subtract two 4 digit decimal numbers (result in accumulator)
Instruction: DSA Y

					EQ	GR	MI	OVF
ACC	4	3	2	1				
Y	9	8	7	6				
ACC RESULT	5	5	5	(–) 5	$\boxed{0}$	$\boxed{0}$	$\boxed{*}$	$\boxed{*}$

6. Exclusive OR two 16 bit numbers (result in memory)
Instruction: ORM X

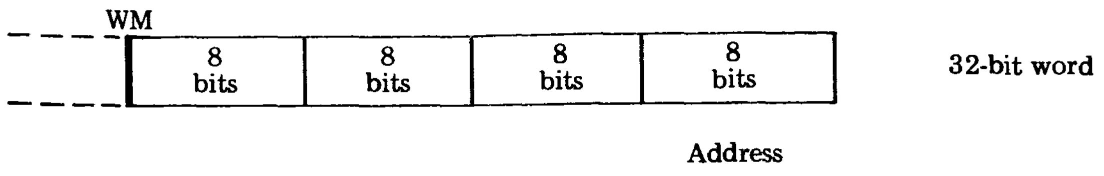
			EQ	GR	MI	OVF
X	1 1 0 0 1 1 0 0	1 1 1 1 0 0 0 0				
ACC	1 1 1 1 0 0 0 0	1 1 0 0 1 1 0 0				
X RESULT	0 0 1 1 1 1 0 0	0 0 1 1 1 1 0 0	$\boxed{0}$	$\boxed{1}$	$\boxed{*}$	$\boxed{*}$

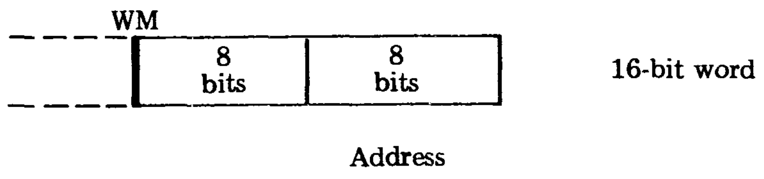
7. Logical AND two 8 bit numbers (result in accumulator)
Instruction: ANA Y

		EQ	GR	MI	OVF
ACC	1 1 0 0 1 1 0 0				
Y	1 1 1 1 0 0 0 0				
ACC RESULT	1 1 0 0 0 0 0 0	$\boxed{0}$	$\boxed{1}$	$\boxed{*}$	$\boxed{*}$

*remains unchanged

Figure 1-3. Variable Word Length Operations.





The word is addressed at its low order byte and is word-marked at its high order end.

Both the accumulator (ACC) and words in core memory are made variable by program control. This flexibility simplifies programming by permitting maximum efficiency in the use of core memory. Software provided by BIT is structured to take maximum advantage of mixed word length processing as desired.

Memory Addressing

Memory locations are addressed by unsigned positive binary numbers. For addressing purposes, the memory is divided into blocks of 256 bytes. Each 2-byte instruction can directly address 512 bytes, in either the page containing the next instruction or a page previously designated by the program as the data page. Indirect addressing to any location within either the program counter page or the page at which the page register is set is likewise possible. The 4-byte instructions can address any location in any page.

Types of Data Words

Data words consist of any multiple of 8-bit bytes, where the most significant byte has a word-mark bit. These bytes of information are used in the computer in the form of four different types of data words:

1. Binary
2. Purely numeric
3. Purely alphabetic
4. Alphanumeric

Timing

The time to fetch an instruction is measured in memory cycles. The time required for one memory cycle, i.e., transferring a byte between the BIT 483 central processor and the memory, is 980 ns. Instruction execution times vary with the instruction type, number of data bytes, use of indirect addressing, etc. For example, the average execution time for a 2-byte jump instruction is .98 microseconds (μs); for a binary add, 2.25 μs; and for a binary subtract, 2.25 μs.

Interrupt Capability

The interrupt subsystem of the BIT 483 consists of an expandable priority chain, with eight priority levels provided as standard equipment. The highest level is always reserved for the automatic power failure interrupt of the memory retention feature. Level 3 is hard-wired on 483 processors for servicing teletype interrupts. The control panel interrupt button is attached to Level 4. Additional interrupt levels can be provided as optional equipment (in groups of eight) up to a maximum of 32. The standard level assignments are summarized below:

Level 1	Power Failure
Level 2	Program Interrupt
Level 3	Teletype
Level 4	Front Panel Interrupt
Level 5	Open
Level 6	Open
Level 7	Open
Level 8	Open

The interrupt system further allows interrupts of interrupts to the maximum number of levels. Upon each entrance to an interrupt, the program counter and indicator flops are stored, by hardware action, for subsequent restoration upon completion of the interrupt subroutine. An interrupt at a given priority level will interrupt all lower priority interrupts.

The automatic power failure interrupt allows the 483 sufficient time to react when the line voltage drops below a prescribed level. When the processor detects a power failure condition, the program counter and all indicators are automatically stored by hardware action. The program is then notified by interrupt that the system is experiencing a power failure so that peripheral devices may be smoothly cycled down. When line voltage is restored, the 483 can resume operation at the point in the program where the interrupt occurred.

Data Transfer

The maximum output data transfer rate of the BIT 483 is 1.02 million bytes per second, or 8.16 million information bits per second. The maximum input data transfer rate is 0.75 million bytes per second, or 6 million information bits per second.

Data transfer commands can be either overlapped (simultaneous with computation) or non-overlapped (machine is dedicated to the data transfer function until completion of the specified transfer). This choice is program controlled by means of a bit in the peripheral transfer instruction. Data transfers are fully buffered by hardware in the 483 direct memory access (DMA) data channels.

Since the 483 is a variable word-length machine, the data transfer instruction can transmit either a single byte or an entire block of data bytes of any length. A transfer can also be commanded to start transmission from any location in core memory.

A number of methods can be employed for terminating a block transfer, depending upon the requirements of the I/O device. One technique, possible because of the flexible 483 word structure, is the ending of the transfer with a word mark. (Instructions for implementing this are provided in BIT I/O literature.)

For an output data transfer, the execution time is .98 μs + .98n μs. (One memory cycle is required to set up the data channel, and one cycle is required for each of n bytes transmitted by the instruction.) For an input data transfer, the execution time is .98 μs + 1.33n μs. (One memory cycle is required to set up the data channel and 1.33 μs are required for each data byte entered into core memory.) When data is entered into core memory from an I/O device, the word mark structure in memory is preserved.

For convenience in designing interfaces, BIT offers a series of dedicated logic-interface cards containing device selection and synchronous logic. These cards simplify implementing the timing functions required in designing an interface, and relieve the user of the task of generating such signals. (BIT I/O literature details this feature.)

SOFTWARE ELEMENTS

Software provided for the BIT 483 ranges from compiler level to assembler level, supported by extensive utility systems and mathematical aids.

Assemblers

Five assemblers are provided for use with the BIT 483 Computer, varying in size, sophistication, and ease of use. They permit the programmer to code instructions in a symbolic language which is more convenient than the binary numbers which actually operate the machine.

1. *ABIT-1:* a symbolic assembly language system designed for 483 units having a 1024- or 2048-byte memory capacity.

2. *ABIT-4:* a symbolic assembly language system designed for 483 units having a 4096-byte memory or larger.

3. *MABIT:* a modified ABIT-4 which includes 17 interpretive floating-point function single-statement calls, including HTAN, ARCTAN, LOG, e^X, and others.

4. *MACROBIT:* a macroassembler for use with 8K memory or larger, with a novel form for macro-calls and a syntax using the full ASCII character set to permit economy of expression. MACROBIT can be thought of as a powerful string processor, followed by a very simple assembler.

   The string processor features include IF and REPEAT statements, definition and redefinition of string macros (variables which expand to strings), and a form for macro-calls in which there are no special delimiters setting of a macro's arguments.

   Operating on the string generated by the string processor, the simple assembler has the features normally found in an assembler. These include complete symbol table control for the programmer, allowing definition, redefinition, purge, read-in, type out, and punch out, and extensive address arithmetic, good error diagnostics, and convenient option selection via the console switches.

   The macroassembler is able to use the same storage allocation algorithm described in IL (below), simplifying the task of fitting small segments of program into page-size pieces.

5. *IL:* an advanced programming language for use with 8K memories or larger, pioneered by BIT for minicomputer applications. It permits programming at a level of detail intermediate between assembly language and the conventional higher level languages. It is possible, for example, to write assignment and conditional statements as in higher level languages, with extensive control over the details of the code generated and full use of all of the features of the machine. The language is more compact to use than an assembler in the sense that a statement in it may encompass a number of operations. As an example, the statement

$$A \leftarrow B + C - D \,\&\, A$$

in IL effects the following: B is placed in the accumulator. C is added to it. D is subtracted from it. The result of this is then logically ANDed with A, and the result is then left in A.

The programmer, using IL, has all the advantages associated with assembly language and at the same time much of the conciseness and readability of the higher level languages.

A unique storage allocation feature is also provided. A program can be written in segments, to each of which an absolute origin can be assigned; if not, the allocator makes an efficient assignment of these segments into the available core space, thus compiling an absolute binary tape. This allocation process greatly simplifies the task of fitting programs efficiently into a core paged machine.

Utility Systems

BIT 483 software features a text editor (EDIT), a symbolic tape edit program (SYTE), an on-line debugging system (BOLD), automatic loaders (BINREAD and LOADER-C), and punch program (BINPUNCH).

1. *EDIT:* a text editor which permits source program modification, such as corrections, additions, deletions, etc., in the assembly language.

2. *SYTE:* for processors with minimum memory sizes not large enough to accommodate EDIT, SYTE permits source program modification in assembly language, corrections, additions, and deletions.

3. *BOLD:* an on-line debugging system for use with new programs, BOLD permits entry of breakpoints, subprogram entries and exits, manual data entry, memory data listing, and pseudo interrupts. It also permits direct operator control of all phases of processing. BOLD uses 1.5K of core storage.

4. *BINREAD and LOADER-C:* to locate and load object programs and data into preselected memory areas. As programs are operating, these loaders reside in memory ready for reuse whenever needed.

5. *BINPUNCH:* a program for independent use or as a segment of BOLD to copy selected memory data or programs onto punched paper tape. It makes possible copying a modified operating program directly onto paper tape for reuse at a later time, thereby eliminating the need for reassembling a program for correction or modification.

FORTRAN and Mathematical Aids

The BIT 483 software further features a FORTRAN compiler; a set of mathematical subroutines, MATH PACK, which perform floating-point arithmetic; and CALC, an automatic system performing 15 arithmetic keyboard calculations.

1. *FORTRAN:* the FORTRAN compiler, designed for use with a 483, having an 8K memory or larger, includes every function of USA STANDARDS BASIC FORTRAN. It is a one-pass compiler which produces an object program. It simplifies the problem of program preparation by enabling users with little or no knowledge of the computer's organization to write programs by expressing problems in a mixture of mathematical statements and English words.

2. *MATH PACK:* a set of subroutines which perform floating-point arithmetic. The set includes add, subtract, multiply, divide, sine, cosine, tangent, square root, exponential and log.

3. *CALC:* essentially converts the 483 into a rapid desk calculator, capable of performing 15 arithmatic functions automatically. The ASR-33 teletype keyboard is utilized for input and output.

HARDWARE ELEMENTS

The BIT 483 consists of the following four major hardware elements:

I/O Network

The BIT 483 I/O network consists of a number of data channels, each capable of operating in a READ-WRITE mode. The central processor can have a maximum of three such READ-WRITE channels, as shown in Figure 1-4. Each of these channels is capable of executing data transfers, while the processor is simultaneously computing. A maximum of 32 peripheral devices and a teletype can be addressed via the I/O network. The teletype used with the 483 (unless otherwise specified) is a standard ASR-33.

Central Processor

The central processor (CPU) basically consists of (1) a set of registers to store various pieces of transient information; (2) transfer buses for transferring information among the various registers or between the central processor registers and other parts of the system; (3) control logic to execute the basic cycle of fetching and decoding an instruction, address modification, etc.; and (4) control logic for executing the various individual instructions in the 483 repertoire. Figure 1-5 is a block diagram of the central processor.

The processor has a single-address instruction capability with access to all memory locations in the main core memory. It is controlled by the program stored in memory, and capable of processing data and performing arithmetic and logical operations. The logic is implemented with extensive medium scale integration (MSI) techniques and transistor-transistor logic (TTL) elements.

Memory

The coincident core memory is the primary storage facility for the BIT 483. It provides rapid, random

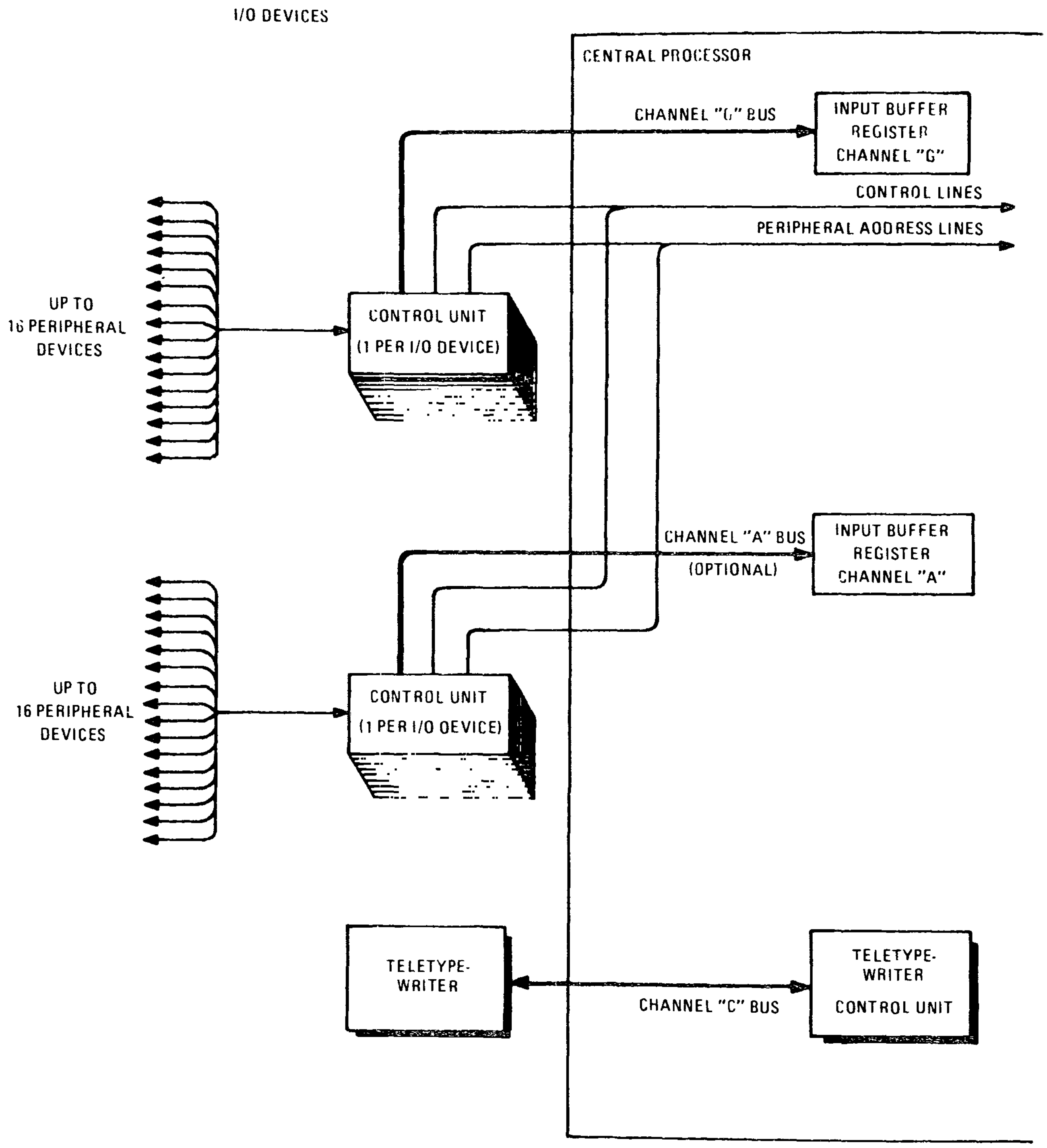

I/O DEVICES
CENTRAL PROCESSOR
CHANNEL "C" BUS
INPUT BUFFER REGISTER CHANNEL "C"
CONTROL LINES
PERIPHERAL ADDRESS LINES
UP TO 16 PERIPHERAL DEVICES
CONTROL UNIT (1 PER I/O DEVICE)
CHANNEL "A" BUS (OPTIONAL)
INPUT BUFFER REGISTER CHANNEL "A"
UP TO 16 PERIPHERAL DEVICES
CONTROL UNIT (1 PER I/O DEVICE)
TELETYPE-WRITER
CHANNEL "C" BUS
TELETYPE-WRITER CONTROL UNIT

Figure 1-4. Block Diagram — I/O Network.

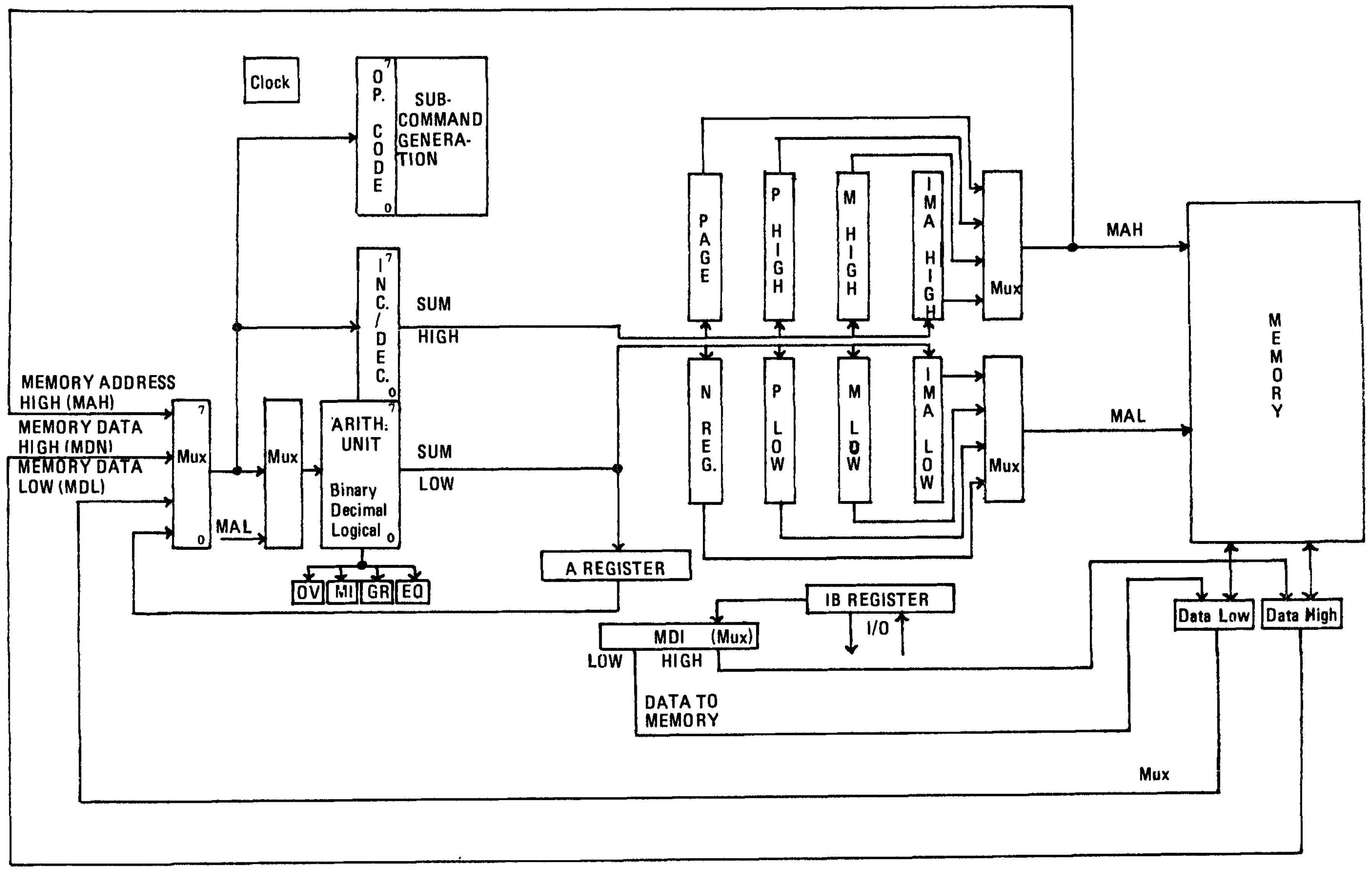


Figure 1-5. Block Diagram — Central Processor.

access data and instruction storage for both the central processor and the I/O network. The basic memory consists of 18 planes, of which 16 store two 8-bit bytes and two planes which store word-mark bits. A pair of additional planes, an optional feature, is available for use as an odd parity bit for each byte. The minimum memory size available is 1024 bytes, expandable times two, up to 65,536 bytes.

Control Panel

The control panel is divided into two basic sections. An upper section contains indicator lamps, while a lower section contains control switches. The indicators display the status of the BIT 483, i.e., stop, illegal instruction, etc., as well as any significant register. In addition to their normal functions, the switches have the capability of accessing and changing the contents of any memory location. Identification and function of panel switches and indicators are covered in the operating instructions presented in Section 2.

section two

memory

MEMORY ORGANIZATION

In the BIT 483 Computer, as noted in the Introduction, the primary storage facility is the coincident core memory. This consists of 18 planes of which 16 are used to store two 8-bit characters called bytes and the two remaining planes store a word-mark bit for each byte. The minimum memory size available is 1,024 bytes, expandable times two, up to 65,536 bytes. A simplified diagram of the memory layout is given in Figure 2-1.

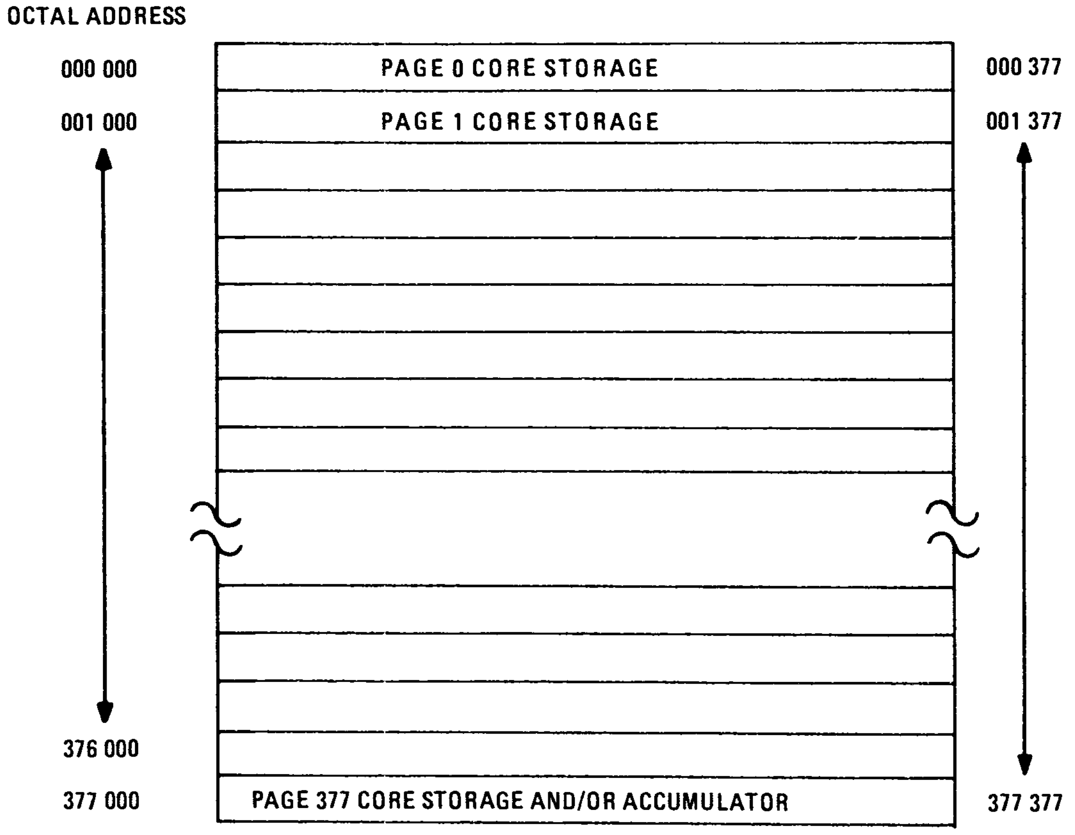


Figure 2-1. Memory Core Layout.

MEMORY ADDRESSING

Memory locations are addressed by unsigned positive binary numbers. For addressing purposes the memory is divided into blocks of 256 bytes, designated as pages. A 2-byte instruction can directly address any byte in either the page containing the next instruction or a page previously designated by the program as the data page. Indirect addressing to any location within either the program counter page or the data page is also possible. The 4-byte instructions can address any location in any page.

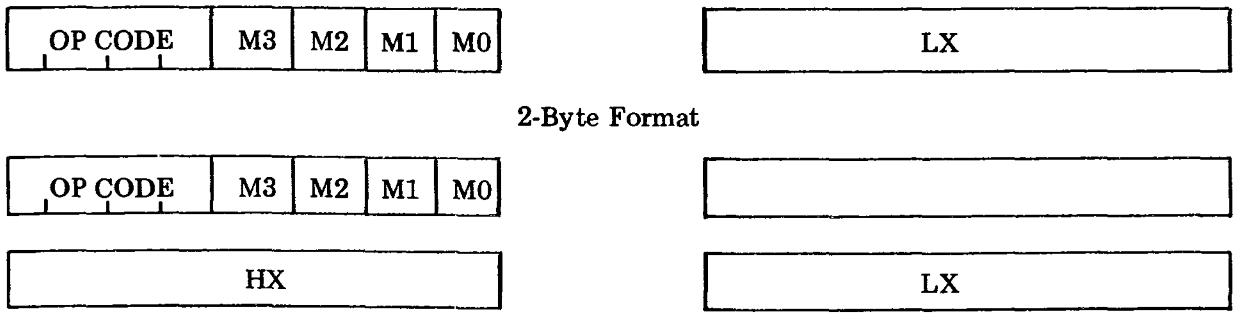
Two registers, the program counter and the page register, within the central processing unit are directly involved in all memory addressing. The size of these registers, tabulated in Table 2-1, is a function of the memory size.

Table 2-1. BIT 483 Register Size

Memory Size	Program Counter Size	Page Register Size
1K	10 bits	2 bits
2K	11 bits	3 bits
4K	12 bits	4 bits
8K	13 bits	5 bits
16K	14 bits	6 bits
32K	15 bits	7 bits
64K	16 bits	8 bits

In normal computer operations the program counter defines the address of the next instruction to be executed. The program counter is divided into two parts: the 8 low-order bits, called the low program counter (LPC); and the 8 high-order bits, called the high program counter (HPC). The LPC addresses the location within a page; the HPC addresses the page.

The page register serves as the high-order bits of the effective address in instances where the operand is not on the same page as the instruction. By program control, the page register is used with the address byte of the 2-byte instruction to address any desired byte in memory (the two types of instruction are illustrated in Figure 2-2).



4-Byte Format

Figure 2-2. Instruction Formats.

The accumulator in the 483 is the last page of memory. It is automatically addressed by the computer in its high order address byte. The size of the accumulator depends upon the size of the word it contains, which may be the total page. In all cases the extent of the accumulator used depends in turn upon the program being operated, and may vary during operation because of the mixed word lengths usually encountered within a program. (It is not necessary to fix the size of the accumulator since this is a result of normal data manipulation.)

WORD FORMATS

Computer words in memory are either instruction or data words. Instruction words define the desired operation, e.g., add, subtract. Data words provide the arguments for the instruction. Instruction words are fixed — either two bytes or four bytes, in length. Data words in the BIT 483, on the other hand, are variable, not fixed in length.

Instruction Word Formats*

Instruction words are either two or four bytes in length. The first byte always defines the operation to be performed. The second byte of the 2-byte instruction, the higher address byte, contains an 8-bit address (LX). It is the low-order byte of the effective address of the operand, in normal operation. When indirect addressing is used, it is the address of a location within the current page where an 8-bit address, used as the low-order byte of the effective address of the operand, is located. The format of each instruction type is illustrated in Figure 2-2.

Two-Byte Instructions

The first byte of the 2-byte instruction is divided into two 4-bit fields. The first field contains a 4-bit operation code, while the remaining field is made up of four modifier bits designated M3, M2, M1, M0. Unless otherwise specified, the function of these modifier bits is

M3 — Special modifier
M2 — Page register selection
M1 — Destination control
M0 — Indirect addressing

Four-Byte Instructions

The 4-byte instructions are jump-on-condition instructions. They contain a 16-bit address in the low-order bytes as Figure 2-2 demonstrates. In most cases they test the status of EQ, GR, MI, and OVF (see Section One). If the test results in a positive reply, the 16-bit address replaces the program counter. The individual jump instructions are explained in detail in Section Three.

The operation code field essentially specifies the particular function to be performed. The operation is further defined by the modifier bit configuration. All operation codes are defined in Table 2-2.

*I/O instructions not included

Table 2-2. Operation Codes

Operation Code	Function
0001	Peripheral transfer instruction
0010	Four-byte jump instructions
0011	Peripheral control instructions
0100	Logical instructions
0101	Binary instructions
0110	Page register instructions
0111	Data copy instructions
1000	Word-mark instructions
1001	Two-byte jump instructions
1010	Decimal instructions

Data Word Format

Data words are stored in memory as consecutive 8-bit bytes. The length of any word is defined by the program. The setting of a word-mark bit at the high-order byte of the word establishes the length. Some typical variable length words are illustrated in Figure 1-3.

Data words occupy memory in ascending addresses starting with the high-order byte of the data word in the low-order address. Since the data word can occupy more than one byte, it is usually addressed at its low-order byte which is the higher order memory address. Figure 2-3 is an example of addressing variable length words in memory.

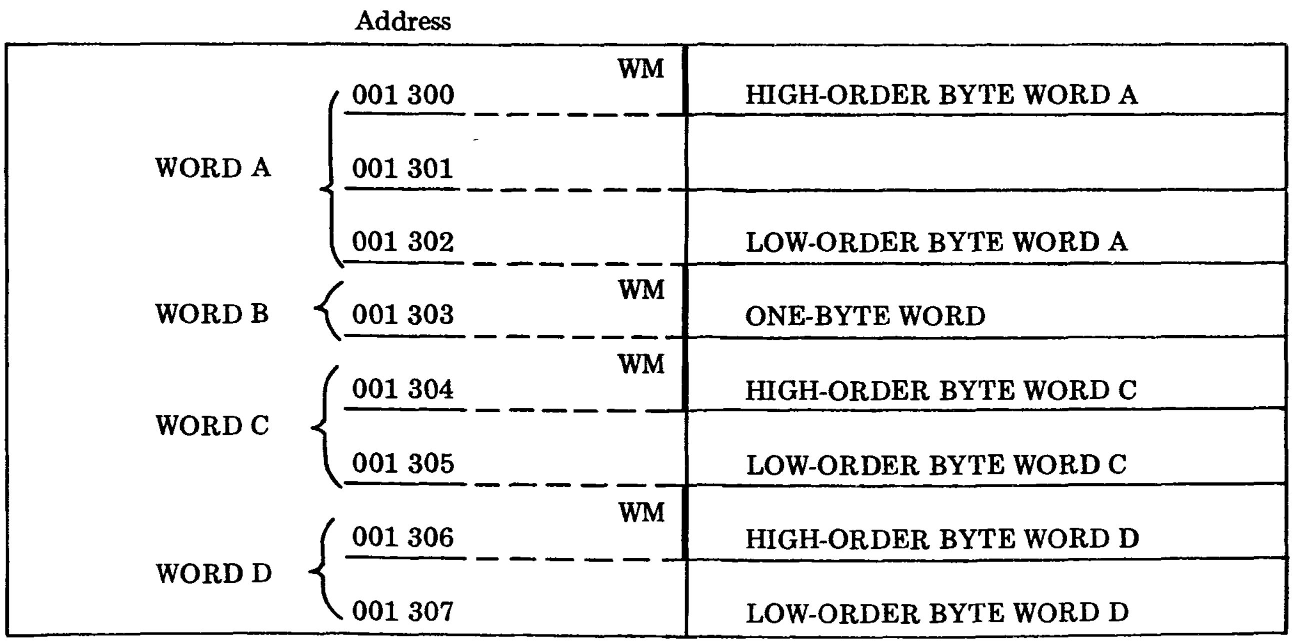


Figure 2-3. Data Words.

Word A is a typical example of a 3-byte or 24-bit word in memory. Its address is page 1 location 302.

Word B is a typical example of a 1-byte or 8-bit word. Its address is page 1 location 303.

Words C and D are both 2-byte or 16-bit words. Their addresses are page 1 location 305 and page 1 location 307, respectively. While it is possible to address all bytes in core, a word's length may be easily modified by modifying its address when used in the instruction word or by changing the word-mark location at the high-order byte.

DIRECT ADDRESSING

When addressing directly, the effective address of an operand consists of two entities, the page and the location within the page. The page value can be supplied by either the page register or the HPC. The location within the page is always represented in the address byte of the instruction. Since the page value can take on the value of one of two registers, two direct addressing modes actually exist. The mode of addressing is selected by the program. The selection is based on the condition of modifier bit 2. A block diagram illustrating the addressing modes including the program counter.and the page register, is presented in Figure 2-4.

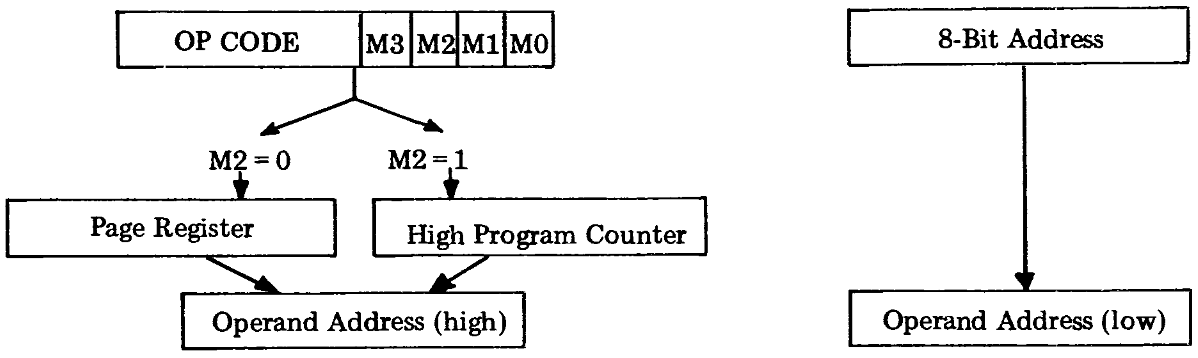


Figure 2-4. Effective Address.

Page Register Mode

The page register is used as the high-order bits of the effective address when M2 is equal to 0. The contents of this register may be changed any time it is necessary to access data from a page other than the program counter page or the page presently represented by the page register. This mode allows for the retrieval of data from any page in memory.

The number of bits in the page register used for direct addressing is a function of memory size. The page register is combined with the address byte of the instruction to form the effective address. The following example binary-adds (BAA) an 8-bit word from memory to the accumulator (the instruction is located in page 1 location 100, while the operand is in page 2 location 250, assuming a memory size of 8K):

Page	Location	Instruction	Address	Mnemonic (ABIT-4)
001	100	120	250	BAA NUMB
≈				≈
002	250	\001		NUMB OCT1 W001

To perform this operation, the page register must be set previously by the program to a value representing the page on which the data is located, specifically 2. The value of the program counter and the page register at the time of execution are:

X	X	X	0	0	0	0	1

High Program Counter

0	1	0	0	0	0	1	0

Low Program Counter

PROGRAM COUNTER

X	X	X	0	0	0	1	0

PAGE REGISTER

The program counter points to the next instruction to be executed at memory location page 1 location 102, and the page register is set to address a memory location in page 2. The assumption of an 8K memory system limits the number of bits used in addressing in both the HPC and the page register to the low-order 5 bits. During execution, the page register combines with the low-order byte of the instruction to form the effective address:

X	X	X	0	0	0	1	0

Operand Address High

1	0	1	0	1	0	0	0

Operand Address Low

High Program Counter (HPC) Mode

The HPC is used as the high-order bits of the effective address when M2 is equal to 1. This mode is used when the data is in the same page as the instruction. As in the case of the Page Register Mode, the 8-bit address contained in the address byte of the instruction is placed in a work register. The contents of the HPC complete the effective address. At the time of execution, this combination accesses the memory location containing the operand. The following example binary adds (BAAP) an 8-bit word from memory to the accumulator, with the operand located in the same page as the instruction:

Page	Location	Instruction	Address	Mnemonic (ABIT-4)
001	200	124	300	BAAP NUM1
≈				≈
001	300	\002		NUM1 OCT1 W002

A flow diagram of the effective address calculation is shown in Figure 2-5.

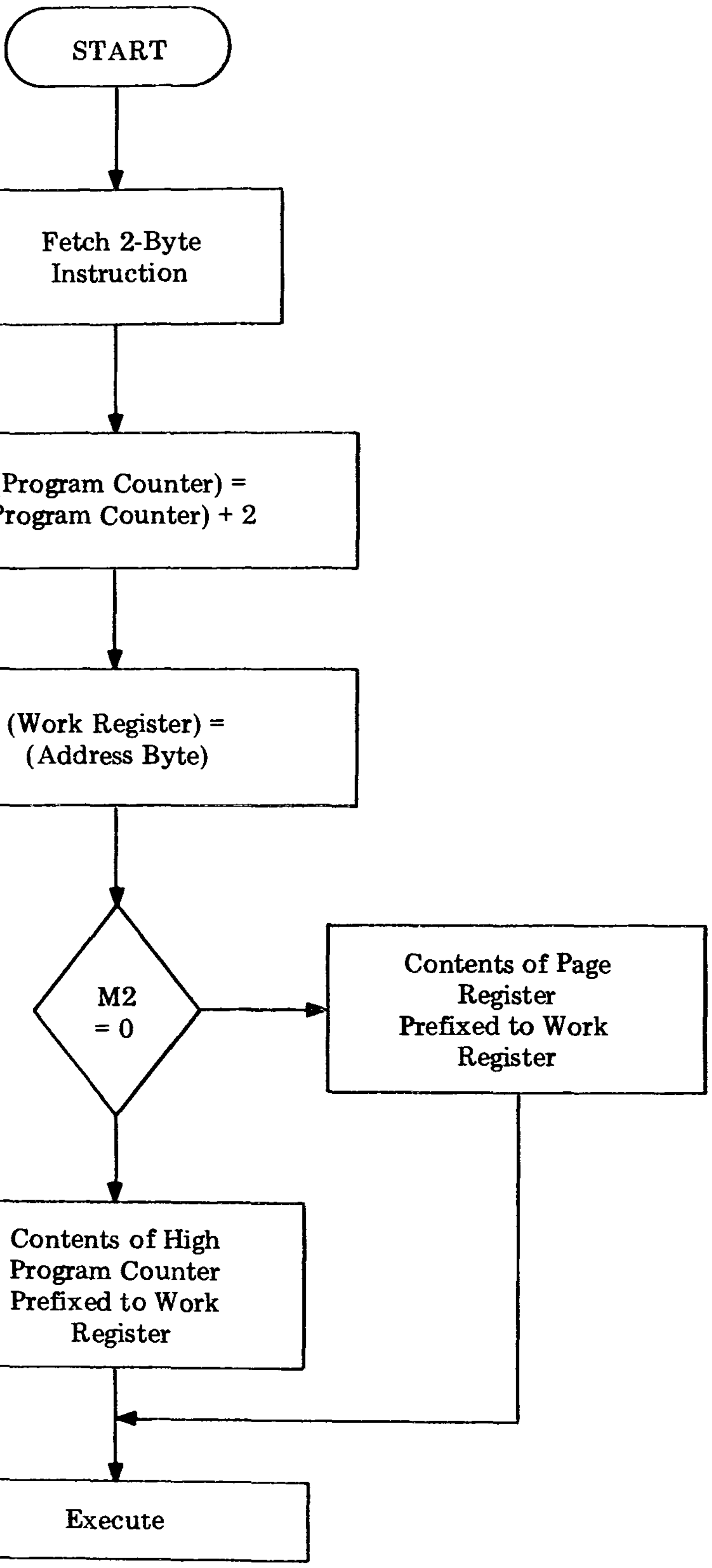

START
Fetch 2-Byte Instruction
(Program Counter) = (Program Counter) + 2
(Work Register) = (Address Byte)
M2 = 0
Contents of Page Register Prefixed to Work Register
Contents of High Program Counter Prefixed to Work Register
Execute

Figure 2-5. Memory Location Effective Address Calculation.

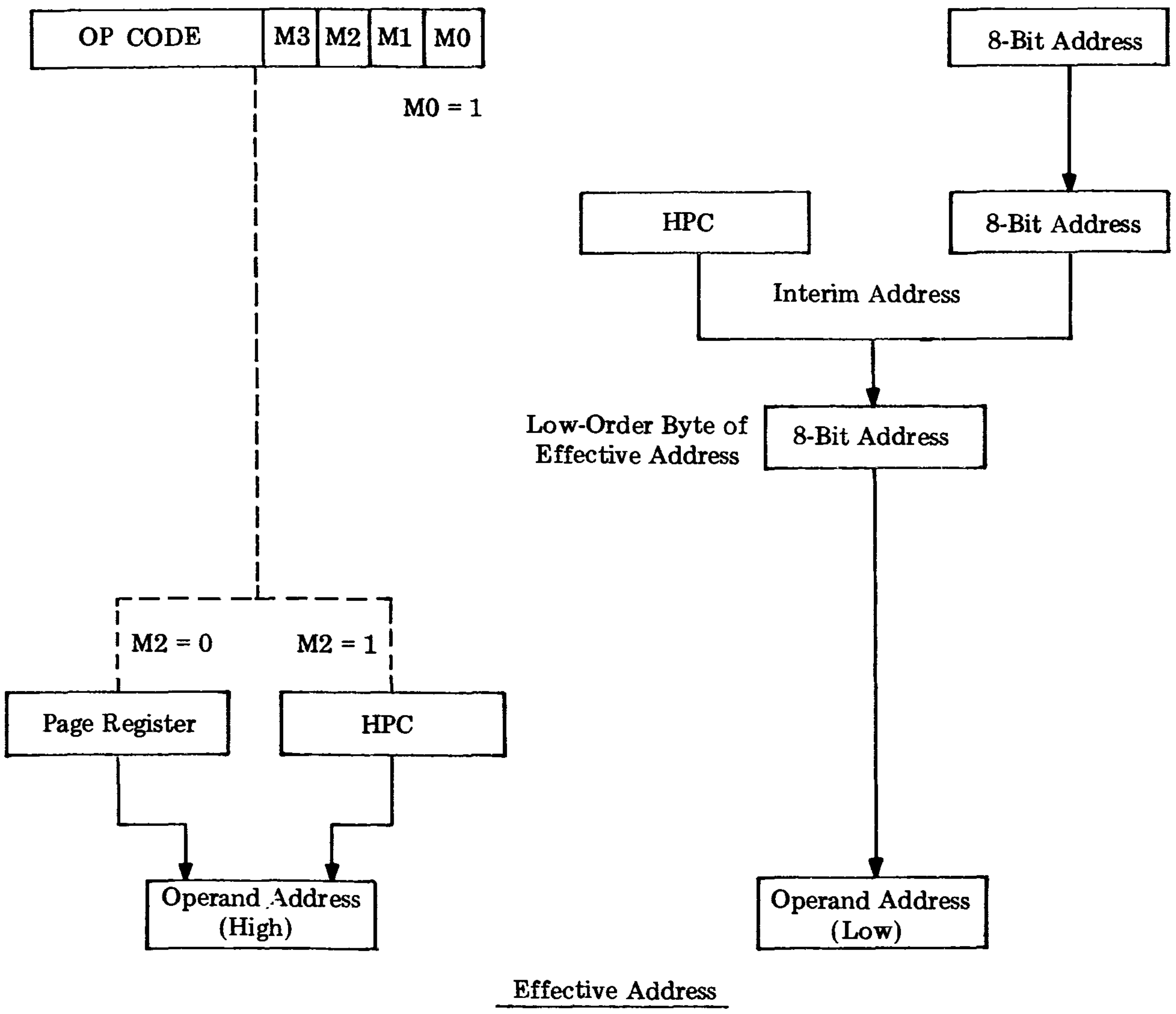


Figure 2-6. Indirect Effective Address Calculations.

INDIRECT ADDRESSING

Indirect addressing is controlled by modifier bit M0. If M0 is equal to 0, the 8-bit address in the address byte of the instruction is used directly with the page register or the high program counter to address a memory location (see preceding discussion).

Indirect addressing is specified when modifier bit M0 is equal to 1. The address in the address byte of the instruction is combined with the HPC to form an interim address. This address points to a location in the program counter page, which contains a final 8-bit address for use in the effective address calculation. After the retrieval of this final 8-bit address modifier, bit M2 is examined for its status. If M2 is equal to 1, the page register is used as the high-order bits of the effective address. If M2 is equal to 0, the HPC is used as the high-order bits of the effective address. Figure 2-6 illustrates the effective address calculation when indirect addressing is specified.

OPERATING INFORMATION

Console

Controls and indicators for operating the BIT 483 Computer are illustrated in Figure 2-7, coordinated with and referenced to Table 2-3.

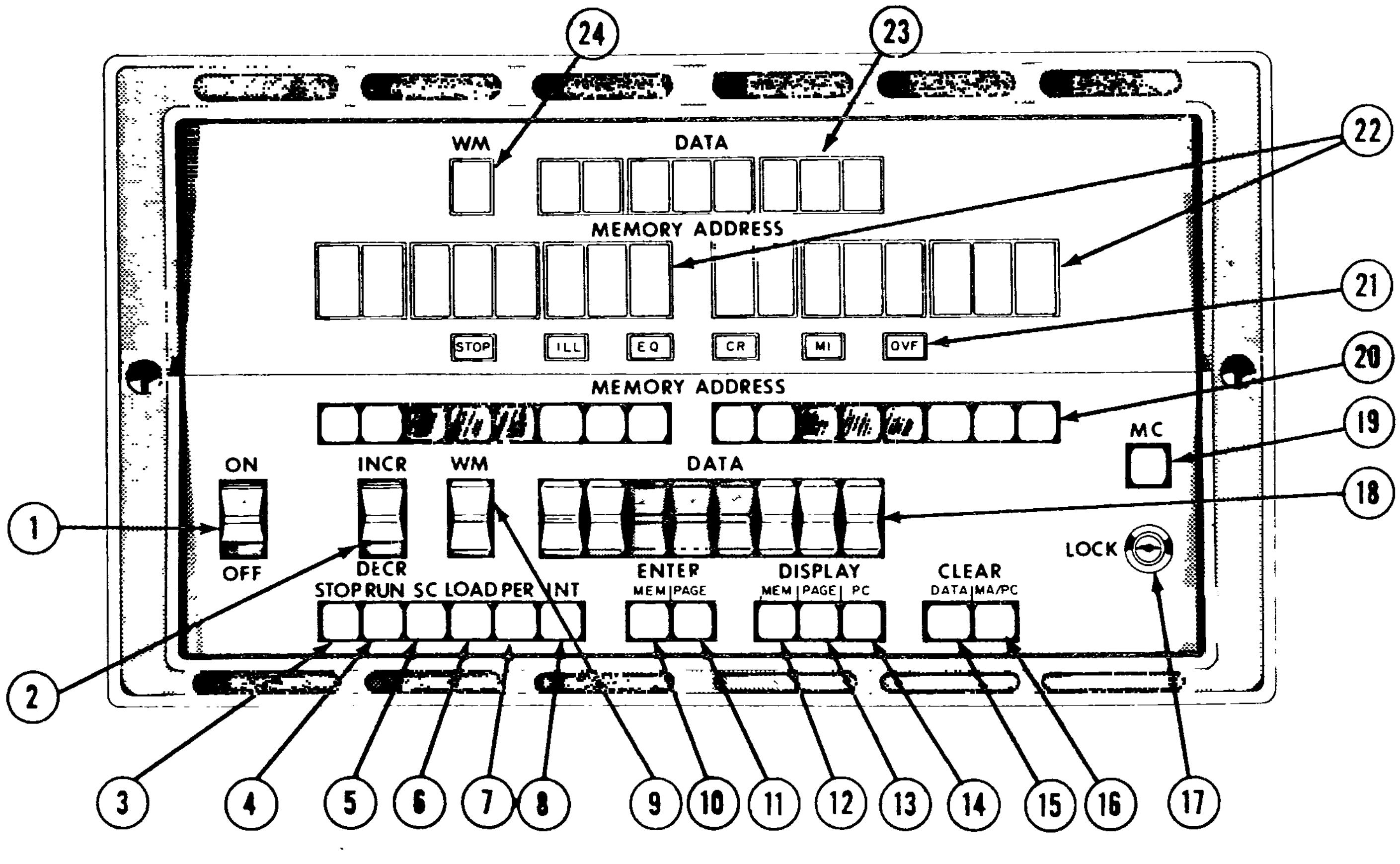


Figure 2-7. 483 Console

Table 2-3. Identification of BIT 483 Controls and Indicators

No.*	Name	Function
		CONTROLS
1	ON/OFF	Turns 115 VAC primary voltage ON and OFF.
2	INCR/DECR	INCREMENT/DECREMENT switch. Causes the main memory address register to be incremented or decremented, depending on INCR or DECR position of switch, when either ENTER MEM or DISPLAY MEM switch is pressed.
3	STOP (Momentary Contact Switch)	STOP switch. When pressed, program execution stops; STOP indicator lamp lights.
4	RUN (Momentary Contact Switch)	RUN switch. When pressed, starts program execution at the address specified in the program counter.
5	SC (Momentary Contact Switch)	SINGLE CYCLE switch. When pressed, executes a single instruction.
6	LOAD (Momentary Contact Switch)	LOAD switch. When pressed, starts to read paper tape into memory from teletype reader; stopped automatically by blank character on tape.
7	PER (Momentary Contact Switch)	PERIPHERAL switch. When pressed, resets all peripheral lines to zero without affecting the 483 processor registers.
8	INT (Momentary Contact Switch)	INTERRUPT switch. When pressed, interrupts processor operation.
9	WM	WORD-MARK switch. A data switch which, when pressed, enters a word mark bit (9th bit of a byte) into the DATA indicators.
10	ENTER MEM (Momentary Contact Switch)	ENTER MEMORY switch. When pressed, data displayed on DATA indicators are actually entered into the memory at the address shown on the MEMORY ADDRESS indicators.
11	ENTER PAGE (Momentary Contact Switch)	ENTER PAGE switch. When pressed, data displayed on DATA indicators are actually entered into the page register.

*Numbers correspond to those in Figure 2-7.

Table 2-3 (continued)

No.	Name	Function
12	DISPLAY MEM (Momentary Contact Switch)	DISPLAY MEMORY switch. When pressed, displays on the DATA indicators the information stored at the address displayed on the MEMORY ADDRESS indicators. The contents of sequential memory locations are displayed each time the switch is pressed.
13	DISPLAY PAGE (Momentary Contact Switch)	DISPLAY PAGE switch. When pressed, displays contents of page register on DATA indicators.
14	DISPLAY PC (Momentary Contact Switch)	DISPLAY PROGRAM COUNTER switch. When pressed, displays location of next instruction in program counter on MEMORY ADDRESS indicators.
15	CLEAR DATA (Momentary Contact Switch)	CLEAR DATA switch. When pressed, clears data appearing on DATA display indicators and in the data register.
16	CLEAR MA/PC (Momentary Contact Switch)	CLEAR MEMORY ADDRESS/PROGRAM COUNTER switch. When pressed, clears the contents of the memory address register, the program counter, and the MEMORY ADDRESS indicators.
17	LOCK	A manual lock and key is provided to lock the control panel switches in order to prevent unintentional setting of switches. When the key is horizontal, the panel is locked; when vertical, open.
18	DATA Switches (upper positions only)	A series of DATA switches provides the capability of manually entering data into the memory. The switches are colored in groups to facilitate reading in the octal system.
18	SENSE Switches (downward position on 8 DATA switches and WM switch)	Nine SENSE switches are available for use on the control panel. Any permutation of these switches can be tested by the program to determine if specialized processing is necessary. It is possible to set and reset any code while the CPU is actively engaged processing other requirements.
19	MC (Momentary Contact Switch)	MASTER CLEAR switch. When pressed, resets all major registers and peripheral lines to zero (has no effect on memory).

Table 2-3 (continued)

No.	Name	Function
20	MEMORY ADDRESS Switches	A series of 16 switches is provided for inspecting the contents of any memory location. This is accomplished by setting the memory address register and the program counter to the specified value.
		INDICATORS
21	STOP	Lights when the STOP switch is pressed, when the program halts, or when the computer tries to execute an illegal instruction.
21	ILL	ILLEGAL OPERATION indicator. Lights when the processor halts in attempting to execute an illegal operation code.
21	EQ	EQUAL-TO-ZERO indicator. Lights when the result of a binary logical or decimal operation is zero, or it and the greater-than-zero (GR) indicator are both set to indicate word size has been exceeded, i.e., when there is a carry out of the highest order byte of the results.
21	GR	GREATER-THAN-ZERO indicator. Lights when the result of a decimal operation is greater than zero, or the result of a binary or logical operation is non-zero, or it and the EQ indicator are both set to indicate word size has been exceeded (as per the preceding explanation).
21	MI	MINUS indicator. Lights when the highest order bit of the result of a binary operation equals one.
21	OVF	OVERFLOW indicator. Lights after a binary operation when either a "carry out" of the highest order bit of the result occurs without an accompanying "carry into," or when a "carry into" the highest order bit occurs without an accompanying "carry out."
22	MEMORY ADDRESS	Light when MEMORY ADDRESS switches are pressed. During operation indicate the current memory address.
23	DATA	Light when DATA switches are pressed. During operation indicate data entering memory.
24	WM	WORD-MARK indicator. Lights when a 9th bit is set indicating the presence of a word mark.

Instructions

Operating instructions for the BIT 483 Computer are provided in Table 2-4.

Table 2-4. Basic BIT 483 Instructions

Instruction	Execute Timing (cycles)	Mnemonic	Binary Code	Octal Code	Functional Description
NOTES: *Fetch Timing = 4 memory cycles (Fetch Timing for all other instructions = 2 memory cycles) Memory Cycle Time = .980 μs Indirect Addressing = 1 cycle n = the number of bytes in either the memory or accumulator operand, whichever is smaller nm = the number of bytes in the memory word na = the number of bytes in the accumulator word S = time between data bytes from peripheral device					
		JUMP INSTRUCTIONS			
Jump return from interrupt	1.0	*JRI	00100000	040	Jump return from interrupt
Unconditional jump	1.0	*JMP	00100001	041	Unconditional jump
Jump if EQ	1.0	*JEQ	00100010	042	Jump if EQ is set
Jump if $\overline{\text{EQ}}$	1.0	*NEQ	00100011	043	Jump if EQ is not set
Jump if GR	1.0	*JGR	00100100	044	Jump if GR is set
Jump if $\overline{\text{GR}}$	1.0	*NGR	00100101	045	Jump if GR is not set
Jump if GR & EQ	1.0	*JGE	00100110	046	Jump if GR and EQ are both set
Jump if $\overline{\text{GR \& EQ}}$	1.0	*NGE	00100111	047	Jump if GR or EQ is not set
Jump on peripheral test	1.0	*JPT	00101000	050	Jump if positive response from peripheral equipment
Jump to subroutine	2.6	*JSR	00101001	051	Jump to subroutine

Table 2-4 (continued)

Instruction	Execute Timing (cycles)	Mnemonic	Binary Code	Octal Code	Functional Description
Jump if MI	1.0	*JMI	00101010	052	Jump if MI is set
Jump if $\overline{\text{MI}}$	1.0	*NMI	00101011	053	Jump if MI is not set
Jump if OVF	1.0	*JOV	00101101	054	Jump if OVF is set
Jump if $\overline{\text{OVF}}$	1.0	*NOV	00101101	055	Jump if OVF is not set
Jump if EQ or GR	1.0	*EOG	00101110	056	Jump if EQ or GR is set
Jump if $\overline{\text{EQ or GR}}$	1.0	*ENG	00101111	057	Jump if GR and EQ are not set
		PERIPHERAL CONTROL INSTRUCTION			
Peripheral select		PCI	0011DDDD		Selects device DDDD
		LOGICAL INSTRUCTIONS			
Exclusive OR to accumulator	2.3 n	ORA	01000P0I	100	Exclusive OR the contents of memory to the contents of the accumulator
Exclusive OR to memory	2.3 n	ORM	01000P1I	102	Exclusive OR the contents of the accumulator to the contents of memory
Logical AND to accumulator	2.3 n	ANA	01001P0I	110	AND the contents of memory to the contents of the accumulator
Logical AND to memory	2.3 n	ANM	01001P1I	112	AND the contents of the accumulator to the contents of memory

Table 2-4 (continued)

Instruction	Execute Timing (cycles)	Mnemonic	Binary Code	Octal Code	Functional Description
		BINARY INSTRUCTIONS			
Binary ADD to accumulator	2.3 n	BAA	01010P0I	120	Binary ADD the contents of memory to the contents of the accumulator
Binary ADD to memory	2.3 n	BAM	01010P1I	122	Binary ADD the contents of the accumulator to the contents of memory
Binary SUBTRACT from accumulator	2.3 n	BSA	01011P0I	130	Binary SUBTRACT the contents of memory from the contents of the accumulator
Binary SUBTRACT from memory	2.3 n	BSM	01011P1I	132	Binary SUBTRACT the contents of the accumulator from the contents of memory
		PAGE REGISTER INSTRUCTIONS			
Change PAGE register	1.0	CHP	01100P0I	140	Replace the contents of the PAGE register with the contents of memory
Store PAGE register	1.3	SPR	01100P1I	142	Save the contents of the PAGE register in memory.
Store SENSE switches	1.3	SSS	01101P1I	152	Save the SENSE switch settings in memory
		COPY INSTRUCTIONS			
Move one word to accumulator	2.0 nm	C1A	01110P0I	160	Copy one word from memory to the accumulator
Move one word to memory	2.0 na	C1M	01110P1I	162	Copy one word from the accumulator to memory

Table 2-4 (continued)

Instruction	Execute Timing (cycles)	Mnemonic	Binary Code	Octal Code	Functional Description
Move two words to accumulator	2.0 + 2.0 nm	C2A	01111P0I	170	Copy two words from memory to the accumulator
Move two words to memory	2.0 + 2.0 nm	C2M	01111P1I	172	Copy two words from the accumulator to memory
		WORD-MARK INSTRUCTIONS			
Erase WORD-MARK	1.3	EWM	10000P0I	200	Erase the word-mark bit in the specified memory location
Set WORD-MARK	1.3	SWM	10000P1I	202	Set the word-mark bit in the specified memory location
Jump on WORD-MARK	1.0	*JWM	10001P0I	210	Jump if the word-mark bit is set in the specified memory location
		TWO-BYTE JUMP INSTRUCTIONS			
Two-byte jump	1.0	PGJ	10010P0I	220	Change program counter
Halt & jump	1.0	HLT	10011P0I	230	Change program counter and halt
Set interrupt & jump	1.0	SIF	10010P1I	222	Change program counter and set interrupt flop
		TELETYPE NON-INTERRUPT MODE			
Transfer from TTY paper tape reader non-interrupt mode	100 milliseconds	RPT	00011010	032	Initiates a transfer from the paper tape reader on the TTY

Table 2-4 (continued)

Instruction	Execute Timing (cycles)	Mnemonic	Binary Code	Octal Code	Functional Description
Transfer from TTY keyboard non-interrupt mode	100 milliseconds	KEY	00011000	030	Initiates a transfer from the keyboard on the TTY
Transfer to TTY punch/keyboard non-interrupt mode	100 milliseconds	PRT	00011001	031	Initiates a transfer to the punch/keyboard on the TTY
PERIPHERAL TRANSFER INSTRUCTIONS					
Non-overlapped input peripheral transfer Channel G	1+1.3N+S	PTG	00011100	034	Initiates a non-overlapped transfer to a device previously selected for input on I/O Channel G
Non-overlapped output peripheral transfer Channel G	1+1.0N+S	PTG	00011100	034	Initiates a non-overlapped transfer to a device previously selected for output on I/O Channel G
Non-overlapped input peripheral transfer Channel A	1+1.3N+S	PTA	00010000	020	Initiates a non-overlapped transfer to a device previously selected for input on I/O Channel A
Non-overlapped output peripheral transfer Channel A	1+1.0N+S	PTA	00010000	020	Initiates a non-overlapped transfer to a device previously selected for output on I/O Channel A
Overlapped input peripheral transfer Channel G	1+1.3N	OTG	00011101	035	Initiates an overlapped transfer to a device previously selected for input on I/O Channel G
Overlapped output peripheral transfer Channel G	1+1.0N	OTG	00011101	035	Initiates an overlapped transfer to a device previously selected for output on I/O Channel G

Table 2-4 (continued)

Instruction	Execute Timing (cycles)	Mnemonic	Binary Code	Octal Code	Functional Description
Overlapped input peripheral transfer Channel A	1+1.3N	OTA	00010000	021	Initiates an overlapped transfer to a device previously selected for input on I/O Channel A
Overlapped output peripheral transfer Channel A	1+1.0N	OTA	00010001	021	Initiates an overlapped transfer to a device previously selected for output on I/O Channel A

section three

instructions

This section describes all BIT 483 Computer instructions, except the I/O instructions covered in Section Four. The instructions are discussed by class, with each class consisting of several instructions.

The instructions of the BIT 483 and the modifiers to these instructions are often inseparable. The modifiers are set by appending the suffixes P and/or I to the mnemonics recognized by the assembler ABIT-4. The suffix P causes modifier bit M2 to be set to one. If this suffix is omitted, modifier bit M2 is set to zero. The suffix I causes modifier bit M0 to be set to one. If the I suffix is omitted, modifier bit M0 is set to zero.

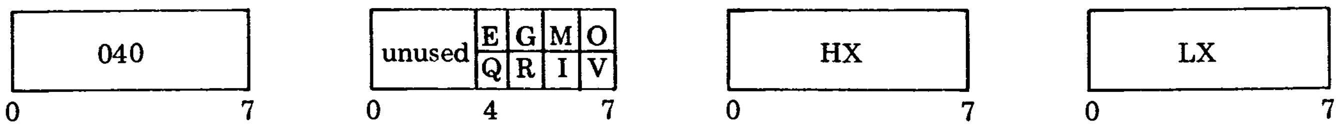
Accordingly, the description of each instruction includes all the possible combinations of that instruction as a result of appending the suffixes which control the modifier bits M2 and M0. Each mnemonic is assembled into a corresponding octal code, and placed in the high-order byte of the instruction. The mnemonic is followed by an English language definition of the instruction and its timing. The timing includes both the fetch and execution times. A block diagram of the instruction format and a description of its function has also been included.

FOUR-BYTE JUMP INSTRUCTIONS

There are sixteen 4-byte jump instructions, and all allow a program branch to any location in memory. Two program statements for each of these instructions are written. The first statement defines the jump command and conditions; the second statement is usually the jump address (JAD) "pseudo-op" (see the ABIT-4 Assembler Manual) or may be any constant, two bytes in length, that provides a complete 16-bit address.

If, when the BIT 483 executes the jump instruction, the conditions are met, the jump address is placed into the program counter. If the conditions are not met, the program counter is incremented by two, thus bypassing the jump address and proceeding to the next sequential instruction. The jump instruction (other than a JRI) does not affect the indicators. Unless otherwise specified, the second byte of the instructions is not used by the instruction. This byte may be used as a data location by the programs when the circumstances permit.

JRI Jump Return From Interrupt 4.9 μs



This instruction performs the same operations as the JMP instruction, plus two additional functions. First, when executed in the appropriate location the JRI resets the interrupt state flop of the present level, which returns the processor to the previous mode of operation from this level of interrupt. Second, the

EQ, GR, MI, and OVF indicators are set according to the values contained in the second byte of the instruction upon execution. If the program does not alter this byte before execution, the indicators will be restored to their original states at the time the interrupt occurred.

JMP Unconditional Jump 4.9 μs

041	unused	HX	LX
0 7	0 7	0 7	0 7

This instruction causes the 16-bit address specified in its low-order two bytes to be placed in the program counter. Control is then transferred to the instruction sequence at that location.

JEQ Jump If EQ 4.9 μs

042	unused	HX	LX

This instruction replaces the program counter with the jump address, if EQ is set. Otherwise, the instruction ends.

NEQ Jump If Not EQ 4.9 μs

043	unused	HX	LX

This instruction replaces the program counter with the jump address, if EQ is not set. Otherwise, the instruction ends.

JGR Jump If GR 4.9 μs

044	unused	HX	LX

This instruction replaces the program counter with the jump address, if GR is set. Otherwise, the instruction ends.

NGR Jump If Not GR 4.9 μs

045	unused	HX	LX

This instruction replaces the program counter with the jump address, if GR is not set. Otherwise, the instruction ends.

JGE Jump If GR and EQ 4.9 μs

046	unused	HX	LX

This instruction replaces the program counter with the jump address, if both GR and EQ are set. Otherwise, the instruction ends.

NGE Jump If Not GR and EQ 4.9 μs

047	unused	HX	LX

This instruction replaces the program counter with the jump address, if GR and EQ are not both set. Otherwise, the instruction ends.

JPT Jump On Peripheral Test 4.9 μs

050	test	HX	LX

This instruction replaces the program counter with the jump address if the selected peripheral device meets the condition specified in the second byte (see I/O Section). Otherwise, the instruction ends.

JSR Jump To Subroutine 7.5 μs

051	unused	HX	LX

This instruction saves the program counter in memory at the location specified by the jump address, and replaces the program counter with the jump address plus 2.

JMI Jump If Minus 4.9 μs

052	unused	HX	LX

This instruction replaces the program counter with the jump address, if MI is set. Otherwise, the instruction ends.

NMI Jump If Not Minus 4.9 μs

053	unused	HX	LX

This instruction replaces the program counter with the jump address, if MI is not set. Otherwise, the instruction ends.

JOV Jump If Overflow 4.9 μs

054	unused	HX	LX

This instruction replaces the program counter with the jump address, if OVF is set. Otherwise, the instruction ends.

NOV Jump If Not Overflow 4.9 μs

055	unused	HX	LX

This instruction replaces the program counter with the jump address, if OVF is not set. Otherwise, the instruction ends.

EOG Jump If EQ or GR 4.9 μs

056	unused	HX	LX

This instruction replaces the program counter with the jump address, if either GR or EQ is set. Otherwise, the instruction ends.

ENG Jump If Not EQ and Not GR 4.9 μs

057	unused	HX	LX

This instruction replaces the program counter with the jump address, if neither GR nor EQ is set. Otherwise, the instruction ends.

LOGIC COMMANDS

Two basic logical operations may be performed: logical exclusive OR and logical AND. The result of the operation may either be stored in the accumulator or in the designated memory location. The logical operations terminate when a word mark is first sensed in either operand. The following example illustrates each operation:

Logical Exclusive OR two 16-bit numbers (results in accumulator)

Instruction	ORA Y					
	WM					
ACC	01101101	11101101	EQ	GR	MI	OVF
	WM					
Y	00001110	01010111				
	WM					
ACC Results	01100011	10111010	0	1	X	X

Logical AND a 16-bit number and an 8-bit number (results in accumulator)

```
Instruction      ANA Z
                WM
ACC              01111011  10101011          EQ    GR    MI    OVF
                               WM
Z                          10001111
                 ___________________
                WM
ACC              01111011  10001011          0     1     X     X
```

Before the execution of a logical instruction, EQ and GR are cleared. If the result of the operation is equal to binary zero, EQ is set. If the result is not equal to binary zero, GR is set. This setting reflects only the bytes operated upon, even if the result field is the longer of the two operands. This instruction class does not alter the state of either MI or OVF.

Format

0	1	0	0	M3	M2	M1	M0

LX

M3: 1 logical AND
    0 exclusive OR

M2: 1 high program counter (HPC)
    0 page register (PR)

M1: 1 memory
    0 accumulator

M0: 1 indirect address
    0 direct address

Logical AND

Beginning with the memory location specified and the highest address in memory (accumulator), a byte at a time is logically ANDed between the contents of the effective address and the contents of the accumulator. The results are stored in either the accumulator or the location specified by the effective address. The destination is controlled by modifier bit M1.

The logical AND operates as follows:

X	0	0	1	1
Y	0	1	0	1
Z	0	0	0	1

AND To Accumulator

AND the contents of the effective address to the contents of the accumulator. The contents of memory are not changed.

ANA (110)	Timing: 4.21 μs + 2.25 μs/byte EFAD: (PR), LX
ANAP (114)	Timing: 4.21 μs + 2.25 μs/byte EFAD: (HPC), LX
ANAI (111)	Timing: 5.19 μs + 2.25 μs/byte EFAD: (PR), [(HPC), LX]
ANAPI (115)	Timing: 5.19 μs + 2.25 μs/byte EFAD: (HPC), [(HPC), LX]

AND To Memory

AND the contents of the accumulator to the contents of the effective address. The contents of the accumulator are not changed.

ANM (112)	Timing: 4.21 μs + 2.25 μs/byte EFAD: (PR), LX
ANMP (116)	Timing: 4.21 μs + 2.25 μs/byte EFAD: (HPC), LX
ANMI (113)	Timing: 5.19 μs + 2.25 μs/byte EFAD: (PR), [(HPC), LX]
ANMPI (117)	Timing: 5.19 μs + 2.25 μs/byte EFAD: (HPC), [(HPC), LX]

Exclusive OR

Beginning with the memory location specified and the highest address in memory (accumulator), a byte at a time is exclusive ORed between the contents of the effective address and the contents of the accumulator, with the results stored in either the accumulator or the location specified by the effective address.

The exclusive OR operates as follows:

X	0	0	1	1
Y	0	1	0	1
Z	0	1	1	0

Exclusive OR To Accumulator

Exclusive OR the contents of the effective address to the word in the accumulator. The contents of the effective address are not changed.

$\boxed{\text{ORA (100)}}$	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(PR), LX
$\boxed{\text{ORAP (104)}}$	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(HPC), LX
$\boxed{\text{ORAI (101)}}$	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(PR), [(HPC), LX]
$\boxed{\text{ORAPI (105)}}$	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(HPC), [(HPC), LX]

Exclusive OR To Memory

Exclusive OR the contents of the accumulator to the contents of the effective address. The contents of the accumulator are not changed.

$\boxed{\text{ORM (102)}}$	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(PR), LX
$\boxed{\text{ORMP (106)}}$	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(HPC), LX
$\boxed{\text{ORMI (103)}}$	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(PR), [(HPC), LX]
$\boxed{\text{ORMPI (107)}}$	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(HPC), [(HPC), LX]

BINARY OPERATIONS

The instructions of the BIT 483 Computer provide for variable-length arithmetic, allowing both signed and unsigned operations. Both operations use 2's complement notation. The operands need not be of the same length, since the operation ends when a word mark is found in either of the operands. During the process of the operation, four indicators, EQ, GR, MI, and OVF, can be set. They are cleared at the beginning of all binary arithmetic operations.

Signed Binary Arithmetic

The first bit position of a fixed point binary number holds the sign of the binary number; the remaining bits designate the magnitude of the number. Positive numbers are represented in a true binary fashion with the sign bit equal to zero. Negative numbers are represented in 2's complement notation with the sign bit equal to 1.

The 2's complement of a binary number (N) with n binary digit positions is:

$$\text{2's complement of } N = 2^n - N$$

Thus, for an 8-bit binary number (N) the 2's complement is:

$$2^8 - N = 100000000 - N$$

The 2's complement of the binary number 00111001, for example, is 2^8 − 00111001, or

```
  100000000
-  00111001
   11000111
```

Note that the subtraction is not required, since the 2's complement can be obtained by inspection of the number. Each bit of the numbers is simply inverted (that is, a 1 is changed to 0, and a 0 is changed to a 1) and a 1 is then added to the low-order (least-significant) bit at right. Thus, the

```
binary number       00111001
is inverted         11000110
1 is added         +       1
to obtain           11000111 = 2's complement of 00111001,
```

which checks with the result obtained above. This conversion technique can be used on any number regardless of size. The following examples illustrate the above method with a 16-bit number and a 24-bit number.

```
Binary number      01101101 01110111
is inverted        10010010 10001000
1 is added        +                1
                   10010010 10001001 = 2's complement of 01101101  01110111.

Binary number      01010111 01111100 00000011
is inverted        10101000 10000011 11111100
1 is added        +                         1
                   10101000 10000011 11111101 = 2's complement of
                                                01010111  01111100  00000011.
```

The BIT 483 regards a number with a zero sign bit as a positive number. Whereas the word length of the 483 is variable, the maximum number is a function of the word's length.

For example:

```
               S
a 16-bit word  01111111  11111111 = 77777₈
               S
an 8-bit word  01111111 = 177₈
               S
a 24-bit word  11111111  11111111  11111111 = 37777777₈
```

Therefore, the largest number (N) is

$$N = 2^{(8n-1)} - 1$$

where n is equal to the number of bytes in a word.

A number will be regarded as negative if the sign bit is a 1. The smallest negative number consists of an all-zero integer field with a sign bit of 1. For example:

```
              S
8-bit number  10000000 = 200₈
              S
16-bit number 10000000  00000000 = 100000₈
```

Therefore, the smallest negative number (M) is

$$M = -\ 2^{(8n-1)}$$

where n is the number of bytes in the number. Whenever the results of a binary operation are negative (sign bit = 1) the MI is set.

Summing up, the positive numbers range from 0 through $2^{(8n-1)} - 1$ and the negative numbers range from $-2^{(8n-1)}$ through -1, where n is the number of bytes in the number.

Overflow

When the result of an add or subtract exceeds the capacity of the field containing the result, an overflow condition results. Since an overflow carries into the left most- or sign-bit position, it changes the sign. Thus, in a positive overflow, the final sum or difference comes out negative, while a negative overflow results in a positive sum.

The presence or absence of an overflow condition may be recognized by the condition of the carries. The result of an operation does not overflow if there is either no carry into the high-order bit position and no carry out, or a carry of 1 into the high-order bit position and also a carry out. In contrast, an operation overflows if there is either a carry into the high-order position and no carry out, or a carry out, but no carry into the high-order bit position. The overflow conditions are tested for by the computer and if a result exceeds the capacity of its destination, the OVF flop will automatically be set.

Examples of 2's complement notation using 8-bit numbers:

No Overflow:

```
EQ    GR    MI    OVF          +62 = 00111110
                               +27 = 00011011
                               ---   --------
0     1     0     0            +89   01011001
```

No Overflow:

```
        EQ     GR     MI     OVF            +62 = 00111110
                                            -27 = 11100101 (2's complement of 0011011 = 27)
        1      1      0      0              +35   1 ← carry in
                                                  00100011
                                                  1 ← carry out
```

Overflow:

```
        EQ     GR     MI     OVF          +  62 = 00111110
                                          +  89 = 01011001
        0      1      1      1            +151   1 ← carry in
                                                 10010111
                                                 no carry out,
```

whereas 151 exceeds 127, which is the maximum number which can be contained in an 8-bit number, an overflow results.

Overflow:

```
        EQ     GR     MI     OVF          -  62 = 11000010 (2's complement of 00111110 = 62)
                                          -  89 = 10100111 (2's complement of 01011001 = 89)
        1      1      0      1            -151   01101001 (2's complement of 10010111 = 151)
                                                 1 ← carry out
```

In this case there was no carry into the high order bit position, but there is a carry out indicating an overflow.

Unsigned Binary Arithmetic

Unsigned binary variable-length word arithmetic is also standard in the BIT 483. Such numbers are represented in a binary fashion, and the operations are conducted using 2's complement arithmetic. The values of the operands range from 0 to $2^{8n} - 1$, where n is the number of bytes in a word. All operands are considered as positive numbers.

As in the case of signed binary arithmetic, the arithmetic operations in unsigned arithmetic proceeds a byte at a time in parallel progressing from the addressed data descending sequentially until a word mark is sensed in either operand (ACC or memory word). The results will be positive or zero. If the result is a positive non-zero value, GR is set. If the result is zero, EQ is set.

If the word in memory contains more bytes than the accumulator, one of two results indicated below occurs.

1. If a carry from the word-marked bytes in the accumulator is generated, GR and EQ are set to indicate overflow and the instruction terminates. The following example illustrates the addition of a 16-bit number in memory to an 8-bit number in the accumulator with the results in the accumulator:

```
                                WM
ACC                             10101100          EQ     GR     MI     OVF
                       WM
X                      11011100 11010011
                       -----------------
                                WM
ACC Results                     01111111          1      1      0      1
                                1 ← carry out
```

2. If no carry from the word-marked byte in the ACC is generated, the instruction terminates with either GR or EQ set in accordance with the above rules. In neither case does the instruction continue beyond the first word mark encountered.

If the accumulator word contains more bytes than the word in memory, the same logical results are obtained. If a carry is generated by the accumulator byte corresponding to the memory byte containing the word mark, the instruction sets GR and EQ, indicating an overflow and terminates without examining or altering any additional bytes.

MI and OVF are not usually used by the program in unsigned binary arithmetic. The programmer is not restricted from using them, however, in the appropriate circumstances. The following flow chart, Figure 3-1, illustrates the method of setting the indicators in both signed and unsigned arithmetic.

BINARY INSTRUCTIONS

As noted earlier, the following binary operations use 2's complement arithmetic. The lengths of the binary operands are some multiple of 8-bit bytes. The operands need not be of the same length, since the operation ends when a word mark is found in either of the operands. The operands are considered either signed or unsigned by program control. During the process of operation four indicators can be set. They are EQ, GR, MI, and OVF. They are cleared at the beginning of all binary operations.

Format

0	1	0	1	M3	M2	M1	M0	LX

M3:	1 subtract 0 addition	M2:	1 high program counter (HPC) 0 page register (PR)
M1:	1 memory 0 accumulator	M0:	1 indirect address 0 direct address

Subtraction

Subtraction is performed in direct binary fashion, with the results stored in either the accumulator or memory. All results are represented in 2's complement notation. The following instructions are used in both signed and unsigned arithmetic, determined by the method by which the program tests the indicators EQ, GR, MI, and OVF.

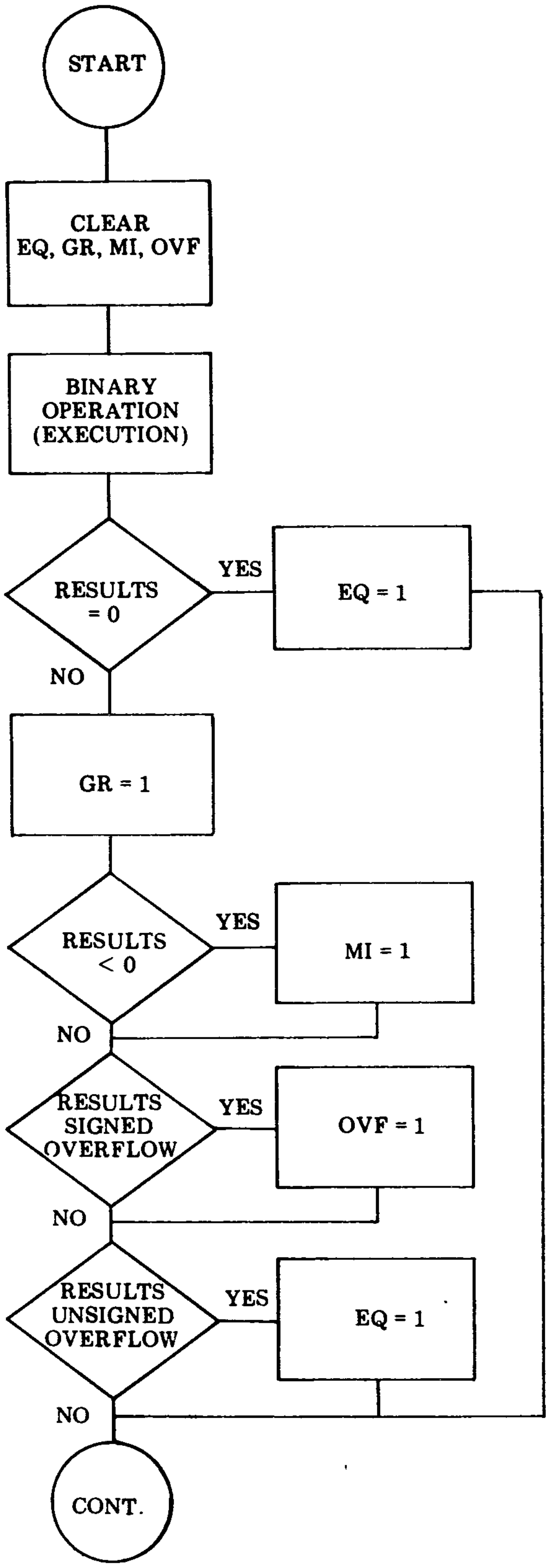


Figure 3-1. Control Flop Setting for Both Signed and Unsigned Arithmetic.

Subtract From Accumulator

The following instructions subtract the contents of the effective address from the word in the accumulator. The contents of the effective address are not changed.

BSA (130)	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(PR), LX
BSAP (134)	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(HPC), LX
BSAI (131)	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(PR), [(HPC),LX]
BSAPI (135)	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(HPC), [(HPC), LX]

Subtract From Memory

The following instructions subtract the contents of the accumulator from the word in memory located at the effective address. The contents of the accumulator are not changed.

BSM (132)	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(PR), LX
BSMP (136)	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(HPC), LX
BSMI (133)	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(PR), [(HPC), LX]
BSMPI (137)	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(HPC), [(HPC), LX]

Addition

Addition is also performed in direct binary fashion. All the result rules are identical to those for subtraction.

Add To Accumulator

The following instructions add the contents of the effective address to the word in the accumulator. The contents of the effective address are not changed.

BAA (120)	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(PR), LX
BAAP (124)	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(HPC), LX
BAAI (121)	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(PR), [(HPC), LX]
BAAPI (125)	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(HPC), [(HPC), LX]

Add To Memory

The following instructions add the contents of the accumulator to the word in memory located at the effective address. The contents of the accumulator are not changed.

BAM (122)	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(PR), LX
BAMP (126)	Timing:	4.21 μs + 2.25 μs/byte
	EFAD:	(HPC), LX
BAMI (123)	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(PR), [(HPC), LX]
BAMPI (127)	Timing:	5.19 μs + 2.25 μs/byte
	EFAD:	(HPC), [(HPC), LX]

PAGE REGISTER INSTRUCTIONS

The page register instruction controls and tests the page register. As discussed previously, the page register can be used as the higher order bits of the effective address when addressing data on a page other than the program counter page. Any single setting of the page register permits the programmer to work with one page of 256 data bytes directly. Each different setting of the page register designates another 256 byte page in memory. A setting of all 1's in the page register addresses the highest page in memory, the accumulator.

These instructions also allow for the testing of the sense switches on the control panel. The switches may be manually set while the program is operating.

Format

0	1	1	0	M3	M2	M1	M0	LX

M3: 1* store sense switch
0 page register operation

M2: 1 high program counter
0 page register

M1: 1 memory
0 page register

M0: 1 indirect address
0 direct address

*memory only

Change Page Register

This instruction causes the single byte (word marked or not) stored at the specified location to be inserted into the page register. This 8-bit value becomes the high-order half of the effective address in succeeding instructions where address control specifies the use of the page register.

CHP (140)	Timing: 2.94 μs	EFAD: (PR), LX
CHPP (144)	Timing: 2.94 μs	EFAD: (HPC), LX
CHPI (141)	Timing: 3.92 μs	EFAD: (PR), [(HPC), LX]
CHPPI (145)	Timing: 3.92 μs	EFAD: (HPC), [(HPC), LX]

Store Page Register

This instruction stores the contents of the page register at the effective address. One byte only is stored, having no effect on the word mark. As discussed in Section Two, the size of the page register is a function of memory size. The unused high-order bits of the result are set to 1's after the operation of this instruction.

SPR (142)	Timing: 3.23 μs	EFAD: (PR), LX
SPRP (146)	Timing: 3.23 μs	EFAD: (HPC), LX
SPRI (143)	Timing: 4.21 μs	EFAD: (PR), [(HPC), LX]
SPRPI (147)	Timing: 4.21 μs	EFAD: (HPC), [(HPC), LX]

Store Sense Switches

This instruction stores the sense-switch setting at the effective address. One byte only, including the word mark, is stored. This configuration may then be tested using normal test methods to determine what response is required by the program's logic.

SSS (152)	Timing:	3.23 μs
	EFAD:	(PR), LX
SSSP (156)	Timing:	3.23 μs
	EFAD:	(HPC), LX
SSSI (153)	Timing:	4.21 μs
	EFAD:	(PR), [(HPC), LX]
SSSPI (157)	Timing:	4.21 μs
	EFAD:	(HPC), [(HPC), LX]

DATA COPY INSTRUCTIONS

Data words are considered as some multiple of 8-bit bytes. They are usually addressed at the low-order byte, and transferred to sequential-descending memory locations until a byte with a word mark is transferred. All data and word marks in the destination field are lost. The word mark at the high-order byte of the data word is transferred with the byte. This effectively copies the entire data word from the address specified into the destination field with the word mark at the high-order byte. The copy instructions are not limited by page boundaries.

Format

0	1	1	1	M3	M2	M1	M0	LX

M3:	1 copy 2 words	M2:	1 high program counter (HPC)
	0 copy 1 word		0 page register (PR)
M1:	1 memory	M0:	1 indirect address
	0 accumulator		0 direct address

Copy One Word

This instruction copies one word which may be a single byte or group of bytes from the specified address to the accumulator or from the accumulator to the memory address specified. The source and destination are controlled by modifier bit M1.

Copy One Word To Accumulator

This instruction transfers a word, whose highest memory location is addressed, a byte at a time to the accumulator. The bytes placed in the accumulator replace its previous contents, erasing any word marks which might have been in the accumulator locations before. Thus, the length of the accumulator becomes the length of the word just placed into it.

$\boxed{\text{C1A (160)}}$	Timing: EFAD:	3.92 μs + 1.96 μs/byte (PR), LX
$\boxed{\text{C1AP (164)}}$	Timing: EFAD:	3.92 μs + 1.96 μs/byte (HPC), LX
$\boxed{\text{C1AI (161)}}$	Timing: EFAD:	4.9 μs + 1.96 μs/byte (PR), [(HPC), LX]
$\boxed{\text{C1API (165)}}$	Timing: EFAD:	4.9 μs + 1.96 μs/byte (HPC), [(HPC), LX]

Copy One Word To Memory

This instruction copies one word with its word mark from the accumulator into sequential-descending memory locations beginning at the effective address.

$\boxed{\text{C1M (162)}}$	Timing: EFAD:	3.92 μs + 1.96 μs/byte (PR), LX
$\boxed{\text{C1MP (166)}}$	Timing: EFAD:	3.92 μs + 1.96 μs/byte (HPC), LX
$\boxed{\text{C1MI (163)}}$	Timing: EFAD:	4.9 μs + 1.96 μs/byte (PR), [(HPC), LX]
$\boxed{\text{C1MPI (167)}}$	Timing: EFAD:	4.9 μs + 1.96 μs/byte (HPC), [(HPC), LX]

Copy Two Words

Beginning at the specified address, two consecutive words are copied to the accumulator or from the accumulator to the memory address specified. The source and destination are controlled by modifier bit M1.

Copy Two Words To Accumulator

This instruction copies two words adjacent in memory with both word marks into the accumulator. The transfer begins exactly as a C1A instruction. The instruction does not terminate when the first word mark is encountered in the operand, however, but continues sequentially through descending memory address locations until it encounters a second word mark. After encountering the second word mark, execution terminates.

C2A (170)	Timing:	5.88 μs + 1.96 μs/byte
	EFAD:	(PR), LX
C2AP (174)	Timing:	5.88 μs + 1.96 μs/byte
	EFAD:	(HPC), LX
C2AI (171)	Timing:	6.86 μs + 1.96 μs/byte
	EFAD:	(PR), [(HPC), LX]
C2API (175)	Timing:	6.86 μs + 1.96 μs/byte
	EFAD:	(HPC), [(HPC), LX]

Copy Two Words To Memory

This instruction copies two words with both word marks from the accumulator to sequential-descending memory locations starting at the address specified.

C2M (172)	Timing:	5.88 μs + 1.96 μs/byte
	EFAD:	(PR), LX
C2MP (176)	Timing:	5.88 μs + 1.96 μs/byte
	EFAD:	(HPC), LX
C2MI (173)	Timing:	6.86 μs + 1.96 μs/byte
	EFAD:	(PR), [(HPC), LX]
C2MPI (177)	Timing:	6.86 μs + 1.96 μs/byte
	EFAD:	(HPC), [(HPC), LX]

WORD-MARK INSTRUCTIONS

The word-mark bit is the ninth bit of each byte throughout memory. It is not used as an additional bit of magnitude of the byte, but for defining the length of a word. It may be set or reset at any byte. It may also be tested as to status by a 4-byte jump instruction.

Format

Word Mark Control

1	0	0	0	M3	M2	M1	M0	LX

M3: 1 test word mark*
0 word mark control

M2: 1 high program counter
0 page register

M1: 1 erase word mark
0 set word mark

M0: 1 indirect address
0 direct address

*specifies a 4-byte jump on word mark operation

Four-Byte Jump on Word Mark

1	0	0	0	M3	M2	M1	M0	LX

HIGH	LOW

Jump Address

M3:	1 test word mark 0 word mark control*	M2:	1 high program counter 0 page register
M1:	unused	M0:	1 indirect address 0 direct address

*specifies a word mark control operation

Erase Word Mark

This instruction erases the word-mark bit at the effective address. The previous state of this bit makes no difference. This instruction operates on a single byte in memory.

EWM (200)	Timing: EFAD:	3.23 μs (PR), LX
EWMP (204)	Timing: EFAD:	3.23 μs (HPC), LX
EWMI (201)	Timing: EFAD:	4.21 μs (PR), [(HPC), LX]
EWMPI (205)	Timing: EFAD:	4.21 μs (HPC), [(HPC), LX]

Set Word Mark

This instruction sets the word mark bit at the effective address. The previous status of the bit makes no difference. This instruction operates on a single byte in memory.

SWM (202)	Timing:	3.23 μs
	EFAD:	(PR), LX
SWMP (206)	Timing:	3.23 μs
	EFAD:	(HPC), LX
SWMI (203)	Timing:	4.21 μs
	EFAD:	(PR), [(HPC), LX]
SWMPI (207)	Timing:	4.21 μs
	EFAD:	(HPC), [(HPC), LX]

Jump on Word Mark

The jump on word mark instruction is a 4-byte test and jump instruction. The second byte of the instruction is combined with either the page register or HPC to specify the byte in memory which is to be tested. This address calculation follows the normal rules of calculating the effective address of an operand. The word mark bit at the effective address is tested. If it is set, the two low-order bytes of the instruction (jump address) are placed into the program counter, and control is then transferred to the instruction sequence at that location. If the word mark is not set at the effective address, the program counter is incremented by two, thus bypassing the jump address and operating the next sequential instruction.

JWM (210)	Timing:	4.9 μs
	EFAD:	(PR), LX
JWMP (214)	Timing:	4.9 μs
	EFAD:	(HPC), LX
JWMI (211)	Timing:	5.88 μs
	EFAD:	(PR), [(HPC), LX]
JWMPI (215)	Timing:	5.88 μs
	EFAD:	(HPC), [(HPC), LX]

TWO-BYTE JUMP INSTRUCTIONS

Two-byte jump instructions permit a jump to any location in either the program counter page or the page represented in the page register. There are three variations of the 2-byte jump, but all have the one common result of being unconditional when the jump is executed. Unlike the 4-byte jump instructions, only one program statement is written for the 2-byte jump.

Format

1	0	0	1	M3	M2	M1	M0	LX

M3: 1 jump and halt
0 jump

M2: 1 high program counter
0 page register

M1: 1 set interrupt
0 unconditional jump

M0: 1 indirect address
0 direct address

Jump and Halt

This instruction replaces the program counter with the effective address. The computer then halts, retaining the status of all indicators and hardware registers. It is possible to restart the program at the memory location represented by the program counter, which now contains the EFAD of the jump and halt instruction, by pressing the RUN switch on the console.

HLT (230)	Timing:	2.94 μs
	EFAD:	(PR), LX
HLTP (234)	Timing:	2.94 μs
	EFAD:	(HPC), LX
HLTI (231)	Timing:	3.92 μs
	EFAD:	(PR), [(HPC), LX]
HLTPI (235)	Timing:	3.92 μs
	EFAD:	(HPC), [(HPC), LX]

Two-Byte Jump

This is a 2-byte instruction which causes an unconditional jump to a memory location within the program counter page or the memory page referenced by the page register.

PGJ (220)	Timing:	2.94 μs
	EFAD:	(PR), LX
PGJP (224)	Timing:	2.94 μs
	EFAD:	(HPC), LX
PGJI (221)	Timing:	3.92 μs
	EFAD:	(PR), [(HPC), LX]
PGJPI (225)	Timing:	3.92 μs
	EFAD:	(HPC), [(HPC), LX]

Set Interrupt and Jump

This instruction causes a program interrupt that places the central processor in the interrupt mode. The sequence of events is as follows:

1. Place the effective address in the program counter.
2. Set interrupt condition for level 2.
3. Store the program counter, i.e., the EFAD of the set interrupt and jump instruction, in address storage for level 2.
4. Store control flops in control flop storage of level 2.
5. Jump in interrupt mode to address 002 in highest page in memory.

SIF (222)	Timing:	2.94 μs
	EFAD:	(PR), LX
SIFP (226)	Timing:	2.94 μs
	EFAD:	(HPC), LX
SIFI (221)	Timing:	3.92 μs
	EFAD:	(PR), [(HPC), LX]
SIFPI (227)	Timing:	3.92 μs
	EFAD:	(HPC), [(HPC), LX]

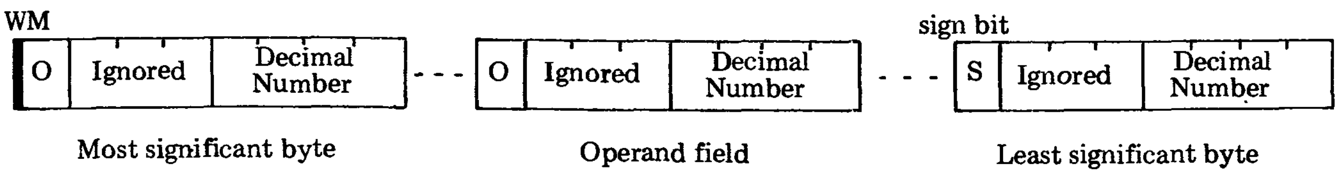
DECIMAL OPERATIONS

A decimal operand is a variable length, signed number starting at the byte in the specified location (least significant digit) and stored in sequentially descending memory locations to the first byte with a word mark (most significant digit). Each byte contains one binary coded decimal digit in its low-order four bits. The sign of the operand is indicated in the high-order bit of the least-significant byte. The following illustrates a 3-byte positive base 10 number and a 3-byte negative base 10 number.

$$+901 = 071\ 060\ 061_8 = 00111001\ 00110000\ 00110001_2$$

$$-852 = 070\ 065\ 262_8 = 00111000\ 00110101\ 10110010_2$$

A 0 in the sign-bit position of the least significant byte indicates a positive number; a 1 in this sign-bit position indicates a negative number. Except for the sign bit in the least significant byte of the number, the high order bits are ignored in the operand field. The sign bit is changed, if the result sign is different than the sign originally in the accumulator. The high order bit of the remaining bytes is always set to 0. The following illustrates the format of any decimal number:



Most significant byte   Operand field   Least significant byte

Decimal Operand

In cases where different length operands are used, the accumulator operand must be at least as long as the operand at the effective address. If the accumulator operand is larger than the operand at the effective address, zeros are assumed until a word mark is detected in the accumulator. The following addition illustrates this rule:

```
                                    WM
ACC       432 = 064  063  062₈ = 00110100  00110011  00110010
                                              WM
X        + 10 =      068  260₈ = 00000000*00111000  00110000
         ____________________________________________________
                                    WM
ACC      +442   065  061  062₈   00110101  00110001  00110010
                                 *zero assumed
```

When performing this addition, a word mark is first encountered in X. From this point, until a word mark is encountered in the accumulator, zero bytes are used with subsequent accumulator bytes.

If an alphanumeric character is sensed in either operand whose low-order four bits are greater than 9, that is greater than 1001_2, the bytes involved are ignored and any carry from the preceding byte pair is passed on to the next byte pair. Such a character causes no change whatever to the character in the result location byte.

Thus, pre-edited numbers may be operated upon with extracting the non-numeric bytes, compacting them, and later inserting the results back into the edited data field. This feature allows the programmer to add directly such numbers as 1,730.90 and 2,943.47 to get 4,674.37, without manipulating either commas or decimal points, as long as they are aligned. It is necessary to align the non-numeric fields. The following examples illustrate this editing feature:

```
              Correct                                Incorrect

ACC            01.43                                   21.37

X             +21.37                                    14.3
              ------                                   -----
               22.80                                   23.30
```

DECIMAL INSTRUCTIONS

The destination of the results of any decimal operation is the accumulator. EQ and GR are set as a function of every decimal operation. If the result is zero, EQ is set. If the result is greater than zero, GR is set. If the result is less than zero, neither is set. If overflow occurs, both GR and EQ are set. MI and OVF are not affected by any decimal operation.

Format

1	0	1	0	M3	M2	M1	M0	LX

M3: 1 decimal subtract
    0 decimal addition

M2: 1 high program counter
    0 page register

M1: 1 illegal
    0 accumulator

M0: 1 indirect address
    0 direct address

Decimal Subtract

This instruction subtracts the word in memory located at the effective address from the word in the accumulator, treating each byte as a decimal integer.

$\boxed{\text{DSA (250)}}$

Timing: 4.21 μs + 2.25 μs/byte
EFAD: (PR), LX

$\boxed{\text{DSAP (254)}}$

Timing: 4.21 μs + 2.25 μs/byte
EFAD: (HPC), LX

$\boxed{\text{DSAI (251)}}$

Timing: 5.19 μs + 2.25 μs/byte
EFAD: (PR), [(HPC), LX]

$\boxed{\text{DSAPI (255)}}$

Timing: 5.19 μs + 2.25 μs/byte
EFAD: (HPC), [(HPC), LX]

Decimal Add

This instruction adds the word in memory located at the effective address to the word in the accumulator, treating each byte as a decimal integer.

$\boxed{\text{DAA (240)}}$

Timing: 4.21 μs + 2.25 μs/byte
EFAD: (PR), LX

$\boxed{\text{DAAP (244)}}$

Timing: 4.21 μs + 2.25 μs/byte
EFAD: (HPC), LX

$\boxed{\text{DAAP (241)}}$

Timing: 5.19 μs + 2.25 μs/byte
EFAD: (PR), [(HPC), LX]

$\boxed{\text{DAAPI (245)}}$

Timing: 5.19 μs + 2.25 μs/byte
EFAD: (HPC), [(HPC), LX]

section four

input/output

The I/O system of the BIT 483 consists of a number of data channels each capable of operating in a Read-Write mode. The processor may have one, two or three such Read-Write channels. These channels are called channel C, channel G and channel A. Channel C and channel G are standard equipment with all BIT 483 computer systems.

All I/O channels have direct memory access capability. Through the control of one instruction, data may be transferred to or from any memory location. Furthermore, channel A or channel G may be used in either an overlapped or non-overlapped mode. This permits data transfers while the processor is simultaneously computing.

I/O INSTRUCTIONS

There are three instructions which are used by the computer to control the I/O system. They are:

Peripheral Control Instruction
Peripheral Transfer Instruction
Peripheral Test and Jump Instruction

Peripheral Control Instruction

The Peripheral Control Instruction (PCI) is used to select and command a peripheral device to perform. The function which it performs is controlled by the second byte of this 2-byte instruction. The following diagram illustrates the general format:

0	0	1	1	I/O	Device	Channels	Control

As in all other 2-byte instructions, the first byte of this instruction indicates the operation to be performed by the central processor. This operation is indicated by the high-order four bit operation code. The operation is further defined by the field designated I/O and the field designated device.

The I/O bit signifies to the I/O device whether the data transfer is to be In or Out of the computer. The significance of the I/O bit is as follows:

I/O = 1	Transfer to device
I/O = 0	Transfer from device

The device field contains the address unique to the desired peripheral device located on the channel specified. Only the device addressed will respond to the command.

The second byte of this instruction contains two significant fields. The high order two bits are used to define one of three possible I/O channels. Two of these channels, A and G, permit simultaneous I/O and compute allowing the execution of data transfers by stealing memory cycles of the core memory only when necessary. The remaining channel, channel C, is dedicated to the standard teletype supplied with the 483. The channel address bits are assigned to data channels as follows:

Channel Address	Channels
00	A
10	C
11	G

The control bits define the operation to be performed on the I/O device which is being addressed. There are four general classes of operation which may be specified by the control bits. Each class is specified by the high order two bits of the control field. In general, the functions are:

Class	Value	Function
I	00	Initiate backspace, rewind, etc.
II	01	Select an I/O device for data transfer
III	10	Select an I/O device for test
IV	11	Control device interrupt capability

The remaining four bits in the field further define operation. The I/O Manual (document No. 48307) explains in detail the use of the control field. As an addendum to this explanation the individual peripheral specifications must be used as the source of the exact bit configuration for each operation the peripheral is capable of performing.

$\boxed{\text{PCID (06D)}}$

Timing: 3.0 μs
EFAD: N/A

The device whose address is D is selected to do the operation indicated by the control field. The transfer is considered to be an input from the device to the computer.

$\boxed{\text{PCOD (07D)}}$

Timing: 3.0 μs
EFAD: N/A

This instruction causes the same result as the PCI except the operation is considered to be an output operation rather than an input operation.

The standard I/O element of the 483 system is an ASR-33 teletype with a paper tape reader and paper tape punch. Channel C is dedicated to this unit. The channel is fully buffered and each device may operate in either the interrupt or non-interrupt mode. The mode of operation is controlled by a select group of PCI instructions. Once a device is selected to operate in either of the two modes it remains in that mode until it is reselected to operate in the alternative mode. If it is desired for a device to operate in the non-interrupt mode and it has not been previously selected to run in the interrupt mode, the PCI instruction is not necessary. The following Peripheral Control Instructions perform the indicated functions:

Printer

0	0	1	1	0	0	1	1		1	0	1	1	0	0	0	0

Select Interrupt Mode (PCI3 00260)

0	0	1	1	0	0	1	1		1	0	1	1	0	0	0	1

Select Non-Interrupt Mode (PCI3 00261)

If the interrupt mode has been selected for the printer, the interrupt will not occur until a peripheral transfer instruction has been executed. Upon this execution the central processor will be placed in the interrupt mode and the following will occur:

1. Set the interrupt condition for level 3.
2. Store the program counter in the address storage locations of level 3.
3. Store the control flops in the control flop storage location of level 3.
4. Set program counter equal to address 376 362 and transfer execution to this location.

Reader

0	0	1	1	0	0	1	0		1	0	1	1	0	0	0	0

Select Interrupt Mode (PCI2 00260)

0	0	1	1	0	0	1	0		1	0	1	1	0	0	0	1

Select Non-Interrupt Mode (PCI2 00261)

If the interrupt mode has been selected for the reader, the results are the same as for those previously discussed for the printer.

Keyboard

0	0	1	1	0	0	0	0		1	0	1	1	0	0	0	0

Select Interrupt Mode (PCI0 00260)

0	0	1	1	0	0	0	0		1	0	1	1	0	0	0	1

Select Non-Interrupt Mode (PCI0 00261)

If the interrupt mode has been selected for the keyboard, the computer may simultaneously compute while waiting for a keyboard input. Upon depressing a key on the keyboard an interrupt will occur placing the central processor in the interrupt mode. Level 3 interrupt condition will be entered as previously discussed as in the case of the printer. To input the data to memory from the data buffer of the channel, a peripheral transfer instruction must be executed following the interrupt.

Peripheral Transfer Instruction

This instruction causes data (single byte or string of bytes) to be accepted from or addressed to the data channel specified. The object I/O device must have been previously selected for the transfer by the PCI instruction.

Data is moved to or from memory at sequential ascending locations beginning with the byte in the location specified by the address field entry. This address will refer to a location within the memory page specified by the page register. For any data channel the transfer will continue across memory page boundaries (without affecting the page register) so that records of any length up to memory capacity may be read in or out with a single command. The length of the data transfer is controlled by the PCI instruction to select the data transfer to or from the object device. The transfer mode (overlapped or non-overlapped) is defined by the transfer instruction itself.

Format

0	0	0	1	M3	M2	M1	M0	LX

Channel C

M3	M2	M1	M0	Function
1	0	0	0	Keyboard input
1	0	0	1	Print
1	0	1	0	Read paper tape

Channel A

M3	M2	M1	M0	Function
0	0	0	0	Non-overlapped transfer
0	0	0	1	Overlapped transfer

Channel G

M3	M2	M1	M0	Function
1	1	0	0	Non-overlapped transfer
1	1	0	1	Overlapped transfer

Modifier bits M3 and M2 identify the channel on which the transfer is to take place. Modifier bits M1 and M0 define the transfer mode for channels A and G; for channel C they identify the I/O function to be performed on the teletype.

PTA (020)

Timing: 3.0 μs + 1.0 μs/byte (output)
3.0 μs + 1.3 μs/byte (input)
EFAD: (PR), LX

Initiates a non-overlapped data transfer on channel A beginning with the byte at the effective address. Computer is stalled until the transfer is completed.

OTA (021)

Timing: 3.0 μs + 1.0 μs/byte (output)
3.0 μs + 1.3 μs/byte (input)
EFAD: (PR), LX

Initiates an overlapped data transfer on channel A beginning with the byte at the effective address. This mode allows simultaneous computation while the transfer is taking place since the I/O logic steals memory cycles only when necessary.

PTG (034)

Timing: 3.0 μs + 1.0 μs/byte (output)
3.0 μs + 1.3 μs/byte (input)
EFAD: (PR), LX

Initiates a non-overlapped data transfer on channel G beginning with the byte at the effective address. Computer remains dedicated to the I/O function until the transfer is completed.

OTG (035)

Timing: 3.0 μs + 1.0 μs/byte (output)
3.0 μs + 1.3 μs/byte (input)
EFAD: (PR), LX

This instruction works exactly as OTA except the transfer is directed to the G channel.

KEY (030)

Timing: 100 msec
EFAD: (PR), LX

In the non-interrupt mode this instruction turns on the "type light" on the ASR-33. The computer is "idled" until a keyboard key is depressed. The bit pattern associated with the key is then transferred into the computer and the "type light" goes off. Characters associated with any key are considered legal and are transferred into the memory locations specified by the instruction.

In the interrupt mode depressing a key on the keyboard causes the buffer register to be loaded with the 8-bit character code. As soon as the register is loaded, an interrupt is created for level 3 of the priority interrupt system. The subsequent processing of this instruction to the keyboard causes the contents of the buffer register to be loaded into the location specified by the effective address. The type light is also on in this mode after the operation of the PCI instruction and until the interrupt occurs.

PRT (031)		
	Timing:	100 msec
	EFAD:	(PR), LX

This instruction causes the teletype buffer register to be loaded with the contents of the effective address. The character is then printed and the instruction ends. If the punch is on, the bit pattern is duplicated on the paper tape. The punch control must be manually activated.

The printer may be operated in either the interrupt or non-interrupt mode. If the printer is in the interrupt mode, an interrupt will occur for level 3 after the buffer register is emptied. If the printer is not in the interrupt mode, control proceeds to the next sequential instruction.

RPT (032)		
	Timing:	100 msec
	EFAD:	(PR), LX

This instruction causes the 8-bit character over the read fingers to be transferred into the teletype buffer register. The character is then loaded into the location specified by the effective address. The paper tape is also advanced one character position.

If the reader is in the interrupt mode of operation, an interrupt will occur for level 3 after the instruction ends. If the reader is not in the interrupt mode, control proceeds to the next sequential instruction.

Peripheral Test and Jump

This is a 4-byte jump instruction in which the low-order 6 bits of the second byte define the condition being tested and in which a reply from the addressed device causes the jump to occur. "No reply" causes the program counter to be incremented by two, thereby bypassing the jump address and operating the next sequential instruction. The peripheral device being tested must have been previously selected for test by a PCI instruction.

The bit configuration defining individual tests for a given peripheral device are explained and identified in the individual peripheral equipment specifications provided by BIT.

Format:

0	0	1	0	1	0	0	0

PA5	PA4	Test Condition

HX	LX

This jump instruction is a variant of the 4-byte jump instruction explained in Section Three.

JPT (050)		
	Timing:	5.0 μs
	EFAD:	N/A

section five

interrupt

GENERAL

The BIT 483 is equipped with an extensive, multi-level priority interrupt system. This allows the normal execution of a program to be interrupted in order to process a program of higher priority or to service a peripheral device. The interrupt may occur on one of eight standard levels. The system may be expanded in groups of eight up to 32 levels. Figure 5-1 illustrates the configuration of the Interrupt System.

The central processor enters the Interrupt Mode when an interrupt is encountered. This mode is maintained until a Jump Return Interrupt (JRI) instruction is executed from the appropriate location. When entering the Interrupt Mode the hardware automatically saves the necessary internal elements in the reserved location of the subject level. The program counter is then altered to transfer control to the interrupt subroutine which will service the interrupt call.

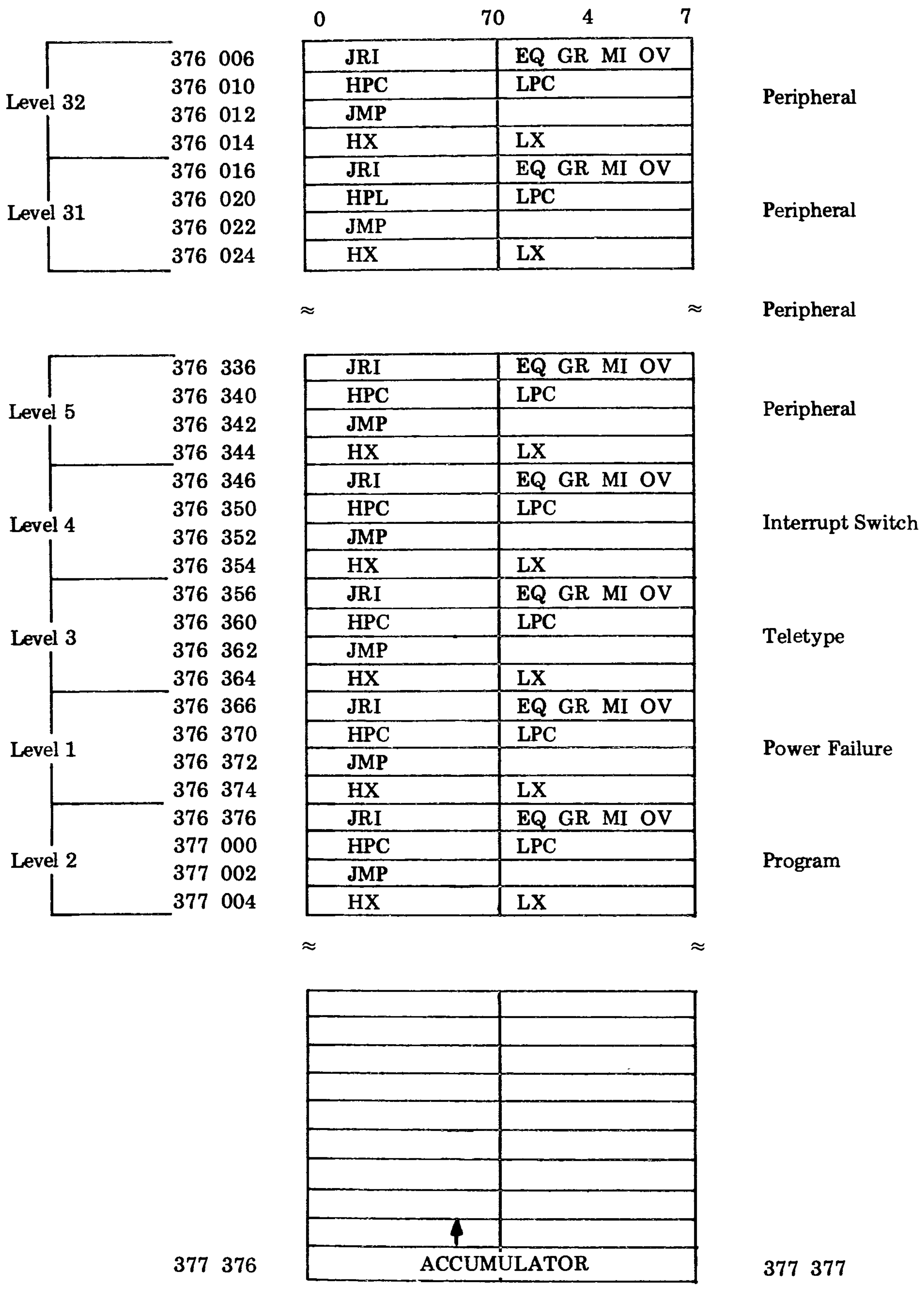
An interrupt at a given priority level will interrupt all outstanding priority interrupts of lower value. If, for example, another interrupt occurs while a prior interrupt is being serviced, the subroutine will be interrupted if it is lower on the priority chain. Conversely, if the additional interrupt is of a lower priority level, the interrupt will not be serviced until all higher levels have been serviced.

Enabling an Interrupt

Interrupt levels 1, 2 and 4 are not dedicated to peripheral devices. They are automatically enabled to interrupt by hardware action and require no special program activity. The remaining levels, however, are usually dedicated to peripheral devices. In most instances it is possible to operate these peripherals in either the interrupt or non-interrupt mode. The mode of operation is controlled by a Peripheral Control Instruction (PCI) discussed in Section Four. The PCI both enables and disables the interrupt capability of a device. Once in either mode a subsequent PCI must be issued to change that mode. A Master Clear places all peripheral devices in the non-interrupt mode.

Priority Chain

An interrupt request from the priority chain is honored following the completion of the instruction under execution. If an overlapped data transfer is in progress the transfer will continue although an interrupt is encountered. Again, an interrupt at a given priority level cannot interrupt any higher priority interrupts. However, it will interrupt any lower priority interrupt. The standard levels of priority are given below. If an interrupt is requested when an interrupt exists, it will enter the waiting state until no higher priority interrupt is waiting or is being serviced. If more than one interrupt enters the waiting state, each will be honored by the central processor according to assigned priority.



Level	Address	0 – 7	0 – 4 – 7	
Level 32	376 006	JRI	EQ GR MI OV	Peripheral
	376 010	HPC	LPC	
	376 012	JMP		
	376 014	HX	LX	
Level 31	376 016	JRI	EQ GR MI OV	Peripheral
	376 020	HPL	LPC	
	376 022	JMP		
	376 024	HX	LX	
		≈	≈	Peripheral
Level 5	376 336	JRI	EQ GR MI OV	Peripheral
	376 340	HPC	LPC	
	376 342	JMP		
	376 344	HX	LX	
Level 4	376 346	JRI	EQ GR MI OV	Interrupt Switch
	376 350	HPC	LPC	
	376 352	JMP		
	376 354	HX	LX	
Level 3	376 356	JRI	EQ GR MI OV	Teletype
	376 360	HPC	LPC	
	376 362	JMP		
	376 364	HX	LX	
Level 1	376 366	JRI	EQ GR MI OV	Power Failure
	376 370	HPC	LPC	
	376 372	JMP		
	376 374	HX	LX	
Level 2	376 376	JRI	EQ GR MI OV	Program
	377 000	HPC	LPC	
	377 002	JMP		
	377 004	HX	LX	
		≈	≈	
	377 376	ACCUMULATOR		377 377

Figure 5-1. Interrupt Structure.

Interrupt Priorities

Priority	Description	Address
1	Power Failure	376 366
2	Two-Byte Jump	376 376
3	Teletype	376 356
4	Interrupt Switch	376 346
5	Peripheral	376 336
6	Peripheral	376 326
7	Peripheral	376 316
8	Peripheral	376 306

Preservation of Machine State

Within the computer there are a number of indicators and registers which constitute the normal mode configuration of the machine. During the initiation of an interrupt subroutine, part or all of this configuration must be preserved for a subsequent return. Generally, the following elements should be preserved:

A. EQ, GR, MI, and OVF Indicators
B. Program Counter
C. Page Register
D. Accumulator

A. *Indicators*

The EQ, GR, MI, and OVF indicators are stored by hardware action in the second byte of the interrupt level requesting an interrupt. They are stored in the four least significant bits as shown below:

				EQ	GR	MI	OVF

A value of one in any of these bit positions indicates the associated indicator is set, and a zero value indicates that it is reset. Upon executing a JRI instruction, the flops will be reset to the values indicated by these bit positions. If this byte is altered during the interrupt subroutine, it will be reflected when the JRI is executed, thereby not restoring the original indicator configuration.

B. *Program Counter*

The Program Counter, during the Normal Mode, contains the address of the next instruction to be operated. This register is automatically saved by hardware action in the third and fourth bytes of the interrupt level requesting the interrupt (figure 5-1 illustrates the storage locations for each level). The third byte of the interrupt will contain the High Program Counter (HPC), and the fourth byte will contain the Low Program Counter (LPC). The original state of the word marks at these locations are maintained.

C. *Page Register*

Because the page register may be used extensively for addressing during the Normal Mode, it should be saved by the interrupt subroutine. It may be preserved by issuing a Store Page Register instruction (see Section 3). Upon exiting from the interrupt subroutine it may be restored by issuing a Change Page Register instruction.

D. *Accumulator*

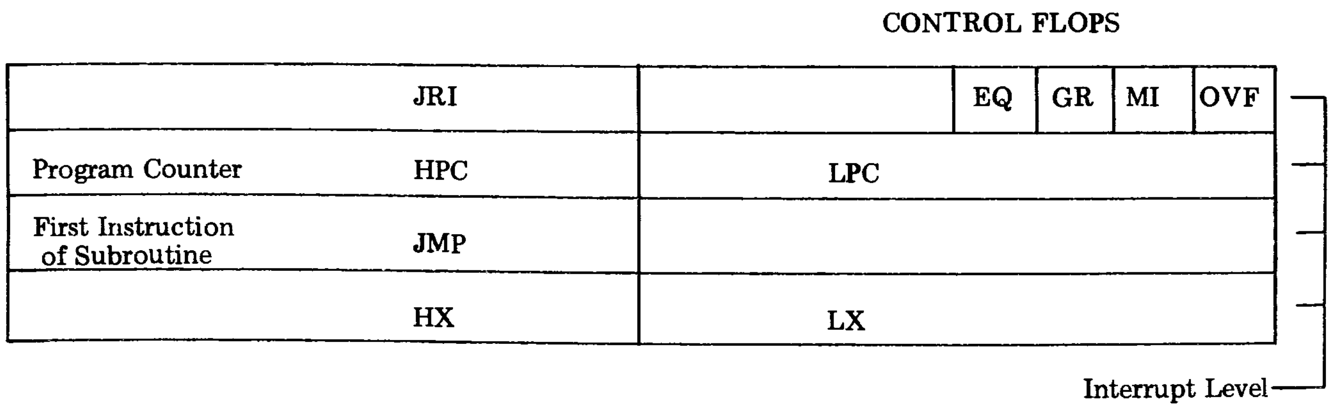
The accumulator is also an essential element of the Normal Mode configuration. It, like the Page Register, is preserved through program action at the beginning of the interrupt subroutine and is restored at the end of the subroutine. The subroutine must guarantee the preservation of all meaningful information in the accumulator by storing and restoring all the bytes of the accumulator which may be destroyed by the interrupt subroutine.

Initiation of the Interrupt

During both the Normal Mode and the Interrupt Mode, the computer tests for the existence of an interrupt request, just before each instruction fetch. Upon detection of a request, the computer copies the program counter into the third and fourth bytes of the requesting level. The interrupt subroutine must be arranged such that the stored program counter constitutes the address portion of a "Jump Return from Interrupt" instruction (see figure 5-1). This instruction, when executed, restores both the requesting level and the computer to the operating mode preceding the interrupt and returns control to the interrupted program.

The states of EQ, GR, MI, and OVF are also saved by hardware action. They are stored in the second byte of the requesting level. Upon executing the Jump Return from Interrupt instruction for this level, they are reinstated from the stored location. If this byte is altered during the interrupt subroutine, EQ, GR, MI and OVF may not be restored to their original configuration.

After the computer terminates its housekeeping activities, the program counter is set to execute the instruction at the fetch byte of the requesting level. This instruction should be an unconditional jump to the interrupt subroutine, thereby permitting the interrupt to be serviced. The last instruction of the interrupt subroutine should generally be a jump to the location containing the JRI. A typical interrupt level configuration is illustrated below:



appendix A

Numeric List of Instructions

Code	Mnemonic	Code	Mnemonic
020	PTA	126	BAMP
021	OTA	127	BAMPI
030	KEY	130	BSA
031	PRT	131	BSAI
032	RPT	132	BSM
034	PTG	133	BSMI
035	OTG	134	BSAP
040	JRI	135	BSAPI
041	JMP	136	BSMP
042	JEQ	137	BSMPI
043	NEQ	140	CHP
044	JGR	141	CHPI
045	NGR	142	SPR
046	JGE	143	SPRI
047	NGE	144	CHPP
050	JPT	145	CHPPI
051	JSR	146	SPRP
052	JMI	147	SPRPI
053	NMI	152	SSS
054	JOV	153	SSSI
055	NOV	154	SSS
056	EOG	155	SSSPI
057	ENG	160	C1A
060	PCI	161	C1AI
070	PCO	162	C1M
100	ORA	163	C1MI
101	ORAI	164	C1AP
102	ORM	165	C1API
103	ORMI	166	C1MP
104	ORAP	167	C1MPI
105	ORAPI	170	C2A
106	ORMP	171	C2AI
107	ORMPI	172	C2M
110	ANA	173	C2MI
111	ANAI	174	C2AP
112	ANM	175	C2API
113	ANMI	176	C2MP
114	ANAP	177	C2MPI
115	ANAPI	200	EWM
116	ANMP	201	DWMI
117	ANMPI	202	SWM
120	BAA	203	SWMI
121	BAAI	204	EWMP
122	BAM	205	EWMPI

Code	Mnemonic	Code	Mnemonic
123	BAMI	206	SWMP
124	BAAP	207	SWMPI
125	BAAPI	210	JWM
211	JWMI	231	HLTI
214	JWMP	234	HLTP
215	JWMPI	235	HLTPI
220	PGJ	240	DAA
221	PGJI	241	DAAI
222	SIF	244	DAAP
223	SIFI	245	DAAPI
224	PGJP	250	DSA
225	PGJPI	251	DSAI
226	SIFP	254	DSAP
227	SIFPI	255	DSAPI
230	HLT		

appendix B

Alphabetic List of Instructions

Mnemonic	Code	Mnemonic	Code
ANA	110	DSA	250
ANAI	111	DSAI	251
ANAP	114	DSAP	254
ANAPI	115	DSAPI	255
ANM	112	ENG	057
ANMI	113	EOG	056
ANMP	116	EWM	200
ANMPI	117	EWMI	201
BAA	120	EWMP	204
BAAI	121	EWMPI	205
BAAP	124	HLT	230
BAAPI	125	HLTI	231
BAM	122	HLTP	234
BAMI	123	HLTPI	235
BAMP	126	JEQ	042
BAMPI	127	JGE	046
BSA	130	JGR	044
BSAI	131	JMI	052
BSAP	134	JMP	041
BSAPI	135	JOV	054
BSM	132	JPT	050
BSMI	133	JRI	040
BSMP	136	JSR	051
BSMPI	137	JWM	210
CHP	140	JWMI	211
CHPI	141	JWMP	214
CHPP	144	JWMPI	215
CHPPI	145	KEY	030
C1A	160	NEQ	043
C1AI	161	NGE	047
C1AP	164	NGR	045
C1API	165	NMI	053
C1M	162	NOV	055
C1MI	163	ORA	100
C1MP	166	ORAI	101
C1MPI	167	ORAP	104
C2A	170	ORAPI	105
C2AI	171	ORM	102
C2AP	174	ORMI	103
C2API	175	ORMP	106
C2M	172	ORMPI	107
C2MI	173	OTA	021
C2MP	176	OTG	035
C2MPI	177	PCI	060
DAA	240	PCO	070
DAAI	241	PGJ	220
DAAP	244	PGJI	221
DAAPI	245	PGJP	224

Mnemonic	Code
PGJPI	225
PRT	031
PTA	020
PTG	034
RPT	032
SIF	222
SIFI	223
SIFP	226
SIFPI	227
SPR	142
SPRI	143

Mnemonic	Code
SPRP	146
SPRPI	147
SSS	152
SSSI	153
SSSP	156
SSSPI	157
SWM	202
SWMI	203
SWMP	206
SWMPI	207

appendix C

The ASCII Teletype Code (Octal and Binary)

Character	Octal	Binary	Character	Octal	Binary
A	301	11000001	!	241	10100001
B	302	11000010	“	242	10100010
C	303	11000011	#	243	10100011
D	304	11000100	$	244	10100100
E	305	11000101	%	245	10100101
F	306	11000110	&	246	10100110
G	307	11000111	’	247	10100111
H	310	11001000	(	250	10101000
I	311	11001001	)	251	10101001
J	312	11001010	*	252	10101010
K	313	11001011	+	253	10101011
L	314	11001100	,	254	10101100
M	315	11001101	—	255	10101101
N	316	11001110	.	256	10101110
O	317	11001111	/	257	10101111
P	320	11010000	:	272	10111010
Q	321	11010001	;	273	10111011
R	322	11010010	<	274	10111100
S	323	11010011	=	275	10111101
T	324	11010100	>	276	10111110
U	325	11010101	?	277	10111111
V	326	11010110	@	300	11000000
W	327	11010111	[	333	11011011
X	330	11011000	\	334	11011100
Y	331	11011001	]	335	11011101
Z	332	11011010	↑	336	11011110
			←	337	11011111
0	260	10110000	Line/Feed	212	10001010
1	261	10110001	Carriage/Return	215	10001101
2	262	10110010	Space	240	10100000
3	263	10110011	Rub-Out	377	11111111
4	264	10110100			
5	265	10110101			
6	266	10110110			
7	267	10110111			
8	270	10111000			
9	271	10111001			

BIT 483 SERIES PROGRAMMING CARD

BIT INC. 5 STRATHMORE RD. NATICK MASS. TELE: 617 - 237-2930 TWX 710 - 386-6494

TWO BYTE INSTRUCTIONS

Instruction	Mnemonic	Octal	Execute Timing (MCT)	Fetch Timing (MCT)
Jam Copy[1]				
One word to ACC	C1A	160	2 0	2
	C1AP	164	2 0	2
	C1AI	161	2 0	3
	C1API	165	2 0	3
One word to mem	C1M	162	2 0	2
	C1MP	166	2.0	2
	C1MI	163	2.0	3
	C1MPI	167	2 0	3
Two words to ACC	C2A	170	4.0	2
	C2AP	174	4 0	2
	C2AI	171	4.0	3
	C2API	175	4 0	3
Two words to mem	C2M	172	4.0	2
	C2MP	176	4.0	2
	C2MI	173	4.0	3
	C2MPI	177	4.0	3
Logical[2]				
XOR to ACC	ORA	100	2 3	2
	ORAP	104	2.3	2
	ORAI	101	2.3	3
	ORAPI	105	2 3	3
XOR to mem	ORM	102	2.3	2
	ORMP	106	2.3	2
	ORMI	103	2.3	3
	ORMPI	107	2.3	3
And to ACC	ANA	110	2.3	2
	ANAP	114	2.3	2
	ANAI	111	2.3	3
	ANAPI	115	2.3	3
And to mem	ANM	112	2 3	2
	ANMP	116	2 3	2
	ANMI	113	2 3	3
	ANMPI	117	2.3	3
Arithmetic				
Add to ACC	BAA	120	2 3	2
	BAAP	124	2 3	2
	BAAI	121	2.3	3
	BAAPI	125	2 3	3
Add to mem	BAM	122	2.3	2
	BAMP	126	2 3	2
	BAMI	123	2.3	3
	BAMPI	127	2.3	3
Sub from ACC	BSA	130	2 3	2
	BSAP	134	2.3	2
	BSAI	131	2.3	3
	BSAPI	135	2 3	3
Sub from mem	BSM	132	2 3	2
	BSMP	136	2 3	2
	BSMI	133	2.3	3
	BSMPI	137	2.3	3

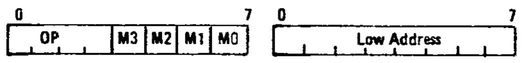
Instruction	Mnemonic	Octal	Execute Timing (MCT)	Fetch Timing (MCT)
Page Instruction				
Change Page	CHP	140	1	2
	CHPP	144	1	2
	CHPI	141	1	3
	CHPPI	145	1	3
Store Page	SPR	142	1.3	2
	SPRP	146	1 3	2
	SPRI	143	1 3	3
	SPRPI	147	1 3	3
Store Sense Switches	SSS	152	1 3	2
	SSSP	156	1 3	2
	SSSI	153	1.3	3
	SSSPI	157	1 3	3
Word Mark Instruction				
Erase WM	EWM	200	1 3	2
	EWMP	204	1.3	2
	EWMI	201	1 3	3
	EWMPI	205	1 3	3
Set WM	SWM	202	1 3	2
	SWMP	206	1 3	2
	SWMI	203	1 3	3
	SWMPI	207	1 3	3
Two Byte Jump				
Halt	HLT	230	1	2
	HLTP	234	1	2
	HLTI	231	1	3
	HLTPI	235	1	3
Set Interrupt	SIF	222	1	2
	SIFP	226	1	2
	SIFI	223	1	3
	SIFPI	227	1	3
Jump	PGJ	220	1	2
	PGJP	224	1	2
	PGJI	221	1	3
	PGJPI	225	1	3
Decimal				
Add to ACC	DAA	240	2.3	2
	DAAP	244	2.3	2
	DAAI	241	2.3	3
	DAAPI	245	2.3	3
Sub from ACC	DSA	250	2.3	2
	DSAP	254	2.3	2
	DSAI	251	2.3	3
	DSAPI	255	2.3	3

Mnemonic Suffix I = Indirect Addressing
Mnemonic Suffix P = High Order Operand Address Bits from Program Counter

(1) Jam Copy Instruction execution time is 2.0 MCT per byte
(2) Logical and Arithmetic execution time is 2.3 MCT per byte

INSTRUCTION FORMATS

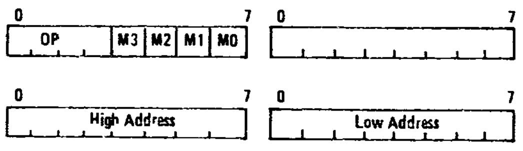
Two Byte Instruction



Modifier Bits

M0 – Indirect Address Bit
M1 – Destination Bit
M2 – Page Selection Bit
M3 – Function Bit

Four Byte Instruction



NOTE Modifier bits are used in the Two Byte Instructions only by the JWM Instruction. Byte two is used to contain the address of the location being tested for a wordmark by the JWM Instruction Otherwise, it is unused and may be used as a data byte.

EFFECTIVE ADDRESS CALCULATION

Modifier	EFAD

M0 = 0
M2 = 0
	(Page Register), Address

M0 = 0
M2 = 1
	(HPC), Address

M0 = 1
M2 = 0
	(Page Register), [(HPC), Address]

M0 = 1
M2 = 1
	(HPC), [(HPC), Address]

I/O INSTRUCTION FORMAT

PERIPHERAL CONTROL INSTRUCTION (PCI)

0							7	0							7
0	0	1	1												

In/Out Device Channel Control

I/O Code	BIT Nos.	Value	Function
In/Out	4	0	Specifies an input instruction (PCI)
	4	1	Specifies an output instruction (PCO)
Device	5-7		Device number
Channel	0-1	00	Channel A
		10	Channel C
		11	Channel G
Control	2-7		
	2-3	00	Initiates functions such as backspace, rewind, or move to top of form.
	2-3	01	Select the specified device for data transmission.
	2-3	10	Select the specified device for test
	2-3	11	Control the specified I/O device.
			To
	7	0	Enable Interrupt
	7	1	Disable Interrupt
	6	0	Transmission will not contain WM.
		1	Transmission will contain WM

PERIPHERAL TRANSFER INSTRUCTION (PTI)

0	0	0	1	M3	M2	M1	M0	Address*

Channel

M3	M2	Channel
0	0	A
0	1	N/A
1	0	C
1	1	G

Channel	M1	M0	Transfer Mode
A	0	0	Non Overlapped
A	0	1	Overlapped
G	0	0	Non Overlapped
G	0	1	Overlapped

*EFAD = (Page Register), Address

Channel	M3	M2	M1	M0	Function (TTY)
C	1	0	0	0	Keyboard Input
C	1	0	0	1	Print
C	1	0	1	0	Read Paper Tape

MODEL ASR/KSR-33
THE ASCII TELETYPE CODE (OCTAL & BINARY)

Character	Octal	Binary	Character	Octal	Binary
A	301	11000001	!	241	10100001
B	302	11000010	"	242	10100010
C	303	11000011	#	243	10100011
D	304	11000100	$	244	10100100
E	305	11000101	%	245	10100101
F	306	11000110	&	246	10100110
G	307	11000111	'	247	10100111
H	310	11001000	(	250	10101000
I	311	11001001	)	251	10101001
J	312	11001010	*	252	10101010
K	313	11001011	+	253	10101011
L	314	11001100	,	254	10101100
M	315	11001101	–	255	10101101
N	316	11001110		256	10101110
O	317	11001111	/	257	10101111
P	320	11010000		272	10111010
Q	321	11010001	,	273	10111011
R	322	11010010	<	274	10111100
S	323	11010011	=	275	10111101
T	324	11010100	>	276	10111110
U	325	11010101	?	277	10111111
V	326	11010110	@	300	11000000
W	327	11010111	[	333	11011011
X	330	11011000	\	334	11011100
Y	331	11011001	]	335	11011101
Z	332	11011010	↑	336	11011110
			←	337	11011111
0	260	10110000	Line/Feed	212	10001010
1	261	10110001	Carriage/Return	215	10001101
2	262	10110010	Space	240	10100000
3	263	10110011	Rub Out	377	11111111
4	264	10110100			
5	265	10110101			
6	266	10110110			
7	267	10110111			
8	270	10111000			
9	271	10111001			

I/O INSTRUCTIONS

Instruction	Mnemonic	Octal	Fetch Timing (MCT)	Execute Timing (MCT)
Peripheral Control	PCIX*	06X	2	1
	PCOX	07X	2	1
Non Overlapped Peripheral Transfer				
A	PTA	020	2	1/byte
G	PTG	034	2	1/byte
Overlapped Peripheral Transfer				
A	OTA	021	2	1/byte
G	OTG	035	2	1/byte
Peripheral Transfer (TTY)†				
C Non-Interrupt	KEY	030	2	10 cps
	PRT	031	2	10 cps
	RPT	032	2	10 cps
C Interrupt	KEY	030	2	1/byte
	PRI	031	2	1/byte
	RPI	032	2	1/byte

*X = Device Number
†Total execute time determined by peripheral device speed

FOUR BYTE INSTRUCTIONS

Instruction	Mnemonic	Octal	Execute Timing (MCT)	Fetch (MCT)
Jump if EQ	JEQ	042	3.0	2
Jump if $\overline{EQ}$	NEQ	043	3.0	2
Jump if GR	JGR	044	3.0	2
Jump if $\overline{GR}$	NGR	045	3.0	2
Jump if GR and EQ	JGE	046	3.0	2
Jump if $\overline{GR \text{ and } EQ}$	NGE	047	3.0	2
Jump Unconditionally	JMP	041	3.0	2
Jump on Peripheral Test	JPT	050	3.0	2
Jump to Subroutine	JSR	051	5.6	2
Jump Return Int	JRI	040	3.0	2
Jump on WM[1]	JWM	210	3.0	2
	JWMP	214	3.0	2
	JWMI	211	3.0	3
	JWMPI	215	3.0	3
Jump if MI	JMI	052	3.0	2
Jump if $\overline{MI}$	NMI	053	3.0	2
Jump if OV	JOV	054	3.0	2
Jump if $\overline{OV}$	NOV	055	3.0	2
Jump if GR or EQ	EOG	056	3.0	2
Jump if $\overline{GR \text{ or } EQ}$	ENG	057	3.0	2

1 Refer to Note of Instruction Format

JUMP ON PERIPHERAL TEST (JPT)

0	0	1	0	M3	M2	M1	M0			CL5	CL4	CL3	CL2	CL1	CL0
High Address								Low Address							

M3 = 1 Peripheral Test sent to all devices

CL5–CL0	Test
001000	I/O Device Busy
000001	Interrupt Requested
010000	Channel Busy

READER'S COMMENTS

BIT, Incorporated continuously strives to maintain and improve the quality of its publications. To help us do this effectively, your criticism is needed.

Space is provided below for your comments on this manual.

Remarks on accuracy, organization, usefulness, etc. ____________________

Errors Noted ____________________

Suggested Improvements ____________________

Other Literature Available:

☐ FORTRAN Manual
☐ FORTRAN Library Vol. I
☐ NUMERICOM (N/C)

Name ____________ Position ____________

Company ____________ Department ____________

Street ____________________

City ____________ State ____________ Zip ________

Cut Along This Line

. . . . Fold Here . . . .

. . . . .Do Not Tear · Fold Here and Staple. . . . .

No
Postage Stamp
Necessary
If Mailed in
U.S.A.

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 50, NATICK, MASS. 01760

Postage Will Be Paid By

BIT, INCORPORATED
5 STRATHMORE ROAD
NATICK, MASSACHUSETTS 01760