INCOTERM CORPORATION

SPD 1Ø/25 INTELLIGENT TERMINAL SYSTEM

PROGRAMMER'S REFERENCE MANUAL

PRELIMINARY COPY

April, 1977

MS-7217.1

PREFACE

The INCOTERM SPD 10/25 Intelligent Terminal is a compact, versatile system which has found application in a wide variety of environments, such as banking systems, airline reservation networks or as a stand-alone intelligent terminal system.

This is the SPD 10/25 Programmers Reference Manual. When used with the SPD Symbolic Assembly Language Reference Manual, it is the only manual needed to program a SPD 10/25 terminal alone, or running under SPD/DOS.

The information in this manual is presented for informational purposes only and is not intended or licensed to be used for the construction of equipment. The information is believed to be accurate, but no responsibility is assumed for inaccuracies or for the consequences of using the information.

Further, INCOTERM Corporation makes no representation that use of the information in this manual will not infringe on existing or future patent rights of INCOTERM or of others.

CONTENTS

1. SPD 10/25 Organization

The SPD 10/25 TPU internal system organization is that of a processor with the connections for all I/O devices, including the peripherals, the display monitor generators, the keyboards, the communications lines, a half duplex communications controller and a cyclic check controller. All other connections are made through a commonly shared I/O bus (see figure 1-1). The bus organization allows outside connections for devices that house their own controllers (for example, a paper tape reader), and inside connections to the built in devices: the keyboards at every display station and the communications controller. The connection of any other peripheral device is made by simply:

* Inserting into a TPU slot a self contained plug-in module, which is the controller for the device, and

* Assigning its address to one of the output channels through which the TPU communicates with the controllers.

The controller module is a board that fits easily into the TPU housing. There are slots for any combination of five single device controllers. The system is readily reconfigured to run with different applications programs by changing the plug-in modules.

1.1. Input-Output Subsystem

The I/O subsystem (figure 1-2) provides the means for all transfers of data into and out of the TPU. The source for and target of all input and output operations is the ACR (accumulator). The ACR is a register which holds 8 bits of data.

The common I/O bus is the path for data transmitted into and out of the TPU. The bus is common to the ACR and the 8 addressable channels. The devices assigned to these channels are chosen when the system is configured. Two targets or sources for data are internal to the TPU: the RTC (real time clock) and the refresh subsytem. The remainder are external.

The data sources or targets are interfaced to the channels by controllers. The controller provides all the necessary logic to conduct the exchange of control information and I/O data, transmitting an interrupt signal (over an exclusive line) and effecting the data transfer over the I/O bus. Information concerning controller programming is not given in this manual. Consult the INCOTERM Data Communications Manual for a description of general controller operation.

Figure 1-1:
SPD 10/25 Internal Organization

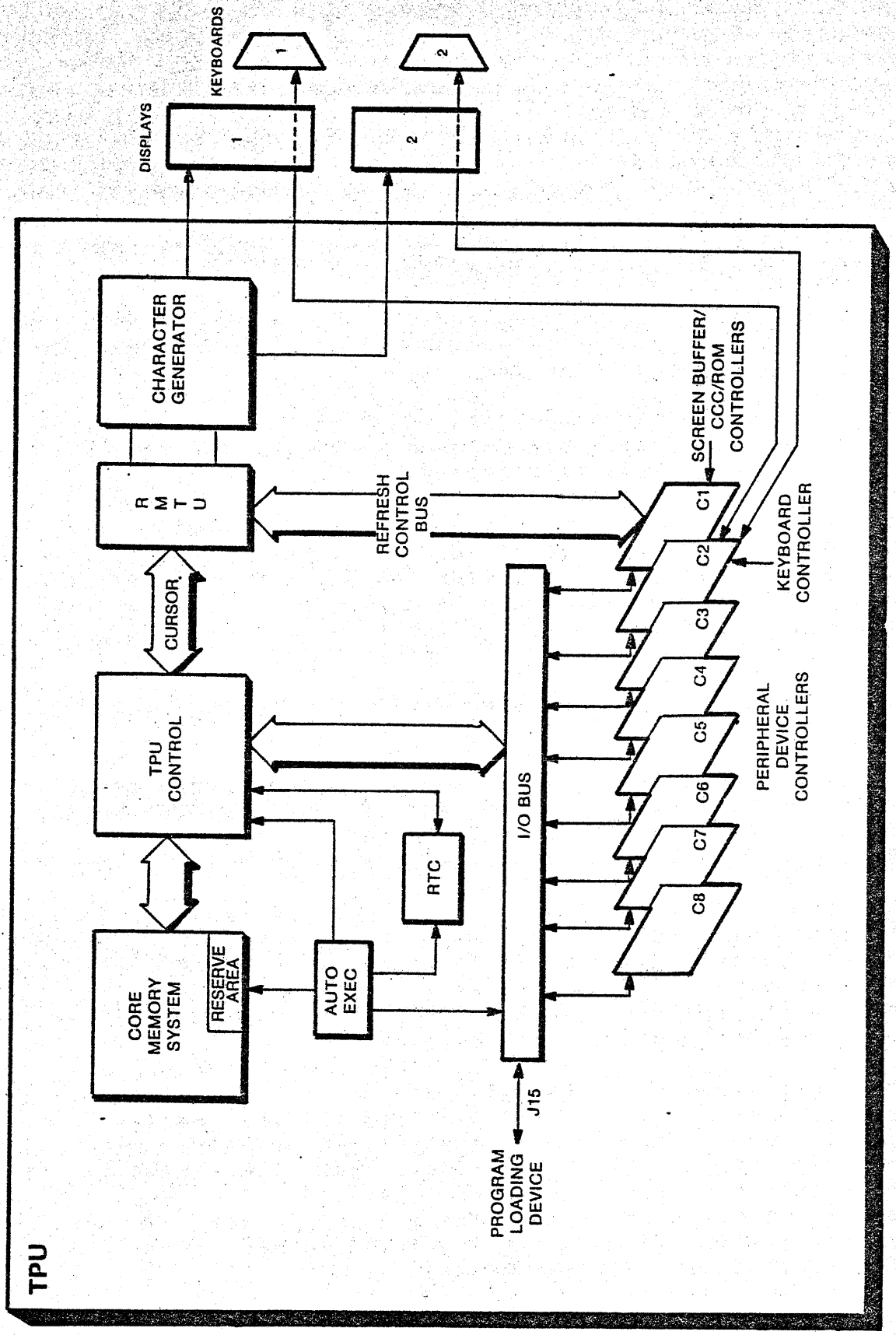
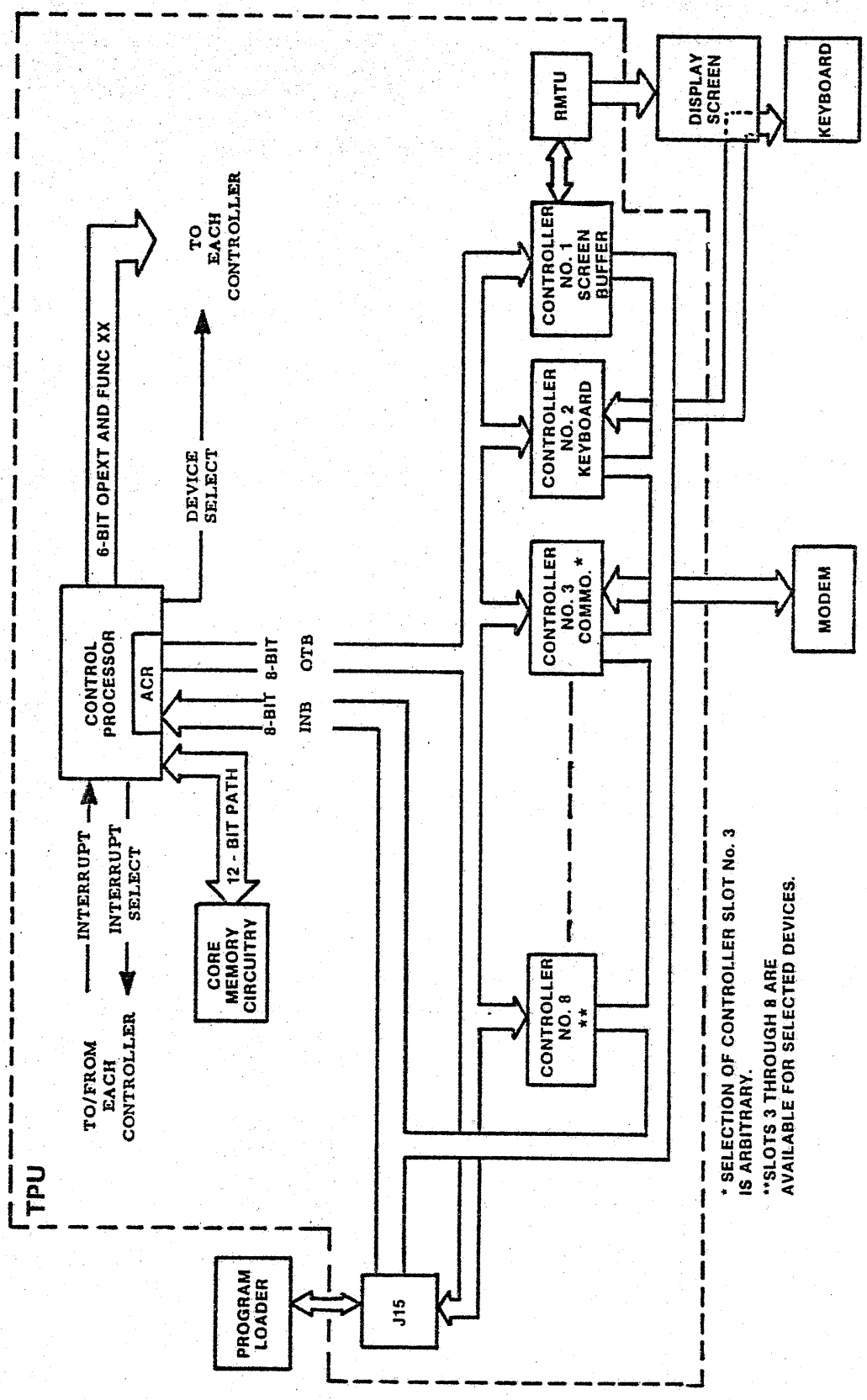


Figure 1-2:
SPD 10/25 I/O Subsystem



SPD 10/25 Functional Characteristics

Type:	Parallel binary, Byte oriented
Data Format:	8 bit byte, 16 bit word
Instruction Length:	One or two words long
Program Memory:	Magnetic Core
Memory Size:	4K Bytes
Memory Cycle:	1.6 microseconds
Addressing:	Direct, multi-level indirect
Arithmetic Code:	Two's complement, Unsigned
Refresh Display:	MOS RAM
Refresh Size:	2K bytes (4K: Dual 1920 Cnfg)

1.2. SPD 10/25 Family Architecture

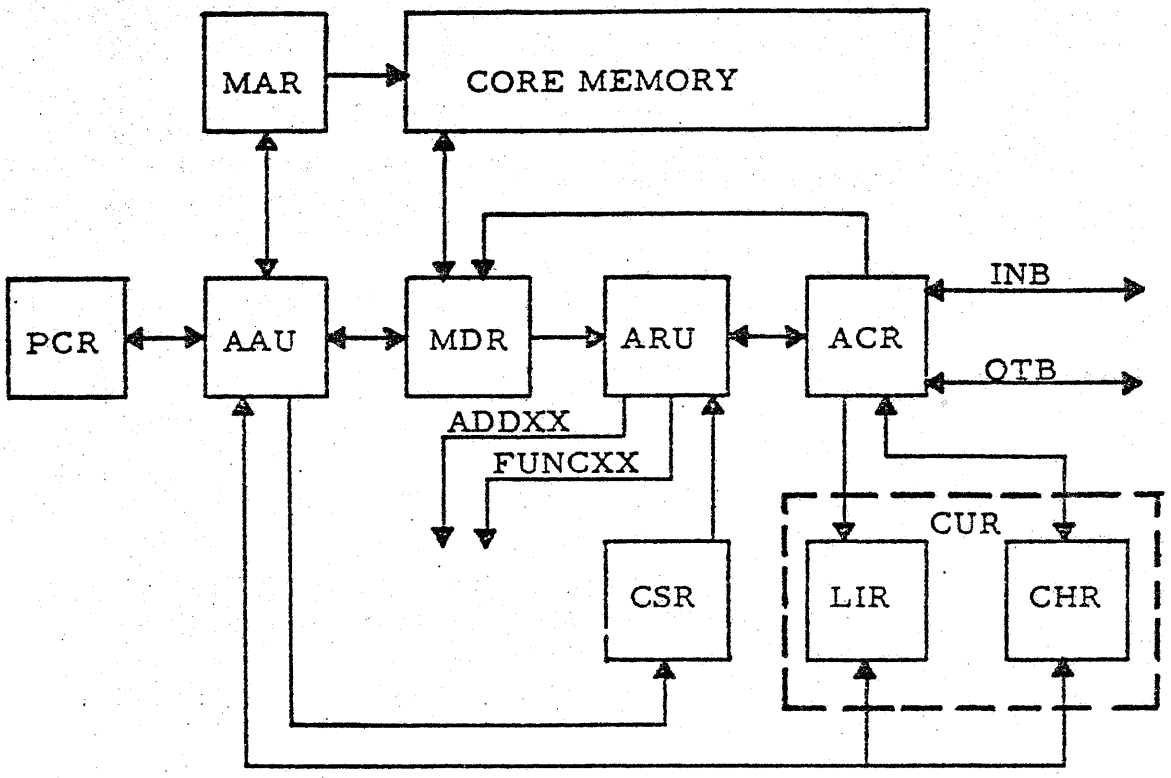
The TPU consists of a magnetic core memory, an Auto-Exec interrupt structure, arithmetic logic hardware, control registers, a power/save restart feature and a real-time clock. Figure 1-3 is a data flow diagram of the TPU illustrating the memory registers, arithmetic unit and I/O busses.

1.2.1. Auto-Exec

Auto-Exec is the name applied to the SPD series unique hardware controlled interrupt structure. Auto-exec provides 8 levels of priority interrupts and performs all register storage and control functions required to service an interrupt.

An interrupt is a randomly occuring signal from a device which requests some action from or provides some information to the TPU and the internally stored program. Devices which may cause interrupts are the RTC, keyboards, and all peripheral device and communications controllers. The auto-exec structure provides, via hardware, most of the functions required to handle an interrupt when it occurs.

Figure 1-3
SPD 10/25 Data Flow Diagram



Auto-exec saves the programmer from the burdensome task of creating special routines to: determine the source of an interrupt, save and restore the contents of the registers, request specific I/O operations and to sequence the specific program routines to be executed when an interrupt occurs. Programs may or may not utilize auto-exec. Programming under the auto-exec structure is described in section 3.

1.2.2. Core Memory

Memory is non-volatile magnetic core: the contents are not destroyed when the system is powered-up or down. Memory is divided into 8 sectors of 512 bytes each giving a total size of 4096 bytes (4K).

Each byte of memory is addressable and available to the program. Programs can modify instructions in memory, making intelligent programming possible.

1.2.3. Power Restart

The first instruction to be executed when power comes on is fetched from the address contained in core location FFC. This is the part of the Memory Reserve area used by the hardware for the Power-Restart feature. Here, the program can store the address of the instruction or routine to be executed whenever the TPU is powered down and up again.

1.2.4. Refresh Subsystem

The refresh subsystem provides all the necessary controls and interfaces to drive the particular configuration in use.

The SPD 10/25 Dual 960 configuration drives two screens of 960 characters each. It is supplied with 2K MOS RAM memory and two keyboards plus two display monitors.

The Dual 1920 configuration drives two 1920 character screens. It is supplied with two displays plus two keyboards and 4K MOS RAM refresh memory. It is possible to drive up to four displays consisting of 960 characters each.

The Single 1920 configuration drives one display of 1920 characters each. It is supplied with one keyboard and display. Except for the lack of an additional keyboard controller and display, this configuration is identical to the Dual 960 configuration.

The cursor is generated by the hardware but is under complete software control. The refresh subsystem also contains the the cyclic check calculation circuit and a Code Conversion ROM table (see section 5.1.).

Character generation, cursor generation, character brightness, and character blink are all controlled by refresh subsystem.

1.2.4.1. Display Monitor

The viewing monitor and screen presentation is based on standard T.V. techniques and uses a 12 inch diagonal CRT (Cathode Ray Tube) monitor. Characters are displayed in an 8 X 12 dot matrix providing clear and highly readable font. The screen uses green P39 phosphor for high light output efficiency and resulting ease of viewing. The outer surface of the CRT is etched to minimize reflection from surrounding sources. The viewing grid is nine inches wide by 6.5 inches high. The operating mode of the display is dynamically selectable by the software program. Selections include screens sizes of either 1920, 2000 or 960 characters and line lengths of 80 or 64 characters.

1.2.4.2. Character Generation

The SPD 10/25 display provides several upper case only and upper/lower case character sets. The character set employed may be selected by the user from those available. The character set chosen is generated by the refresh subsystem and the read only memory character generator.

Each character is constructed from a matrix of dots which fits into a display position that is in an 8 X 12 dot matix area within a 10 X 16 envelope. The unused portion of the display allows for spacing between characters and lines. Programming the refresh subsystem is explained in section 4.

1.2.5. Real-Time Clock: RTC

The RTC, a part of the screen module and control and timing unit (SMCTU), indicates the passage of actual time to the programmer. Its interrupts receive the second highest priority. The RTC interrupts 60 times a second. One use of the RTC is to control the execution of multiple tasks so that each one is performed.

For programming purposes, the RTC is treated as an external device. Programming information for the real time clock is given in section 7.

1.2.6. TPU Registers

Six registers control the TPU program operations. They are briefly described in the following sections. Unlike core, the contents of the registers (with the exception of the PCR) are random when the TPU is powered-up.

1.2.6.1. Accumulator: ACR

The ACR, an 8 bit register, is the primary arithmetic and logical register of the TPU. This register is the source or target for data in I/O operations.

1.2.6.2. Condition Status Register: CSR

The CSR is a three bit register which indicates the status as a result of all arithmetic and compare operations. The layout of the CSR is:

NG Negative	EQ Equal	CO Carry Out
Set if Msb of ACR is 1 following an arithmetic oper.	Set if the result of an arithmetic oper. is 0 or if a compare oper detects equality	Set if an overflow occured during an arithmetic oper or if a greater than or equal condition was detected

1.2.6.3. Cursor Register: CUR, $C or $X

The CUR is a 12 bit register used to control the display position of the cursor symbol. It can also be used as a general data register to manipulate 12 bit quantities.

The cursor register may be loaded from or stored into a word address in core. A comparison between the CUR and a word in core can be made to set the CSR.

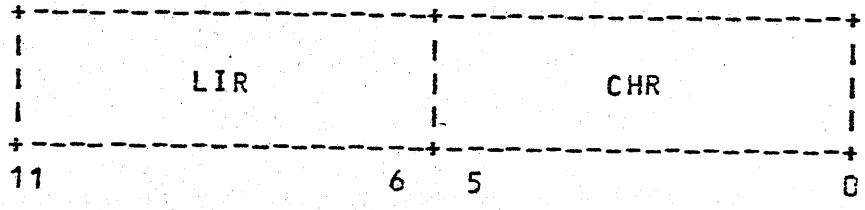
The CUR is unique in that it is not only a hardware register but

also resident in the top two bytes of the Top sector, FFE-FFF. Whenever the cursor register is modified by an instuction, the hardware automatically updates the core word located at FFE.

In case of a power-down, the contents of the CUR are lost, although the contents of core locations FFE-FFF are not. In order to rephase the CUR with the contents of this location, a LDC $C instruction must be issued. The assembler automatically defines $C to be FFE.

Information on positioning the cursor is given in section 4., Refresh Subsystem Programming.

The cursor register is divided into two fields, the line register field (LIR) and the character register field (CHR):



Cursor Register

In 64 character/line mode, the LIR and CHR correspond to the line and character position respectively where the cursor will appear when enabled. Instructions are provided for moving the ACR to and setting it from these fields.

The LIR and CHR are NOT separate registers, nor do they necessarily indicate where data will be displayed on a screen. They are cursor register fields that can be used as general work areas to suit the needs of the software program.

1.2.6.4. Memory Address Register: MAR

The MAR, a 12 bit register, stores the address of the memory byte or word.

1.2.6.5. Memory Data Register: MDR

The MDR, a 12 bit register, transfers all data to and from core memory.

1.2.6.6. Program Counter Register: PCR

The PCR, a 12 bit register, contains the address of the next instruction to be executed. The PCR always references the high-order (even) byte of the two or four byte instruction. The PCR is set after the execution of an instruction, or as a result of auto-exec action. On powering up the TPU, the PCR is automatically set to the contents of the word at the power restart location, FFC.

1.3. Memory Addressing

The core memory is divided into 8 sectors corresponding to a memory size of 4K. The Top and Top-1 sectors maintain a special significance: some instructions can address these sectors from anywhere in core.

Normally, one word memory reference instructions can not reference addresses out of the current or Top sectors. However, a word containing the referenced address (called a link or pointer) can be placed in the Top sector and that address can be referenced from anywhere in core. The technique of resolving address references employs indirect addressing, which is described in section 1.3.2.

The Top sector contains the memory reserve area. This area, FD4 through FFF, contains 20 bytes used by the auto-exec interrupt hardware, 2 bytes to denote the value of the cursor register and 2 bytes for the power restart address.

The memory reference instructions use direct or indirect addressing to obtain an effective address in the core memory. An effective address, as opposed to the operand address specified as part of the memory reference instruction, is the ultimate address on which the instruction acts. The two types of addressing are described in the following sections.

1.3.1. Direct Addressing

In a two word instruction, the second word contains the 12 bit effective address. In a one word memory reference instruction, a 12 bit address is derived from the 9 bit address field of the instruction and the PCR, if the reference is to an address within the current sector. If the address lies in the Top sectors, the remaining address bits are derived from the sector field, bit location 9, instead of the PCR.

As noted earlier, one-word memory reference instructions may

never directly reference addresses out of the current sector, except when they lie in the Top sectors.

1.3.2. Indirect Addressing

Indirect addressing in fundamental to SPD programming. This type of addressing is best explained by an analogy.

Suppose a messenger is to deliver a parcel to a certain office number. He does not know to what office the parcel must ultimately be delivered, but he is told to go to room 400 where he will receive further instructions.

Upon arriving at room 400 he finds a note instructing him not to leave the parcel here, but to continue to room 500. At room 500, the messenger finds a secretary waiting expectantly for the parcel. This is multi-level indirect addressing.

The SPD hardware operates in a similar fashion. A memory reference instruction is using indirect addressing if bit 10 in a one word instruction or bit 15 in the address word of a two word instruction is one. The word whose address is given by the instruction is then inspected. If bit location 15 is 0, the word contains the effective address.

If bit location 15 is 1, then a second word given by this indirect address is examined. This process continues until a word whose bit 15 is 0 is inspected. The address contained in this word is the effective address.

All instructions using indirect addressing must always reference an even (word) address.

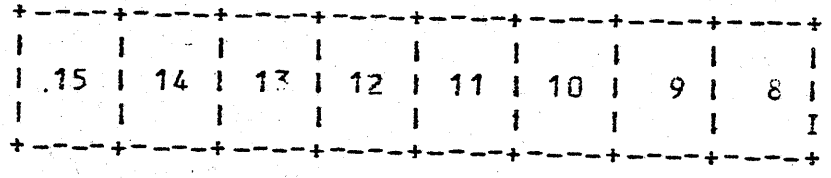
One pitfall which must be avoided when employing indirect addressing is having the program enter an infinite indirect loop. This situation may stem from one of two roots. An indirect word may be referring back to the original indirect reference, or the program may have 'blown up' and is executing random core data. Both situations indicate fatal programming errors.

2. Instruction Repertoire

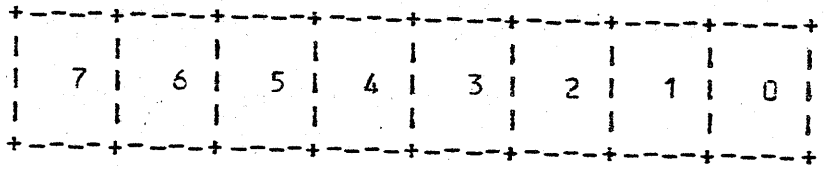
The instruction repertoire can be divided into nine classes: byte, cursor, increment, compare-and-jump, test-jump, immediate and operate.

2.1. Word Format

A memory word has 16 bits and is divided into an even and an odd byte. Bits 0 through 7 are the low order byte and bits 8 through 15 are the high order (even) byte. Some instructions can reference the odd or the even byte of the memory word, while some can only reference the even byte. The format of a memory word is given below:



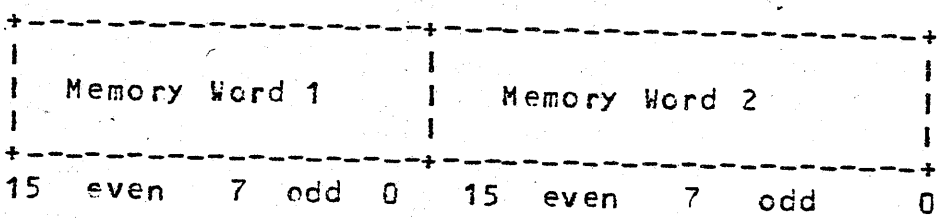
Even Byte Address = X



Odd Byte Address = X + 1

Memory words are addressed at their high order (even) byte. Instructions which act on words (as opposed to bytes) should therefore give an even address as their address operand, though some applications dictate addressing an odd byte. However, to achieve near SPD 20 Family compatibilty, word class instructions should address only an even byte.

Memory words are arranged in core in a linear fashion, their odd byte always being the higher address:



2.2. Instruction Formats

This section describes the format of the one-word non-memory reference, one-word memory reference and two-word memory reference instruction set used in all SPD 10/25 terminals.

2.2.1. Notation Used

All machine instruction codes are given in hexadecimal notation. Hex notation is standard SPD series convention, since the machine structure is based on powers of 16 (i.e., 16 bits = 1 word).

Special lower case symbols are incorporated into the machine code of those instructions which contain implicit instruction fields (fields which are defined at code generation time). An explanation of the symbols used follows.

Memory reference instructions can address two types of data. Instructions which can reference odd or even bytes represented by the occurence of ea in their instruction code. Instructions which can act only on an even address (i.e., word address) are represented by the occurence of wa in their code.

The address word of two-word instructions is represented by a -dash. Function and device channels for the TIO, CIO, RIO and WIO instructions are represented by f and c respectively.

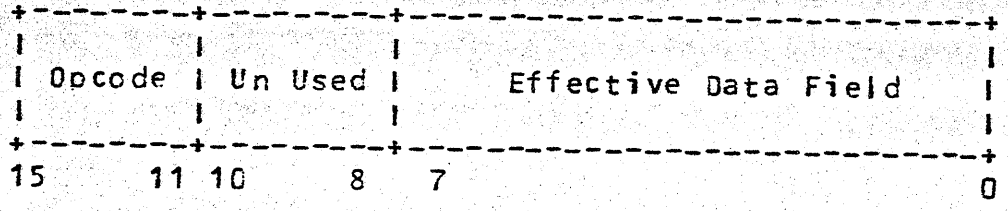
Instructions which contain their own effective data (e.g., immediate class) are represented by the occurence of ed in their instruction code.

2.2.2. One-word Non-memory Reference

The one-word, non-memory reference instructions have three formats. The immediate, I/O and operate class of instructions use these formats.

2.2.2.1. Immediate Class Format

The Immediate class of instructions contain their own data constants in bits 0 through 7 of the the second byte. This byte is called the effective data field. Bits 8 through 10 are not used. Bits 11 through 15 denote the operation code (opcode) which is used by the TPU to select the operation to be performed. The format of the immediate class of instructions is illustrated below:



The instructions of the immediate class are described on the following pages.

Mnemonic	Definition	Code	Description
LDI	Load Immediate	80ed	Places the effective data in the ACR. The original contents are lost. The CSR does not change.
ADI	Add Immediate	90ed	Adds the effective data to the contents of the ACR, leaves sum in ACR. The original contents are lost. The bits in the CSR are set according to table 2-2.

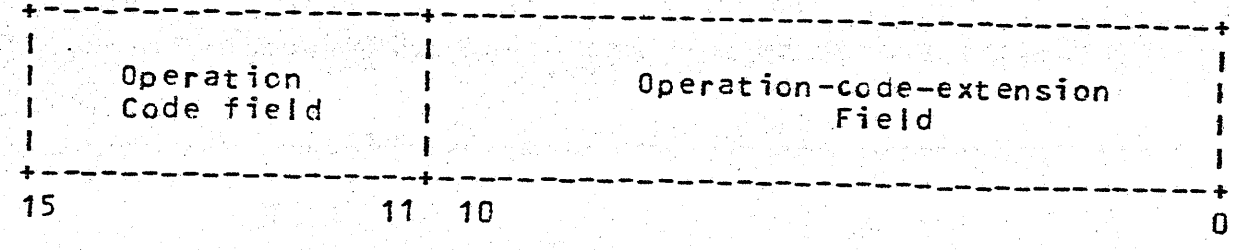
Mnemonic	Definition	Code	Description
SBI	Subtract Immediate	98ed	Adds the Two's complement of the effective data to the ACR. The results are placed in the ACR and the original contents are lost. The CSR is set if the ACR was greater than or equal to the value of the effective data. The other bits are set according to table 2-2.
ANI	And Immediate	A8ed	Logically Ands effective data with the contents of ACR, leaves result in ACR, destroys the original contents. The CSR does not change.
ORI	Or Immediate	B0ed	Logically Ors effective data with contents of the ACR, leaves result in ACR, destroys original contents. The CSR does not change.
XOR	Exclusive Or Immediate	D0ed	Logically exclusive Ors the effective data with the contents of the ACR. Leaves result in ACR, destroys original contents. The CSR does not change.
CLA	Clear ACR	8000	An assembler short-cut for a LDI 0 instruction. Sets ACR to 0. CLA is not a hardware instruction.

2.2.2.2. Operate class of instructions

The format of the operate class of instructions has an operation-code-extension (opext) field. Bits 0 through 10 the opext field, define the various instructions. Bits 11 through 15 denote the opcode field, defining the class of the instruction. Thus, the opcode field is a constant: the binary codes do not change for the various instructions. These instructions either

influence the general status of the ACR or control various TPU operations.

The format of these instructions is given below:



2.2.2.2.1. Operate Class Registers:

These instructions influence the general status of the ACR. The PCR is incremerted by two following execution. The opcode for all the instructions is 11000.

Mnemonic	Definition	Code	Description
CLC	Clear CHR	C00D	Sets CHR to zero.
CLL	Clear LIR	C00C	Sets LIR to zero.
MAC	Move ACR to CHR	C005	Moves the 6 Lsb of the ACR to the CHR
MAL	Move ACR to LIR	C003	Moves the 6 Lsb of the ACR to the LIR
MCA	Move CHR to ACR	C004	Moves the contents of the CHR to the ACR and sets the ACR two Msb off (0).
MLA	Move LIR to ACR	C002	Moves the contents of the LIR to the ACR and set the ACR two Msb off.
SHL4	Shift ACR Left 4	C009	Transfers the four Lsb of the ACR to the four Msb of the ACR. The four Lsb of the ACR are replaced with zero. The CSR is not affected.

2.2.2.2.2. Operate Class: Control

These instructions affect the general execution time or affect the auto-exec interrupt structure of a program. The PCR is incremented by two. The ACR and CSR are not changed. The opcode for these instructions is 11000. They are listed on the following pages.

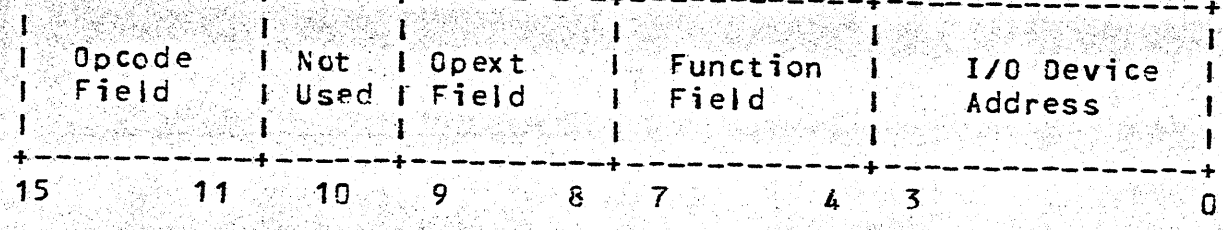
Mnemonic	Definition	Code	Description
NOP	No Operation	C000	Delays processing for one memory cycle
HALT	Halt	C001	Used only with a programmers console or built in maintenance panel when it is enabled. All activities of the TPU are inhibited. If a programmers console is not used, or if the panel is not enabled, the instruction is treated as a NOP. On depression of Run, when the console is enabled, processing resumes with the next instruction or an auto-exec cycle occurs,- whichever is appropriate.
IOR	I/O reset	C008	Resets any pending interrupts, prevents any new interrupts from entering the TPU and causes the controllers for the peripheral devices to enter their reset states. This instruction causes an automatic DSB instruction to be issued. Screen display is shut off as the result of an IOR.

Mnemonic	Definition	Code	Description
ENB	Enable Interrupt	C006	Places the TPU in a mode which allows interrupts to occur. The effect of this instruction is inhibited until the next instuction is executed. When a peripheral NAKs, (fails to acknowledge a RIO, WIO) the auto-exec issues an automatic ENB.
DSB	Disable Interrupt	C007	Places the TPU in a mode which prevents interrupts from occurring. All current interrupts remain active however. It is issued as a result of an IOR instruction or as the result of an interruot from a peripheral device.
WAIT	Wait	C00F	Enables interrupts and causes the TPU to wait. No instructions are executed until an interrupt occurs. An interrupt causes normal auto-exec action. When the interrupt processing is completed, the instruction following the WAIT is executed.

2.2.2.3. I/O class format

The format for the I/O class of instructions have five fields. Bits 0 through 3, the device address field, denote the peripheral device being addressed. Bits 4 through 7 the function field, denote to the controller what action is tc be performed. Bits 8 and 9, the opext field, define the various I/O instructions (CIO, RIO, WIO). Bit 10 is not used and can be set to zero. Bits 11 through 15, the opcode field, denote the class of instruction.

The format for the I/O instructions is given below:



These instructions provide access to the various controllers, RTC, and Refresh Subsystem. These functions include reading and writing data, masking and unmasking interrupts and other device specific controls. The I/O address selects the peripheral device to which the information is going, the refresh subsystem or the the RTC. The function field instructs the selected device of what action is to be taken. The CSR is not affected. The opcode is 11001. The instructions in this class are given below:

Mnemonic	Definition	Code	Description
CIO	Control I/O	C9fc	When this instruction is issued, the selected peripheral device c performs the specified function f. The contents of the ACR are available if auxiliary data is required to execute the command. This instruction never causes action by the auto-exec. The ACR is not affected and the PCR is incremented by two. An interrupt cannot occur between a CIO instruction and the immediately following instruction.

Mnemonic	Definition	Code	Description
RIO	Read I/O	CAfc	This instruction requests data from the selected peripheral device c. The type of data requested is determined by the function field f. If the peripheral device is prepared to transfer the required data, it acknowledges the request with an ACK signal. The TPU loads the data from the peripheral device into the ACR, and the original contents are lost. If the selected device has no data, it does not send an ACK signal; it NAKs the request. The auto-exec system is then set into motion.
WIO	Write I/O	CBfc	This instruction sends the data in the ACR to the selected peripheral device c. The type of data is defined by the function field f. If the device is ready to accept the data, it sends an ACK signal. Then, the TPU transfers data to the peripheral device. If the device is not ready to accept the data, it does not send an ACK signal; it NAKs the requests and the auto-exec system is activated.

2.2.3. One-word Memory Reference

Bit 9, the sector field, denotes the particular sector in which the referenced location resides. If bit 9 is one the instruction can reference a location in the Top sector. If the instruction is

already in the Top sector, it can reference the Top-1 sector when bit 9 is zero.

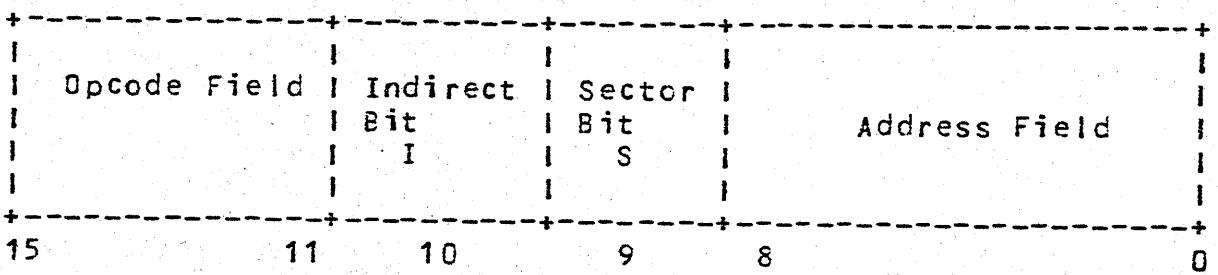
Bit 10, the indirect bit, is one if the instruction is using indirect addressing and zero if not.

To calculate an effective address, the following procedure is implemented by the hardware: the sum of the address of the referenced sector and the value of the address field of the instruction is calculated. If bit 10 is zero, the instruction is using direct addressing, and the calculated sum is the effective address.

Table 2-1 Sector Addressing

Bit 9 S	Sector being referenced: Current NE Top	Current EQ Top
0	Current	Top-1
1	Top	Top

If bit 10 is a 1, the instruction is using indirect addressing. The word whose address is the sum of the above calculation must then be inspected. If bit 15 of that memory word is zero, the effective address is equal to the value of bits 0 through 14 of that word. If bit 15 of the memory word is one, an additional word whose address is equal to the value of bits 0 through 14 is inspected. This process continues until a word with bit 15 equal to zero is obtained. This iterative process is multi-level indirect addressing. The effective address is equal to bits 0 through 14 when bit 15 is zero.



2.2.3.1. Instruction Codes

One-word memory reference instructions can always be represented unambiguously in binary notation. However, since binary instruction representation is notoriously cumbersome, representation of machine codes is usually given in hexadecimal or octal. The SPD series instruction set is given in hex notation.

Instructions of the one-word memory reference class use fields which do not break up into groups of 4 bits, making alternate representations of the instructions possible. For example, a current sector Load instruction from address 100 has the representation:

```
00000 0 0 100000000          0100
                    2           16
```

A Load from address 0, however, is represented as:

```
00000 0 0 000000000          0000
                    2           16
```

Thus, the first two hex digits, which contain the opcode, indirect and sector bits, vary depending on whether the Msb of the 9 bit address is on, whether the instruction is using indirect addressing or whether the referenced byte lies in the current sector.

The hex representation of instruction codes used in this manual always gives the lowest representation of the instruction (all sector and indirect bits, plus bit 8, off). However, the alternate representations should be kept in mind when working out hex instructions codes of the one-word memory reference instructions.

2.2.3.2. Byte Class

Each instruction utilizes the ACR and the contents of the byte at that effective address. Instructions of this class may reference an odd or an even byte.

Mnemonic	Definition	Code	Description
LD	Load	00ea	Replaces the value in The ACR with the contents of the byte at the effective address. The CSR and the contents of the effective address do not change
ST	Store	08ea	Replaces the contents at the effective address with the value in the ACR. The original contents of the byte are lost. If the effective address is in the memory reserve area FD7-FFF, the contents of the bytes used by auto-exec, power restart, and the cursor register may be modified. The ACR and CSR are not affected.
AD	Add	10ea	Adds the contents of the byte at the effective address to the contents of the ACR. The result is placed in the ACR, replacing the original the contents of the contents. The bits in the CSR are set according to table 2-2.

Mnemonic	Definition	Code	Description
SB	Subtract	18ea	Adds the Two's complement of the byte at the effective address to the contents of the ACR. The result is placed in the ACR, and the original contents are lost. The CO bit in the CSR is set if the ACR was greater than or equal to the contents of the byte at the effective address. The remaining bits are set according to table 2-2.
CM	Compare	20ea	Adds the Two's complement of the contents of the byte at the effective address to the contents of the ACR, and tests the result of this calculation. The CSR is set according to table 2-2. The contents of the ACR remains unchanged.
AN	Anc	28ea	Logically And's the contents of the byte at the effective address with the contents of the ACR, the result is placed in the ACR and the original contents are lost. The contents of the byte at the effective address and the CSR are not affected.

Mnemonic	Definition	Code	Description
OR	Or	30ea	Logically Or's the contents of the byte at the effective address with the contents of the ACR. The original contents are lost but the contents of the byte and the CSR are not affected.

2.2.3.3. Increment class

These instructions perform an arithmetic operation on the word at the effective address. They are one-word memory reference instructions. If the effective address is in the reserve area, the specified operation is performed on the contents of the bytes used for: the auto-exec, power restart, and the cursor register. The ACR is not affected by these instructions, the PCR is incremented by two after execution. The instructions of the increment class are listed below:

Mnemonic	Definition	Opcode	Description
INC	Increment	50wa	Increases by one the value of the word at the effective address. The original contents of that word are lost. The CSR is set under the conditions given in table 2-2 except that the NG bit is not used. INC acts only on the 12 Lsb of the word, the upper 4 bits are ignored.

Mnemonic	Definition	Code	Description
DEC	Decrement	58wa	Decreases by one the value of the word at the effective address. The original contents of the word are lost. The CSR is set under the following conditions: the CO bit is set for each execution except when the value of the word becomes zero. The EQ bit is set if the result is zero. The NG bit is not affected. DEC acts only on the 12 Lsb of the word. The upper 4 bits are ignored.
IN2	Increment By two	60wa	Increases by two the the value of the word at the effective address; the original contents are lost. The CSR is set as shown in table 2-2. The NG bit is not used. IN2 acts only on the 12 Lsb of the word. The upper 4 bits are ignored.

2.2.3.4. Jump class

These instructions cause an unconditional jump and directly modify the PCR. The ACR and CSR are not affected. The instructions of the jump class are described below:

Mnemonic	Definition	Opcode	Description
JMP	Uncondl. Jump	B8wa	Sets the PCR to the value of the effective address. The next instruction executed is at this effective address.
JSR	Jump and Store Return	78wa	Adds two to the value of the PCR and stores the result in the word at the effective address. The PCR is then set to the value of the effective address plus two, and execution begins there. The original contents of the word at the effective address are lost.

2.2.3.5. Cursor class

Each instruction operates on the Cursor register and the contents of the word at the effective address. The cursor register may be used to manipulate the cursor symbol or as a general 12 bit data register. For each instruction, the ACR does not not change and the PCR is incremented by two. The instructions for the Cursor class are described on the following pages.

NOTE: To rephase CUR with FFE, the CUR memory reserve location, execute the following:

```
LDC $C
or
LDC X'FFE'
```

It is important to realize that only

INC $C or IN2 $C, DEC $C

affect the contents of <u>BOTH</u> $C and the CUR.

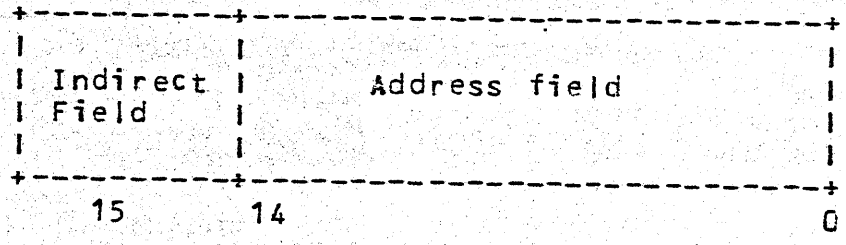
Mnemonic	Definition	Opcode	Description
LDC	Load Cursor Register	40wa	Replaces the contents of the CUR and the word beginning at FFE (CUR reserve location) with the contents of the 12 Lsb of the word at the effective address. The original contents of the CUR and CUR reserve area are lost.
STC	Store Cursor Register	48wa	Stores the contents of the CUR in the word 12 Lsb of the word at the effective address. The original contents of the 12 Lsb of the word are lost.
CMC	Compare Cursor Register	70wa	Adds the Two's complement of the 12 Lsb of the word at the effective address to the contents of the CUR. The result sets the CSR according to table 2-2. The contents of the CUR is not changed.

2.2.4. Two-Word Memory Reference

The two-word memory reference instructions have three formats. The test-jump and compare-and-jump classes of instructions use these formats. The first word of a two word memory reference instruction contains the opcode field and the conditions or data to be tested. The second word contains the address to which the program jumps if the tested conditions are met. The second word is identical in all three classes of instructions: bit 15 is the indirect bit and bits 0 through 14 contain the address field.

To obtain an effective address, the following is used: if bit 15 of the address word is 0, the effective address is equal to the value of bits 0 through 14. If bit 15 is a one, an additional

word whose address is equal to the value of bits 0 through 14 is inspected. This procedure continues until a word whose bit 15 is 0 is found, thus allowing multi-level indirect addressing.

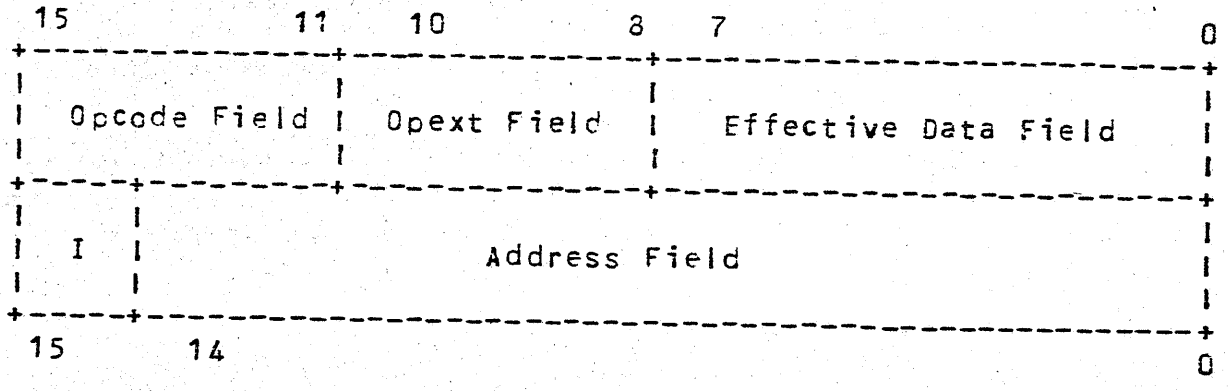


Word 2 of two-word memory reference

The three classes of two-word memory reference instructions are described on the following pages.

2.2.4.1. Compare-and-jump class

The format for the compare and jump class has five fields as illustrated below:



Bits 0 through 7 of the first word contain the effective data; bits 8 through 10 the opext field, define the conditions being tested for; and bits 11 through 15, the opcode field, denote the class of instructions. Bits 0 through 14 of the second word denote the address in core to which the program will branch if the condition being tested for is met. Bit 15 (I) denotes direct addressing if off or indirect addressing if on.

The instructions in the compare-and-jump class make an unsigned comparison with the effective data and set appropriate bits in the CSR. The status of the CSR is then examined. The opext field

and the CSR are compared for various conditions. If the conditions defined by the opext field are met by the CSR, the PCR is set to the effective address contained in the second word of the instruction. If the various conditions are not met by the CSR, the PCR is incremented by four. The ACR is not affected.

Table 2-2 lists the various conditions possible in CSR when the appropriate NG, EQ and CO bits are set. The instructions occupy two words of memory. The opcode is 10100. The instructions of the compare-and-jump class are listed on the following pages.

Table 2-2 CSR Condition Settings

NG bit Set: when ACR Msb is one	EQ bit Set: when Arith. Op. Produces 0	CO bit Set: when Arith. Op. Produces CO	Conditions Satisfied
---	0	0	LT, LE or NE
---	0	1	GT, GE or NE
---	1	0	EQ (0 + 0), LE
---	1	1	EQ (when an overflow occurs or a subtract. causes 0),GE
0	---	---	Positive (PO) when Msb of result is 0
1	---	---	Negative (NG) when Msb of result is one

2.2.4.1.1. Performing a Compare

In many programs it becomes necessary to make a comparison between operands and determine if one is equal to, or greater or less than the other.

If the comparison is made using the ACR and effective data (i.e., data contained within the instruction), the instructions of the compare-and-jump class are used.

The CJ class performs the indicated comparison on an ACR vs data basis. That is, one can remember how these instructions perform their comparison by simply replacing the vs with the desired comparison operand:

```
* ACR  GT  Data,
* ACR  GE  Data,
* ACR  LT  Data,
* ACR  LE  Data,
* ACR  EQ  Data,
* ACR  NE  Data.
```

All that remains is to find the instruction which uses the desired comparison.

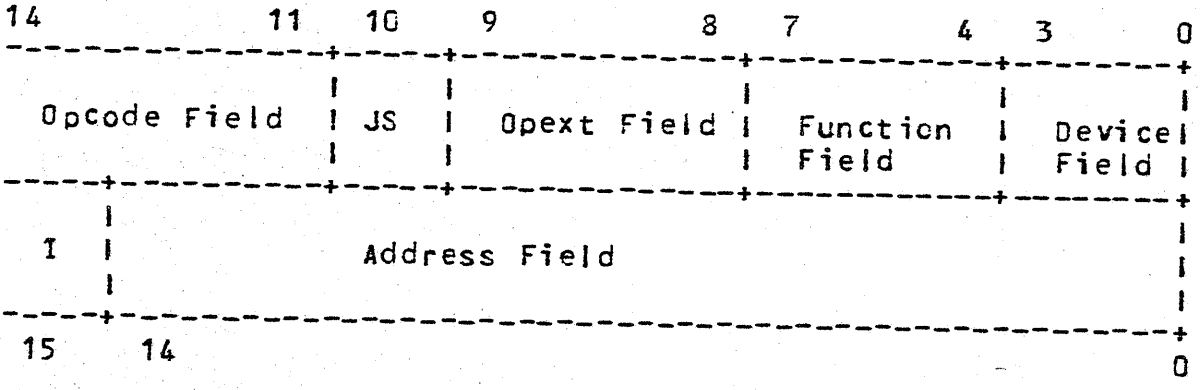
2.2.4.2. Compare-and-jump class Instructions

Mnemonic	Definition	Code	Description
CJFAL	Compare-and Jump on Condition False	A0ed-	Used to the set the CSR Never jumps.
CJTRU	Compare-and Jump on Condition True	A4ed-	Sets the CSR and always jumps.

Mnemonic	Definition	Code	Description
CJLT	Compare-and Jump on Condition LT	A1ed-	Jumps if the ACR is less than the effective data.
CJEQ	Compare-and Jump on Condition EQ	A2ed-	Jumps if the ACR is equal to the effective data.
CJLE	Compare-and Jump on Condition LE	A3ed-	Jumps if the ACR is less than or equal to the effective data.
CJGE	Compare-and Jump on Condition GE	A5ed-	Jumps if the ACR is greater than or equal to the effective data.
CJNE	Compare-and Jump on Condition NE	A6ed-	Jumps if the ACR is not equal to the effective data.
CJGT	Compare-and Jump on Condition GT	A7ed-	Jumps if the ACR is greater than the effective data.

2.2.5. Test-jump: I/O Class

The format for the test-jump, I/O class, has seven fields as illustrated below:



Bits 0 through 3 of the first word, the device field, denote which particular peripheral device controller is being tested.

Bits 4 through 7, the function field, denote what test is being performed. Bits 8 and 9, the opext field, are always 00 and define this instruction to be a test-jump, I/O instruction. Bit 10, the jump sense (JS) field denotes if the condition being tested for is true or false (0 = true, 1 = false). Bits 11 through 15, the opcode field, denote the general I/O class of instructions. Bits 1 through 14 of the second word denote the address in core to which the program will branch if the condition being tested is met. Bit 15 denotes direct addressing if off and indirect addressing if on.

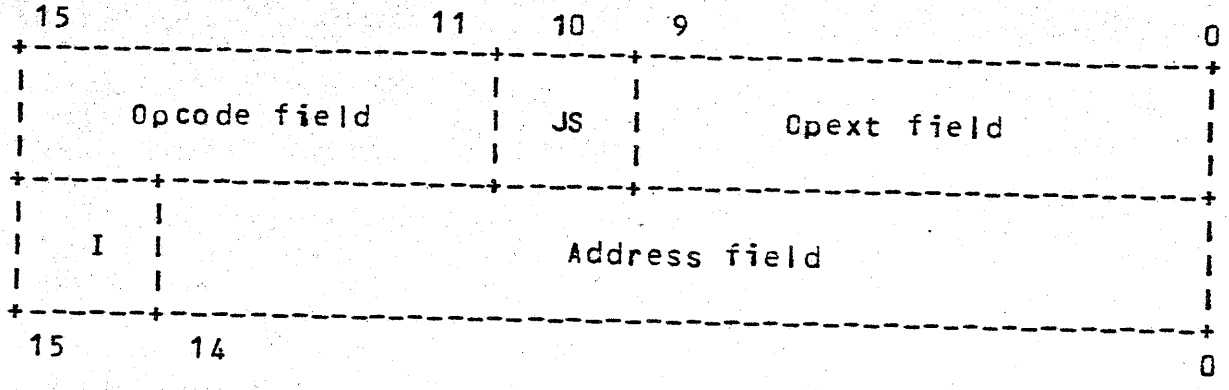
The instructions in the test-jump, I/O, class perform test on the status of the various controllers. In general, a controller may or may not respond with an acknowledge signal as the result of a specific test. The ACR, CSR, and cursor register are not changed by these instructions. If a jump is to be taken as a result of these instructions, the PCR is set to the effective address contained in the 2rd word of the instruction. Otherwise, the PCR is incremented by four. These instructions require two words of memory. The test jump I/O class of instructions consists of a JFACK and JTACK instruction. However, for simplicity, the instructions will be collectively referred to as TIO instructions. The opcode is 11001 and the opext is 00. The jump sense (JS) bit determines whether a jump should be taken as the result of a controller acknowledge.

These instructions can be used tc test the ready state of an I/O device, thus allowing the program to cause or prevent a NAK response. If the jump is not taken, an interrupt cannot occur before the next instruction is executed.

Mnemonic	Definition	Code	Description
JTACK	Jump if I/O ACK signal Is true	C8fc-	Instruction causes a jump if the contrcller being tested responds with an ACK signal.
JFACK	Jump if I/O ACK signal Is false	CCfc-	Instruction causes a jump if the controller being tested does not respond with an ACK signal.

2.2.5.1. Test-jump: Register Class

The format for the test-jump, register class of instructions has five fields as illustrated below:



Bits 0 through 9 of the first word, the opext field, define the conditions being tested. Bit 10, the JS field, denotes if the condition being tested is true or false. Bits 11 trough 15 denote the class of instructions used for testing the CSR, ACR or both. Bits 0 through 14 of the second word denote the address in core to which the program will branch if the condition being tested is met. Bit 15 (I) denotes direct addressing if off or indirect addressing if on. The instructions of the test-jump, register class, of instructions are described on the following pages.

The test-jump, register class, of instructions performs tests on the CSR or ACR. If a test is met, the next instruction to be executed is located at the effective address. Otherwise, the PCR is incremented by four and the next sequential instruction is executed. The ACR, cursor register, and CSR are not affected. These instructions require two words of memory. The tests performed are based on the opext and jump-sense (JS) fields. The opcode is 10100.

Mnemonic	Definition	Code	Description
JCFAL	Never Jump	8800-	Never causes a jump. PCR incremented by four
JCTRU	Always Jump	8C00-	Unconditional jump. PCR incremented by four
JCLT	Jump on Condition LT	8900-	Jumps if the C0 bit is zero.

Mnemonic	Definition	Code	Description
JCLF	Jump on Condition LE	8B00-	Jumps if the EQ bit is one or the CO bit is zero.
JCEQ	Jump on Condition EQ	8A00-	Jumps if the EQ bit is one.
JCGE	Jump on Condition GE	8D00-	Jumps if the CO bit is one
JCGT	Jump on Condition GT	8F00-	Jumps if the EQ bit is zero and the CO bit is one.
JCNE	Jump on Condition NE	8E00-	Jumps if the EQ bit is zero
JCNG	Jump on Condition NG	8B01-	Jumps if the NG bit is one
JCPO	Jump on Condition PO	8C01-	Jumps if the NG bit is zero
JCOD	Jump on Condition Odd	8B02-	Jumps if the Lsb of of the ACR is one.
JCEV	Jump on Condition Even	8C02-	Jumps if the Lsb of The ACR is zero

Mnemonic	Definition	Code	Description
JCCO	Jump on Condition CO	8D00-	Variant of JCGE. Jumps if the CO bit is 1.
JCNC	Jump on Condition NC	8900-	Variant of JCLT. Jumps is the CO bit is 0.
SKP	Skip next Word	8800-	Never jumps. A variant of JCFAL. The skipped word is actually in the second word of the test-jump instruction.
WJMP	Full word Jump	8C00-	Unconditionally jumps. A variant of JCTRU. The effective address is actually in the second word of the instruction.

To make a decision based on a comparison, keep the following in mind:

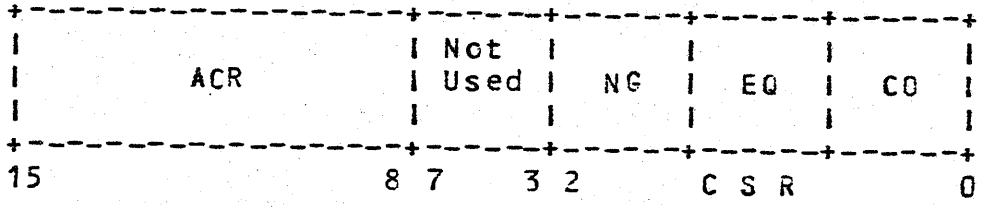
```
LD   DATA1

CM   DATA2
```

The comparison test, as executed by the test-jump instructions, is performed on a DATA1 (GT, LT, EQ, etc.) DATA2 basis.

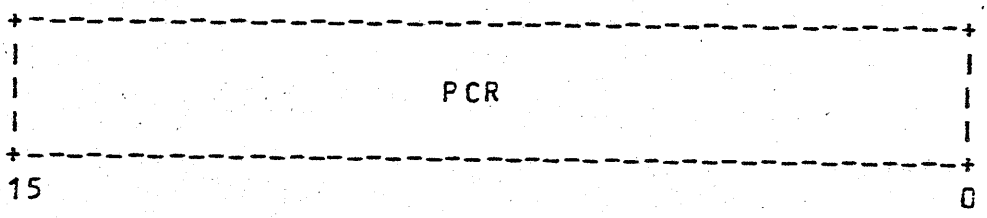
3. Programming with Auto-Exec

Auto-exec utilizes certain specific core locations in the Top sector memory reserve area. The auto-exec reserve area, FD4-FFB, is used to store and set the value of the PCR, ACR and CSR during interrupt processing.

Each device which may cause an interrupt has two words reserved for it for register storage. This block is called the stateword.



WORD X



WORD X + 1

Machine instructions are available which control the actions of the auto-exec. These instrutions allow the program to prevent or allow interrupts from all or specific devices, to wait for interrupts and to reset interrupts.

Each interrupting device is assigned a priority which schedules the devices' interrupt. The assigned priority, together with the entire memory reserve area is illustrated in the following table.

Table 4-1
Memory Reserve Area

Address	Area	Priority
FFE-FFF	Cursor Register	
FFC-FFD	Power Restart	
FF8-FFB	Background	
FF4-FF7	Real Time Clock	1
FF0-FF3	Screen Buffer, ROM	*
FEC-FEF	Device 1 Loader	2
FE8-FEB	Device 2 Keyboard	3
FE4-FE7	Device 3	4
FE0-FE3	Device 4	5
FDC-FDF	Device 5	6
FD8-FDB	Device 6	7
FD4-FD7	Device 7	8

* used as the default controller slot

3.1. The Background-Foreground Concept

Processing may be considered to fall into two categories:

Background Processing:

The processing of tasks which are of a lower priority when higher pricrity I/O tasks are not using the system resources, and,

Foreground Processing:

The processing of the I/O requests that have been designated to preempt the use of the computing facilities (real-time programs, for example).

The auto-exec interrupt structure provides a clean, automatic, program reducing tool which handles the transition from background to foreground tasks. Actually, auto-exec provides the facilities for multi-task programming: Foreground tasks are processed, on a priority basis, when their interrupt occurs.

The auto-exec program/response is as follows:

* The program is in foreground, processing an I/O task. Interrupts are not allowed to occur (disabled).

* A NAK signal occurs in response to an I/O request that cannot be completed (device in a not-ready state).

* Auto-exec saves the contents of the PCR, ACR and CSR at the reserve address of that device, the Sateword.

* Auto-exec sets the PCR, ACR and CSR from from the two word block at FF8, the background auto-exec area.

* Interrupts are allowed to occur (enabled).

* Execution of the background task begins at the address specified by the PCR.

Execution of the background task continues until an interrupt occurs signalling that the I/O device which previously issued a NAK, is now ready to transmit or receive data. Should two foreground tasks issue interrupts simultaneously, the higher priority task is serviced first. In response to an interrupt signal, auto-exec:

* Via instruction completion logic, waits for the current background instruction to be completed,

* Determines the interrupting device's reserve address,

* Disables all interrupts,

* Stores the contents of the ACR, CSR and PCR into the background auto-exec area, FF8,

* Sets the ACR, CSR and PCR from the device's stateword,

* Begins execution at the foreground I/O instruction.

Execution of the foreground task continues until the hardware detects another NAK signal, whereupon the auto-exec cycle is repeated.

3.2. Programming and Auto-Exec

The auto-exec is an extremely powerful programming tool which can be found only on INCOTERM computers. It should be noted that there can be up to 8 interrupting devices and respective I/O handlers. Two programming precautions, however, are necessary when using auto-exec:

1. Interrupt Nesting: allowing an interrupt to occur while already in foreground is a fatal error if not tightly controlled.

2. Background NAKing: Allowing a NAK to be received in response to a background issued I/O instruction is possible, but tricky to program (see section 3.3.1.1.1.)

3. Use of a non-addressable interrupt handler. Allowing an interrupt-response to a device whose handler address is not on the auto-exec reserve area is not permitted.

Several machine instructions are inherently auto-exec: the CIO mask and unmask, DSB, ENB, IOR, WAIT and the RIO/WIO input-output instructions. Each of these are described in the following sections.

3.2.1. CIO: Mask/Unmask Controller

Each device that can cause an interrupt has a set of CIO commands that mask and unmask the device's controller.

Unmasking a controller enables that device to issue an interrupt upon transition from a not ready to a ready state. Each device must be unmasked if it is to issue an interrupt from the auto-exec foreground.

Masking a controller prevents that device from issuing an interrupt signal. However, masking does not generally clear an interrupt. It remains pending until an IOR instruction is issued, a power-down, or until the device is unmasked and the interrupt occurs.

3.2.2. DSB/ENB Interrupts

DSB disables places the TPU in a mode which prevents interrupts from occuring. This instruction, as opposed to a mask, prevents ALL devices from issuing an interrupt. One use of DSB is to

establish a temporary foreground so that I/O instructions, which may cause interrupts, can be issued in background in a straightforward manner. This is explained further in section 3.3.1.1.1.

ENB enables all interrupts. Thus, this instruction, together with the corresponding device unmask instruction, must be issued at least once if interrupts are to cause normal auto-exec action.

Once the auto-exec is initialized, the program is not required to issue DSB/ENB instructions. Auto-exec automatically issues these instructions as a result of an interrupt or NAK signal, or an IOR.

In addition, the DOS and PD/FMS loaders pass control to the applications program with interrupts enabled, but all devices masked. This is equivalent to having the loaded program execute a ENB.

3.2.3. IOR

The IOR instruction clears any pending interrupts, and causes the controllers for the peripheral devices to enter their reset states. The instruction disables all interrupts and masks all devices.

Note: Programs running under DOS or PD/FMS must not issue an IOR instruction! The DOS and PD/FMS loaders set the controllers and TPU before passing control to the loaded program. In addition, the IOR instruction causes comatibility problems with DOSLIB and FMSLIB routines.

3.2.4. WAIT

This instruction enables interrupts and causes the TPU to wait. No instructions are executed until an interrupt occurs. An interrupt causes normal auto-exec action. When a NAK signal is issued, the instruction following the WAIT is executed.

A typical use of this instruction is to establish a scan loop of certain program flags, where WAIT is the last instruction of the scan. The loop must, in general, be executed disabled. Otherwise, race conditions could cause the program to remain in the wait state overly long or indefinitely.

3.3. Auto-Exec Initialization

The software must make the initial entries into the background auto-exec area and must ensure that interrupts are disabled before the foreground task is started. In addition, it is the software's responsibility to unmask all interrupting devices and to start up the initial foreground task handler.

Consider a routine which uses the RTC to count the passing seconds in the keyboard lights and which computes the value of PI when the RTC is not interrupting.

We code the initialization as follows:

```
DSB                     Disable Interrupts
CIO   1,15              Unmask RTC
LDC   =PITSK            Set location of background task
STC   X'7FFA'           and store into Background area
JMP   CLTSK             Start up clock task
```

After initializing the auto-exec and starting the foreground task, the remainder of the processing is handled by the auto-exec. All that remains then, is to code the two routines PITSK and CLTSK:

```
CLTSK   JSR   RTC       Subroutine to count seconds
        JMP   CLTSK     Keep looping until RTC NAKs

PITSK   JSR   PI        Compute PI
        JMP   PITSK     Hang in background loop
```

3.3.1.1. Background Processing

A control technique commonly used in the background program is the Wait Loop. In the Wait Loop technique, the last instruction executed in background before the loop proceeds, is a WAIT instruction. The loop does not resume until auto-exec returns control from an interrupt handler. The loop can consist of, for example, a scan of certain program flags.

There are several control techniques which do not utilize the WAIT instruction. For example, a continuous scan loop might consist of a task control block ring upon which a number of tasks are queued. The RTC can be used to make certain each task has an equal opportunity to be serviced. The actual alogorithms used in techniques of this type are quite complicated and beyond the scope of this discussion; the programmer is urged to consult the abundance of literature on the subject.

3.3.1.1.1. Background NAKing

To avoid background NAKing, the appropriate TIO, device busy or not ready, may be used in background I/O processing.

```
JFACK   4,2,$   Keyboard input ready?
RIO     1,2     Read Key
```

However, in many instances it is necessary to temporarily enter the foreground so that background I/O can be processed. For example, DOSLIB routines are designed to be used in background, with interrupts enabled. Many devices used by DOSLIB do not have the appropriate controller busy TIO. Thus, there is a routine, D&ENTF, that allows foreground processing of code.

This routine establishes a temporary background hang loop for the auto-exec so that I/O instructions that normally would cause a background NAK can be issued. Interrupts are disabled. Code executed in this temporary foreground will hold up other foreground tasks. Thus, interrupts must be enabled as soon as possible after the code is processed.

For an example of the coding involved in a routine such as D&ENTF, the reader is referred to the source listing of DOSLIB.

3.3.1.2. Execution Time

The execution time of an RIO or WIO instruction is affected as follows:

Execution time is 1.6 microseconds if the device acknowledges. Otherwise, execution of the background task continues until an interrupt occurs. There is a 8.0 microsecond overhead before execution of the foreground I/O task commences.

3.3.1.3. Default Condition

A default condition exists if an RIO/WIO instruction attempts to address an empty controller slot. Auto-exec detects this situation and stores the ACR, PCR and CSR in the default stateword located at FF0.

Auto-exec then sets the ACR,PCR and CSR from the background area and excution continues there.

4. Refresh Subsystem Programming

The refresh subsystem interfaces the display terminals with the refresh memory and the Read Only Memory (ROM) character generator. There are up to 2 (standard with a Dual 1920 configuration) MOS Random Access Memory modules of 2048 bytes each which contain all the memory necessary for the refresh subsytem.

4.1. I/O Command Set

For programming purposes the refresh subsystem is treated as if it were a peripheral device and is always referenced as device address 8.

The commands that are used to control the refresh subsystem are, Control I/O commands (CIO), Read I/O commands (RIO), Write I/O commands (WIO), and Test I/O commands (TIO). Each of the commands are summarized in table 4-1 and given a more thorough treatment in the following discussions.

Table 4-1 Refresh Instruction Set

Function Code	CIO	RIO	WIO	TIO
0,8	Mode Select	Read Data No Change Pre-fetch	Write Data; No Change Pre-fetch	----
1,8	Enable Cursor	Read Data; Inc Address Pre-fetch	Write Data Inc Address Pre-fetch	-----
2,8	Disable Cursor	Read Data Dec Address Pre-fetch	Write Data Dec Address Pre-fetch	Refresh Busy?
3,8	Load MPR Lsb	Read MPR Lsb	-----	-----
4,8	Load MPR Msb	Read MPR Msb	-----	-----

Function Code	CIO	RIO	WIO	TIO
5,8	Reset Dsb Cursor, Sel Scrn 0, Turn off Scrn Dsply	Read Pre-fetch	-----	----
6,8	----	-----	-----	----
7,8	Select Screen	-----	------	----
8,8	----	-----	------	Refresh Present (ACK)

Figure 4-1
SPD 10/25 Refresh Subsystem

To Be Supplied

4.2. Functional Description

The basic structure of the refresh subsystem is shown in figure 4-1. Addressing any one of the possible 2 (960 character, 2K half-screen mode) display stations, and transmitting the information to be displayed at the stations is under complete program control. The display stations are tied directly to the Memory Pointer Register (MPR). The desired display is addressed by setting the appropriate bit pairs in the MPR. Information can then be written to or read from the currently selected screen memory by selecting one of the WIO/RIO 0, WIO/RIO 1, or WIO/RIO 2 instructions. The MPR points to a specific location in the screen's RAM, which determines where on the screen the data is to be displayed. Characters in the RAM are painted onto the selected screen by a procedure called refreshing. The hardware implements this by automatically loading internal line buffers in preparation for the refreshing process.

Screens are divided into screen pairs. For the full screen Dual 1920 configuration, (2 screens with 30 lines of 64 characters or 2 screens with 25 lines of 80 characters), each display pair screen has its own refresh memory bank. There is, in addition to the 1920/2000 displayable character area (30 X 64 = 1920 80 X 25 = 2000), a 128/48 character buffer immediately following the screen's refresh area in the refresh memory bank. Although undisplayable, this buffer is available to the software program, making the total individual screen area in this mode to be 1920 + 128 = 2048 or 2000 + 48 = 2048 characters.

For the half-screen configuration, there is a 960 character displayable area for each of the possible 4 screens; each of the 2 memory banks is shared by two display screens. In this mode, the screens can have either 12 lines of 80 characters or 15 lines of 64 characters. There is a 64 character non-displayable buffer immediately following each screen's refresh area which is available to the software program, making the total individual screen refresh area to be 960 + 64 = 1024 characters.

Since 1 or 2 memory banks may be installed depending upon the system's configurations, it is possible to attempt to reference non-installed memory by selecting and either reading from or writing to the banks associated with the non-installed memory. Any attempt to read via those banks will yield a zero in the ACR.

When the TPU is initially powered-on, all screens and the cursor are disabled. The first CIO Select command enables all screens. There is no power-save function for the refresh memory banks. Therefore, when the system is turned on, all screen memories will contain random data.

There is no 'End-of-Data' attribute, and therefore, the entire 2000/1920/960 character positions on the screen must be

displayed.

Centering, or otherwise manipulating and editing data on the display screen, is completely under the control of the software program and is accomplished by merely placing the necessary data in the appropriate locations within the screen's refresh memory area.

4.3. Refresh Busy Condition

While the refresh subsystem is automatically loading the line buffers in preparation for refreshing the display screens, refresh is considered 'busy'. While the line buffers are being loaded, the following I/O instructions should not be used as they will be totally ignored by the TPU (a Command-Reject condition is said to have occured):

* CIO 1 Enable Cursor
* CIO 2 Disable Cursor
* WIO 0 Write
* WIO 1 Write Increment
* WIO 2 Write Decrement
* RIO 0 Read
* RIO 1 Read Increment
* RIO 2 Read Decrement
* RIO 5 Read Pre-Fetch

4.3.1. TIO 2, Refresh Busy Command

To determine if a refresh busy condition exists, a TIO 2 (Refresh Busy) command may be used. However, the TIO Refresh Busy command will actually acknowledge that refresh is busy 14 TPU cycles (22.4 microseconds) before the line buffers begin to be loaded. Therefore, if the refresh is not busy, there is always at least 14 CPU cycles in which I/O instructions can be executed before the refresh becomes busy. Loading of the line buffers requires 128 microseconds. The TIO 2, Refresh Busy, tests for a total of 128 + 22 = 150 microseconds. Although refresh is busy for 150 microseconds, the command reject condition applies only to the 128 microseconds required for loading the line buffers.

The JTACK instruction should be the TIO 2 used to test for a refresh busy condition. Use of the JFACK instruction does not guarantee valid I/O processing. Interrupts are enabled when a TIO instruction causes a jump. Therefore, it is possible that an interrupt could occur if a branch to a refresh I/O request is caused by the JFACK instruction. If this happens, there would be no guarantee that the refresh would be ready upon the return from the interrupt processing. However, a JTACK instruction jumps only when refresh is busy, thus ensuring that interrupts will be disabled during the I/O request.

The following code is recommended for testing a refresh busy condition:

```
JTACK  2,8,$     Refresh Busy?
WIO    0,9       Write Data Into RAM
```

If a non I/O instruction is required after the JTACK, but preceding the I/O instruction, a DSB must be used to disable interrupts with a subsequent ENB after the last I/O instruction.

4.3.2. Cursor Enabled Mode

When the cursor is enabled, the execution of the loaded program is slowed by 29.3%. This is the time required for the refresh subsystem to make comparisons with the CUR and display the cursor at the proper location. These comparisons do not add to refresh time, but cause program execution time loss.

When the cursor is enabled, the following occur:

* The program is suspended during refresh,
* Auto-exec is halted during refresh,

  and,

* TIO 2 , Refresh Busy, is NOT necessary before any instruction except a CIO 1, Cursor Enable, and CIO 2, Cursor Disable.

When the cursor is disabled, program exection time returns to normal, auto-exec time returns to normal, and TIO 2 instructions must be issued where appropriate.

4.4. CIO 5, Command Reset

This command is equivalent to an IOR except that its effect is limited to the refresh subsystem. The CIO 5 instruction:

* Disables the cursor,
* Selects screen 0 for cursor display
* Causes the video to all screens to be turned off. The screen data memory is unmodified.

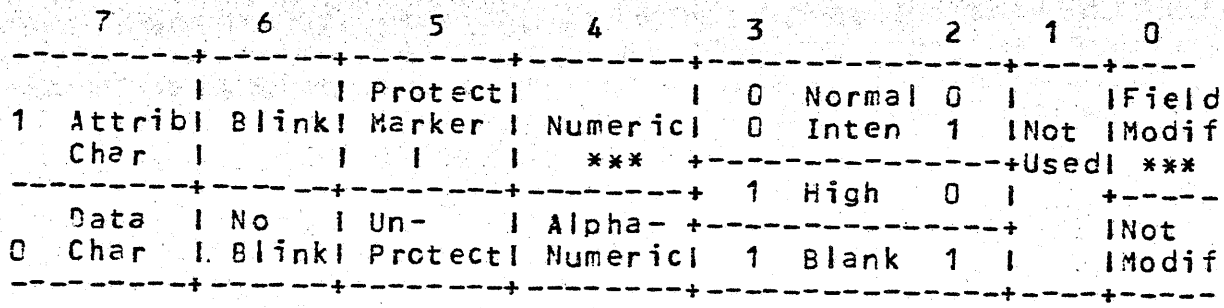
4.5. Attribute Characters

Attribute characters are non-printable control codes which define the characteristics of the display field that follows. Hardware controlled conditions include two levels of brightness (normal and high intensity), field blinking, and blanking. The field is defined as the attribute character position plus all the data following it up until the next attribute character or the end of the screen if no attribute character follows.

The effect of an attribute, however, doesn't wrap from the end of the screen back to the start of the screen. The attribute character itself is not normally displayable, but with an available system option, the software can allow the attribute character to be displayed as a vertical line (|) character at the attribute character position. When displayed, the vertical line is subject to the same parameters the attribute character specified, such as brightness, blinking, or blanking. Each time a character in the screen memory equal to or greater than Hex 80 is detected, the refresh subsystem will consider it to be an attribute character and all the following alphanumeric characters will be modified appropriately.

Blinking is performed at the rate of 1.88 cycles per second and may be done in either normal or high intensity mode. At the first position of the display, the screen is always unblanked, non-blinking, and normal intensity, since a character can not be placed before it. The format for the specification of an attribute is given below:

Entry In Refresh RAM Byte:



***Note: Software convention bits may be tested or set by program for the conditions (arbitrary) indicated.

4.6. Screen Initialization

Before any operation on the refresh subsystem can be performed, the program must issue certain initialization commands to configure the subsystem into a known mode and to specify to which screen the I/O operations are to be performed and the cursor displayed. These are the functions of the CIO 0, Mode Select, and CIO 7, Screen Select commands, which are described in the following sections.

4.6.1. CIO 0, Mode Select Command

The functions of the bit values are as follows. Bit zero indicates whether the system has up to four screens of 960 characters each or up to two screens of 1920/2000 characters each.

Bit 1 indicates if the system is to have an 80 character line or a 64 character line. Bit 2 indicates if the Protect Field Marker is enabled or disabled. This bit allows or inhibits the displaying of a vertical slash (|) where an attribute protected field code exists, except when the code indicates blanking.

Bit 3 indicates if the visible attribute character actions, blinking, blanking, and high intensity to be either enabled or disabled. If these attribute activities are disabled, then the attribute code bits may be used as software flag; that screen position will be blanked.

Bit 4 specifies execution of the Bisync cyclic check function if 1, or PARS cyclic check if 0. The cyclic check generator is

described in section 8. Bit 5 through 7 are not used.

When the system is initially powered-up, the refresh subsystem is not in a known mode. Additionally, the current mode cannot be sensed by the software program. Therefore, the mode select instruction, CIO 0, must be issued by the program at least once to configure the system into a known mode.

BIT:

	7-5	4	3	2	1	0
1	Not Used	Bisync Cyclic Check	DSB Attribute	ENB Field Marker	80 Char	Half Screen
0		PARS Cyclic Check	ENB Attribute	DSB Field Marker	64 Char	Full Screen

4.6.2. CIO 7, Screen Select Command

The refresh subsystem contains 1 cursor register. Since there may be up to four screens, this instruction is used to specify on which screen pair the cursor will be displayed.

Once a screen pair has been selected, all subsequent cursor commands will be directed to that pair. To direct a cursor command to a screen pair other than the one currently selected, it is necessary to issue a new select command to the desired pair. A one cycle delay (1.6 microseconds) is required between successive CIO 7 Select and/or IOR commands. A NOP instruction can be used to wait out the one cycle delay. The issuance of either a CIO Reset or an IOR resets the selected screen to 0. Cursor values are not changed by either one of these instructions. The value in the ACR at the time the select command is executed defines the selected pair as shown below.

7	5	4	3	0
0	Not Used	Screen 0	Not Used	
1		Screen 1		

Cursor display on a particular screen within a screen pair is determined by the range of RAM memory addresses set in the CHR.

In the full screen mode ALL of a RAM memory module is dedicated to a particuar screen, and their are no screen 'pairs'.

In the half screen mode, each RAM can address both screens of a particular screen pair. This is illustrated in table 4-2:

Table 4-2 CUR and RAM Screen Addresses

Screen Cnfg	Screen Pair	Screen	RAM Range	CUR Range
Full 1920/2000	0	--	0000-07FF	0000-07FF
	1	--	1000-17FF	0000-07FF
Half 960	0	0	0000-03FF	0000-03FF
		1	0400-07FF	0400-07FF
	1	0	1000-13FF	0000-03FF
		1	1400-17FF	0400-07FF

4.7. CIO 1/2 Enable/Disable Cursor

The determination as to whether the cursor is to be visible on the screens is programmable via the CIO 1, Enable or CIO 2, Disable Cursor command. These commands do not affect the data displayed on the screen. When the cursor is enabled, it slows program execution by 29.3% (see section 4.3.2.).

The cursor is a rectangular blob which may be positioned anywhere on the display. When positioned over a character, it obliterates it but does not wipe it from the RAM. It should be noted that the position of the cursor is completely unrelated to the positions of data in the RAM.

Since the CUR is frequently used as a 12-bit data register, it is a good idea to limit non-display usage, when the cursor is enabled, to prevent 'ghost' cursors from haunting the screen. Either the cursor must be disabled, or only off-screen values should be loaded.

4.8. Setting the Cursor

The cursor is set by loading the CUR with the desired display position. If the display is operating in 64 char/line mode, the LIR field corresponds to the line position and the CHR corresponds to the character position where the cursor will be displayed, within the screen pair selected by the CIO 7, Select Screen, instruction. See table 4-2 for the address range of the particular screens within the selected screen pair, on which the cursor will be positioned.

If the refresh is operating in the 80 char/line mode, an absolute address must be calculated and loaded into the CUR to position the cursor to a relative line/column location.

This is achieved as follows:

* Select the screen pair on which the cursor is is to be displayed via the CIO 7 instruction, (in 4K RAM configurations only. i.e., Dual 1920)

* Select the starting address of the screen within the screen pair on which the cursor is to be displayed. This will be the base used in the calculation (for example, X'0400' is the starting address of screen 1, 0000 is the starting address of screen 0).

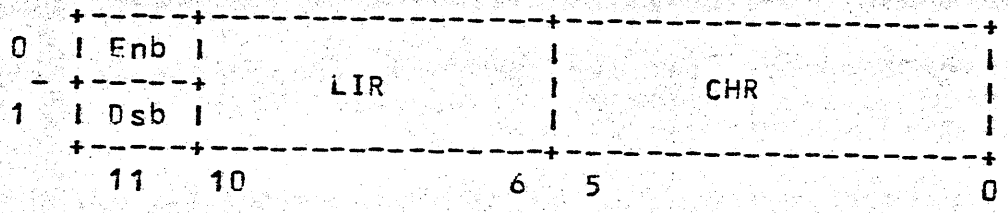
* Compute the absolute CUR location corresponding to the given line/character (where line/char are numbered from 1):

Base + 80 X (line) + (char) = CUR setting

To cause the cursor to appear on screen 1 of screen pair 0, in a half-screen configuration, at char position 5, line position 2, the following is coded:

```
LDI    X'00'              Select screen 0
CIO    7,8                ***
LDC   =80*(2)+(5)+X'400'  Compute absolute position
JTACK  2,8,$              Wait for refresh
CIO    1,8                Enable cursor
```

The CUR is a 12-bit register in the following format:



Bits 0 through 10 contain the absolute value of the cursor within its own refresh memory area. Legal values are 000 through Hex 7FF for 1920 character per screen configurations, and Hex 000 through Hex 3FF and Hex 0400 through Hex 7FF for 960 character per screen configurations.

Bit 11 is the enable/disable bit. If bit 11 is a 1, the cursor is displayed at an address greater than the the legal value (e.g. effectively disabled) If bit 11 is a 0, the cursor is can reference a legal address and is effectively enabled. This is an alternative to issuing a CIO 1/2, Enable/Disable cursor instruction, though the actual mechanism is different.

The cursor may be manipulated with the following instructions:

INC $C	Increment cursor by one
IN2 $C	Increment cursor by two
DEC $C	Decrement cursor
MLA	Move LIR to ACR
MAL	Move ACR to LIR
MCA	Move CHR to ACR
MAC	Move ACR to CHR
CLC	Clear CHR to 0
CLL	Clear LIR to 0

Note that the MAL, MLA, MCA, MAC, CLL, and CLC instructions, in 64 char/line full screen mode, act correspondingly on the line and character position of the cursor.

4.9. Setting the MPR

The MPR is a 16 bit register used to determine the screen location of displayed characters. In the 64 char/line mode, the

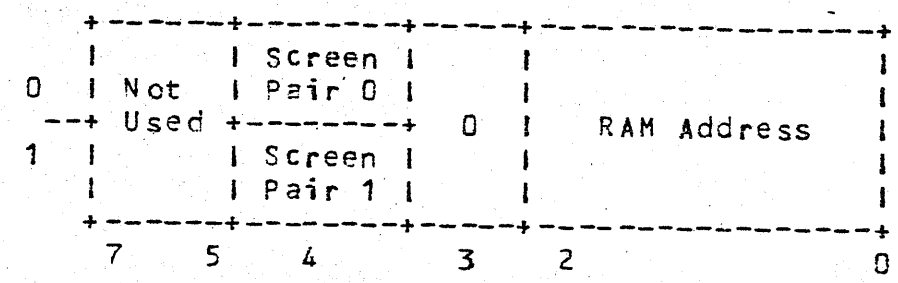
MPR is set to an absolute address within the screen RAM, unlike the CUR register which employs line/char addressing.

In 80 character mode, the ROM converter (described in section 5.) can be used to set the MPR on a line/char basis. This is the ONLY instance when the MPR is set this way.

Two instructions are used to set the MPR. They are the CIO 3 and CIO 4 instructions and set the MPR Lsb and Msb 8 bits respectively.

4.9.1. CIO 4: Set MPR Msb

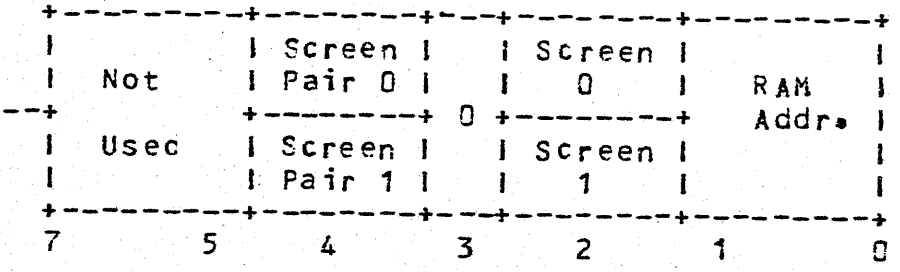
This instruction transfers the contents of the ACR to the 8 Msb of the MPR. When addressing Full screen (1920/2000) configurations the MPR Msb has the following format:



Bit 5 through 7 are not used. Bit 4 gives RAM addresses greater than 7FF when on (these are addresses in screen pair 1).

Bit 3 is set off, and bits 0 through 2 give the address of a location within a screen pair.

When addressing half-screen (960) configurations, the following format is used:

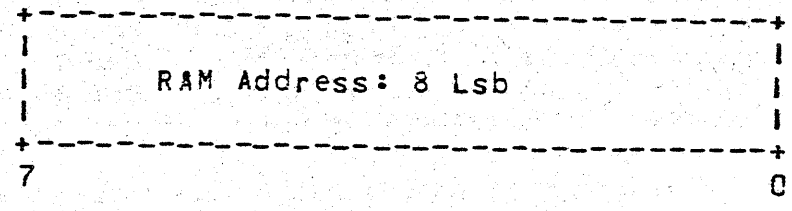


Bits 5 through 7 are not used. Bit 4 allows addresses greater than 7FF when on, thus selecting screen pair 1. Bit 3 is set to 0. Bit 2 allows addresses greater than 3FF when on, thus selecting a screen within the screen pair.

Bits 0 through 1 give the Msb address of a location within a selected screen.

4.9.2. CIO 3: Set MPR Lsb

This instruction moves the contents of the ACR to the 8 Lsb of the MPR. The MPR Lsb has the same format regardless of configuration:



4.9.3. Using the CUR to Set the MPR

Ordinarily, setting the MPR on an absolute basis requires the programmer to translate a line/char address in a manner similar to that described in section 4.8. for the CUR. However, in many 64 char/line applications, the CUR can be used to set the MPR on a line/char basis. This requires that the CUR be dedicated to this purpose, or some mechanism for saving the contents of the CUR be provided.

The call of the subroutine is:

```
     JSR   SETMPR        Entry: (CUR) CHR: Character position
                                LIR: Line position
                                (ACR)     Irrelevant

                         Exit:  (ACR):     Destroyed
                                (CUR):     Saved

SETMPR  DAC   **          Save return point
        STC   WORK        Store Cursor contents at work loc
        LD    WORK+1      Load ACR with Lsb
        CIO   3,8         Set MPR Lsb
        LD    WORK        Load ACR with Msb
        CIO   4,8         Set Mpr Msb
        JMP*  SETMPR      Return to caller

WORK    DAC   **          Location used to set MPR
```

4.10. Write I/O Commands

The Write I/O commands and their functions are described in the

following sections.

4.10.1. WIO 0 Write Data; No Address Change

This command causes the data in the ACR to be written to the currently selected screen's refresh memory at the current MPR address. If this address is within the screen's displayable refresh area, is not within a blanking attribute protected field and if the screens are enabled, then the character will also be displayed on the selected screen. The MPR address remains unchanged after execution of the instruction; it is left pointing to the address of the last written character. This command is always acknowledged but will be ignored during a refresh-busy condition.

A one cycle delay is required between successive WIO 0 and/or RIO 0, RIO 1, RIO 2, RIO 5, WIO 1 and WIO 2 commands. A NOP instruction may be used to wait out the required delay.

After the data is written to the screen's RAM, a read cycle is perfcrmed at the address contained within the MPR. This is equivalent to a program-issued RIO 5, Read Pre- Fetch instruction.

4.10.2. WIO 1 Write Data; Increment Address

This command functions in the same manner as the WIO 0 command (descibed in section 4.10.1.) except that the MPR address is incremented by one following execution of the instruction. Therefore, the MPR address is left pointing to one position beycnd the last written character. A one cycle delay (1.6 microseconds) is required between two successive WIO 1 and/or RIO 0, RIO 1, RIO 2, RIO 5, WIO 0 or WIO 2 commands. A NOP is inserted between the instruction pairs to wait out the one cycle delay. The WIO 1 command must not be issued while a refresh-busy condition exists. A read pre-fetch is performed at the address contained in the MPR after the intruction has been executed.

4.10.3. WIO 2 Write Data; Decrement Address

This command is similar to the WIO 1 command except that the MPR address is decremented by one following execution of the instruction. The MPR is left pointing to the address immediately preceding the last written character.

A one cycle (1.6 microseconds) delay is required between two successive WIO 2 and/or RIO 0, RIO 1, RIO 2, RIO 5, WIO 0 or WIO

1 commands. A read pre-fetch is performed at the address contained in the MPR, after the instruction has been executed. This instruction must not be issued while a refresh-busy condition exists.

4.11. Read I/O Commands

The Read I/O commands and their functions are described in the following sections.

4.11.1. RIO 0 Read Data; No Address Change

This command is always acknowledged but may not be issued while a refresh-busy condition exists. This command causes the controller to transfer data from the currently selected screen at the current cursor location to the ACR. The cursor address is not changed upon completion of the instruction.

This command must be immediately preceded by a RIO 5 Read Pre-Fetch Command (described in section 4.11.4.) if the data read is to be valid. This pre-fetch loads a buffer register with the data to be transferred. The data is loaded into the ACR from the buffer by the RIO Read Data commands.

After this instruction has been exectued, a read pre-fetch is automatically issued at the address contained in the MPR. This eliminates the necessity for issuing RIO 5, Read Pre-fetch commands when performing consecutive reads.

This instruction requires a 1 cycle delay between successive RIO 0 and/or RIO 1, RIO 2, RIO 5, WIO 0, WIO 1 and WIO 2 commands.

4.11.2. RIO 1 Read Data; Increment Address

This command is similar to the RIO 0 command in that it must not be issued during refresh-busy and must be preceded by a RIO 5 Read Pre-Fetch to validate the data being read. In addition, a one cycle delay is required between two successive RIO 1 and/or RIO 0, RIO 2, RIO 5, WIO 0 WIO 1 or WIO 2 commands.

After execution of the RIO 1 command, the MPR is incremented by one and is left pointing to the address immediately following the character read. This command may be used for the sole purpose of incrementing the MPR, and when used for such, need not be preceded by a RIO 5 Read Pre-Fetch command. No valid data can be read under this condition.

After excution of this instruction, an automatic read-prefetch is executed which makes it unecessary to issue a RIO 5 command before the next consecutive read or write instruction.

4.11.3. RIO 2 Read Data; Decrement Address

This command is similar to the RIO 1 command. It must be preceded by a RIO 5 Read Pre-Fetch to validate the data being read, must not be issued during refresh-busy, and must not be issued without a 1 cycle delay after the last RIO 2 and/or RIO 0, RIO 1, RIO 5, WIO 0 WIO 1 or WIO 2 instruction. After execution of the instruction, the MPR is decremented and is left pointing to the position immediately preceding the address of the character read. The RIO 2 instruction can be used for the sole purpose of decrementing the cursor, and when used for such, need not be preceded by a RIO 5 Read Pre-Fetch instruction.

After execution of this instruction, an automatic RIO 5, Read Pre-fetch instruction is issued.

4.11.4. RIO 5 Read Pre-Fetch

This command initiates a data read from the currently selected screen at the current cursor address. It loads a buffer in preparation for the RIO 0, 1 or 2 commands which fetch the data from the buffer and transmit it to the ACR. The RIO 5 command may not be issued during refresh. It must be issued exactly 1 cycle (1.6 microseconds) before an RIO 0, RIO 1 or RIO 2 if the data being read is to be valid. This command is only necessary to validate the data being read, and need not be issued if the RIO 1 or RIO 2 command is being used simply to position the MPR.

The WIO 0, WIO 1, WIO 2, RIO 0, RIO 1 and RIO 2 automatically prefetch data from the MPR setting after the instruction has been executed. Therefore, if a read is to be performed immediately after executing one of the above instructions, a RIO 5 command is not required.

Note that the automatic pre-fetch cycle is valid only if the address of the MPR is not changed by a CIO 3 or CIO 4 command. One use of this feature would be to read data from a particular screen field.

A subroutine to implement this is:

```
JSR   RDFLD     (MPR)    Set to the beginning of field
DAC   BUFFER    (ACR)    Length of field

                BUFFER   Pointer to a location to store field

RDFLD   DAC     **       Save parameter address
        ST      CTR+1    Counter for length of field
        LD*     RDFLD    Area to place read field
        ST      WORK     ***
        INC     RDFLD    Bump return to next byte
        LD*     RDFLD    ***
        ST      WORK+1   ***
        INC     RDFLD    Bump return past parameters
        RIO     5,8      Required pre-fetch
LOOP    RIO     1,8      Read data, inc address, pre-fetch
        ST*     WORK     Store in supplied buffer
        DEC     CTR      Check for end of field
        JCGT    LOOP     Continue reading if not at end
        JMP*    RDFLD    Else, return to caller

WORK    DAC     **       Address of buffer
CTR     WORD    **       Counter for loop control
```

4.11.5. Refresh Size

For 64 character per line configurations, the total refresh memory size of either 2048 or 1024 is an exact multiple of 64. In 1920 character configurations, the total screen memory size of 2048 permits 30 full lines of 64 characters in the displayable refresh memory and 2 full lines of 64 characters in the buffer area as shown in figure 4-2a.

Figure 4-2a

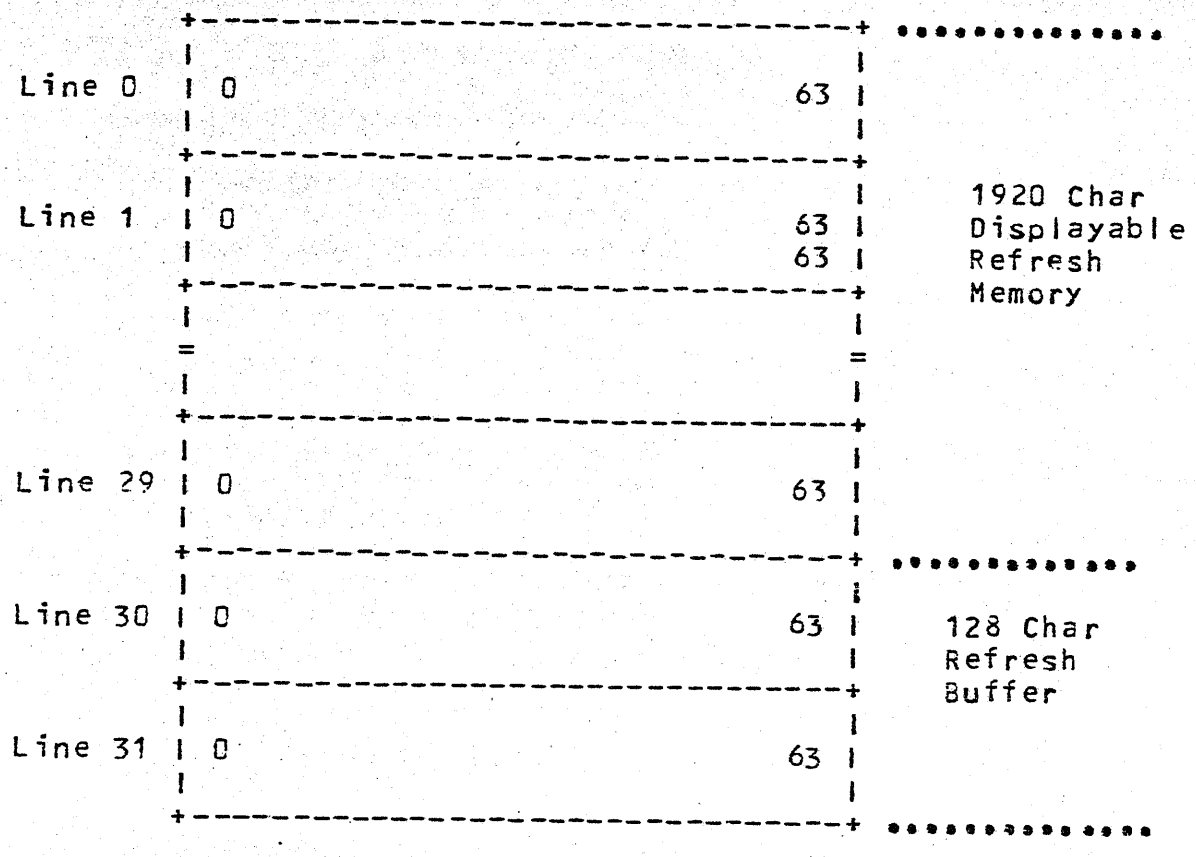

Line 0
0
63
Line 1
0
63
63
1920 Char
Displayable
Refresh
Memory
Line 29
0
63
Line 30
0
63
128 Char
Refresh
Buffer
Line 31
0
63
64 Char/Line
Full-Screen

Figure 4-2b

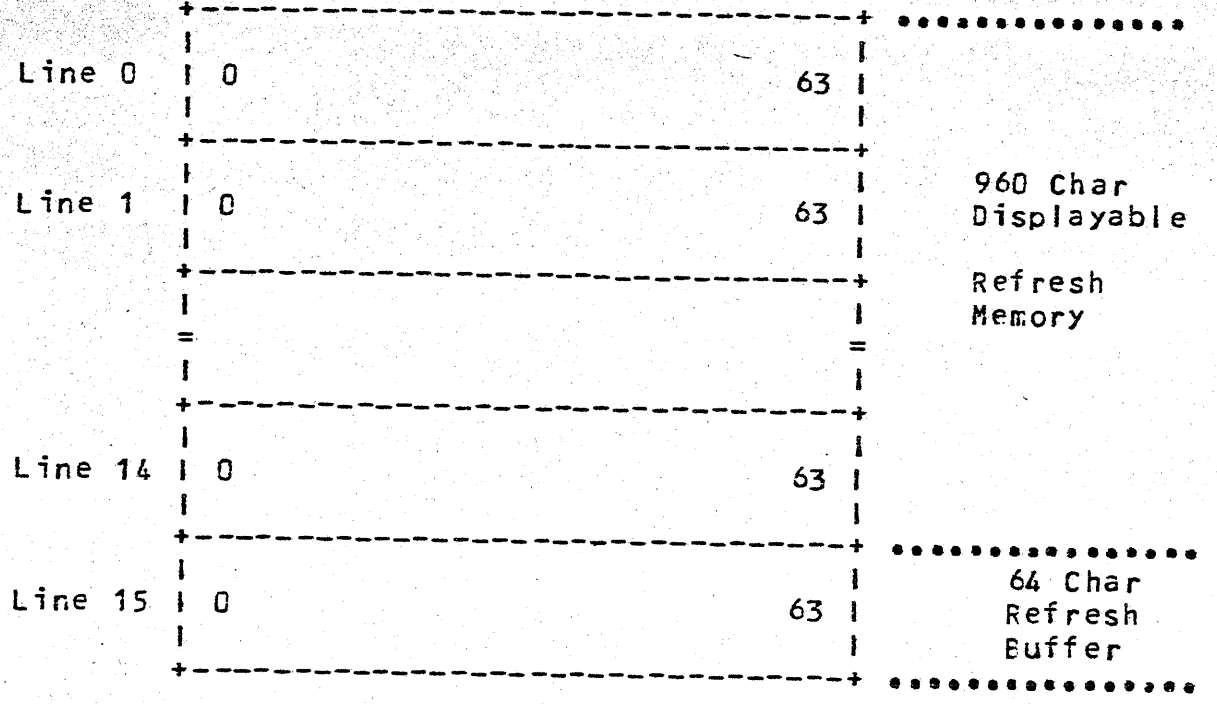
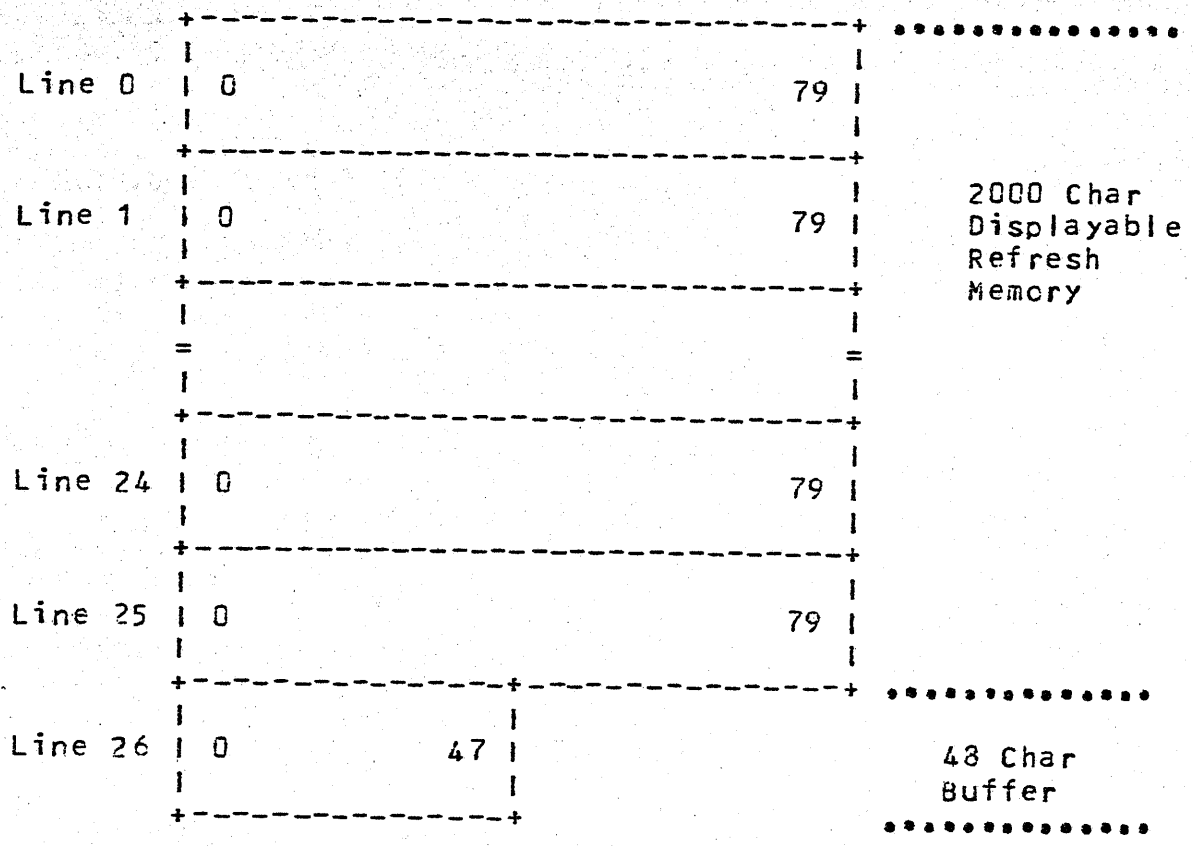

Line 0
0
63
Line 1
0
63
960 Char
Displayable
Refresh
Memory
Line 14
0
63
Line 15
0
63
64 Char
Refresh
Buffer
64 Char/Line
Half-Screen

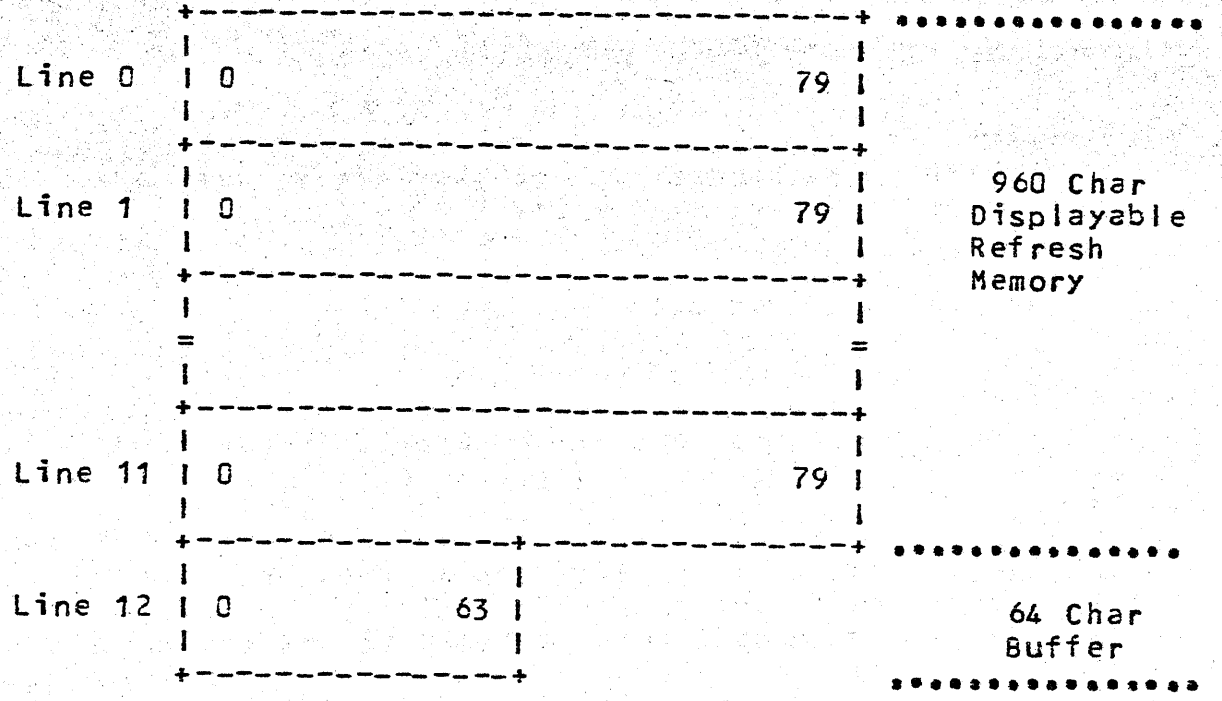
Figure 4-2c



80 Char/Line

Full Screen

Figure 4-2d



In half-screen, 960 character display, systems, the total screen memory size of 1024 permits 15 full lines of 64 characters in the displayable refresh memory and 1 full line in the refresh memory buffer area (See Figure 4-2b). However, in 80 character configurations, the total refresh memory size of 2000 or 1024 is not an exact multiple of 80. Therefore, in full 2000 character display systems, the total screen memory size of 2048 permits 25 full lines of 80 characters in the displayable refresh memory, and one partial line of 48 characters in the refresh memory buffer area (Figure 4-2c).

In half-screen 960 character configurations, the total screen memory size of 1024 permits 12 full lines of 80 characters in the displayable refresh memory and one partial line of 64 characters in the refresh memory buffer area (Figure 4-2d).

4.11.6. RIO 3 Read MPR Lsb

This command is always acknowledged and may be issued at any time, with 64 character per line configurations. The 8 Lsb of the MPR are transferred to the ACR in the same manner as described

for the CIO 3 command in section 4.9.2..

4.11.7. RIO 4 Read Line

This command is the same as the RIO 3 command except that the 8 Msb of the MPR are transferred to the ACR in the same format described in section 4.9.1. for the CIO 4 command.

4.12. Illegal Line/Character References

In all 64 character per line configurations and in 80 character per line configurations employing absolute addressing, it is impossible to attempt to reference an illegal screen memory location.

However, in 80 character per line configurations, when addressing memory using the line/character address mode, (ROM Code Converter), it is possible to attempt to reference an illegal character or an illegal line. In full screen configurations, a character reference greater than 47 on line 26 is illegal. In the half screen mode, a character reference greater than 63 on line 12 is illegal.

In the line/char addressing mode, the hardware uses bits 0 through 4 when loading the ROM in full screen configurations. This permits an attempt to reference up to 32 lines; the legal limit is 26 lines. In the half-screen mode, the hardware uses bits 0 through 3 of the ACR, permitting an attempt to reference up to 16 lines when the legal limit is 12.

In addition, mode, when using line/character addressing, the line and character addresses are not directly tied to the MPR as they are in the 64 character mode or in the absolute addressing mode. Instead, the values loaded into the line and character registers are converted to an absolute memory location by the hardware Read Only Memory (ROM) look-up table. If the ROM detects a line/character combination that would reference either an illegal line or character, that reference is converted by the ROM to a legal location in memory in accordance with the rules described below, and the line and character registers are updated to reflect the new reference.

The basic rules are as follows:

1. For 2000 character screen configurations, any character reference from character 48 through 79 inclusive on line 26, or any character greater than 79 on any line, or any line reference greater than 26 will be converted to one of the last 16 locations in the screen's buffer (not displayable).

Which of the 16 locations is used depends on the value of the least significant Hex digit in the character register at the time of the the illegal reference. (e.g. character reference = 4F: location used is set by F).

2. For 960 character screen configurations, any character reference from 64 through 79 inclusive on line 12 or any character reference greater than 79 on any line will be converted to one of the last 16 locations in the screen's buffer memory (not displayable). Which of the 16 locations is used is determined as described above. Any line reference greater than line 12 will be wrapped back up into the screen's visible refresh memory area.

Note: Because of the possibility that one of the 16 locations of the buffer memory may be altered inadvertantly by an illegal character reference, it is recommended that these 16 locations not be used by the software program in 80 character per line configurations when the line/character addressing mode is used.

4.13. TIO 8: 10/25 System Identification

This instruction is always ACKnowledged in the SPD 10/25. It is used by the SPD/DOS nucleus to distinguish the SPD 10/25 from the SPD 10/20 in DOS 'compatible' programs.

5. ROM Code Converter

The ROM code converter function is contained in the refresh subsystem. It is addressed as device 8. The code converter implements a translate table look-up function utilizing 16,384 bit static ROM.

Eleven differnet tyoes of translate-tables are available:

1. Line/Char to Absolute (80 char/line only)
2. Absolute to Line/Char (80 char/line only)

3. ASCII to EBCDIC .....
4. EBCDIC to ASCII
5. BCD to ASCII
6. ASCII to BCD — General Code
7. BAUDOT to ASCII — Conversion Tables
8. ASCII to Baudot
9. Shift Right 4
10. Rotate Right 1
11. Decimal Multiply .....

In addition, definitive ROM addresses are given for all ASCII character representations, as shown in table 5-2. This table also shows the correspondence between ASCII and EBCDIC representations.

5.1. Code Converter Instruction Set

The input word to be translated is loaded into the MPR via the CIO 3 and CIO 4 instructions. A CIO 8 instruction is then used to initiate the code translation cycle. A one cycle delay is required before reading the translated data. The RIO 7 and RIO 8 commands are used to extract the translated data.

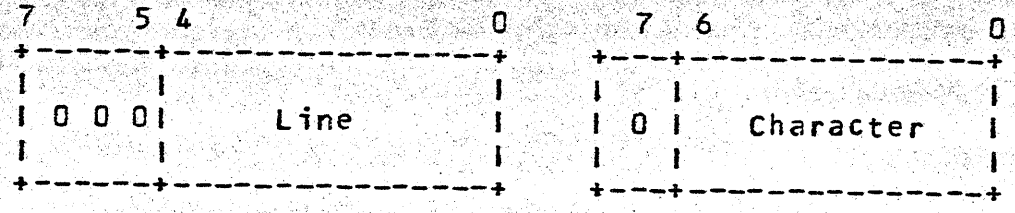
ROM Code Converter Instruction Set

Function Code	CIO	RIO
3,8	Load MPR 8 Lsb	----
4,8	Load MPR 8 Msb	----
7,8	-----	Read Code Converter LSB
8,8	Execute Code Conversion	Read Code Converter MSB

5.2. Line/Character to Absolute Translation

This translation table yields the absolute RAM location of each line/character input. The line/character designation is readily related to a physical point on the display screen while the data displayed at that point is always stored in the MPR as an absolute address. This table simply yields the equivalent RAM address for each line/character position on the display screen. This translation is valid for 80 character screen formats only. The MPR must remain unchanged until after the RIO 7 and RIO 8 commands are issued for this table.

The line/character to absolute translation data input format is shown below:



Ai

CIO 4,8

Bi

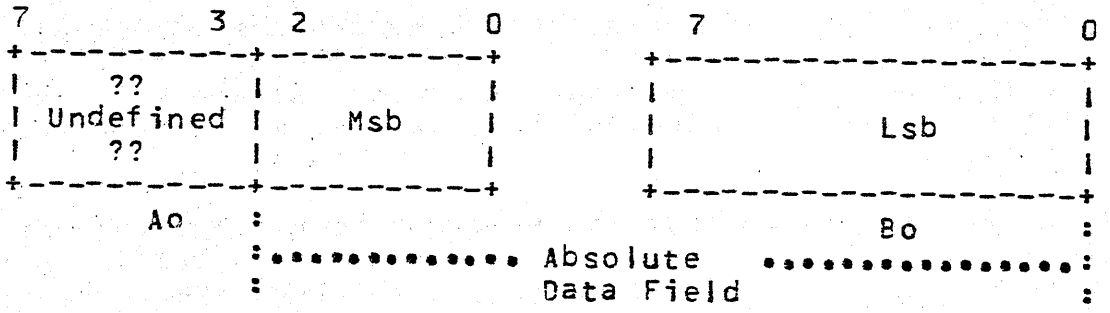
CIO 3,8

Byte Ai is stored in the MPR Msb via the CIO 4 instruction. Bits 5 through 7 must be zero. The valid range of line values are 00 through 1A hex.

Byte Bi is stored in the MPR Lsb via the CIO 3 instruction. Bit 7 must be 0. The valid range of character values are 00 through 4F hex.

5.2.1. Output Data Format: Absolute

After executing the conversion, the data is stored in the MPR in this format:



Byte Ao (Msb) is transferred to the ACR via the RIO 8 command. Bits 3 through 7 are undefined and should be masked by the program. Bits 0 through 2 contain the three Msb of the absolute data field.

Byte Bo (Lsb) is transferred to the ACR via the RIO 7 command. Byte Bo comprises the 8 Lsb of the absolute data field.

5.2.2. Example

This example illustrates the operation of the Line/Char to Absolute Code converter, and of the ROM Code Converter in general.

It is necessary to traslate any line/char position into an absolute position and set the MPR. The following subroutine is one possible solution.

```
  JSR   STMPR
  BYTE  Row,Col

STMPR   DAC   **          Parameter location
        LD*   STMPR       Fetch row position
        CIO   4,8         Transfer to MPR Msb
        INC   STMPR       Move to column position
        LD*   STMPR       Fetch column position
        INC   STMPR       Bump return past parameter
        CIO   3,8         Transfer to MPR Lsb
        CIO   8,8         Execute Code Conversion
        NOP               Wait one required cycle
        RIO   8,8         Read Msb Code Converter Output
        ANI   X'07'       Mask out garbage values
        CIO   4,8         Set MPR Msb
        RIO   7,8         Read Lsb Code Converter Output
        CIO   3,8         Set MPR Lsb
        JMP*  STMPR       Return to caller
```

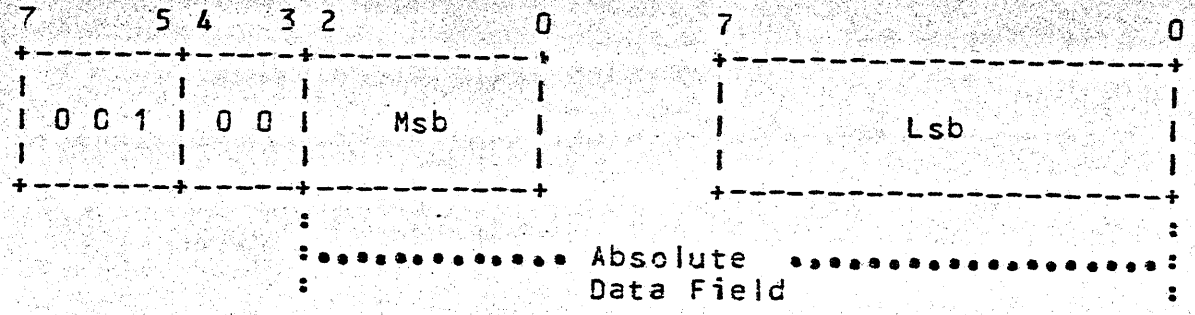
5.3. Absolute to Line/Char Traslation

This translation table yields the line/character equivalent of each absolute refresh RAM input. The purpose of this translation is to relate each absolute location of the screen RAM memory to the line/character location on the screen at which that character is displayed.

This table is merely the inverse of the line/character to absolute translation described in section 5.2. and is valid for 80 character/line screen formats only. The MPR must remain unchanged until after the RIO 7 and RIO 8 commands are issued for this translation table.

5.3.1. Input Data Format (Absolute)

The absolute to line/character input data format is described as follows:

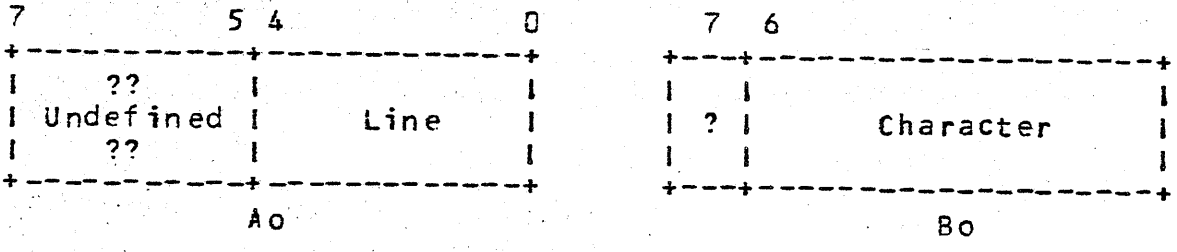


Byte Ai is stored in the MPR Msb via the CIO 4 instruction. Bits 3, 4, 6, and 7 of byte Ai must be zero. Bit 5 must be one. Bits 0 through 2 of byte Ai comprise the three Msb of the absolute data field.

Byte Bi comprises the eight Lsb of the absolute data field. The range of the absolute data field is 00 through 07FF.

5.3.2. Output Data Format (Line/Char)

The output format of the ROM converter is as follows:



Byte Ao is transferred to the ACR via the RIO 8 command. Bits 5 through 7 of byte Ao are undefined and should be masked by the program.

Byte Bo is transferred to the ACR via the RIO 7 command. Bit 7 of byte Bo is undefined and should be masked by the program.

5.4. General Code Conversion Tables

The nine general code conversion tables and the input required to implement them, are shown in the following table:

Table 5-1
General Code Conversion Translation Table

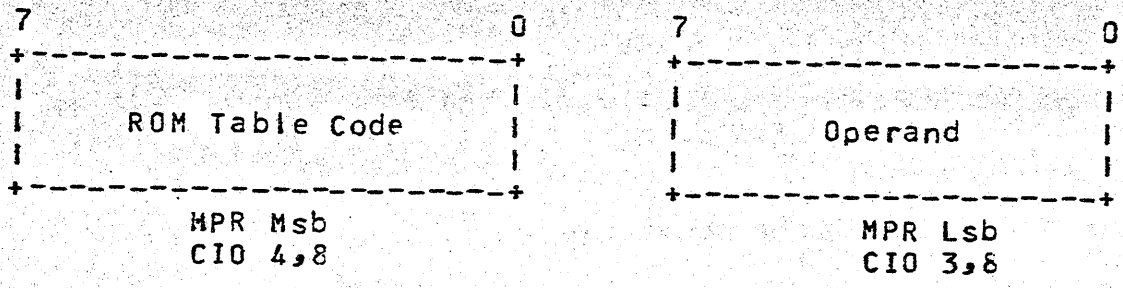
Table No.	Description	MPR Input Msb (HEX)	Lsb
5-2	ASCII - EBCDIC	40	ASCII Character
5-3	EBCDIC - ASCII	60	EBCDIC Character
5-4	ASCII - BCD	80	ASCII Character + X'20'
5-5	BCD - ASCII	80	BCD Character
5-6	ASCII - Baudot	E0	ASCII Character + X'A0'
5-7	Baudot - ASCII	80	Baudot Character + X'80'
5-8	Shift Rt Four	A0	Character to be Shifted
5-9	Rotate Rt One	C0	Character to be Rotated
5-10	Decimal Mult.	F0	Two operands: 0-9

5.4.1. General Data Formats

The input data formats for the general code conversion tables are similar. Data is input to the ROM converter via the MPR CIO 3 and CIO 4 commands. While the MPR Msb is given explicitly in table 5-1, the Lsb input is described separately for each table.

The translation result is transferred to the ACR via the RIO 7 command. In the general code conversion tables, only one byte (Lsb) is of interest (RIO 8 not required).

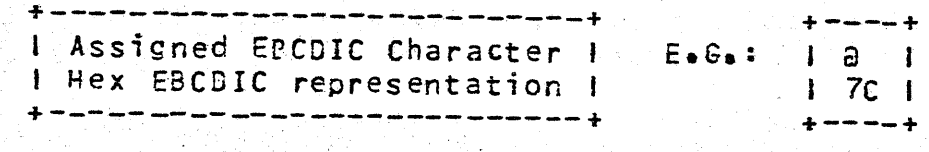
Input Data Format



5.4.2. ASCII to EBCDIC

Table 5-2 is the ASCII to EBCDIC Code conversion table. The MPR Lsb is set to the ASCII representation of the character to be translated.

Each entry in table 5-2 is given in the format:

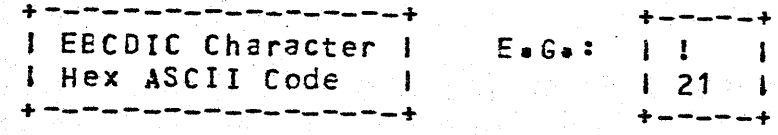


The table starts at MPR address 4000 hex, and ends at address 40FF hex.

5.4.3. EBCDIC to ASCII

Table 5-3 depicts the EBCDIC to ASCII translations.

Each entry is given in the form:



The table starts at MPR address 6000 hex, and ends at address 60FF.

5.4.4. ASCII to BCD

Table 5-4 depicts the ASCII to BCD code translations. This table begins at MPR address 8020 hex. Thus, it is necessary to set the MPR Msb to 80 and to add 20 hex to the Lsb ASCII character portion of the MPR before the ROM translation is initiated.

5.4.5. BCD to ASCII

Table 5-5 depicts the BCD to ASCII code translations. This table begins at MPR address 8000 hex. The MPR Lsb is set to the BCD character and the Msb portion to the start of the table (80).

5.4.6. Baudot Code Conventions

Baudot code representation has certain conventions that the programmer should be aware of.

Baudot code is a 5 bit code in the range of 00 through 1F. 1F is the highest code transmitted between stations. Therefore, a convention has been adopted which allows stations to distinguish the entire character set. The Ltrs convention applies to alphabetic characters. The Figs convention applies to numeric and special characters.

A station transmits a 1F to indicate that the following characters belong to the set defined by the Ltrs convention.

A station transmits a 1B to indicate that the following characters belong to the set defined by the Figs convention.

Thus, a station always knows what type of characters it is transmitting or receiving by keeping track of the current Figs or Ltrs mode.

5.4.6.1. ASCII to Baudot

Table 5-6 depicts the Baudot to ASCII code translations. This table begins at MPR address E0A0. Thus it is necessary to set the MPR Msb to E0 and add A0 to the ASCII character Lsb portion.

The Ltrs/Figs state of the result is bit 5. Thus, the program must send a Ltrs code (1F) if bit 5 is on, and a Figs code (1B) if bit 5 is off.

5.4.6.2. Baudot to ASCII

Table 5-7 depicts the Baudot to ASCII code translations. The table begins at MPR address 8080. Thus it is necessary to set the MPR Msb to 80 and to add 80 to the Baudot character Lsb portion.

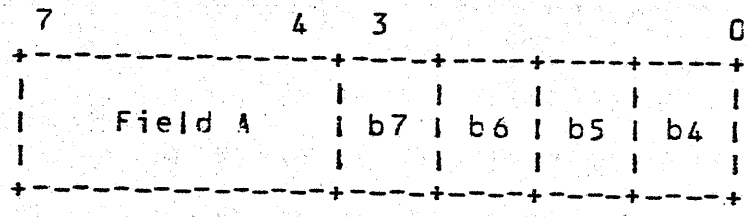
To use this table properly, the software must determine whether the characters to be traslated belong to the Figs or Ltrs set. If

Figs characters are to be translated, no further processing is required. If Ltrs characters are to be translated, 20 hex must be added to the Baudot character before ROM Conversion is initiated.

The output from the code converter is the ASCII representation of the Baudot code.

5.4.7. Shift Right 4 - Special Field A

Table 5-8 depicts the shift right 4/special field A code conversion table. Output from the converter is as follows:



Field A represents the number of bits that were one in the input byte. Bits b4, b5, b6 and b7 are the 4 Msb of the input byte.

The table comprises MPR address locations A000 through A0FF.

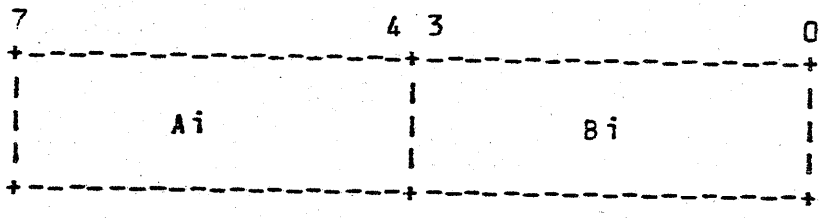
5.4.8. Rotate Right One

Table 5-9 depicts the rotate right one code translation table. Each entry in the table represents the output byte in hex notation. The table starts at MPR address E000 and continues through address E0FF.

5.4.9. Decimal Multiply

Table 5-10 depicts the 9 x 9 decimal multiply code translation.

The input byte is formed as follows.



Ai, Bi: 0-9 decimal digit

The result, via RIO 7, is given in packed decimal form.

Table 5-2: ASCII - EBCDIC

ASCII INPUT 4 LSB (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII INPUT 4 MSB (HEX)
0	NUL 00	DLE 10	SP 40	0 F0	@ 7C	P D7	\ 79	p 97	20	30	41	58	76	9F	B8	DC	
1	SOH 01	DC1 11	! 4F	1 F1	A C1	Q D8	a 81	q 98	21	31	42	59	77	A0	B9	DD	
2	STX 02	DC2 12	" 7F	2 F2	B C2	R D9	b 82	r 99	22	1A	43	62	78	AA	BA	DE	
3	ETX 03	DC3 13	# 7B	3 F3	C C3	S E2	c 83	s A2	23	33	44	63	80	AB	BB	DF	
4	ECT 37	DC4 3C	$ 5B	4 F4	D C4	T E3	d 84	t A3	24	34	45	64	8A	AC	BC	EA	
5	ENQ 2D	NAK 3D	% 6C	5 F5	E C5	U E4	e 85	u A4	15	35	46	65	8B	AD	BD	EB	
6	ACK 2E	SYN 32	& 50	6 F6	F C6	V E5	f 86	v A5	06	36	47	66	8C	AE	BE	EC	
7	BEL 2F	ETB 26	' 7D	7 F7	G C7	W E6	g 87	w A6	17	08	48	67	8D	AF	BF	ED	
8	BS 16	CAN 18	(4D	8 F8	H C8	X E7	h 88	x A7	28	38	49	68	8E	B0	CA	EE	
9	HT 05	EM 19	) 5D	9 F9	I C9	Y E8	i 89	y A8	29	39	51	69	8F	B1	CB	EF	
A	LF 25	SUB 3F	* 5C	: 7A	J D1	Z E9	j 91	z A9	2A	3A	52	70	90	B2	CC	FA	
B	VT 0B	ESC 27	+ 4E	; 5E	K D2	[4A	k 92	{ C0	2B	3B	53	71	9A	B3	CD	FB	
C	FF 0C	FS 1C	, 6B	< 4C	L D3	E0	l 93	| 6A	2C	04	54	72	9B	B4	CE	FC	
D	CR 0D	GS 1D	- 60	= 7E	M D4	] 5A	m 94	} D0	09	04	55	73	9C	B5	CF	FD	
E	SO 0E	RS 1E	. 4B	> 6E	N D5	^ 5F	n 95	~ A1	0A	3E	56	74	9D	B6	DA	FE	
F	SI 0F	US 1F	/ 61	? 6F	O D6	_ 6D	o 96	DEL 07	1B	E1	57	75	9E	B7	DB	FF	

Table 5-3: EBCDIC - ASCII

EBCDIC INPUT 4 MSB (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 00	DLE 10	80	90	SP 20	& 26	- 2D	BA	C3	CA	DI	D8	{ 7B	} 7D	\ 5C	0 30
1	SOH 01	DCI 11	81	91	A0	A9	/ 2F	BB	a 61	j 6A	~ 7E	D9	A 41	J 4A	9F	1 31
2	STX 02	DC2 12	82	SYN 16	A1	AA	B2	BC	b 62	k 6B	S 73	DA	B 42	K 4B	S 53	2 32
3	ETX 03	DC3 13	83	93	A2	AB	B3	BD	c 63	l 6C	t 74	DB	C 43	L 4C	T 54	3 33
4	9C	9D	84	94	A3	AC	B4	BE	d 64	m 6D	u 75	DC	D 44	M 4D	U 55	4 34
5	HT 09	85	LF 0A	95	A4	AD	B5	BF	e 65	n 6E	76	DD	E 45	N 4E	V 56	5 35
6	86	BS 08	ETB 17	96	A5	AE	B6	C0	f 66	o 6F	w 77	DE	F 46	O 4F	W 57	6 36
7	DEL 7F	87	ESC 18	EOT 04	A6	AF	B7	C1	g 67	p 70	x 78	DF	G 47	P 50	X 58	7 37
8	97	CAN 18	88	98	A7	B0	B8	C2	h 68	q 71	y 79	E0	H 48	Q 51	Y 59	8 38
9	8D	EM 19	89	99	A8	B1	B9	` 60	i 69	r 72	z 7A	E1	I 49	R 52	Z 5A	9 39
A	8E	92	8A	9A	[5B	] 5D	¦ 7C	: 3A	C4	CB	D2	E2	E8	EE	F4	FA
B	VT 0B	8F	8B	9B	. 2E	$ 24	' 2C	# 23	C5	CC	D3	E3	E9	EF	F5	FB
C	FF 0C	FS 1C	8C	DC4 14	< 3C	* 2A	% 25	@ 40	C6	CD	D4	E4	EA	F0	F6	FC
D	CR 0D	GS 1D	ENQ 05	NAK 15	(28	) 29	_ 5F	! 27	C7	CE	D5	E5	EB	F1	F7	FD
E	SO 0E	RS 1E	ACK 06	9E	+ 2B	; 3B	> 3E	= 3D	C8	CF	D6	E6	EC	F2	F8	FE
F	SI 0F	US 1F	BEL 07	SUB 1A	! 21	^ 5E	? 3F	" 22	C9	D0	D7	E7	ED	F3	F9	FF

4 LSB (HEX)

EBCDIC INPUT

Table 5-4: ASCII - BCD

ASCII	BCD	ASCII	BCD	ASCII	BCD	ASCII	BCD
20	1C	30	0A	40	1D	50	27
21	3D	31	01	41	31	51	27
22	20	32	02	42	32	52	28
23	00	33	03	43	33	53	12
24	30	34	04	44	34	54	13
25	0E	35	05	45	35	55	14
26	0F	36	06	46	36	56	15
27	10	37	07	47	37	57	16
28	1F	38	08	48	38	58	17
29	2E	39	09	49	39	59	18
2A	0B	3A	2A	4A	21	5A	19
2B	3F	3B	2C	4B	22	5B	0C
2C	3E	3C	2B	4C	23	5C	1B
2D	1A	3D	2F	4D	24	5D	1E
2E	3B	3E	3C	4E	25	5E	2D
2F	11	3F	3A	4F	26	5F	0D

Table 5-5: BCD - ASCII

BCD	ASCII	Displayed Character	BCD	ASCII	Displayed Character	BCD	ASCII	Displayed Character
00	23	#	16	57	W	2B	3C	<
01	31	1	17	58	X	2C	3B	;
02	32	2	18	59	Y	2D	5E	¬
03	33	3	19	5A	Z	2E	29	)
04	34	4	1A	2D	-	2F	3D	=
05	35	5	1B	5C	|	30	24	$
06	36	6	1C	20	(space)	31	41	A
07	37	7	1D	40	@	32	42	B
08	38	8	1E	5D	▶	33	43	C
09	39	9	1F	28	(	34	44	D
0A	30	ø	20	22	≡	35	45	E
0B	2A	*	21	4A	J	36	46	F
0C	5B	◿	22	4B	K	37	47	G
0D	5F	_	23	4C	L	38	48	H
0E	25	%	24	4D	M	39	49	I
0F	26	&	25	4E	N	3A	3F	?
10	27	'	26	4F	O	3B	2E	.
11	2F	/	27	50	P	3C	3E	>
12	53	S	28	51	Q	3D	21	⧖
13	54	T	29	52	R	3E	2C	,
14	55	U	2A	3A	:	3F	2B	+
15	56	V						

Table 5-6: ASCII - Baudot *

ASCII	Baudot	Display	ASCII	Baudot	Display	ASCII	Baudot	Display	ASCII	Baudot	Display
00	00	Not Used	18	00	Not Used	30	16	0	48	34	H
01	00	↑	19	00	↑	31	17	1	49	26	I
02	00	↕	1A	00	↕	32	13	2	4A	2B	J
03	00	↕	1B	00	↕	33	01	3	4B	2F	K
04	00	↕	1C	00	↕	34	0A	4	4C	32	L
05	00	↕	1D	00	↕	35	10	5	4D	3C	M
06	00	Not Used	1E	00	↕	36	15	6	4E	2C	N
07	0B	Bell	1F	00	Not Used	37	07	7	4F	38	O
08	00	Not Used	20	04	Space	38	06	8	50	36	P
09	00	Not Used	21	00	!	39	18	9	51	37	Q
0A	02	LF	22	00	"	3A	0E	:	52	2A	R
0B	00	Not Used	23	14	#	3B	00	;	53	25	S
0C	00	Not Used	24	09	$	3C	00	<	54	30	T
0D	08	CR	25	00	%	3D	1E	=	55	27	U
0E	00	Not Used	26	1A	&	3E	00	>	56	3E	V
0F	00	↑	27	05	'	3F	19	?	57	33	W
10	00	↕	28	0F	(	40	00	@	58	3D	X
11	00	↕	29	12	)	41	23	A	59	35	Y
12	00	↕	2A	00	*	42	39	B	5A	31	Z
13	00	↕	2B	11	+	43	2E	C	5B	00	Not Used
14	00	↕	2C	0C	,	44	29	D	5C	00	↑
15	00	↕	2D	03	-	45	21	E	5D	00	↕
16	00	↕	2E	1C	.	46	2D	F	5E	00	↕
17	00	Not Used	2F	1D	/	47	3A	G	5F	00	Not Used

Table 5-7: Baudot - ASCII *

	Baudot	ASCII	Display	Baudot	ASCII	Display	Baudot	ASCII	Display
	00	00	Null	16	30	0	2B	4A	J
	01	33	3	17	31	1	2C	4E	N
	02	0A	LF	18	39	9	2D	46	F
	03	2D	-	19	3F	?	2E	43	C
	04	20	Space	1A	26	&	2F	4B	K
	05	27	'	1B	FF	Not Used	30	54	T
	06	38	8	1C	2E	.	31	5A	Z
	07	37	7	1D	2F	/	32	4C	L
	08	ØD	CR	1E	3D	=	33	57	W
	09	24	$	1F	FF	Not Used	34	48	H
	0A	34	4	20	00	Null	35	59	Y
	0B	07	Bell	21	45	E	36	50	P
*	0C	0C(2C)	Error(,)	22	0A	LF	37	51	Q
	0D	FF	Not Used	23	41	A	38	4F	O
	0E	3A	:	24	20	Space	39	42	B
**	0F	08(28)	Error(()	25	53	S	3A	47	G
	10	35	5	26	49	I	3B	FF	Not Used
	11	2B	+	27	55	U	3C	4D	M
	12	29	)	28	0D	CR	3D	58	X
	13	32	2	29	44	D	3E	56	V
	14	23	#	2A	52	R	3F	FF	Not Used
	15	36	6						

* The ASCII output is 0C and should be 2C.
**The ASCII output is 08 and should be 28.

Table 5-8: Shift Right Four

$b_3 b_2 b_1 b_0$ INPUT BYTE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	INPUT BYTE 4 MSB (HEX)
0	00	11	12	23	14	25	26	37	18	29	2A	3B	2C	3D	3E	4F	
1	10	21	22	33	24	35	36	47	28	39	3A	4B	3C	4D	4E	5F	
2	10	21	22	33	24	35	36	47	28	39	3A	4B	3C	4D	4E	5F	
3	20	31	32	43	34	45	46	57	38	49	4A	5B	4C	5D	5E	6F	
4	10	21	22	33	24	35	36	47	28	39	3A	4B	3C	4D	4E	5F	
5	20	31	32	43	34	45	46	57	38	49	4A	5B	4C	5D	5E	6F	
6	20	31	32	43	34	45	46	57	38	49	4A	5B	4C	5D	5E	6F	
7	30	41	42	53	44	55	56	67	48	59	5A	6B	5C	6D	6E	7F	
8	10	21	22	33	24	35	36	47	28	39	3A	4B	3C	4D	4E	5F	
9	20	31	32	43	34	45	46	57	38	49	4A	5B	4C	5D	5E	6F	
A	20	31	32	43	34	45	46	57	38	49	4A	5B	4C	5D	5E	6F	
B	30	41	42	53	44	55	56	67	48	59	5A	6B	5C	6D	6E	7F	
C	20	31	32	43	34	45	46	57	38	49	4A	5B	4C	5D	5E	6F	
D	30	41	42	53	44	55	56	67	48	59	5A	6B	5C	6D	6E	7F	
E	30	41	42	53	44	55	56	67	48	59	5A	6B	5C	6D	6E	7F	
F	40	51	52	63	54	65	66	77	58	69	6A	7B	6C	7D	7E	8F	

Table 5-9: Rotate Right One

INPUT BYTE 4 LSB (HEX)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
1	80	88	90	98	A0	A8	B0	B8	C0	C8	D0	D8	E0	E8	F0	F8
2	01	09	11	19	21	29	30	39	40	49	50	59	60	69	70	79
3	81	89	91	99	A1	A9	B1	B9	C1	C9	D1	D9	E1	E9	F1	F9
4	02	0A	12	1A	22	2A	32	3A	42	4A	52	5A	62	6A	72	7A
5	82	8A	92	9A	A2	AA	B2	BA	C2	CA	D2	DA	E2	EA	F2	FA
6	03	0B	13	1B	23	2B	33	3B	43	4B	53	5B	63	6B	73	7B
7	83	8B	93	9B	A3	AB	B3	BB	C3	CB	D3	DB	E3	EB	F3	FB
8	04	0C	14	1C	24	2C	34	3C	44	4C	54	5C	64	6C	74	7C
9	84	8C	94	9C	A4	AC	B4	BC	C4	CC	D4	DC	E4	EC	F4	FC
A	05	0D	15	1D	25	2D	35	3D	45	4D	55	5D	65	6D	75	7D
B	85	8D	95	9D	A5	AD	B5	BD	C5	CD	D5	DD	E5	ED	F5	FD
C	06	0E	16	1E	26	2E	36	3E	46	4E	56	5E	66	6E	76	7E
D	86	8E	96	9E	A6	AE	B6	BE	C6	CE	D6	DE	E6	EE	F6	FE
E	07	0F	17	1F	27	2F	37	3F	47	4F	57	5F	67	6F	77	7F
F	87	8F	97	9F	A7	AF	B7	BF	C7	CF	D7	DF	E7	EF	F7	FF

INPUT BYTE bbbb 3210

Table 5-10: Decimal Mutiply

INPUT BYTE	FIELD A: 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	4 MSB (HEX)
0	00	00	00	00	00	00	00	00	00	00							
1	00	01	02	03	04	05	06	07	08	09							
2	00	02	04	06	08	10	12	14	16	18							
3	00	03	06	09	12	15	18	21	24	27							
4	00	04	08	12	16	20	24	28	32	36							
5	00	05	10	15	20	25	30	35	40	45							
6	00	06	12	18	24	30	36	42	48	54							
7	00	07	14	21	28	35	42	49	56	63							
8	00	08	16	24	32	40	48	56	64	72							
9	00	09	18	27	36	45	54	63	72	81							
A	00	00	00	00	00	00	00	00	00	00							
B	00	00	00	00	00	00	00	00	00	00							
C	00	00	00	00	00	00	00	00	00	00							
D	00	00	00	00	00	00	00	00	00	00							
E	00	00	00	00	00	00	00	00	00	00							
F	00	00	00	00	00	00	00	00	00	00							

FIELD B: $b_3 b_2 b_1 b_0$

6. Keyboard Subsystem

The keyboard subsystem comprises a Dual Extended Drive ROM option Keyboard Controller which can service one or two keyboards with no loss of efficiency or functionality of either.

The keyboard controller is located within the housing of the TPU, and each keyboard is connected to the controller by a cable consisting of four twisted pairs. A keyboard may be up to 2000 feet from the controller. the controller, when used as a single keyboard controller, is program compatible with any standard 10/20 keyboard controller (Model 001-02-02) and with the Dual Extended Drive Keyboard Controller (Model 010-46-02) when the latter is used as a single keyboard controller.

6.1. Addresses and Interrupts

Each keyboard has its own unique address which normally coincides with the screen address of the nearest screen; however, the relationship between a keyboard and a display subsystem is totally under program control.

The controller uses two addresses and one interrupt level. One address and the interrupt level are used for keyboard control. The other address is used to read the terminal address option block and to reset the watchdog timer.

The controller is always installed in I/O controller slot C2 of the SPD 10/25 chassis and when so installed, the following relationships are fixed:

* I/O port J1 (Master Terminal Keyboard) carries keyboard address 0.
* I/O port J2 (Auxiliary Terminal Keyboard) carries keyboard address 0.

6.1.1. Boot Switches

The Boot switch located near J15 initiates a ROM boot when wired to slot C2.

Switch SW1 on the keyboard controller initiates a keyboard boot.

Figure 6-1: Standard Keyboard

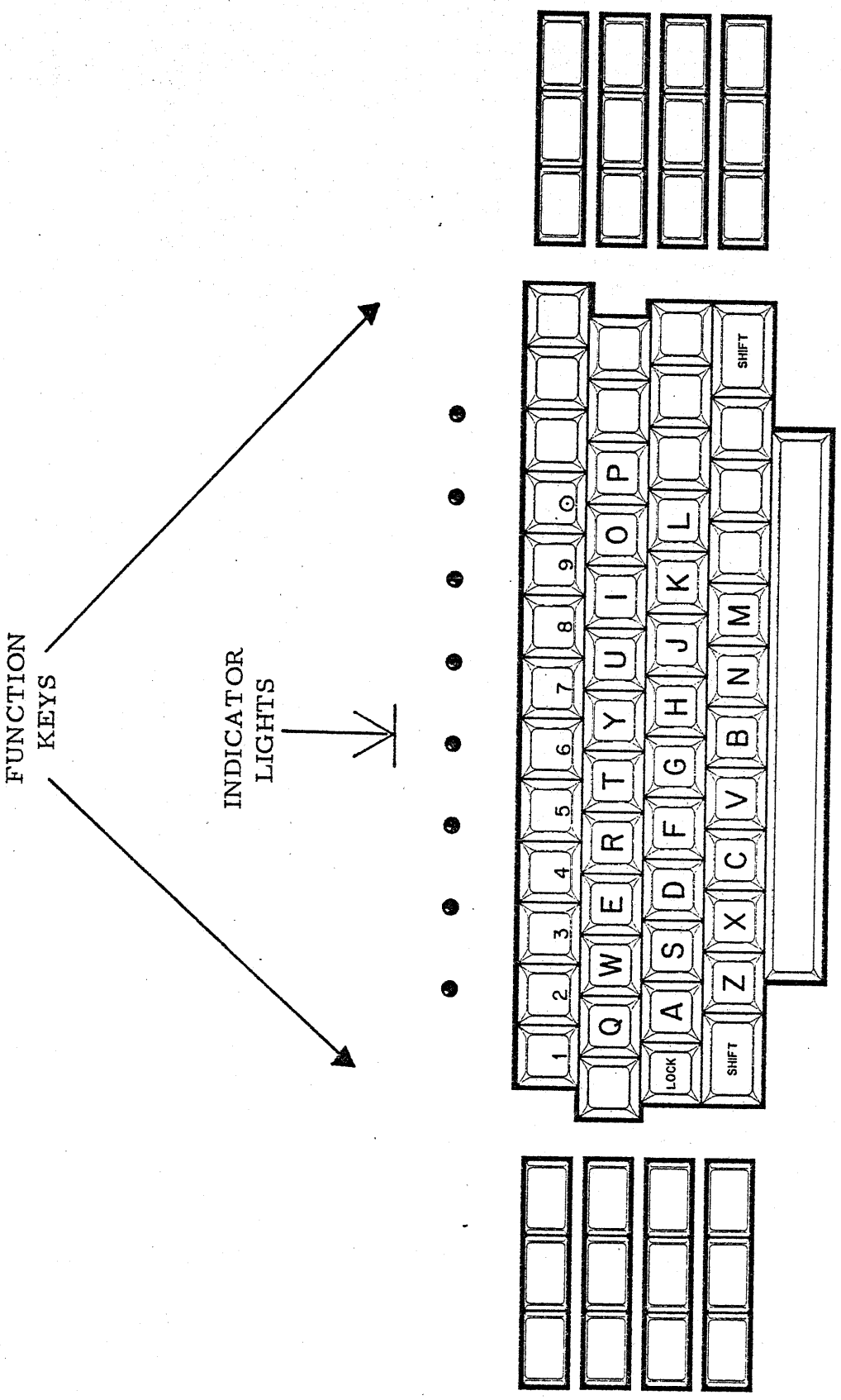
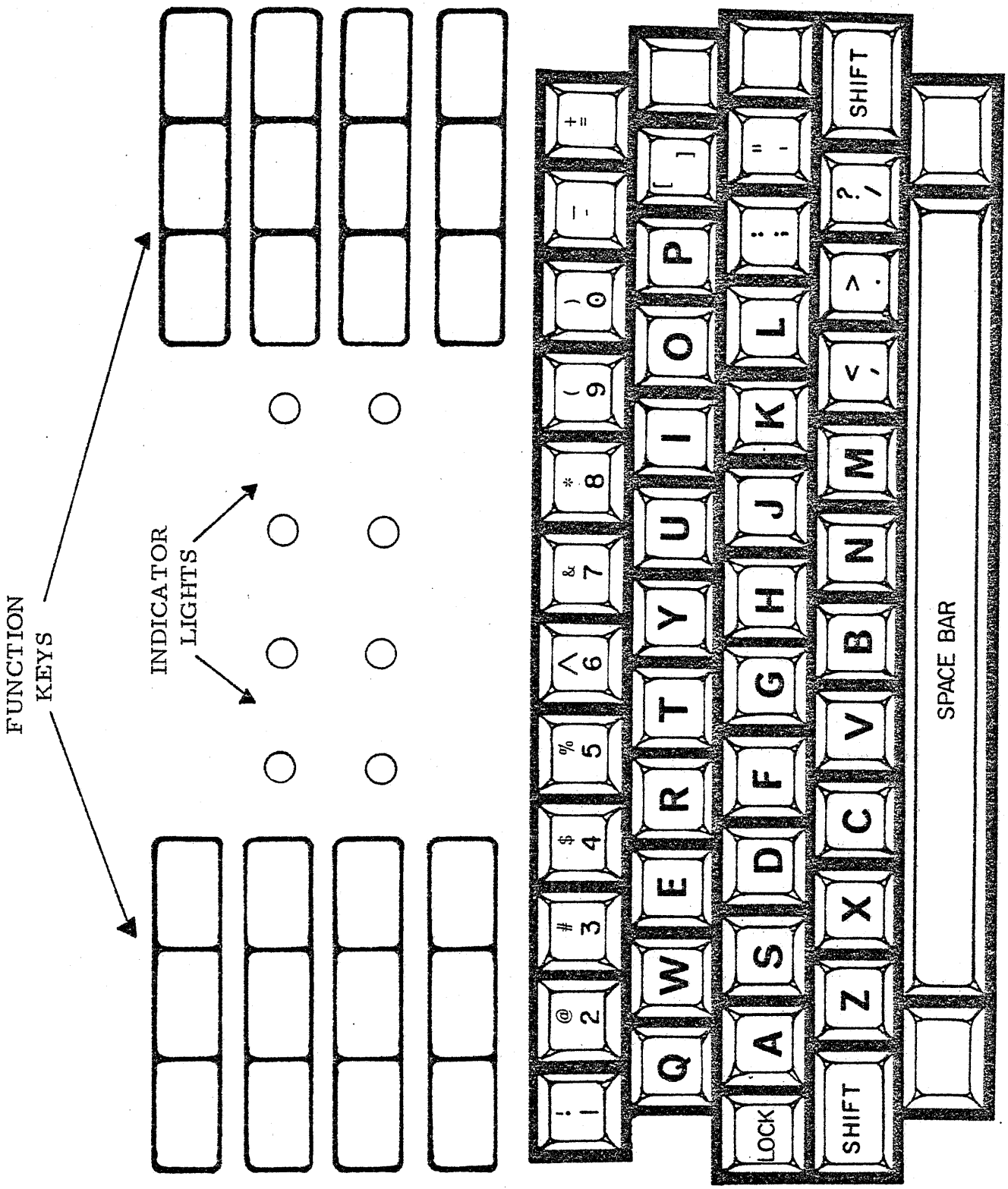


Figure 6-2: Executive Keyboard



6.2. Keyboards

Two models of keyboards are available for the SPD 10/25. They are the Standard Keyboard and the Executive Keyboard, shown in figures 6-1 and 6-2.

From the standpoint of software, the Executive Keyboard is identical to the Standard keyboard with the following exceptions:

* If a program uses the Executive Keyboard lights, they must be refreshed at least every 500ms. If they are not, the lights will be extinguished. This prevents the processor from displaying false information if the power is lost or the program crashes. It is a good idea to include a refresh routine in every program that uses the keyboard lights, so that compatibility problems are minimized.

* As noted in figures 6-1 and 6-2, the left and right key banks are positioned differently on the two keyboards.

* The repeat rate of the Executive Keybcard is higher than that of the Stanadard Keyboard. This causes a DOS interface problem.

6.3. Keycode Generation

The keyboard transmits keycode data, generated by the a key(s) depression, to the keyboard controller. Upon receipt of the information, and depending upon the software design, the controller may interrupt the processor.

For each keyboard option, the codes generated by the subsystem are unique. The following routine can be used to check the generated code of each key:

```
LOOP    JFACK 4,2,$       Wait for keyboard input
        RIO   0,2         Read key
        CIO   2,2         Display code in keyboard lights
        JMP   LOOP        Continue
```

6.4. Keyboard Instruction Set

Table 6-1 summarizes the keyboard controller commands. Each is described in further detail in separate sections.

Function Code	CIO	RIO	WIO	TIO
0,2	----	Read Character	---	Device Present?
1,2	Enable Repeats	---	---	---
2,2	Set Lights	Read Status	---	---
4,2	Mask Interrupts	---	Sound Alarm	Input Ready?
5,2	Controller Reset	---	---	---
6,2	---	---	---	Keyboard 0 Ready?
8,2	Unmask Interrupts	---	---	Controller Busy?
9,2	Select Keyboard	----	----	---

6.4.1. CIO 1: Enable Repeats

This command enables the interrupting keyboard to generate multiple interrupts if the key is continually depressed. After the initial interrupt, a second interrupt is generated approximately 1/4 to 1/2 second later. All subsequent interrupts are generated at approximately 1/10 second intervals.

This instruction must be used after all interrupts if the RIO 2, Read Status, command is to be used effectively as the auto-exec instruction.

6.4.2. CIO 2: Set Lights

This command sends the contents of the ACR to the eight Light Emitting Diodes (LEDs) on the keyboard selected by the CIO 9, Select Keyboard, command, or keyboard 0 if the CIO 2 instruction is issued immediately after an IOR or power up.

The data is sent in the following format:

Bit:								
	7	6	5	4	3	2	1	0	ACR Bits
	7	6	5	4	3	2	1	0	LED position

Thus, the rightmost Keyboard LED corresponds to the ACR Lsb.

This instruction will be ignored by the hardware if the controller is busy (TIO 8 ACKnowledged).

Transfer of data to the LEDs takes approximately 0.30ms.

6.4.3. CIO 4: Mask Interrupts

This command prevents any keyboard from interrupting the processor. Interrupts are also masked by the issuance of an IOR or a CIO 5, Reset, command or by a power-up.

6.4.4. CIO 5: Controller Reset

This command clears all controller logic, masks interrupts and selects keyboard 0 for light or alarm transmission. Polling of the keyboards commences with keyboard 0.

6.4.5. CIO 8: Unmask Interrupts

This command allows the keyboards to interrupt the processor.

6.4.6. CIO 9: Select Keyboard Output

This command selects the keyboard which is to receive light or alarm data. The ACR is used as follows:

(ACR)	Action
0	Select Keyboard 0
1	Select Keyboard 1

Note: 1. Power-up, IOR or CIO 5 commands select keyboard 0.

2. The CIO 9 command will be ignored if the controller is busy (TIO 8 ACKs).

6.4.7. RIO 0: Read Character

This command reads the keyboard data into the ACR. If a parity error occurred in the transfer of data from the keyboard to the controller, then all zeroes will be transferred to the ACR.

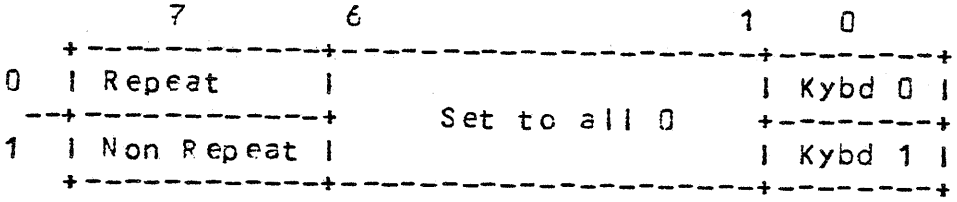
Note:

1. This command will not be acknowledged if data is not available for transfer to the ACR (a key has not been depressed).

2. Polling of the keyboards will resume upon execution of this instruction.

6.4.8. RIO 2: Read Status

This command reads the keyboard address and repeat status into the ACR as follows:



Note:

1. If the RIO 2 command is used, it must be issued prior to the RIO 0, Read Character, command.

2. This command is acknowledged only when the controller has received data from a keyboard.

3. The repeat status bit is only useful if CIO 1, Enable Repeat, was issued after the prior interrupt for that keyboard.

6.4.9. WIO 4: Sound Alarm

This command generates an audible alarm at the keyboard.

Note:

1. The keyboard is selected via the CIO 7 command, or keyboard 0 is selected after an IOR, CIO 5 or power-up.

2. This instruction will be ignored by the hardware if the controller is busy (TIO 8 ACKs).

3. This command is always acknowledged.

6.4.10. TIO 0: Device Present

This command will be acknowleged if the controller is present.

This command does NOT indicate the presence of a keyboard.

6.4.11. TIO 2: Controller Ready

This command will be acknowledged if the controller has received data from the keyboard.

6.4.12. TIO 6: Keyboard 0 Ready

This command will be acknowledged if data has just been read from keyboard 0. This instruction is valid only after the RIO 0, Read Character, command has been issued.

This command can be used to check 10/20 compatibility.

6.4.13. TIO 8: Lights/Alarm Busy

This command will be acknowledged if the controller is in the process of sending light or alarm data to a keyboard.

6.5. Terminal Address Designation

The controller contains an option block which allows the user to specify a unique, 15 bit terminal address for each terminal. This address is read by issuing an RIO C then RIO 1 command addressed to device NINE (9). The address is specified by option block wiring on the controller.

Note:

These two RIO commands may be used to read the hard wired values of the low and high order bytes specific to a given terminal. This allows the same program in several different SPD 10/25 terminals, whose two parameter values are different for each terminal, to identify its terminal address.

6.5.1. RIO 0,9: Read Terminal Address

This command loads the low order eight bits at the address into the ACR.

Note:

1. This command is used in conjunction with the RIO 1, Read Interchange Address, command.
2. This command is always acknowledged.
3. The low-order 8-bits will be all zeroes if no option block is present.

6.5.2. RIO 1,9: Read Interchange Address

RIO 1 loads the high order seven bits at the address into the ACR.

Note:

1. This command is used in conjunction with the RIO 0, Read Terminal Address, command.
2. The high order bit of the data transferred to the ACR is always 0.

3. This command is always acknowledged.

4. The high-order 7 bits will be all zeroes if no option block is present.

6.6. ROM Bootstrap Loader

The Dual Extended Drive/ROM option keyboard controller contains a PROM bootstrap loader. The loader provides an easy method of self-loading an operational program through standard communications: disk, tape, etc.. PROM data is encoded in standard eight-bit core format, not in boot format. Each PROM contains 256 bytes and the SPD 10/25 can accomodate up to two PROMs (512 bytes) total.

A PROM bootstrap may be initiated either automatically or by the watchdog timer, or by manually depressing the BOOT Button located near J15.

Bootstrap is terminated by either:

* The detection of FF in an even location of the PROM, or

* After 512 bytes have been loaded into memory from the PROM.

6.6.1. Watchdog Timer

The watchdog timer may be used to automatically initiate bootstraps. It is enabled/disabled by a toggle switch (SW2) on the controller.

One instruction is provided:

TIO 0,9: Reset Watchdog Timer

The Wathcdog timer starts a PROM load if it is not reset in three seconds. It may be reset by:

* PROM Load

* Power-On

* Issuance of a TIO 0, Reset Watchdog Timer, command.

7. Real Time Clock Programming

The RTC indicates the passage of actual time and has the highest level interrupt priority. The RTC interrupts 60 times a second (every 16.7ms). For programming purposes, the RTC is utilized in the same manner as an external device, and is addressed as device 15.

Three commands are available: RIO 0, Read, CIO 1, Unmask, and CIO 2, Mask.

The read command is acknowledged only when the RTC has been unmasked and has interrupted. If the Read command is acknowledged, the interrupt and ACK signal are cleared. The data transferred to the ACR is all zeroes.

If the Read command is not acknowledged, the auto-exec interrupt structure returns program control to the background processing. A summary of RTC commands is given in the following table.

RTC Command Summary

Function Code	Command
RIO 0,15	Read
CIO 1,15	Unmask
CIO 2,15	Mask

8. Cyclic Redunancy Check

A cyclic redundancy check (CRC) is a method of error detection to determine if a transmitted message has been received correctly. While it is possible to do the required CRC calculations by programming, their complexity would require a large amount of memory and execution time, thus a hardware circuit is provided.

A cyclic redundancy check is performed as follows:

* The transmitting terminal (with a cyclic check controller) develops a cyclic check character (CCC) by accumulating a logic polynomial equation remainder as each character is transmitted.

* At the same time, the receiving terminal's controller performs the same operation on each received character until the final character of the message is received.

* The transmitting terminal sends the final remainder of the polynomial division immediately following the last character of the message.

* The receiving terminal compares the CCC developed at the transmitting terminal with the CCC developed at the receving terminal.

* If the two cyclic check characters are identical, the message is assumed to be correct.

A CRC operation centers around two registers, a 16 bit accumualtor (AC) register and a 8 bit shift register (SR). The cyclic check character being calculated is stored into the AC and various commands are available to extract and restore intermediate results. This feature allows a single cyclic check character generator to be used with a number of communications I/O controllers, all of which use the same cyclic check polynomial.

8.1. CRC Character Generator

The SPD 10/25 Cyclic Redundancy Check Character Generator (CCC portion of the Screen Buffer/CCC/ROM Controller performs the algorithm required to do cyclic check calculations.

Controller address eight is used to acess this function. The CCC logic neither checks nor computes individual character parity.

When operating under conditions of Cursor Disabled (see section 4.3.2.), the WIO 6 and WIO 7 commands must not be issued during refresh (TIO 2 ACKed)

The CCC generator is capable of calculating either of two software selectable polynomials.

8.1.1. PARS Cyclic Check

This mode of the CCC generator works with six-bit data characters and a polynomial:

$$X^6 + X^5 + 1$$

The CCC generator consists of a 6-bit Receiving register. The CCC is six bits long.

8.1.2. Bisync Cyclic Check

This mode of the CCC generator works with eight-bit data characters and a polynomial:

$$X^{16} + X^{15} + X^2 + 1$$

The CCC generator consists of a 16-bit CCC Accumulator and an 8-bit Receiving register. The CCC is 16 bits long (two 8-bit characters).

8.2. CRC Instruction Set

The CRC command set is summarized in table 7-1 and described in detail in the remaining sections.

Table 7-1 CRC Instruction Set

Function Code	CIO	RIO	WIO	TIO
0,8	Mode Select	----	----	----
2,8	----	----	----	Refresh Busy?
6,8	Clear CCC Gener Accumulator	Extract CCC Gener Accumulator	Cyclic Check Accumulate	----
7,8	----	----	Load CCC Generator Accumulator	----

8.2.1. CIO 0: Mode Select

This command determines whether the CCC generator will execute PARS or Bisync Cyclic check computations.

Note:

When use is made of the CIO 0 instruction to select the particular Cyclic Check algorithm, care must be exercised to ensure that other refresh subsystem attributes defined by the CIO 0 instruction are not inadvertently modified. See section 4.6.1. for further information.

8.2.2. CIO 6: Clear CCC Generator Accumulator

This command unconditionally clears the CCC generator accumulator.

This command is issued prior to loading the CCC generator accumulator.

8.2.3. RIO 6: Extract CCC Generator Accumulator

The contents of the CCC generator accumulator are transferred to the ACR. This command is always acknowledged.

In PARS applications, only one RIO 6 command is necessary to read the CCC generator accumulator. In Bisync applications, however, two RIO 6 commands are necessary.

generator accumulator.

In Bisync applications, two successive WIO 7 commands are ney to issue a WIO 7 with the ACR set to zero. This action shifts the 8 Lsb's to the location occupied by the 8 Msb's in the CCC generator accumulator. A second RIO 6 is then issued to extract the contents of the CCC generator accumulator 8 Lsb's (which are now contained in the 8 Msb location).

8.2.4. WIO 6: Cyclic Check Accumulate

This command causes the contents of the ACR to be transferred to the CCC generator receiving register. The data is processed and included in the updated Cyclic Check in the CCC generator accumulator.

This command is always acknowledged. A one cycle delay is required between successive WIO 6 and/or WIO 7 commands.

8.2.5. WIO 7: Load CCC Generator Accumulator

This commmand causes the contents of the ACR to be transferred to the CCC generator accumulator.
In PARS applications, only one WIO 7 utilities for use by disk

This section famapplications, two successive WIO 7 commands are necessary. The first WIO 7 command loads the CCC generator accumulator with eight Msb's and the second WIO 7 command loads the CCC generator accumulator eight Lsb's.

In both applications, the WIO 7 command must be preceded by a CIO 6, Clear CCC Generator Accumulator, command.

The WIO 7 command is always acknowledged. A one cycle delay is required between successive WIO 7 and/or WIO 6 commands.

9. SPD/DOS Diskette Operating System

SPD/DOS is a diskette resident operating system for use with the SPD Family of computer terminals equipped with a floppy diskette unit. It provides facilities for program development, maintenance, and storage and implements a file system plus attendant utilities for use by diskette applications programs.

This section familiarizes the programmer with the general operation of the DOS utilities. For information concerning the inclusion of relevent DOSLIB routines into the applications program, refer to the manual entitled 'SPD/DOS Programmers Reference Manual'.

9.1. SPD Family Required Hardware

The following hardware is required for operation of SPD/DOS on an SPD Family machine:

* Single or dual diskette attached to channel 7,
* 64 or 80 character per line display on the refresh channel, screen 0, and
* An upper-case or upper/lower-case keyboard attached to the keyboard channel, keyboard 0.

9.1.1. Optional Hardware

The following peripheral devices may be used by DOS:

* Line printer models LP125, LP200, LP250, LP300, or LP400 attached to any channel using the parallel controller.
* Character printer models P-100, P-120C, P-165/165A, or P-165B attached to any channel using the parallel controller.
* Character printer models P-100-2, P-120C-2, P-165/P-165A-2 or P-165B-2 character printer attached to any unit of a SPD 20 Family multiple printer controller on channel 13.
* P-158 printer attached to an asychronous controller on any channel or to any unit of an SPD 20 Family mulitple printer controller.
* Termiprinter attached to an asychronous controller on

any channel.

* Card reader/punch on any channel.
* Paper tape reader on any channel.
* Magnetic tape transport attached to any channel.
* Cassette tape transport attached to any channel.
* Built-in, on SPD 20/20 and 20/30 models only, cassette loader on channel 11.

9.2. Diskette Construction

The floppy disk is housed in a protective platic case which cleans the disk while it is revolving in the drive unit. No attempt should ever be made to touch the exposed surface of the disk, or to clean it. Only a soft felt tip pen should be used to write on the disk's plastic case. The disk should be kept in its protective envelope when not it is not in the drive.

The disk is divided into 64 tracks numbered consecutively, from the outer edge to the center, from 0 to 63. Each track is composed of 32 sectors numbered consecutively from 0 to 31. Diskette I/O is done on a sector by sector basis at the hardware level.

SPD/DOS uses the first three tracks of the disk leaving tracks 3 to 63 free for program use. Track 0 contains the bootstrap loader and nucleus code. Track 1 contains the diskette file directory and track 2 is reserved for the core image which is automatically saved when an abnormal or manual boot occurs.

9.2.1. Diskette I/O

For a thorough treatment of the subject at the hardware level, refer to the SPD d-250 Diskette Reference Manual. For a discussion of diskette I/O DOSLIB routines, refer to the SPD/DOS Programmers Manual.

Diskette I/O involves two series of commands: those that affect the transfer of data between the diskette buffer and the TPU (WIO, RIO series), and those that involve the transfer of data between the buffer and the diskette (CIO series).

The diskette buffer is a 256 byte MOS RAM memory which resides on the diskette controller board. 128 bytes of the buffer is used to store the data to be transmitted to the diskette. The remaining

128 bytes can be used by the software program for its own purposes. In particular, DOS uses these bytes to store control information for access by its utilities.

9.2.2. Diskette Write Protection

Any diskette can be protected from accidental write operations by engaging the Write-Lock button on the drive to be protected. In this mode, all write operations to that diskette will be inhibited. DOS utilites return appropriate error codes if an attempt is made to write on a disk that has its Write-Lock engaged.

9.3. Operation of Nucleus

Track 0 of every SPD/DOS diskette contains a boostrap loadable program called the nucleus. When a boctstrap occurs, a short loader program is booted into core which then loads the nucleus. The nucleus provides a set of basic commands for the maintenance of a file system. The remainder of the diskette is used for a file directory and data file area. The files are of four types:

Source Files (S) A source file is a sequential file containing characters from the standard graphic set and is normally used for Assembly language source programs.

Object Files (O) These are executable programs typically produced as output from the SPD/DOS Assembler.

Relocatable Files (R) These are libraries of relocatable modules produced by the SPD/DOS Relocatable Assembler. A given file can contain one or more modules. Relocatable modules are included in absolute assemblies by use of the IN (IN Rfile,module) Assembler pseudo-op.

Data Files (D) A data file is a contiguous set of tracks. DOS imposes no particular structure on a data file. The strucure of the data file is determined by the program which makes use of the file. Within DOS itself, data files are used as direct access work files.

9.3.1. Nucleus Load Procedure

The nucleus is loaded from the diskette by following the diskette standard boot procedure:

Load any SPD/DOS diskette into unit 0 of a single or dual

diskette and depress the boot button. The nucleus will be loaded automatically.

If a diskette fails to boot, the following procedure is recommended. Remove the diskette and carefully revolve the center hub of the diskette, making certain that no hand contact is made with the diskette surface. Replace diskette in unit and depress boot button. If the diskette still fails to boot, its bootstrap record may have been destroyed. In this case, it is necessary to re-format the boostrap code to rescue the disk (see section 9.8.2.1., FORMAT Utility).

In addition, virgin disks cannot be booted. All disks must be FORMATted. FORMAT writes sector identification and nucleus code onto the diskette.

Indication that the nucleus has been booted in is given by the appearance of the nucleus display on screen 0. The nucleus displays the file directory as, illustrated. From this it is seen that the file directory is displayed on the screen using one line of display for each active file.

9.3.2. Directory Display Format

Every SPD/DOS diskette is assigned a Disk Serial Number (DSN) of up to eight alphanumeric characters in length. Following a boot manual boot procedure, the directory of unit 0 is always displayed. The unit whose directory is currently being displayed is referred to as the Currently Selected Unit. The currently selected unit concept is fundamental to DOS file identification.

The meaning of the various fields of the directory is as follows:

S File status:

Blank for an active file, * for an erased file, and ? for a questionable file. The ? status occurs as the result of certain operations in which errors were detected. A file with a ? status may be used as though it was an active file, although it is highly probable that the information contained in that file was not recorded properly.

T Type:

O for Object, S for Source, R for Relocatable and D for Data file.

Name:

This field contains the file-name which is composed of up to eight alphanumeric characters, the first of which must be alphabetic.

FT First Track:

In decimal, the number of the initial track, ranging from 0 to 63.

IF Interlace Factor:

The interlace factor is the separation between logically sequential sectors. It is possible to read up to 6 sectors on one revolution of the disk. However, the real time that is used for processing one sector read may be longer than the time required to read the next logical sector. Therefore, in the worst case, the disk may have to make another complete revolution in order to process the next logical sector.

Thus, the interlace factor is used to specify the order in which sectors are read and/or written. For example it is set to 1 to read sectors in the sequence 0,1,2,3...31. If the interlace factor is set to 5, it will cause the sectors to be read or written in the sequence 0,5,10...30 during the first revolution of the disk and in the sequence 3,8,13...28 during the second revolution of the disk. Sectors can be read or written in this sequence until all 31 have been processed.

Within SPD/DOS, all O files have an interlace factor of 5. R files have a factor of 9. Source file created by the standard source file utility routines have an interlace factor of 11. For purely random access of files, the interlace factor is set to 0. In general, the interlace factor may be any odd value from 1 to 31.

NT Number of Tracks:

This field shows, in decimal, the number of consecutive tracks assigned to the file. It is a value between 1 and 61.

LABEL:

The label on a file is up to 40 characters of information from the graphic character set. The label is used for identification purposes only, and the choice of the label is entirely up to the user.

If the file directory is greater than that which will fit on a

single screen, the space bar may be pressed to page through the remainder of the directory.

9.3.3. Keyboard Indicator Lights

Following a manual boot of the nucleus, the rightmost indicator lamp is lit. The next to rightmost indicator lamp is lit if a program returns to the nucleus in an abnormal manner. All indicators are off following a normal return to the nucleus. Indication that the core image has been saved (barring a write protected unit 0) in given if either indicator is lit.

Entering any command causes all indicator lamps to be turned off.

```
S P D / D O S  SPD DISKETTE OPERATING SYSTEM VERSION 7.02

                UNIT 0    DSN=SYSTEM

ST ..NAME..  FT IF NT..............LABEL.....................
 0 ASSEMBLE  03 05 06  SPD/DOS ASSEMBLE   V7.19  76-08-17-1500
 0 CNFG      09 05 01  SPD/DOS CNFG       V7.01  76-07-07-2100
 0 COPY      10 05 04  SPD/DOS COPY       V7.06  76-08-01-2100
 0 CREATE    14 05 01  SPD/DOS CREATE     V7.01  76-08-01-2000
 0 DCOPY     15 05 01  SPD/DOS DCOPY      V7.06  76-08-01-2100
 0 EDIT      16 05 02  SPD/DOS EDIT       V7.02  76-06-07-1700
 0 ERASE     18 05 01  SPD/DOS ERASE      V7.01  76-08-01-2115
 0 FORMAT    19 05 02  SPD/DOS FORMAT     V7.02  76-07-07-1430
 0 LIST      21 05 01  SPD/DOS LIST       V7.01  76-08-01-1545
 0 PACK      22 05 01  SPD/DOS PACK       V7.01  76-08-06-1300
 0 RASSEMBL  23 05 06  SPD/DOS RASSEMBL   V7.06
 0 RENAME    29 05 01  SPD/DOS RENAME     V7.01  76-08-01-1600
 0 TMOVE     30 05 01  SPD/DOS TMOVE      V7.01  76-08-01-1530
 0 UPDATE    31 05 02  SPD/DOS UPDATE     V7.01  76-07-31-0800
 0 VERIFY    33 05 01  SPD/DOS VERIFY     V7.01  76-08-01-1630
 0 XDISK     34 05 01  SPD/DOS XDISKY     V7.01  76-07-19-1400
 0 ZAP       35 05 02  SPD/DOS ZAP        V7.03  76-08-05-1000
```

9.3.4. File-Name Format

Two files on the same diskette of the same type (O,R,S,D) may not have the same file name. However, it is permissable to have two files of the same type and file-name on two different units. Thus, a standard DOS format has been adopted to differentiate between files:

U.FILENAME

Whenever a file name is required, it can be specified in this format. U is the number of the diskette unit on which the file resides (0 or 1) and FILENAME is the 8 character name of the desired file.

Note: U. may be omitted if the file resides on the Currently Selected Unit, as shown in the current directory display. If FILENAME is the name of a file to be created, then it will be created on the unit specified by U. or, if omitted, on the currently selected unit.

The file-names AUTO, LIST and WORK have special significance to DOS and should be considered to be reserved.

9.4. Nucleus Commands

Nucleus commands are those which are processed by the nucleus itself. They always start with a period as the first character. This period acts to distinguish nucleus commands from program load requests (see 9.4.2., Program Loading).

9.4.1. Entry of Nucleus Commands

Commands can be entered from the keyboard, card reader, the currently executing program or from a file.

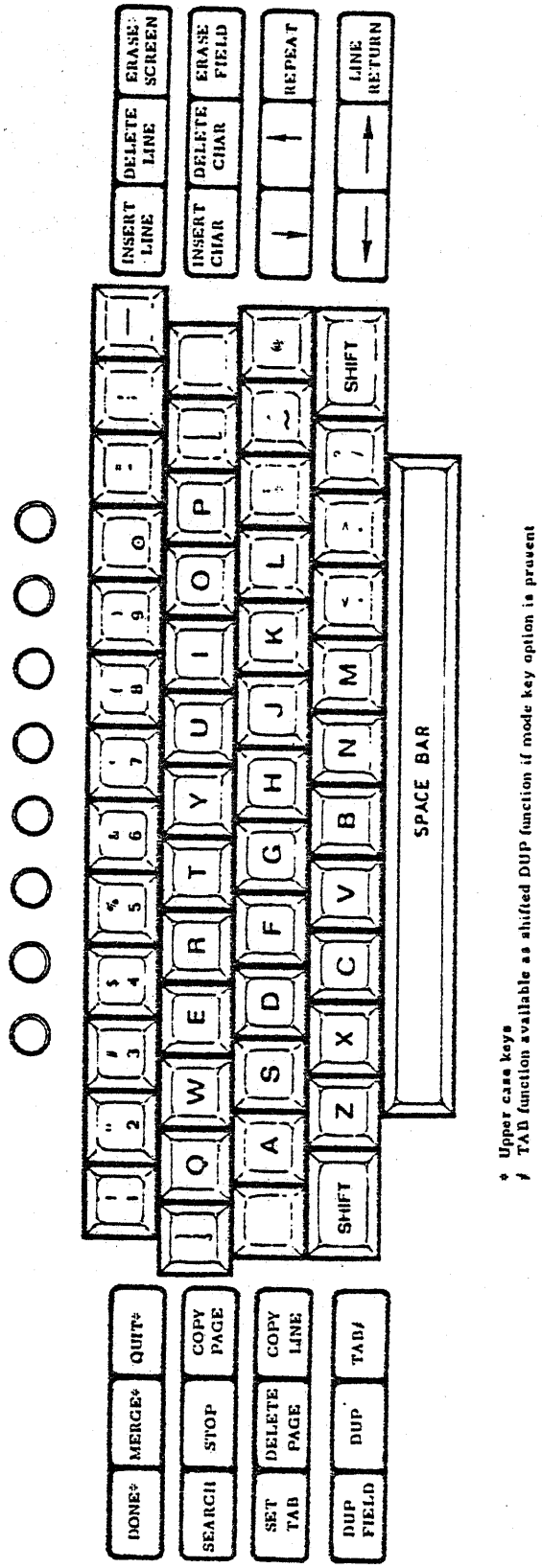
The keyboard command source employs two special function keys, as shown in figure 4-2. The left cursor key (<--) is used to provide a backspace and erase function for error correction. The LINE RETURN key is used at the end of a command to terminate the entry.

9.4.1.1. .C Set Card Input Mode

This command is entered as: .C

It causes subsequent commands to be read from the system card reader, if one has been configured for the system (see CNFG utility, section 9.8.2.2.). Commands are punched left justified on successive cards, one command to a card. The format for the punched commands is exactly the same as that described for the keyboard except that no line return function is needed or available. Commands are read until a .E (End of File, section 9.4.1.2.) is encountered. Control is then passed to the nucleus and keyboard command source.

Figure 9-1 DOS Editor Keyboard Layout



This command can be used to set up a DOS card reader input stream, which can, for example, automatically: read, assemble, and load a program.

9.4.1.2. .E End of File

This command is entered as: .E

It is used to pass control to the nucleus and return to keyboard entry mode. Thus it is meaningful only when entered from a card reader or tape reader.

9.4.1.3. .F Set File Command Input Mode

This command is entered as: .F U.FILENAME

This command causes all subsquent commands to be read from the named source file. Commands in the file are written left justified, one command to a line. Commands are automatically read until a .E or end of file is encountered, at which time control is returned to the keyboard. Once a .F command has been entered, the disk containing the named file may not be dismounted until all of the commands have been read.

The .F command may be contained within the file, but the effect is to simply pass control to the named file. There is no nesting of .F commands.

9.4.1.4. .L and .N Set/Cancel Log Mode

This command is entered as: .L

Use of this command causes all subsequent commands and error messages to be logged onto the system printer, if one has been configured for the system (CNFG, section 9.8.2.2.).

This command is entered as: .N

This command cancels the effect of the .L command. Subsequent commands and error messages will not be logged to the system printer.

In addition, a manual or abnormal boot cancels .L mode.

9.4.1.5. .M Operator Message

This command is entered as: .M Message

While processing commands from a batch source, such as cards or paper tape, it often becomes necessary for the operator to mount a disk or service a particular device. The .M command can be used to suspend processing and display an appropriate message to the operator until a particular action has occured. Processing continues when the space bar is pressed.

9.4.1.6. .T Set Tape Input Mode

This command is entered as follows: .T

The .T command is used to initiate reading of commands from paper tape. A paper tape channel must be configured (CNFG utility, section 9.8.2.2.) for this command to be legal. The format of command input from paper tape is standard 7 channel ASCII coding. The parity bit is ignored and may be odd, even or missing. Each record is terminated by a carriage return (CR) character. Following the final record, there should be an end of tape (EOT) character. All other control characters such as line feed, null or rubout are ignored. The actual end of line sequence could be carriage return, line feed, carriage return or any other convenient sequence.

The character underline or left arrow (ASCII X'5F') has a special function if it occurs immediately prior to the carriage return character. In this case, the entire line is omitted from the input and ignored. This effect occurs only if the underline character occurs immediately prior to the CR character. In any other position, it is treated as a normal data character.

Once the .T command has been entered, commands are read from paper tape until either a .F command is read or the EOT character is detected, at which time command input returns to the keyboard mode.

9.4.1.7. .0 and .1 Select Unit 0/1

This command is entered as: .0
.1

The effect of this command causes unit 0 or 1 to be set as the currently selected unit. The file directory of unit 0 or 1 will appear on the display. No line return is needed to enter these commands.

9.4.2. Program Loading

Object programs are loaded by using the following standard DOS format:

U.FILENAME<,OPTIONS> <PARAMETER1,PARAMETER2,...>

The information enclosed in <brackets> is a standard way of indicating that which is optional. It is used in this context throughout the SPD/DOS section.

U: The unit on which the file resides. It may be omitted if the file resides on the currently selected unit.

FILENAME: The name of an O file which is to be loaded. It may be abbreviated to the first two letters of the filename. If two programs have identical two letter prefixes (i.e. FILE1, FILE2) the abbreviated form of the FILENAME will cause the first (nearest to track 0) program to be loaded. All DOS utility programs have unique initial characters, so that they can always be loaded using the two-letter abbreviation.

OPTIONS: A series of letters (I.E., VOSI) delineated from the FILENAME by a comma and containing no commas or embedded blanks. Any of the 26 letters of the alphabet can be used and the order is unimportant; the program can determine which of the particular letters were set (refer to the SPD/DOS Programmers manual for a thorough discussion). The particular effect of the specified options is determined by the loaded program. If not needed, the leading comma and options field may be omitted. Nearly all DOS utilities require the specification of options upon loading.

PARAMETER: A string of characters delineated from the option field by a leading blank. More than one parameter can be specified and these are offset from each other by a comma. The loaded program can scan out the parameter field for the information it needs to execute. This field may be omitted if parameters are not required.

Following execution of a program load command, the specified program is loaded and executed. DOS utilities return to the

nucleus upon error detection or completion. Application programs may or may not return. Following execution of a program which does not return automatically, control may be returned to the nucleus by manually rebooting the system.

9.5. SPD/DOS Reserved File-names

The source file names AUTO and LIST, the object file name AUTO and the data file name WORK have special significance to the DOS utilities.

If a diskette on unit 0 is booted manually or an abnormal nucleus return occurs and there is a source file named AUTO on unit 0, then commands are automatically read from this file as if .F AUTO had been specified. If there is an object file named AUTO, then it is loaded automatically as though the command AUTO had been given. If both source and object files of AUTO exist on the same unit, the object file will preferentially be loaded.

If the N option for ASSEMBLE had been specified and there is a source file named LIST on either unit, the assembler will automatically write the source listing into that file, preferentially choosing the unit, opposite to the source.

The assembler also uses the data file called WORK for various processing chores during assembly.

9.6. Error Messages

If an error condition is detected during operation of the nucleus or of one of the operating system utilities, the alarm is sounded (if available) and the following message is displayed (and printed if the L mode is set):

```
****E R R O R****   CODE=XXNN
```

XX is the first two characters of the utility program name or NU for a nucleus error. NN is a numeric code from 01 to 99.

The error codes produced by the DOS utilities and the nucleus are listed at the end of this section.

The error condition is cleared by pressing the space bar, which always forces a return to keyboard command mode. Alternatively pressing C, for continue, causes a return to the current command source.

9.7. Core Image Saved on Boot

When the nucleus is booted in manually by pressing the boot button or when a program makes an abnormal return to the nucleus, that part of the current contents of memory which is used by SPD/DOS (excluding locations X'0000'- X'00FF') is saved on track 2 of unit 0 if unit 0 is not write protected. The saved core image is unaffected by the operation of other utility programs (except FORMAT) or the nucleus. A manual boot has no effect on the image if unit 0 is write protected. Execution of other application programs, especially on the SPD 20 Family, may destroy the image.

9.8. SPD/DOS Utility Programs

This section describes the operation of the DOS utilities. Each utility is an object file which is loaded from the diskette on which it resides by the standard nucleus load command, possibly including options and parameters.

Following completion of a task, or upon detection of an error condition, each utility returns to the nucleus to indicate either normal termination or to return an error code.

The load requests indicate which options or parameters are optional by enclosing them within <brackets>.

The DOS utility programs are grouped into three categories: Development, Maintenance and Peripheral Support.

9.8.1. V and M Option Note

Most utilities support the V option (Verify). This option forces all diskette write operations to be verified by using the re-read check. Output to the WORK and LIST files of ASSEMBLE is never verified.

Free use of the V option slows down processing considerably and is very rarely necessary. This prevents the SPD/DOS utilities from operating at their maximum efficiency.

One notable exception is the FORMAT utility. Here, the V phase is very fast and may be used to verify that all sectors have been correctly formatted. Utilities which affect only the directory may use the V option freely (e.g. ERASE, RENAME).

DCOPY and PACK are fairly efficient when operating with the V option set (50% loss of efficiency). It may be wise to use this

option in some situations (e.g., in-place PACK). The M option, available on COPY, DCOPY, FORMAT and PACK, allows the utilities to be loaded from a separate diskette. Thus, this option allows these utilities to operate on a single drive system. In addition, this option allows COPY, DCOPY and PACK to be loaded from, for example, a system disk and then operate on separate disks in a dual drive system.

9.8.2. Development

This section contains the DOS utilities which relate to program and diskette development.

9.8.2.1. FORMAT: Format Diskette

Load:

FORMAT<,Options> Unit<,Disk-Name>

Options:

D — Format File directory only. This causes the the formatting of the file area, tracks 3 to 63, to be skipped. FORMAT can thus be used as an efficient way to clear a directory.

M — Mount. Pause to mount diskette. Allows FORMAT to be loaded from a separate diskette in a one disk system. FORMAT continues when the space bar is pressed.

N — Format and write Nucleus only. The formatting operation is restricted to tracks 0 and 2 only. The files and file directory remain unchanged. This option may be used to reformat a defective nucleus or write a new version of DOS to the nucleus track.

V — Verify. After the entire diskette has been formatted, a verify phase will be initiated which will check to see if all sectors have been formatted correctly. Used to verify that the disk surface is undamaged.

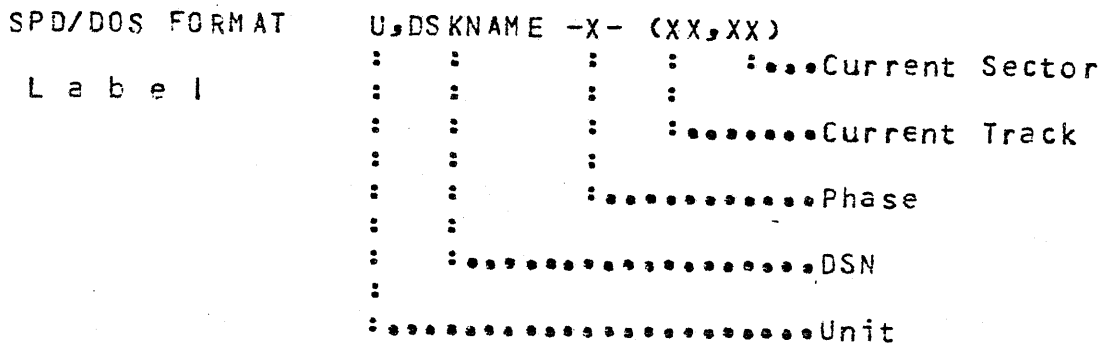
The FORMAT utility is used to format a virgin disk so that subsequent disk operations by the nucleus and possibly the application program can be performed. Format also writes the nucleus code onto track 0 of the disk. All disks must be formatted correctly to run under SPD/DOS.

Under some conditions, a diskette may not function correctly as

evidenced by its failure to boot properly or by constant return of disk I/O error codes, such as Read Check (see section 9.9., Diskette Errors). The FORMAT utilty may be able to rescue such a disk. FORMAT is loaded with the unit number of the diskette to be rescued, and the V option set. FORMAT will enter an E phase, described in the following paragraph, if an error is detected.

The configuration parameters written to the newly formatted disk are copied from those currently active. These parameters are described in the CNFG utility section.

FORMAT DISPLAY



FORMAT operates in five phases as indicated on the display:

M Mount. Pause to mount diskette to be intitialized. This allows FORMAT to be loaded from a separate diskette. FORMAT continues when the space bar is pressed.

F Format. All sectors are formatted with initial contents of zero. The display continuously reports the current sector and track being formatted.

V Verify. The success of phase F is checked. This is done only if the V option is specified.

W Write. The nucleus code is written onto the disk.

E Error. Displayed if a Write Error is detected. FORMAT pauses so that the number of the track and sector where the error occured can be noted. Pressing the space bar causes return to the nucleus with the appropriate error code.

If the diskette to be formatted has already been given a Disk-name, as noted in its directory display, this parameter need not be given. In such a case, the old Disk-name is retained.

Example:

```
FO,V 1,DSK01
FO,N 0
```

The diskette residing in unit 1 is to be formatted and verified. It will have the Disk-name DSK01.

The diskette on unit 0 will have its nucleus reformatted and rewritten. The original Disk-name will be retained.

9.8.2.2. CNFG: Configure Diskette/Buffer

Load: CNFG (No Options or Parameters Permitted)

CNFG is used to establish new system peripheral codes in the diskette controller buffer and possibly write them to track 0 of the desired diskette. DOS uses these codes to determine what type of equipment is available to its utilities. Therefores, the diskette buffer must be properly configured before any of the SPD/DOS utilities are used. Whenever a diskette is booted, the information in track 0 is written to the diskette buffer. Thus it is necessary to make certain that the proper system configuration parameters are contained in the diskette before it is manually booted.

CNFG operates on a question and answer basis. The questions asked depend on the machine model in which CNFG is loaded. The following sections describe the questions and possible answers. They are not presented in any specific order.

9.8.2.2.1. Action Required?

0 - Write Parameters to unit 0
1 - Write Parameters to unit 1
S - Set New Parameters
X - Exit to Nucleus

Answers 0 and 1 write the diskette buffer parameters to units 0 and 1 respectively. Thus, these answers may be given to simply propagate current parameters from disk to disk, or to write new parameters, established after entering S, onto disks. Once parameters have been written onto a disk, they will be written to the diskette buffer whenever the disk is booted in.

Answer S forces CNFG to further interrogate the user. The questions asked depend on the machine model and the various responses. As each question in this phase is answered, the newly set parameter is written to the diskette buffer. Upon compeetion of this phase, 'Action Required?' is again displayed and a new set of parameters may be in effect, though they are not written to the disks unless 0 or 1 is keyed in. Entering the S key anytime during the interrogation forces a return to the first question, 'Printer Type?'.

Answer X causes CNFG to immediately return to the nucleus. Any parameters set by responses to the utiltity, even though they were not written to the diskette, are in effect until they are modified by a manual or abnormal disk boot or CNFG.

9.8.2.2.2. Printer Type?

For answers to this question, and to the related questions, 'Controller Type?' and 'Printer Channel?', consult the specifications manual of the printer or system in use. If no printer is used, answer N.

After entering the printer and controller type, the user will be asked to supply pace depth and line width information. This will affect all printed output produced by the SPD/DOS utilities and should be compatible with the current form in use.

9.8.2.2.3. End Refresh? (SPD 10/20 only)

Answer 0 for upper/lower case keyboards and 7 for upper case. This parameter is used by the SPD/DOS utilities to control the display.

9.8.2.2.4. Screen Size? (SPD 10/20 only)

Choose F for full screen, 30 X 64 display, or H for half screen 15 X 64 display.

9.8.2.2.5. Keyboard Type?

Choose 1 for upper case only, 2 for upper/lower case, 3 for upper/lower case converted (allows upper case only editing on an upper/lower keyboard), 4 for IDES upper/lower case and 5 for IDES upper/lower case converted.

9.8.2.2.6. Additional Questions

Other questions ask information concerning the existance of various peripherals, such as a paper tape reader. The questions simply request the assigned device channel of the peripheral. Type N for each device not used.

9.8.2.3. ASSEMBLE/RASSEMBL: Assemble Source File

Load:

```
ASSEMBLE<,Options> Sfile<,b<,Ofile <,Label>>>
AS<,Options> Sfile
```

RASSEMBL: Relocatable Assembly

Load:

```
RASSEMBL<,Options> Sfile<,b<,Rfile <,Label>>>
RA<,Options> Sfile
```

Note the nesting of parameters. This specifies the parameters that may be omitted.

Options:

A	Alternate Unit. The object or relocatable file is generated on the unit opposite from that implied by the call.
B	Both pass. Print on pass 1 and pass 2.
C	Clean. Enforce LIF 0: IF/ENDF and deleted code are never printed.
*D	Definitions. INclude DOS standard definitions module. Use of this option can be avoided by INcluding D&DEFS at the appropriate location in the code.
E	Erase duplicate object file.
F	Full list. Enforce LIF 2 mode: All IF/ENDF, plus deleted code is printed.
*G	Generate IN code: Enforce LIN 1, INcluded code is listed.
*H	Hold IN code: Enforce LIN 0. INcluded code is listed only for errors.
I	Inhibit object file production.
K	Kill Hash comments. Enforce List 2 mode. Ejects and subtitles are used to format the listing.
L	List. Enforce List 3. Hash comments are listed.
N	No Printer: Print source listing in LIST file.
*O	Omit. Omits literal cross references.
P	Paper save. Enforce List 1: eject and subtitles are treated as comments. Hash comments are killed.
Q	Quick Assembly: XREF 0. No cross reference is generated.
R	Reference unassembled: Enforce XREF 2. Cross reference for deleted IF/ENDF code is generated.
S	Short List: Print only errors.
T	Table of Contents. Shows the initial page number of each subtitle line. It is printed immediately before the listing of the first subtitle line.
U	Unlist deleted code: Enforce LIF 1. Code deleted, plus IF/ENDF lines in false IF ranges, is not printed.

V Verify writes to object. WORK and LIST are never verified.

X Cross Reference: Enforce XREF 1. Collect cross references in assembled areas.

* ASSEMBLE utility only

ASSEMBLE and RASSEMBL are used to produce absolute and relocatable object code from souce files. These utilities require that a Data file called WORK be established prior to their loading. The WORK file may be created by the CREATE utility.

Example:

CR,D 45 establishes a 45 track data file called WORK on the currently selected unit.

Use of the N option requires the establishment of a Source file called LIST prior to the assembly. The size of the list file should be one and a half times as large as the source file itself.

Example:

CR,S 20 establishes a 20 track scurce file called LIST on the currently selected unit.

The utilities will preferentialy select the WORK or LIST file on the unit opposite to the source if the files reside on both units. For optimum efficiency, the WORK file should immediately follow ASSEMBLE or RASSEMBL, though the utilities will work no matter how the files are positioned.

ASSEMBLE searches the currently selected unit first when INcluding a given module. Thus, maximum efficiency is obtained when the currently selected unit is set to that containing the relocatable file(s) prior to loading the utility.

ASSEMBLE no lonqer defines DOS standard symbols. They can be incorporated into an assembly in one of two ways. The program must be assembled with DOSLIB and the D option used which automatically INcludes D&DEFS as line 0 of the assembly. Or, the program itself INcludes the D&DEFS module. Both actions must not be taken together or M flags will result. The latter method is preferable since it is more efficient and cleaner looking, documentation-wise.

The b parameter must be given as four hex digits. If the b parameter is omitted, $B defaults to 0. If the object file and label are omitted, the filename and label are copied from the source file. Note the nesting of parameters shown in the load request (e.g. b cannot be omitted if ofile and label are given).

If the source program is contained on more than one disk, sfile is given in the format:

Filename.Diskname1.Diskname2<.Diskname3>

Filename specifies the name of all the files, which must be the same, and the unit on which the initial and subsequent files reside.

Diskname specifies in sequence the names of the disks on which the files reside. As shown, up to three disks may be specified. The parts of this parameter are separated by periods.

During assembly, a pause will occur for mounting the disks. The diskette containing the work file, the object file and the utility must NEVER be reloaded.

Examples:

```
0.AS CHESS
AS,FSO DETOURS,1965,1.WHO,GENERATION
RA,A ALOT.DISK1.DISK2,FFFF
```

The source file CHESS on the currently selected unit is assembled. The object file is produced on the same unit and carries the same file name and label.

The source file DETOURS is assembled to produce the object file WHO which will carry the label GENERATION. $B is set to X'1965'. Any previous object file called WHO will be erased. No printed listing, except errors, will be generated.

The source program is made up of two files called ALOT on disks DISK1 and DISK2. $B is set to X'FFFF'. The A option ensures that the relocatable file is generated on the unit opposite to the remount unit, as required.

9.8.2.4. EDIT: Edit Source File

Load:

Edit Existing File:

```
EDIT<,Options> Input-file,Output-file,Label
```

Edit a New File:

```
ED<,Options> ,Output-file,Label
```

Examine Existing File:

```
ED<,Options> Input-file
```

Options:

E	Erase Duplicate Source File
V	Verify Writes. Note: this option cuts efficiency considerably!

Special Function Keys

DONE*	Copy remaining input file to output file. Close output file and exit.
MERGE*	Merge new input file into output file.
QUIT*	Delete remaining input file from output file. Close output file and exit.
COPY LINE	Copy 1 line of input file to output file.
COPY PAGE	Copy page image to output file. Read next page to screen.
DELETE LINE	Delete 1 line of input from output file.
DELETE PAGE*	Delete page image from output file. Read next page to screen.
DELETE CHAR	Delete 1 character at current cursor position.
DUP	Duplicate 1 character from previous line.
DUP FIELD	Duplicate, until the next tab stop, the previous line.
ERASE FIELD	Erase from current cursor position until the next tab stop.
ERASE SCRN*	Erase screen image from output file.
INSERT CHAR	Inserts a blank into current cursor position.
INSERT LINE	Inserts a blank line into current line position.
LINE RETURN	Move cursor to start of next line.
REPEAT	Repeat function of any key pressed.
SEARCH	Search to line number or label. A specific line will be requested by EDIT. EDIT can then be used to automatically COPY/DELETE PAGE or COPY/DELETE LINE until the desired line is reached. LINE RETURN cancels search mode (though not the actual search). The back-arrow can be used to backspace through the entered label or line number. STOP can be used to immediately terminate the search.
SET TAB	Define start of new field.
SET TAB*	Remove all tabs.
STOP	Stop the current operation in progress. This key can be used to cancel a SEARCH or DONE command.
TAB	Move cursor to start of next field.

The following keys repeat when held down:

--->	Move cursor to right
<---	Move cursor to left
↑ ↓	Move cursor up or down.

*	Upper-case function keys

The EDIT utility has three main purposes. It may be used to either edit an existing source file and write the edited text into a new output file or to create a new source file and enter text directly from the keyboard. The EDIT utility can also be used to simply inspect an existing file, in which case all write operations are inhibited. The mode which one enters EDIT is dependent upon which of the three parameter formats is chosen when EDIT is loaded.

If EDIT is operated on an upper/lower case keyboard, and the system has been configured for such, then EDIT will accept upper/lower case text.

EDIT provides for automatic search-and-edit up to a specified line. The SEARCH key is pressed and a specfic line number or label, of up to 8 numeric or alphanumeric characters, is requested. Then, one of the COPY PAGE/LINE or DELETE PAGE/LINE functions must be selected. EDIT will automatically perform the selected function until the desired line is reached. STOP can be used to terminate the search at any time.

When all editing is completed, the user selects one of the two end-of-job commands: DONE or QUIT. DONE copies the remaining input file into the output file. QUIT immediately closes the file, deleting any remaining input file text from the output file.

Edit also provides for merging another file into the output file or merging the input file into 'itself'. When the Merge key is pressed, all input is suspended and a new unit and file name is requested. When these are entered and the Line Return key is pressed, the named file becomes the input file. It is possible to name the previous input file as the MERGE file or to name the output file as the MERGE file. The new input file is selectively edited, by using the standard EDIT functions, SEARCH and DELETE/COPY PAGE/LINE, until the MERGE file has been processed.

When merging an output file into itself, the user should be aware that it is not possible to copy closer than one disk sector.

To leave the MERGE mode, either the QUIT or DONE function is selected and input resumes from the previous file.

Note: EDIT makes every attempt to close the output file. In fact, an end of file and a last track of 63 is written to the output file immediately after it is created. This helps to ensure that, even in the event of a power down, most of the file will be recoverable.

Unrecoverable input errors do not terminate the edit. Instead, a message **BAD BLOCK** is read onto the screen corresponding to one or more unreadable records. The operator may use the keyboard to repair the error.

Blank records are deleted by EDIT. Therefore it is not possible to directly insert blank records into the output file.

Example:

Suppose that a syntax error was detected by the assembler in line 550 of a source program PPONG. The following commands may be entered in sequence to repair the error:

```
ED 0.PPONG,1.PPONG,PING-PONG V3.08  76-04-27-2400
SEARCH
550
COPY PAGE
Make correction
DONE
```

Note that the output file is created on the unit opposite from the input file. This allows us to retain both the original and edited files without having to change the filename.

Now suppose that this program has been assembled, loaded and verified bug-free. Suppose it becomes necessary to attach PPONG to the end of a main program called PPONG1. MERGE can be used to solve this dilema:

```
ED 0.PPONG1,1.PPONG1,MAIN PROGRAM PPONG
SEARCH
EPRGRM
COPY PAGE
MERGE
1
PPONG
DONE
DONE
```

Note that SEARCH is used to COPY PAGE until the label EPRGRM is found. This is where the subprogram will be merged. When MERGE is pressed, the unit on which the new file resides is entered, followed by the filename. The entire file is copied into PPONG1 by using DONE. DONE (QUIT could have been used as well) is used again to exit from the main edit.

9.8.2.5. CREATE: Create File

Load:

```
CREATE,Options No-Tracks,Interlace,Filename,Label
CR,Options No-tracks
```

Options:

D	A Data file is to be created.
E	Erase existing duplicate file.
S	A Source file is to be created. Normally, this option is used only to create the LIST file required by the ASSEMBLE and RASSEMBL utilities (N option set).
V	Verify all write operations.

Note: If neither D or S is specified, D is assumed.

The Create utility is used to establish a new data or source file. The alternate form of the load request allows default values to be specified for the filename and interlace factor. Note that the number of tracks (No-tracks) is always required.

Option	Interlace Factor	File Name	Label
D	0	WORK	none
S	11	LIST	none

Example: CR,D 55,5,SPOOL,MAIN SPOOLING FILE
CR 15

A 55 track Data file called SPOOL will be created on the currently selected unit and carry the given label. The interlace factor will be set to 5.

A 15 track data file called WORK will be established on the currently selected unit, carrying an all blank label. The interlace factor will be set to 0.

9.8.2.5.1. Rescuing a Disk

CREATE can be used to rescue a disk by attempting to create a file over the disk area in question. If CREATE finds bad sectors, it will attempt to reformat them. If it succeeds, a CR10 error code, file area exhausted will be returned. The return of any other code means that the disk is defective and a backup copy should be used.

Example:
CR 80 (Make certain we create past last track)

Error Return: ***E R R O R*** CODE=CR10

A 'file area exhausted' error is returned, meaning CREATE has either found no bad sectors, or it has reformatted the

questionable sectors.

9.8.3. Maintenance

This section includes the utilities needed to maintain the developed files and diskette.

9.8.3.1. DCOPY: Diskette to Diskette Copy

Load:

```
DCOPY<,Options> From-filename,To-filename<,Label>
DC<,Options> Filename,To-Unit<,Label>
```

Options:

D	Data file to be copied.
E	Erase file with duplicate filename.
M	Mount diskette: Load DCOPY from separate diskette.
O	Object file to be copied.
R	Relocatable file to be copied.
S	Source file to be copied.
V	Verify all write operations.

DCOPY is used to copy a file from one diskette to another. DCOPY can be loaded from a diskette other than that which contains the file to be copied, by using the M option. DCOPY pauses until the Input/Output diskettes are mounted and resumes execution when the space bar is pressed. The M option can be used to copy files in a single drive system.

```
Example:  DC,MO REX,O
          DC,R 0.UTIL,1.ULIB,UTILITY LIBRARY 76-07-23-0945
```

The file REX on unit 0 is to be copied onto another diskette in a one-drive system. DCOPY copies the file in half-sector chunks, pausing for the diskettes to be mounted, until all of REX is copied.

The relocatable file UTIL is copied into the file ULIB which DCOPY will create on unit 1. It will carry the named label.

9.8.3.2. ERASE: Mark File as Erased

Load:
  ERASE<,Options> File1,File2,File3....

Recover Erased File:
  RE,E<Options> File

OPTIONS

D	Data file to be erased
I	Ignore Error: Not used in RE utility
O	Object file to be erased
R	Relocatable file to be erased
S	Source file to be erased
V	Verify erase.

The ERASE utility is used to mark a file as erased (*). The space occupied by this file is not reclaimed until the diskette is PACKed. The RENAME utility can be used to recover a file marked as erased.

Normally it is considered an error to erase a non-existant file. This situation is often encountered when ERASE is used to mark a string of files as erased. Therefore the I option may be set to ignore this error condition.

EXAMPLE: ER,VOIS TELE,TELE1,TELE2
RE,ES TELE

The object files TELE, TELE1 and the source file TELE2, TELE will be marked and verified as erased. The fact that there is no object TELE2 or source TELE1 will be Ignored by ERASE

The source file TELE will be marked active in the file directory.

9.8.3.3. PACK: Pack Diskette

Load:
PACK<,Options> Unit

To Pack From 1 Diskette to Another:
PA<,Options> Unit,Unit

Options:

D	Retain all Data Files
M	Pause to Mount Diskette
O	Retain all Object Files
R	Retain all Relocatable Files

S	Retain all Source Files
V	Verify Writes

Note: If none of the D, O, S or R options are specified, all active files are retained.

PACK is used to remove all erased files from the directory and pack the disk into the reclaimed space. It may be possible to rescue an accidently packed diskette by using the XDISK utility to rewrite the directory.

PACK can be used to pack from 1 disk to another selectively retaining all active files or only active source, object, data or relocatable files. Thus the utility provides an efficient method for producing diskette copies.

If an error occurs while copying a file, the file status on the output unit is set to '?', error. PACK always completes its operations before signalling an error and returning to the nucleus.

No attempt should be made to stop a PACK operation prematurely. This will cause the file directory to be written incorrectly, making any attempt to load or manipulate the files futile.

Example: P A,V
P A,V0 O,1

All erased files are deleted from the currently selected unit and the disk is packed into place. All write operations are verified.

All active object files are packed from unit 0 to unit 1. All writes are to be verified.

9.8.3.4. LIST: List File

Load:

```
LIST<,Options> <file>
LI<,Options> <Unit>
```

Options:

A	Align. Enter printer alignment sequence. This must be the only option, if it is specified.
D	File to be listed is a data file.
H	Hexadecimal. Listing for data or relocatable files given in hexadecimal.
I	Ignore errors. Bad data records will not cause error termination.
N	Number. Lines of S files will be numbered.
O	Object. File to be listed is an object file.
R	Relocatable. File to be listed is a relocatable file.
S	Source. File to be listed is a source file.

The LIST utility is used to list files. It also has an option, A, to generate printer alignment data. LIST can also be used in the alternate form of the load request to list the file directory of a particular unit. If the unit is not specified, the currently selected unit directory will be printed.

A printer must have been configured (CNFG utility) for LIST to function.

The use of the A option generates alignment data for the printer and is used only in the keyboard command mode. For a character printer, the operator is invited to type an F or V for form eject or vertical tab alignment respectively. For a line printer, a digit from 1 to 4 is entered to align the correspoding forms channel. The S key prints test data with no form feed. The alignment data may be generated repeatedly until the X key is pressed to return to the nucleus. The printer must be in its READY state (selected) for any of the alignment commands to be operative.

More than one file of the same name but different type may be listed by specifying more than one file type option (D,O,R or S).

Data files are listed by sector according to the SIF for the file. Data and files are listed in ASCII unless the H option is specified. Object files are listed in hexadecimal by sectors Relocatable files list the module directory and also the hexadecimal sector data if the H option is given. Source files are listed by logical records and numbered if the N option is specified.

```
EXAMPLE:  LI,SON DOC
          LI 1
```

The source and object files of DOC will be listed. Source records will appear numbered.

The file directory of unit 1 is to be listed.

9.8.3.5. RENAME: Rename File

Load:

```
RENAME,Options Old-name<,New-name><,Label>
```

Options:

D	Rename data file.
E	Rename erased file.
O	Rename object file.
R	Rename relocatable file.
S	Rename source file.
V	Verify all write operations.

Note: Exactly one of the D,O,R or S options must be specified.

The RENAME utility is used to rename an existing file. The file may be active or, if the E option is set, erased. If several erased files with the same names exist, only the first will be marked active.

In no case is it possible to use RENAME to generate two active files of the same name and type.

```
Example:  RE,S OLD,NEW,THIS FILE HAS BE RENAMED
          RE,O SAM,,THIS FILE HAS BEEN RELABELED
```

The source file OLD is to be renamed and given a new label.

The object file SAM is to be given a new label.

9.8.3.6. XDISK: Examine Diskette

Load:

```
XDISK<,Options> No Parameters Permitted
```

Options:

F	Full buffer. Display entire 256 byte diskette buffer

area. Normally, only the 128 byte sector data area is displayed.

V — Verify all writes to the diskette.

Special Function Keys:

H — Hard copy of screen written to printer.
L — Position cursor to the left.
P — Modify cursor position. Two hexadecimal digits are entered after the P function is selected.
R — Position the cursor to the right.
S — Enter sector number, as two decimal digits.
T — Enter track number, as two decimal digits.
U — Enter unit number, in decimal. XDISK assigns the unit number to be the currently selected unit.
W — Write hexadecimal data, as modified on the screen, to the diskette buffer.
X — Exit to nucleus.

XDISK is used to examine and alter the data contained on the diskette. XDISK works on a sector by sector basis, reading data from the selected track and sector to the diskette buffer. XDISK displays the sector data area of the buffer, but will display the entire buffer if the F option is set during the load request.

XDISK also displays the current unit, the current track and the current sector being examined. The data appearing on the screen may be modified by positioning the cursor over the desired byte, using one of the L, R or P function keys, and entering the new data, in hexadecimal. When all modifications have been made to that sector, the W key is used to write the new information back onto the diskette.

XDISK displays the status of all read or write operations performed on the disk in the ERROR-N field, where N is:

0 — No error.
1 — Search error. Sector not found.
2 — Read or Verify error.
3 — Device inoperable.
4 — Track number is out of range.
5 — Unit is write protected.

Thus, one use of XDISK is to inspect a disk that has become defective. It also may be used to rescue a directory which has been destroyed or altered by an abnormal PACK or FORMAT operation. In this case, the track positions of every file must be known so that the directory can be reconstructed properly.

Example:

It is necessary to recover a file which has been accidentaly erased and packed. The file was at the end of the directory and was not written over. All the vital information, first and last track location, interlace factor, filename and label are all known. Therefore it is possible to use XDISK to rewrite this entry in the file directory, which is located on track 1 of unit 1.

```
XD              Load XDISK
U               Select unit 1
1
T               Select track 1
01
S               Search for end of directory
00              Not here
S
05              Not here
S
10              End of directory at sector 10
Rewrite entry
W               Write to disk
X               Exit
```

9.8.3.7. ZAP: Patch/Examine Object Program

Load:

To Patch/Fxamine and Store into New File:
ZAP<,Options> ifile,ofile,label

For In Place Modification:
ZA<,Options> ifile

To Examine Core Image after Breakpoint or Boot:
ZA<,Options>

To Save Core Image in a New file
ZA<,Options> ,ofile,label

Options:

A	Alternate Work File. Unit 1 is used as the work file.
E	Erase duplicate object file.
T	Three. Three, instead of four digits are required to specify an address. Useful in small programs (CNFG 10 or 25).
V	Verify all write operations.

ZAP is a multi-purpose utility that allows the programmer to:

* Inspect, modify and restart the core image saved on a manual or abnormal boot,
* Patch an object program,
* Copy a program from one external medium to diskette or 20/20, 20/30 cassette.
* Write a program directly in machine code.

9.8.3.7.1. Efficient Usage of ZAP

ZAP uses the track 2 of unit 0 as a scratch area, unless the A option is set which causes the utility to use unit 1. ZAP loads and stores segments between the work area and the input and output files. Thus, it is more efficient to have the work file on the unit opposite to the output file and input file.

A good rule of thumb to remember when using DOS utilities in general, is that a drive to drive copy is always more efficient than a one-disk copy.

9.8.3.7.2. Patch/Examine Program

To examine and patch a program, ZAP can be loaded using one of the two forms:

```
ZA<,Options> Ifile,Ofile,Label
ZA<,Options> Ifile
```

The first format is used to store the modified program into another file, preserving the original program.

If in place modification is desired, that is, if the patch is to be stored into the input file, the second format is used. Alternatively, if the modification is not to be stored at all (if the program is simply to be started) this form may also be used to load ZAP.

To perform the required modifications to the input file the following procedure is used:

L Load the program (or segment nn, as required) into the core image.

Aaaaa Set the desired address aaaa.

Mdddd Modify to the required contents. A jump to a patch area is a possibility.

Saaaa.aaaa,eeee,<nn>
Store modified program or segment into the output file, or if none was specified, into the input file. Note that the abbreviated command, Snn, must be used if no output file was specified (in place modification).

Jaaaa Start core image at aaaa, or if not specified, start at the entry point as read from the input file. Note that many times it is desirable to start the core image directly after modification, eliminating the S step. This allows one to check the effectiveness of the patch before making patches permenant.

Also, when patching many programs, it may become necessary to compute an absolute address when all that is known is the relocatable address. This is solved by setting the relocation base to the required offset.

For instance, suppose that the source listing of an absolute assembly showed that the address where a module of relocatable code was inserted (using the IN pseudo-op) is X'A00'. This is set as the relocatable base: RA00. Now, any relocatable address, can be specified in terms of the absolute address by using the command ARaaaa. The effect is to add the relocatable address to the relocation base and produce an absolute address:

```
AEAE
RA00
M8CC0,R0120
```

The modification sets a jump command to relocatable address 120 + A00 = B20.

Any addresses appearing in any command may be given as a relocatable address Raaaa and the effect is to add in the current location base (e.g. in the alternate form of the Aaaaa command: bRaaaa, where b is a blank.

9.8.3.7.3. Breakpoints and Saved Core Image

When debugging a program, it is often helpful to know the contents of the ACR and Index (Cursor) register at various points in the program. ZAP can be used to place Breakpoint instructions at the point where the ACR and Index are to be displayed. This is done by using the Baaaa command:

```
ZA FILE
B02EE
J100
Program is executed until breakpoint
ZA
ACR and Index displayed at breakpoint
```

The core image is then started using the Jaaaa command and the program is executed until the breakpoint is reached. Control is passed to the nucleus by an abnormal boot. This saves the core image, which can be tested by ZAP. ZAP is loaded without specifying parameters. An automatic T instruction is executed and the contents of the ACR and Index Register are displayed at the breakpoint.

Use of ZAP's breakpoint feature requires that the DOS subroutine D&POWR be included into the main program:

```
IN   DOSLIB,D&POWR        INCLUDE DOSLIB ROUTINE
IN   DOSLIB,D&SBPW        USED BY D&POWR
IN   DOSLIB,D&WRBF        ...
IN  DOSLIB,DGEXIT
ORG  $X-2                 SET D&POWR RESTART LINKAGE
DAC  D&POWR               ...
```

9.8.3.7.4. Copying Programs from Peripherals

It is possible to use ZAP to transfer a program from other external media, such as paper tape, to diskette or cassette tape.

The procedure for achieving this requires the use of ZAP's third format:

ZAP<,Options> ,Ofile,Label.

Note that a label is required in this case. The program to be copied is loaded from the desired peripheral, a paper tape loader, say, and a manual boot is initiated. This saves the core image for ZAP to process. ZAP is then loaded and the following command is given:

Saaaa,aaaa,eeee,OO

This will save the specified region(s) of core in Ofile. Note that it is necessary to determine in what regions of core the program resides and to determine the entry point. It is also possible, on the SPD 20/20 and 20/30 models, to store the program on cassette in bootstrap loadable form by using the W command.

It is sometimes useful to clear core before an external load. This is achieved by loading ZAP, clearing the core image, and then physically loading, but not starting, the image:

```
ZA
Z
I
```

Note that some programs won't work if they have a self-destructing initialization routine. One possible solution, for the paper tape format, is to cover the blank end record of paper tape with self adhesive tape. In the punched card format, remove the last card.

Programs can be entered in directly, via ZAP, in machine code. The desired entry point is set, typically X'100', and a series of modify commands are given to enter the program. It is useful, and less error-prone, to have these commands read from a file. For example, the following commands are stored in a file called T:

```
ZA,T ,HELP,TEST PROGRAM     Call ZAP
A100                        Set origin to X'100'
MCB42,B900                  Code
S100.103,100,00             Store
J100                        Start up
```

To enter and start this program, the command .F T is given.

9.8.3.7.5. ZAP Considerations

When the output file is closed (J or E commands) and if there is an input file specified, then all segments of the input file which have not been stored by S commands are copied unchanged to the output file. In the case of the J command, subsequent segment loads will be from the new output file.

Note that when an S command is given with no address ranges, for example, in the case of an in place update (Snn), only those areas originally in the program are stored. The following areas will NOT be stored:

* BSS areas,
* Areas skipped by ORG commands,
* Bytes skipped by word alignment,
* Areas outside the bounds of the program.

Attempts to modify such areas and then store the result do not work and give no error condition. Always use the full form of the

Store command when patching a program.

ZAP uses a work area (track 2) on unit 0 to store the current core image (4K); thus unit 0 must not be write protected. Neither the utility programs (except FORMAT and ASSEMBLE/RASSEMBL, see note below) nor the nucleus disturb this image, except on a manual or abnormal boot. Thus if ZAP errors out, it may be restarted without losing the core image being worked on. However, the current output file may be lost.

Version 7 of the assembler sets the unused portion of the literal area to X'FF'. This can make a nice patching area when debugging a program.

To save the image which is contained in core, it is necessary to follow this procedure:

```
ZA ,SAVF,LABEL
Cssss
S0100.7FFD,0100,00
E
```

9.8.3.7.6. ZAP Commands

Note: All ZAP commands, except X and a form of A, are entered using the standard SPD/DOS Line Return Key.

In addition, ZAP commands can be read from a file. They are stored, left justified, one command to a line, in the forms given below. Performing a ZAP run in this manner has the advantage of keeping a permanent record of the changes performed. It's faster, too.

Aaaaa	Set Hex address aaaa as the current address. aaaa may be three or four digits.
baaaa	An alternate form of the above command. A blank (b) followed by four hex digits (three if T option set) which sets the current address. No line return is needed to enters this command.
B	A breakpoint instruction, X'7FFC', is placed at the current address. See the discussion of breakpoints for use of this command.
Baaaa	Place a breakpoint at the specified address aaaa.
Cssss	Specify a logical core size, ssss, that is smaller than the actual physical size. This is the core size that will be written to the output file (e.g. 3FFF: 16K, 7FFF: 32K, FFFF: 64K)
Db<text>	Dump all core onto printer with message <text>. A blank (b) separates D from <text>.
Daaaa.aaaa,....b<text>	Dump from specified region(s) of core aaaa.aaaa <text> may be supplied as a header and must be separated from the last address by a blank (b).
E	End ZAP run. Close output file and exit.
I	Load core image. Zap terminates processing with the core image loaded, but not started. The machine waits in a DISABLED spin. The output file, if any, is closed before the wait occurs.
J	Control is passed to the program entry point as read from the input file. Valid only if an input file was specified.
Jaaaa	Control is passed to location aaaa. The output file, if any, is closed before the jump.

Jaaaa<,Options> Parameters

Start execution of core image at aaaa. Options and parameters are made available to the program just as if a nucleus load command had been given.

J<,Options> Parameters

As above, but control is passed to the entry point as read from the input file.

K

Causes the contents of the current address to be set as the current address. The contents of that address is displayed. This can be used for tracing linked lists.

Lnn

Load segment nn (two hex digits). If nn is omitted, the main or only segment is loaded into the core image

Mdddd,dddd..

The current address and the words following are modified to contain dddd (four hex digits).

N

Next address. The next word is set as the current address.

P

Previous address. The previous word is set as the current address.

Raaaa

Set Relocation Base. See the discussion of Relocatable addresses for an explanation of this command.

Saaaa.aaaa,...,eeee,nn

Store segment nn at aaaa to aaaaa. Enter at eeee. In this case, a segment may only be stored once. As shown, more than one region can be stored. This command is not valid unless an output file is specified.

S<nn>

Store segment nn for in place update. This is the only form allowed when no output file is specified, in which case it permissable to store a segment more than once. As shown, nn may be omitted, in which case the currently loaded segment is stored.

T

Display registers at breakpoint (if any).

Vdddd

Verify the current location to contain dddd. Terminate ZAP run with error if not.

Waaaa.aaaa....,eeee,c

The command is used on the SPD 20 Family series

to write the specified regions (up to 7) cnto the 20/20 or 20/30 cassette load tape in debug dump form suitable for boot loading. C is a digit giving the number of copies to be written (from 1 to 7, normally 5). As shown, more than one region may be stored.

X — Immediate exit. The output file is not closed. That part of the core image saved on track 2 remains intact. Segments stored in core may be lost.

Z — Clear core image to X'00'.

9.8.4. Peripherals

Four DOS utilities are particularly suited to peripheral management. These are: COPY, TMOVE, UPDATE and VERIFY.

9.8.4.1. COPY: Copy File

Load:

```
COPY,Options From-file,To-file<,Label>
```

Copy Source File from Current Command Source:

```
CO,S<Options> ,To-file,Label
```

Copy Diskette File

```
CO,Options File,unit
```

Options:

D	A data file is to be copied.
E	Erase file with duplicate file name. This is meaningful only if the To-file is a diskette file.
L	Logical copy. See the discussion for an explanation. This is valid only in the disk to disk case.
M	Mount. This option is valid only in the disk to disk case. Allows COPY to be loaded from a separate disk.
O	An object file is to be copied.
R	A relocatable file is to be copied.
S	A source file is to be copied.
V	Verify. Meaningful only if the To-file is a diskette file. May also be used to verify multiple copies of punched paper tape or punched card output.

The COPY utility is used to copy a file to and from a wide range of external peripheral devices. It can also be used to copy a file from one diskette to another. However, DCOPY should be used in this case.

It should be noted that the ZAP utility has the capability of writing core images as object files on diskette and also building core images from object files. This may be used as an additional mechanism for moving object files to and from external media in formats other than those supported by the copy program.

To copy files to the SPD 20/20 or 20/30 built-in cassette, do NOT use COPY. Instead ZAP (section 1.8.3.7.) and its W command must be used.

To designate external peripheral devices other than the diskette, the From-file and To-file parameters are given in the following form:

.TTC<.NNN> Where C is the hexadecimal device channel and TT is the two letter device type shown below. NNN is three decimal digits specifying the number of copies to be punched. It is used only when the To file parameter specifies a paper tape or card punch, or when the copies are being verified in which case the devices are a paper tape or card reader.

PP	Paper tape Punch
PR	Paper tape Reader
CT	Cassette Tape
MT	Magnetic Tape- 1/2 inch, 9 track
CR	Card Reader
CP	Card Punch/Reader

The following sections describe the operating procedures to be used with the various peripheral devices.

9.8.4.1.1. Paper Tape Input: PR

The paper tape must be positioned at the start of the file to be read or on the leader. Source files and command files are in the same format except that source files may use the back-slash \ as a tab character. Tab stops are set to 1-10-16-30 for use in assembly programs. Object format is identical to the output of the H716 assembler. If a bootstap loader is present, it is skipped. A C020, error reading external medium) is posted if a checksum error is encountered.

9.8.4.1.2. Paper Tape Output: PP

The output generated is compatible with the paper tape format described in the preceding section. It includes a 100 character null leader at the beginning and end of the tape. Output to the paper tape punch may be verified via a special use of the V option.

Following a copy to a paper tape punch, the verify may be used as follows. Manually remount the tape on a paper tape reader on channel 12, (channel 1 on SPD 10 series) and verify the copy by:

```
CO,OV 1.X,.PRC.nnn
```

Where X is the original From-file and .nnn the number of copies

to be verified.

9.8.4.1.3. Cassette Tape Input: CT

Files are read in a format compatible with that generated by the cassette tape output, described in the following section. Error reading external medium is signalled if the tape is in manual mode, or if it stalls during read, or if a dropout or parity error is detected. This applies only to the model T13-01 cassette unit and not to the SPD 20 family built in cassette tape drive.

9.8.4.1.4. Cassette Tape Output: CT

One file is written onto one cassette. The cassette is rewound before starting if necessary. The format for source files is the same as that used by the SPD cassette tape assembler and editor. Object files, which are not compatible with the cassette tape assembler, are written in 512 byte blocks. For unsegmented programs, the output includes a boostrap loader which uses device address 12 on the SPD 20 Family machines, and device 1 or 3 (whichever is ready) on a 10/20. A C021, error writing external medium, is signalled if the tape is in manual mode, or if it stalls during the write or if it is write protected.

9.8.4.1.5. Magnetic Tape Input: MT

The file is read without positioning the tape, thus the tape must be positioned to the beginning of the desired file using TMOVE (section 9.8.4.2.). After reading the last file, the tape is left positioned ready to read the next file. Source file format is compatible with that used by the H716 assembler system. Error reading external medium is signalled if the tape is off line or if the formatter is switched off, or a read error is encountered, or if the file format is incorrect.

9.8.4.1.6. Magnetic Tape Output: MT

The file is written without positioning the tape, thus the tape must be positioned to the beginning of the desired file using TMOVE. After writing the file, two end of file marks are written and the tape is left positioned following the first end of file, ready to write the next file. Error writing external medium is

signalled is the tape is offline or the formatter is switched off or a write error is encountered.

9.8.4.1.7. Punched Card Input: CR or CP

On the card reader, the deck is loaded into the hopper and the RESET button is pressed. On the reader/punch, the deck is loaded into the primary (rear) hopper and the reader reset. Following reading the last card (.E end of file), this card must be run out manually on the reader punch. Error reading external medium is signalled if a checksum error is detected in an object deck. Other card reader errors cause COPY to pause for appropriate operator intervention.

9.8.4.1.8. Punched Card Output: CP

The printing-reader-punch must be cleared and reset before starting. Blank cards are fed from the secondary (front) hopper. In the case of object files for unsegmented programs, the output includes a bootstrap loader. This loader is compatible with the reader-punch or the card reader. It uses the same device address as the punch unless a fourth digit is given on the device specification.

For example, .CP36 specifies a file to be written to the punch on channel 3 with boostrap loader for device address 6. Error writing external medium is signalled if an attempt is made to punch on non blank cards. In other error situations, COPY waits for appropriate operator intervention.

COPY can also produce multiple copies from a paper tape punch with verify.

9.8.4.1.9. Error Detection

In general, COPY attempts to complete the copy operation even if errors are detected, the appropriate error code being posted upon completion. If the output is a diskette file and such an error occurs, the file is closed and marked with ?, error status.

9.8.4.1.10. Examples

```
CO,RE AFILE,1
CO,O  0.MT3,,CT4
CO,D  0.ICU,1.IMURES,VERSION TWO
CO,S  .CR6,1.FILENAME,VIKING II
```

The relocatable file AFILE on the currently selected unit, presumed to be unit 0, is copied to unit 1 with the name and label unchanged. Any previous relocatable file AFILE on unit 1 is erased.

An object file is copied from a magnetic tape on channel 3 to a cassette tape on channel 4 with the original label.

The data file ICU on unit 0 is copied to unit 1 where it is named IMURES with label VERSION TWO.

A punched card source file is copied from a card reader on channel 6 into FILENAME on unit 1. It carries VIKING II as a label.

9.8.4.2. TMOVE: Tape Move

Load:

```
TM<,Options> Channel<,N>
```

Options:

B	Backspace files.
F	Forward space files.
L	List directory.

Note: At most, only one of these options may be present.

TMOVE is used to position a magnetic tape unit to a particular file. It is used in conjunction with the COPY utility to manage multiple files on a single tape. TMOVE also contains a directory listing option.

The form of the command used to position the tape to the start of a specified file, numbered from 1, does not take any options:

```
TM Channel,File-number
```

To backspace, the B option is used. The number of back-space operations performed is given by File-number. Backspacing one file involves moving backwards one file mark and continuing to move backwards until the next file mark or beginning of tape is

reached. If a file mark is reached, the tape is spaced forward past this mark.

Forward spacing operations are similar. The F option is used to forward space files from the current position.

To list the initial record of every file on the tape, the L option is used. The initial record is typically a label record. Thus this command acts to give directory listings. The tape is rewound following completion of the listing. It is assumed that the last file is terminated by a double end of file mark.

Examples: TM 3,4
TM,B 3,6
TM,L 0

The tape on channel 3 is positoned to file 4.

The tape on channel 3 is backspaced 6 files.

The tape on channel 0 is listed.

9.8.4.3. UPDATE: Batch Edit

Load:

UPDATE<,Options> Ifile,Ofile,Label

Options:

E	Erase any duplicate source file.
L	Log update input lines to the printer.
V	Verify all diskette output.

UPDATE is used to edit a source file in batch mode. The update data is taken from the current input device (cards, tape, file or keyboard) depending on the current mode.

UPDATE Commands:

$Keyword Comments

This is the UPDATE command format. $ indicates an UPDATE command is to follow, which is designated by the Keyword. Comments can be specified on any UPDATE command line and do not affect processing.

$=any single character

The UPDATE command character will be replaced by the indicated character from now on. This character may be changed by another occurance of the command, Character=New character.

```
$* Text
```

Text may be inserted into the UPDATE command stream by this command.

```
$&any-character
$&&
```

Subsequent occurances or the character & will be replaced by the character indicated. $&& resets the normal mode.

```
$n,m
$n
$,m
```

These commands control Ifile to Ofile copy: n indicates to what line (card image) text will be deleted (non-inclusive). m indicates to what line text will be copied. Thus, the first form indicate lines n through m of the input file will be copied to the output file, the second form indictes UPDATE will delete any lines up to n, but not including n, from the output file. The third form will cause UPDATE to copy the input file from the current position through line m.

Examples

```
$50
$50,60   is ILLEGAL, but

$11,20
$21      is LEGAL
```

```
$DONE
$QUIT
```

DONE causes the remainder of the input file to be copied to the output file until and end of file is detected. Unless in MERGE mode, control returns to the system. QUIT causes the remainder of the input file to be deleted from the output file. Unless in MERGE, control is then passed to the system.

```
$MERGE filename
```

This command switches the input to the named file. Input continues from this file until $DONE or $QUIT keywords are detected. Command then resumes from the previous input file.

```
$PAUSE Text
```

This command causes UPDATE to halt and display Text to the operator. Pressing the spacebar continues UPDATE.

Note: Commands where $ is followed by a KeyWord may be abbreviated to their first letter (E.G., $P for $PAUSE).

Source lines read from cards or tape are in standard ASCII form. The backslash \ may be used as a tab character as described for Paper Tape Input in section 9.8.4.1.1.

A source line must not begin with the UPDATE command character: $. A source line, if it appears in the UPDATE command stream, is copied onto the output file.

```
Example:  UPDATE,L IN,OUT,BLACK
          THIS IS A SOURCE LINE
          $,23
          $26,28
          $30
          $DONE
```

The source file IN on the currently selected unit is updated and the output written to source file OUT with LABEL BLACK on the same unit. The new file consists of THIS IS A SOURCE LINE followed by lines 1 through 23, 26 through 28 and 30 though the end of the file. All update input is logged to the printer.

9.8.4.4. VERIFY: Verify File/Diskette Label

Load:

```
VERIFY Unit,DSN
VE,Option File,Label
```

Options:

D	File is a data file.
O	File is an object file.
R	File is a relocatable file.
S	File is a source file.

Note: Options are only permitted for the second load request.

VERIFY is used in batch (cards, tape, file command) mode to verify that a diskette has the expected dskname (that the correct diskette is loaded).

VERIFY can also be used to verify that a file is present and has the expected label by loading the utility in the second load format.

VERIFY signals an error condition if the expected DSN or file label does not match the actual one.

Example: VE 1,IMADSK

```
VE,0 1.FILE,LABEL
```

Verify that the diskette on unit one has dskname IMADSK.

Verify that the object file FILE on unit one has label LABEL.

9.9. Diskette Errors

Certain general types of errors in diskette input/output occur as follows:

Search Check	A sector can not be found. Either the diskette is improperly formatted, the heads are seeked to the wrong track, the prior sector had a cyclic check error, no diskette is mounted, the drive's door is opened or the hardware is malfunctioning.
Read Check	A cyclic check comparison repeatedly failed after several attempts at reading the data. This condition indicates that the data was recorded incorrectly.
Verify Check*	A cyclic check comparison failed during a write operation. The error may be in the previous sector (indicating incorrectly recorded data) or in the sector being written if the verify option is used (indicating a damaged disk surface).
Unit Inoperable	The diskette unit is inoperable. This condition is most usually caused by reloading the unit at a time when a reload was not permitted (E.G., reloading the source file diskette unit in the middle of an assembly).
Write Protect	A write was attempted to a diskette which was write protected (write tab attached or write protect button depressed).
Segment Load Error	In an overlay program, an attempt to load an overlay segment failed due to a diskette input error. If this happens, the object program file should be abandoned and a backup copy used.

* Formerly called a Write Check.

9.10. SPD/DOS Error Codes

9.10.1. ASSEMBLE Error Codes

NOTE: These error codes have not been updated to reflect changes in SPD/DOS. Hence some are wrong.

AS 01	Error opening source file. Search check.
AS 02	Error opening source file. Read check.
AS 03	Error opening source file. Unit inoperable
AS 04	Error opening source file. File not found.
AS 05	Error creating object file. Search check.
AS 06	Error creating object file. Read check.
AS 07	Error creating object file. Unit inoperable.
AS 08	Error creating object file. Duplicate file name.
AS 09	Error creating object file. Write protect.
AS 10	Error creating object file. File area full.
AS 11	Error opening work file. Search check.
AS 12	Error opening work file. Read check.
AS 13	Error opening work file. Unit inoperable.
AS 14	Error opening work file. File not found
AS 15	Work file too small.
AS 17	Error reading source file. Search check.
AS 18	Error reading source file. Read check.
AS 19	Error reading source file. Unit inoperable.
AS 20	Error reading source file. Missing end of file.
AS 21	Error writing object file. Search check.
AS 22	Error writing object file. Verify check.
AS 23	Error writing object file. Unit inoperable.
AS 24	Error writing object file. File area exhausted.
AS 25	Error writing object file. Write protect.
AS 26	Error reading work file. Search check.
AS 27	Error reading work file. Read check.
AS 28	Error reading work file. Unit inoperable.
AS 30	Error writing work file. Search check.
AS 31	Error writing work file. Verify check.
AS 32	Error writing work file. Unit inoprable.
AS 33	Error writing work file. Work file full.
AS 34	Error writing work file. Write protect.
AS 35	Segment load error.
AS 36	Parameter format error.
AS 37	Error opening list file. Search check.
AS 38	Error opening list file. Read check.
AS 39	Error opening list file. Unit inoperable.
AS 40	Error opening list file. File not found.
AS 41	Error opening list file. Write protect.
AS 42	Error writing list file. Search check.
AS 43	Error writing list file. Verify check.
AS 44	Error writing list file. Unit inopeable.
AS 45	Error writing list file. File area exhausted.
AS 46	Error writing list file. Write protect.
AS 47	Incorrect file allocation for disk reload.

9.10.2. CNFG Error Codes

CN01	Attempted use in other than keyboard mode.
CN02	Error reading label record. Search check.
CN03	Error reading label record. Read check.
CN04	Error reading label record. Unit inoperable.
CN05	Error writing label record. Search check.
CN06	Error writing label record. Verify Check.
CN07	Error writing label record. Unit inoperable.
CN08	Error writing label record. Write protect.

9.10.3. COPY Error Codes

C001	Error opening input file. Search check.
C002	Error opening input file. Read check.k.
C003	Error opening input file. Unit inoperable.
C004	Error opening input file. File not found.
C005	Error creating output file. Search check.
C006	Error creating output file. Read check.
C007	Error creating output file. Unit inoperable.
C008	Error creating output file. Duplicate file name.
C009	Error creating output file. Write protect.
C010	Error creating output file. File area full.
C011	Error reading input file. Search check.
C012	Error reading input file. Read check.
C013	Error reading input file. Unit inoperable.
C014	Error reading input file. Mising end of file
C015	Error writing output file. Search check.
C016	Error writing output file. Verify check.
C017	Error writing output file. Unit inoperable.
C018	Error writing output file. File area exhausted.
C019	Error writing output file. Write protect.
C020	Error reading external medium.
C021	Error Writing external medium.
C022	Boot mode program too long to write to cassette.
C023	Missing D, O, R, S option or more than one opt.
C024	Parameter format error.
C025	Segment load error.
C026	Verify error.

9.10.4. Create Error Codes

CR01	Error creating file.	Search check.
CR02	Error creating file.	Read check.
CR03	Error creating file.	Unit inoperable.
CR04	Error creating file.	Duplicate file name.
CR05	Error creating file.	Write protect.
CR06	Error creating file.	File area full.
CR07	Error initializing file.	Search check.
CR08	Error initializing file.	Write check.
CR09	Error initializing file.	Unit inoperable.
CR10	Error initializing file.	File area exhausted.
CR11	Error initializing file.	Write protect.
CR12	Parameter format error.	

9.10.5. DCOPY Error Codes

DC01	Error opening input file.	Search check.
DC02	Error opening input file.	Read check.
DC03	Error opening input file.	Unit inoperable.
DC04	Error opening input file.	File not found.
DC05	Error creating input file.	Search check.
DC06	Error creating input file.	Read check.
DC07	Error creating input file.	Unit inoperable.
DC08	Error creating input file.	Duplicate file name.
DC09	Error creating input file.	Write protect.
DC10	Error creating input file.	File area full.
DC11	Error reading input file.	Search check.
DC12	Error reading input file.	Read check.
DC13	Error reading input file.	Unit inoperable.
DC14	Error reading input file.	Missing end of file.
DC15	Error writing output file.	Search check.
DC16	Error writing output file.	Verfiy check.
DC17	Error writing output file.	Unit inoperable.
DC18	Error writing output file.	File area exhausted.
DC19	Error writing output file.	Write protect.
DC23	Missing D, O, R, or S option or more than one option.	
DC24	Parameter format error.	
DC25	Segment load error.	

9.10.6. EDIT Error Codes

ED01	Error opening input file.	Search check.
ED02	Error opening input file.	Read check.
ED03	Error opening input file.	Unit inoperable.
ED04	Error opening input file.	File not found.
ED05	Error creating output file.	Search check.
ED06	Error creating output file.	Read check.
ED07	Error creating output file.	Unit inoperable.
ED08	Error creating output file.	Duplicate file name.
ED09	Error creating output file.	Write protect.
ED10	Error creating output file.	File area full.
ED11	Error writing output file.	Search check.
ED12	Error writing output file.	Verify check.
ED13	Error writing output file.	Unit inoperable.
ED14	Error writing output file.	File area exhausted.
ED15	Error writing output file.	Write protect.
ED16	Segment load error.	
ED17	Parameter format error.	

9.10.7. ERASE Error Codes

ED01	Error reading directory.	Search check.
ED02	Error reading directory.	Read check.
ED03	Error reading directory.	Unit inoperable.
ED04	Error writing directory.	Search check.
ED05	Error writing directory.	Verify check.
ED06	Error writing directory.	Unit inoperable.
ED07	Error writing directory.	Write protect.
ED08	Specified file not found and I option not set.	
ED09	Missing D, O, R, or S option.	
ED10	Parameter format error.	

9.10.8. FORMAT Error Codes

F001	Error formating diskette.	Search check.
F002	Error formating diskette.	Verify check.
F003	Error formating diskette.	Unit inoperable.
F004	Error formating diskette.	Write protect.
F005	Error writing diskette.	Search check.
F006	Error writing diskette.	Verify check.
F007	Error writing diskette.	Unit inoperable.
F008	Error writing diskette.	Write protect.
F009	Parameter format error.	
F010	Segment load error.	
F011	Error reading label record.	Search check.
F012	Error reading label record.	Read check.
F013	Error reading label record.	Unit inoperable,

9.10.9. LIST Error Codes

LI01	Error opening file.	Search check.
LI02	Error opening file.	Read check.
LI03	Error opening file.	Unit inoperable.
LI04	Error opening file.	File not found.
LI05	Error reading file.	Search check.
LI06	Error reading file.	Read check.
LI07	Error reading file.	Unit inoperable.
LI08	Error reading file.	Missing end of file.
LI09	Missing D, O, R or S option.	
LI10	Parameter format error.	

9.10.10. Nucleus Error Codes

NU01	Error reading directory.	Search check.
NU01	Error reading directory.	Read check.
NU01	Error reading directory.	Unit inoperable.
NU01	Error reading label sector.	Search check.
NU01	Error reading label sector.	Read check.
NU06	Error reading label sector.	Unit inoperable.
NU07	Error opening object file.	Search check.
NU08	Error opening object file.	Verify check.
NU09	Error opening object file.	Unit inoperable.
NU10	Error opening object file.	File not found.
NU11	Error loading program.	Search check.
NU12	Error loading program.	Read check.
NU13	Error loading program.	Unit inoperable.
NU14	Error loading program.	Improper object file format.
NU15	Error loading program.	Wrong system or Boot mode.
NU16	Error loading program.	Insufficient memory on 20/20 or 10/24.
NU17	Error opening command file.	Search check.
NU18	Error opening command file.	Read check.
NU19	Error opening command file.	Unit inoperable.
NU20	Error opening command file.	File not found.
NU21	Error reading command file.	Search check.
NU22	Error reading command file.	Read check.
NU23	Error reading command file.	Unit inoperable.
NU24	Error reading command file.	Improper format.
NU25	Card reader not configured.	
NU26	Tape reader not configured.	
NU27	Command format error.	
NU28	Error writing label sector.	Search check.
NU29	Error writing label sector.	Write check.
NU30	Error writing label sector.	Unit inoperable.
NU31	Error writing label sector.	Write protect.

9.10.11. PACK Error Codes

PA01	Error reading diskette.	Search check.
PA02	Error reading diskette.	Read check.
PA03	Error reading diskette.	Unit inoperable.
PA04	Error writing diskette.	Search check.
PA05	Error writing diskette.	Verify check.
PA06	Error writing diskette.	Unit inoperable.
PA07	Error writing diskette.	Write protect.
PA08	Parameter format error.	

9.10.11.1. RASSEMBL Error Codes

RA01	Error opening source file.	Search check.
RA02	Error opening source file.	Read check.
RA03	Error opening source file.	Unit inoperable
RA04	Error opening source file.	File not found
RA05	Error creating relo file.	Search check.
RA06	Error creating relo file.	Read check.
RA07	Error creating relo file.	Unit inoperable
RA08	Error creating relo file.	Duplicate file name.
RA09	Error creating relo file.	Write protect.
RA10	Error creating relo file.	File area full.
RA11	Error opening work file.	Search check.
RA12	Error opening work file.	Read check.
RA13	Error opening work file.	Unit inoperable
RA14	Error opening work file.	File not found
RA15	Work file size too small.	
RA17	Error reading source file.	Seach check.
RA18	Error reading source file.	Read check.
RA19	Error reading source file.	Unit inoperable
RA20	Error reading source file.	Missing EOF
RA21	Error writing relo file.	Search check.
RA22	Error writing relo file.	Verify check.
RA23	Error writing relo file.	Unit inoperable
RA24	Error writing relo file.	File area full
RA25	Error writing relo file.	Write protect.
RA26	Error reading work file.	Search check.
RA27	Error reading work file.	Read check.
RA28	Error reading work file.	Unit inoperable
RA30	Error writing work file.	Search check.
RA31	Error writing work file.	Verify check.
RA32	Error writing work file.	Unit inoperable
RA33	Error writing work file.	Work file full.
RA34	Error writing work file.	Write protect.
RA35	Segment load error.	
RA36	Parameter format error.	
RA37	Error opening list file.	Search check.
RA38	Error opening list file.	Read check.
RA39	Error opening list file.	Unit inoperable
RA40	Error opening list file.	File not found.
RA41	Error opening list file.	Write protect.
RA42	Error writing list file.	Search check.
RA43	Error writing list file.	Verify check.
RA44	Error writing list file.	Unit inoperable.
RA45	Error writing list file.	File area full.
RA46	Error writing list file.	Write protect.
RA47	Incorrect file allocation for diskette reload.	

9.10.12. RENAME Error Codes

RE01	Error reading directory.	Search check.
RE02	Error reading directory.	Read check.
RE03	Error reading directory.	Unit inoperable.
RE04	Specified file not found.	
RE05	Error writing directory.	Search check.
RE06	Error writing directory.	Verify check.
RE07	Error writing directory.	Unit inoperable.
RE09	Error writing directory.	Write protect.
RE10	New file name not unique.	
RE11	Missing or duplicate option.	
RE12	Parameter format error.	

9.10.13. TMOVE Error Codes

TM01	Tape unit is off line or formatter turned off.
TM02	Attempt to backspace past BOT.
TM03	Attempt to position past double end of file.
TM04	Attempt to position past EOT.
TM05	Read error during list option.
TM06	Parmeter format error.

9.10.14. UPDATE Error Codes

UP01	Error opening input or merge.	Search check
UP02	Error opening input or merge.	Read check.
UP03	Error opening input or merge.	Unit inoperable.
UP04	Error opening input or merge.	File not found.
UP05	Error creating output file.	Search check.
UP06	Error creating output file.	Read check.
UP07	Error creating output file.	Unit inoperable.
UP08	Error creating output file.	Duplicate file.
UP09	Error creating output file.	Write protect.
UP10	Error creating output file.	File area full.
UP11	Error reading input or merge.	Search check
UP12	Error reading input or merge.	Read Check
UP14	Error reading input or merge.	Missing EOF.
UP15	Error writing output file.	Search check.
UP16	Error writing output file.	Verify check.
UP17	Error writng output file.	Unit inoperable.
UP18	Error writng output file.	File area full.
UP19	Error writng output file.	Write protect.
UP20	Error reading command file.	Search check.
UP21	Error reading command file.	Read check.
UP22	Error reading command file.	Unit inoperable.
UP23	Error reading command file.	Source format.
UP24	End of file on command input.	
UP25	UPDATE command error.	Unrecognized type.
UP26	UPDATE command error.	Syntax error.
UP27	UPDATE command error.	Line number error.
UP28	Parameter format error.	
UP29	Segment load error.	

9.10.15. VERIFY Error Codes

VE01	Error reading.	Search check.
VE02	Error reading.	Read check.
VE03	Error reading.	Unit inoperable.
VE04	Specified file not found.	
VE05	File label match verify error.	
VE06	Diskette serial number match verify error.	
VE07	Parameter format error.	

9.10.16. ZAP Error Codes

ZA01	Error opening input file. Search check.
ZA02	Error opening input file. Read check.
ZA03	Error opening input file. Unit inoperable.
ZA04	Error opening input file. File not found.
ZA05	Error creating output file. Search check.
ZA06	Error creating output file. Read check.
ZA07	Error creating output file. Unit inoperable.
ZA08	Error creating output file. Duplicate file name.
ZA09	Error creating output file. Write protect.
ZA10	Error creating output file. File area full.
ZA11	Error reading input file. Search check.
ZA12	Error reading input file. Read check.
ZA13	Error reading input file. Unit inoperable.
ZA14	Error writing output file. Search check.
ZA15	Error writing output file. Verify check.
ZA16	Error writing output file. Unit inoperable.
ZA17	Error writing output file. File area exhausted.
ZA18	Error writing output file. Write protect.
ZA19	Error reading command file. Search check.
ZA20	Error reading command file. Read check.
ZA21	Error reading command file. Unit inoperable.
ZA22	Error reading command file. Missing end of file.
ZA23	Error reading core image work file. Search check
ZA24	Error reading core image work file. Read check.
ZA25	Error reading core image work file. Unit inoper.
ZA26	Error writing core image work file. Search check
ZA27	Error writing core image work file. Verify check
ZA28	Error writing core image work file. Unit inoper.
ZA29	Error writing core image work file. Write prot
ZA30	End of command file encountered.
ZA31	Parameter format error.
ZA32	ZAP command format error.
ZA33	L command invalid (no input file)
ZA34	S command invalid (no output file or segment already stored.
ZA36	Wrong system (wrong model TPU, external memory present).
ZA37	Segment load error.
ZA38	ZAP V reject due to data mismatch.
ZA39	Error writing load cassette.

10. SPD/DOS Assembler Guide

The SPD/DOS Assembler is a utility which allows the user to program directly in mnemonics. It permits full access to the capabilities of the processing unit (TPU) while relieving the programmer of the mammouth tasks associated with writing machine code.

The mnemonics recognized by the assembler are identical to those given in section 2., Instruction Repertoire. In addition, various pseudo-ops are available which control the assembly process itself.

The ASSEMBLE utility allows assembling source statements into absolute code in a form suitable for loading by the DOS loader. The RASSEMBL utility assembles source code into relocatable modules suitable for inclusion into absolute assemblies through the IN pseudo-op. This section is intended as a quick reference guide to the SPD/DOS assembler and relocatable assembler utilities. For a complete description of the syntax of the language and more detailed information, consult the SPD Symbolic Assembly Language Reference Manual.

10.1. Assembly Source Format

Source line length: 80 characters, but the SPD/DOS editor restricts line length to 64 characters.

Character set: Upper case ASCII set, codes X'20' - X'5F'.

Special characters:		Typical Use:
$	Dollar	Assembly symbols
&	Ampersand	Title modification
'	Quote	Delimiter
(	Open parenthesis	Expressions
)	Close parenthesis	Expressions
*	Asterisk	Multiplication, comment
%	Percent	Names and Labels
#	Hash	Comments
+	Plus	Expressions
,	Comma	Operand separation
-	Minus	Subtraction
.	Period	Operands
/	Slash	Division
0-9	Digits	***
=	Equals	Literals, expressions
@	At	Indexing
A-Z	Letters	***

10.1.1. Source Line Format

Column Conventions:

```
1               10          16          30
Label           Opcode      Operands    Comments
```

Comment Lines:

```
1
*       Comments
#       Deletable Comments
```

Title Modification

```
1
% Main Title Modification
' Sub-title Modification
```

10.2. Expressions

Data Constant Specification

n or D'n'	Decimal integer
B'n'	Binary integer
O'n'	Octal integer
X'n'	Hexadecimal integer
'a'	Single character ASCII constant *
'ab'	Double character ASCII constant

Assembler Symbols

$	Current value of Location counter
$B	16 bit value set during Assembler load
$D	Duplication count
$L	Current value of Load counter
$X or $C	The location of the CUR: FFE
**	Zero constant operand

* To include a quote one simply writes ''''.

Assembler Operators
Order of Precedence

Highest

* /
+ -
.RS. .LS.
.EQ. .NE. .GT. .LT. .LE. .GE.
.AND.
.OR.
.XOR.

10.2.1. ASSEMBLE/RASSEMBL: Options

Load:

```
ASSEMBLE<,Options> Sfile<,b<,Ofile><,Label>>
AS<,Options> Sfile
```

RASSEMBL: Relocatable Assembly

Load:

```
RASSEMBL<,Options> Sfile<,b<,Rfile><,Label>>
RA<,Options> Sfile
```

Note the nesting of parameters. This specifies the parameters that may be omitted.

Options:

A	Alternate Unit. The object or relocatable file is generated on the unit opposite from that implied by the call.
B	Both pass. Print on pass 1 and pass 2.
C	Clean. Enforce LIF 0: IF/ENDF and deleted code are never printed.
*D	Definitions. INclude DOS standard definitions module. Use of this option can be avoided by INcluding D&DEFS at the appropriate location in the code.
E	Erase duplicate object file.
F	Full list. Enforce LIF 2 mode: All IF/ENDF, plus deleted

code is printed.

*G Generate IN code: Enforce LIN 1, INcluded code is listed.

*H Hold IN code: Enforce LIN 0. INcluded code is listed only for errors.

I Inhibit object file production.

K Kill Hash comments. Enforce List 2 mode. Ejects and subtitles are used to format the listing.

L Enforce List 3. Hash comments are listed.

N No Printer: Print source listing in LIST file.

*O Omit. Omits literal cross references.

P Paper save. Enforce List 1: eject and subtitles are treated as comments. Hash comments are killed.

Q Quick Assembly: XREF 0. No cross reference is generated.

R Reference unassembled: Enforce XREF 2. Cross reference for deleted IF/ENDF code is generated.

S Short List: Print only errors.

T Table of Contents. Shows the initial page number of each subtitle line. It is printed immediately before the listing of the first subtitle line.

U Unlist deleted code: Enforce LIF 1. Code deleted, plus IF/ENDF lines in false IF ranges, is not printed.

V Verify writes to object. WORK and LIST are never verified.

X Cross Reference: Enforce XREF 1. Collect cross references in assembled areas.

* ASSEMBLE utility only

10.2.2. Assembler Pseudo-Operations

FUNCTION	KEY	MNEMONIC		MEANING
DATA DEFINITION		ADDR	ed,ed...	Generate two byte data,
		ADDR	n<ed>	n=explicit decimal integer.
		ALGN		Align data to word boundary
		BSS	n	Define n storage bytes
		BSZ	n	Zeros n storage bytes
		BYTE	ed,ed...	Generate byte data
		BYTE	n<ed>	n=explicit decimal integer
	a,	DAC	<address>	Generate address const
	a,	DAC*	<address>	Set indirect bit on
		HEX	<hex>	Generate <hex> data
		LBL		Generate label data
		LTXT	d<string>d	Generate lower case text.
		LTX8	d<string>d	Set MSB on
		TEXT	d<string>d	Generate text string
		TXT8	d<string>d	Set MSB on
	a,	WORD	ed,ed...	Generate word data
	a,	WORD	n<ed>	n=Explicit decimal integer
PROGRAM CONTROL	x	BOOT		Set Boot mode
	a,	BSL	n	Specify local literals n=Number of bytes, even, available
		CNFG	n	Specify Configuration n=10,20,24,25,0
	r,	DEF	symbl	Define symbol
		DUP	n	Duplicate source line n times.
	*	EAM	n	Enter Extended Addressing Mode if n=1. If n not given, EAM 1.
		END	<entry>	End Assembly. <entry> optional
		ENDF		End Cond. Assembly
	symbl	EQU	n	Equate symbol to value
	x	ESEG	<entry>	End overlay segment <entry>=Segment entry point, optional
		IF	<test>	Start Conditional Assem <test>=0: Code to ENDF ignored
	a,x,	IN	file,module	Include Relocatable Mod

FUNCTION	KEY	MNEMONIC		MEANING
	x	LIT	<symbl>,,n	Specify main literals n=Value to be placed in main pool
	Label	MOD		Define Relocatable Mod
	x	NOBJ		Turn off object output
	x	OBJ		Turn on object output
	x,l	ORG	<address>	Set Assembly origin
	r	TOP		Top sector code follows
	x,Label	SEG	<address>	Start overlay segment <address>=Origin of segment,optional
	Symbl	SET	n	Set symbol to value
		SIZE	sz,lnk,xsz	Specify memory size sz=Last byte address, Top limit for Lit. table. xsz=External memory size.
	l,x	XORG	<loc><,ld>	Set execution origin loc=Location counter: $, ld=Load counter: $L
	r	XTN	symbl	External symbol(s)
PROGRAM LISTING	x	EJECT		Eject listing
		LIF	n	Control IF listing n=0: Never printed, n=1 false ranges not listed, n=2 Full list
		LIN	n	Control IN listing n=0 No listing except errors
		PAGE	n	Set Page Depth to n
		XREF	n	Set Cross Reference n=0 No Xref, n=1: Xref in Assembled areas n=2: Full Xref
		&title		Main title modification & must start in column 1
		'subtitle		Subtitle modification ' must start in column 1

NOTE: Insert after CNFG

r	copy	File,Module	copy reloc module into a reloc assembly.

KEY:

a	Aligned to word boundary
d	Delimiter: any character not in string
l	Label permitted
n	As defined
r	Relocatable Assemblies Only
x	Absolute Assemblies Only
ed	Effective data
wa	Word Address, must be even
*	SPD 20 Family models Only
Label	required label
symbl	Valid assembler symbol
address	Valid address, as enforced by SIZE

10.3. Addressing Restrictions

CNFG	CORE	SIZE	Low*	High
10(25)	4K	Default	0000	0FFF
20	8K	1FFF	0000	18FF
			7C00	7FFF
	16K	3FFF	0000	39FF
			7A00	7FFF
	32K	7FFF	0000	7FFF
	64K	FFFF	0000	FFFF
24	8K	1FFF	0000	18FF
			7C00	7FFF
	16K	3FFF	0000	38FF
			7C00	7FFF
Compatible	4K	0000	0000	0DFF
			7E00	7FFF
	16K	0000	0000	3DFF
			7E00	7FFF
	32K	0000	0000	7FFF

* Note: Segment 0 Low = 0100

10.4. Error Flags

A	Invalid address
B	Violation of BOOT mode restriction
C	Erroneous character
D	Disk Error for Source or INcluded Data
E	Odd operand whee even required
F	Forward reference not allowed
H	Missing END line
I	IF-ENDF nesting error
L	Label error
M	Multiple definition
N	Numeric error
O	Invalid operation code
P	Parenthesis depth error
Q	Invalid SIZE or CNFG parameter
R	Relocatability error
S	SEG-ESEG sequence error
T	Tabel overflow (Literal or Symbol)
U	Undefined symbol
V	Missing operand
W	Extra operand
X	Invalid load location
Y	Relocatable library or module not found
Z	Symbol undefined due to symbol table overflow
@	Indexing error
=	Invalid literal
*	Invalid use of indirect addressing

11. Machine Codes Quick Reference

SPD 10/25
INSTRUCTION SET

ASSEMBLER MNEMONIC		CODE	EXEC. TIME	ACTION
AD*	ea	10ea	2	ACR = <ea> + ACR
ADI	ea	90ed	1	ACR = ed + ACR
AN*	ea	28ea	2	ACR = ACR.*.<ea>
ANI	ed	A8ed	1	ACR = ACR.*.<ea>
CIO	f,c	C9fc	1	Controller function
CJTRU*	ed,wa	A4ed-	2	PCR = ea
CJFAL*	ed,wa	A0ed-	1	PCR = PCR + 4
CJEQ*	ed,wa	A2ed-	#	PCR = wa ACR=ed
CJGT*	ed,wa	A7ed-	#	PCR = wa ACR>ed
CJGE*	ed,wa	A5ed-	#	PCR = wa ACR>=ed
CJLT*	ed,wa	A1ed-	#	PCR = wa ACR<ed
CJLE*	ed,wa	A3ed-	#	PCR = wa ACR<=ed
CJNE*	ed,wa	A6ed-	#	PCR = wa ACR!ea
CLA		8000	1	ACR = 0
CLL		C00C	1	LIR = 0
CM*	ea	20ea	2	ACR:<ea>
CMC*	=wa	70wa	3	CUR:<wa>
DEC*	wa	58wa	2	<wa> = <wa> - 1
DSB		C007	1	DSB Interrupts
ENB		C006	1	ENB Interrupts
XOR	ed	50ed	1	ACR = ACR.-.ed
HALT		C001	##	STOPS TPU
INC*	wa	50wa	2	<wa> = <wa> + 1
IN2*	wa	60wa	2	<wa> = <wa> + 2
IOR		C008	1	Resets TPU
JFACK*	f,c,wa	CCfc-	#	PCR = wa IF NAK
JTACK*	f,c,wa	C8fc-	#	PCR = wa IF ACK

ASSEMBLER MNEMONIC		CODE	EXEC. TIME	ACTION
JCCO*	wa	8D00-	#	PCR = wa CO = 1
JCEQ*	wa	8A00-	#	PCR = wa EQ = 1
JCEV*	wa	8C02-	#	PCR = wa ACR LSB = 0
JCGT*	wa	8F00-	#	PCR = wa (CO = 1, EQ = 0)
JCGE*	wa	8D00-	#	PCR = wa CO = 1
JCLT*	wa	8900-	#	PCR = wa CO = 0
JCLE*	wa	8B00-	#	PCR = wa EQ = 1 or CO = 0
JCNG*	wa	8801-	#	PCR = wa Result Byte Msb = 1
JCNC*	wa	8900-	#	PCR = wa CO = 0
JCNE*	wa	8E00-	#	PCR = wa EQ = 0
JCOD*	wa	8802-	#	PCR = wa ACR LSB = 1
JCPO*	wa	8C01-	#	PCR = wa Result Byte Msb = 0
JCFAL*	wa	8800-	1	PCR = PCR + 4
JCTRU*	wa	8C00-	2	PCR = wa
JMP*	wa	B8wa	1	PCR = wa
JSR*	wa	78wa	2	<wa> = PCR + 2. PCR = wa + 2
LD*	ea	00ea	2	ACR = <ea>
LDI	ed	80ed	1	ACR = ed
LDC*	=wa	40wa	3	CUR = <wa>
MAC		C005	1	CHR = ACR
MAL		C003	1	LIR = ACR
MCA		C004	1	ACR = CHR
MLA		C002	1	ACR = LIR
NOP		C000	1	Execution Delay: 1 cycle
OR*	ea	30ea	2	ACR = ACR.+.<ea>
ORI	ed	B0ed	1	ACR = ACR.+.ed
RIO	f,c	CAfc	1	ACR = INB
SHL4		C009	1	ACR = Arith Shift Left 4
SB*	ea	18ea	2	ACR = ACR - <ea>
SBI	ed	98ed	1	ACR = ACR - ed
SKP		8800	1	Skip next instruction
ST*	ea	08ea	2	<ea> = ACR
STC*	wa	48wa	2	<wa> 12 LSB = CUR
WAIT		C00F	###	ENB, Interrupt Wait
WIO	f,c	CBfc	1	OTB = ACR
WJMP	wa	8C00-	1	PCR = wa (16 bits)

LEGEND:

c	Device channel
f	Device function
ea	Byte Address, even or odd
ed	effective data
wa	Word Address, must be even
< >	The CONTENTS of the address
.+.	Logical Or
.-.	Logical Exclusive Or
.*.	Logical And
*	Indirect addressing permitted. Literal use permitted if indirect addressing is employed and if memory reference.
=	Literal use permitted.
>	GT
>=	GE
<	LT
<=	LE
!	NE
=	EQ
ACR	Accumulator
CHR	Character Register
CUR	Cursor Register
INB	Input Data Bus
LIR	Line Register
OTB	Output Data Bus
PCR	Program Counter Register

Execution Times:

#	1 cycle if no jump is taken, else 2 cycles
##	Wait for action from programmers console, 1 cycle if no console.
###	Wait for next interrupt, return on NAK to instruction following WAIT.

Indirect addressing: 1 cycle per level

1 TPU cycle = 1.6 microseconds.

Data Comparisons:

Replace : with comparison to be performed (E.G, GT,LT, Cursor:<wa> becomes, JCGT jumps if Cursor.GT.<wa>)

12. Summary of Controller Programming

This section summarizes the INCOTERM Communications Controller command set. For detailed programming information, consult the INCOTERM Communication Controller Reference Manual.

12.1. Asynchronous Controller Command Summary

Command	Function
CIO 0	Reset RTS
CIO 1	General Reset
CIO 2	Set Transmit Mode
CIO 3	Set Receive Mode
CIO 4	Set Line Brake/Priority Data
CIO 4	Set RTS
CIO 12	Mask Interrupts
CIO 8	Unmask Interrupts
WIO 1	Write Data
WIO 2	Set Data Terminal Ready
WIO 4	Reset Data Terminal
WIO 8	Set RTS
RIO 1	Read Data
TIO 0	ACK if Controller Present
TIO 1	ACK if Data Set on Line/No EOT
TIO 2	ACK if No Data Set Error
TIO 4	ACK if No Overrun
TIO 8	ACK if No Line Break/No Cancel

12.1.1. Summary of Merged CIO Commands

Command	Function
CIO 13	General Reset plus Mask Interrupts
CIO 9	General Reset plus Unmask Interrupts
CIO 14	Set Transmit Mode plus Mask Interrupts
CIO 6	Set Transmit Mode plus Set Line Break/Priority Data

Command	Function
CIO 15	Set Receive Mode plus Unmask Interrupts
CIO 11	Set Receive Mode plus Unmask Interrupts

12.2. Asynchronous Cntrlr/SPD-M Multiplexer

Command	Function
CIO 1	General Reset
CIO 2	Set Transmit Mode
CIO 3	Set Receive Mode
CIO 4	Set Priority Data
CIO 12	Mask Interrupts
CIO 8	Unmask Interrupts
WIO 1	Write Data
RIO 1	Read Data
TIO 0	ACK if Controller Present
TIO 1	ACK if No EOT
TIO 2	ACK if No Data Set Error
TIO 4	ACK if No Overrun
TIO 8	ACK if No Cancel

12.3. Party Line Controller Summary

Command	Function
CIO 1	General Reset
CIO 2	Set Transmit Mode
CIO 3	Set Receive Mode
CIO 4	Set Line Break
CIO 12	Mask Interrupts
CIO 8	Unmask Interrupts
WIO 1	Write Data
RIO 1	Read Data
TIO 0	ACK if Controller Present
TIO 4	ACK if No Overrun
TIO 8	ACK if No Line Break

12.4. Synchronous Controller

Command	Function
CIO 2	General Reset
CIO 4	Mask Interrupts
CIO 8	Unmask Interrupts
CIO 10	Set Transmit Mode
CIO 6	Set Receive Mode
WIO 0	Write Data
RIO 0	Read Data
TIO 0	ACK if Controller Present
TIO 2	ACK if No Data Set Error/Cancel
TIO 4	ACK if No Overrun
TIO 8	ACK if Data Set On Line/EOT

12.5. Synchronous Contrlr/SPD-M Multiplexer

Command	Function
CIO 1	Set Priority Data
CIO 2	General Reset
CIO 4	Mask Interrupts
CIO 8	Unmask Interrupts
CIO 10	Set Transmit Mode
CIO 6	Set Receive Mode
WIO 0	Write Data
RIO 0	Read Data
TIO 0	ACK if Controller Present
TIO 2	ACK if No Cancel
TIO 4	ACK if No Overrun
TIO 8	ACK if No EOT

DOCUMENT PRINTED BY: DOC V2.03 76-10-07-0930

SOURCE FILE NAME: TEN

LABEL: SPD 10/25 MANUAL 76-09-09-0900

DSN: TEN

NUMBER OF DETECTED ERRORS: 1

65 WALNUT STREET
WELLESLEY HILLS, MA 02181
(617) 237-2100