PRELIMINARY COPY

CIP/2200 COMPUTER REFERENCE MANUAL

7 000 0076MA

December, 1972

CIP/2200 Computer Reference Manual

Publication 7 000 0076MA

Errata Sheet

Page 16

Direct relative

The mode designator, bit 8, should be a '1' instead of a '0'.

Line 4. After "--- 2's complement" add the word 'signed'.

Indirect relative

Line 3. Insert the word 'signed' between "complement" and "dis-placement".

Page 39

Line 3. "LDX/" should read "LDX=".

Page 44

In "LRA - Logical Right A" the shifting diagram should show register A and not B.

Page 75

"MVL - Move character string left" should have an op-code of 5C and not 5A.

Page 76

"MVR - Move character string right" should have an op-code of 5D and not 5B.

Page 112

Add the words "and clear high" to the comments of the two 13XX instructions.

Table of Contents

1.0	INTRODUCTION	4
2.0	GENERAL CHARACTERISTICS	6
2.1	Basic Machine Architecture	6
2.2	Information Formats	9
2.3	Memory Addressing	15
2.4	Interrupt Structure	19
2.5	Input/Output Facilities	21
2.6	Control Stack Facility	24
3.0	MACHINE INSTRUCTIONS	27
3.1	Binary Arithmetic and Logical Instructions	27
3.2	Shift Instructions	41
3.3	Variable Word Length Instructions	47
3.4	Memory Immediate Instructions	56
3.5	Memory to Memory Instructions	60
3.6	Transfer of Control Instructions	81
3.7	Control Instructions	90
3.8	Input/Output Instructions	95
4.0	COMPUTER OPERATION	104
4.1	Front Panel	104
4.2	Basic Panel Operation	105
4.3	System Panel Operation	109
4.4	Bootstrap Loader	114
4.5	Disk IPL Option	115

APPENDIXES

A.	Instructions Listed Numerically by Opcode	117
B.	Instructions Listed Alphabetically	122
C.	Instruction Execution Times	127
D.	Power Fail/Auto Restart Programming	133
E.	DMA Channel Programming	135
F.	Firmware Extension Procedures	138
G.	Instruction Flowcharts	142
H.	Dedicated Memory Locations	147
I.	Internal Codes	149

1.0 INTRODUCTION

The CIP/2200 is a general purpose byte oriented minicomputer designed primarily for dedicated system applications. The CIP/2200 has an extensive instruction set including binary and decimal arithmetic capability, character string and bit manipulation instructions.

The CIP/2200 has an 8 bit hardware data path and memory. The CPU registers, however are 16 bits in length. The instruction set includes a complete set of 16 bit register to memory and register to register binary arithmetic instructions. In addition to the 16 bit word instructions, the CIP/2200 includes a variable length binary arithmetic capability, allowing binary arithmetic to be performed on 8, 16, 24, or 32 bit data without resorting to multiple precision software routines. A third group of instructions provides memory to memory decimal arithmetic on ANSCII format character strings up to 16 digits in length. Other memory to memory operations include character string move and compare, code conversion, and decimal editing.

Main memory consists of 8 bit/byte or 9 bit/byte core memory with a 1.1 microsecond full cycle time. Memory is expandable in 8k byte modules from a minimum of 8k bytes to a maximum memory size of 32k bytes. Memory parity protection is available as an option on systems with 9 bit/byte memory.

The CIP/2200 I/O structure consists of a microprogrammed serial I/O interface, a byte I/O facility, firmware supported Direct Memory Channel concurrent transfers, and the capability of attaching up to two independent Direct Memory Access (DMA) processors. The serial I/O interface is a microprogram for controlling serial data transfers, normally used to control a teletype or other similar terminal device. The byte I/O facility is used for transmission of 8 bit data between any one of up to 32 peripheral devices and either an accumulator register or main memory under program control. Associated with the byte I/O facility is the external priority interrupt system. The CIP/2200 has a capacity for up to 64 external priority interrupts. Byte I/O interfaces normally contain the logic for data transfer related interrupts. Non device-related interrupts may be added in groups of 8 for a total of 64 interrupts.

The Direct Memory Channel feature (DMC) allows micro-program controlled, high speed data transfer to occur concurrently with program execution. The maximum DMC transfer rate is 86,000 bytes per second for asynchronous devices and 25,000 bytes per second for synchronous devices. Requirements for faster operation (up to 910,000 bytes per second) may be filled with a hardware Direct Memory Access processor. The Direct Memory Access unit is an independent hardware controller which competes with the CPU for use of main memory time.

The CIP/2200 is a microprogrammed general purpose mini-computer based on the CIP/2000 Computer. The use of microprogramming has allowed instructions of considerable power and flexibility to be implemented at modest cost. Microprogramming offers the user the added power of features such as a bootstrap loader, an integral serial I/O facility, a high speed Direct Memory Channel, and complex instructions such as "Edit and Mark" and "Translate and Test under Mask" in a low cost system. The CIP/2200 provides special instructions to allow transfer of control to special user written application microprograms for greater flexibility in meeting specific system requirements.

.0 GENERAL CHARACTERISTICS

.1 Basic Machine Architecture

Registers

The CIP/2200 has three programmable registers, the accumulator (A), the accumulator extension (B), and the index register (X). Two other registers, the program counter (P) and the status register (S) are also of importance to the programmer.

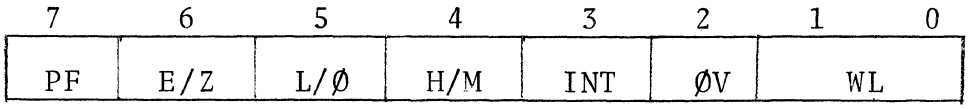
The A register is the 16 bit main accumulator register. It is an operand source for all binary arithmetic and logical instructions, and receives the result of all binary operations except store. Serial and byte mode I/O operations may transfer data to or from the low 8 bits of the A register.

The B register is also 16 bits in length and serves as an accumulator extension for variable length binary arithmetic and logical operations. Byte mode I/O operations may transfer data to or from the low 8 bits of the B register.

The index register (X) is a 16 bit register used for address modification and base relative addressing. Several specialized instructions are provided for index value modification.

The P register (program counter) contains the address of the next machine instruction to be executed. The P register contents are stored by the subroutine transfer instruction (RTJ) and are altered by transfer of control instructions (jump, skips, etc.). When an instruction modifies the P register contents, the address of the next machine instruction to be executed is determined by the contents of the P register after it has been modified.

The status register (S) is an 8 bit register containing the CIP/2200 internal status indicators. The indicators are stored as shown in the illustration below. The meaning of each indicator is described in Figure 2-1.



MNEMONIC	NAME
PF	Power Failure Indicator
E/Z	Equal/Zeros Result Indicator
L/Ø	Low/Ones Result Indicator
H/M	High/Mixed Result Indicator
INT	Interrupt System Disabled Indicator
ØV	Overflow Indicator
WL	Word Length Indicator

Figure 2-1. CIP/2200 Status Register

Power Failure (PF)

The power failure indicator reports the status of the computer's electrical power supply. The power fail/automatic restart option monitors the power supply and generates an internal interrupt whenever the input power line voltage becomes insufficient for reliable operation. The firmware detects the interrupt, sets the power failure indicator, and causes the CIP/2200 program to transfer to a user programmed power failure interrupt service routine. When normal power supply voltage is restored, the power fail/automatic restart option causes another internal interrupt. The firmware, in response to the power restart indication, resets the power failure indicator and transfers to the user written power restart routine. If the power fail/automatic restart option is not installed, the power failure indicator is set to zero by a RESET and is not changed by the hardware.

Arithmetic and Logical Indicators (E/Z, L/Ø, H/M)

The indicators represented by bits 4-6 of the status register are collectively called the arithmetic and logical indicators (ALI). These three bits are used to save information about the results of the memory to memory and memory immediate arithmetic and logical instructions. Only one indicator will be set at any given time.

The "E/Z" (Equal/Zeros result) indicator is set when the operands of a compare are equal or when the result of an arithmetic or logical operation is zero. The "L/Ø" (Low/Ones result) indicator is set when the source of a compare is lower (more negative) than the target, when the result of an arithmetic operation is negative, or when the result of a logical operation contains all ones. The "H/M" (High/Mixed result) indicator is set when the source of a compare is higher (more positive) than the target, when the result of an arithmetic operation is positive, or when the result of a logical operation contains mixed ones and zeros.

The meaning of the individual indicators depends upon the type of operation which caused them to be set. For example, an arithmetic operation which produces a negative (low) result will cause bit 5, the Low/Ones indicator, to be set. Similarly, a logical operation which causes all bits of the result to be ones will also set the Low/Ones indicator. Note that the arithmetic and logical indicators are set only by the memory to memory and memory immediate instructions and are not altered by binary arithmetic and logical instructions. The indicators may be tested with a conditional branch instruction; testing does not cause the indicators to be reset.

Interrupt System Disabled Indicator (INT)

The INT indicator reports the external interrupt system status. The INT indicator is zero when external interrupts are enabled and one when they are disabled or masked. The INT indicator is meaningless in systems which do not have the interrupt enable/disable option installed.

Overflow Indicator (ØV)

The overflow indicator is set when an arithmetic overflow is detected during a binary or decimal arithmetic operation. The overflow indicator may be tested and reset by conditional skip instructions or tested without resetting by a conditional branch instruction. Instructions are also provided to set and reset the overflow indicator without testing.

The overflow indicator is not reset by binary arithmetic operations. Thus, an overflow test over multiple binary operations may be achieved by resetting the indicator before

the sequence of instructions to be tested and testing after the last operation. If one or more of the instructions caused an overflow, the indicator will be set.

An arithmetic overflow occurs when the result of an operation is too large to be contained in the receiving field or register. For decimal operations this occurs whenever the target field is shorter than the source field, or when a carry occurs when operating on the left-most target digit. For example, when 10 is added to 995 in a three digit field, an overflow occurs. A binary overflow occurs whenever the result of the operation is too large to be contained by the receiving register. This condition is detected by comparing the carry into and out of the sign bit (left-most bit) of the register; if these carries are not the same an overflow has occurred.

Word Length Indicator (WL)

The word length indicator defines the word length currently in effect for variable length binary operations. Variable length data may be from 1-4 bytes long. Values of 00_2 to 11_2 in the two bit word length indicator correspond to variable data sizes of 1 to 4 bytes, respectively. The word length value is set programmatically with control instructions. A system reset causes the word length to be set to 1 byte (WL = 00_2).

System Save Area

The system save area is a 16 byte area of memory reserved for use by the CIP/2200 firmware. The system save area is located immediately after the external interrupt locations, from 180_{16} to $18F_{16}$.

2.2 Information Formats

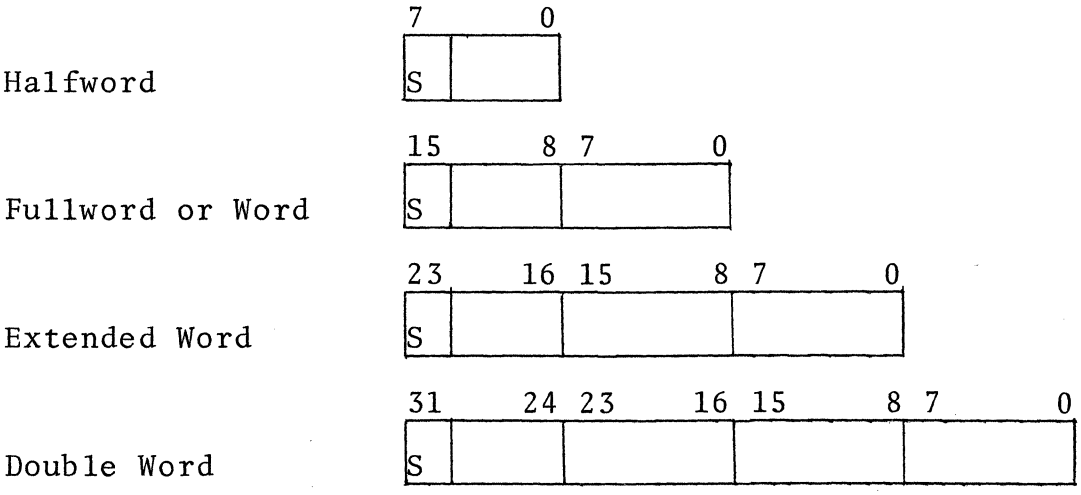
The 8 bit byte is the basic data element from which all data, addresses, and instructions are formed. The bits within the byte are numbered from 7 to 0 from left to right.

An optional 9th bit may be added to each byte in memory to provide a memory parity check. This 9th bit is used only to detect memory malfunctions and is set, reset, and checked in hardware by the memory parity check option. The CPU cannot alter or sense the parity bit.

Three basic types of information are used by the CIP/2200: data, addresses, and instructions. Data are items of information acted upon by the computer as directed by the program, and may have several forms. Addresses are information items containing the location of other items in memory. An address may be either part of an instruction or a separate item. In the latter case, the address is often referred to as a "pointer" since the address "points to" the location of some other item. Instructions are the information items which control the action of the computer.

Data Formats

Data may be stored and processed in several forms by the CIP/2200. The form chosen depends upon the nature of the data and the operations to be performed. Numerical data may be represented in either 2's complement notation binary (base 2) or sign and magnitude notation decimal (base 10) form. Logical data are represented as unsigned binary items. Alphanumeric data (text) are represented as strings of bytes, each byte being the ANSCII code for a single character.



Data Format	Assembler Coding	Size	Range
Halfword	H'n'	1 byte	-128 ≤ n ≤ 127
Fullword or Word	F'n'	2 bytes	-32768 ≤ n ≤ 32767
Extended Word	E'n'	3 bytes	-8,388,608 ≤ n ≤ 8,388,607
Double word	D'n'	4 bytes	-2,147,483,648 ≤ n ≤ 2,147,483,647

Figure 2-2 Binary Data

Binary data may be stored and processed in either 2 byte (16 bit) words or in variable length byte strings up to 4 bytes (32 bits) in length. The address of a data item is the address of the leftmost (most significant) byte. The lower order bytes occupy higher addressed bytes in memory. The various binary data formats, the range of values representable in each format, and the assembly language specification for each are shown in Figure 2-2.

Decimal numbers are represented within the CIP/2200 as strings of ANSCII decimal digit characters in memory. Decimal numbers may vary in length from 1 to 16 digits. Each digit is represented in memory as one byte containing an ANSCII zoned decimal digit. The low four bits (bits 3-0) of the byte contain the binary equivalent of the decimal digit. The high four bits (bits 7-4, the zone bits) of a decimal operand digit contain the ANSCII decimal digit zone, 1011_2, with the exception of the low order digit (highest addressed byte) of a decimal number. The zone bits of the low order digit of a number contain the sign of the decimal number. A minus sign is represented by zeros in the zone of the low order bits; a plus sign by a zone of 1011_2 (B_{16}).

The illustration in Figure 2-3 shows typical decimal numbers as they appear in memory. The leftmost byte is the addressed byte; the remainder of the number occupies successively higher locations in memory.

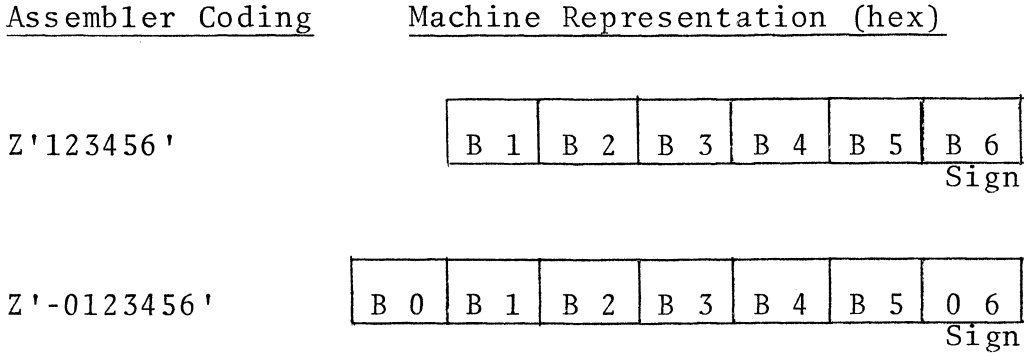
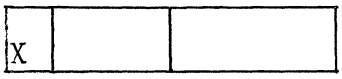


Figure 2-3 Decimal Data

Logical data is represented as either 2 byte words or variable length byte strings of from 1 to 4 bytes. The representation is similar to binary data, except that logical data is unsigned.

Alphanumeric data (text) and other character string data are represented in the CIP/2200 as variable length byte strings in memory. The individual bytes are treated as unsigned 8 bit logical items. Byte strings of from one to 256 bytes may be operated upon by memory to memory instructions. The standard internal code used by the CIP/2200 is ANSCII with bit 7 set to a 1 (See Appendix I), but other codes (e.g., EBCDIC) may be used as desired by the programmer. The CIP/2200 CPU is insensitive to the choice of code except for decimal number representation and certain characters used for editing. I/O devices and system software, however, are code sensitive and usually require ANSCII code.

Addresses are represented in several forms within the various instructions. Addresses not part of an instruction (pointers) are always represented in memory as 2 byte words as shown below. The memory address appears as a 15 bit positive integer in bits 14-0 of the address word;an address word can therefore address 32,768 different bytes. Since negative addresses are not used, the sign bit (bit 15) of an address is always zero and is not needed. The sign bit of an address word is therefore available to specify indexing. A 1 in bit 15 of an address word in memory is used to indicate that the effective address is the sum of the index register contents and the address contained in bits 14-0 of the address word. If bit 15 is a 0, the effective address is simply the value stored in bits 14-0 of the address word.



Instructions

CIP/2200 instructions vary in length from one to eight bytes depending upon the number of memory addresses required, the addressing modes specified, and the other information contained in the instruction. Variable instruction length permits efficient utilization of memory while permitting a wide range of instruction types.

In all cases the first byte of the instruction contains a code indicating the operation to be performed. The second byte of less frequently used instructions contains an extended operation code. The basic instruction formats are shown in Figure 2-4.

The control and register operate instructions do not refer to memory and consist of a one or two byte operation code only. Conditional skip instructions allow the program to test various machine conditions and alter the execution path of the program depending upon the result. Conditional skips have a one byte displacement address allowing control to be transferred to instructions within ± 128 bytes of the conditional skip. Shift instructions also use the two byte format; the first byte contains the operation and the second byte contains the count of bit positions to be shifted. Input and output (I/O) instructions require either two bytes (for transfers to or from a register) or four bytes (for transfers to or from memory). Memory immediate instructions occupy four bytes and provide various operations between a data byte in the instruction itself and a byte in memory. The memory to memory format is used for decimal arithmetic and byte string operations.

The memory reference, extended memory reference, and literal formats are used by most binary arithmetic, logical, and transfer of control instructions. Eight addressing modes allow considerable flexibility and power while conserving memory.

The CIP/2200 provides a control stack mechanism to aid the programmer in saving the machine state when entering subroutines or interrupt service routines. The machine state saved on the stack by the save instruction consists of the contents of the program counter (P), the machine status register (S), and the B, A, and X registers as shown below.

P	S	B	A	X

The status register contains the most recent arithmetic or logical indicator setting, the overflow indicator, the external interrupt enable state, the word length currently in effect, and the state of the power fail detector. The full machine state may be saved on the stack or returned to the active registers by the save and return instructions.

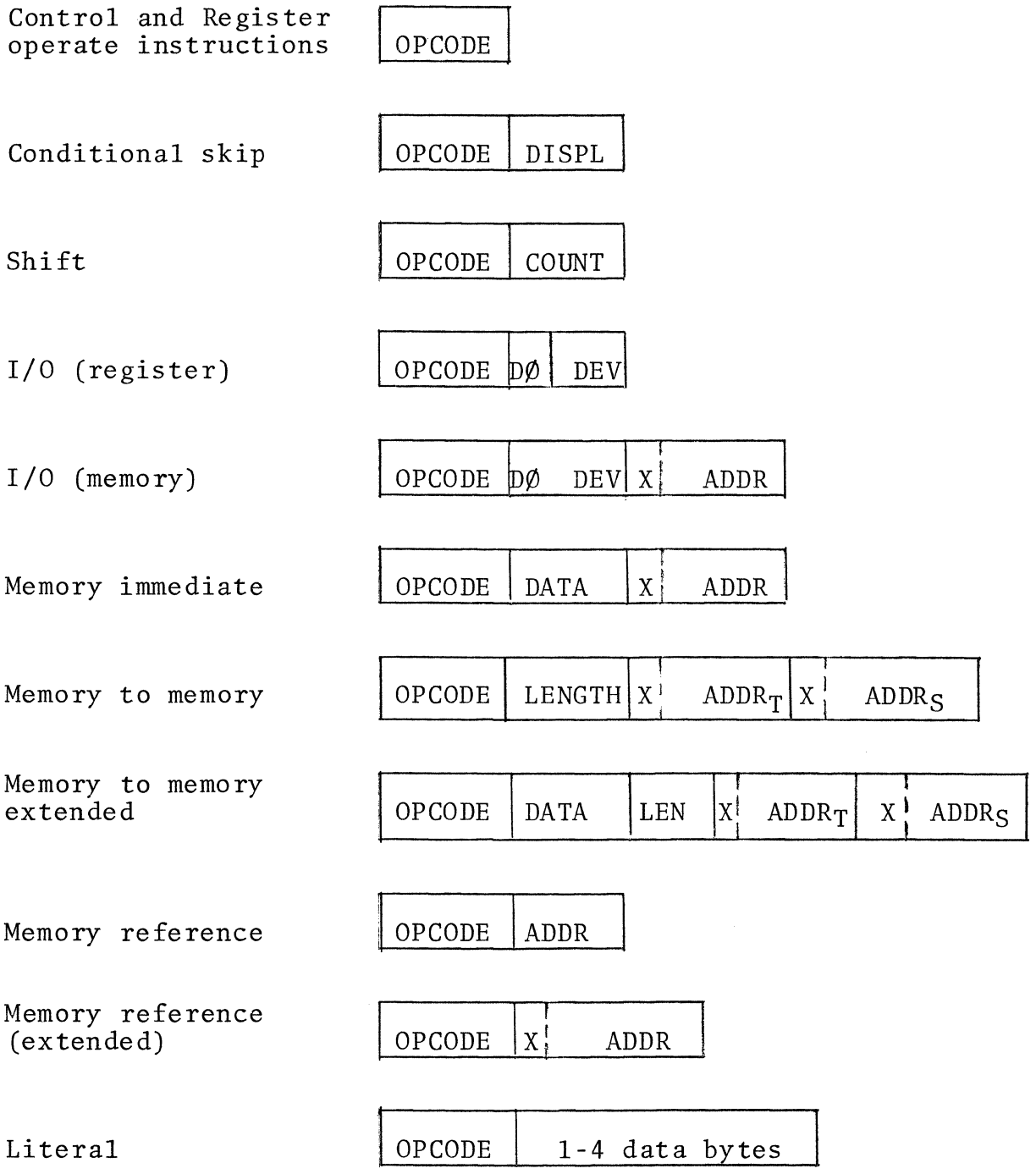
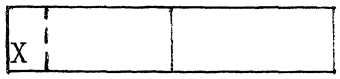


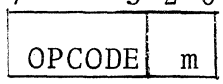
Figure 2-4 CIP/2200 Instruction Formats

2.3 Memory Addressing

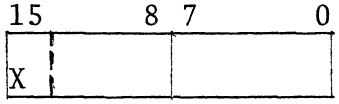
The CIP/2200 has the capability of addressing up to 32,768 bytes of memory. The basic addressing mode, extended addressing, utilizes a 16 bit address word. The low 15 bits contain the binary value of the address. Bit 15 is used to request indexing. Indexing may be specified for any memory reference using extended addressing. A single level of indirect addressing is provided for certain memory referencing instructions. When indirect addressing is used, the address word specified by the instruction may be indexed.



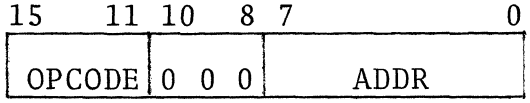
In addition to extended addressing, many memory reference type instructions have seven other addressing modes, for a total of eight modes. These additional addressing modes include various short (1 byte) address types, literal addressing, and indirect addressing. The low three bits of the operation code byte of these instructions contain a three bit addressing mode, m, as shown below. The remainder of the instruction consists of from zero to four bytes depending on the addressing mode selected. The operation specified in bits 3-7 of the instruction is performed on the data located at the effective address (EA), the memory location specified by the result of the indicated address computation.



The individual addressing modes are discussed in the following paragraphs. Modes 2 and 3, (and mode 7 for JMP and RTJ) specify indirect addressing. In all three cases the effective address is the location specified by an address word. The effective address is the contents of bits 14-0 of the address word if bit 15 is 0. A one in bit 15 causes the address specified by bits 14-0 of the address pointer to be modified by addition of the contents of the index register.

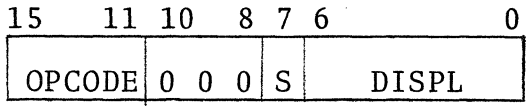


Direct page 0 (m=0) EA = ADDR



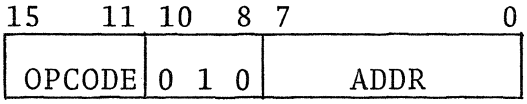
The effective address is given by the second byte of the instruction. This mode allows a short, fast means for accessing data in the first 256 locations in memory. Direct page zero addressing is commonly used for sharing data between programs. Considerable savings in program size can result from placing commonly used data (such as system parameters) on page zero.

Direct relative (m=1) EA = (P) + DISPL



The effective address is given by the sum of the program counter (P) and the 2's complement eight bit displacement contained in the second byte of the instruction. The program counter contains the address of the first byte of the next instruction when the address computation is performed. A displacement of zero, therefore, addresses the next instruction. This mode provides a savings in memory required when addressing the 256 byte area of memory around the current instruction, from 127 bytes ahead to 128 bytes behind the first byte of the next instruction. Since programs tend to refer most frequently to nearby locations, this mode allows considerable economy of storage. Note that this mode is identical to the addressing mode of the conditional skip instructions.

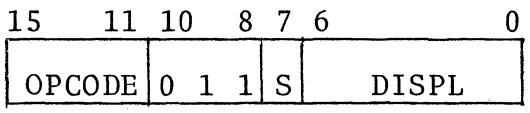
Indirect page 0 (m=2) EA = (ADDR)



The second byte of mode 2 instructions contains an 8 bit absolute address specifying a location in page zero. The two byte address word at the specified location is used to obtain the effective address as explained above for indirect addressing. Indexing may be specified by the indirect address word. This mode provides a convenient means of using shared address pointers.

Programming Note: A series of page zero pointers (a transfer vector) may be used to provide dynamically alterable linkage between subroutines.

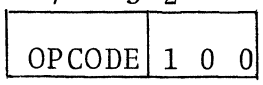
Indirect relative (m=3) EA = ((P) + DISPL)



The sum of the contents of the program counter and the 8 bit 2's complement displacement gives the address of an address word. The address word is evaluated to form the effective address. This mode provides a means of sharing address words located within a section of a program. Indirect relative addressing also provides a means of returning from a subroutine by an indirect jump.

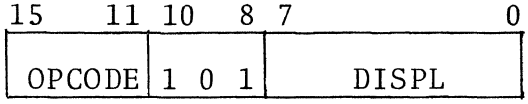
Programming Note: Indirect relative addressing can be used to save memory by sharing address words when multiple instructions located close together refer to a distant location. One reference is made to the desired datum using extended addressing (mode 6). All other references within relative range can be made indirect relative using the extended address word portion of the mode 6 instruction as a pointer.

Base addressing (m=4) EA = (X)



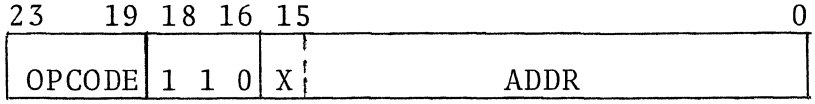
This mode provides a fast, one byte instruction format. The effective address is the contents of the index register (X). Mode 4 addressing is particularly useful for string processing or table processing where the index register is used to point to the byte in the string or entry in the table currently being processed.

Base plus displacement addressing (m=5) EA = (X) + DISPL



The effective address is given by the sum of the contents of the index register and the 8 bit unsigned displacement contained in the second byte of the instruction. This mode may be regarded as an indexed page zero address (for page zero data tables) or as a base plus displacement mode where the index register contains the address of the datum.

Extended addressing (m=6) EA = ADDR or ADDR + (X)



This mode is the basic addressing mode discussed in the beginning of this section. If indexing is not specified, the effective address is contained in the low 15 bits of the second and third bytes of the instruction. If indexing is specified, the effective address is the sum of the contents of the ADDR field and the index register. This mode allows addressing of all of a fully expanded 32,768 byte memory.

Literal addressing (m=7) EA = 2nd byte of instruction

OPCODE 1 1 1	DATA (1-4 bytes)

Literal addressing provides a means of coding data in the instruction itself, thus saving the memory otherwise required to specify the address of the data. The length of the data may vary from one to four bytes depending upon the requirements of the opcode.

Extended indirect addressing (m=7, jump and return jump only) EA = (ADDR or ADDR + (X))

OPCODE 1 1 1	X	ADDR

The address word located in the second and third bytes of the instruction is evaluated as described for mode 6. The result specifies the location of an address word, which is evaluated to form the effective address of the instruction. This mode is provided in lieu of literal addressing for the jump and return jump (subroutine transfer) instructions. Extended indirect addressing provides a means of utilizing transfer vectors (jump tables) outside of page zero.

The assembly language coding of the various addressing modes is shown below.

```
OPC     ADDR       modes 0,1 (direct page zero and direct
                             relative)
OPC*    ADDR       modes 2,3 (indirect page zero and in-
                             direct relative)
OPC-               mode 4    (base addressing)
OPC+    N          mode 5    (base plus displacement)
OPC/    ADDR(X)    mode 6    (extended addressing)
OPC=    DATA       mode 7    (literal addressing)
OPC=    ADDR(X)    mode 7    (extended indirect addressing
                             for JMP and RTJ only)
```

Figure 2-5 Address Mode Assembly Language Coding

2.4 Interrupt Structure

Basic Interrupt Action

The CIP/2200 interrupt system is responsive to two types of interrupts. Internal interrupts are generated by the computer on the occurrence of an operational fault, a console interrupt, or by the interval timer. External interrupts are generated by an I/O device or another externally supplied signal. Each interrupt is assigned an address word in main memory called an interrupt transfer location. These assignments are shown in Fig. 2-6.

The interrupt transfer location specifies the address of a software routine which will be executed whenever the associated interrupt is recognized. When an interrupt occurs, the computer responds by executing a subroutine transfer of control to the address specified by the interrupt transfer location. This mechanism for interrupting the execution of one program and starting execution of another program allows the computer to respond to external events and execute the appropriate portions of the program on a priority basis.

Location	Assignment
080-081	Console Interrupt
082-083	DMA Channel Interrupt
086-087	Interval Timer Interrupt
08A-08B	Memory Parity Error Interrupt
08C-08D	Control Stack Under/Overflow Interrup
08E-08F	Power Fail Interrupt
090-091	Power Restart Interrupt
100-101	External Interrupt 0
102-103	External Interrupt 1
.	
.	
.	
17E-17F	External Interrupt 63

Figure 2-6 Interrupt Transfer Locations

Internal Interrupts

The internal interrupts are used to report computer operation faults or the occurrence of events with a high priority. Internal interrupts are generated by the console interrupt switch or a trap instruction, the DMA channel, the interval timer, the memory parity option hardware, a control stack under/overflow, and the power fail/restart option hardware.

The console interrupt is generated by depressing the console interrupt switch on the computer front panel or by executing a trap instruction.

The DMA channel interrupt is generated by the DMA processor at the end of a DMA transfer. If a system has more than one DMA processor, both interrupts will occur through the same interrupt transfer location. A status check on one of the DMA processors must be used to determine which one interrupted. For details of operation of the DMA channel, refer to the DMA processor manual and Appendix E.

The interval timer interrupt occurs when the interval timer counter value reaches zero. The interval timer counter in memory locations 84 - 85_{16} is incremented by the interval timer (real time clock) option on the processor option board. The rate at which the timer is incremented is selected by jumpers on the processor option board. The board is supplied with the jumpers set for a 1 millisecond interval, but other intervals may be obtained by changing the jumpers as described in the Processor Option Board Manual (publication #7 000 0052MA). The EIT and DIT instructions are used to start and stop the interval timer.

The memory parity failure interrupt occurs when a parity failure (an odd number of bits in a byte) is detected by the memory parity check option hardware. The memory parity check option consists of a 9 bit memory and the parity check option on the processor option board. If the memory parity check option is not supplied, the interrupt will not be generated.

The control stack under/overflow interrupt is caused when the control stack cannot hold or supply sufficient data. This interrupt can only occur as a result of execution of a save or return instruction (SAV, RET, or RTN; see Section 2.6).

The power failure interrupt occurs when the primary power supply voltage becomes insufficient for reliable computer operation. The power restart interrupt occurs when the power supply is restored to the computer after a power failure. Certain requirements must be met by the power failure interrupt service routine if the contents of memory and the machine state are to be preserved. Refer to Appendix A for notes on the design of this program. The power fail and automatic restart interrupts are set by the power fail/automatic restart option on the processor option board, and will not occur unless that option is installed.

External Interrupts

External interrupts may be generated by I/O device controllers or other external hardware devices. The CIP/2200 has provision for a total of 64 external interrupts. The external interrupt system may be disabled (masked) to defer recognition of interrupt requests if the interrupt enable/disable option is installed.

2.5 Input/Output Facilities

The CIP/2200 provides four basic types of I/O transfers ranging from bit serial transfers at 10 characters per second to Direct Memory Access (DMA) byte transfers at 910,000 bytes per second. The serial I/O facility is a low speed interface suitable for operating a low speed terminal device. Byte I/O provides either program loop or interrupt I/O capabilities for medium speed devices. The Direct Memory Channel (concurrent I/O) facility provides a firmware managed high speed data transfer at a rate of up to 86,000 bytes per second concurrent with and transparent to normal program execution. Direct Memory Access I/O utilizes a separate hardware channel into memory to provide extremely fast I/O at the maximum memory rate of 910,000 bytes per second.

Serial I/O

The serial I/O interface provides a simple, inexpensive means for communicating with a teletype or other similar terminal device. The serial I/O interface is a microprogram which controls the transfer of bit serial data between the computer and a serial I/O device. Serial mode transfers always use the low 8 bits of the A register as the output source or input target. The data transfer rate is 110 bits per second. Program execution and detection of all interrupts is suspended during a serial I/O transfer.

Byte I/O

The CIP/2200 byte I/O facility provides for program controlled data transfers over the byte I/O bus. Each byte of data transferred to or from an I/O device controller requires the execution of an I/O instruction. The byte I/O bus is also used to transfer status information from devices to the computer and to transmit function bytes containing control information to the devices. The byte I/O facility is related to the external interrupt facility in that device controllers are able to cause an interrupt when ready for a data transfer or when an error is detected. The interrupt service routine then uses the byte I/O instructions to test the device status and to take appropriate action. This allows I/O transfers to be overlapped with background (non-interrupt) program execution.

The byte I/O facility may also be used without interrupts by testing the device status continually until the device is found to be ready to accept or send data. The upper limit of the byte I/O transfer rate is approximately 10,000 bytes per second.

The byte I/O facility offers considerable flexibility to the programmer. Data and status or control information can be transferred between external devices and the A or B register or memory. Up to 32 device controllers may be connected to the byte I/O bus.

Direct Memory Channel (DMC)

The Direct Memory Channel (concurrent I/O) facility allows automatic block data transfers between devices attached to the byte I/O bus and memory. DMC transfers are firmware controlled and occur concurrently with normal program execution.

Concurrent operation of I/O and program execution is accomplished by assigning higher priority to DMC data transfers than to instruction execution. Before each instruction is executed, a firmware check is made for pending DMC transfers. All such data transfers for DMC operations are finished before an instruction is executed. A DMC transfer may also force short breaks in the execution of long instructions.

The maximum rate at which DMC transfers can occur is determined by the time between the arrival of a data transfer request and the servicing of that request by the microprogram. The microprogram tests for pending DMC requests at discrete intervals. The longest interval possible between the occurrence of a DMC transfer request by the highest priority DMC controller and the answering data transfer is 40 microseconds. The CIP/2200 can therefore support a 25,000 byte/second transfer rate for a single synchronous device. (A synchronous device is a device which requires data transfers at a fixed rate determined by the device itself, for example, a magnetic tape drive.) When multiple devices are simultaneously transferring data, the maximum time between a request and the transfer is 40 microseconds for the highest priority device, 52 microseconds for the second highest priority device 64 microseconds for the third highest priority device, and so on. The maximum aggregate data transfer rate for the DMC channel is approximately 86,000 bytes/second. This rate can be maintained only if requests arrive at less than 12 microsecond intervals. Since service for these requests can only be guaranteed at a rate of 25,000 bytes/second, rates of more than 25,000 bytes/second can only be supported on asynchronous devices (devices with variable transfer rate, i.e., devices which can operate at rates slower than the maximum such as a tape perforator).

The location and amount of data transferred during a DMC block transfer operation is determined by a four byte DMC descriptor. A two word (4 byte) dedicated page zero location is reserved for the DMC descriptor for each I/O device. The address of this descriptor is calculated by multiplying the device number by 4. The descriptor contains two address words, a current address and an ending address. The current address word contains the address of the next byte to be transferred by the I/O operation. The ending address word contains the address of the last byte to be transferred. The current address is incremented after each byte transfer. When the current address becomes greater than the ending address, the DMC operation is terminated.

DMC operations are started by initializing the descriptor and executing the appropriate byte I/O instruction. After a DMC I/O operation is initiated by a processor instruction, byte transfers proceed automatically until the last byte of the block is transferred. An interrupt may be triggered at the end of a DMC transfer if the programmer so requests. If termination with interrupt is used, the interrupt occurs through the interrupt transfer location assigned to that I/O device. The address of the device interrupt location

is twice the device address plus 100_{16}. If termination without an interrupt is desired, the program must check the device status word to determine the state of the DMC transfer.

DMA Operations

The DMA processor may be used for high speed data transfer directly to memory at a maximum rate of 910,000 bytes per second. The DMA processor is an independent hardware device which is attached directly to the main memory address, data, and control busses. Data transfers performed by the DMA processor are completely independent from operation of the main computer except that both devices use the same memory. The DMA processor takes precedence over the CIP/2200 CPU for memory operations. Programming for DMA transfers is described in Appendix E.

.6 Control Stack Facility

The CIP/2200 control stack facility provides a method for saving and restoring the computer state information. The implementation of state switching using a control stack greatly increases programming ease and flexibility, particularly in an interrupt environment.

Computer State Information

The computer state information uniquely defines the internal state of the computer at a given point in a program. The CIP/2200 state information consists of the contents of the A, B, and X registers, the location of the next instruction to be executed (P), and all status indicators (S). This information is reloaded into the computer registers when the machine state is restored. The machine state must be saved by a subroutine or an interrupt service routine to allow the programmer to use registers, change word length, or alter other status indicators freely in the subroutine or interrupt service routine without affecting the values stored in the computer registers during the execution of other programs. State saving is essential for interrupt service routines since the interrupted program must be able to continue as before when the interrupt service routine returns.

Stack Operations

A stack is a data storage element containing a varying number of entries. Data is entered and removed from the stack in a Last In-First Out ("LIFØ") manner. LIFØ operation means that when data is removed from the stack, the information removed first is that which was entered into the stack most recently. Entry of data into a stack is called "pushing" and data removal is known as "popping".

Control Stack Use

The control stack is used to save the computer state. Each entry in the CIP/2200 control stack consists of a complete set of state information. The most recently saved machine state is on the "top" of the stack, the oldest machine state on the "bottom". In normal use each subroutine saves the machine state immediately after being called. The subroutine returns the state information by using the RET or RTN instruction to "pop" the old machine state from the stack. When multi-level subroutine calls are used, the control stack will contain more than one entry during execution of the lower level subroutines. As each subroutine terminates, one more entry is removed from the stack. When control returns to the original calling routine, the control stack will be empty. Uniform use of a save and restore convention greatly enhances the reliability of programs, especially in an interrupt environment.

Control stack operations are performed by the SAV, RET, and RTN instructions. The location of the control stack is defined by the control stack pointer which occupies a dedicated page zero location - $92 - 93_{16}$. The control stack pointer always contains the address of the first (lowest addressed) byte of the most recently stacked machine state information. The control stack occupies the 256 byte memory page indicated by the control stack pointer. The control stack pointer cannot be made to cross a page boundary without generating a control stack under/overflow interrupt.

The length of the control stack area used by a program is determined by the number of levels of subroutines and interrupts allowed. For example, a program having a four level subroutine structure (where at most four subroutines are simultaneously in use - A calls B which calls C which calls D) requires 36 bytes for its control stack (9 bytes per SAV) regardless of the total number of subroutines in the program. Additional control stack space must be reserved for interrupt processing because interrupts can, in general, occur at any time. Each interrupt requires a minimum of 9 additional bytes of reserved control stack space. Interrupt service routines may also require further control stack space for subroutines.

3.0 MACHINE INSTRUCTIONS

This section describes the operation of the CIP/2200 machine instructions. The instructions are grouped by logical type into several subsections. Details of memory addressing are given in section 2.3 and are not repeated for each instruction. The instruction descriptions include the format of the instruction as it appears in memory and the CIP/2200 assembly language coding. An asterisk to the right of the machine instruction format diagram indicates that the eight addressing modes discussed in section 2.3 apply. The assembly language coding for those cases assumes extended addressing (mode 6).

The abbreviations used in this chapter are shown below.

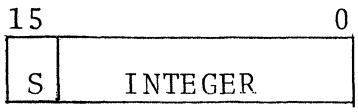
ALI	Arithmetic and logical indicators
E/Z	Equal/zero indicator
L/Ø	Low/ones indicator
H/M	High/mixed indicator
INT	Interrupt system disabled indicator
ØV	Arithmetic overflow indicator
WL	Word length indicator
PF	Power fail indicator
A	A register (accumulator)
B	B register (extended accumulator)
X	Index register
P	Program counter

3.1 Binary Arithmetic and Logical Instructions

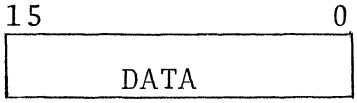
The instructions in the binary arithmetic and logical instruction group perform binary arithmetic operations on fixed length 16 bit 2's complement numbers or perform logical operations on 16 bit unsigned binary data. In general, one operand resides in a register, usually the A register, and the other is in memory. Instructions are provided for moving data between registers; transferring data between registers and memory; and for adding, subtracting, counting, masking, and inverting 16 bit data. Special instructions are provided to assist in implementing binary multiplication and division.

Binary arithmetic is provided both for general computation and for address and index computation. Since the binary arithmetic operand length is consistent with the CIP/2200 address word size, the full arithmetic and logical capabilities of the instruction set may be used for address computation.

The two fixed length binary data formats are shown below. Some of the instructions in this group also use 8 bit immediate data. When a shorter field is added to a 16 bit integer the sign of the short field is extended to provide 16 data bits. Note that when zero is complemented in 2's complement notation no change occurs.



arithmetic data



logical data

The results of all arithmetic instructions are tested for an overflow (result greater than $2^{15}-1$ or less than -2^{15}). The overflow indicator is set if an overflow occurs but it remains unchanged if no overflow occurs.

Instructions in the variable length arithmetic group (Section 3.3) and in the shift group (Section 3.2) supplement the fixed length arithmetic and logical capabilities of the CIP/2200. When mixing fixed and variable length instructions, care must be taken to ensure that the registers are used consistently.

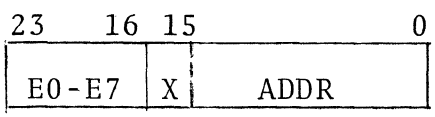
The binary arithmetic and logical instruction group is shown in Figure 3-1. The table lists the name, the assembler mnemonic, the operands in assembler notation, the indicators affected, and the machine operation code in hexadecimal notation. Those instructions allowing eight addressing modes are indicated by an asterisk following the operation code.

NAME	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Load A	LDA	ADDR(X)		E0-E7*
Load B	LDB	ADDR(X)		C0-C7*
Load X	LDX	ADDR(X)		80-87*
Store A	STA	ADDR(X)		F0-F7*
Store B	STB	ADDR(X)		C8-CF*
Store X	STX	ADDR(X)		88-8F*
Transfer A to B	TAB			2B
Transfer B to A	TBA			2F
Transfer A to X	TAX			4C
Transfer X to A	TXA			4E
Transfer B to X	TBX			4D
Transfer X to B	TXB			4F
Interchange A and B	IAB			03
Interchange A and X	IAX			35
Interchange B and X	IBX			36
Add to A	ADA	ADDR(X)	ØV	A0-A7*
Subtract from A	SBA	ADDR(X)	ØV	B0-B7*
Increment A	INA		ØV	48
Increment B	INB		ØV	49

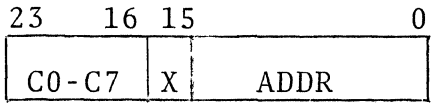
NAME	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Increment X	INX		ØV	44
Decrement A	DCA		ØV	23
Decrement B	DCB		ØV	27
Decrement X	DCX		ØV	45
Add to Index Immediate	AXI	I	ØV	5F02
Ones Complement A	ØCA			4A
Ones Complement B	ØCB			4B
Multiply Step	MST	ADDR(X)		90-97*
Divide Step	DST	ADDR(X)		98-9F*
Increment Word in Memory	IWM	ADDR(X)	ØV,ALI	70-77*
Decrement Word in Memory	DWM	ADDR(X)	ØV,ALI	78-7F*
Add to Word Immediate	AWI	I,ADDR(X)	ØV,ALI	50
And Memory with A	ANA	ADDR(X)		D0-D7*
OR B to A	ØRA			40
OR A to B	ØRB			42
Exclusive OR B to A	XRA			41
Exclusive OR A to B	XRB			43

Figure 3-1. Binary Arithmetic and Logical Instructions

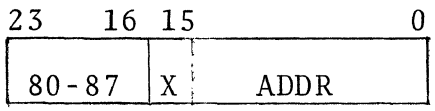
LDA - Load A



LDB - Load B



LDX - Load X

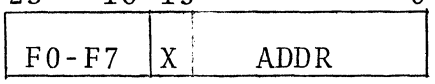


The two byte operand located at the effective address replaces the contents of the register specified by the operation code. The operand in memory is not changed.

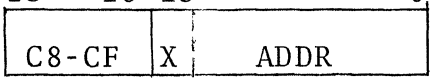
Indicators Affected: none.
Assembly Lanugage Coding:

```
LDA/    ADDR(X)
```

STA - Store A

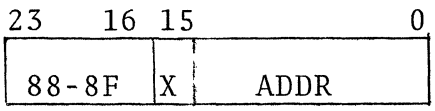


STB - Store B



The operand contained in the register specified by the operation code replaces the contents of the 16 bit word located at the effective address. The operand in the register is not changed.

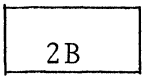
STX - Store X



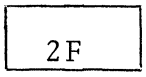
Indicators Affected: none.
Assembly Language Coding:

```
STA/    ADDR(X)
```

TAB - Transfer A to B

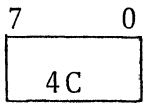


TBA - Transfer B to A

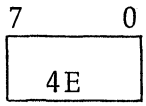


The contents of the source register (specified by the second character of the mnemonic operation code) replace the contents of the target register (specified by the last character). The source register contents are not changed.

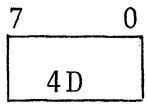
TAX - Transfer A to X



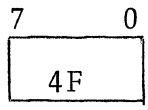
TXA - Transfer X to A



TBX - Transfer B to X



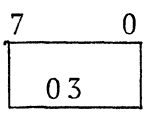
TXB - Transfer X to B



Indicators Affected: none
Assembly Language Coding:

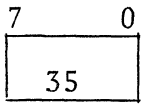
TXA

IAB - Interchange A and B

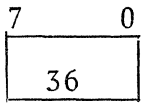


The contents of the registers specified by the opcode are interchanged.

IAX - Interchange A and X



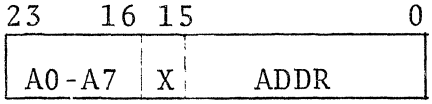
IBX - Interchange B and X



Indicators Affected: none.
Assembly Language Coding:

```
IAB
```

ADA - Add to A

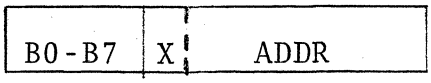


The 16 bit operand at the effective address is added to the contents of the A register; the sum replaces the contents of the A register. If the result cannot be contained in 16 bits the arithmetic overflow indicator is set. The operand in memory is not changed.

Indicators Affected: ØV
Assembly Language Coding:

```
ADA/    ADDR(X)
```

SBA - Subtract from A

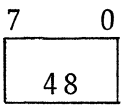


The 16 bit operand at the effective address is subtracted from the contents of the A register; the difference is placed in the A register. The operand in memory is unchanged. The overflow indicator is set if the result cannot be represented in 16 bits.

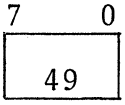
Indicators Affected: ØV
Assembly Language Coding:

```
SBA/    ADDR(X)
```

INA - Increment A

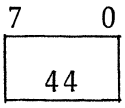


INB - Increment B



The operand in the register specified by the operation code is incremented by one; the incremented operand is placed in the register. An arithmetic overflow results if the register contains $2^{15}-1$ ($7FFF_{16}$) before the increment instruction is executed.

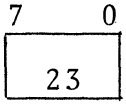
INX - Increment X



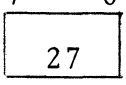
Indicators Affected: ØV
Assembly Language Coding:

```
INA
```

DCA - Decrement A

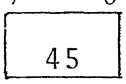


DCB - Decrement B



The operand in the register specified by the operation code is decremented by one; the decremented operand is placed in the register. An arithmetic overflow results if the register contains -2^{15} (8000_{16}) before the decrement instruction is executed.

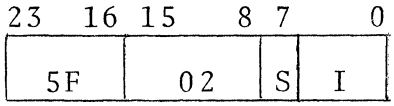
DCX - Decrement X



Indicators Affected: ØV
Assembly Language Coding:

DCX

AXI - Add to Index Immediate

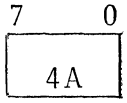


The contents of the immediate data byte located in bits 7-0 of the instruction are added to the contents of the index register; the sum is placed in the index register. The imemdiate data is treated as a signed 8 bit number. If the sum cannot be contained in the 16 bit index register the overflow indicator is set.

Indicators Affected: ØV
Assembly Language Coding:

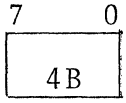
AXI I

ØCA - Ones Complement A



The operand located in the specified register is inverted. The operation is performed on a bit by bit basis. Each 0 bit in the register is replaced by a 1, each 1 by a 0.

ØCB - Ones Complement B



Indicators Affected: none.
Assembly Language Coding:

ØCA

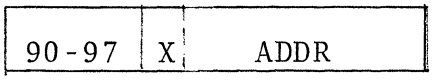
Programming Note:

The 2's complement of a number held in either the A or B register may be obtained by taking the 1's complement and incrementing the result. For example, the sequence:

```
ØCA
INA
```

will give the negative of the contents of the A register in 2's complement notation.

MST - Multiply Step



The multiply step instruction is designed to assist in the multiplication of 2's complement binary integers. Each execution of the multiply step instruction uses a single multiplier bit and generates a new partial product; the multiply step instruction must be executed once for each bit in the multiplier to perform a complete multiplication. Multiplication uses both the A and B registers. A multiply operation is performed by setting the A register to zero, loading the B register with the multiplier, and executing the Multiply Step instruction once for each bit of the multiplier.

The multiply step instruction tests the low order bit of the B register (multiplier). If the tested bit is a 1, the addressed operand (multiplicand) is added to the partial product (A); if the bit is a zero, no add is performed. The A and B registers are then shifted one place to the right, thus discarding the multiplier bit just used and positioning the partial product one place to the right.

When all of the multiplier bits have been processed the product is in the A register and the high order bits of B. Note that the number of bits in the product is equal to the number of bits in the multiplicand plus the number of bits in the multiplier.

The method of multiplication supported by this instruction requires that the multiplier be a positive number.

The multiplier must therefore be complemented if negative, and the sign of the product must be adjusted after the multiplication process.

The example below illustrates binary multiplication using the multiply step instruction. For brevity, all operands as well as the A and B registers are assumed to be 4 bits long instead of their actual length of 16 bits.

Multiplier: $5_{10} = 0101_2$
Multiplicand: $2_{10} = 0010_2$

Step	A	B	
1	0000	0101	
	+0010		
	0010	0101	
	0001	0010	(after shift)
2	0001	0010	
	0000	1001	(after shift)
3	0000	1001	
	+0010		
	0010	1001	
	0001	0100	(after shift)
4	0001	0100	
ANSWER	0000	1010	(after shift)

Indicators Affected: none
Assembly Language Coding:

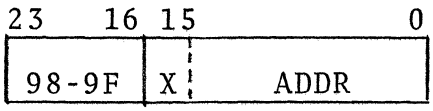
```
                        MST/   ADDR(X)
```

Programming Note:

The following code will multiply two positive 16 bit numbers.

```
   LDA=   F'0'
   LDB/   MPLIER
   LDX=   F'16'
   MST/   MCAND
   DCX                DECREMENT BIT COUNT
   NXZ    *-4         SKIP BACK IF COUNT ≠ 0
```

DST - Divide Step



The divide step instruction is designed to aid in implementing a fixed point binary divide routine. To perform a complete division the divide step instruction is executed once for each quotient bit desired. Before the division is started the dividend is loaded into the A and B registers; the divisor is located at the effective address in memory. Both the divisor and the dividend are positive binary numbers. The sign of the quotient must be determined using the rules of algebra, and the quotient complemented if necessary after division. The dividend may have a maximum length of 30 magnitude bits, and is right justified in the A-B register pair with a double sign bit in bits 15 and 14 of the A register. The quotient bits are shifted into the low order position (bit 0) of the B register as they are formed. When the divide is complete the B register contains the quotient right justified, and the A register contains the remainder.

The divide step instruction performs the following operations.

1. The A-B register is shifted left one position. A zero is shifted into the low bit of the B register.
2. The divisor is subtracted from the contents of the A register. A zero or positive result causes the low bit of the B register to be set to 1. If the result was negative, the divisor is added to the A register and the low bit of the B register is left zero.

Indicators Affected: none.
Assembly Language Coding:

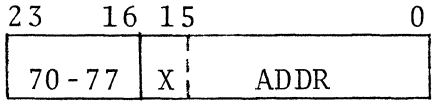
```
DST/    ADDR(X)
```

Programming Note:

The divide step instruction may be used to implement a full division as shown. The example assumes a 30 bit positive dividend and a 16 bit positive divisor.

```
R04
LDV/    DIVDND
LDX/    F'16'           LOAD QUOTIENT BIT COUNT
DST/    DIVISR
DCX                     DECREMENT BIT COUNT
NXZ     *-4
```

IWM - Increment Word in Memory

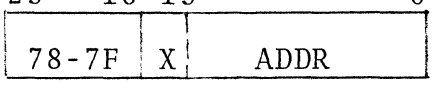


The two byte operand located at the effective address is incremented by one. If the operand contains $2^{15}-1$ ($7FFF_{16}$) before the execution of the IWM instruction an arithmetic overflow will result and the overflow indicator will be set. The arithmetic and logical indicators are set by this instruction.

Indicators Affected: ØV, ALI
Assembly Language Coding:

```
IWM/    ADDR(X)
```

DWM - Decrement Word in Memory

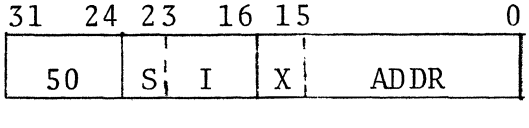


The two byte operand located at the effective address is decremented by one. If the operand contains -2^{15} (8000_{16}) before the execution of the DWM instruction an arithmetic overflow will result and the overflow indicator will be set. The arithmetic and logical indicators are set according to the sign and value of the result.

Indicators Affected: ØV, ALI
Assembly Language Coding:

```
DWM/    ADDR(X)
```

AWI - Add to Word Immediate



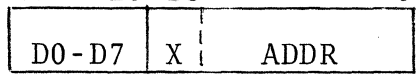
The immediate data in the second byte of the instruction is added as a signed 8-bit number to the contents of the two byte operand at the effective address; the result replaces the operand in memory. If

the result is greater than $2^{15}-1$ or less than -2^{15}, an arithmetic overflow will occur and the overflow indicator will be set. The arithmetic and logical indicators are set according to the result.

Indicators Affected: ØV, ALI
Assembly Language Coding:

```
AWI/    I,ADDR(X)
```

ANA - AND Memory with A

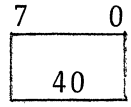


The two byte operand indicated by the effective address is ANDed with the contents of the A register. The result replaces the contents of the A register. The operation is performed on a bit by bit basis. The resulting bit is a one if both bits are one, zero otherwise.

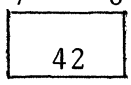
Indicators Affected: none.
Assembly Language Coding:

```
ANA/    ADDR(X)
```

ØRA - ØR B to A



ØRB - ØR A to B

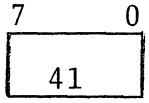


The contents of the source register (the register not named in the mnemonic operation code) is ØRed with the target register (specified in the mnemonic operation code); the result is placed in the target register. For example, ØRA replaces A with the result of the operation (A) ØR (B). The source register (B in the example is not changed. The operation is performed on a bit by bit basis. If both bits are zeros, the resulting bit is a zero; otherwise the result is a one.

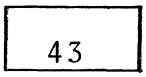
Indicators Affected: none.
Assembly Language Coding:

```
ØRA
```

XRA - Exclusive ØR B to A



XRB - Exclusive ØR A to B

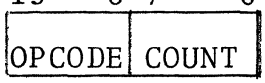


The contents of the register not indicated in the mnemonic opcode are Exclusive ØRed with the contents of the register indicated by the last character of the mnemonic. The result is placed into the named target register. For example, XRA replaces the A register contents with the result of the exclusive ØR operation. The source register (B in the example) is not changed. The operation is performed on a bit by bit basis. If both bits are alike (both zeros or both ones), the resulting bit is a zero; otherwise the resulting bit is a one.

Indicators Affected: none.
Assembly Language Coding:

```
XRA
```

3.2 Shift Instructions

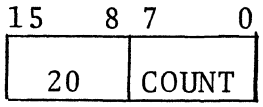


The shift group includes arithmetic and logical shifts and a rotate for the A register, the B register, and the linked A-B register pair. The shift instructions are two bytes long. The operation code determines the type of shift operation. The second byte contains an 8 bit 2's complement count which specifies the number of bit positions to be shifted. If the shift count is negative, no operation is performed. Figure 3-2 is a summary of the shift instructions. The "long" shifts operate on the linked A-B register pair. Bits shifted out of bit 0 of the A register by a long right shift are shifted into bit 15 of the B register. Similarly, bits shifted out of bit 15 of the B register by a long left shift are shifted into bit 0 of the A register. The assembly language operand "N" in Figure 3-2 specifies the shift count in all cases.

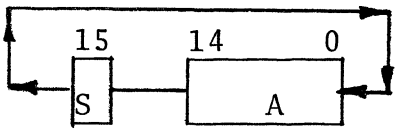
INSTRUCTION	MNEMONIC	OPERAND	INDICATORS	OPCODE
Rotate Left A	RLA	N		20
Rotate Left B	RLB	N		21
Rotate Left Long	RLL	N		22
Logical Right A	LRA	N		24
Logical Right B	LRB	N		25
Logical Right Long	LRL	N		26
Arithmetic Left A	ALA	N		28
Arithmetic Left B	ALB	N		29
Arithmetic Left Long	ALL	N		2A
Arithmetic Right A	ARA	N		2C
Arithmetic Right B	ARB	N		2D
Arithmetic Right Long	ARL	N		2E

Figure 3-2 Shift Instructions

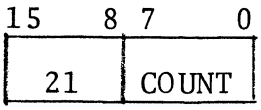
RLA - Rotate Left A

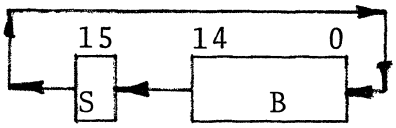


The contents of the selected register are rotated left by the number of bit positions specified in COUNT. Bits shifted out of the most significant position are shifted into the least significant position.

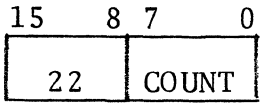


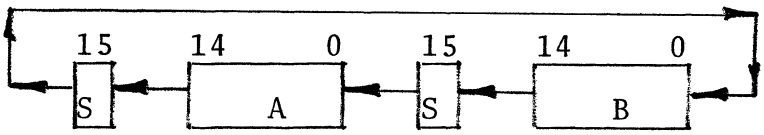
RLB - Rotate Left B





RLL - Rotate Left Long



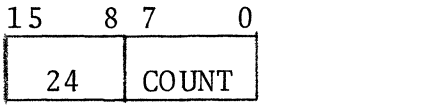


Indicators Affected: none.
Assembler Language Coding:

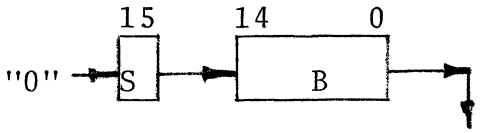
```
RLA    N
```

Programming Note: The rotate instruction is useful for moving data within a register without destroying any information.

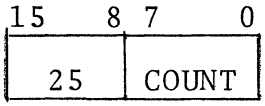
LRA - Logical Right A

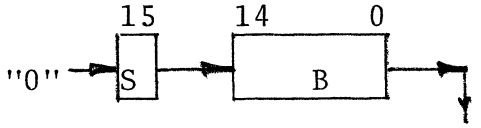


The contents of the selected register are shifted right by the number of bit positions specified in COUNT. Zeros are shifted into the most significant position. Bits shifted out of the least significant position are lost.

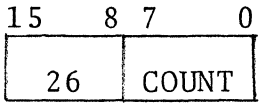


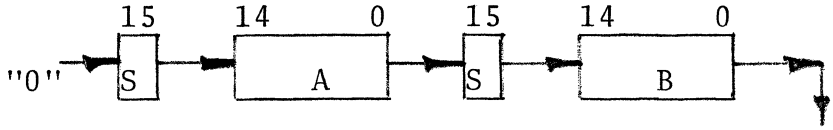
LRB - Logical Right B





LRL - Logical Right Long

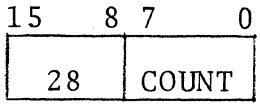


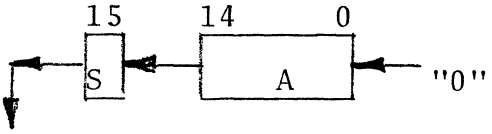


Indicators Affected: none.
Assembler Language Coding:

```
LRA    N
```

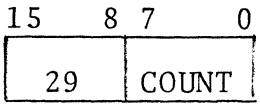
ALA - Arithmetic Left A

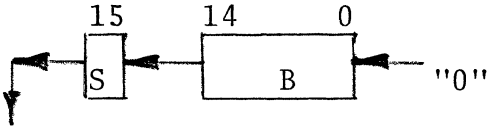




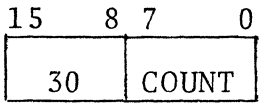
The contents of the selected register are shifted left by the number of bit positions specified in COUNT. Zeros are shifted into the least significant position and bits shifted out of the sign position are lost.

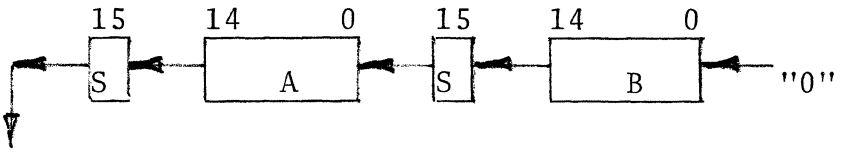
ALB - Arithmetic Left B





ALL - Arithmetic Left Long



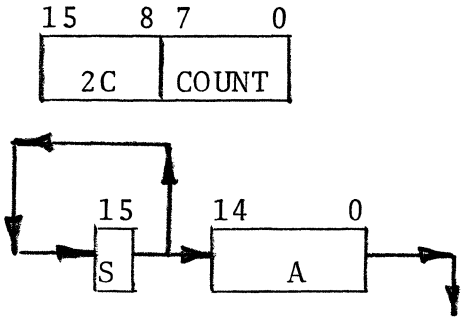


Indicators Affected: none.
Assembler Language Coding:

```
ALA   N
```

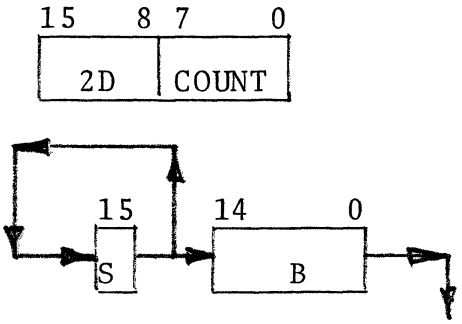
Programming Note: The arithmetic left shift can be used for multiplication by a power of two. An arithmetic left shift of n positions is equivalent to multiplication by 2^n.

ARA - Arithmetic Right A

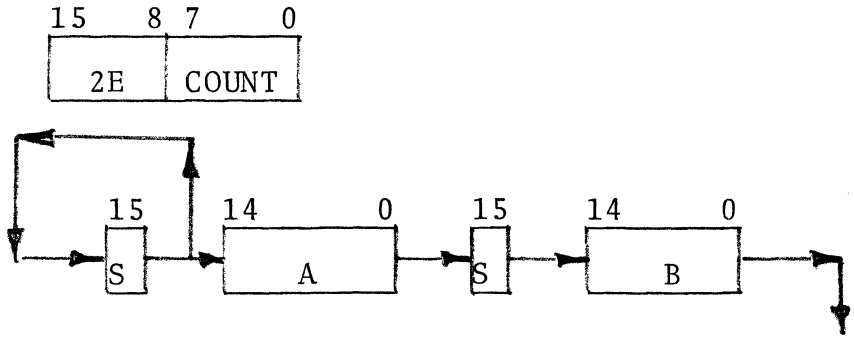


The contents of the selected register are shifted right by the number of bit positions specified in COUNT. The sign bit of the register is propagated during shifting. Bits shifted out of the least significant position are lost.

ARB - Arithmetic Right B



ARL - Arithmetic Right Long



Indicators Affected: none.
Assembler Language Coding:

```
ARA    N
```

Programming Note: The arithmetic right shift can be used for division by a power of two. An arithmetic right shift for n positions is equivalent to division by 2^n.

3.3 Variable Word Length Instructions

The length of binary data which must be processed varies from one application to another and from one instance to another within a given application. Variable word length instructions perform binary arithmetic and logical operations on one, two, three or four bytes of data. This is especially useful for character operations, single byte arithmetic, and extended precision arithmetic on 24 or 32-bit quantities.

The length of the binary data operated upon is specified by the word length indicator in the machine status register. Instructions described later in this section allow the programmer to specify the length of variable length data.

Variable word length instructions use two operands, one of which is in the A or A-B register, the other in memory. Except for the store instruction, the result of an operation replaces the contents of A or A-B. For one and two byte operations, only the A register participates, while for three and four byte operations, both the A and B registers are used. Figure 3-3 gives the exact operation for various word length values.

Word Length Value	Operands	Location of Result	Operation
1	low byte of A and 1 byte memory operand	register	The result replaces the contents of the low byte of A. Bit seven of A is propagated thru the high 8 bits of A.
1	low byte of A and 1 byte memory operand	memory (store instruction)	The contents of the low byte of A replace the contents of the specified byte in memory. A is unchanged.
2	A and two byte memory operand	register	The result replaces the contents of the A register.

Word Length Value	Operands	Location of Result	Operation
2	A and two byte memory operand	memory (store instruction)	The contents of A replace the contents of the specified two byte operand in memory. A is unchanged.
3	low byte of A and both bytes of B, and three byte memory operand	register	The result replaces the contents of the B register and the low byte of the A register. Bit seven of the A register is propagated thru the high 8 bits of the A register.
3	low byte of A and both bytes of B, and three byte memory operand	memory (store instruction)	The contents of the low byte of the A register replaces the contents of the first (lowest addressed) byte of the three byte memory operand and the contents of the B register replace the remaining two bytes. A and B are unchanged.
4	A-B register pair and four byte memory operand	register	The result replaces the contents of the A and B registers.
4	A-B register pair and four byte memory operand	memory (store instruction)	The contents of the A and B registers replace the contents of the four byte memory operand. The high byte of A in the first (lowest addressed) byte, the low byte of A in the second byte, the high byte of B in the third byte, and the low byte of B in the fourth (highest addressed) byte. A and B are unchanged.

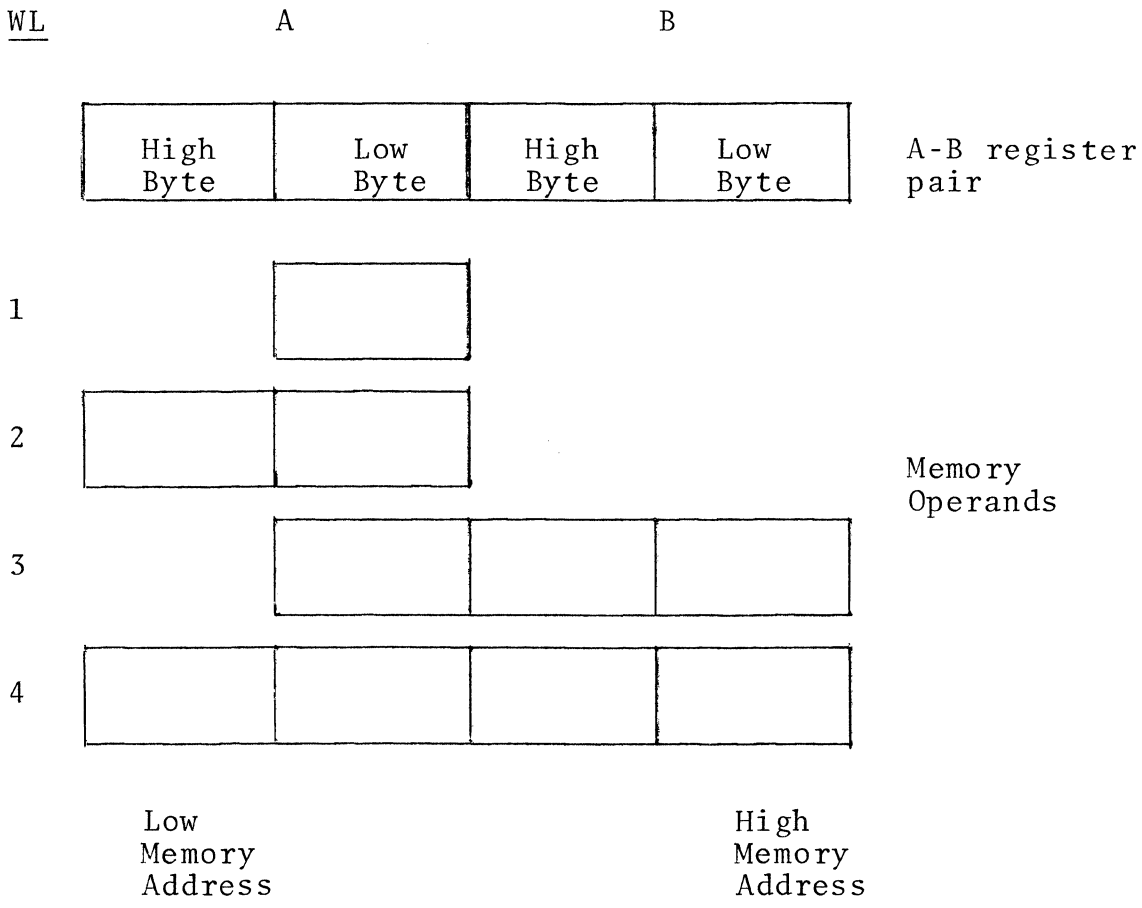


Figure 3-3 Variable Length Binary Data Formats

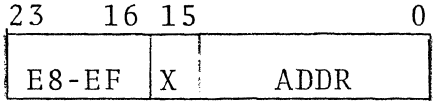
The variable length and associated instructions are shown in Figure 3-4. The mnemonic, operand coding, indicators affected, and machine language opcode are shown for each instruction. Those instructions allowing the eight memory addressing modes are indicated by an asterisk following the opcode.

INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Load Variable	LDV	ADDR(X)		E8-EF*
Store Variable	STV	ADDR(X)		F8-FF*
Add Variable	ADV	ADDR(X)	ØV	A8-AF*
Subtract Variable	SBV	ADDR(X)	ØV	B8-BF*
AND Variable	ANV	ADDR(X)		D8-DF*
Reset ØV, Set WL=1	RØ1		ØV,WL	08
Reset ØV, Set WL=2	RØ2		ØV,WL	09
Reset ØV, Set WL=3	RØ3		ØV,WL	0A
Reset ØV, Set WL=4	RØ4		ØV,WL	0B
Set ØV, Set WL=1	SØ1		ØV,WL	0C
Set ØV, Set WL=2	SØ2		ØV,WL	0D
Set ØV, Set WL=3	SØ3		ØV,WL	0E
Set ØV, Set WL=4	SØ4		ØV,WL	0F
Add Word Length to X	AWX		ØV	46
Subtract Word Length From X	SWX		ØV	47

Figure 3-4 Variable Word Length Instructions

Instruction Operation

LDV - Load Variable

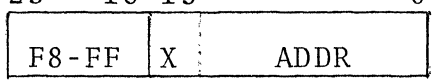


The one to four byte operand at the effective address is placed in the A or A-B register. The memory operand is unchanged.

Indicators Affected: none.
Assembly Language Coding:

LDV/ ADDR(X)

STV - Store Variable

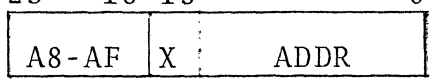


The contents of the appropriate bytes of the A or A-B register are placed in the one to four byte operand at the effective address. A and B remain unchanged.

Indicators Affected: none.
Assembly Language Coding:

STV/ ADDR(X)

ADV - Add Variable

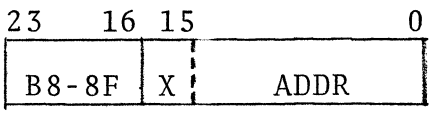


The one to four byte operand at the effective address is added to the appropriate bytes of the A or A-B register and the sum is placed in the A or A-B register. If the magnitude of the sum is greater than can be contained in A or A-B with the current word length, the overflow indicator is set. The memory operand remains unchanged.

Indicators Affected: ØV
Assembly Language Coding:

ADV/ ADDR(X)

SBV - Subtract Variable

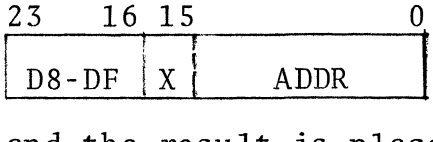


The one to four byte operand at the effective address is subtracted from the appropriate bytes of the A or A-B register and the difference is placed in the A or A-B register. If the magnitude of the difference is greater than can be contained in A or A-B with the current word length, the overflow indicator is set. The memory operand remains unchanged.

Indicators Affected: ØV
Assembly Language Coding:

```
SBV/    ADDR(X)
```

ANV - AND Variable



The one to four byte operand at the effective address is ANDed with the appropriate bytes of the A or A-B register and the result is placed in the A or A-B register. The memory operand remains unchanged. The operation is performed on a bit by bit basis. If both bits are ones, the resulting bit is a one, otherwise the resulting bit is a zero.

Indicators Affected: none.
Assembly Language Coding:

```
ANV/    ADDR(X)
```

Programming Notes:

The sign bit is propagated for a word length value of one when the result is placed in the A register. Small constants may therefore be loaded with an LDV (word length = 1) and then operated upon by instructions which use all of A as an operand.

The use of variable word length instructions with a word length value of two is logically equivalent to the use of corresponding instructions which operate on all of A (LDA, STA, ADA, SBA, ANA).

Operations with word length values of one and two affect only A while operations with word length values of three and four affect both A and B. The A register shift operations (RLA, LRA, ALA, and ARA) should be used for shifting one and two byte data; long shift operations (RLL, LRL, ALL, and ARL) should be used for three and four byte data.

Great care should be employed when using variable word length instructions with the literal addressing mode. If the word length value at execution time does not agree with the number of bytes of immediate data following the operation code byte, instruction execution will not continue at the proper location following the variable word length instruction.

The method for taking the negative of a number in the A register has already been described. To take the negative of a number in the A-B register pair, the following code can be used:

```
ØCA              1'S COMPLEMENT A
ØCB              1's COMPLEMENT B
INB              INCREMENT B
NBZ    *+3       INCREMENT A FOR
INA              A CARRY OUT OF B
```

A short zero test when using word length values of 3 and 4 may be performed by ØRing the B register contents into the A register and testing A. This method destroys the contents of A if B is nonzero and can therefore be used only for zero tests if the contents of A and B are not required after the test. A typical test might be:

```
RØ4
LDV    DATA     IS DATA
SBV/   TEST     = TEST?
ØRA
SAZ    EQUAL    YES - SKIP
```

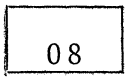
Associated Instructions

Associated with variable word length instructions are instructions which allow the setting, testing, and using of the word length value contained in bits 0 and 1 of the machine status register. All of the instructions described here are one byte in length.

Instruction Operation

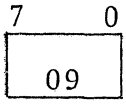
SØn, RØn - Set or Reset Overflow and Set Word Length

RØ1 - Reset ØV, Set WL=1

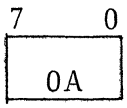


The overflow indicator is set or reset and the word length indicator is set as specified in Figure 3-5.

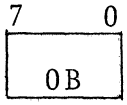
RØ2 - Reset ØV, Set WL=2



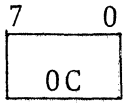
RØ3 - Reset ØV, Set WL=3



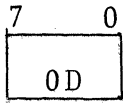
RØ4 - Reset ØV, Set WL=4



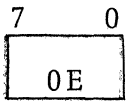
SØ1 - Set ØV, Set WL=1



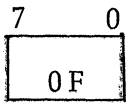
SØ2 - Set ØV, Set WL=2



SØ3 - Set ØV, Set WL=3



SØ4 - Set ØV, Set WL=4



INSTRUCTION MNEMONIC	OPERATION CODE	WORD LENGTH INDICATOR	RESULTING WORD LENGTH VALUE
RØ1	08	00_2	1
RØ2	09	01_2	2
RØ3	0A	10_2	3
RØ4	0B	11_2	4
SØ1	0C	00_2	1
SØ2	0D	01_2	2
SØ3	0E	10_2	3
SØ4	0F	11_2	4

Figure 3-5 Word Length and Overflow Operations

Indicators Affected: ØV, WL
Assembly Language Coding:

RØ1

AWX - Add Word Length to X

7 0

46

The word length value (1, 2, 3, or 4) is added to the contents of the X register and the sum is placed in X. If the sum is greater than 2^{15}-1 the overflow indicator is set.

Indicators Affected: ØV
Assembly Language Coding:

AWX

SWX - Subtract Word Length from X

7 0

47

The word length value (1, 2, 3, or 4) is subtracted from the contents of the X register and the difference is placed in X. If the difference is less than -2^{15} the overflow indicator is set.

Indicators Affected: ØV
Assembly Language Coding:

SWX

Programming Notes:

The AWX and SWX instructions are normally used for stepping X through tables one, two, three or four bytes in width. AWX may also be used to determine the current word length value by clearing X and then adding the word length with the AWX instruction. SØV and NØV conditional skip instructions can be used to test and reset the overflow indicator. The word length and overflow indicators can also be tested without being reset using the BØC instruction with the appropriate mask.

For a given program, it is desirable to establish a convention for the word length value. Any routine which uses a different word length value must restore WL to the standard value after finishing its operations at the non-standard word length.

In a subroutine, the SAV instruction can be used to save the machine state, including the WL indicator in the machine status register. The word length value is then restored when the RET or RTN instruction is used to return from the subroutine.

Depressing the reset switch on the control panel causes the machine status register, including the overflow and word length indicators, to be cleared. This results in a word length value of one. If the power fail/automatic restart option is implemented the word length is set to 1 and the overflow indicator is reset when a power restart interrupt occurs.

3.4 Memory Immediate Instructions

Memory immediate instructions provide character and bit operations on any byte of memory without requiring the use of registers. All memory immediate instructions are four bytes in length. The first byte contains the operation code, the second byte contains the immediate logical data, and the third and fourth bytes contain an indexable address word. Most memory immediate instructions set the arithmetic and logical indicators which may then be tested with the BØC instruction.

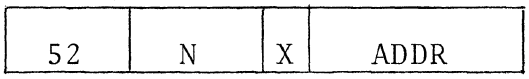
The memory immediate instructions are listed in Figure 3-6.

NAME	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Move Immediate	MVI	N,ADDR(X)		52
Compare Logical Immediate	CLI	N,ADDR(X)	ALI	53
Test Under Mask Immediate	TMI	N,ADDR(X)	ALI	54
Set Bits Under Mask Immediate	SMI	N,ADDR(X)	ALI	55
Clear Bits Under Mask Immediate	CMI	N,ADDR(X)	ALI	56
Invert Bits Under Mask Immediate	IMI	N,ADDR(X)	ALI	57

Figure 3-6 Memory Immediate Instructions

Instruction Operation

MVI - Move Immediate

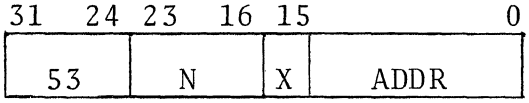


The byte of immediate data, N, replaces the byte at the effective address.

Indicators Affected: none.
Assembly Language Coding:

```
MVI   N,ADDR(X)
```

CLI - Compare Logical Immediate



The byte of immediate data, N, is compared to the byte at the effective memory location. Both operands are treated as unsigned 8-bit binary numbers. The arithmetic and logical indicators are set according to the results as follows:

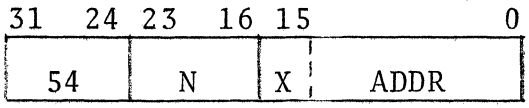
N < addressed data - L/Ø indicator set (low)
N = addressed data - E/Z indicator set (equal)
N > addressed data - H/M indicator set (high)

Only one ALI is set and the others are reset.

Indicators Affected: ALI
Assembly Language Coding:

```
CLI   N,ADDR(X)
```

TMI - Test Under Mask Immediate



The byte of immediate data, N, is used to select bits of the byte at the effective address. A one bit in N selects the corresponding bit in the addressed data. Selected bits are tested and the arithmetic and logical indicators are set as follows:

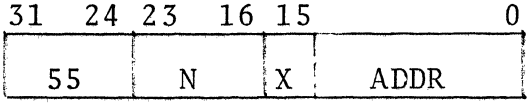
all selected bits = 1 - L/Ø indicator set (ones)
all selected bits = 0 - E/Z indicator set (zeros)
mixed 1's and 0's - H/M indicator set (mixed)

Only one ALI is set and all others are reset. The byte at the effective address remains unchanged.

Indicators Affected: ALI
Assembly Language Coding:

```
TMI   N,ADDR(X)
```

SMI - Set Bits Under Mask Immediate



The byte of immediate data, N, is used to select bits of the byte at the effective address. A one bit in N selects the corresponding bit in the addressed data. Selected bits are set to one; the remaining bits are not changed. The resulting data byte is tested and the arithmetic and logical indicators are set as follows:

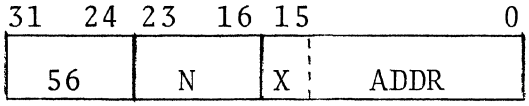
all data bits = 1 - L/Ø indicator set (ones)
all data bits = 0 - E/Z indicator set (zeros)
mixed 1's and 0's - H/M indicator set (mixed)

Only one indicator is set; all others are reset.

Indicators Affected: ALI
Assembly Language Coding:

```
SMI   N,ADDR(X)
```

CMI - Clear Bits Under Mask Immediate

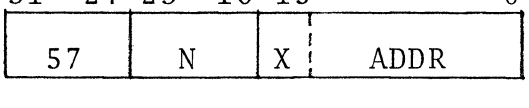


The byte of immediate data, N, is used to select bits of the byte at the effective address. A one bit in N selects the corresponding bit in the addressed data. Selected bits are set to zero; the remaining bits are unchanged. The resulting data byte is tested and the arithmetic and logical indicators are set as in the SMI instruction.

Indicators Affected: ALI
Assembly Language Coding:

```
CMI    N,ADDR(X)
```

IMI - Invert Bits Under Mask Immediate



The byte of immediate data, N, is used to select bits of the byte at the effective address. A one bit in N selects the corresponding bit in the addressed data. A selected one is set to zero; a selected zero is set to one. The unselected bits remain unchanged. The resulting data byte is tested and the arithmetic and logical indicators are set as in the SMI instruction.

Indicators Affected: ALI
Assembly Language Coding:

```
IMI    N,ADDR(X)
```

3.5 Memory to Memory Instructions

The memory to memory instruction group includes both decimal arithmetic and byte string instructions. Instructions are provided for decimal arithmetic on numbers of up to 16 digits and for byte string moves, compares, edits, and tests on strings up to 256 bytes in length. The instructions in this group operate on two operands in memory and generally do not change the register contents. The two operands are addressed by 16 bit indexable address words, the first containing the address of the target (receiving) field and the second containing the address of the source field. The arithmetic and logical indicators are set by those memory to memory instructions performing arithmetic operations or logical tests. All instructions in this group have a length byte specifying the number of bytes of data operated upon. The length byte of the decimal instructions contains separate 4 bit length fields for the target (L_T) and source (L_S) fields. All other memory to memory instructions contain a single 8 bit length specification (L).

Figure 3-7 lists the memory to memory instructions described in this section.

NAME	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Add Decimal	ADD	$ADDR_T(L_T,X)$ $ADDR_S(L_S,X)$	ALI,ØV	58
Subtract Decimal	SBD	$ADDR_T(L_T,X)$ $ADDR_S(L_S,X)$	ALI,ØV	59
Multiply Step Decimal	MSD	$ADDR_T(L_T,X)$ $ADDR_S(L_S,X)$	ØV	5A
Divide Step Decimal	DSD	$ADDR_T(L_T,X)$ $ADDR_S(L_S,X)$	ØV	5B
Edit and Mark	EDT	$ADDR_T(L,X)$ $ADDR_S(X)$	ALI	5F04
Move Character String Left	MVL	$ADDR_T(L,X)$ $ADDR_S(X)$		5C
Move Character String Right	MVR	$ADDR_T(L,X)$ $ADDR_S(X)$		5D
Compare Logical Character	CLC	$ADDR_T(L,X)$ $ADDR_S(X)$	ALI	5F05
Translate Under Mask	TRM	$N,ADDR_T(L,X)$ $ADDR_S(X)$		5F06
Translate and Test Under Mask	TTM	$N,ADDR_T(L,X)$ $ADDR_S(X)$	ALI	5F07

Figure 3-7 Memory to Memory Instructions

Decimal Arithmetic

The CIP/2200 decimal arithmetic instructions operate on zoned decimal data (see section 2.2.). The choice between zoned decimal and binary data representation for a given application should be made after considering the calculations to be performed. Zoned decimal arithmetic offers the two advantages of operating directly on ANSCII input and output format data and operating directly in decimal. For applications requiring relatively simple computation, the less efficient storage utilization and slower execution time of decimal instructions are more than offset by the time and space saved by eliminating code conversion and decimal to binary and binary to decimal steps. The use of decimal arithmetic can also result in simpler program logic by eliminating subroutine calls for number base conversion, simplifying problems of handling decimal fraction quantities, (especially in print fields) and by freeing the accumulator registers for program loop control, counting, etc.

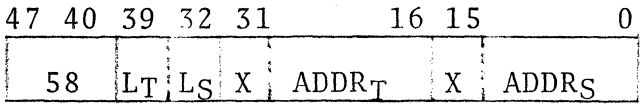
Programs requiring extensive tables of data, complex calculation sequences, or those programs operating on data obtained directly in binary form should use binary arithmetic (see section 3.1 and 3.2). The extra steps involved in converting input data to binary and output data to decimal (if necessary) are offset by the greater storage efficiency and basic instruction speed of the binary arithmetic and variable length arithmetic instructions.

The decimal instructions operate on byte strings containing from 1 to 16 zoned decimal digits. Each instruction is six bytes in length and operates on two operands in memory. The multiply step decimal and divide step decimal instructions use the A register in addition to the operands in memory. The third and fourth bytes of decimal instructions contain the address of the target (receiving) string. The target field length is in the high four bits of the second byte, with values of 0 to 15 for strings of 1 to 16 bytes, respectively. The fifth and sixth bytes contain the source field address word; the source field length is contained in the low four bits of the second byte.

The Edit and Mark instruction is used for formatting decimal strings for output. The Edit and Mark opcode occupies the first two bytes of the instruction, the third byte gives the length of the target field (allowing target strings of up to 256 bytes), and the fourth through the seventh bytes contain the pattern and source field addresses.

In processing decimal strings, no check is made for the validity of the digits. Non-decimal digit characters in decimal operands cause erroneous results and should be removed from the data fields before computation. The "translate and test under mask" instruction can be used to test for valid decimal operands.

ADD - Add Decimal

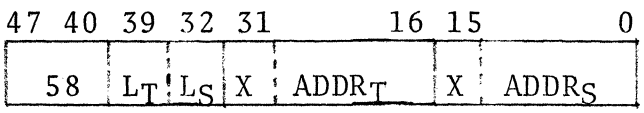


The decimal number located at the source address is added to the decimal number at the target address; the sum is stored at the target address. The source field is unchanged. if the source field length is less than the target field length, high order source zeros are supplied. The overflow indicator is set if the target field is shorter than the source field or if the addition causes a carry out of the leftmost target digit. The arithmetic and logical indicators are set according to the value of the result placed in the target field.

Indicators affected: ØV, ALI
Assembly Language Coding:

ADD $ADDR_T(L_T,X),ADDR_S(L_S,X)$
$1 \leq L_T, L_S \leq 16$

SBD - Subtract Decimal



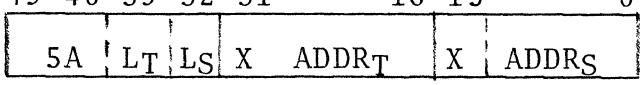
The decimal number located at the source address is subtracted from the decimal number at the target addresses. The difference is stored at the target address and the source field is unchanged.

The length of the source field must be less than or equal to the length of the target. If the source field is shorter than the target field, high order source zeros are supplied. The overflow indicator is set if the source field is longer than the target field or if the difference is too large to be contained in the target field. The arithmetic and logical indicators are set according to the value of the result placed in the target field.

Indicators Affected: ØV,ALI
Assembly Language Coding:

SBD $ADDR_T(L_T,X),ADDR_S(L_S,X)$
$1 \leq L_T, L_S \leq 16$

MSD - Multiply Step Decimal



The multiply step decimal instruction is designed to assist in implementation of a decimal multiplication routine. The decimal multiplicand located at the source address is multiplied by the digit contained in the low four bits of the A register. The resulting decimal product is added to the partial product at the target address. This sum replaces the partial product. The length of the partial product generated is always one greater than the length of the multiplicand. If the new partial product is longer than the specified partial product field, the overflow indicator is set. A complete decimal multiply may be performed by executing the MSD instruction once for each digit in the multiplier. This results in a product whose length is equal to the sum of the lengths of the multiplier and multiplicand fields.

Each execution of the multiply step decimal instruction processes one multiplier digit. The multiplication program must decrement the target address by one after each multiply step, thus shifting each partial product left one place from the previous partial product. This relative shift effectively multiplies the multiplier digit by the appropriate power of 10. The accompanying example illustrates the various steps in a decimal multiplication.

Assume that a 4 digit number at location 1000_{16} is to be multiplied by a 3 digit number located at 1100_{16} and the 7 digit product is to be generated at location 1200_{16}. The multiplication program must ensure that the partial product area (1200_{16} to 1206_{16}) is zero before the multiplication begins. The multiplier, multiplicand, and product area before multiplication are shown below.

MULTIPLICAND (Location 1000_{16})	B8	B9	B0	B2	($+8902_{10}$)
MULTIPLIER (Location 1100_{16})	B9	B1	B1		($+911_{10}$)

PRODUCT (Location 1200_{16})	B0	B0	B0	B0	B0	B0	B0

The multiplication program processes the multiplier digits from right to left. The first (rightmost) digit of the multiplier is placed in the A register, and the first multiply step is executed. The instructions required and the resulting partial product are shown below.

```
LDV/   X'1102'                  LOAD 1ST MULTIPLIER DIGIT
MSD    X'1202'(5),X'1000"(4)    FORM 1ST PARTIAL PROD.
```

PARTIAL PRODUCT (Location 1200_{16})	B0	B0	B0	B8	B9	B0	B2

Note that the partial product field length was specified as 5, one digit longer than the multiplicand field. The second and third multiplier digits are processed in a similar fashion:

```
LDV/   X'1101'                  LOAD 2ND MULTIPLIER DIGIT
MSD    X'1201'(5),X'1000'(4)    FORM 2ND PARTIAL PROD.
```

PARTIAL PRODUCT (Location 1200_{16})	B0	B0	B9	B7	B9	B2	B2

```
LDV/    X'1100'              LOAD 3RD MULTIPLIER DIGIT
MSD     X'1200'(5),X'1000'(4)     FORM FINAL PRODUCT
```

PRODUCT (Location 1200_{16})

B8	B1	B0	B9	B7	B2	B2

Note that the third partial product was 5 significant digits long (9 X 8902 = 80118).

The sign of the partial products produced is always positive. The using program can determine the proper product sign by examining the signs of the multiplier and multiplicand. If both signs are alike, the product sign is correct (+); if the multiplier and multiplicand signs differ, the product sign must be set negative.

Indicators Affected: ØV
Assembly Language Coding:

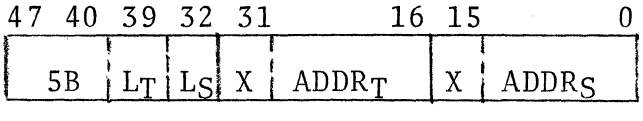
MSD $\quad ADDR_T(L_T,X),ADDR_S(L_S,X)$
$1 \leq L_T, L_S \leq 16$

Programming Note:

The following code zeros the product field (PPRODUCT). The fifteen digit multiplicand (MLTPLCND) is then multiplied by the sixteen digit multiplier (MLTPLIER). The product is accumulated in PPRODUCT.

```
            MVI     C'0',PPRODUCT
            MVL     PPRODUCT+1(30),PPRODUCT ZERO PRODUCT FIELD
            RØ1
            LDX=    F'15'                LENGTH-1 OF MULTIPLIER
LOOP        LDV/    MLTPLIER(X)           LOAD MULTIPLIER DIGIT
            MSD     PPRODUCT(16,X),MLTPLCND(15)
            DCX                                 DECREMENT COUNT
            NXN     LOOP                              NOT DONE
            LDV/    MLTPLCND+14         MULTIPLICAND SIGN - (B)
            TAB
            LDV/    MLTPLIER+15           MULTIPLIER SIGN - (A)
            XRA                              COMPARE THE SIGNS
            ANV=    X'F0'               KEEP ONLY THE ZONE BITS
            SAZ     *+6                  SKIP IF SIGNS ARE SAME
            CMI     X'F0',PPROD+30         SET PRODUCT NEGATIVE
ENDMLT      NØP                          END OF MULTIPLICATION
PPRODUCT    DS      31
```

DSD - Divide Step Decimal



This instruction is designed to assist in the implementation of a decimal division routine. The partial dividend located at the target address is divided by the decimal divisor located at the source address. The single decimal quotient digit developed is placed in the low 4 bits of A and the rest of A is set to zero. The decimal remainder resulting from the divison replaces the partial dividend. The number of resulting remainder digits is equal to the number of divisor digits. A complete decimal division may be performed by executing the DSD instruction once for each desired quotient digit. The length of the resulting remainder is equal to the number of digits in the dividend minus the number of quotient digits generated.

In order to perform a valid division the first partial dividend and the divisor must be adjusted so that the execution of the first DSD gives a quotient digit in the range of 0-9 and a remainder less than the divisor. The execution of DSD with invalid scaling or division by zero causes the overflow indicator to be set.

Indicators Affected: ØV
Assembly Language Coding:

DSD $ADDR_T(L_T,X),ADDR_S(L_S,X)$
$1 \leq L_T, L_S \leq 16$

Programming Notes:

To insure that a valid divide can be performed, it is sufficient to insure that (1) there are more significant dividend digits than significant divisor digits and, (2) if "n" is the number of significant divisor digits the first n+1 significant digits of the dividend are less than ten times the divisor. If these rules are followed in setting up for the first DSD instruction in performing a complete divison, scaling for all subsequent steps will be valid.

In the following example, proper scaling of the operands is assumed. A four digit number is divided into an eight digit number. The result is a four digit quotient in the left four digits of the dividend field and a four digit remainder in the right four digits of the dividend field.

```
        RØ1
        LDB=    C'0'                        ANSCII ZERO
        LDX     F'0'
LOOP    DSD     DIVIDEND(5,X),DIVISOR(4)
        ØRA                 SET ZONE IN QUOTIENT DIGIT
        STV/    DIVIDEND(X)
        INX
        TXA
        SBV=    H'4'        FOUR QUOTIENT DIGITS YET?
        NAZ     LØØP                            NOT DONE
```

EDT - Edit and Mark

56 48	47 40	39 32	31	16	15	0
5F	04	L	X	$ADDR_T$	X	$ADDR_S$

The Edit and Mark instruction is used for formatting decimal strings for output. Typical uses of this instruction include the removal of leading zeros, placement of punctuation and special symbols such as "$", "*", ",", or "." in a decimal field; and the combination of numeric information with text. Editing is done by moving digits from the decimal source string into an output or "pattern" string, replacing the pattern string with an edited result. Since the pattern is always altered by the edit process, normal practice is to copy the pattern string to a work area and perform the edit there.

The fourth and fifth bytes of the Edit and Mark instruction contain the pattern field address. The third byte of the instruction contains the pattern string length with values of 0-255 for pattern strings of 1-256 bytes.

The address of the zoned decimal digit string to be edited is contained in the sixth and seventh bytes of the instruction. The decimal source string is processed from left to right (low to high address) until the pattern string has been exhausted, as described below.

The format of the decimal numbers from the source string after editing is specified by control characters contained in the pattern string before the editing operation begins. The edit instruction scans both the pattern string and the decimal string from left to right, replacing certain pattern string characters with either digits from the decimal string or with a fill character. The character which appears at each location in the resulting edited string is determined by three things: the pattern character before editing, the next available decimal digit, and the state of an internal indicator called the significance switch.

Leading zeros and text characters in the pattern string are replaced by the fill character if the significance switch is off.

Pattern Characters

The pattern string consists of the fill character followed by control characters and textual data arranged to specify the format of the edited output string. Control characters are replaced with either a digit from the decimal string or the fill character; text characters are either left unchanged in the output string or replaced with the fill character. Three control characters are recognized: the digit select character, the significance start character, and the field separator character.

The digit select character (20_{16}) indicates that the location it occupies in the pattern string may be filled with the next decimal digit from the source string. The digit select character is replaced with either the selected decimal digit or the fill character. If the significance switch is off and the selected digit is non-zero, the decimal digit replaces the digit select character, and the significance switch is turned on to indicate that significant digits have been encountered. If the source digit is zero and the significance switch is off, the significance switch remains off and the digit select control character is replaced by the fill character.

The significance start character (21_{16}) is used to indicate that the next decimal digit processed is to be the first significant digit even if it is zero. The significance start character is equivalent to a digit select character except that it always turns the significance switch on. The choice of the character which replaces the significance start character is based on the state of the significance switch before the significance start character is processed, and the choice is made exactly as with the digit select character.

The table shown in Figure 3-8 describes the effect of the digit select and the significance start control characters.

Significance switch	OFF	OFF	ON	ON
Source digit	=0	≠0	=0	≠0
Pattern character replaced by	fill char	decimal digit	decimal digit	decimal digit
Resulting significance switch if pattern character is a digit select	Off	On	On	On
Resulting significance switch if pattern character is a significance start	On	On	On	On

Figure 3-8 Edit and Mark Significance Switch Meaning

The function of the field start character (22_{16}) is to create a new field within the edited string by turning the significance switch off. The field start character is always replaced with the fill character.

Text Data

All characters other than control characters appearing in the pattern string will be left in the string if the significance switch is on. If the significance switch is not on, the text characters will be replaced with the fill character.

Fill Character

The first character in the pattern string is used as the fill character. The fill character replaces all pattern characters encountered before the significance switch is turned on. The fill character is not altered during the edit operation and will always be present in the edited string.

Significance Switch

The state of the significance switch determines whether or not zeros from the source string will be treated as significant. When the significance switch is on, digit select and significance start characters in the pattern are always replaced by digits from the source string and text characters in the pattern string are not changed. The significance switch is internal to the Edit and Mark instruction and is always off at the start of the editing operation. During editing, the state of the significance switch is altered by control characters in the pattern string; the "significance start" character turns the significance switch on and the "field start" character turns it off. The significance switch is always turned on by the processing of a non-zero decimal digit in the source string.

Marking

The edit instruction "marks" the character position in the last edited field which precedes the first significant digit in the output (edited) string. Marking consists of loading the index register with the address of the character before the first significant digit. This feature may be used to insert the currency symbol or a minus sign to the left of a decimal string.

Indicators

The edit instruction sets the arithmetic indicators according to the sign of the last field edited. Only the H/P (positive) and L/N (negative) indicators may be set; the E/Z (zero) indicator is always reset. The H/P indicator is set if the last edited field is zero or positive and the L/N indicator is set if the last edited field is negative.

Examples

The following examples show some typical uses of the edit and mark instruction. Control characters in the pattern string are represented by the symbols shown below:

digit select (20_{16}) = ds
significance start (21_{16}) = ss
field start (22_{16}) = fs

Text characters are written in quotation marks for these examples. Text characters are stored in memory in ANSCII representation.

The first example illustrates the use of the edit instruction to suppress leading zeros. The fill character specified in the pattern string is a blank, "Ø". The significance switch is off at the start of the edit. The first pattern character past the fill character is a digit select character. Since the first decimal digit is a zero and the significance switch is off, the digit select character is replaced with the fill character, a blank. Processing of the pattern string continues, with the fill character replacing the pattern characters until the 6 is found in the decimal string. When the 6 is encountered, it is recognized as a non-zero decimal digit and replaces the fourth digit select character in the pattern string. The significance switch is turned on to indicate that a non-zero digit has been processed. The next pattern character is also a digit select and the next decimal digit is a zero. Since the significance switch is now on, the 0 is stored in the pattern string, replacing the digit select character. Finally, the last pattern character is replaced with a 2. The net result is that only significant digits have been retained and leading zeros have been replaced with blanks. The H/P indicator will be set since the decimal number edited was positive.

Example 1

Decimal String

0	0	0	6	0	2

Pattern String

"Ø"	ds	ds	ds	ds	ds	ds

Edited Result

"Ø"	"Ø"	"Ø"	"Ø"	6	0	2

Example 2 demonstrates the use of the significance start character and illustrates the insertion of a text character into the edited string. The first two pattern characters are replaced with the fill character as in example 1. When the significance start character is detected, the corresponding decimal digit is found to be 0 and the fill character is again used. After the fill character is inserted in place of the significance start character, the significance switch is turned on. A digit select character is encountered next and the corresponding decimal digit is a 0. Because the significance switch is on, the 0 rather than the fill character will be placed in the pattern string.

The next pattern character found is a text character. The significance switch is on, so the text character remains in the pattern string. Note that the processing of text characters does not affect scanning within the decimal string. The last two digit select characters are replaced with the last two decimal digits, 2 and 7.

Example 2

Decimal String

0	0	0	0	2	7

Pattern String

"Ø"	ds	ds	ss	ds	"."	ds	ds

Edited Result

"Ø"	"Ø"	"Ø"	"Ø"	0	"."	2	7

The third example shows how the state of the significance switch affects the appearance of text data in the final string. In case A, the significance switch is turned on by the digit 1 and the comma remains in the edited result. In case B, the singificance switch is off when the comma is processed and the comma is replaced with the fill character. Thus, the punctuation is inserted only when it is needed.

Example 3

Decimal Strings

(A)	1	6	4	7	2	7
(B)	0	5	2	4	7	3

Pattern String

"ϕ"	ds	","	ds	ss	ds	"."	ds	ds

Edited Results

(A)	"ϕ"	1	","	6	4	7	"."	2	7
(B)	"ϕ"	"ϕ"	"ϕ"	5	2	4	"."	7	3

Example 4 illustrates the effect of a pattern string containing the field start character. The field start character is replaced with the fill character. The significance switch is turned off by the field start character, causing the 0's to be replaced by blanks in the second field.

Example 4

Decimal String

0	1	2	3	0	2

Pattern String

"ϕ"	ss	ds	"."	ds	ds	"ϕ"	"ϕ"	"Q"	"T"	"Y"	"ϕ"	fs	ss	ds

Edited Result

"ϕ"	"ϕ"	1	"."	2	3	"ϕ"	"ϕ"	"Q"	"T"	"Y"	"ϕ"	"ϕ"	"ϕ"	2

Indicators Affected: ALI

Assembly Language Coding:

EDT $ADDR_T(L,X),ADDR_S(X)$

$1 \leq L \leq 256$

Programming Note

The indicator setting and marking features of the Edit and Mark instruction can be used to place a minus sign before a negative number. The code sequence shown will remove the leading zeros from the 5 digit decimal number at "NUMBER" and place a "-" to the left of the most significant digit if the number is negative.

```
         MVL    WORK,PATTERN       MOVE PATTERN TO WORK AREA
         EDT    WORK,NUMBER        DO THE EDIT
         BØC    NØTLØW,*+8         SKIP UNLESS RESULT NEGATIVE
         MVI    C'-',0(X)          NEGATIVE RESULT - INSERT "-"
          :
          .
NUMBER   DS     5
WORK     DS     6
PATTRN   DC     C' '               FILL CHARACTER
         DC     X'202020'          3 DIGIT SELECT CHARACTERS
         DC     X'2120'            SIGNIFICANT START, DIGIT SELECT
NOTLOW   EQU    X'50'
```

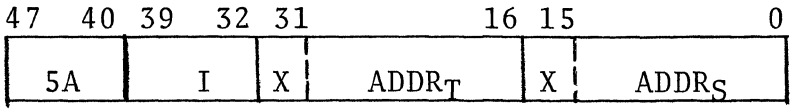
Character String Move and Compare Instructions

The string move and compare instructions provide basic operations on strings from 1 to 256 bytes in length. The strings may overlap in any way. The move instructions are six bytes in length and have one opcode byte. The compare instruction is seven bytes in length with a two-byte opcode. The first byte after the opcode gives the length of the strings to be operated upon. The number in the length byte can range from 0 to 255 in value, corresponding to string length of 1 to 256 bytes. The next two bytes contain the address of the target string, which is the receiving string for move instructions. The last two bytes contain the address of the source string. The operation of these instructions is illustrated by flowcharts in Appendix G.

In addition to the primary uses of the move instructions for transferring data from one location to another, moves are also useful for shifting decimal strings left or right and for propagating characters through fields.

Instruction Operation

MVL - Move Character String Left



The character string start- at the source address is moved to successive memory locations at the target address. Characters are moved one at a time, beginning with the leftmost byte (the byte at the lowest address) and continuing through successively higher locations until the number of bytes specified by the length byte of the instruction have been moved.

The source and target strings may overlap in any way. If the target address is lower than the source address (that is, the target is to the left of the source) and the strings overlap, the MVL instruction shifts the source string left to the target string. If the target address is higher than the source address (that is, the target is to the right of the source) and the strings overlap, the MVL instruction replaces the target string with repeated copies of that portion of the source string not overlapped by the target string.

Indicators Affected: none
Assembly Language Coding:

MVL $ADDR_T(L,X),ADDR_S(X)$
$1 \leq L \leq 256$

Programming Note:

The MVL instruction can be used to perform a decimal shift left (thus multiplying a decimal value by powers of 10) by using overlapping source and target fields with the target to the left of the source. The sign is also shifted left and must be restored to the least significant digit after shifting. The move instruction does not fill the vacated digit positions with zeros. Zero fill may be achieved by having a zero placed to the right of the field to be shifted as illustrated. The instruction shown below will multiply the contents of DEC by 10.

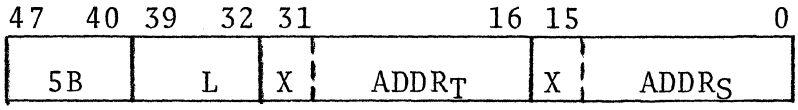
```
     MVL   DEC(10),DEC+1
           .
           .
           .
DEC  DS    10        DECIMAL FIELD TO BE SHIFTED
     DC    Z'0'      ZERO FOR FILLING VACATED PLACED
```

MVL can also be used to propagate a character through a field from left to right. The instructions

```
MVI   C' ',X'150'
MVL   X'151'(79),X'150'
```

will blank the 80 byte area beginning at location 150_{16}.

MVR - Move Character String Right



The character string starting at the source address is moved to successive memory locations at the target address. Characters are moved one at a time, beginning with the rightmost byte (the byte at the highest address) and continuing through successively lower locations until the number of bytes specified by the second byte of the instruction has been moved.

The source and target strings may overlap in any way. If the target address is higher than the source address (that is, the target is to the right of the source) and the strings overlap, the MVR instruction shifts the source string right to the target string. If the target address is lower than the source address (that is, if the target is to the left of the source) and the strings overlap, the MVR instruction replaces the target string with repeated copies of that portion of the source string not overlapped by the target string.

Indicators Affected: none
Assembly Language Coding:

MVR $ADDR_T(L,X),ADDR_S(X)$
$1 \leq L \leq 256$

Programming Note:

The MVR instruction can be used to perform a decimal shift right (thus dividing a decimal value by powers of 10) by using overlapping source and target fields with the target to the right of the source. The sign is also shifted right and must be restored to the least significant digit after shifting. Zero fill of high order digits may be insured by providing a leading zero character. The instruction below will divide the contents of DEC by 10.

```
      MVR   DEC(10),DEC-1
       :
       :
      DC    Z'0'
DEC   DS    10
```

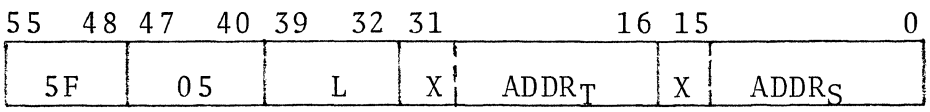
MVR can also be used to propagate a character through a field from right to left. The instructions

```
      MVI   C'.',TABLELINE+39
      MVR   TABLELINE(39),TABLELINE+1
```

will propagate dots through the 40-byte string TABLELINE from right to left.

If the source and target field do not overlap, MVR and MVL are identical in effect. In the non-overlap case, the choice between MVL and MVR should be made on the basis of speed of execution. MVL is faster for for strings of 40 or fewer characters; MVR is faster for strings of 42 or more characters.

CLC - Compare Logical Character String



The character string at the source address is compared to the character string at the target address. The strings are compared one byte at a time from left to right. The comparison treats the data bytes as unsigned binary numbers. The compare operation continues until the end of the target string is reached or the strings are determined to be not equal. The arithmetic and logical indicators are set as shown:

source > target - H/P indicator
source = target - E/Z indicator
source < target - L/Ø indicator

Only one ALI is set; the others are reset.

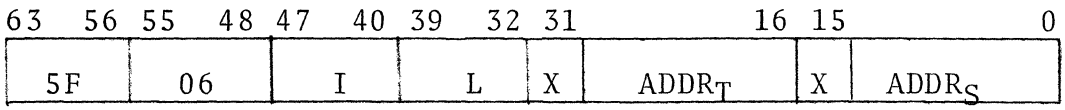
Indicators Affected: ALI
Assembly Language Coding:

CLC ADDR$_T$(L,X),ADDR$_S$(X)
$1 \leq L \leq 256$

Translate Instructions

The translate instructions provide a means for converting one character encoding scheme to another and for testing for certain values within a character string. The two instructions in this class are 8 bytes in length. The first two bytes contain the operation code. The third byte contains a mask which is used to limit the required size of the translation table. The fourth byte contains a value of 0 to 255 giving the length of the string to be translated, corresponding to string lengths of 1 to 256 bytes. The next two bytes contain the address of the target, the string to be translated. The last two bytes contain the address of the source table to be used in the translation process. The operation of these instructions is illustrated by flowcharts in Appendix G.

TRM - Translate Under Mask



The translate instruction replaces characters in the target string with characters from a table located at the source address. The primary use of Translate under Mask is code conversion between arbitrary 8 bit or less codes. Characters from the target string are fetched, one at a time. The mask byte of the instruction is used to select significant bits from the target bytes. The target byte bits corresponding to "1" bits in the mask are retained while those target byte bits corresponding to "0" bits in the mask are discarded. The selected bits are grouped, right justified, to form a displacement which is added to the source address. The character located at the resulting address replaces the original character from the target string. This operation continues until all characters in the target string have been translated.

The purpose of the mask byte is to reduce the size of the translate table required. As an example, translation of BCD data to ANSCII decimal representation requires only a 10 digit table, regardless of the position of the BCD digit within the byte. A mask byte of $F0_{16}$ and a target byte of $3F_{16}$ would result in the target byte being replaced by the source table entry located at (source address +3).

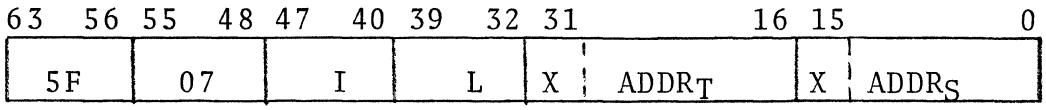
Indicators Affected: none
Assembly Language Coding:

TRM I,$ADDR_T$(L,X),$ADDR_S$(X)
$1 \leq L \leq 256$

Programming Note:

The TRM instruction can be used to translate a string of EBCDIC characters to internal ANSCII representation, or a string of ANSCII characters in internal representation to even parity representation.

TTM - Translate and Test Under Mask



The Translate and Test Under Mask instruction provides a fast means of scanning a target string for the first occurrence of one or more characters as specified by the source table. Characters from the target string are fetched, one at a time. The mask byte of the instruction is used to select significant bits from the target bytes. The target byte bits corresponding to "1" bits in the mask are retained while the target byte bits corresponding to "0" mask bits are discarded. The selected target byte bits are grouped and right justified to form a displacement which is added to the source address. If the character located at the resulting address is zero, no action is taken; the next character from the target string is fetched and the operation continues. If the byte at the resulting source table address is non-zero, it is placed in the low byte of the A register with bit 7 propagated through the high 8 bits of A. The X register is loaded with the address of the target string character being tested, and the translation is terminated. At the conclusion of the execution of the TTM instruction the arithmetic and logical indicators are set as follows:

1. If the entire target string was translated and no non-zero entries were found in the translation table, the E/Z indicator is set.
2. If the entire target string was translated and the last character in the target string character selected a non-zero entry from the source translation table, the H/M indicator is set.

3. If a non-zero entry from the source translation table is selected by some character other than the last character in the target string, the L/Ø indicator is set.

Only one ALI is set and the others are reset.

Indicators Affected: ALI
Assembly Language Coding:

```
TTM    I,ADDR_T(L,X),ADDR_S(X)
       1_L_256
```

Programming Note:

The TTM instruction can be used to test the validity of a string of decimal digits. The following instruction and table could be used to test the validity of the low 4 bits of a decimal number located at "DECIMAL".

```
       TTM   X'0F',DECIMAL,TEST
        :
        .
TEST   DC    H'0'     0
       DC    H'0'     1
       DC    H'0'     2
       DC    H'0'     3
       DC    H'0'     4
       DC    H'0'     5
       DC    H'0'     6
       DC    H'0'     7
       DC    H'0'     8
       DC    H'0'     9
       DC    H'10'    10
       DC    H'11'    11
       DC    H'12'    12
       DC    H'13'    13
       DC    H'14'    14
       DC    H'15'    15
```

3.6 Transfer of Control Instructions

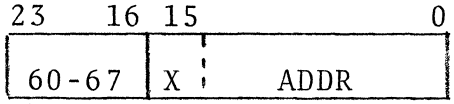
Transfer of control instructions change the sequence of instruction execution. There are both conditional and unconditional methods for transfer of control provided by the CIP/2200. The conditional transfer of control instructions provide a means of altering the order of instruction execution depending on register conditions or the status indicators. Unconditional transfer of control instructions always cause the specified transfer.

The CIP/2200 transfer of control instructions are shown in Figure 3-8. Those instructions which allow the eight addressing modes are indicated by an asterisk following the operation code.

INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Jump	JMP	ADDR(X)		60-67*
Return Jump	RTJ	ADDR(X)		68-6F*
Skip if Overflow Set	SØV	ADDR	ØV	10
Skip if A=0	SAZ	ADDR		11
Skip if B=0	SBZ	ADDR		12
Skip if X=0	SXZ	ADDR		13
Skip if A Negative	SAN	ADDR		14
Skip if X Negative	SXN	ADDR		15
Skip if A=B	SAB	ADDR		16
Skip if A=X	SAX	ADDR		17
Skip if Overflow Not Set	NØV	ADDR	ØV	18
Skip if A≠0	NAZ	ADDR		19
Skip if B≠0	NBZ	ADDR		1A
Skip if X≠0	NXZ	ADDR		1B
Skip if A Not Negative	NAN	ADDR		1C
Skip if X Not Negative	NXN	ADDR		1D
Skip if A≠B	NAB	ADDR		1E
Skip if A≠X	NAX	ADDR		1F
Branch On Condition	BØC	N,ADDR(X)		51
Save Machine State	SAV			5F00
Return	RET		all	5F01
Return Displaced	RTN	I	all	5F03

Figure 3-8 Transfer of Control Instructions

JMP - Jump

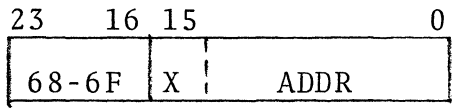


The jump instruction causes an unconditional transfer of control to the effective address. The effective address is loaded into the P register, thereby causing the next instruction to be fetched from the effective address. The recognition of interrupts is deferred until after the next instruction.

Indicators Affected: none.
Assembly Language Coding:

```
JMP/  ADDR(X)
```

RTJ - Return Jump



The return jump instruction is used for unconditional transfer of control to subroutines. The address of the instruction following the RTJ (that is, the return address) is stored at the effective address of the RTJ. The effective address +2 is then loaded into the P register. The first instruction of the subroutine is therefore located 2 bytes past the effective address of the RTJ. The recognition of interrupts is deferred until after the execution of the next instruction.

Indicators Affected: none.
Assembly Language Coding:

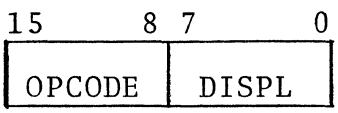
```
RTJ/  ADDR(X)
```

Programming Note:

RTJ and JMP instructions are used for transfer of control to and from subroutines. RTJ is used to transfer to a subroutine. It stores the return address in the first two bytes of the subroutine and transfers control to the instruction beginning in the third byte of the subroutine. A subroutine return may be accomplished by doing an indirect jump through the first word of the subroutine. The subroutine structure is as follows:

```
SUBNAME   DC       **          RESERVED FOR RETURN ADDRESS
*                                STORED BY "RTJ"
          ØPCØD    ØP1,ØP2     FIRST SUBROUTINE INSTRUCTION
            :
            .
            :
          ØPCØD    ØP1         LAST SUBROUTINE INSTRUCTION
          JMP*     SUBNAME     SUBROUTINE RETURN
```

Conditional Skips



The conditional skip instructions test register conditions or the overflow indicator and transfer control if the specified condition is met. The transfer of control may be to any location between 128 bytes behind and 127 bytes ahead of the byte following the skip instruction. If the tested condition is not met, instruction execution continues at the byte following the skip instruction. The following table lists the conditional skip instruction mnemonics and opcodes.

INSTRUCTION	MNEMONIC	OPCODE
SKIP IF OVERFLOW SET	SØV	10
SKIP IF A=0	SAZ	11
SKIP IF B=0	SBZ	12
SKIP IF X=0	SXZ	13
SKIP IF A NEGATIVE	SAN	14
SKIP IF X NEGATIVE	SXN	15
SKIP IF A=B	SAB	16
SKIP IF A=X	SAX	17
SKIP IF OVERFLOW NOT SET	NØV	18
SKIP IF A≠0	NAZ	19
SKIP IF B≠0	NBZ	1A
SKIP IF X≠0	NXZ	1B
SKIP IF A NOT NEGATIVE	NAN	1C
SKIP IF X NOT NEGATIVE	NXN	1D
SKIP IF A≠B	NAB	1E
SKIP IF A≠X	NAX	1F

Indicators Affected: ØV is reset by SØV and NØV

Assembly Language Coding:

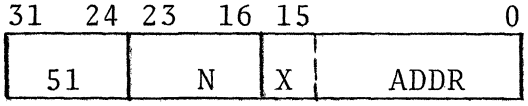
```
        SØV   ADDR
```

Programming Note:

The SØV and NØV instructions test and reset the overflow indicator. The arithmetic instructions set the overflow indicator when an overflow occurs but do not reset overflow if the operation does not result in an overflow, thus providing the ability to test the result of a series of arithmetic steps. SØV and NØV may be used to reset the overflow indicator without altering the word length by coding

```
    SØV   *+2
or
    NØV   *+2
```

BØC - Branch on Condition



This instruction is used to test the word length, overflow, interrupt system disabled, arithmetic and logical, or power failure indicators. The indicators to be tested are selected by a mask contained in the immediate data byte, "N". Each bit of the mask which is a 1 causes the corresponding indicator to be tested. If any of the selected indicators is a one the branch will occur. If none of the selected indicators are ones, the next sequential instruction is executed. The following table shows bit assignments for the mask.

CIP/2200 Indicator Selection

Mask Bit	Hex Mask Byte Value	Indicator Selected
0-1	00 to 03	Word Length. Values of 00_2 to 11_2 correspond to word length 1 to 4, respectively.
2	04	Overflow
3	08	Interrupt System Disabled.
4	10	High/Mixed Result
5	20	Low/Ones Results
6	40	Equal/Zero Result
7	80	Power Failure

Indicators Affected: none.
Assembly Language Coding:

```
BØC   N,ADDR(X)
```

Programming Note:

The Branch on Condition instruction differs from the conditional skip instruction in several ways. The BØC uses a full 2 byte address word, allowing a transfer to any part of memory. The conditional skips allow only relative addressing, with a restricted range. Conditional skips are, however, shorter (only 2 bytes long) and are faster in execution. The conditional skips test the current register contents while BØC tests the results of the last memory to memory or memory immediate operation. BØC also provides a means of testing the overflow indicator without altering its state.

SAV - Save Machine State

15 8	7 0
5F	00

The save instruction is used to save the machine state on the control stack. The machine state saved consists of nine bytes, arranged in the stack as shown in Figure 3-9.

X_{7-0}	high address	(stack pointer + 8)
X_{15-8}		.
A_{7-0}		.
A_{15-8}		.
B_{7-0}		.
B_{15-8}		.
STATUS		.
P_{7-0}		(stack pointer + 1)
P_{15-8}	low address	(stack pointer)

Figure 3-9 Control Stack Data Format

The address of the lowest addressed byte of the most recently saved machine state is contained in the stack pointer (locations 92_{16} and 93_{16}). The execution of the SAV instruction causes the stack pointer to be decremented by 9. If the stacking of the current machine state would cause the stack pointer to cross a page boundary (low 8 bits change from 00_{16} to FF_{16}) the machine state is not stacked and a control stack under/overflow interrupt is generated.

The SAV instruction stacks the current values of the A, B, X, and status registers. The P register value stacked is obtained from the 2 byte memory location immediately preceding the save instruction. This allows stacking the

BEFORE EXECUTION OF SAVE

P	0123
X	00FF
A	FFB2
B	003D
S	40

AFTER EXECUTION OF SAVE

P	0125
X	00FF
A	FFB2
B	003D
S	40

0121 0B04

SAVE INST. 0123 5F00

STACK POINTER MEMORY LOC. $92\text{-}93_{16}$	ADDR	STACK: EMPTY		STACK POINTER MEMORY LOC. $92\text{-}93_{16}$	ADDR	STACK: SAVED MACHINE STATE		
0FFF	0FFF	0	0		0FFF	0	0	
	0FFE	0	0		0FFE	F	F	- X
	0FFD	0	0		0FFD	0	0	
	0FFC	0	0		0FFC	B	2	- A
	0FFB	0	0		0FFB	F	F	
	0FFA	0	0		0FFA	3	D	- B
	0FF9	0	0		0FF9	0	0	
	0FF8	0	0		0FF8	4	0	- S
	0FF7	0	0		0FF7	0	4	- P
	0FF6	0	0	0FF6	0FF6	0	B	
	0FF5	0	0		0FF5	0	0	
	0FF4	0	0		0FF4	0	0	
	0FF3	0	0		0FF3	0	0	
	0FF2	0	0		0FF2	0	0	
	0FF1	0	0		0FF1	0	0	
	0FF0	0	0		0FF0	0	0	
	0FEF	0	0		0FEF	0	0	

Figure 3-10 SAV Instruction Control/Stack

return address in a subroutine called by an RTJ if the SAV follows the subroutine entry point. The contents of the machine registers are not altered by the save instruction.

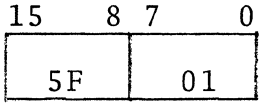
Recognition of interrupts is deferred until after execution of the instruction following the SAV.

Figure 3-10 shows the effect of the save instruction on the control stack, the stack pointer, and the machine registers.

Indicators Affected: none.
Assembly Language Coding:

```
SAV
```

RET - Return from Subroutine



The machine state most recently saved on the control stack is "popped" from the control stack and loaded into the P, A, B, X, and status registers. The control stack pointer is incremented to point to the previously saved machine state. Since the program counter is loaded with the value retrieved from the stack, control is transferred to the location specified by the stacked program counter value, normally the return address for the subroutine.

Recognition of interrupts is deferred until after execution of the next instruction. Execution of the RET instruction causes the stack pointer in locations 92_{16} and 93_{16} to be incremented by 9 to point to the previous machine state on the stack. If the addition of 9 to the stack pointer will cause the stack pointer value to cross a memory page boundary (the low 8 bits change from FF_{16} to 00_{16}) the RET instruction is not executed and a control stack under/overflow internal interrupt is generated.

BEFORE EXECUTION OF RET

P	0 1 3 A
X	0 1 0 0
A	0 0 0 0
B	F F F F
S	4 0

AFTER EXECUTION OF RET

P	0 B 0 4
X	0 0 F F
A	F F B 2
B	0 0 3 D
S	4 0

STACK POINTER MEMORY LOC. $92\text{-}93_{16}$	ADDR	STACK: SAVED MACHINE STATE
	0FFF	0 0
	0FFE	F F
	0FFD	0 0
	0FFC	B 2
	0FFB	F F
	0FFA	3 D
	0FF9	0 0
	0FF8	4 0
	0FF7	0 4
0 F F 6	0FF6	0 B
	0FF5	0 0
	0FF4	0 0
	0FF3	0 0

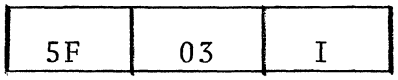
STACK POINTER MEMORY LOC. $92\text{-}93_{16}$	ADDR	STACK: EMPTY
0 F F F	0FFF	0 0
	0FFE	F F
	0FFD	0 0
	0FFC	B 2
	0FFB	F F
	0FFA	3 D
	0FF9	0 0
	0FF8	4 0
	OFF7	0 4
	0FF6	0 B
	0FF5	0 0
	0FF4	0 0
	0FF3	0 0

Figure 3-11 RET Instruction/Control Stack

Indicators Affected: all
Assembly Language Coding:

```
RET
```

RTN - Return Displaced from Subroutine



This instruction is identical in operation to the Return from Subroutine (RET) instruction except that the P register value retrieved from the stack is modified by the addition of the signed, 2's complement 8 bit displacement contained in the third byte of the RTN instruction.

Figure 3-11 shows the effect of the return instruction (RET) on the stack, stack pointer, and machine registers. The effect of the RTN instruction is identical except for the modification of the stacked P register value.

Indicators Affected: all
Assembly Language Coding:

```
RTN  I
```

Programming Note:

The SAV and RET (or RTN) instructions may be used in conjunction with the RTJ or JMP instructions to save the machine state prior to execution of a subroutine and to restore it when execution is complete. The subroutine may be called using an RTJ instruction as described previously. The subroutine, if written in the form shown below, will save the machine state at the entry point.

Call:

```
RTJ/    SUBR
```

Subroutine:

```
SUBR   DC     **
       SAV
        :
        :
        :
        .
       RET
```

If desired, the calling program can cause saving of the machine state, relieving the subroutine of the responsibility. An additional benefit of having the calling pro

gram save the machine state is that subroutines with multiple entry points need not skip around unused entry point locations.

```
Call:                                    Subroutine:

   :
   .
JMP     *+4                              SUBR   OPC    OPND
DC      A'*+7    RETURN ADDRESS                :
SAV                                               ;
                                                  RET
JMP/    SUBR
```

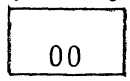
3.7 Control Instructions

Control Instructions have one or two byte opcodes and do not refer to memory. They perform miscellaneous control functions on the computer. The table below lists the instructions in this section. Instructions marked ! activate hardware options which may not be implemented on all CIP/2200 systems.

INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Halt	HLT			00
Trap	TRP			01
Enter Sense Switches	ESW			02
Disable Interrupt System	DIN!		INT	04
Enable Interrupt System	EIN!		INT	05
Disable Interval Timer	DIT!			06
Enable Interval Timer	EIT!			07
No Operation	NOP			34
ROM Exit	XIT!			5E
Secondary ROM Exit	XT2!	N		5F08-5FFF

Figure 3-12 Control Instructions

HLT - Halt

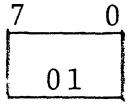


The processor and DMC I/O operations are halted. Interrupts are not recognized by the CPU when in the halt state. The P register contains the address of the HLT instruction after the computer halts. Depressing either the console run or step switches will cause the next instruction to be executed. Depressing the console interrupt switch will cause the computer to enter the run state and execute the interrupt.

Indicators Affected: none.
Assembly Language Coding:

HLT

TRP - Trap

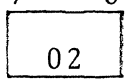


The TRP instruction causes an internal interrupt through the console interrupt location ($80\text{-}81_{16}$). The contents of the P register (the address of the TRP) are stored at the two byte memory location specified by the two byte address word at location 80_{16}. Then the contents of the two byte address word (at 80_{16}) plus two replace the contents of the P register. Interrupts are not recognized before the execution of the next instruction. Depressing the console interrupt switch has the same effect as executing the TRP instruction.

Indicators Affected: none.
Assembly Language Coding:

TRP

ESW - Enter Sense Switches



The status of the four console sense switches replace bits 15-12 of the A register; switch 4 in bit 15, 3 in 14, 2 in 13, and 1 in 12. Bits 11-8 of the A-register are set to ones and the remainder of the register is unchanged.

Indicators Affected: none.
Assembly Language Coding:

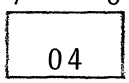
ESW

Programming Note:

After the sense switch status has been entered into the A register using the ESW instruction, the state of a particular sense switch may be tested by shifting the A register to bring the sense switch bit into bit 15 of the A register (the sign bit) and then testing the A register for a negative or non-negative condition using the SAN and NAN conditional skip instructions. For example, to test for sense switch 2 being set, the following instruction sequence may be used.

```
ESW
ALA   2
SAN   SSW2ON      SKIP IF SS#2 ON
```

DIN - Disable Interrupt System

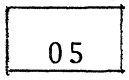


External interrupts are masked, causing the recognition of external interrupt requests to be deferred. The interrupt system disabled indicator in the machine status register is set to a one. While the interrupt system is disabled external interrupt requests are saved by the byte I/O system.

Indicators Affected: INT
Assembly Language Coding:

```
DIN
```

EIN - Enable Interrupt System



External interrupts are unmasked, allowing the processor to recognize saved and subsequent requests for interrupts. Interrupt requests are not recognized until after the execution of the instruction following the EIN instruction. The interrupt system disabled indicator is reset to a zero.

Indicators Affected: INT
Assembly Language Coding:

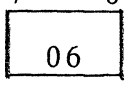
```
EIN
```

Programming Note:

DIN and EIN are used to mask and unmask external interrupts. The SAV instruction saves the contents of the machine status register, including the interrupt system disabled indicator. When control is transferred using the RET or RTN instructions the state of the interrupt system is restored to the state indicated by the saved interrupt system disabled indicator if the interrupt enable/disable hardware option is installed. A subroutine or interrupt service routine which must disable interrupts should use SAV and RET or RTN to restore the interrupt system to the original state, as illustrated below.

```
SUBR    DC    **
        SAV
         :
         :
        RET
```

DIT - Disable Interval Timer

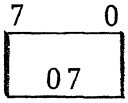


Updating of the two-byte interval timer at location $84\text{-}85_{16}$ and generation of interval timer interrupts are inhibited. Interrupt requests are not recognized until after execution of the next instruction.

Indicators Affected: none.
Assembly Language Coding:

```
DIT
```

EIT - Enable Interval Timer



Updating of the two-byte interval timer at location $84\text{-}85_{16}$ and the generation of interval timer interrupts are allowed. The counter is updated by adding one at each basic timer interval, normally one millisecond. The rate of updating the interval timer depends upon the strapping of the real time clock option on the CPU option board. An interval timer interrupt is generated when incrementing the counter in location $84\text{-}85_{16}$ causes the counter value to become zero.

Indicators Affected: none.
Assembly Language Coding:

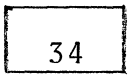
```
EIT
```

Programming Note:

To obtain an interval timer interrupt after a given number of basic clock intervals, the timer counter is set to the negative of the number of timer periods desired and the interval timer interrupt location ($86\text{-}87_{16}$) is initialized with the address of the interval timer interrupt service routine. The following code will set up the timer to interrupt after one second, assuming standard strapping of the interval timer.

```
START    LDA=   F'-1000'      SET INTERVAL COUNTER
         STA    X'84'         FOR 1 SEC
         LDA=   A'TIMER'      SET UP INTERRUPT
         STA    X'86'           ADDRESS
         EIT                  START THE TIMER
          :
          :
TIMER    DC      **           TIMER ISR
         SAV
         LDA=   F'-1000'      RESET THE COUNTER
         STA    X'84'
          :                   BODY OF TIMER ISR
          :
         RET                  RETURN TO BACKGROUND
```

NØP - No Operation

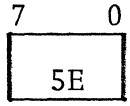


This instruction performs no operation. Program execution continues at the next byte. The machine state is not altered by the NØP instruction.

Indicators Affected: none.
Assembly Language Coding:

```
NOP
```

XIT - CIP/2200 ROM Exit

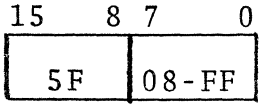


This instruction provides for the addition of a special instruction to the standard CIP/2200 instruction set. Execution of this instruction causes a microprogram transfer of control to the first word of the first unused ROM page. If no special ROM is present, execution of an XIT causes the CPU to enter a non-interruptable microprogram loop. Normal execution may be resumed from this state by pushing Clock-Reset-Run or Clock-Reset-Interrupt. Appendix F contains a description of the procedure for adding custom microcode to the standard CIP/2200.

Indicators Affected: none.
Assembly Language Coding:

XIT

XT2 - Secondary CIP/2200 ROM Exit



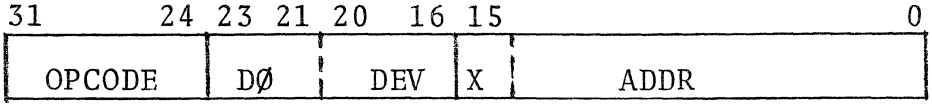
This instruction provides for the addition of special instructions to the standard CIP/2200 instruction set. Execution of an XT2 causes a microprogram transfer of control to the second word of the first unused ROM page. At the time control is transferred, the CIP/2000 file register 1 contains the second byte of operation code. Decoding of this byte by the user microprogram provides up to 248 additional instructions. Execution of an XT2 without special ROM causes the CPU to enter an endless microprogram loop which is not interruptable. This condition may be cleared by pushing Clock-Reset-Run or Clock-Reset-Interrupt. Appendix F contains a description of the procedure for adding custom microprogrammed extensions to the standard CIP/2200 firmware.

Indicators Affected: none.
Assembly Language Coding:

XT2 N (0 $\leq$ N $\leq$ 247)

3.8 Input/Output Instructions

The input/output instructions consist of either two or four bytes; an opcode, a byte for control information, and an address word when necessary.



The opcode byte specifies the type of I/O instruction to be performed. Byte mode instructions use the second instruction byte for control information. The device order, DØ, contains a 3 bit code for the operation to be performed. The device number, DEV, is a 5 bit number which specifies the I/O device to be used. Byte mode instructions which transfer data to or from memory have an additional two byte address word following the control byte. Serial I/O instructions do not use the control byte but its presence is required. All I/O instructions have a one instruction interrupt umbrella, i.e., interrupts are not recognized following an I/O instruction. The CIP/2200 I/O instructions are listed in Figure 3-13.

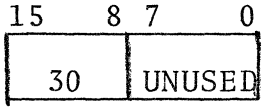
INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Input Byte Serially	IBS			30
Input Byte to A	IBA	DØ,DEV		31
Input Byte to B	IBB	DØ,DEV		32
Input Byte to Memory	IBM	DØ,DEV, ADDR(X)		33
Output Byte Serially	ØBS			38
Output Byte From A	ØBA	DØ,DEV		39
Output Byte From B	ØBB	DØ,DEV		3A
Output Byte From Memory	ØBM	DØ,DEV, ADDR(X)		3B

Figure 3-13 I/O Instructions

Serial I/O Instructions

The serial I/O instructions are used for communicating with a low speed terminal device such as the teletype. When a serial I/O instruction is executed, the computer suspends all processing until the complete data byte has been transferred.

IBS - Input Byte Serially

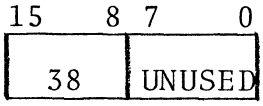


A byte is input to the low byte of A from the serial I/O device. The eight high order bits of A are unchanged. Execution of this instruction terminates when a complete byte has been received. If no device is connected to the serial I/O interface, the IBS instruction performs no operation.

Indicators Affected: none.
Assembly Language Coding:

```
IBS
```

ØBS - Output Byte Serially



A byte is output from the low byte of A to the serial I/O device. The eight low order bits of A are set to 1's; the eight high order bits are unchanged. Execution of this instruction terminates when the complete byte has been output. If no device is connected to the serial I/O interface, the only effect of the ØBS is to set the low byte of A to 1's and to cause a 100ms pause in computer and I/O operations.

Indicators Affected: none.
Assembly Language Coding:

ØBS

Byte I/O Instructions

The byte I/O instructions are used for all byte mode operations and for controlling DMC and DMA transfers.

Each I/O device controller has a unique 5-bit device number assigned to it for identification purposes. The device number is used in all byte I/O instructions to specify which device controller is to perform the operation. The device number also defines a unique interrupt transfer location for the associated external interrupt. Device numbers range from 0 to 31_{10}; the external interrupt location is twice the device number plus 100_{16}.

The device order field of a byte I/O instruction specifies the operation to be performed. The standard device orders are shown in Figure 3-14. In general, all CIP/2200 I/O devices use these device order definitions, but these meanings may be altered for a particular device. The product performance specification for each device gives the exact interpretation of the various device orders and should always be consulted when writing I/O programs. The following paragraphs describe the standard device order interpretations.

The data order (000_2) is used to perform byte transfers between the addressed device controller and the A register, the B register, or memory. The direction of transfer (input or output) is determined by the opcode field of the instruction.

The status/function order (001_2) is used to input device status to, or output a control byte from, the A register, the B register, or memory. The opcode field of the instruc-

Device Order	Operation	Description
0	Data	The data order causes a data byte to be transferred between the processor and the addressed device. The direction of transfer depends on the type of instruction (input or output).
1	Status/ function	The status/function order causes a status byte to be transferred from the addressed device to the processor, or a function byte to be transferred from the processor to the device depending on the type of instruction (input or output).
2	Arm	The arm device order notifies the device to set its external interrupt sequencer into an armed state so that it can generate a request for service to the processor.
3	Disarm	The disarm device order notifies the device to set its external interrupt sequencer into a disarmed state.
4	Disconnect	The disconnect order causes the block input or output operation in progress to be stopped. An external interrupt is generated if an interrupt would normally have been generated at the end of the block transfer.
5	Block Input	The block input order notifies the device to start a DMC block input into memory.
6	Block Output	The block output order notifies the device to start a block output from memory (DMC).
7	Device Dependent Order	This order may be assigned definitions unique to particular I/O controllers.

Figure 3-14 Standard I/O Device Orders

tion determines whether a status or a function byte is transferred. Output instructions cause a function byte to be placed on the byte I/O bus. Input instructions cause the status byte of the addressed device to be read. The status byte contains information about the state of the I/O device, the device controller, and the last data transfer. The standard meanings for bits 0-3 of the status byte are shown in Figure 3-15. Device dependent bit assignments may be found in the product performance specification for a particular I/O device. The function byte consists of device dependent control information which is detailed in the product performance specification for the various controllers.

Bit Number	Condition	Description
0	Ready	This bit is set to 1 when the I/O device is in a ready state.
1	Input Flag	This bit is set to 1 when the I/O device has a byte ready for input to the processor.
2	Output Flag	This bit is set to 1 when the I/O device is ready to receive a byte from the processor.
3	Error	This bit is set to 1 when an error has occurred during a transfer operation. The error may be the result of timing, parity, or a device malfunction.
4-7	Device Dependent	These four bits are unique for each I/O device.

Figure 3-15 Standard Status Bit Assignments

The arm (010_2) and disarm (011_2) orders affect the device controller interrupt sequencer. Each device controller has a four state interrupt sequencer. A controller in the armed state may generate an external interrupt request to the computer in response to some stimulus (data ready, error, etc.). When an interrupt request is generated the device controller enters the wait state. On recognition of an interrupt request by the CPU, the computer

executes an RTJ through the interrupt transfer location assigned to the requesting device. Acceptance of the request by the CPU places the controller in the active state. The active state is maintained until either an arm or disarm order is issued. When the controller is in the disarmed state it cannot generate an interrupt request.

The disconnect order (100_2) stops the I/O operation currently in progress. If the device controller is armed, the disconnect order will also cause an external interrupt request to be generated.

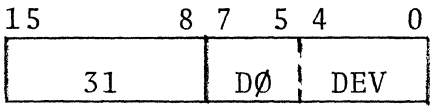
The block input (101_2) and block output (110_2) orders are used to start DMC operations. Before issuing a block input or block output order the programmer must initialize the DMC descriptor, check the device status byte for the device ready state, and arm the device controller if an interrupt is to be generated at the end of the DMC operation. The DMC descriptor for each device is a four byte area located on page zero at the I/O device address times 4 (e.g., the descriptor for device 2 occupies memory locations $08\text{-}0B_{16}$). The first two bytes contain the address of the next byte to be transferred; the last two bytes contain the address of the last byte to be transferred. After the DMC transfer is started, it continues automatically until the current address in the DMC descriptor is one greater than the ending address. The last byte is transferred to or from the location specified by the ending address. The transfer is then terminated and an interrupt request is generated if the controller is in the armed state.

A DMC operation always results in the transfer of at least one data byte regardless of the current and ending addresses. This allows an interrupt to be generated for each character transferred in the DMC mode if the current address is equal to or greater than the ending address before the DMC transfer starts.

The device dependent order code (111_2) specifies non-standard I/O operations unqiue to a particular device. Refer to the individual device controller product performance specifications for the specific action taken.

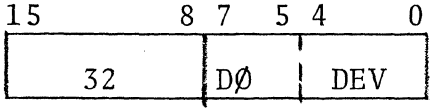
DMA I/O transfers require special programming as described in Appendix E and in the DMA controller Manual.

IBA - Input Byte to A



The device order, DØ, is sent to the device designated by DEV. The byte of data on the I/O bus is then placed in the low 8 bits of the designated register. The eight high order bits of the register are unchanged.

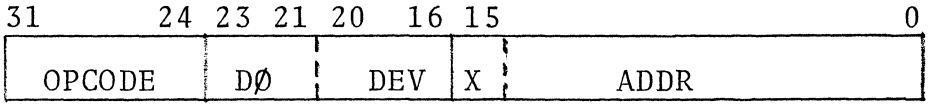
IBB - Input Byte to B



Indicators Affected: none.
Assembly Language Coding:

```
IBA   DØ,DEV
0≤DØ≤7, 0≤DEV≤31
```

IBM - Input Byte to Memory

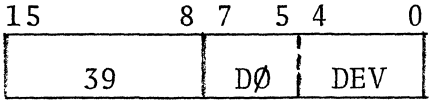


The device order, DØ, is sent to the device designated by DEV. The byte of data on the input bus is then input to the memory location specified by the address word in the third and fourth bytes of the instruction.

Indicators Affected: none.
Assembly Language Coding:

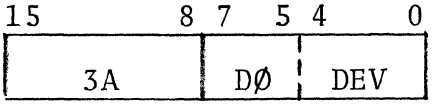
```
IBM  DØ,DEV,ADDR(X)
     0≤DØ≤7, 0≤DEV≤31
```

ØBA - Output Byte From A



The device order, DØ, is sent to the device designated by DEV. The data byte contained in the low byte of the designated register is then placed on the I/O bus. The data in the register is not altered.

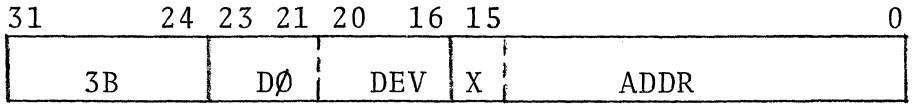
OBB - Output Byte From B



Indicators Affected: none.
Assembly Language Coding:

```
OBA   DØ,DEV
      0≤DØ≤7, 0≤DEV≤31
```

OBM - Output Byte from Memory



The device order, DØ, is sent to the device designated by DEV. A byte of data is then output from the memory location indicated by the address word in the third and fourth bytes of the instruction.

Indicators Affected: none.
Assembly Language Coding:

```
OBM  DØ,DEV,ADDR(X)
     0≤DØ≤7, 0≤DEV≤31
```

Programming Note:

When an interrupt request is recognized by the CPU, an RTJ to the interrupt service routine (ISR) entry point is forced by the computer. The entry point of the ISR is specified by the interrupt location for the requesting device. The address of the interrupt location for a device is twice the device address plus 100_{16}. Device 3, for example, interrupts through locations 106 - 107_{17}. The RTJ causes the current value of the program counter (P register) to be saved in the ISR entry point and control to be transferred to the entry point +2. To save registers and machine status the instruction at "entry point +2" should be SAV. Following the SAV instruction is the body of the ISR. When the function of the ISR has been completed, the interrupting device interrupt sequencer must be removed from the active state by arming or disarming the device. The I/O instruction used to perform the arm or disarm operation should be executed immediately before the execution of the return instruction to prevent recursive calls on the ISR. If the machine state was saved using the SAV instruction, the RET instruction should be used to return control to the interrupted program. If for some reason the SAV instruction was not used (as, for example, in the control stack overflow/underflow ISR), a JMP indirect through the ISR entry point should be used.

The structure of an ISR is shown below.

```
INTSUB    DC    **            INTERRUPT SERVICE ROUTINE
*                                 RETURN ADDRESS.
          SAV                 SAVE MACHINE STATE
           .
           .                  BODY OF ISR
           .
          ØBA ARM,DEV         ARM DEVICE CONTROLLER
*                                 FOR NEXT INTERRUPT.
          RET                 RETURN TO INTERRUPTED
*                                 PROGRAM.
*
ARM       EQU 2               ARM DEVICE ORDER
```

The first word of the interrupt service routine must be reserved for the return address.

4. COMPUTER OPERATION

This chapter describes the CIP/2200 operating procedures. The operation of the various front panels is described first, followed by a description of the firmware bootstrap loader and the disk IPL option.

4.1 Front Panel

There are three front panel configurations available for the CIP/2200:

1. Blank Panel

   This panel is intended for dedicated system applications where no operator control is required. There are no external switches or indicators on the console. The power fail/automatic restart option must be installed in systems having a blank front panel to allow handling of power on and off conditions.

2. Basic Panel

   This panel provides the basic control capabilities required for program development and general program execution, including power on/off, the switches and indicators associated with starting, stopping, and interrupting execution, and four sense switches.

3. System Panel

   The system panel provides the highest level of control and display facilities. The fundamental control capabilities of the basic panel are supplemented with 16 command switches and a bank of indicator lights which display the contents of important internal machine registers.

Figure 4-1 shows the system panel. The functions of the various switches and indicators are described in the following section. Indicator lights are illuminated when the bit represented is a one. Depressing the top of a switch places it in the "on" or 1 condition.

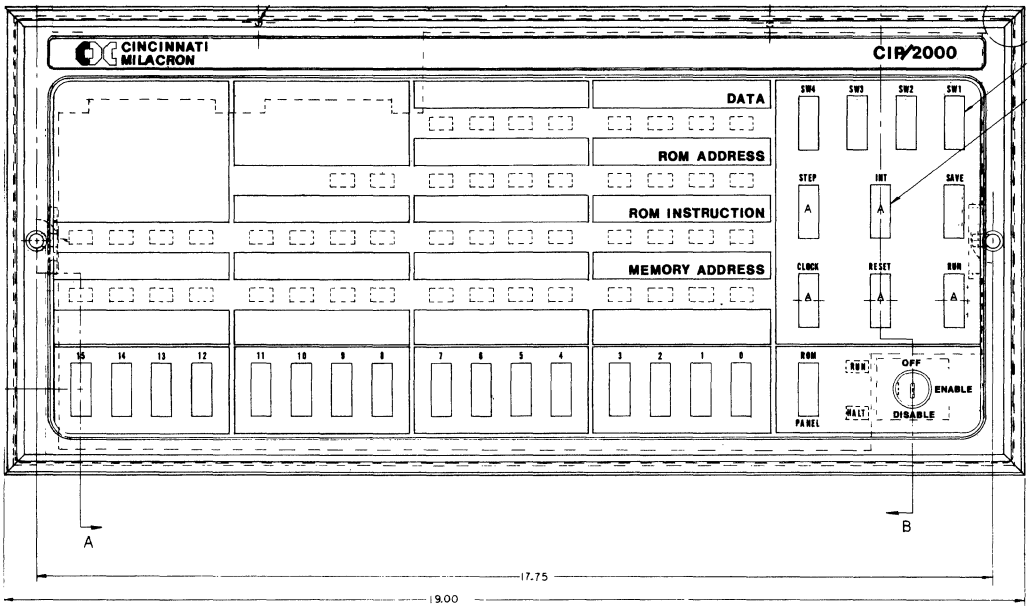


Figure 4-1 System Panel

Basic Panel Operation

The control switches described in this section are present on both the basic panel and the system panel. These controls allow the operator to load a program; start, stop, or interrupt a program; and to exert limited control on program execution (via sense switches).

POWER OFF/ENABLE/DISABLE

This is a three position key switch which controls the main power and front panel access. In the OFF position, power is removed from the computer. In the ENABLE position power is on and the front panel switches are enabled. In the DISABLE position power is on but all front panel switches are disabled, preventing interference with a running program through inadvertent switch operation.

RUN Light

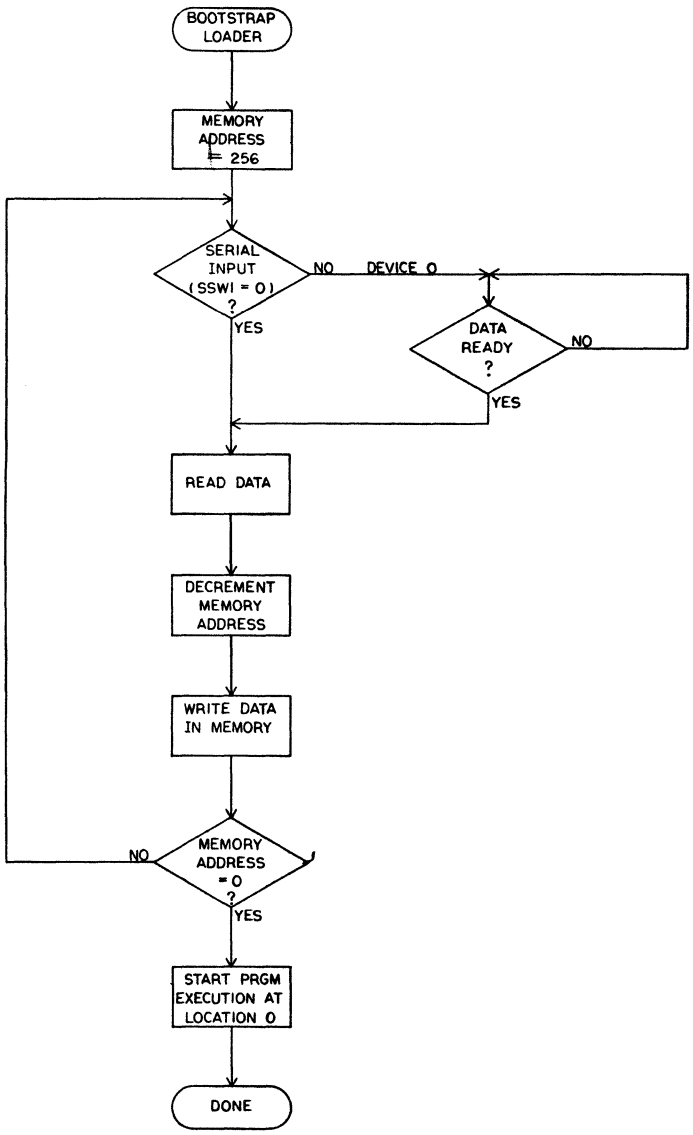
This light is illuminated when the computer is operating, or in the "RUN" state.

HALT Light

This light is illuminated when the computer is halted.

BOOTSTRAP LOADER FLOWCHART

FIG. 4-2



CLOCK

The CLOCK switch is a momentary contact switch which causes the computer to halt at the end of the current microinstruction. Depressing CLOCK when the computer is halted causes one microinstruction to be executed. Execution may be resumed by pushing RUN.

RESET

This momentary contact switch halts the computer and clears the P register and the machine status byte. When the computer resumes execution, the first instruction processed will start at memory location 0. RESET also halts any I/O transfer in progress and resets all I/O interface logic and interrupt sequencers.

NOTE: The RESET switch should not be used to halt the computer if the program currently executing is to be saved. RESET prevents the current instruction from being completed normally, and may cause a byte of memory to be altered because a memory cycle may be interrupted. To prevent this possible loss of data, the computer should be halted by pushing the STEP switch (see below).

RUN

The momentary contact RUN switch starts computer operation when the computer is in the HALT state. The sequence STEP-RESET-RUN causes the computer to halt; resets all hardware indicators and initializes the I/O; sets the status register and the program counter (P register) to zero; and starts execution with the instruction located at memory address zero. Normal software convention is to place a JMP instruction to the actual start of the program at location zero, thus allowing a program to be started or restarted by pushing STEP-RESET-RUN.

NOTE: If sense switch 4 is on, RESET-RUN causes execution of the bootstrap loader (see Section 4.3).

STEP

This momentary contact switch is used to halt the computer at the end of the next interruptable instruction. Each time STEP is pushed while the computer is halted the computer executes the next instruction and halts if the

instruction is interruptable. If the instruction executed by a STEP is not interruptable, execution continues until an interruptable instruction is encountered. If a loop composed entirely of non-interruptable instructions is entered, STEP has no effect and CLOCK must be used to halt the computer. If the STEP switch is released before the completion of an instruction or sequence of non-interruptable instructions, the computer will not halt. In practice, this occurs only in the case of IBS and OBS instructions.

INTERRUPT

The momentary contact INTERRUPT switch generates a console interrupt. If the computer is in the HALT state, the INTERRUPT switch first places the computer into the RUN state, then causes a console interrupt. Normally the INTERRUPT switch alone should be depressed to cause a console interrupt. If, however, the computer is in a non-interruptable loop, a console interrupt may be forced by pushing CLOCK-RESET-INTERRUPT. The latter method, of course, results in a zero return address being stored by the interrupt.

SAVE

This switch performs the same function as the RESET switch except that it maintains the "reset" condition. If the system does not have the power fail/automatic restart option, the SAVE switch must be on to protect main memory contents when power is turned on or off. The computer should first be halted by pressing STEP; i.e., the proper power off sequence is STEP-SAVE-POWER OFF.

SENSE SWITCHES: SW1, SW2, SW3, SW4

These switches allow a running program to respond to operator inputs. The state of the sense switches may be input to the A register with an ESW instruction and tested by the program to allow the operator limited control of the program without requiring an I/O device.

The sense switches are also used by the CIP/2200 to allow the operator to initiate the firmware initial program load process. When execution is started after a RESET, the CPU examines the sense switch settings, and if SW4 is on, performs a bootstrap load or "IPL" (Initial Program Load) as described in Sections 4.4 and 4.5.

4.3 System Panel Operation

The system panel contains all of the controls described previously for the basic panel, and in addition provides facilities for displaying the contents of various internal registers via four sets of indicator lights. The 16 command switches may be used to enter microinstructions to examine or alter the contents of memory and the CIP/2200 registers.

ROM/PANEL Select

This switch controls the source of microinstructions executed by the computer. The normal position is ROM, which causes the computer to fetch microinstructions from the internal read only memory. When the ROM/PANEL switch is placed in the PANEL position, the 16 command switches are the source of microinstructions. The CPU executes the microinstruction defined by the command switches when CLOCK is depressed.

COMMAND Switches

These 16 switches are substituted for the internal read only memory when the ROM/PANEL select switch is in the PANEL position. The state of the command switches determines a 16 bit microinstruction which can be executed once by pushing CLOCK or repeatedly by pushing RUN.

MEMORY ADDRESS Display

This display shows the current contents of the CIP/2200 memory address register. When the CIP/2200 halts normally (by executing a HLT) the memory address display gives the address of the HLT instruction.

ROM INSTRUCTION Display

These indicators display the next microinstruction to be executed.

ROM ADDRESS Display

These indicators display the address of the next micro-instruction to be executed.

DATA Display

This display gives the current contents of the internal data bus, the A bus.

System Panel Procedures

The system panel may be used to display and modify the contents of the CIP/2200 registers or main memory without software support. The program counter (P register) may also be modified via the system panel to start program execution at a given memory location. These operations are performed by executing micro-instructions entered through the command switches. This is done by placing the ROM/PANEL switch in the PANEL position, setting the 16 command switches to the bit pattern representing the desired microcommand, and executing the microinstruction by pushing CLOCK. The DATA lights are used to display the contents of the selected byte.

Register Display

The current contents of the A, B, X, S or P register can be displayed on the DATA indicators. The micro-instruction used to display registers is:

C<u>f</u>00

The second hexadecimal digit of the command ("f"), specifies the register to be displayed on the DATA lights. Since the DATA display is 8 bits long, each 16 bit register is displayed in two parts. The following table contains the hexadecimal digit corresponding to each CIP/2200 register. The subscript L denotes the low 8 bits of the register; U refers to the high 8 bits.

REGISTER	f
X_L	2
X_U	3
A_L	4
A_U	5
B_L	6
B_U	7
S	8
P_L	C
P_U	D

For example, to display the A register contents, the computer is halted by pushing STEP, the ROM/PANEL switch is set to PANEL, the command switches are set to C400, and CLOCK is depressed. This displays the low byte of A. The high byte may be displayed by setting the command switches to C500 and pushing CLOCK.

Register Modification

The CIP/2200 register contents may be modified by the microcommand

2fXX

The register to be changed is given by "f" as in the display operation. The value to be placed into the register is given by "XX", the low byte of the command.

For example, to load the A register with 260_{10} (0104_{16}) the following sequence of steps is executed.

1. Halt the computer by pushing STEP and set the ROM/PANEL switch to PANEL.
2. Set the command switches to 2404_{16}; push CLOCK.
3. Set the command switches to 2501_{16}; push CLOCK.

Starting Execution

Execution may be started at a specific location by loading the P register with the desired address and starting the CIP/2200 microprogrammed instruction fetch routine. The following steps are used.

1. Load the P register with the address at which execution is to start, as described in the previous section.
2. Execute a microinstruction jump to the start of the firmware instruction fetch routine. This microinstruction is 1409 for the CIP/2200.
3. Set the ROM/PANEL switch to ROM and push RUN.

For example, to start execution at location 200_{16}, the following microinstructions are entered and executed via the command switches:

2C00	load P_L with 00_{16}
2D02	load P_U with 02_{16}
1409	jump to 2200 instruction fetch

The ROM/PANEL switch is then placed in the ROM position and RUN is pushed to start program execution.

Memory Display and Modification

CIP/2200 main memory locations can be displayed on the front panel DATA indicators with the microcommand sequence shown:

```
13XX   load low byte of memory address
12XX   load high byte of memory address
A000   read memory
B020   display memory data
```

To read the contents of memory location 200_{16} the instructions 1300 and 1202 are used to load the memory address register. The instructions A000 and B020 are executed to place the memory data in the DATA display.

To change a memory location the sequence shown below is used:

```
13XX   load low byte of memory address
12XX   load high byte of memory address
11XX   load memory data
A010   write data
```

For example, to load location 200_{16} with $B3_{16}$ the following instructions are executed:

```
1300   load memory
1202   address register
11B3   load memory data
A010   write
```

The procedures described above are applicable for reading or changing a small number of memory locations. The CIP/2200 also has a microprogrammed memory access routine that is more convenient for the display or modification of a larger number of contiguous memory locations using the system panel. The system panel memory access program displays successive memory locations in the DATA indicators and allows the operator to insert changes where necessary. The computer halts after each data byte is displayed. The data can be changed by depressing sense switch 4 and entering the new data in the low order 8 COMMAND switches. When RUN is pushed the new data is read from the low 8 COMMAND switches and written at the current location (displayed in the MEMORY ADDRESS indicator). The memory address is then incremented

and the next data byte is read and displayed with the computer in the HALT state. If sense switch 4 is off, a write does not occur and the next data byte is displayed.

To execute the system panel memory access program the following setup steps are necessary.

1. Halt the computer using STEP and place the ROM/PANEL switch to PANEL.
2. Load file registers A and B with the low and high bytes respectively of the address of the location which precedes the first location to be displayed.
3. Perform a microprogram jump to the front panel routine. This microinstruction is 1DFE.
4. Set the ROM/PANEL switch to ROM and push RUN.
5. Display and change memory as desired, pushing RUN to step to the next location and setting sense switch 4 as shown below.

OPERATION	SSW 4	COMMAND SWITCHES 0-7
Read	ØFF	0000 0000
Write	ØN	DATA DATA

To display and modify a block of locations beginning at location 510_{16} the following microinstructions would be executed.

```
2A0F   load low byte of (address -1)
2B05   load high byte of (address -1)
1DFE   jump to front panel routine
```

Set the ROM/PANEL switch to ROM and push RUN to execute the memory modification routine. When the desired memory examination or modification has been completed, RESET-RUN or RESET-INTERRUPT will return the computer to normal operation.

4.4 Bootstrap Loader

The bootstrap loader provides an initial program load (IPL) facility for the CIP/2200. The loader is a firmware routine which reads 256 bytes of data from a teletype or other byte oriented input device and stores it in memory locations 0 - FF_{16}. After loading 256 bytes of data, the bootstrap loader transfers control to location 0.

Description of Operation

The bootstrap loader is executed by turning sense switch 4 on and sense switches 2 and 3 off; pushing the "RESET-RUN" switches, and starting the IPL device. Sense switch 1 specifies the input device. The IPL microprogram reads from the serial teletype interface if sense switch one is off, and reads from byte I/Ø device zero if sense switch 1 is on. The status byte and device orders for device zero must conform to the standard I/Ø status byte and device order definition if the IPL is to work properly.

Figure 4-2 shows the functional flowchart for the bootstrap loader. The loader reads data from either the serial teletype or device zero, according to the state of sense switch 1. The data is stored in the first 256 bytes of memory starting with location FF_{16} and continuing through successively lower addresses until 256 bytes have been loaded. When loading is complete, control is transferred to location 0, causing the initial program to start execution.

Initial Program Requirements

The program loaded by the IPL process will generally be a loader for reading the system software into memory. A program to be loaded by IPL must be prepared in the special IPL format. Because the bootstrap loader loads from location FF_{16} to location 0, the initial program must be in reverse order in the input medium, formatted so that the first byte loaded corresponds to the data for location FF_{16}. The IPL device must be positioned so that the first byte of data read will be the first byte of the initial program, as the IPL microprogram simply reads the first 256 bytes of data present. The IPL microprogram performs no device control functions, so a means must be provided for the operator to start the IPL device manually (e.g., the teletype tape reader

has a switch which starts the reader). Paper tapes to be loaded via the IPL facility are usually prepared with a rubout (all channels punched) punched as the first byte of the 256 byte IPL record to enable the operator to position the tape properly in the reader.

Bootstrap Loading from the Serial TTY

The following procedure is used for performing an initial program load from the serial teletype.

1. Turn sense switch 4 on, turn sense switches 1, 2, and 3 off.
2. Place the basic loader tape in the serial teletype reader, positioning the first rubout over the read station.
3. Press the RESET and RUN switches on the front panel. This starts the IPL microprogram which will wait for the teletype reader to be started.
4. Start the teletype reader. The bootstrap loader operation can be verified on the system front panel by watching the memory address decrement from FF_{16} to 0.
5. Stop the teletype reader manually at the end of the bootstrap operation.

4.5 Disk IPL Option (CIP/2210)

The disk IPL option provides an initial program load facility for the CIP/2200 from a disk memory attached to the DMA channel. The IPL program moves up to 32k bytes of core image data from the disk to main memory and transfers control to a user specified memory location. This option requires a disk drive, a disk controller, and a DMA channel.

Core Load Requirements for Disk IPL

The IPL core image data must start at the beginning of the disk and occupy contiguous disk locations. The maximum core load size is 32,768 bytes, the minimum is the amount of data stored in two disk sectors. Loading begins at memory location zero and continues until all data has been entered.

The IPL process begins by reading the first two sectors from cylinder 0, track 0 into memory beginning at location 0 and continuing through successively higher locations. After the contents of the first two sectors have been moved into memory, the IPL microprogram examines the 8 byte area at memory location 180 - 187_{16}. The four words in this area specify the parameters for the remainder of the loading process.

These parameters are as follows:

Location	Description
$180\text{-}181_{16}$	Address of last byte of data to be loaded. This parameter specifies tthe core load size.
$182\text{-}183_{16}$	Number of data bytes contained in one disk sector.
$184\text{-}185_{16}$	Number of data bytes contained in one disk cylinder.
$186\text{-}187_{16}$	Execution address. Control will be transferred to this location at the end of the IPL.

IPL Operation

The IPL program is activated by turning sense switches three and four on, sense switches one and two off and pushing RESET-RUN. After loading the core image data the IPL program transfers control to the memory location specified in the core load. Errors detected by the disk controller will cause the computer to halt with the disk controller status bytes in file registers 4 and 5. The IPL may be retried at any time.

APPENDIX A

INSTRUCTION SET LISTED NUMERICALLY BY OPCODE

The following terms are used in Appendixes A and B to describe valid assembly language operands for the various instructions.

ADDR Operand expression used to calculate an address
I An absolute expression used to specify data which may be negative.
N An absolute expression used to specify data which may not be negative, such as counts, masks, etc.
L An absolute expression used to specify an explicit length value.
Instructions marked * have all addressing modes.
Instructions marked † have interrupt umbrella.

OPCODE	INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS
00†	Halt	HLT		
01†	Trap	TRP		
02	Enter Sense Switches	ESW		
03	Interchange A & B	IAB		
04†	Disable Interrupt System	DIN		INT
05†	Enable Interrupt System	EIN		INT
06†	Disable Interval Timer	DIT		
07†	Enable Interval Timer	EIT		
08	Reset ØV, Set WL=1	RØ1		ØV,WL
09	Reset ØV, Set WL=2	RØ2		ØV,WL
0A	Reset ØV, Set WL=3	RØ3		ØV,WL
0B	Reset ØV, Set WL=4	RØ4		ØV,WL
0C	Set ØV, Set WL=1	SØ1		ØV,WL

OPCODE	INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS
0D	Set ØV, Set WL=2	SØ2		ØV,WL
0E	Set ØV, Set WL=3	SØ3		ØV,WL
0F	Set ØV, Set WL=4	SØ4		ØV,WL
10	Skip if Overflow Set	SØV	ADDR	ØV
11	Skip if A=0	SAZ	ADDR	
12	Skip if B=0	SBZ	ADDR	
13	Skip if X=0	SXZ	ADDR	
14	Skip if A Negative	SAN	ADDR	
15	Skip if X Negative	SXN	ADDR	
16	Skip if A=B	SAB	ADDR	
17	Skip if A=X	SAX	ADDR	
18	Skip if Overflow Not Set	NØV	ADDR	ØV
19	Skip if A≠0	NAZ	ADDR	
1A	Skip if B≠0	NBZ	ADDR	
1B	Skip if X≠0	NXZ	ADDR	
1C	Skip if A Not Negative	NAN	ADDR	
1D	Skip if X Not Negative	NXN	ADDR	
1E	Skip if A≠B	NAB	ADDR	
1F	Skip if A≠X	NAX	ADDR	
20	Rotate Left A	RLA	N	
21	Rotate Left B	RLB	N	
22	Rotate Left Long	RLL	N	
23	Decrement A	DCA		ØV
24	Logical Right A	LRA	N	
25	Logical Right B	LRB	N	
26	Logical Right Long	LRL	N	
27	Decrement B	DCB		ØV
28	Arithmetic Left A	ALA	N	
29	Arithmetic Left B	ALB	N	
2A	Arithmetic Left Long	ALL	N	
2B	Transfer A to B	TAB		
2C	Arithmetic Right A	ARA	N	
2D	Arithmetic Right B	ARB	N	
2E	Arithmetic Right Long	ARL	N	

OPCODE	INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS
2F	Transfer B to A	TBA		
30†	Input Byte Serially	IBS		
31†	Input Byte to A	IBA	DØ,DEV	
32†	Input Byte to B	IBB	DØ,DEV	
33†	Input Byte to Memory	IBM	DØ,DEV ADDR(X)	
34	No Operation	NØP		
35	Interchange A & X	IAX		
36	Interchange B & X	IBX		
38†	Output Byte Serially	ØBS		
39†	Output Byte From A	ØBA	DØ,DEV	
3A†	Output Byte From B	ØBB	DØ,DEV	
3B†	Output Byte From Memory	ØBM	DØ,DEV ADDR(X)	
40	ØR B to A	ØRA		
41	Exclusive ØR B to A	XRA		
42	ØR A to B	ØRB		
43	Exclusive ØR A to B	XRB		
44	Increment X	INX		ØV
45	Decrement X	DCX		ØV
46	Add Word Length to X	AWX		ØV
47	Subtract Word Length from X	SWX		ØV
48	Increment A	INA		ØV
49	Increment B	INB		ØV
4A	Ones Complement A	ØCA		
4B	Ones Complement B	ØCB		
4C	Transfer A to X	TAX		
4D	Transfer B to X	TBX		
4E	Transfer X to A	TXA		
4F	Transfer X to B	TXB		
50	Add to Word Immediate	AWI	I,ADDR(X)	ØV,ALI
51	Branch On Condition	BØC	N,ADDR(X)	
52	Move Immediate	MVI	N,ADDR(X)	

OPCODE	INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS
53	Compare Logical Immediate	CLI	N,ADDR(X)	ALI
54	Test Under Mask Immediate	TMI	N,ADDR(X)	ALI
55	Set Bits Under Mask Immediate	SMI	N,ADDR(X)	ALI
56	Clear Bits Under Mask Immediate	CMI	N,ADDR(X)	ALI
57	Invert Bits Under Mask Immediate	IMI	N,ADDR(X)	ALI
58	Add Decimal	ADD	$ADDR_T(L_T,X)$, $ADDR_S(L_S,X)$	ØV,ALI
59	Subtract Decimal	SBD	$ADDR_T(L_T,X)$, $ADDR_S(L_S,X)$	ØV,ALI
5A	Multiply Step Decimal	MSD	$ADDR_T(L_T,X)$ $ADDR_S(L_S,X)$	ØV
5B	Divide Step Decimal	DSD	$ADDR_T(L_T,X)$, $ADDR_S(L_S,X)$	ØV
5C	Move Character String Left	MVL	$ADDR_T(L,X)$, $ADDR_S(X)$	
5D	Move Character String Right	MVR	$ADDR_T(L,X)$, $ADDR_S(X)$	
5E	RØM Exit	XIT		
5F00†	Save Machine State	SAV		
5F01†	Return	RET		all
5F02	Add to X Immediate	AXI	I	ØV
5F03†	Return Displaced	RTN	I	all
5F04	Edit and Mark	EDT	$ADDR_T(L,X)$, $ADDR_S(X)$	ALI
5F05	Compare Logical Character	CLC	$ADDR_T(L,X)$, $ADDR_S(X)$	ALI
5F06	Translate Under Mask	TRM	N,$ADDR_T(L,X)$ $ADDR_S(X)$	
5F07	Translate and Test Under Mask	TTM	N,$ADDR_T(L,X)$, $ADDR_S(X)$	ALI
5F08-5FFF	Secondary ROM Exit	XT2	N	
60-67*†	Jump	JMP	ADDR(X)	
68-6F*†	Return Jump	RTJ	ADDR(X)	
70-77*	Increment Word In Memory	IWM	ADDR(X)	ØV,ALI
80-87*	Load X	LDX	ADDR(X)	
88-8F*	Store X	STX	ADDR(X)	
90-97*	Multiply Step	MST	ADDR(X)	
98-9F*	Divide Step	DST	ADDR(X)	
A0-A7*	Add to A	ADA	ADDR(X)	ØV

OPCODE	INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS
A8-AF*	Add Variable	ADV	ADDR(X)	ØV
B0-B7*	Subtract From A	SBA	ADDR(X)	ØV
B8-BF*	Subtract Variable	SBV	ADDR(X)	ØV
C0-C7*	Load B	LDB	ADDR(X)	
C8-CF*	Store B	STB	ADDR(X)	
D0-D7*	AND Memory To A	ANA	ADDR(X)	
D8-DF*	AND Variable	ANV	ADDR(X)	
E0-E7*	Load A	LDA	ADDR(X)	
E8-EF*	Load Variable	LDV	ADDR(X)	
F0-F7*	Store A	STA	ADDR(X)	
F8-FF*	Store Variable	STV	ADDR(X)	

APPENDIX B

INSTRUCTION SET LISTED

ALPHABETICALLY BY INSTRUCTION

The following terms are used in Appendixes A and B to describe valid assembly language operands for the various instructions.

ADDR	Operand expression used to calculate an address
I	An absolute expression used to specify data which may be negative.
N	An absolute expression used to specify data which may not be negative, such as counts, masks, etc.
L	An absolute expression used to specify explicit length value.

Instructions marked * have all addressing modes.
Instructions marked † have interrupt umbrella.

INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Add Decimal	ADD	$ADDR_T(L_T,X)$, $ADDR_S(L_S,X)$	ØV,ALI	58
Add to A	ADA	ADDR(X)	ØV	A0-A7*
Add to X Immediate	AXI	I	ØV	5F02
Add to Word Immediate	AWI	I,ADDR(X)	ØV,ALI	50
Add Variable	ADV	ADDR(X)	ØV	A8-AF*
Add Word Length to X	AWX		ØV	46
AND Memory To A	ANA	ADDR(X)		D0-D7*
AND Variable	ANV	ADDR(X)		D8-DF*
Arithmetic Left A	ALA	N		28
Arithmetic Left B	ALB	N		29
Arithmetic Left Long	ALL	N		2A
Arithmetic Right A	ARA	N		2C
Arithmetic Right B	ARB	N		2D
Arithmetic Right Long	ARL	N		2E

INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Branch On Condition	BØC	N,ADDR(X)		51
Clear Bits Under Mask Immediate	CMI	N,ADDR(X)	ALI	56
Compare Logical Character	CLC	$ADDR_T(L,X)$, $ADDR_S(X)$	ALI	5F05
Compare Logical Immediate	CLI	N,ADDR(X)	ALI	53
Decrement A	DCA		ØV	23
Decrement B	DCB		ØV	27
Decrement X	DCX		ØV	45
Decrement Word in Memory	DWM	ADDR(X)	ØV,ALI	78-7F*
Disable Interrupt System	DIN		INT	04†
Disable Interval Timer	DIT			06†
Divide Step Decimal	DSD	$ADDR_T(L_T,X)$, $ADDR_S(L_S,X)$	ØV	5B
Divide Step	DST	ADDR(X)		98-9F*
Edit and Mark	EDT	$ADDR_T(L,X)$, $ADDR_S(X)$	ALI	5F04
Enable Interrupt System	EIN		INT	05†
Enable Interval Timer	EIT			07†
Enter Sense Switches	ESW			02
Exclusive ØR A to B	XRB			43
Exclusive ØR B to A	XRA			41
Halt	HLT			00†
Increment A	INA		ØV	48
Increment B	INB		ØV	49
Increment X	INX		ØV	44
Increment Word In Memory	IWM	ADDR(X)	ØV,ALI	70-77*
Input Byte to A	IBA	DØ,DEV		31†
Input Byte to B	IBB	DØ,DEV		32†
Input Byte To Memory	IBM	DØ,DEV, ADDR(X)		33†
Input Byte Serially	IBS			30†

INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Interchange A & B	IAB			03
Interchange A & X	IAX			35
Interchange B & X	IBX			36
Invert Bits Under Mask Immediate	IMI	N,ADDR(X)	ALI	57
Jump	JMP	ADDR(X)		60-67*†
Load A	LDA	ADDR(X)		E0-E7*
Load B	LDB	ADDR(X)		C0-C7*
Load Variable	LDV	ADDR(X)		E8-EF*
Load X	LDX	ADDR(X)		80-87*
Logical Right A	LRA	N		24
Logical Right B	LRB	N		25
Logical Right Long	LRL	N		26
Move Character String Left	MVL	$ADDR_T(L,X)$, $ADDR_S(X)$		5C
Move Character String Right	MVR	$ADDR_T(L,X)$, $ADDR_S(X)$		5D
Move Immediate	MVI	N,ADDR(X)		52
Multiply Step	MST	ADDR(X)		90-97*
Multiply Step Decimal	MSD	$ADDR_T(L_T,X)$, $ADDR_S(L_S,X)$		5A
No Operation	NØP			34
1's Complement A	ØCA			4A
1's Complement B	ØCB			4B
ØR B to A	ØRA			40
ØR A to B	ØRB			42
Output Byte From A	ØBA	DØ,DEV		39†
Output Byte From B	ØBB	DØ,DEV		3A†
Output Byte From Memory	ØBM	DØ,DEV, ADDR(X)		3B†
Output Byte Serially	ØBS			38†
Reset ØV, Set WL=1	RØ1		ØV,WL	08
Reset ØV, Set WL=2	RØ2		ØV,WL	09
Reset ØV, Set WL=3	RØ3		ØV,WL	0A
Reset ØV, Set WL=4	RØ4		ØV,WL	0B
Return	RET		all	5F01†
Return Jump	RTJ	ADDR(X)		68-6F*†
Return Displaced	RTN	I	all	5F03†
ROM Exit	XIT			5E
Rotate Left A	RLA	N		20

INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Rotate Left B	RLB	N		21
Rotate Left Long	RLL	N		22
Save Machine State	SAV			5F00†
Secondary ROM Exit	XT2	N		5F08-5FFF
Set Bits Under Mask Immediate	SMI	N,ADDR(X)	ALI	55
Set ØV, Set WL=1	SØ1		ØV,WL	0C
Set ØV, Set WL=2	SØ2		ØV,WL	0D
Set ØV, Set WL=3	SØ3		ØV,WL	0E
Set ØV, Set WL=4	SØ4		ØV,WL	0F
Skip if A=B	SAB	ADDR		16
Skip if A≠B	NAB	ADDR		1E
Skip if A Negative	SAN	ADDR		14
Skip if A Not Negative	NAN	ADDR		1C
Skip if A=0	SAZ	ADDR		11
Skip if A≠0	NAZ	ADDR		19
Skip if A=X	SAX	ADDR		17
Skip if A≠X	NAX	ADDR		1F
Skip if B=0	SBZ	ADDR		12
Skip if B≠0	NBZ	ADDR		1A
Skip if Overflow Set	SØV	ADDR	ØV	10
Skip if Overflow Not Set	NØV	ADDR	ØV	18
Skip if X=0	SXZ	ADDR		13
Skip if X≠0	NXZ	ADDR		1B
Skip if X Negative	SXN	ADDR		15
Skip if X not Negative	NXN	ADDR		1D
Store A	STA	ADDR(X)		F0-F7*
Store B	STB	ADDR(X)		C8-CF*
Store Variable	STV	ADDR(X)		F8-FF*
Store X	STX	ADDR(X)		88-8F*
Subtract Decimal	SBD	$ADDR_T(L_T,X)$, $ADDR_S(L_S,X)$	ØV,ALI	59
Subtract from A	SBA	ADDR(X)	ØV	B0-B7*
Subtract Variable	SBV	ADDR(X)	ØV	B8-BF*
Subtract Word Length From X	SWX		ØV	47
Test Under Mask Immediate	TMI	N,ADDR(X)	ALI	54

INSTRUCTION	MNEMONIC	OPERANDS	INDICATORS	OPCODE
Transfer A to B	TAB			2B
Transfer A to X	TAX			4C
Transfer B to A	TBA			2F
Transfer B to X	TBX			4D
Transfer X to A	TXA			4E
Transfer X to B	TXB			4F
Translate Under Mask	TRM	N,$ADDR_T$(L,X), $ADDR_S$(X)		5F06
Translate and Test Under Mask	TTM	N,$ADDR_T$(L,X), $ADDR_S$(X)	ALI	5F07
Trap	TRP			01†

APPENDIX C

CIP/2200 Instruction Execution Times

INSTRUCTION	MNEMONIC	EXECUTION TIME (MICROSECONDS)	NOTES
Add Decimal	ADD	84.5+7.5 Per Digit	
Add to A	ADA	11.7	1,2,4
Add to X Immediate	AXI	35.2	2
Add to Word Immediate	AWI	17.8	2
Add Variable	ADV	13.4	1,2,3,4
Add Word Length to X	AWX	7.0	2
AND Memory to A	ANA	12.1	1,4
AND Variable	ANV	13.9	1,3,4
Arithmetic Left A	ALA	5.9+3.5 Per Bit Position Shifted	
Arithmetic Left B	ALB	5.9+3.5 Per Bit Position Shifted	
Arithmetic Left Long	ALL	5.9+3.7 Per Bit Position Shifted	
Arithmetic Right A	ARA	5.9+3.3 Per Bit Position Shifted	
Arithmetic Right B	ARB	5.9+3.3 Per Bit Position Shifted	
Arithmetic Right Long	ARL	5.9+4.0 Per Bit Position Shifted	
Branch On Condition	BØC	11.4 No Jump 12.8 Jump	4
Clear Bits Under Mask Immediate	CMI	15.8	4
Compare Logical Character	CLC	55.4+8.6 For Each Character Compared	4,5
Compare Logical Immediate	CLI	15.0	2,4
Decrement A	DCA	6.2	2
Decrement B	DCB	6.2	2
Decrement X	DCX	7.0	2
Decrement Word In Memory	DWM	12.5	1,2,4
Disable Interrupt System	DIN	5.9	
Disable Interval Timer	DIT	4.8	

INSTRUCTION	MNEMONIC	EXECUTION TIME (MICROSECONDS)	NOTES
Divide Step Decimal	DSD	102.5+80 Per Divident Digit-Average 102.5+126 Per Dividend Digit-Maximum	4,6
Divide Step	DST	14.5	
Edit and Mark	EDT	77.9+11.9 For Each D.S. or S.S. Character +5.3 For Each Text Character +5.7 For Each F.S. Character	
Enable Interrupt System	EIN	5.9	
Enable Interval Timer	EIT	4.4	
Enter Sense Switches	ESW	4.8	
Exclusive Or A to B	XRB	6.6	
Exclusive Or B to A	XRA	6.4	
Halt	HLT	5.7	
Increment A	INA	7.0	2
Increment B	INB	7.0	2
Increment X	INX	7.0	2
Increment Word In Memory	IWM	12.5	1,2,4
Input Byte to A	IBA	8.4	
Input Byte to B	IBB	8.8	
Input Byte to Memory	IBM	14.3	4
Input Byte Serially	IBS		8
Interchange A & B	IAB	11.6	
Interchange A & X	IAX	11.0	
Interchange B & X	IBX	11.0	
Invert Bits Under Mask Immediate	IMI	15.6	
Jump	JMP	10.1	1,4
Load A	LDA	12.1	1,4
Load B	LDB	14.4	1,4
Load Variable	LDV	13.9	1,3,4
Load X	LDX	12.5	1,4
Logical Right A	LRA	5.9+3.3 Per BIt Position Shifted	
Logical Right B	LRB	5.9+3.3 Per Bit Position	
Logical Right Long	LRL	5.9+4.0 Per Bit Position Shifted	

INSTRUCTION	MNEMONIC	EXECUTION TIME (MICROSECONDS)	NOTES
Move Character String Left	MVL	67.7+5.5 Per Character Moved	
Move Character String Right	MVR	75.9+5.3 Per Character Moved	
Move Immediate	MVI	13.0	
Multiply Step	MST	20.5	
Multiply Step Decimal	MSD	62.5+14.7 Per Digit - Average +22.0 Per Digit - Maximum	
No Operation	NØP	4.4	
1's Complement A	ØCA	6.6	
1's Complement B	ØCB	6.6	
Or B to A	ØRA	6.4	
Or A to B	ØRB	6.4	
Output Byte From A	ØBA	8.4	
Output Byte From B	ØBB	9.2	
Output Byte From Memory	ØBM	14.5	
Output Byte Serially	ØBS	100.0 Milliseconds	
Reset ØV, Set WL=1	RØ1	5.3	
Reset ØV, Set WL=2	RØ2	5.3	
Reset ØV, Set WL=3	RØ3	5.3	
Reset ØV, Set WL=4	RØ4	5.3	
Return	RET	69.0	
Return Jump	RTJ	13.0	1,4
Return Displaced	RTN	69.3	
ROM Exit	XIT	4.8	9
Rotate Left A	RLA	5.9+3.5 Per Bit Position Shifted	
Rotate Left B	RLB	5.9+3.5 Per Bit Position Shifted	
Rotate Left Long	RLL	5.9+3.7 Per Bit Position Shifted	
Save Machine State	SAV	70.0	
Secondary ROM Exit	XT2	31.5	9
Set Bits Under Mask Immediate	SMI	15.6	

INSTRUCTION	MNEMONIC	EXECUTION TIME (MICROSECONDS)	NOTES
Set ØV, Set WL=1	SØ1	5.3	
Set ØV, Set WL=2	SØ2	5.3	
Set ØV, Set WL=3	SØ3	5.3	
Set ØV, Set WL=4	SØ4	5.3	
Skip if A=B	SAB	8.4	7
Skip if A≠B	NAB	8.4	7
Skip if A Negative	SAN	7.5	7
Skip if A Not Negative	NAN	7.5	7
Skip if A=0	SAZ	7.7	7
Skip if A≠0	NAZ	7.7	7
Skip if A=X	SAX	8.1	7
Skip if A≠X	NAX	8.1	7
Skip if B=0	SBZ	7.5	7
Skip if B≠0	NBZ	7.5	7
Skip if Overflow Set	SØV	6.8	7
Skip if Overflow Not Set	NØV	7.7	7
Skip if X=0	SXZ	7.3	7
Skip if X≠O	NXZ	7.3	7
Skip if X Negative	SXN	7.3	7
Skip if X Not Negative	NXN	7.3	7
Store A	STA	11.2	1,4
Store B	STB	13.0	1,4
Store Variable	STV	10.3	1,3,4
Store X	STX	12.5	1,4
Subtract Decimal	SBD	84.5+7.5 Per Digit	
Subtract From A	SBA	12.1	1,2,4
Subtract Variable	SBV	13.9	1,2,3,4
Subtract Word Length From X	SWX	7.0	2
Test Under Mask Immediate	TMI	16.5	
Transfer A to B	TAB	5.7	
Transfer A to X	TAX	7.0	
Transfer B to A	TBA	5.9	
Transfer B to X	TBX	7.0	
Transfer X to A	TXA	7.3	
Transfer X to B	TXB	7.3	
Translate Under Mask	TRM	72.8+20.0 Per Character	
Translate and Test Under Mask	TTM	79.1+21.1 Per Character Translated	
Trap	TRP	15.8	

NOTES

1. The time shown is for the extended addressing mode (mode 6). For other modes, the execution time is obtained by adding the adjustment factor from the table below.

MODE	NAME	ADJUSTMENT (MICROSECONDS)	
0	Direct Page 0	-1.1	
1	Direct Relative	0.0	Pos. Displacement
		+0.7	Neg. Displacement
2	Indirect Page 0	+2.2	
3	Indirect Relative	+3.3	Pos. Displacement
		+4.0	Neg. Displacement
4	Base Addressing	-1.6	
5	Base + Displacement	-0.6	
6	Extended	0.0	
7	Literal		
	a) Fixed Length	+1.3	
	b) Variable Length	+1.3	
	c) With A Register	+1.8	
	d) JMP & RTJ	+4.2	

2. If overflow occurs, add .7.

3. Time shown for variable word length instructions is for word length=1. For other word length values add the adjustment factor from the following table.

WORD LENGTH	CORRECTION
2 SBV	+1.3
2 all others	-0.4
3 SBV	+5.1
3 all others	+2.6
4 SBV	+6.4
4 all others	+2.2

4. Instructions having indexed address words require an additional 1.3 microseconds.

5. The CLC instruction compares corresponding source and target characters until a difference is detected. The ALI are then set and the instruction terminates.

6. Execution time for the DSD instruction is a function of the number of dividend digits and the quotient value produced. The exact relation is:

$$\text{DSD execution time} = 45.3+11.44\ [Q+N_d(Q+2)]$$

where Q = quotient digit value
N_d = number of dividend digits

7. The times given for conditional skip instructions are for execution without skipping. If the skip is taken, add the time shown in the following table.

INSTRUCTION	ADJUSTMENT FOR SKIP PØS.	NEG.
SØV	1.8	2.0
NØV	0	.2
All Others	.9	1.1

8. Execution of the IBS instruction terminates when a byte is transferred from the serial I/O device. The minimum time is 86 milliseconds.

9. Time given for the ROM exit instructions is the time required to transfer control to the first unused ROM page.

APPENDIX D

PROGRAMMING FOR POWER FAIL/AUTOMATIC RESTART OPTION

The power fail/automatic restart option monitors the computer's external power source and interrupts processing whenever a significant change is detected. There are two system interrupts reserved for the power fail/automatic restart facility. When the external power supply voltage falls below the minimum level required for dependable operation, a "power fail" interrupt occurs. When power is restored, the "power restart" interrupt is generated. The corresponding interrupt service routines are used for programmed control of power on and power off procedures.

The primary purpose of the power fail automatic restart option is to protect the contents of main memory from being destroyed by the loss of power. This function is automatically performed by the hardware and is transparent to the programmer. In general, the computer operation in progress at the time of a power failure cannot be continued after power is restored because I/O transfers are abnormally terminated by the loss of power. The power failure ISR is responsible for bringing the computer system to a safe halt. The power restart ISR is responsible for reinitializing the system and achieving the normal operating state once again. In some cases it may be possible to program the power fail/restart ISR's so that the loss of power becomes transparent to the computer user.

Power Failure ISR

The power fail interrupt occurs upon detection of the loss of primary power. A period of 2 milliseconds of reliable computer operation remains after the interrupt occurs. The power fail ISR must therefore perform all operations necessary to bring the system to an orderly halt within a short period of time. Once the ISR is activated, no more interrupts are recognized. DMC transfers will continue. If it is desirable to save the contents of the machine registers, they must be stored in memory. The power fail ISR should end with a HLT instruction to prevent loss of memory contents.

Power Restart ISR

The power restart interrupt occurs immediately after power is restored to the system. The purpose of the power restart ISR is to re-initialize the computer so that normal operation can be resumed. If necessary, the operator should be notified that a power failure occurred so that he can take the appropriate action.

Systems having the blank front panel must have the power fail/automatic restart option installed. Without this facility there is no method for initializing the system on start-up, or protecting it during shut-down. Serial I/O should not be used in systems which require immediate action in response to the power fail interrupt, because interrupts are not recognized during serial transfers which take a minimum of 100 milliseconds.

APPENDIX E

DMA CHANNEL PROGRAMMING

Introduction

The CIP/2200 Computer optionally may have one or two DMA channels. A CIP/2200 program can start or stop the DMA channel, read status information, or be interrupted by the channel. Once started by the CPU, the DMA controller transfers data to or from memory independently of the central processing unit, competing with the CPU on a priority basis for access to main memory.

The DMA channel can be programmed to transfer data to or from one to four data buffers. There are two basic modes of operation: cyclic mode or single block mode of transfer. Cyclic mode transfer is a continuous data transfer operation which is terminated by program command. Single block mode transfer is a non-continuous data transfer operation which is terminated normally by the DMA channel, itself.

Buffer Control Words

The buffer control words define the location and amount of data to be transferred during a DMA operation. There are four buffer control word pairs for each channel stored in dedicated page 0 locations as shown in Figure 1.

The DMA channel maintains a four position counter to indicate which buffer is currently being processed. The buffer counter may be reset and mal also be incremented automatically after a buffer is processed.

Each buffer control word pair consists of a starting address and an ending address. The starting address word specifies the location of the first byte in the data buffer; the ending address word specifies the location of the last data byte. Address values, which may be between 0 and 32767, are stored in the low 15 bits of each word. The most significant bit of each buffer control word contains a flag which further controls DMA channel action.

Bit 15 of the starting address word of each buffer control word pair is a link flag used for addressing up to four buffers. The link flag is tested when processing of the data buffer is finished. If the link flag is set, then the buffer counter is incremented by one to address the next set of buffer control words. If the link flag is reset, then the buffer counter is reset to address the first set of buffer control words. Automatic wrap-around from buffer 4 to buffer 1 is provided if the link flag is set on the fourth group of buffer control words.

The INPUT STATUS command performs two functions. The first is to reset the DMA controller interrupt request. The DMA interrupt service routine must input status before executing any instruction which does not have an interrupt umbrella. Failure to do so may result in either an endless loop or a control stack overflow interrupt if a SAV instruction follows the ISR entry point.

The input status command also transfers eight bits of status information from the DMA controller to a dedicated page zero memory location. The DMA status byte format is shown in Figure 3. Bit 0, the channel busy bit, may be tested to determine if a DMA transfer is in process. Bits 1 thru 6 contain device dependent status information. Definition of these bits may be found in individual device controller product performance specifications. The zero state for bit 7 indicates that an end of buffer interrupt has been generated.

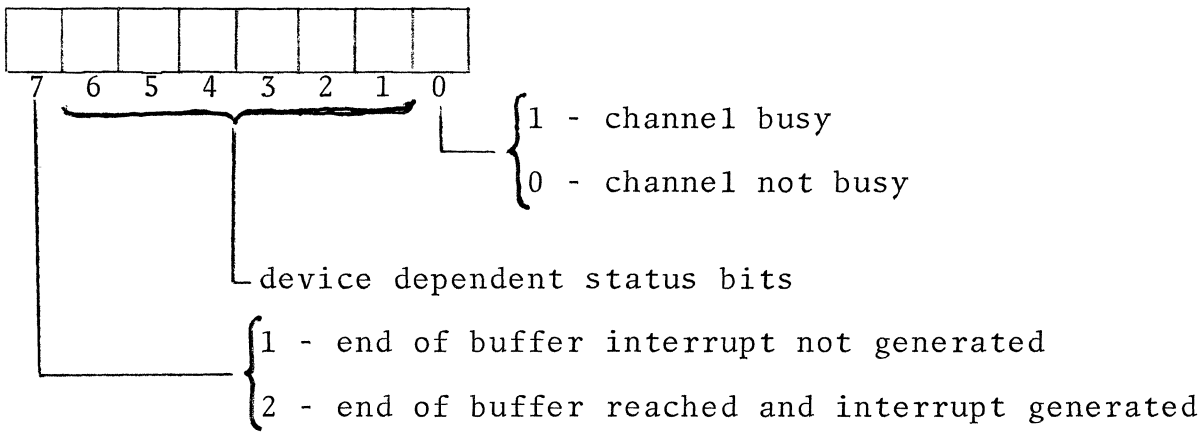


Figure 3. DMA Status Byte

The DMA interrupt service routine for systems with two DMA controllers must check bit 7 of both status bytes to determine which channel interrupted.

START CHANNEL - CYCLIC MODE, device order 1, starts the channel operation. Before this command is issued, the program must initialize the buffer control words. The start channel command causes the DMA channel to read the current buffer address control words and start operation. When transferring in the cyclic mode, the channel may move more than one data buffer as determined by the link flags in the starting address control words. This command must be preceeded with a RESET command if the last operation occurred in the single buffer mode or was terminated by a STOP CHANNEL AT BUFFER END command.

STOP CHANNEL AT BUFFER END causes the channel to stop at the end of the current data buffer regardless of the condition of the link flags.

START CHANNEL - SINGLE BUFFER MODE performs the same function as device order 1 but only current buffer is transferred. However, if the link flag was set, then the next time this command is given the DMA channel will fetch the next set of buffer control words, unless the buffer counter has been reset with the RESET CHANNEL command.

RESET CHANNEL forces an immediate halt to the DMA operation in progress. The buffer counter is reset.

APPENDIX F

CIP/2200 FIRMWARE EXTENSIONS

The CIP/2200 has been designed to allow the addition of custom microprogramming to the basic computer. Custom firmware may be useful for extending the standard instruction set to include application dependent software instructions or I/O operations. Special microcode can also be added to perform processing functions in response to front panel commands.

TRANSFER OF CONTROL TO CUSTOM FIRMWARE

There are two methods available for transferring control from the standard microprogram to custom firmware. The first is programmed; the second is initiated from the front panel. Programmed transfer of control provides the means for implementing instruction set extensions. When a ROM exit opcode is detected by the standard firmware, the CIP/2200 executes a microprogram jump to a dedicated location in ROM page 6.

The primary exit instruction, "XIT", is a one byte instruction which transfers control to the first word location in ROM page 6, 600_{16}. The secondary exit instruction, "XT2", is a two byte instruction which transfers control to the second word location on ROM page 6, 601_{16}. The second byte of the XT2 instruction contains a value between 8_{16} and FF_{16} which can be interpreted to provide up to 248 secondary opcodes. When the custom microprogram receives control, file registers C_{16} and D_{16} contain the low and high bytes, respectively, of the address of the last byte of the ROM exit instruction. If the secondary ROM exit instruction is used, file register 1 will contain the value of the second byte of the instruction. File registers 2 to 7 and C_{16} and D_{16} are stored in the system save area as shown in the following table.

$Location_{16}$	$File\ Register_{16}$
180	D - P_{15-8}
181	C - P_{7-0}
182	7 - B_{15-8}
183	6 - B_{7-0}
184	5 - A_{15-8}
185	4 - A_{7-0}
186	3 - X_{15-8}
187	2 - X_{7-0}

The elapsed time between the start of the exit instruction and the transfer of control to custom ROM is 4.8 microseconds for XIT and 32.3 microseconds for XT2.

Operator initiated transfer of control is performed by turning sense switches three and four on, followed by a "reset-run" sequence. The operator controlled transfer causes a micro-program jump to the third word on page 6, 602_{16}. This transfer method is useful for initiating special functions on command from the front panel, e.g. a special initial program load microprogram.

Writing Custom Firmware

Custom extensions to the CIP/2200 are placed on ROM pages 6 and 7, allowing a maximum of 512 words of custom code. When the appropriate method for transfer of control has been selected, the corresponding dedicated page 6 location (600_{16}, 601_{16} or 602_{16}) must be initialized with a jump to the custom microprogram. All microprogramming is done in the standard fashion as documented in the CIP/2000 literature.

Custom firmware extensions may terminate by returning to the CIP/2200 microprogram for continuation of normal software instruction processing. When control is returned to the standard firmware, file registers C_{16} and D_{16} must contain the low and high

bytes, respectively, of the address of the last byte of the special instruction. The added instruction has the option of checking for interrupts before fetching the next software instruction. If interrupts are to be recognized, the custom firmware should return to the CIP/2200 by a jump to location $00E_{16}$. An interrupt umbrella is obtained by jumping to location 009_{16}. In the latter case, the address in file registers C_{16} and D_{16} must be the address of the first byte of the next CIP/2200 instruction.

CIP/2000 control storage is partitioned into 1024_{10} word segments referred to as memory banks. The normal jump microinstruction may be used to jump anywhere within a single memory bank. Jumps between memory banks require a two instruction sequence consisting of a jump instruction followed by a bank switching command. The jump instruction is written in the normal fashion, causing the target address to be evaluated modulo 1024_{10}. The bank switching command is a literal class instruction whose opcode mnemonic is "LZ". The literal field specifies the memory bank to which control will be transferred. The literal values are assigned as shown.

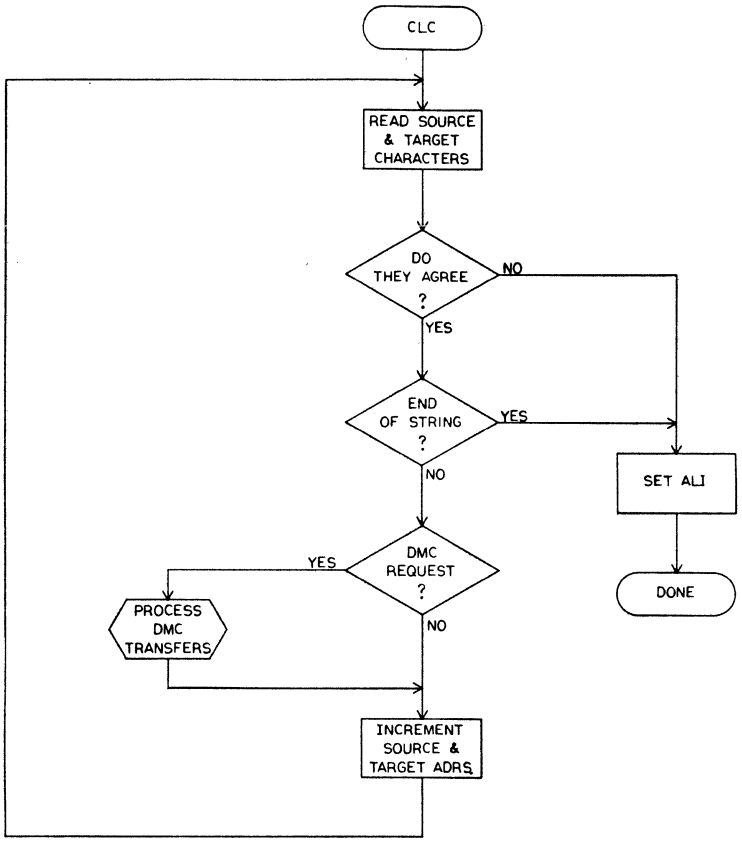
Memory Bank	Addresses	Literal
0	$000-3FF_{16}$	11_{16}
1	$400-7FF_{16}$	12_{16}

Custom firmware may also be added to the CIP/2210 by using the procedures described in this appendix with the following changes. The CIP/2210 leaves 256 words of ROM available for special microprogramming. The XIT and XT2 instructions transfer control to locations 700_{16} and 701_{16}, respectively. Operator initiated transfer of control is reserved for the disk IPL functio

APPENDIX G

Instruction Flowcharts

COMPARE LOGICAL CHARACTER INSTRUCTION FLOWCHART



D

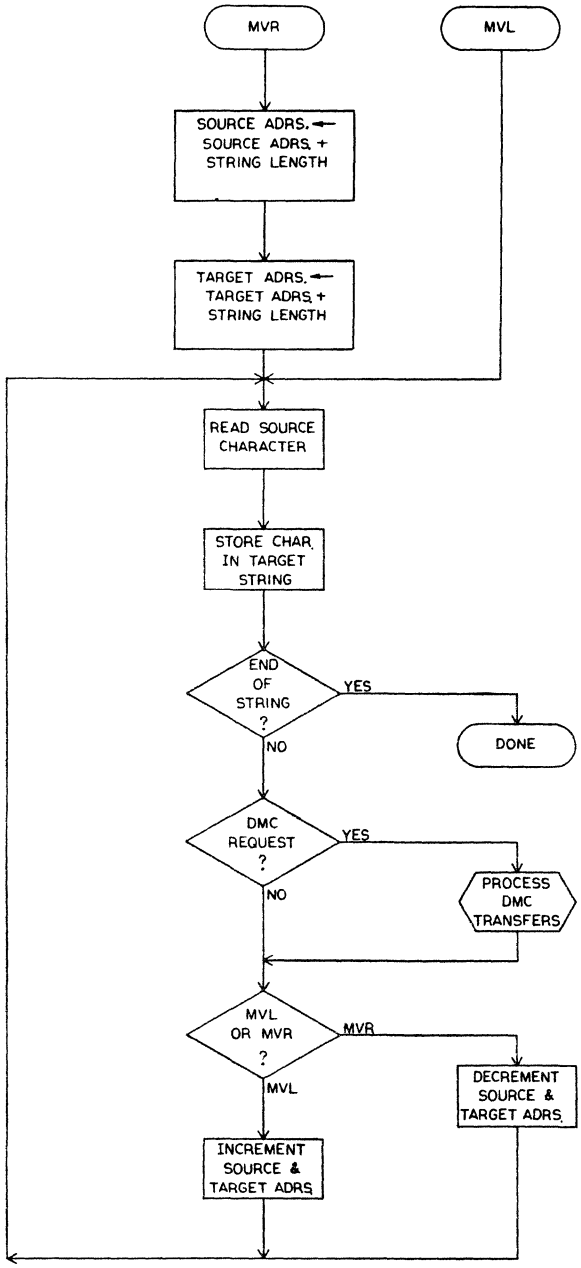
COMPARE LOGICAL CHAR. INSTR.

CIP/2200 REF.

7 000 0076 F

SHEET 1 OF 1

MOVE LEFT & MOVE RIGHT INSTRUCTION FLOWCHART



D

PRODUCT CIP/2200 REF

TITLE MOVE LEFT & MOVE RIGHT INSTR

DRAWN

CHECKED

ENGINEER

DATE

APPROVED

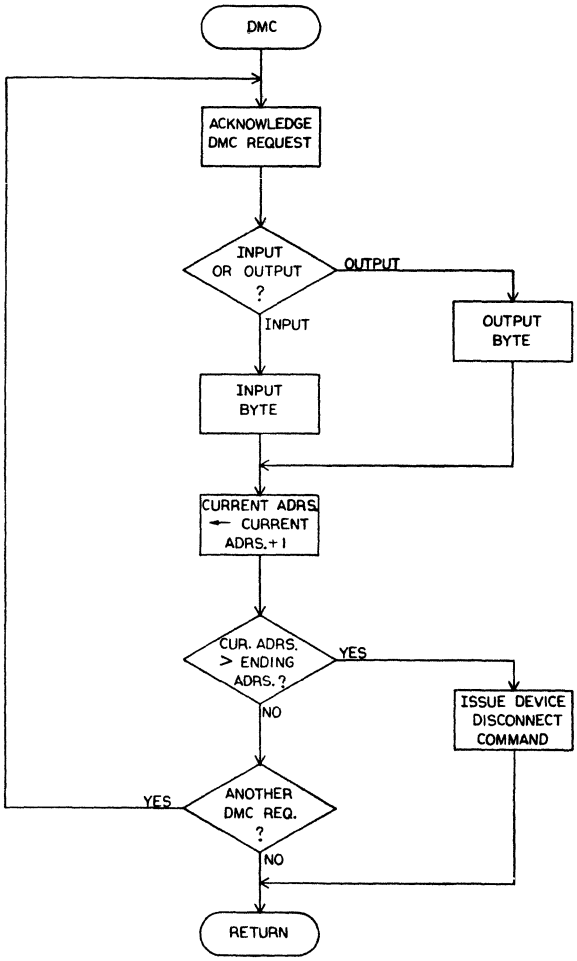
DRAWING NO. 7 000 0076 F

SHEET 1 OF 1

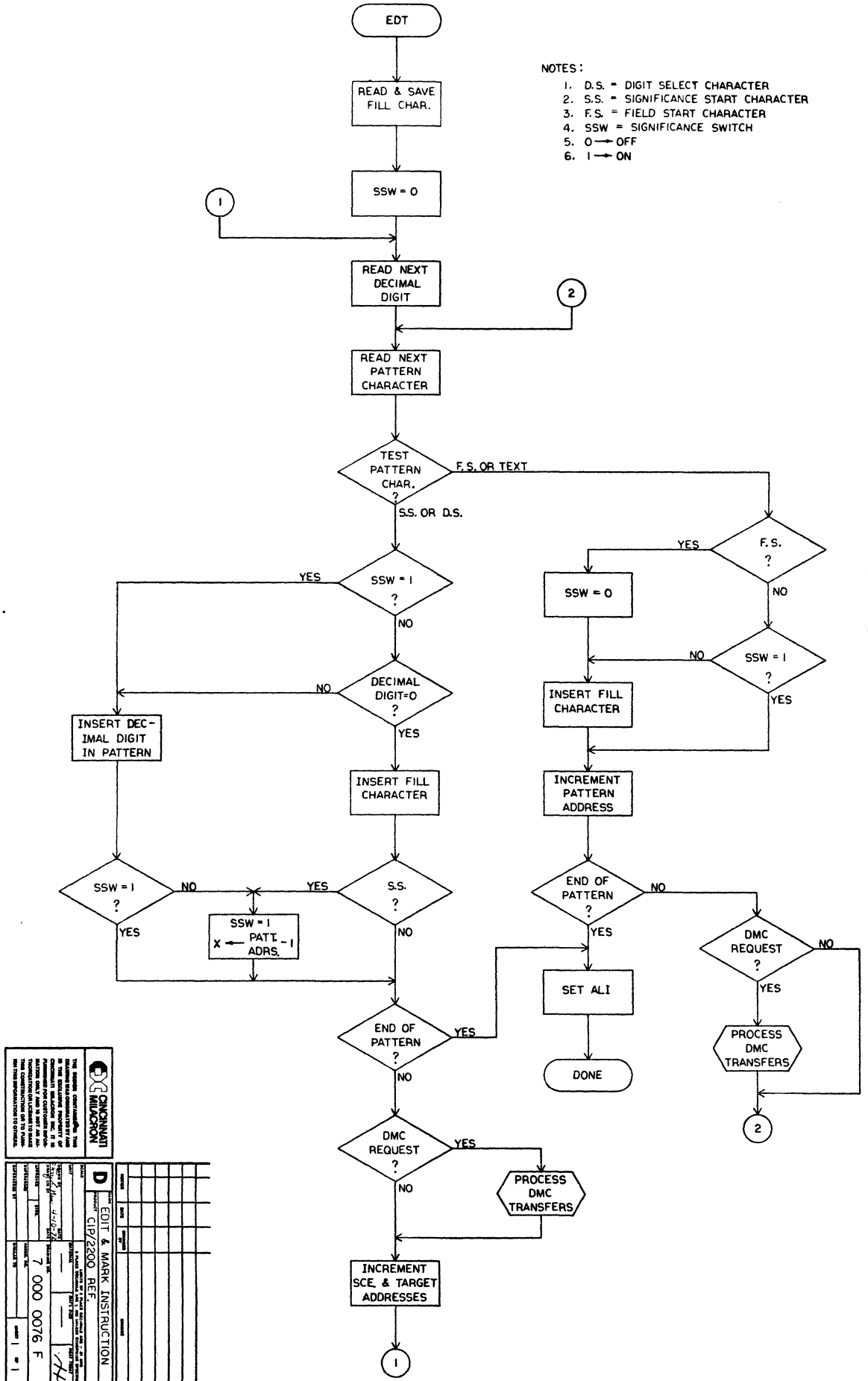
SIZE

REVISIONS

DMC TRANSFER FLOWCHART



EDIT AND MARK INSTRUCTION FLOWCHART



NOTES:
1. D.S. = DIGIT SELECT CHARACTER
2. S.S. = SIGNIFICANCE START CHARACTER
3. F.S. = FIELD START CHARACTER
4. SSW = SIGNIFICANCE SWITCH
5. 0 → OFF
6. 1 → ON

CINCINNATI MILACRON

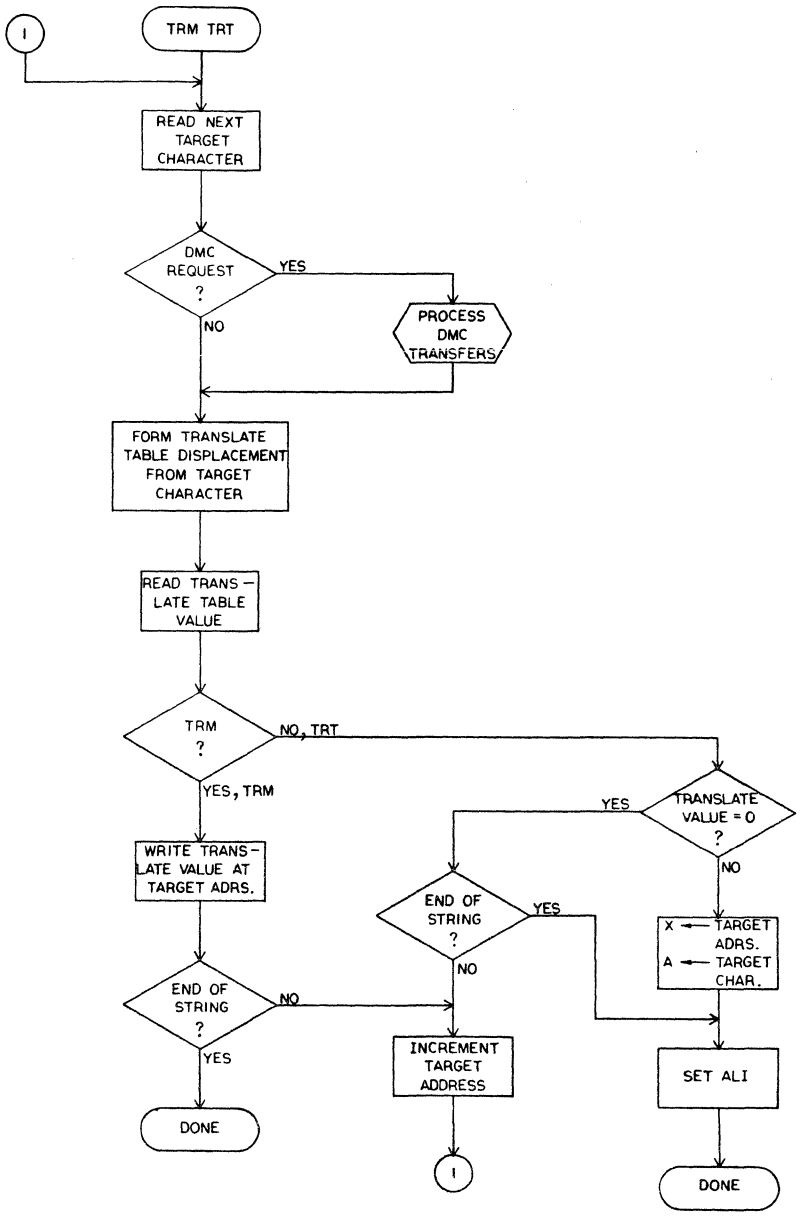
D

EDIT & MARK INSTRUCTION

CIP/2200 REF.

7 000 0076 F

TRANSLATE & TRANSLATE AND TEST UNDER MASK INSTRUCTION FLOWCHART



APPENDIX H

DEDICATED MEMORY LOCATIONS

DMC ADDRESS ASSIGNMENTS:

000-001	Device 0 Current Address
002-003	Device 0 Ending Address
004-005	Device 1 Current Address
006-007	Device 1 Ending Address
-	- - -
07C-07D	Device 31 Current Address
07E-07F	Device 31 Ending Address

DMA ADDRESS ASSIGNMENTS:

58	DMA1 Status
5C	DMA2 Status
60-61	DMA1 Buffer 1 Start Address
62-63	DMA1 Buffer 1 End Address
64-65	DMA1 Buffer 2 Start Address
-	- - -
6E-6F	DMA1 Buffer 4 End Address
70-71	DMA2 Buffer 1 Start Address
-	- - -
7E-7F	DMA2 Buffer 4 End Address

INTERRUPT TRANSFER LOCATIONS & RESERVED SYSTEM AREAS:

080-081	Console Interrupt
082-083	DMA Interrupt
084-085	Interval Timer Counter
086-087	Interval Timer Interrupt
088-089	Reserved for Future Use
08A-08B	Memory Parity Interrupt
08C-08D	Stack Overflow/Underflow Interrupt
08E-08F	Power Fail Interrupt
090-091	Power Restart Interrupt
092-093	Control Stack Pointer
094-0FF	Free

EXTERNAL INTERRUPTS:

100-101	External Interrupt 0
102-103	External Interrupt 1
-	- - -
17E-17F	External Interrupt 63

SYSTEM SAVE AREA:

180-18F	Reserved for CIP/2200 Firmware. Not available for programming.

APPENDIX I

INTERNAL CODES

STANDARD CHARACTER CODES

SYMBOL	ANSCII	EBCDIC	EBCDIC (CARD CODE)	SYMBOL	ANSCII	EBCDIC	EBCDIC (CARD CODE)
blank	A0	40	blank	@	C0	7C	8-4
!	A1	5A	11-8-2	A	C1	C1	12-1
"	A2	7F	8-7	B	C2	C2	12-2
#	A3	7B	8-3	C	C3	C3	12-3
$	A4	5B	11-8-3	D	C4	C4	12-4
%	A5	6C	0-8-4	E	C5	C5	12-5
&	A6	50	12	F	C6	C6	12-6
'	A7	7D	8-5	G	C7	C7	12-7
(	A8	4D	12-8-5	H	C8	C8	12-8
)	A9	5D	11-8-5	I	C9	C9	12-9
*	AA	5C	11-8-4	J	CA	D1	11-1
+	AB	4E	12-8-6	K	CB	D2	12-2
,	AC	6B	0-8-3	L	CC	D3	11-3
-	AD	60	11	M	CD	D4	11-4
.	AE	4B	12-8-3	N	CE	D5	11-5
/	AF	61	0-1	O	CF	D6	11-6
0	B0	F0	0	P	D0	D7	11-7
1	B1	F1	1	Q	D1	D8	11-8
2	B2	F2	2	R	D2	D9	11-9
3	B3	F3	3	S	D3	E2	0-2
4	B4	F4	4	T	D4	E3	0-3
5	B5	F5	5	U	D5	E4	0-4
6	B6	F6	6	V	D6	E5	0-5
7	B7	F7	7	W	D7	E6	0-6
8	B8	F8	8	X	D8	E7	0-7
9	B9	F9	9	Y	D9	E8	0-8
:	BA	7A	8-2	Z	DA	E9	0-9
;	BB	5E	11-8-6	[	DB	4F	12-8-7
<	BC	4C	12-8-4		DC	4A	12-8-2
=	BD	7E	8-6	]	DD	5F	11-8-7
>	BE	6E	0-8-6	↑	DE	6D	0-8-5
?	BF	6F	0-8-7	←	DF	6A	0-8-2

ANSCII CONTROL AND TRANSMISSION CODES

FUNCTION	ANSCII	FUNCTION	ANSCII
NULL	80	DC1(Reader on)	91
SØH	81	DC2(Punch on)	92
STX	82	DC3(Reader off)	93
ETX	83	DC4(Punch off)	94
EØT	84	NAK	95
ENØ	85	SYNC	96
ACK	86	ETB	97
BELL	87	CAN	98
BS	88	EM	99
H TAB	89	SUB	9A
LINE FEED	8A	ESC	9B
V TAB	8B	FS	9C
FØRM	8C	GS	9D
CARRIAGE RETURN	8D	RS	9E
SØ	8E	US	9F
SI	8F	DEL(Rubout)	FF
DLE	90		