13290A/2649A

Reference Manual

Part Number: 13290-90003
Printed: October 1977

DATA TERMINAL
TECHNICAL INFORMATION

HEWLETT PACKARD

Printed In U.S.A.

13290A/2649A

Reference Manual

Part Number: 13290-90003
Printed: October 1977

PREFACE

This manual provides overview documentation for the HP 13290 Development/2649 Mainframe terminals. It serves as a guide to related documentation and contains techniques for modifying or adding to the basic terminal firmware. It also provides a brief description of the terminal hardware. Detailed descriptions of the terminal are contained in the Technical Information Package (13255A for hardware and Option 003 for 2645 firmware) and information on the firmware development process can be found in the Firmware Support Package (13256A).

The manual assumes that you are familiar with Intel 8080 assembly language or a similar microcode language. In addition it assumes that you are familiar with HP 2645 terminals and how they operate. Information on the 8080 Assembly language and the HP 2645 terminal can be found in the following reference documents:

- *Intel 8080 Assembly Language Programming* (MCS-482-0275/15K)
- *HP 2645A User's Manual* (02645-90001)
- *HP 2645A Reference Manual* (02645-90003)
- *HP 2645A Service Manual* (02645-90005)
- *HP 13255A Technical Information Package* (13255-91000)
- *HP 2645A Operating System Microcode Listings* (13255-90003)

This manual is made up of the following sections and appendices:

Section I – Introduction. This section describes the HP 13290 and 2649 terminals. In addition, a brief description of HP 2645 terminal architecture and firmware organization is presented. A list of related documentation is also provided.

Section II – Configuration and Turn-On. This section describes how boards are installed in the 2649A terminal and how to load firmware code into a 13290A terminal.

Section III – Firmware Development. This section describes the various techniques that can be used to develop custom firmware packages for the terminal.

Section IV – Hardware Development. This section describes briefly the techniques for developing additional hardware for use with the terminal.

Section V – Support Accessories. This section indicates the purpose and use of the 13291, 13292, 13293, and 13295 accessories.

Section VI – Main Code Module. This section describes the Main Code module and how it is interfaced.

Section VII – Keyboard Code Module. This section describes the Keyboard Code module and how it is interfaced.

Section VIII – Standard Device I/O. This section describes how to perform I/O using the GETIO/ PUTIO routines and the standard device drivers (left CTU, right CTU, display, and printer).

Section IX – Alternate I/O. This section describes the Alternate I/O Code Module and how it can be implemented.

Section X – Data Comm Module. This section describes the Data Communications Code module and how it is interfaced.

Appendix A – Program Reference Tables. This appendix provides programming reference information.

CONTENTS

CONTENTS (continued)

ILLUSTRATIONS

TABLES

Section I. INTRODUCTION

HP 13290/2649 STANDARD AND OPTIONAL ASSEMBLIES

The HP 13290 terminal is designed to allow you to develop a custom terminal for specific applications. It uses RAM memory modules which allow you to selectively load and modify terminal firmware.

The standard and optional assemblies for the HP 13290A and HP 2649A terminals are given in table 1-1. Note that there is a total of 15 card slots available in both the 13290 and the 2649. The right hand column of table 1-1 tells how many slots are used by each assembly. The total card slots used by all assemblies in one 13290 or 2649 cannot exceed 15.

Table 1-1. Standard and Optional Assemblies

Assembly		# of Slots Used
13290A	Development Terminal	13
-013	5 Mini cartridges	
-015	50 Hz operation	
13291A	4K PROM Module	1
-001	Zero insertion sockets	
13292A	8K Writable Control Store (WCS)	1
-001	5-Wide top plane connector	
13293A	Diagnostic Module	1
13294A	5-Day Training Course	
13295A	Keycap Kit	
2649A	Terminal	4
-007	Dual cartridge tape	2
-100	Upper case display ROM	—
-101	Lower case display ROM	—
-200	2645A Keyboard and interface	1
-201	Simplified keyboard and interface	1
-400	24K ROM module	1
-401	8K ROM, 1K RAM module	1
-402	16K ROM module	1
-500	2645A Basic firmware	—
-501	Diagnostic/loader ROM	—
-600	2645A Keyboard firmware	—
-801	Blank keyboard overlay	—

HP 13290/2649 ACCESSORIES

The HP 13290A and HP 2649A terminals can use the same accessories as the standard HP 2645A terminal. A brief list of accessories is given in table 1-2.

Table 1-2. Accessories

Accessory		# of Slots Used
13231A	Display Enhancements	1
-201	Math set	—
-202	Line drawing set	—
-203	Large character set	—
13234A	4K Byte Memory Module	1
13238A	Duplex Register	1
13245A	PROM Character Set Generation Kit	1
13255A	Technical Information Package (T.I.P.)	
-001	2640B Source listing	
-002	2644A Source listing	
-003	2645A Source listing	
13256A	Firmware Support Package	
13260A	Standard Asynchronous Communication Interface	1
-002	Delete ROM/overlay	
13260B	Extended Asynchronous Communication Interface	1
-002	Delete ROM/overlay	
13260C	Asynchronous Multipoint Communication Interface	1
-001	Monitor mode	
-002	Delete ROMs/overlay	
13260D	Synchronous Multipoint Communication Interface	1
-001	Monitor mode	
-002	Delete ROMs/overlay	

Once you have developed new firmware or modified the existing firmware to suit your needs, the HP 2649 terminal can be used to execute the code. The HP 2649 terminal can be ordered in a variety of configurations. This allows you to use only those modules that are required for your application.

In addition to the basic HP 2645 terminal documentation (User Manual, Reference Manual, and Service Manual) the HP 13290 is supported by the following:

- HP 13255A Technical Information Package (T.I.P.)
- Firmware Support Package (F.S.P.)
- Firmware Listings (HP 2640B, 2644A, 2645A)

The remainder of this section describes the organization of the terminal firmware. The terminal hardware structure is described here only briefly. If you require more detailed information on terminal hardware refer to the appropriate terminal Reference Manual and the Technical Information Package (part number 13255-91000).

HARDWARE ORGANIZATION

The following topics present a brief general overview of the hardware organization of the 13290/2649 terminals.

The Processor Board

The processor board of the HP 2641, 2645, and 2648 terminals contains the Intel 8080 microprocessor that controls the operation of the terminal.

INSTRUCTION SET. The instruction set used by the HP 2641, 2645, and 2648 terminals is that of the Intel 8080 microprocessor. For information on the assembly language formats and how to write 8080 assembly language programs refer to the *Intel 8080 Assembly Language Programming Manual* (part number MCS-482-0275/15K).

ADDRESS SPACE. Because it uses 16 address lines, the processor can directly address up to 64K bytes of memory. 4K of these bytes are reserved for memory-mapped I/O which is discussed as a separate topic below.

TOP VS. BOTTOM PLANE MEMORY ACCESS. All of the boards in the terminal share a common bottom plane bus containing 16 address lines, 8 data lines, and various other signal lines. Access to the bus is granted on a priority basis. If two or more modules request access to the bus simultaneously, the module with the highest priority is granted access to it and the lower priority module(s) must wait.

The priorities are established by the physical location of the boards within the terminal. The one closest to the power supply has the highest priority and the remaining boards have progressively lower priorities, with the board furthest from the power supply having the lowest.

Because of the overhead involved in anticipating and resolving multiple requests for the bus, the average time for fetching an instruction over the bottom plane bus is 800 nanoseconds. This fetch time, while adequate for many of the modules, is much too slow for the execution of processor microprograms. For that reason, the terminal also includes a top plane bus that connects the processor board to the control memory board and three 8K RAM memory boards. The average time for fetching an instruction over the top plane bus is only 400 nanoseconds because there is no overhead for resolving priority conflicts.

When the processor board needs to fetch an instruction, it automatically accesses the top plane bus. If after 120 nanoseconds no acknowledgement signal is received, the processor reissues the fetch, only this time using the bottom plane bus.

ROM VS. RAM ACCESS. Typically the 13290 has 24K of memory devoted to code plus up to 12K of display memory.

Using a five-board top plane connector, the 13290 can accommodate the processor board, a ROM-based control memory board, and three 8K RAM memory boards. The control memory contains a 2K-byte binary loader ROM for reading microcode from the left cartridge tape unit into the 24K of RAM. At the conclusion of the loading process, the binary loader automatically disables the ROM-based control memory board. The contents of the three 8K RAM boards (just read in from the CTU) thus become the first 24K of the terminal's memory.

MEMORY-MAPPED I/O. The memory addresses 32K through 36K (decimal) are reserved for memory-mapped I/O, an addressing scheme whereby the processor can access any of the I/O modules residing in the terminal. The format of these addresses is shown in figure 1-1.

Address Format:

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

I/O Module Address Assignments:

0001	:	Asynchronous Data Comm Interface
0011	:	Keyboard Interface
0101	:	Serial Printer Interface
0111	:	Display Memory Access
1011	:	Cartridge Tape Interface
1100	:	High-Speed Parallel Interface
1110	:	Multi-Point Data Comm Interface

Figure 1-1. I/O Module Addressing

The high-order four bits (15-12) are always set to 1000, bits 11-8 specify the desired module, and bits 7-0 can be used for addressing specific components, such as individual registers, within the particular I/O module. Although it does not affect the programmatic use of these addresses, you should be aware that bits 4 and 8 of these addresses are physically swapped on the bottom plane address lines (that is, if you take a probe and examine address lines 4 and 8 you will be reading bits 8 and 4, respectively).

For example, when the processor wants to transfer a byte from the keyboard interface to one of its registers, it issues the appropriate "move" instruction and specifies the hexadecimal address 8300 (1000 0011 0000 0000). By convention this memory address specifies the keyboard interface.

FUNCTIONS OF THE IN/OUT INSTRUCTIONS. The IN and OUT instructions of the 8080 instruction set are not used for transferring data to and from external devices. Instead the OUT instruction is used for setting the contents of the mode latch and the IN instruction is not used at all.

The mode latch is used for enabling, disabling, and resetting the 10 ms timer and for enabling and disabling certain interrupts. The mode latch bit definitions are shown in table 1-3.

Table 1-3. Mode Latch Bit Definitions

Mode Bit	State	Meaning
0	1	10 Millisecond Timer On
1	0	Timer Interrupt Acknowledged
	1	Timer Reset for Next Interrupt
2	1	Firmware Interrupt Request
3	1	$\overline{\text{INT20}}$ Interrupt Disabled
4	1	Data Comm Interrupts Disabled
5	1	Timer Interrupts Disabled
6	1	Poll Interrupts
7	1	Disable Top Plane ROM

To disable the timer you issue an OUT instruction to clear bit 0 of the latch, making sure not to alter the remaining bits in the mode latch.

To reset the 10 ms timer you issue an OUT instruction to first clear bit 1 of the mode latch (i.e., set it to a zero) and then issue another OUT instruction to set it to a one.

To disable data comm interrupts, for example, you use an OUT instruction to set bit 4 of the mode latch to a one. In this case the data comm will still issue interrupt requests, but no action is taken as a result of them.

When setting or clearing a particular bit of the mode latch, care should be taken to ensure that the other bits in the latch are not disturbed.

VECTORED INTERRUPTS. The memory locations 10B through 70B are used for responding to interrupts. When an interrupt occurs, control transfers to the appropriate memory location as shown in table 1-4. The memory location, in turn, passes control to the particular segment of code that is designed to handle the interrupt. You will notice that the 10 ms timer has its own separate interrupt address.

Table 1-4. Hardware Interrupt Addresses

Priority	Interrupt Address	Source
Lowest	10B	Firmware
:	20B	(Not used)
:	30B	10 Millisecond Timer
:	40B	Data Comm Cards (attention #1)
:	50B	CTU Cards (attention #2)
:	60B	(Not used)
Highest	70B	Test Point

Memory Modules

After you have developed your code using the 13290, you have a variety of memory modules available for storing the code in a 2649. The various alternatives, along with certain criteria for choosing one over another, are shown in table 1-5. The overall memory layout of the terminal with regard to types of memory modules is illustrated in figure 1-2.

Table 1-5. 2649A Code Storage Alternatives

	Module Density	**Minimum Quantity**	**Flexibility**	**Speed**	**Non-Volatile**
Writable Control Store 13292A	8K bytes	None	High. (Can't be used for display storage).	High	No
Memory Module 13234A	4K bytes	None	Maximum	Low	No
PROM Module 13291A	4K bytes	None	Good. (PROMs can be reprogrammed)	Low	Yes
Fast ROM 2649A-400	24K bytes	~1000	Low	High	Yes
Slow ROM 2649A-401	8K bytes (plus 1K RAM)	~100	Low	Low	Yes
Slow ROM 2649A-402	16K bytes	~100	Low	Low	Yes

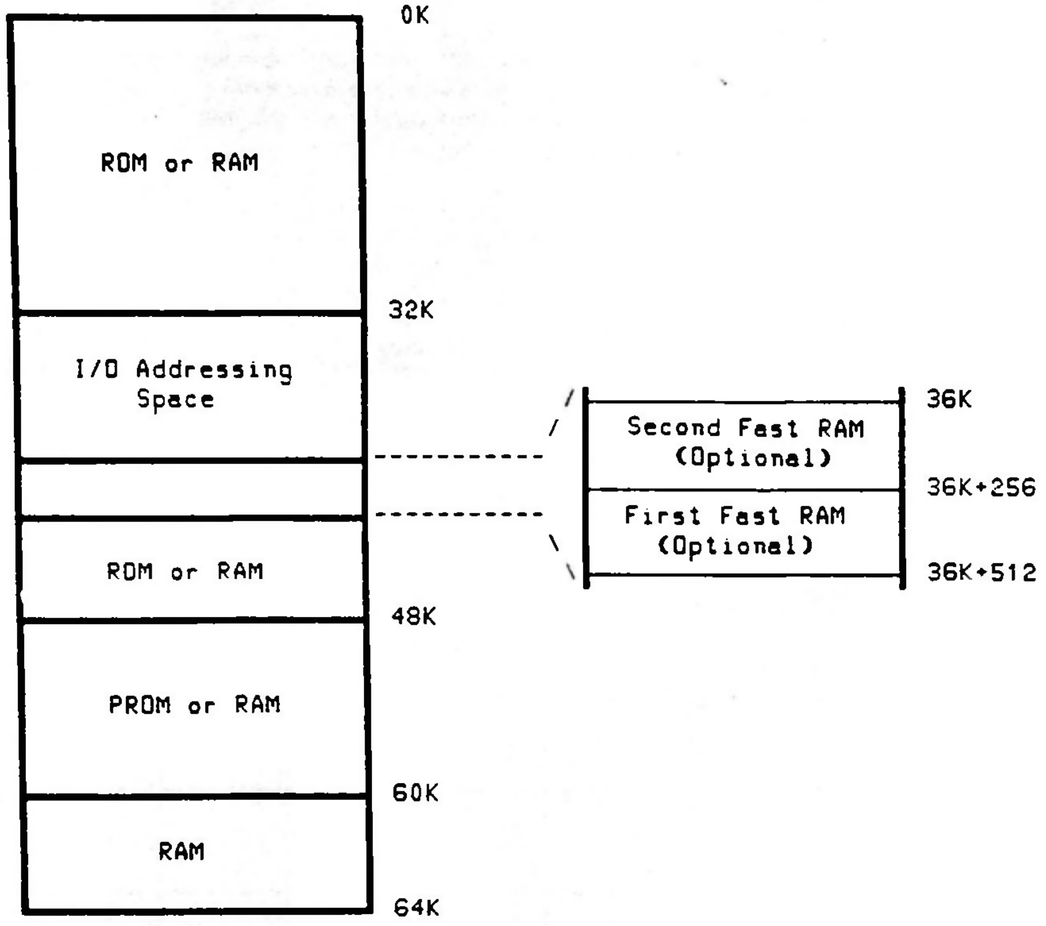


Figure 1-2. Overall Memory Map

DMA and Display Memory

Display memory consists of one to three 4K RAM boards and can be accessed only by way of the bottom plane bus. Control memory, on the other hand, may consist of any of the following and is accessed by way of the top plane bus:

- A ROM-based control memory board (containing up to 24K of read-only memory).
- Three 8K RAM boards.
- Two ROM-based control memory boards (each containing up to 24K of read-only memory).
- A ROM-based control memory board (containing up to 24K of read-only memory) and three 8K RAM boards.

The display memory access (DMA) board fetches data over the bottom plane bus from the display memory RAM boards to continually refresh the CRT screen. The DMA board contains two 80-character buffers. At any given time one of the buffers is being used for refreshing the CRT display while the other is being filled with the next line from display memory. When the contents of the first buffer have been displayed and the second buffer is filled, the two buffers reverse their roles and the process is repeated.

When filling one of its buffers from display memory, the DMA board can recognize "end-of-line" control codes. When it detects such a code the DMA board automatically fills the remainder of the current buffer with special "fill" codes, thus reducing some of its use of the bottom plane bus.

Display memory begins at the highest address (FFFF or 64K) and proceeds downward. The lower portion of display memory can be used for storing some of your own microcode or data (i.e., code or data that is to be used in execution, not displayed). You obtain such access to display memory by manipulating a "fence", named DSPBGN, within display memory. If you have 4K of display memory, DSPBGN is normally located at 60K. If you have 8K of display memory, DSPBGN is normally located at 56K. If you have 12K of display memory, DSPBGN is normally located at 52K. If you move DSPBGN up to 62, 58, or 54K, respectively, you have effectively converted the lower 2K bytes of display memory into additional code or data storage area.

FIRMWARE ORGANIZATION

The processor is capable of directly addressing up to 64K bytes of memory. In general, the first 48K of memory (0-48K) is used for code, the next 4K (48K-52K) is used for buffer space, and the last 12K (52K-64K) is used to store display data. The existing code takes up 24K and is located on the Control Memory PCA. The remaining 24K of code space is available for use in special application terminals (HP 2645S, HP 2641, etc), display memory, or custom terminal firmware. In order to use this additional code space a second Control Memory PCA can be used. (A RAM memory board can also be used.) Figure 1-3 provides a map of terminal memory.

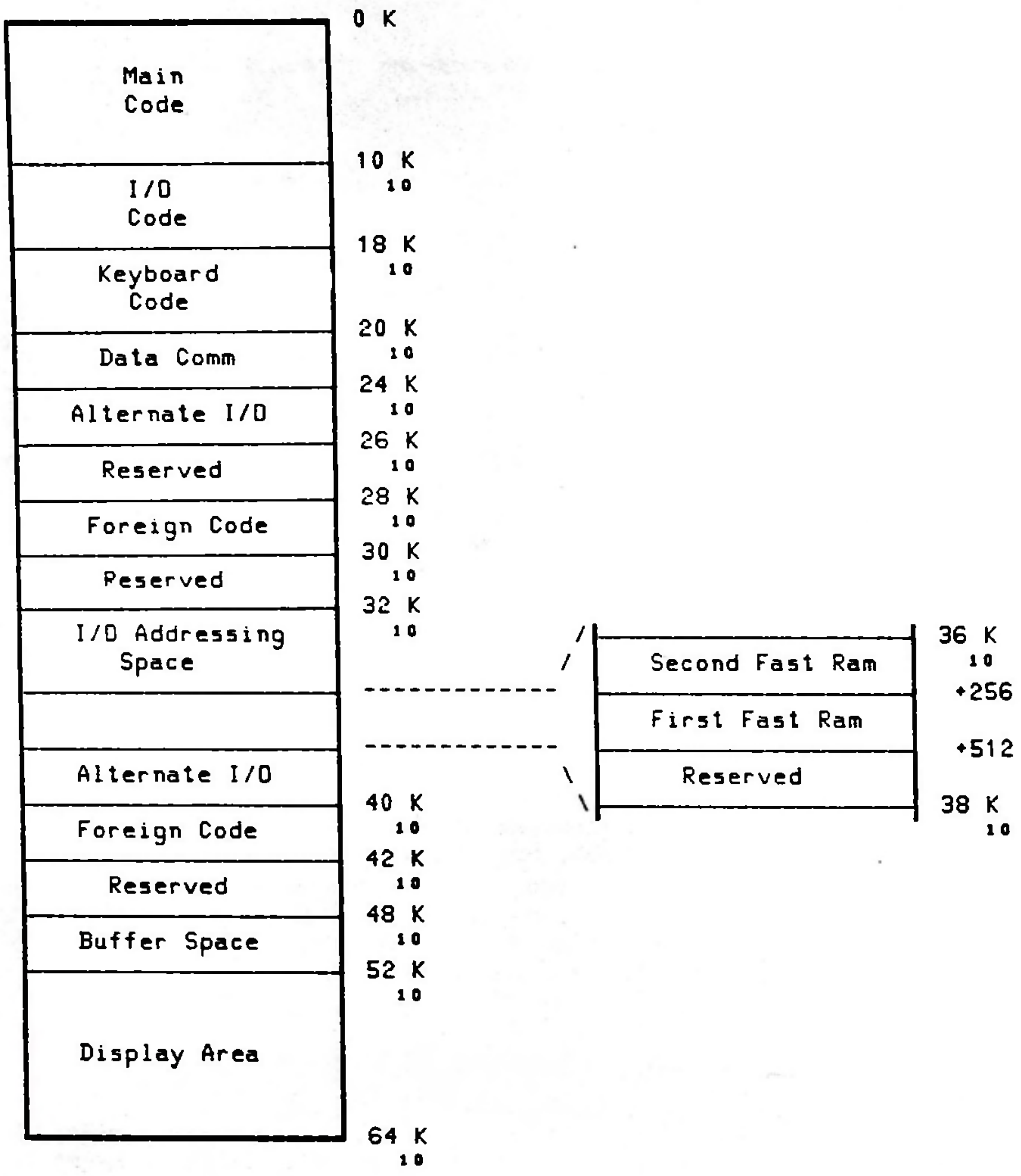


Figure 1-3. Memory Allocation Map

The terminal firmware is organized into modules. Each module is responsible for controlling a major terminal function. The modules are assigned to a particular block of terminal memory and contain their own variable storage areas.

- Main Code Module
- Device Support Code Module
- Keyboard Code Module
- Data Communications Code Module
- Alternate I/O Code Module

The modules are oganized as shown in figure 1-4. The standard terminal uses only the first four modules. The Alternate I/O module is reserved for user defined I/O operations. Each of the modules is discussed in more detail in later sections.

In addition to the code modules, a portion of the RAM memory display storage area is used to hold pointers and variables for the various code modules. Figure 1-5 shows the assignment of display memory storage.

The standard firmware is divided into 2K partitions. Each partition is stored in a single ROM. This allows modification of individual partitions of code without affecting all of the ROMs.

Each partition is formatted as follows:

> bytes 1 and 2 = code revision and chip #
> bytes 3 thru 3774 (octal) = code
> bytes 3775 through 3777 = data check characters

The first two bytes are used to identify the code version and to verify that the ROM chip is installed in the proper location. The first byte contains a value from "P" to "—" (0101 0000 to 0101 1111). The upper four bits are always 0101 and the lower four bits indicate version 0 to 15. The second byte is set to the most significant 8 bits of the memory address used to access the chip. This second byte is checked by the diagnostic to ensure that the ROM has been installed in its proper location.

The last three bytes in each partition (ROM) contain a 16-bit CRC-16 remainder and an 8-bit checksum for the bit pattern in the partition. The checksum is contained in the last byte.

DISPLAY MEMORY

The Display Memory area is used to store display data, variables, and pointers used by the firmware. Figure 1-5 shows the organization of display memory.

The upper portion of display memory (FE00-FFFF) contains code variables, the middle (FC00-FE00) contains I/O device buffers, and the remainder (52K-FC00) is used to hold display data. If there is no memory installed and configured as the data comm buffer space (48K-52K), the firmware will automatically allocate data comm buffers from the display area beginning at the lowest address of the display area. The remaining display area is used to store display data.

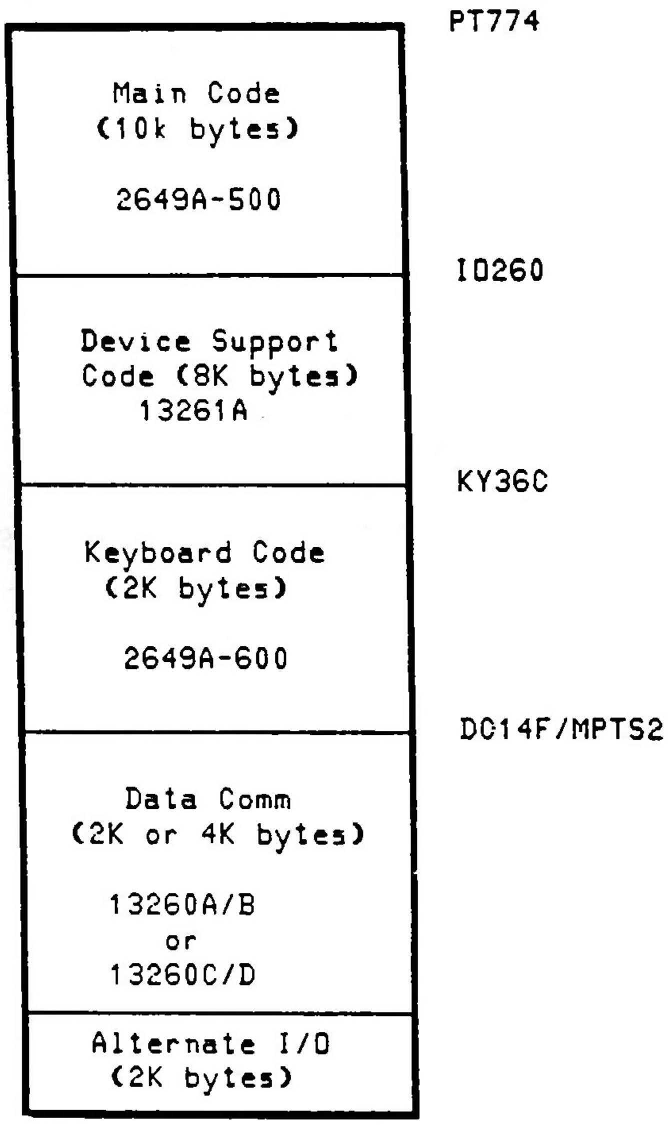


Figure 1-4. Firmware Code Modules

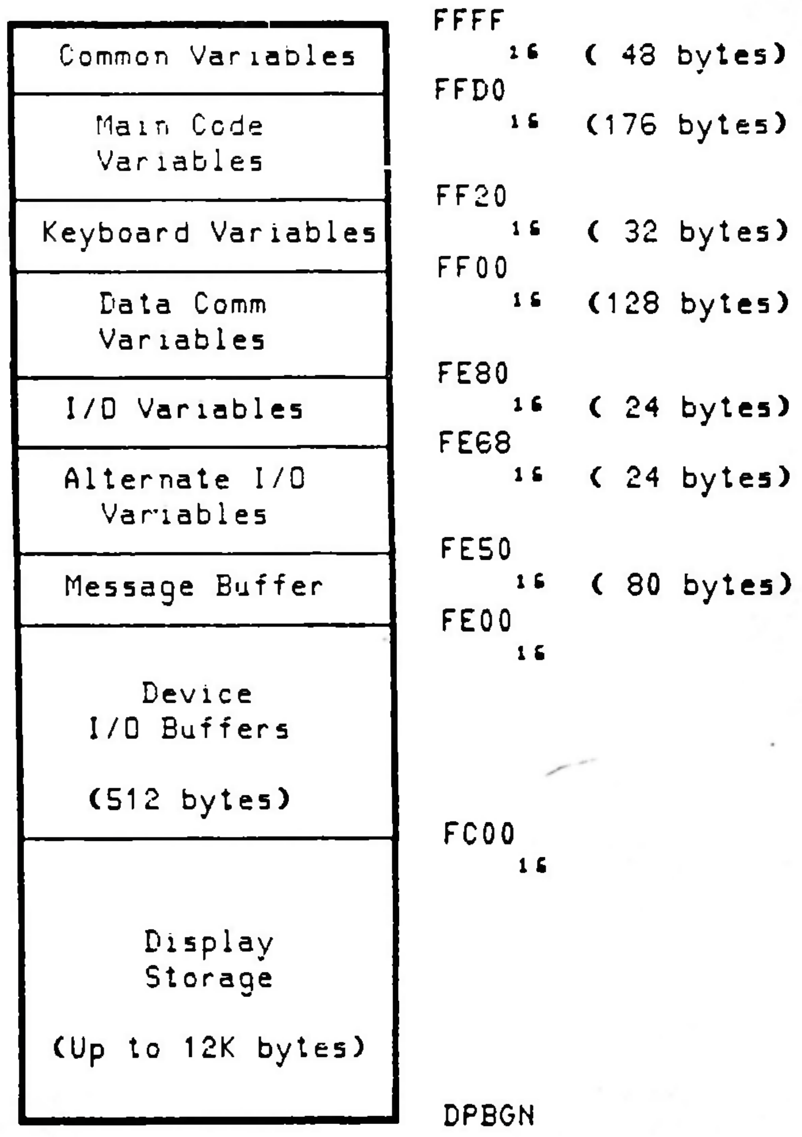


Figure 1-5. Display Memory Allocation Map

A minimum of 4K of display memory (60K-64K) is used in the base terminal configuration. Note that as additional memory is added to the terminal, each succeeding board is configured with a lower starting address. Figure 1-6 shows how additional display memory is allocated. The display area must be configured as one contiguous block.

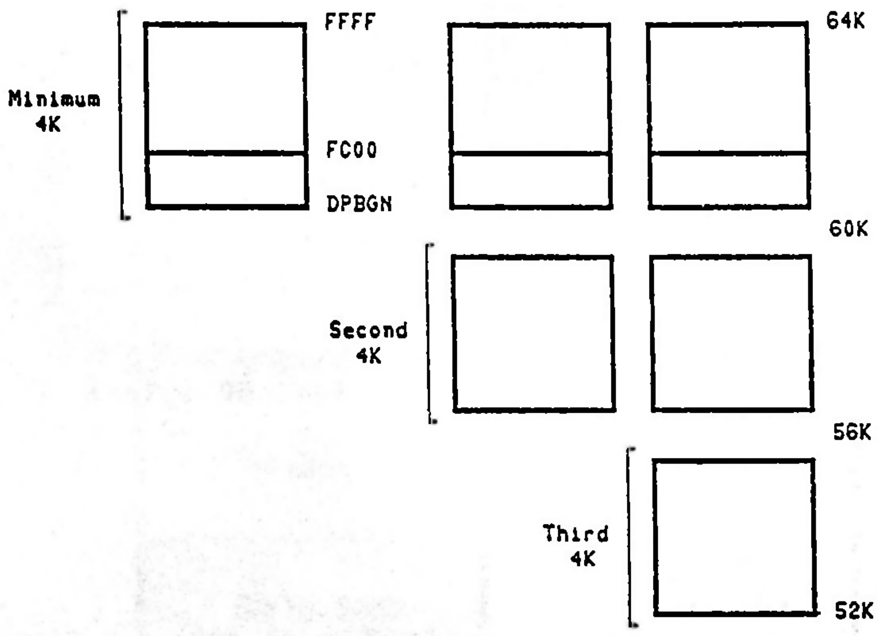


Figure 1-6. Effect of Additional Display Memory

FAST RAM MEMORY

In addition to the code variables stored in the display memory area, the code modules use a small amount of fast RAM storage for frequently used data. This RAM is called "fast RAM" because it is accessed by the terminal's processor over the top plane connector without waiting for the bot-tomplane bus protocol. The RAM serves as a scratch pad memory.

This RAM storage is located begining at 9100 (base 16) and consists of at least one block of 256 bytes. The first 256 byte block of RAM memory (9100-91FF) is provided on the Control Memory PCA. Figure 1-7 shows the organization of the first fast RAM memory. If a second Control Memory PCA is used, its RAM contains addresses 9000-90FF. This second RAM is not used by the standard terminal code and is available for custom applications.

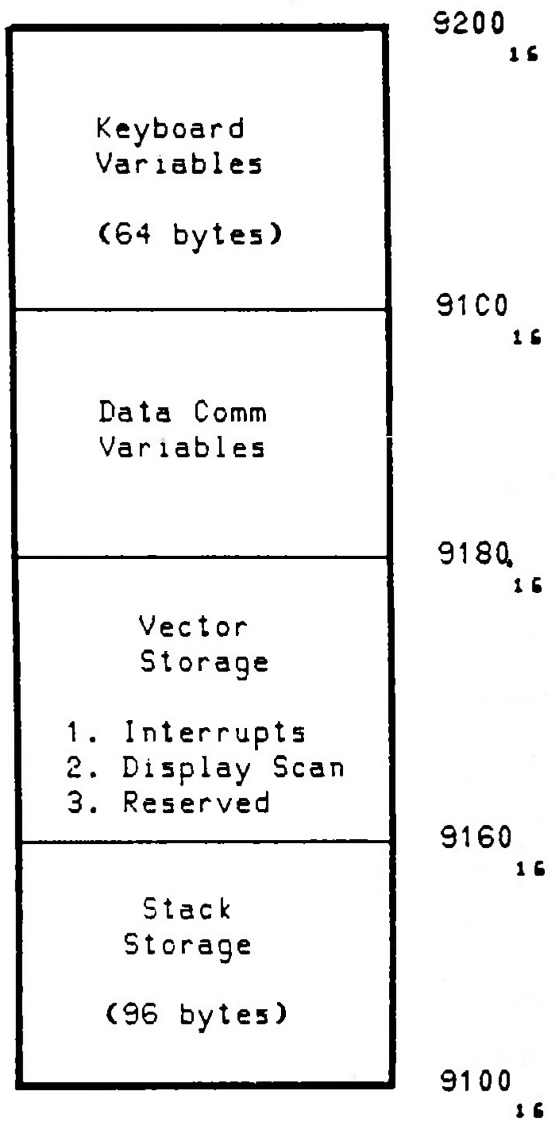


Figure 1-7. Fast RAM Memory Allocation Map

ENTRY VECTORS

Entry into each code module is made through vectors stored in the low address portion of the module. These vectors are usually "jumps" (JMP) to routines within the module.

Section II. CONFIGURATION AND TURN-ON

2649A MAINFRAME TERMINAL

The recommended order of the various boards within a 2649A is shown in figure 2-1.

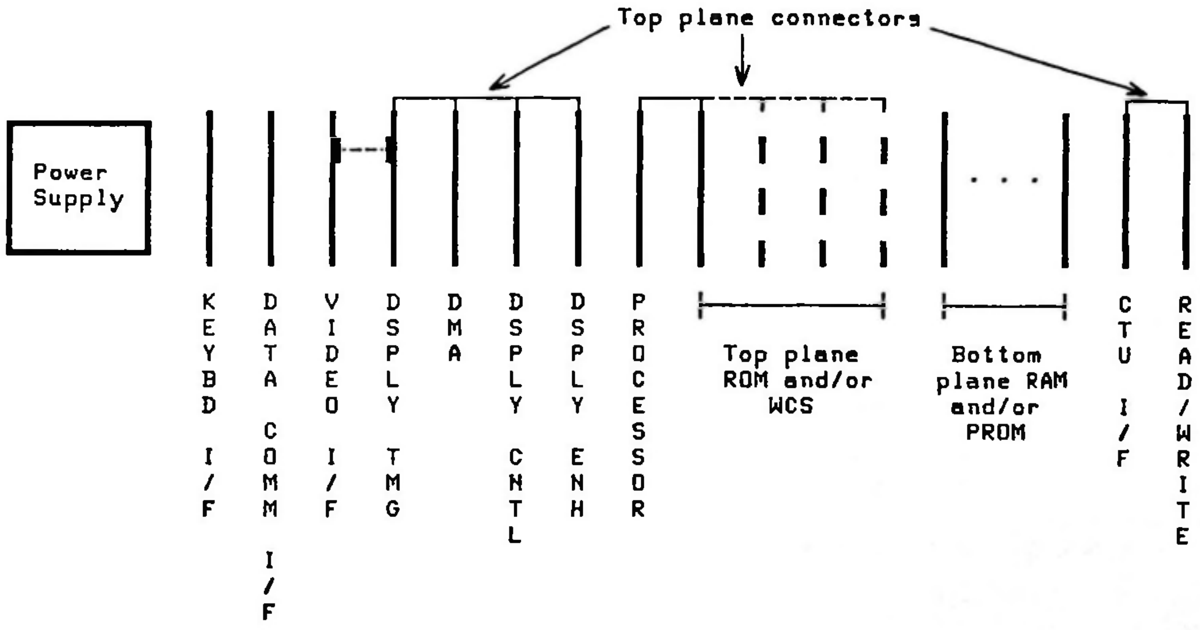


Figure 2-1. Recommended 2649A Board Order

The display timing, display control, DMA, and display enhancements boards must be in contiguous slots because they are connected to one another by a prefabricated top plane connector. The video interface board, if present, must be next to the display timing board because the two boards are joined to one another by a short jumper cable. All top plane memory boards (whether ROM or WCS) must be installed in contiguous slots immediately to the right of the Processor board because they are connected to the Processor board by a prefabricated top plane connector. The two CTU boards must also be in contiguous slots because they, too, are joined to one another by a top plane connector. They are most often installed in the farthest two slots from the power supply.

13290A DEVELOPMENT TERMINAL

To turn on the 13290A, first make sure that the various boards and top plane connectors are properly installed and that all pertinent cable hoods (such as those for the keyboard or a data comm cable) are connected to the proper boards. Then plug the power cord into both the back of the power supply and an appropriate electrical outlet and set the power rocker switch on the back of the power supply to the "ON" position.

At this point the terminal is on, but for all practical purposes it is "dead". No "TERMINAL READY" or similar message appears on the screen and none of the alphanumeric, editing, mode control, or cursor control keys on the keyboard has any effect.

Terminal Self-Test

The single ROM chip on the 13293 diagnostic board contains a binary object program loader, an ASCII object program loader, and a terminal self-test program. When you press the TEST key this ROM-resident self-test program begins displaying full screens of characters (24 rows, 80 characters per row). Observe the screen for about 20 to 25 screensful of data and make sure that the characters seem recognizable and that the cursor is progresssing systematically up the left side of the display. When you are satisfied that the terminal is working properly, press the RESET TERMINAL key. The display is cleared and the terminal is back in its "dead" state.

Loading Binary Object Code From Cartridge Tape

The binary loader residing on the 13293 diagnostic board is used to enter programs into the HP 13290A terminal after being powered up. This loader is accesed by pressing the RESET TERMINAL key and then the B key. Data on the left cartridge tape is then loaded into the terminal according to the format shown in figure 2-2.

Record #1:

0-80 bytes of ASCII characters (label)

Record #2 to n:

2 bytes = 377_8 = FF_{16}

2 bytes = Starting Address MSB first

128 bytes of binary data

1 byte = 0

1 byte = checksum of preceding 133 bytes

Record #1

Label (0-80 ASCII characters)

Remaining Records

FF	MSB LSB	128 bytes of data	0	check

Figure 2-2. Binary Loader Format

Loading Binary Object Code From Data Comm

The ROM chip on the 13293 diagnostic board also contains an ASCII loader. This ASCII loader can be used to load programs and data into any RAM location in the 13290A terminal over a data comm line from a remote computer. If you attempt to load into a ROM location or a non-existent memory block the data will simply be ignored.

The ASCII loader is accessed by an escape sequence. The program and/or data follows the escape sequence. The format of the data to be loaded is shown in figure 2-3. The ASCII loader can be accessed through the terminal keyboard, data comm interface, or from one of the cartridge tape units. Figure 2-4 contains an example of a program loaded using the ASCII loader.

␛ & b — Accesses the loader and displays LOADER on screen. (␛ & c performs the same function without the message.)

a — Causes the preceding octal address to be placed in the address register.

d — Causes the preceding three octal digits to be loaded into the memory location stored in the address register. The address register is then incremented.

c — Causes a checksum to be computed. All characters entered following the loader escape sequence (␛ & b) are included in the checksum. This checksum is compared to the octal number preceding the "c". If the checksum is correct, a bit will be set in the terminal status. If not, the loader aborts and the terminal is reset.

e — Transfer control to the address contained in the address register.

The loader escape sequence is terminated when an upper case a, c, d, or e is received. The characters `<CR>`, `<LF>`, `<DC3>`, and `<SPACE>` are ignored. If any character other than the above is received, the loader will abort and the terminal will be reset.

Remember that your program is being loaded using an escape sequence. The escape processor sets various flags that may cause problems if your program calls any of the 2645 maincode routines. It is therefore strongly recommended that one of the first things your program does it to terminate the escape processor by calling the routine ESCEND.

Figure 2-3. ASCII Loader Format

```
  LOC     OBJECT CODE                            SOURCE STATEMENTS

177700        .   .   .   CURROW   EQU    177700Q    CURRENT ROW POSITION OF CURSOR
103440        .   .   .   IOCCRW   EQU    103440Q    CURSOR ROW ADDRESS
044551        .   .   .   GTKEY    EQU    044551Q    SCAN KEYBOARD ROUTINE
013701        .   .   .   XPUTDC   EQU    013701Q    TRANSMIT CHARACTER ROUTINE
000015        .   .   .   CR       EQU    015Q       CARRIAGE RETURN CODE
002225        .   .   .   ESCEND   EQU    002225Q    END OF ESCAPE SEQUENCE ROUTINE
000000        .   .   .            ORG    177000Q
177000    072 300 377              LDA    CURROW
177003    062 040 207              STA    IOCCRW     TURN ON DISPLAY
000000        .   .   .   LOOP     EQU    *
177006    315 151 111              CALL   GTKEY      SCAN KEYBOARD
177011    302 000 000              JNZ    LOOP       CONTINUE SCAN IF NO KEY HIT
177014    365     .   .            PUSH   PSW        SAVE CHARACTER
177015    315 301 027              CALL   XPUTDC     XMIT CHARACTER
177020    361     .   .            POP    PSW        RETRIEVE CHARACTER
177021    376 015     .            CPI    CR         IS CHARACTER A CR?
177023    302 000 000              JNZ    LOOP       CONTINUE SCANNING IF NOT CR
177026    315 330 121              CALL   050730Q    FLUSH DATACOMM BUFFER
177031    303 225 004              JMP    ESCEND     EXIT TO WAIT LOOP
177034        .   .   .            END
```

```
Escape sequence:
   <ESC>&b177000a
   072d300d377d062d040d207d315d151d111d302d000d000d
   365d315d301d027d361d376d015d302d000d000d315d330d
   121d303d225d004d177000aE
```

Figure 2-4. Example of Program Formatted for the ASCII Loader

Section III. FIRMWARE DEVELOPMENT

This section describes techniques for developing firmware for the terminal. The firmware can be any combination of the following:

- Existing code modules
- Modifications to existing modules
- New modules

EXISTING CODE MODULES

In order to use the existing code modules simply select the desired code from table 3-1. This code is available in ROM or in the form of source code on a 9-track NRZ compatible magnetic tape as part of the 13256A Firmware Support Package. The ROM chips mount in the specified sockets on the 02640-60136 Control Memory board. The part number of each ROM is imprinted on the top of the chip.

Table 3-1. 2645A Maincode Firmware Code Modules (ROM)

Function	Board Number	Socket Number	Part Number
2645A Maincode	02640-60136	U17	1818-0203
2645A Maincode	02640-60136	U37	1818-0205
2645A Maincode	02640-60136	U47	1818-0206
2645A Maincode	02640-60136	U18	1818-0207
2645A Maincode	02640-60136	U27	1818-0287

MODIFICATIONS TO EXISTING CODE MODULES

Modifications to the existing code modules can be made by first studying the firmware descriptions and the code listings and then making the necessary code changes. The HP 2645 source code can then be assembled together with the changes to obtain the new code. If any of the original modules are unchanged they can be purchased in ROM from Hewlett-Packard. New blocks of code can be used in a variety of forms. Table 1-5 in Section I lists some of the considerations in deciding the form in which to store the new code.

NEW MODULES

New code modules are generated in the same manner as changed modules. In this case you are only interested in the interfacing between the new module and any standard modules. Table 1-5 can then be used to select the proper form of code storage for your application.

Firmware Development

One possible approach to developing your 13290 code is to use a microcomputer development system such as the Intellec* Microcomputer Development System. Debugging user-generated object code requires the use of commercially available tools such as the Intel* ICE-80. This in-circuit-emulator replaces the 13290's microprocessor chip and allows you to set break points, examine and modify RAM, and single-step program execution. A typical microcomputer development system configuration is illustrated in figure 3-1.

NOTE

This manual in no way recommends the Intel* system over any other. It is used here only as an example of a typical development system.

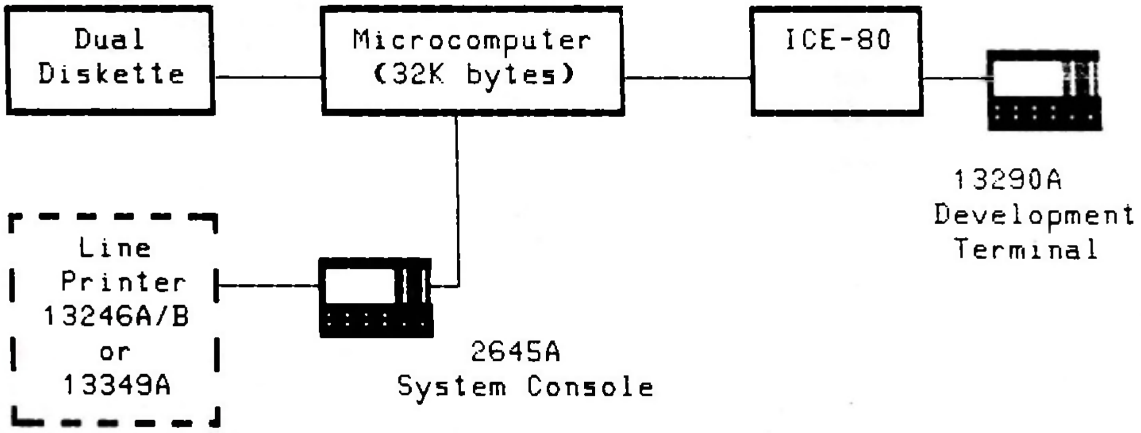

Dual Diskette
Microcomputer (32K bytes)
ICE-80
13290A Development Terminal
Line Printer 13246A/B or 13349A
2645A System Console

Figure 3-1. Microcomputer Development System

*Registered trademark, Intel Corporation

Another approach is to use a cross assembler on a conventional computer system. The HP 13256A Firmware Support Package includes a cross assembler that is compatible with both the RTE operating system and File Management Package of the HP 1000 Computer System. With this capability you can prepare, edit, and assemble 13290A source code on the HP 1000. Your terminal must be either a 2645A or a 13290A (running 2645A object code) because the object code produced by the cross assembler is output to cartridge tape.

Modification of this cross assembler to run on different systems would primarily involve the EXEC calls used for controlling input and output.

For information on how to use the cross assembler, refer to the documentation provided with the 13256A Firmware Support Package.

Section IV. HARDWARE DEVELOPMENT

The terminal is made up of hardware modules. These modules are listed in table 4-1. Normally only memory modules, I/O interfaces or special interface cables will need to be designed and fabricated. It is assumed that such major terminal hardware as the power supply, backplane, case, and display and control circuitry will seldom require modification.

MEMORY MODULE DESIGN

If it is necessary to design a new type of memory module, refer to the Technical Information Package for a discussion of bottom and top plane bus protocols and signal specifications. Using an existing similar module as a model is often helpful.

I/O INTERFACE DESIGN

When designing an I/O interface or special purpose PCA, refer to the Technical Information Package for a discussion of bottom plane bus protocol and signal specifications. Using an existing similar module as a model is often helpful.

CABLE DESIGN

If it is necessary to design or build a cable refer to the Technical Information Package for details on signal levels, propogation times, and material specifications.

Additional information on data communication cables is contained in the 2645A terminal reference manual.

Table 4-1. 13290A/2649A Hardware Modules

Module Number	Module Nomenclature	13290A	2649A
13255-91001	Backplane	S	S
13255-91142	Power Supply	S	S
13255-91018	Keyboard	S	S
13255-91093	Processor (8080A-2)	S	S
13255-91095	Sweep	S	S
13255-91112	Display Controller	S	S
13255-91124	Extended DMA	S	S
13255-91007	4K UV PROM	A	—
13255-91024	Display Expansion	A	A
13255-91031	Term Duplex Register	A	A
13255-91032	Cartridge Tape Unit	O	O
13255-91053	PROM Character	A	A
13255-91063	Display Test	K	K
13255-91064	+2K Memory	A	A
13255-91065	+4K Memory	SA	A
13255-91069	Simplified Keyboard	—	O
13255-91082	CTU Test	K	K
13255-91086	Asynch Data Comm	S	A
13255-91089	GP Asynch Data Comm	A	A
13255-91106	Asynch Multipoint	A	A
13255-91107	Synch Multipoint	A	A
13255-91119	Comp Video I/F	A	A
13255-91123	Extended Kybd I/F	S	O
13255-91136	Control Memory (AMD)	S	O
13255-91137	Extended CTU I/F	O	O
13255-91143	GP Asynch Data Comm	A	A

LEGEND

A=Accessory O=Option S=Standard K=Service Kit

Section V. SUPPORT ACCESSORIES

This section briefly describes the characteristics and use of the 13291, 13292, 13293, and 13295 support accessories.

13291 4K PROM BOARD

The 13291 is a 4K PROM board that contains two separately addressable 2K modules. The two modules can be configured as consecutive 2K blocks of memory or they can be configured with widely-separated starting addresses. The 13291 accommodates up to sixteen Intel 1702A UV-erasable PROMs (up to eight per 2K module) and the content of the PROMs can be accessed only by way of the bottom plane bus. Either 2K module can be enabled or disabled independently of the other.

You could use the 13291, for example, to replace a couple of the main code ROM chips with your own PROM-resident code (without disturbing the other main code ROM chips). To do this, you remove jumpers on the 02640-60136 Control Memory board to disable the particular ROM chips. Then you configure the two modules of the 13291 board to the appropriate starting addresses (such as 12K and 18K, respectively) and install your PROM chips at the start of each module. Thereafter, whenever a location between 12K and 14K is addressed the code in your first PROM block is executed instead of the corresponding ROM-resident code. The same is true for locations 18K to 20K and your second PROM.

NOTE

The 02640-60136 Control Memory PCA has two banks of jumper pins in the upper left corner of the component side (near the PCA label). The pin positions are labeled 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, +24, and RAM DISAB. When a jumper is installed in one of the slots labeled 0 through 22, the ROM for the specified 2K address range is enabled. To disable the 14-16K ROM chip, for example, remove the jumper from the slot labeled 14.

When the +24 jumper is installed, the ROM addresses are as labeled and lie within the range 0-24K. If the +24 jumper is removed, the specified ROM addresses are effectively incremented by 24K and lie within the range 24-48K. The RAM DISAB jumper is used for enabling or disabling the 256-byte fast RAM on the Control Memory PCA. When the jumper is installed, the fast RAM is enabled; when it is removed, the fast RAM is disabled.

The 13291 PCA (02640-60007) also contains two banks of jumper pins. The pin positions on each bank are labeled 2K, 4K, 8K, 16K, 32K, and DISABLE. When jumpers are installed in all the numbered slots, the module's starting address is 0. When a particular jumper is removed, add the labeled value to the starting address of the particular PROM module. For example, if you remove the jumpers from the slots labeled 4K and 8K, the starting address of the module is 12K. When the jumper is installed in the DISABLE slot, the particular PROM module is disabled; when it is removed, the module is enabled.

13292 8K WCS BOARD

The 13292 is a high-speed (400 nanosecond) 8K WCS board that can be accessed only by way of the top plane bus. If you specify option -001, the 13292 also includes a five-wide top plane connector. Because it is accessed by way of the top plane bus, the content of the WCS chips is fetched and executed at the same speed as the standard ROM-based control code of the other 2640-series terminals.

You would typically use three 13292 boards (connected to the Processor board and a 13293 Diagnostic board by a five-wide top plane connector) to house the first 24K of your terminal's main code.

NOTE

The 13292 8K WCS board also has a bank of jumper pins. The pin positions on the bank are labeled 2K, 4K, 8K, 16K, 32K and DISABLE. When jumpers are installed in all the numbered slots, the board's starting address is 0. When a particular jumper is removed, add the labeled value to the board's starting address. For example, if you remove the jumpers from the slots labeled 8K and 16K, the starting address of the board is 24K. When the jumper is installed in the DISABLE slot, the entire WCS board is disabled; when it is removed, the board is enabled.

13293 DIAGNOSTIC BOARD

The 13293 is a control memory board that contains a single ROM chip: the binary loader. As with the standard control memory board, it must be connected to the processor board by way of the top plane bus. It is referred to as a "diagnostic" board because it is meant to be used for loading standard firmware into 2649 terminals in the field so as to create a known firmware environment within which terminal malfunctions can more quickly and easily be diagnosed.

13295 KEYCAP KIT

The 13295 is a kit containing approximately 50 key caps (with clear plastic covers and blank key cap inserts) and a blank keyboard overlay. The keyboard overlay is painted and properly punched but contains no lettering. With the 13295, users can relabel individual keys and silkscreen their own lettering on the keyboard overlay to create prototype keyboards for specialized applications.

Section VI. MAIN CODE MODULE

INITIALIZATION

When the terminal's power is first turned on or when the TERMINAL RESET key is pressed, the processor performs either a hard or soft reset.

To determine which type of reset to perform, the processor examines the contents of location FFCD (177715B). If the location contains a JMP instruction to the "soft reset" code, the processor executes that instruction. If the location contains anything else, the processor passes control to the "hard reset" code.

One of the first things the terminal initialization code does when you turn on the power is to clear the contents of location FFCD. Thus, whenever you turn on the terminal's power the processor performs a hard reset.

One of the last things the "hard reset" code does is to store the proper JMP instruction in location FFCD, thus making it possible to subsequently perform a soft reset.

Pressing the TERMINAL RESET key causes a soft reset because location FFCD contains the proper JMP instruction. Pressing the TERMINAL RESET key twice in quick succession (i.e., within a half second) automatically clears location FFCD, and thus causes a hard reset.

HARD RESET

A hard reset consists of the following operations:

a. Clear the common variable area in display memory.

b. Determine the starting address of display memory (52K, 56K, or 60K). The firmware does this by first writing a bit pattern to location 63K and then reading the contents of that location. If the proper bit pattern is read back, the processor knows that there is RAM memory located at 63K-64K. It then does the same for locations 62K, 61K, 60K, and so forth, down to 52K.

c. Determine whether or not there is RAM memory installed for locations 48K-52K. This is done in the same manner as described in step b, above.

d. Pass control to the initialization routines for the keyboard, data comm interface (if present), and printer (if present).

e. Determine if alternate I/O code is present. If it is, pass control to the alternate I/O initialization routine.

f. Generate the free blocks list for display memory.

g. Initialize the soft key definitions.

h. Reset the keyboard, data comm interface (if present), and CTUs (if present).

i. Enable all interrupts.

j. Display the message "TERMINAL READY" in the upper left corner of the screen.

k. Store the proper JMP instruction in location FFCD.

l. Go to the wait loop.

SOFT RESET

A soft reset consists of the following operations:

a. Reset the keyboard.

b. Reset the data comm interface (if present).

c. Reset the CTUs (if present). Tape motion, if any, is stopped and the cartridges are rewound to their load points.

d. Restore the user's normal display.

e. Enable all interrupts.

f. Go to the wait loop.

THE WAIT LOOP

Whenever the terminal is not actually responding to a keystroke or an interrupt it executes what is referred to as the wait loop. Essentially this wait loop systematically checks the keyboard to determine if a key has been pressed and examines the data comm interface to see if one or more characters have been received from a remote computer or device. It also monitors the cartridge tape units (if present) to determine whether a cartridge has been inserted or removed.

The functions performed during the wait loop are detailed in figure 6-1 (these same flow charts also appear in the 13255A Technical Information Package).

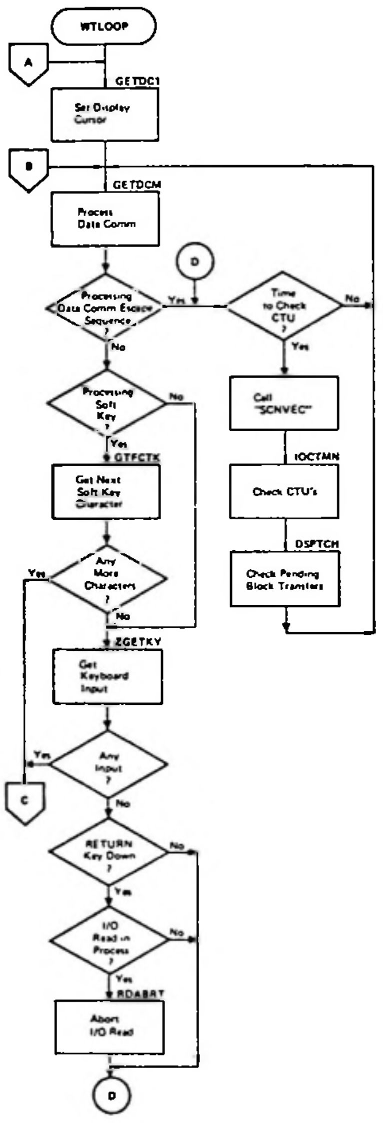


Figure 6-1. Wait Loop Flow Chart

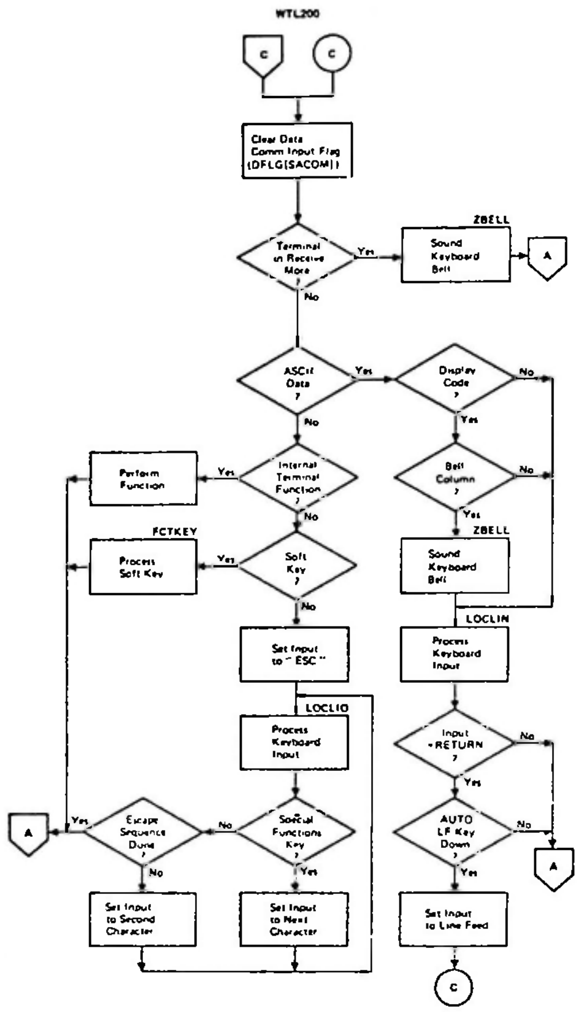


Figure 6-1. Wait Loop Flow Chart (Continued)

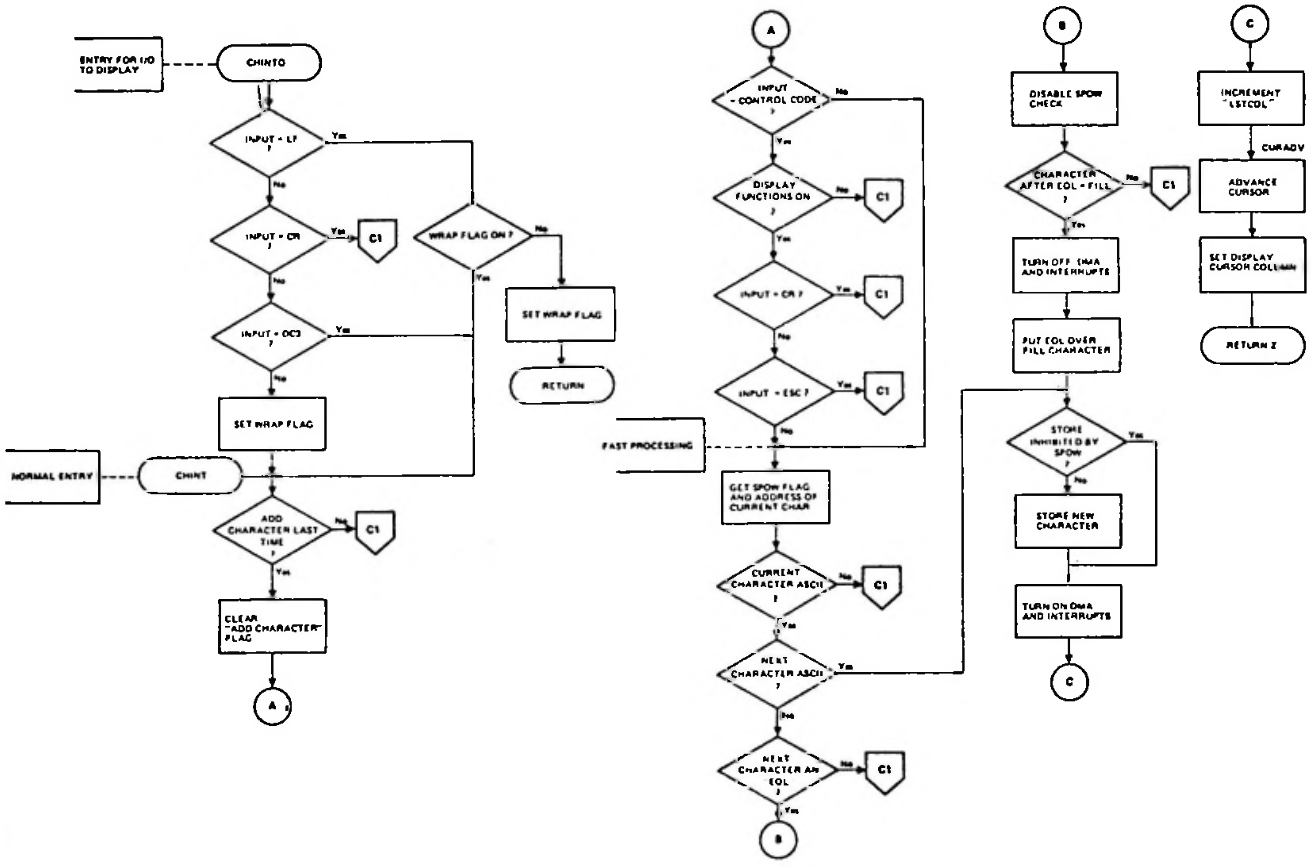


Figure 6-1. Wait Loop Flow Chart (Continued)

DISPLAY MEMORY ORGANIZATION

Display memory data is stored in 16-byte blocks. Each line of display data is made up of one or more of these blocks. The blocks make up a doubly linked list. Each block contains a pointer to the next sequential block. The first block in a display line contains pointers to the next and previous lines. The last block in a line contains a pointer to the first block of that line. Figures 6-2 through 6-5 illustrate the blocks used in the linked list.

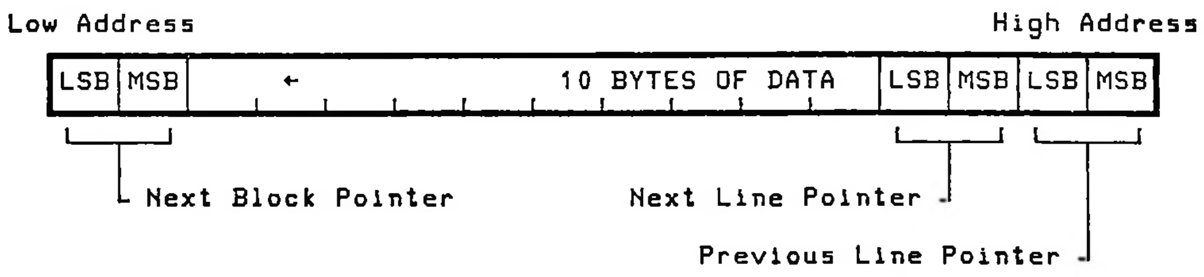


Figure 6-2. First Block in a Line

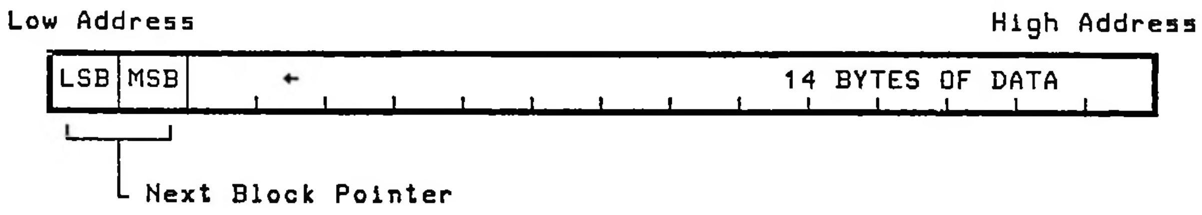
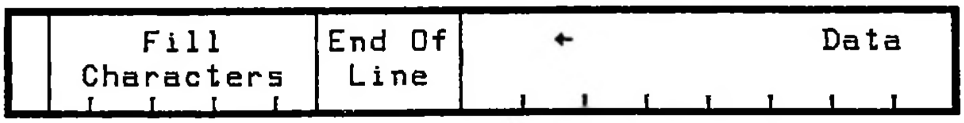


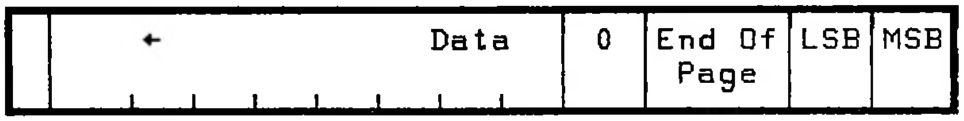
Figure 6-3. Remaining Blocks



End Of Line = 314_8 = CC_{16}

Fill = 303_8 = $C3_{16}$

Figure 6-4. Last Block in a Line



End Of Page = 316_8 = CE_{16}

Figure 6-5. First Block in Last Line of Display List

When the terminal is turned on or a full reset is performed, the display memory is initialized and all blocks are returned to a "free list". As display data is entered into memory new blocks are assigned and formatted as required. This continues until all of the available free blocks are used. When this point is reached and additional data is entered, one of three actions is performed:

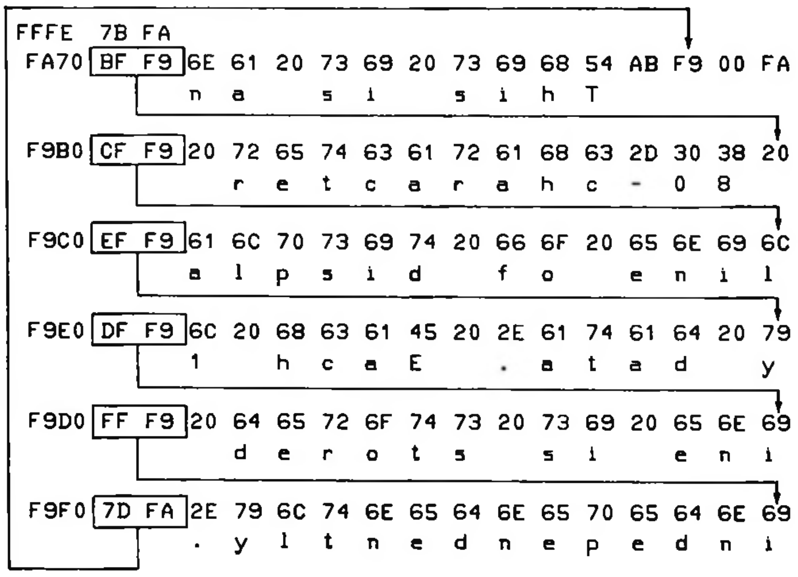
1. The first blocks in display memory are reassigned as new blocks. This causes the data at the beginning of the display list to be lost.

2. In Edit or Data Logging Mode the first blocks of display data are transferred to the "TO" device(s) before the blocks are reassigned.

3. If the terminal is in Memory Lock Mode the keyboard will be locked and the newly entered data will be lost.

Display memory is scanned by the display hardware in order of decreasing addresses. This is the reason that the display data is stored in reverse order. The display hardware is capable of using the points to display each line of data on the screen.

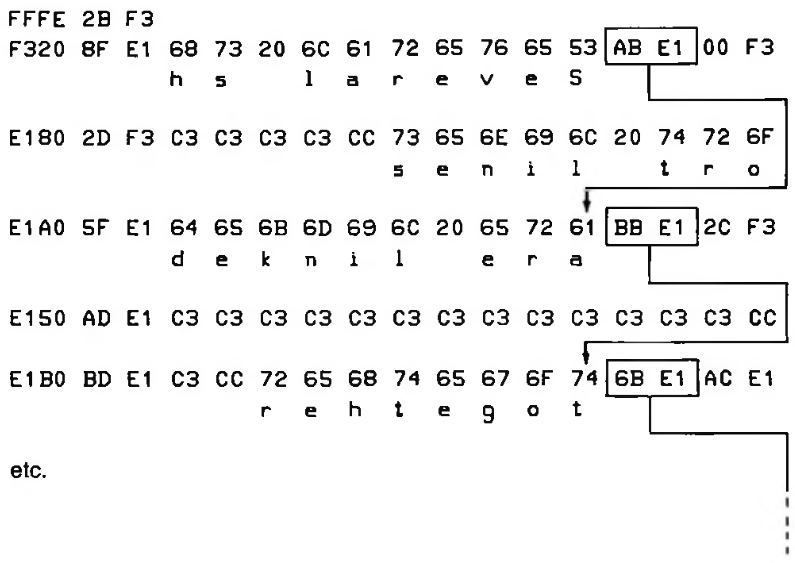
This is an 80-character line of display data. Each line is stored independently.



```
FFFE  7B FA
FA70 BF F9 6E 61 20 73 69 20 73 69 68 54 AB F9 00 FA
           n  a     s  i     s  i  h  T
F9B0 CF F9 20 72 65 74 63 61 72 61 68 63 2D 30 38 20
              r  e  t  c  a  r  a  h  c  -  0  8
F9C0 EF F9 61 6C 70 73 69 74 20 66 6F 20 65 6E 69 6C
           a  l  p  s  i  d     f  o     e  n  i  l
F9E0 DF F9 6C 20 68 63 61 45 20 2E 61 74 61 64 20 79
           l     h  c  a  E     .  a  t  a  d     y
F9D0 FF F9 20 64 65 72 6F 74 73 20 73 69 20 65 6E 69
              d  e  r  o  t  s     s  i     e  n  i
F9F0 7D FA 2E 79 6C 74 6E 65 64 6E 65 70 65 64 6E 69
           .  y  l  t  n  e  d  n  e  p  e  d  n  i
```

Figure 6-6. Memory Linkages for an 80 Character Line

Several short lines are linked together in this manner



```
FFFE 2B F3
F320 8F E1 68 73 20 6C 61 72 65 76 65 53 AB E1 00 F3
           h  s     l  a  r  e  v  e  S
E180 2D F3 C3 C3 C3 C3 CC 73 65 6E 69 6C 20 74 72 6F
                          s  e  n  i  l     t  r  o
E1A0 5F E1 64 65 6B 6D 69 6C 20 65 72 61 BB E1 2C F3
           d  e  k  n  i  l     e  r  a
E150 AD E1 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 CC
E1B0 BD E1 C3 CC 72 65 68 74 65 67 6F 74 6B E1 AC E1
                 r  e  h  t  e  g  o  t
etc.
```

Figure 6-7. Interline Data Links

Note that display data characters are limited to values 0-177. This means that the high order bit (bit 6) of a display character is always 0. Included as data are all the ASCII characters and display enhancement flags. Table 6-1 contains a list of display enhancement flags. Bytes with values 300-317 are interpreted as software flags. A list of software flags is given in table 6-2. Bytes with values 320-377 are interpreted as the MSB (most significant byte) of a block pointer. The byte following a MSB is interpreted as the least significant byte of the block pointer.

Table 6-1. Display Enhancement Flags

$$\begin{array}{lcccccccc} \text{Bit} & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ & 1 & 0 & \multicolumn{2}{c}{\underbrace{c\quad c}_{\text{character set}}} & \multicolumn{4}{c}{\underbrace{e\quad e\quad e\quad e}_{\text{enhancement}}} \end{array}$$

Character Set (cc) Codes

00	= $^{E}_{C}$) @	= Base set
01	= $^{E}_{C}$) A	= Alternate set #1
10	= $^{E}_{C}$) B	= Alternate set #2
11	= $^{E}_{C}$) C	= Alternate set #3

Enhancement (eeee) Codes

0	– End Enhancement (@)
1	– Blinking (A)
2	– Inverse Video (B)
3	– Blinking, Inverse Video (C)
4	– Underline (D)
5	– Underline, Blinking (E)
6	– Underline, Inverse Video (F)
7	– Underline, Inverse Video, Blinking (G)
8	– Half-Bright (H)
9	– Half-Bright, Blinking (I)
A	– Half-Bright, Inverse Video (J)
B	– Half-Bright, Inverse Video, Blinking (K)
C	– Half-Bright, Underline (L)
D	– Half-Bright, Underline, Blinking (M)
E	– Half-Bright, Underline, Inverse Video (N)
F	– Half-Bright, Underline, Inverse Video, Blinking (O)

Examples:

81	= Base set, blinking
95	= Alternate set #1, underline, blinking
AA	= Alternate set #2, half-bright, inverse video
BD	= Alternate set #3, half-bright, underline, blinking

Table 6-2. Software Display Flags

Bit	7	6	5	4	3	2	1	0
	1	1	0	0	s	s	s	s

$1\ 1\ 0\ 0\ \underbrace{s\ s\ s\ s}_{\text{software display code}}$

Software Display (ssss) Codes

0	-	End of Unprotected or Transmit Only Field
1	-	Begin Unprotected Field
2	-	Begin Transmit Only Field
3	-	Fill character
4	-	Non-displaying Terminator
5	-	Alpha Field
6	-	Numeric Field
7	-	Alphanumeric Field
8	-	Soft key attribute field
9	-	(not used)
A	-	(not used)
B	-	(not used)
C	-	End of Line
D	-	(not used)
E	-	End of Page
F	-	(not used)

Examples:

C1	=	Begin unprotected field
C3	=	Fill character
CC	=	End of line

NOTE

The hex codes D0 through FF are interpreted as the most significant byte (MSB) of a display memory pointer. The next higher byte (next byte to the left) is interpreted as the least signficant byte (LSB) of the display memory pointer.

The End Of Line (EOL) code is used to indicate that the last display data in a line has been reached. The EOL code (CC) is followed with fill characters (C3) to fill up any unused bytes in the block.

The End Of Page (EOP) code is used to mark the end of the display list. Note that this is not necessarily the end of the current screen. The EOP code (CE) is stored in the MSB part of the Next Line pointer of the last line of the display list.

Adding A Character To A Line

A new line is started with a single display block. The next block pointer points to the MSB of the next line pointer. Next and previous line pointers are set as required. The first data byte is an EOL (CC) followed by nine fill characters (C3). See figure 6-8.

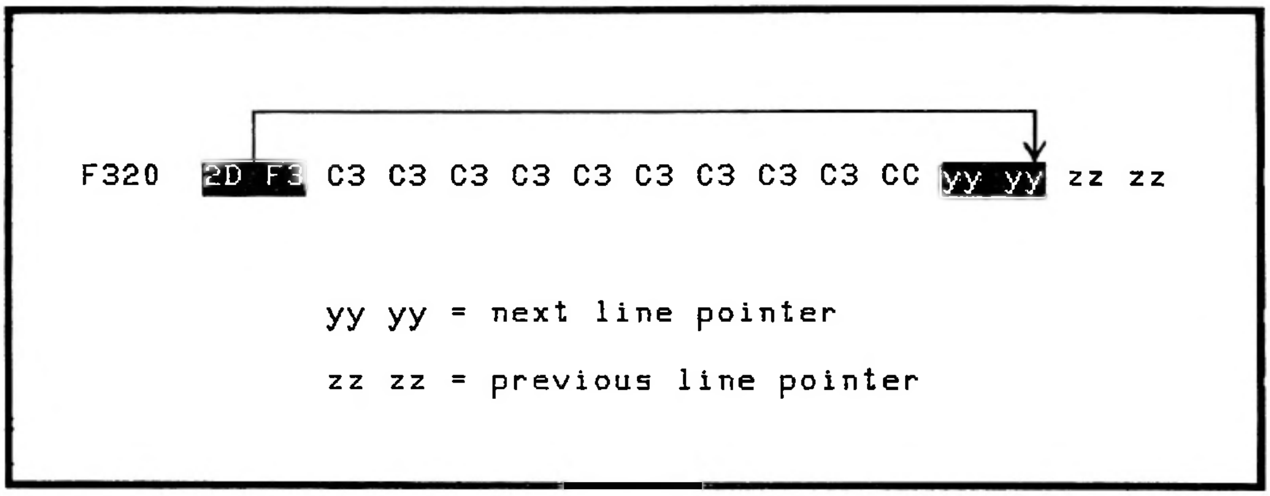

F320 2D F3 C3 C3 C3 C3 C3 C3 C3 C3 C3 CC yy yy zz zz
yy yy = next line pointer
zz zz = previous line pointer

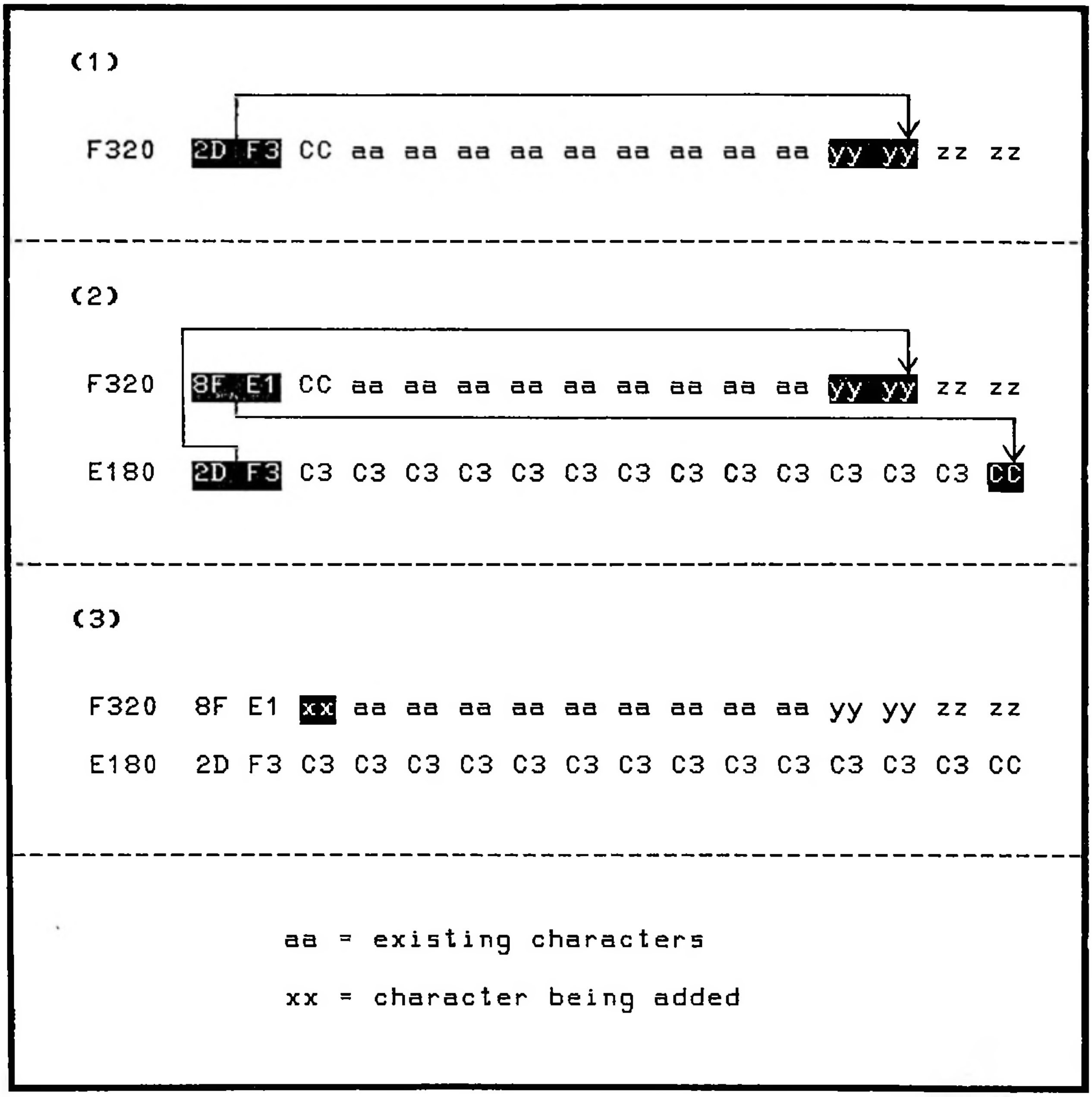
Figure 6-8. Initializing A New Line

When a character is added to the block an EOL (CC) is first written over the first fill character. This prevents the possibility of displaying a line with no end of line marker. The new character then replaces the old EOL byte. See figure 6-9.

```
(1)

  F320   2D F3 C3 C3 C3 CC aa aa aa aa aa aa yy yy zz zz

--------------------------------------------------------------

(2)

  F320   2D F3 C3 C3 CC CC aa aa aa aa aa aa yy yy zz zz

--------------------------------------------------------------

(3)

  F320   2D F3 C3 C3 CC xx aa aa aa aa aa aa yy yy zz zz

--------------------------------------------------------------

               aa = existing characters

               xx = character being added
```

Figure 6-9 Adding A New Character To A Block

When the current block is full, a new block is allocated from the free block list. The new block is loaded with an EOL (CC hex) and fill characters (C3 hex) and the next block pointer is copied from the previous block. The next block pointer of the previous block is updated to point to the beginning of the new block. The new character is then written over the EOL character in the previous block. See figure 6-10.



(1)
F320 2D F3 CC aa aa aa aa aa aa aa aa aa yy yy zz zz
(2)
F320 8F E1 CC aa aa aa aa aa aa aa aa aa yy yy zz zz
E180 2D F3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 CC
(3)
F320 8F E1 xx aa aa aa aa aa aa aa aa aa yy yy zz zz
E180 2D F3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 CC
aa = existing characters
xx = character being added

Figure 6-10. Adding a New Block

When 80 displayable characters have been entered in a line and the last character takes up the last byte of the display block, no EOL byte is required. The terminal assumes an EOL and begins a new line.

In addition to the display characters, software flags are also stored as display data. These flags define special fields and alternate character sets. A maximum of about 100 software flags can be stored in a line. These are in addition to the 80 displayable characters. This is the maximum that can be processed by the display hardware during the refresh cycle.

Display Memory Links

The display hardware detects the end of a display block by reading a byte in the range D0-FF. When such a byte is read it is interpreted as the MSB of a two byte pointer to the next display block. The standard terminal uses a 16 byte block format. If you have an application where the display data can be more efficiently stored using a different block size — or no blocks at all, you can do this by changing the positions of the link pointers.

Rolling the Display Up Or Down

Rolling up or down is done by changing the display start pointer. To roll up, the next line pointer of the current top display line is copied into the display start pointer (FFFE). To roll down, the previous line pointer minus one in the current top display line is copied into the display start pointer.

Insert/Delete Line

Inserting a line is done by getting a block from the free list and formatting the block into a new line. The next line pointer for the new line is copied from the previous line and the previous line pointer for the new line is copied from the next line. The pointers in the lines preceding and following the new line are set to point to the new line.

Deleting a line is done by copying the previous line pointer of the line to be deleted into the following line and the next line pointer of the line to be deleted into the preceding line. The blocks of the deleted line are then added to the free block list.

Swapping Display Lines

Figure 6-11 contains an example of the code required to perform a line swap. Note the order in which the block pointers are changed. A variety of similar display manipulations can be performed in this manner.

```
XLINE  EQU  *
       CALL RCADRA       SET DISPLAY CONTEXT TO CURRENT LINE
       LHLD LSTLIN       GET ADDRESS OF CURRENT LINE
       MOV  E,M          PUT ADDRESS OF NEXT LINE IN REGISTERS
       INX  H              D AND E
       MOV  D,M
       INX  H            PUT ADDRESS OF PREVIOUS LINE IN REGISTER
       MOV  C,M            B AND C
       INX  H
       MOV  B,M
*
*   B,C = PREVIOUS LINE POINTER VALUE OF CURRENT LINE
*   D,E = NEXT LINE POINTER VALUE OF CURRENT LINE
*   H,L = ADDRESS OF MSB PART OF PREVIOUS LINE POINTER FOR CURRENT LINE
*
       XCHG
       DCX  D            SET REGISTERS D AND E TO ADDRESS OF FIRST
       DCX  D              CHARACTER IN CURRENT LINE
       DCX  D
       DCX  D
       INX  H            H,L = ADDRESS OF NEXT LINE POINTER IN NEXT LINE
       MOV  A,M          PUT NEXT LINE POINTER VALUE OF CURRENT LINE
       MOV  M,E            INTO NEXT LINE POINTER OF NEXT LINE AND
       MOV  E,A            PUT VALUE OF NEXT LINE POINTER OF NEXT LINE
       INX  H              INTO REGISTERS D AND E
       MOV  A,M
       MOV  M,D
       MOV  D,A
       INX  H            PUT PREVIOUS LINE POINTER VALUE OF CURRENT LINE
       MOV  M,C            INTO PREVIOUS LINE POINTER OF NEXT LINE
       INX  H
       MOV  M,B
```

Figure 6-11. Line Swapping Example

```
*
*  B,C = PREVIOUS LINE POINTER VALUE OF CURRENT LINE
*  D,E = ADDRESS OF FIRST CHARACTER IN LINE FOLLOWING NEXT_LINE
*  H,L = ADDRESS OF MSB PART OF PREVIOUS LINE POINTER FOR NEXT LINE
*
       DCX   H              PUT ADDRESS OF FIRST CHARACTER IN NEXT LINE
       DCX   H                INTO NEXT LINE POINTER OF PREVIOUS LINE
       DCX   H
       DCX   H
       MOV   A,L
       STAX  B
       INX   B
       MOV   A,H
       STAX  B
       MOV   B,H
       MOV   C,L
       LHLD  LSTLIN
*
*  B,C = ADDRESS OF FIRST CHARACTER IN NEXT LINE
*  D,E = ADDRESS OF FIRST CHARACTER IN LINE FOLLOWING NEXT LINE
*  H,L = ADDRESS OF LSB PART OF NEXT LINE POINTER FOR CURRENT LINE
*
       MOV   M,E            SET NEXT LINE POINTER OF CURRENT LINE TO
       INX   H                ADDRESS OF FIRST CHARACTE IN LINE FOLLOWING
       MOV   M,D              NEXT LINE
       INX   H
       INX   B              SET PREVIOUS LINE POINTER OF CURRENT LINE
       MOV   M,C              TO ADDRESS OF LSB PART OF NEXT LINE POINTER
       INX   H                FOR NEXT LINE
       MOV   M,B
       XCHG
       DCX   D
       DCX   D
       DCX   D
       INX   H
       INX   H
       INX   H
       MOV   M,E            SET PREVIOUS LINE POINTER OF LINE FOLLOWING
       INX   H                NEXT LINE TO ADDRESS OF LSB PART OF NEXT
       MOV   M,D              LINE POINTER FOR CURRENT LINE
       MOV   L,C            SET CURRENT LINE POINTER TO ADDRESS OF LSB
       MOV   H,B              PART OF NEXT LINE POINTER FOR NEXT LINE
       SHLD  LSTLIN
       MVI   A,80           SET LAST COLUMN PROCESSED TO 80 TO FORCE
       STA   LSTCOL           LINE RE-SCAN
       RET                  RETURN
```

Figure 6-11. Line Swapping Example (Continued)

Section VII. KEYBOARD CODE MODULE

The keyboard code module controls all keyboard subsystem functions. The module detects key hits, sets indicators on the keyboard, and performs the alpha and numeric field checking operations. The keyboard code occupies memory locations 4800 to 5000 (base 16).

KEYBOARD SUBSYSTEM

As shown in figure 7-1, the keyboard subsystem consists of the following three sections:

- Keyboard
- Keyboard interface
- Keyboard code module

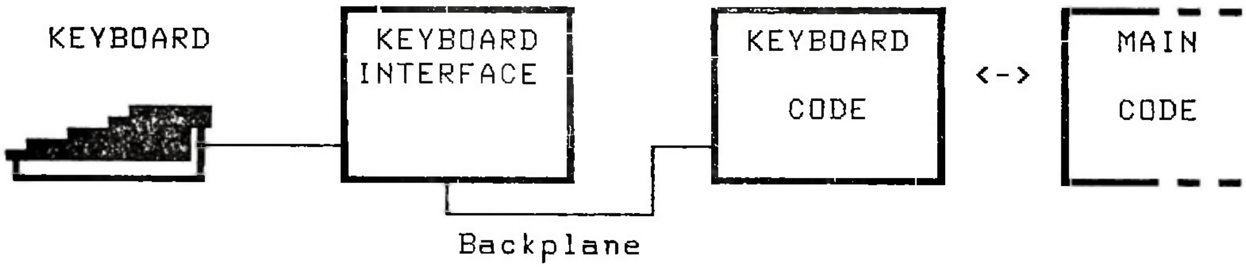
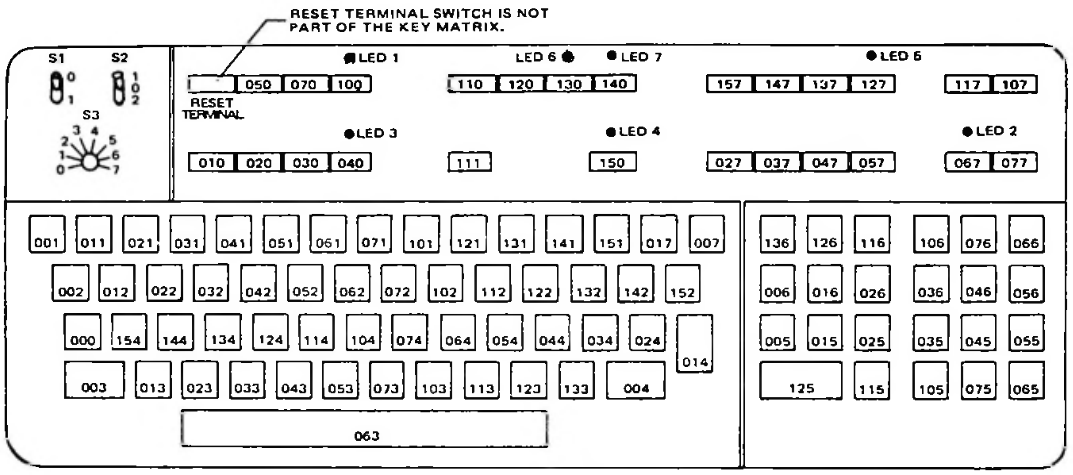


Figure 7-1. Keyboard Subsystem

Keyboard

The keyboard contains most of the operator controls (keys, data communication switches, etc.). The keyboard is available in several configurations with different key arrangements and labeling. It accepts the basic mechanical input from the terminal operator. A detailed discussion of the keyboard together with schematics and signal lists is given in the 13255A Technical Information Package.

Figure 7-2 shows the layout of the keyboard. Each key (whether present or not) is assigned to a matrix position. When a key is pressed or released, its row and column positions are sent to the keyboard interface.



NOTE: ALL SWITCHES AND LEDS ARE SHOWN AS THEY ARE LOADED ON THE PCA.

Figure 7-2. Keyboard Layout

Keyboard Interface

The keyboard interface accepts input (key and switch action) from the keyboard for use by the keyboard code. It also interprets commands from the code and controls the keyboard (sets indicators, transmits status, rings the bell, etc.).

In addition, the keyboard interface contains 24 switches that can be used by the operator to affect terminal operation. These switches can be controlled programmatically. Since most terminal operations use only a few of these switches, they can often be used by special application firmware.

Programming information for the keyboard interface is provided later in this section. Detailed information on the operation of the keyboard interface PCA is contained in the 13255A Technical Information Package.

Keyboard Firmware

The keyboard firmware performs most of its functions using subroutines. The 13255A Technical Information Package (T.I.P.) contains detailed descriptions of each of the routines and their parameters. You can access these routines to perform most keyboard tasks.

The keyboard module provides a flexible structure for creating alternate keyboard layouts or foreign language terminals. In addition, special functions can be assigned with a minimum of change to the terminal code. The keyboard module is made up of the following sections.

- Monitor (KBMON)
- Input processor (GTKEY)
- Subroutines
- Utilities

Monitor

The monitor scans the keyboard for changes in key state. For this purpose the keyboard is divided into a matrix of 14 columns of 8 keys each.

The monitor routine maintains the current state of the keyboard in a 14 byte table. Each bit represents the current state of a key (1 = pressed or set, 0 = up or clear). A 40 byte transition buffer is also used to hold up to 20 state changes. Each time the monitor routine is called, it scans two columns of the keyboard. Table 7-1 shows how key information is returned on the bus lines.

If a key transition is detected, an entry is made in the transition buffer and the current state table is updated. The formats of the current state table and the transition buffer are shown in figure 7-3.

Table 7-1. Keyboard Data Bus Bits

KEYBOARD COLUMNS											
Column				Data Bus Bit							
A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	007	006	005	004	003	002	001	000
0	0	0	1	017	016	015	014	013	012	011	010
0	0	1	0	027	026	025	024	023	022	021	020
0	0	1	1	037	036	035	034	033	032	031	030
0	1	0	0	047	046	045	044	043	042	041	040
0	1	0	1	057	056	055	054	053	052	051	050
0	1	1	0	067	066	065	064	063	062	061	060
0	1	1	1	077	076	075	074	073	072	071	070
1	0	0	0	107	106	105	104	103	102	101	100
1	0	0	1	117	116	115	114	113	112	111	110
1	0	1	0	127	126	125	124	123	122	121	120
1	0	1	1	137	136		134	133	132	131	130
1	1	0	0	147			144		142	141	140
1	1	0	1	157			154		152	151	150

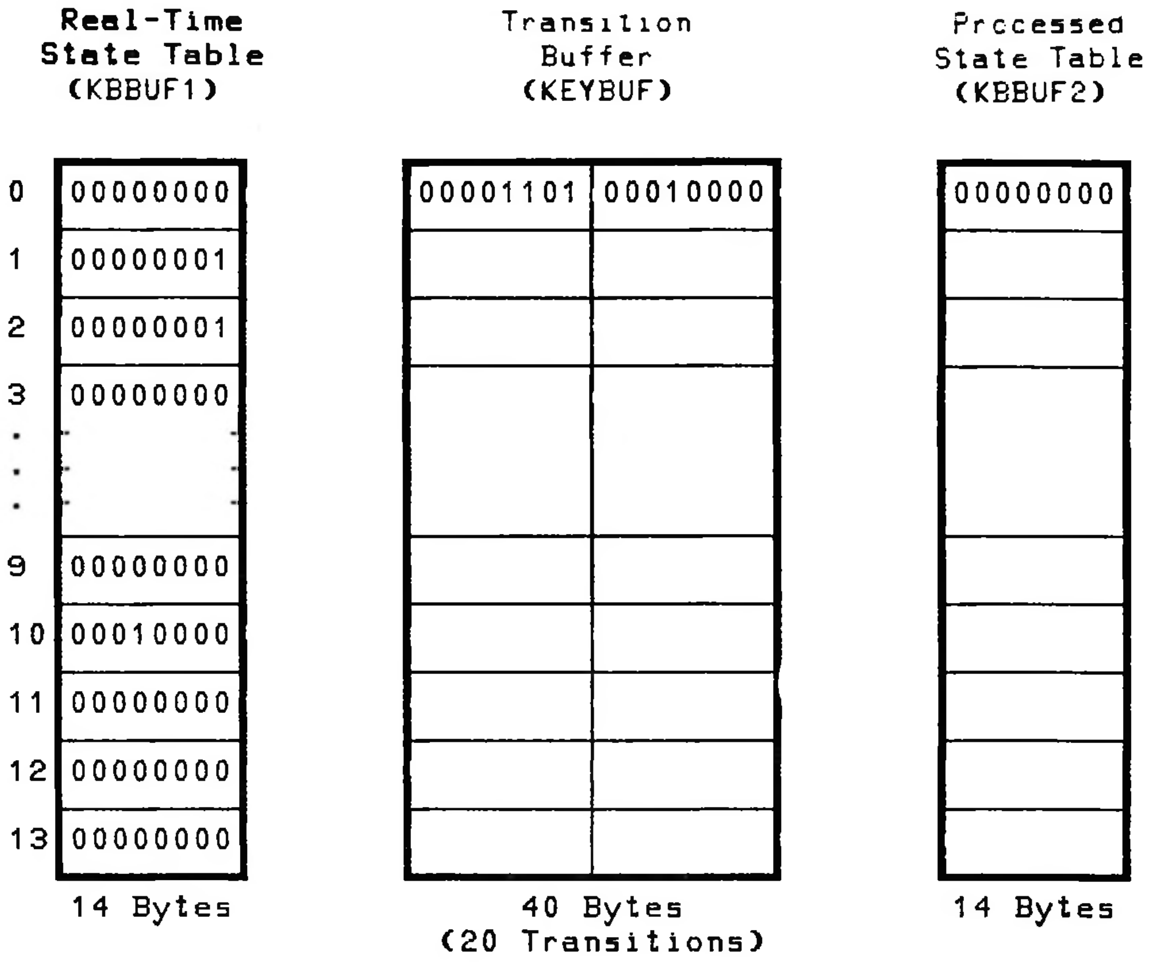


Figure 7-3. State Tables and Transition Buffer

The transition buffer is made up of 2 bytes for each transition. The first byte contains the column number in which the transition occurred and the second contains the new state of the column.

Keyboard Input Processor (GTKEY)

The keyboard input processor takes input from the transition buffer and performs the necessary action. This may be the return of a character code to the routine that called the input processor or the performance of an internal function. The keyboard input processor maintains an alternate state table containing the transitions that have been processed.

The row and column position of a pressed key is used to select an entry from one of two tables. The table selected depends on the state of the SHIFT keys. Each table entry consists of a single byte. If the high order bit of the entry is a 0, then the entry is an ASCII character which is simply returned to the calling routine. Entries with higher values indicate that more complex actions are to be performed.

Translation Tables (Unshift/Shift)

000-177	ASCII character to be returned.
200-207	Latching key functions (MEMORY LOCK, etc) returns no character or a character in the range 260-377.
210-227	Reserved for new keyboard functions.
230-237	Defined terminal functions (READ, RECORD, etc.).
240-257	Reserved for terminal functions.
260-357	Generates escape sequences for external functions (Home Up, Home Down, etc.).
360-367	Generates sequences for F1-F8 keys. Normally a two byte escape sequence. The second byte is obtained by masking the high order bit.
370-376	Reserved for special functions.
377	Display Enhancements (CNTL F1).

TRANSLATION TABLES

Pointer = (col * 8) + row + Trans Table Base Address

Lower Case Table
(LWRASC)

0	00	1B	09	00	00	31	34	08
1	86	31	71	7A	0D	32	35	5C
2	83	32	77	78	5D	33	36	F4
3	80	33	65	63	3A	C4	D3	F5
.								
.								
.								
13	99	5E	5F	00	61	00	00	F0

Upper Case Table
(UPRASC)

00	1B	09	00	00	31	34	08
86	21	51	5A	0D	32	35	7C
83	22	57	58	7D	33	36	F4
80	23	45	43	2A	C4	D3	F5
99	7E	7F	00	41	00	00	F0

Section VIII. STANDARD DEVICE I/O

Your own firmware can perform input/output operations to and from the display, the left or right cartridge tape unit, a printer device, or an alternate I/O device by making calls to the routines GETIO or PUTIO.

GETIO, which resides at location 40603 (octal) or 4183 (hex), is used for accepting input from the display, cartridge tape units, or an alternate input device.

PUTIO, which resides at location 40631 (octal) or 4199 (hex), is used for transferring output to the display, the cartridge tape units, a printer device, or an alternate output device.

THE I/O BUFFERS

Data is passed to and from I/O devices one record at a time. Each record is passed via one of the two I/O buffers. Each buffer is 256 bytes long and has three variables (status, type, and length) associated with it. The locations of the buffers and their associated variables are as follows:

Location			
Octal	Hex	Name	Description
176000	FC00	IOBUF1	[illegible]
176377	FCFF		[illegible]
177472	FF3A	B1STAT	[illegible] I/O buffer
177471	FF39	B1TYPE	[illegible] I/O buffer
177470	FF38	B1LEN	[illegible] I/O buffer
176400	FD00	IOBUF2	[illegible]
176777	FDFF		[illegible]
177467	FF37	B2STAT	Status variable for second I/O buffer
177466	FF36	B2TYPE	Type variable for second I/O buffer
177465	FF35	B2LEN	Length variable for second I/O buffer

Data (IOBUFx)

One to 256 bytes of data are placed in the buffer, either by a device driver (for input) or by your firmware (for output), starting at address IOBUFx. There are no restrictions on the data.

Status Variable (BxSTAT)

The five low order bits are assigned to individual devices as follows (note that bits 5-15 are reserved):

Bit	Device Selected When Bit Set
0	Left cartridge tape unit.
1	Right cartridge tape unit.
2	Display.
3	Printer.
4	Alternate I/O device.

A buffer is free if its status is zero; otherwise it is owned by all devices whose bits are set.

Type Variable (BxTYPE)

There are basically three types of records: data records, end-of-file marks, and end-of-data marks. The content of BxTYPE specifies what type of record has been received (for input) or is to be transmitted (for output).

Minus one (−1 or 377 octal) specifies a data record, zero specifies an end-of-file mark, and plus one specifies an end-of-data mark. When reading from the display using GETIO, a BxTYPE of +2 specifies the end of the display page and +3 specifies the end of display memory.

Length Variable (BxLEN)

When transmitting or receiving a data record, the content of BxLEN specifies the number of valid data bytes in the I/O buffer. Note that a BxLEN of zero specifies a record size of 256 bytes (i.e., there are no "zero length" records).

When transmitting or recieving an end-of-file or end-of-data mark, BxLEN has no significance except when writing an end-of-file mark to one of the CTUs; in that particular case, BxLEN must be set to +1.

USING GETIO

To use GETIO you must first load some parameters into the following named locations:

Name	Octal Address	Hex Address
INPDEV	177516	FF4E
XFRLIM	177507	FF47

INPDEV specifies the input device from which data is to be read, as follows:

Bit	Device Selected When Bit Set ("1" or On)
0	Left cartridge tape unit.
1	Right cartridge tape unit.
2	Display.
4	Alternate input device.

XFRLIM specifies how much data is to be read, as follows:

Value	Meaning
−1	Read one record.
0	Read one file.
+1	Read until end-of-data.

The only use of XFRLIM is to control double buffering on CTU input. When XFRLIM is set correctly, the CTU input routine can determine (after reading a record) whether additional records must be read to satisfy the input request. If XFRLIM is set to 0 or +1 and an end-of-file or end-of-data mark has not yet been encountered, the CTU driver will start reading the next record from the cartridge tape to the other I/O buffer (under interrupt control) while your program is processing the current record.

The significance of XFRLIM when reading from an alternate input device depends entirely on the design of the particular alternate I/O driver.

Once INPDEV and XFRLIM have been properly set, you issue a call to the GETIO routine. GETIO obtains one of the I/O buffers and then reads a data record from the specified device into the buffer.

NOTE

If you are reading a record from the display, you should issue a call to the INTDSP routine before calling GETIO.

When control returns to your program the Carry flag indicates whether or not a read error occurred. If Carry is set, an error occurred; if it is reset, the record was transferred from the device to the I/O buffer successfully.

After testing the Carry flag, your program should check to see what the I/O buffer contains. Upon return from GETIO, the D and E registers contain a pointer to the first of the three status bytes associated with the particular buffer.

The first of these bytes (BxSTAT) contains the device parameter supplied via INPDEV; the second (BxTYPE) contains a record type parameter that specifies whether the buffer contains a valid data record, an end-of-file mark, or an end-of-data mark; the third (BxLEN) contains a byte count specifying the length (in bytes) of the record contained in the buffer.

BxTYPE is the byte you should examine first. Decrement the D register, load the contents of the specified location into the A register, OR the contents of the A register to itself, and then test the A register for −, 0, or +1. Minus indicates that the I/O buffer contains a valid data record; zero indicates that it contains an end-of-file mark; plus one indicates that it contains an end-of-data mark.

If the buffer contains a valid data record, examine the content of BxLEN to determine the size of the data record and then issue a call to the GETPTR routine. GETPTR uses the address of the status bytes (still contained in the D and E registers) to determine which I/O buffer is being used and then returns the address of that buffer to your program via the H and L registers. You then move the data from the buffer to where you want it stored.

After you have removed the data record from the I/O buffer you must clear BxSTAT (all zeros) to make the buffer available for other I/O operations.

You perform the above sequence of events once for each valid data record that is transferred from the specified device to the I/O buffer (note, however, that INPDEV and XFRLIM need to be initialized only the first time through the loop).

The following code illustrates the use of GETIO to read a file from the right cartridge tape unit.

```
GETIO   EQU   40603Q
INPDEV  EQU   177516Q
XFRLIM  EQU   177507Q
         .
         .
         .
        MVI   A,1
        STA   INPDEV      INPDEV = RIGHT CTU
        MVI   A,0
        STA   XFRLIM      XFRLIM = TRANSFER ONE FILE
   READ CALL  GETIO       CALL THE GETIO ROUTINE
        JC    ERROR       JUMP TO ERROR ROUTINE IF CARRY SET
        PUSH  D           SAVE STATUS POINTER
        DCX   D           CHECK FOR VALID DATA RECORD, EOF,
        LDAX  D              OR END-OF-DATA
        ORA   A
        JZ    EOF         END-OF-FILE
        JP    EOD         END-OF-DATA
        CALL  GETPTR      VALID DATA RECORD - GET BUFFER ADDRESS
              (ADDRESS RETURNED IN H,L)
         .
         .       MOVE DATA FROM BUFFER
         .
        POP   D           GET STATUS POINTER
        XRA   A           RELEASE THE BUFFER
        STAX  D
        JMP   READ        READ NEXT RECORD
```

USING PUTIO

To use PUTIO you must first load a parameter into the variable named OUTDEV OUTDEV, which resides at location 177515 (octal) or FF4D (hex), specifies the output device(s) to which data is to be sent, as follows (note that bits 5 through 15 are reserved)

Bit	Device Selected When Bit Set ("1" or On)
0	Left cartridge tape unit
1	Right cartridge tape unit
2	Display
3	Printer
4	Alternate output device

Once OUTDEV has been properly set, you issue a call to the GTIOB0 routine. GTIOB0 locates a free I/O buffer and then returns the address of the first of the buffer's three status bytes (BxSTAT) to your program via the H and L registers.

You must then claim the I/O buffer by setting BxSTAT to a non-zero value (zero indicates that the buffer is available for any program to use) and set BxTYPE to specify the type of record to be transmitted and BxLEN to specify the size (in bytes) of the record.

BxTYPE specifies what type of record is to be transferred, as follows

Value	Meaning
[illegible]	[illegible]
[illegible]	[illegible] mark
-1	[illegible] data mark

When writing to the display, a BxTYPE of 0 or -1 has no effect (BxLEN is also ignored). When writing to a printer device, a BxTYPE of 0 or -1 causes a form feed (BxLEN is ignored). When writing to an alternate output device, the effect of BxTYPE and BxLEN depends entirely on the design of the particular alternate I/O driver. When writing to an CTU, a BxTYPE of 0 causes an end-of-file mark to be written on the particular cartridge tape (BxLEN must be set to 1) and a BxTYPE of -1 causes an end-of-data mark to be written on the particular cartridge tape (BxLEN is ignored).

NOTE

PUTIO automatically moves the contents of OUTDEV into BxSTAT. When writing to the display, a printer, or an alternate output device, you may claim the I/O buffer by setting BxSTAT = OUTDEV. Because of the way the CTU interrupt handler operates, however, this practice could cause the buffer to be prematurely written to the tape. To be safe, it is recommended that you always set BxSTAT to octal 200 when claiming an I/O buffer.

Next you must determine the address of the buffer itself. You do this by moving the contents of H and L to the D and E registers and then issuing a call to the GETPTR routine. GETPTR uses the address of the status bytes (in D and E) to determine which buffer is being used and then returns the address of the buffer to your program via the H and L registers. You then use the buffer address to transfer the record from your program to the I/O buffer.

Once the data record is in the I/O buffer you issue a call to PUTIO. PUTIO, which expects the address of BxSTAT to be in the D and E registers, transfers the record to the specified output device(s) and then returns control to your program. Upon return from PUTIO, the Carry flag specifies whether or not a write error occurred. If Carry is set, an error occurred; if it is clear, the record was successfully transferred from the I/O buffer to the specified device(s). Note that a write operation to a cartridge tape will always end successfully unless the tape stalls or is removed.

***** IMPORTANT *****

Upon return from PUTIO, your program must ensure that the buffer is freed (BxSTAT=0). PUTIO automatically clears the device bits (0-4) but will not clear bit 8 which is set by the octal 200 you used in claiming the buffer. If you leave bit 8 set, the buffer will no longer be available for use by any program.

You perform the above sequence of events once for each data record that is transferred from your program to the specified device (note, however, that OUTDEV and XFRLIM need be initialized only the first time through the loop).

The following code illustrates the use of PUTIO to transfer a 72-byte record from your program to the display.

```
PUTIO   EQU   40631Q
OUTDEV  EQU   177515Q
XFRLIM  EQU   177507Q
         .
         .
         .
        MVI   A,4
        STA   OUTDEV       SELECT DISPLAY AS OUTPUT DEVICE
WRITE   CALL  GTIOB0       GET A BUFFER
        MVI   M,200Q       CLAIM IT
        PUSH  H            SAVE STATUS POINTER
        DCX   H
        MVI   M,377Q       SET BxTYPE TO -1
        DCX   H
        MVI   M,72         SET BxLEN TO 72
        XCHG               SWAP H,L AND D,E
        CALL  GETPTR       GET BUFFER ADDRESS
                             (ADDRESS RETURNED IN H,L)
         .
         .                 MOVE DATA RECORD TO BUFFER
         .
        POP   D            RESTORE STATUS POINTER
        CALL  PUTIO
        XCHG               SWAP H,L AND D,E
        MVI   M,0          FREE THE BUFFER (BxSTAT=0)
        JNC   WRITE        IF NO ERROR, WRITE NEXT RECORD
         .                 (ERROR HANDLING CODE)
         .
         .
```

Section IX. ALTERNATE I/O

The standard 2645A main code firmware already includes the mechanisms for selecting, either from the keyboard or programmatically using escape sequences, an input ("from" or 5s) device other than the CTUs or display and an output ("to" or 5d) device other than the CTUs, display, or printer. These two devices are known as alternate I/O devices.

When you press the gold key followed by the INSERT LINE key, the main code firmware recognizes that you have selected the alternate input device as the "from" device. Similarly, when you press the gold key followed by the INSERT CHAR key, the main code firmware recognizes that you have selected the alternate output device as the "to" device.

Try it. Press the gold key followed by the INSERT LINE key followed by the DISPLAY key. This selects the alternate input device as the "from" device and the display as the "to" device. Now press the READ key. The message "NO DEVICE DRIVER" appears on the screen. What does this mean? It means that the main code firmware recognized your device selections as being valid, but that when you tried to read from the "to" input device it branched to a location in memory where it expected to find the device driver for the alternate input device and found no driver. The capability is all there. You just have to write an appropriate device driver and load it into the memory area where the main code firmware expects to find it.

The same will happen when you use escape sequences. Load the sequence ␛ & p 5s 3d 4R into one of the function keys and then press it.

Now let's select an alternate output device as the "to" device. Press the gold key followed by the DISPLAY key followed by the INSERT CHAR key. This selects the display as the "from" device and the alternate output device as the "to" device. Now type something using the alphanumeric keys and then press the RECORD key. We get the same result: "NO DEVICE DRIVER". Again the capability is all there. We merely have to supply a driver.

Again the same will happen when you use escape sequences. Load the sequence ␛ & p 3s 5d 4R into one of the function keys, type something using the alphanumeric keys, and then press the function key.

ALTERNATE I/O DEVICE DRIVERS

If you'll think back to the memory map illustrated in figure 1-3, the area from 24K to 26K (decimal) is allocated to alternate I/O. When you are supplying alternate I/O code, the first location of this area (60000 octal) must contain the code for the ASCII character "P" to tell the main code firmware that a device driver is present. The second location must contain the value 24K/256. This value is used as an address check by the main code firmware. The next 27 bytes are used as an entry vector table that allows the main code firmware to pass control to the various routines in your driver. All access from the main code modules to your driver routines is achieved by CALLs to the appropriate entry point in this vector table. Control returns from your routines to the main code via subroutine RETurn calls.

Each entry in the vector table consists of three bytes. The first byte contains a JMP instruction (303 octal or C3 hex) and the next two bytes contain the address of the particular driver routine. If you do not need to use a particular routine, you must issue a RETurn instruction instead of a JMP in the vector table. The main code firmware expects the various entries in the vector table to be as follows:

Location	Driver Routine
060002	Initialization Routine
060005	Initialization Continuator
060010	Interrupt Processor
060013	Monitoring Routine
060016	Input Routine
060021	Output Routine
060024	Control Routine
060027	Status Routine
060032	Device Name Message

Descriptions of all of these routines, including the register contents when the routine is called and the expected register contents when control is returned to the main code, are presented under alternate I/O in the firmware portion of the 13255A Technical Information Package.

DEVICE STATUS

When you have alternate I/O code loaded into the terminal you can issue a device status escape sequence for device 5 and receive the status of the alternate I/O device. When you issue the escape sequence ᴱc & p 5 ^ control passes to the Status Routine (location 060027 in the Alternate I/O entry vector table) and this routine passes back three bytes of status information. As with the CTU and printer drivers, bits 8 through 5 of each byte always contain 0011. The remaining four bits of each byte are set at the discretion of the Alternate I/O Status Routine. (Note: The Status Routine always sets bits 8-5 of each byte to zeros; these bits are automatically set to 0011 by the terminal's main code).

SCNVEC AND INTVEC

There are two locations, named SCNVEC (9168 hex) and INTVEC (9165 hex), which are of particular interest to you when coding an alternate I/O module.

Control passes to SCNVEC from the wait loop. If SCNVEC contains a JMP instruction (303 octal or C3 hex) to one of your own routines, then your routine will be called once every time the terminal executes that portion of the wait loop.

Similarly control passes to INTVEC every time any type of interrupt occurs. If INTVEC contains a JMP instruction to one of your own routines, then your routine will be able to perform its functions based on the occurrence of a particular interrupt. When control passes to INTVEC, the A-register contains a code specifying what type of interrupt occurred. When your routine issues a RETurn instruction, the interrupt is then processed normally.

SAMPLE ALTERNATE I/O DRIVER

The source and object code for two alternate I/O drivers are presented on the next few pages.

```
==============================================================================================
ITEM     LOC    OBJECT CODE   SOURCE STATEMENTS                                    PAGE    1
==============================================================================================
       000000      .    .    .          ASB   BIN       ALEX1 - BUFFRD IN,OUT,CONTRL
       000000      .    .    .    *
       000000      .    .    .    *
       000000      .    .    .    *   ALTERNATE I/O EXERCISE 1:  WRITE ROUTINES TO
       000000      .    .    .    *   INPUT AND OUTPUT LINES OF TEXT FROM AN RS-232
       000000      .    .    .    *   DEVICE AT 9600 BAUD.  ALSO, WRITE A ROUTINE
       000000      .    .    .    *   THAT WILL ALLOW THE USER TO SEND A STRING OF
       000000      .    .    .    *   ASCII ONE'S TO THE DEVICE.  DO NOT USE
       000000      .    .    .    *   INTERRUPTS.
       000000      .    .    .    *
       000000      .    .    .    *   THIS CODE TALKS TO THE GP ASYNC DATACOM CARD
       000000      .    .    .    *   STRAPPED FOR MODULE 17, 9600 BAUD.
       000000      .    .    .    *
       000000      .    .    .    *   STRAPPING:
       000000      .    .    .    *       IAT CLOSED  (INHIBIT INTERRUPTS)
       000000      .    .    .    *       ALL OTHERS OPEN
       000000      .    .    .    *
       000000      .    .    .    *   THE CONTROL OPERATION IS INVOKED BY
       000000      .    .    .    *   "REWINDING" THE ALTERNATE I/O DEVICE.  THE
       000000      .    .    .    *   OPERATION MAY BE INVOKED VIA ESCAPE SEQUENCE,
       000000      .    .    .    *   I.E.  ESC & P 5U 0C (LOWER-CASE P AND U)
       000000      .    .    .    *   OR FROM THE KEYBOARD, I.E.  <GREEN> <REWIND>
       000000      .    .    .    *   <INSERT CHAR>.  THE NUMBER OF ASCII ONE'S TO
       000000      .    .    .    *   BE SENT MAY BE SPECIFIED AS AN UNSIGNED
       000000      .    .    .    *   PARAMETER IN AN ESCAPE SEQUENCE, OR MAY BE
       000000      .    .    .    *   SUPPLIED FROM THE KEYBOARD IN RESPONSE TO A
       000000      .    .    .    *   PROMPT FROM THE CONTROL ROUTINE.
       000000      .    .    .    *
       000000      .    .    .    *   NOTE: THE FIRMWARE DOES NOT LIKE RECORDS
       000000      .    .    .    *   BEGINNING WITH A LF.  IF A TERMINAL IS THE
       000000      .    .    .    *   ALTERNATE I/O DEVICE, AND TAPES ARE BEING USED
       000000      .    .    .    *   FOR INPUT FROM THE ALTERNATE I/O TERMINAL,
       000000      .    .    .    *   STRAPS E, G, AND H SHOULD BE OPEN ON THE
       000000      .    .    .    *   ALTERNATE I/O TERMINAL.
       000000      .    .    .    *
```

```
=================================================================================================
 ITEM     LOC    OBJECT CODE   SOURCE STATEMENTS                                      PAGE
=================================================================================================
   37   000000    .    .    .  **********************************
   38   000000    .    .    .  *  USEFUL VARIABLES, ENTRY POINTS  *
   39   000000    .    .    .  **********************************
   40   000000    .    .    .  *
   41   000000    .    .    .  *  I/O BUFFERS
   42   000000    .    .    .  *
   43   176000    .    .    .  IOBUF1 EQU   176000B
   44   177472    .    .    .  B1STAT EQU   177472B
   45   177471    .    .    .  B1TYPE EQU   177471B
   46   177470    .    .    .  B1LEN  EQU   177470B
   47   000000    .    .    .  *
   48   176400    .    .    .  IOBUF2 EQU   176400B
   49   177467    .    .    .  B2STAT EQU   177467B
   50   177466    .    .    .  B2TYPE EQU   177466B
   51   177465    .    .    .  B2LEN  EQU   177465B
   52   000000    .    .    .  *
   53   000000    .    .    .  *  ERROR RETURN VARIABLE
   54   000000    .    .    .  *
   55   177517    .    .    .  IOCERR EQU   177517B
   56   000000    .    .    .  *
   57   000000    .    .    .  *  INFORMATION FOR CONTROL ROUTINE
   58   000000    .    .    .  *
   59   177730    .    .    .  IOCTYP EQU   177730B   TYPE OF CONTROL OPERATION
   60   177725    .    .    .  IOCCNT EQU   177725B   PARAMETER VALUE
   61   177734    .    .    .  IOPSGN EQU   177734B   PARAMETER SIGN
   62   000000    .    .    .  *
   63   000000    .    .    .  *  MAIN CODE ROUTINES
   64   000000    .    .    .  *
   65   000100    .    .    .  DSPMSG EQU   100B      DISPLAY CHARACTER STRING
   66   177761    .    .    .  MSGPT1 EQU   177761B     POINTER TO STRING
   67   000202    .    .    .  CHINT  EQU   202B      PROCESS CHARACTER
   68   000106    .    .    .  DCNUM  EQU   106B      ACCUMULATE NUMERIC INPUT
   69   177724    .    .    .  RADIX  EQU   177724B     BASE FOR ACCUMULATION
   70   177736    .    .    .  IODATA EQU   177736B     ACCUMULATOR
   71   177610    .    .    .  CHAR   EQU   177610B     NEXT NUMERAL
   72   000000    .    .    .  *
   73   000000    .    .    .  *  KEYBOARD ROUTINES
   74   000000    .    .    .  *
   75   044005    .    .    .  ZGETKY EQU   44005B    GET KEYBOARD INPUT
   76   044024    .    .    .  ZBELL  EQU   44024B    RING BELL
 78  000000   .   .   .           ORG  60000B
 79  060000   .   .   .    ALSTRT EQU  *
 80  060000  120  .   .           DEF  120B        VERSION, CODE PRESENT FLAGS
 81  060001  140  .   .           DEF  ALSTRT/256  CODE PRESENT FLAG
 82  060002  303 043 140          JMP  INIT        INITIALIZATION
 83  060005  303 032 140          JMP  INIT2
 84  060010  303 034 140          JMP  INTRET      NO INTERRUPTS
 85  060013  303 040 140          JMP  RETURN      NO MONITOR
 86  060016  303 161 140          JMP  GETBUF      INPUT ROUTINE
 87  060021  303 106 140          JMP  PUTBUF      OUTPUT ROUTINE
 88  060024  303 303 140          JMP  CONTRL      CONTROL
 89  060027  303 041 140          JMP  DUMMY       NO STATUS
 90  060032   .   .   .    *
 91  060032   .   .   .    *  DUMMY INITIALIZATION ROUTINES
 92  060032   .   .   .    *
 93  060032   .   .   .    INIT2  EQU  *
 94  060032  267  .   .           ORA  A           NC => NO ERROR
 95  060033  311  .   .           RET
 96  060034   .   .   .    *
 97  060034   .   .   .    *  DUMMY INTRRUPT ROUTINE
 98  060034   .   .   .    *
 99  060034   .   .   .    INTRET EQU  *
100  060034  341  .   .           POP  H
101  060035  361  .   .           POP  PSW
102  060036  373  .   .           EI
103  060037  311  .   .           RET
104  060040   .   .   .    *
105  060040   .   .   .    *  DUMMY MONITOR
106  060040   .   .   .    *
107  060040   .   .   .    RETURN EQU  *
108  060040  311  .   .           RET
109  060041   .   .   .    *
110  060041   .   .   .    *  DUMMY STATUS ROUTINE
111  060041   .   .   .    *
112  060041   .   .   .    DUMMY  EQU  *
113  060041  067  .   .           STC              C => SEND ALL 0'S
114  060042  311  .   .           RET
```

```
=============================================================================
ITEM    LOC-   OBJECT CODE  SOURCE STATEMENTS                              PAGE
=============================================================================
   107440   .   .   .   IOSTAT EQU  107440B   I/O CARD STATUS
   000001   .   .   .   DATPRS EQU  1           DATA PRESENT BIT
   000002   .   .   .   RDYSND EQU  2           READY TO SEND BIT
   107400   .   .   .   IOINPT EQU  107400B   I/O INPUT ADDRESS
   107540   .   .   .   IOOUTP EQU  107540B   I/O OUTPUT ADDRESS
   107500   .   .   .   IOCNTL EQU  107500B   SET PARITY, BAUD, ETC.
   060043   .   .   .   *
   060043   .   .   .   * * * * * * * * * * * * * * * * * * * * * * * * *
   060043   .   .   .   *
   060043   .   .   .   *      INIT - SET UP CARD
   060043   .   .   .   *
   060043   .   .   .   INIT   EQU  *
   060043  076 077  .          MVI  A,77B      SELECT NO PARITY, 9600
   060045  062 100 217         STA  IOCNTL
   060050  001 000 000         LXI  B,0        NO BUFFER NEEDED
   060053  311  .   .          RET
   060054   .   .   .   *
   060054   .   .   .   * * * * * * * * * * * * * * * * * * * * * * * * *
   060054   .   .   .   *
   060054   .   .   .   *      GETCHR - GET ONE CHARACTER FROM THE
   060054   .   .   .   *           ALTERNATE I/O PORT
   060054   .   .   .   *
   060054   .   .   .   *      ENTRY:  DON'T CARE
   060054   .   .   .   *
   060054   .   .   .   *      EXIT :  A = CHARACTER
   060054   .   .   .   *
   060054   .   .   .   GETCHR EQU  *
   060054  072 040 217         LDA  IOSTAT     GET CARD STATUS
   060057  346 001  .          ANI  DATPRS     DATA PRESENT?
   060061  312 054 140         JZ   GETCHR     NO - WAIT
   060064  072 000 217         LDA  IOINPT     YES - GET DATA
   060067  311  .   .          RET
   060070   .   .   .   *
   060070   .   .   .   * * * * * * * * * * * * * * * * * * * * * * * * *
   060070   .   .   .   *
   060070   .   .   .   *      PUTCHR - OUTPUT ONE CHARACTER TO THE
   060070   .   .   .   *           ALTERNATE I/O PORT
   060070   .   .   .   *
   060070   .   .   .   *      ENTRY:  A = CHARACTER
   060070   .   .   .   *
   060070   .   .   .   *      EXIT :  NO CHANGES
   060070   .   .   .   *
   060070   .   .   .   PUTCHR EQU  *
   060070  365  .   .          PUSH PSW        SAVE CHAR
   060071   .   .   .   PCH010 EQU  *
   060071  072 040 217         LDA  IOSTAT     GET CARD STATUS
   060074  346 002  .          ANI  RDYSND     READY TO SEND?
   060076  312 071 140         JZ   PCH010     NO - WAIT
   060101  361  .   .          POP  PSW        YES - OUTPUT CHAR
   060102  062 140 217         STA  IOOUTP
   060105  311  .   .          RET
```

```
 168   Ø6Ø1Ø6    .   .   .   *
 169   Ø6Ø1Ø6    .   .   .   * * * * * * * * * * * * * * * * * * * * * * * * *
 17Ø   Ø6Ø1Ø6    .   .   .   *
 171   Ø6Ø1Ø6    .   .   .   *      PUTBUF - OUTPUT A BUFFER TO THE ALTERNATE
 172   Ø6Ø1Ø6    .   .   .   *           I/O PORT
 173   Ø6Ø1Ø6    .   .   .   *
 174   Ø6Ø1Ø6    .   .   .   *      ENTRY:  D,E -> BUFFER STATUS
 175   Ø6Ø1Ø6    .   .   .   *
 176   Ø6Ø1Ø6    .   .   .   *      EXIT :  D,E -> BUFFER STATUS
 177   Ø6Ø1Ø6    .   .   .   *              ALT I/O BIT (2ØB) CLEARED IN STATUS
 178   Ø6Ø1Ø6    .   .   .   *              IOCERR = S
 179   Ø6Ø1Ø6    .   .   .   *              NC (NO ERRORS POSSIBLE)
 18Ø   Ø6Ø1Ø6    .   .   .   *
 181   Ø6Ø1Ø6    .   .   .   PUTBUF EQU  *
 182   Ø6Ø1Ø6   Ø33  .   .          DCX  D            WHAT TYPE OF RECORD?
 183   Ø6Ø1Ø7   Ø32  .   .          LDAX D               1 => EVD  (JUST RETURN)
 184   Ø6Ø11Ø   267  .   .          ORA  A               Ø => EOF  (JUST RETURN)
 185   Ø6Ø111   362 145 14Ø         JP   PTB1ØØ         -1 => DATA (OUTPUT DATA)
 186   Ø6Ø114    .   .   .   *
 187   Ø6Ø114    .   .   .   *  DATA RECORD - OUTPUT IT
 188   Ø6Ø114    .   .   .   *
 189   Ø6Ø114   Ø33  .   .          DCX  D            GET LENGTH
 19Ø   Ø6Ø115   Ø32  .   .          LDAX D
 191   Ø6Ø116   1Ø7  .   .          MOV  B,A          B = COUNTER
 192   Ø6Ø117   173  .   .          MOV  A,E          GET POINTER TO BUFFER
 193   Ø6Ø12Ø   376 Ø7Ø  .          CPI  B1LEN
 194   Ø6Ø122   Ø41 ØØØ 374         LXI  H,IOBUF1
 195   Ø6Ø125   312 133 14Ø         JZ   PTBØ1Ø
 196   Ø6Ø13Ø   Ø41 ØØØ 375         LXI  H,IOBUF2
 197   Ø6Ø133    .   .   .   *
 198   Ø6Ø133    .   .   .   *  LOOP OUTPUTS EACH CHAR
 199   Ø6Ø133    .   .   .   *
 2ØØ   Ø6Ø133    .   .   .   PTBØ1Ø EQU  *
 2Ø1   Ø6Ø133   176  .   .          MOV  A,M          GET CHAR
 2Ø2   Ø6Ø134   Ø54  .   .          INR  L            UPDATE POINTER
 2Ø3   Ø6Ø135   315 Ø7Ø 14Ø         CALL PUTCHR       OUTPUT IT
 2Ø4   Ø6Ø14Ø   ØØ5  .   .          DCR  B            LAST CHAR?
 2Ø5   Ø6Ø141   3Ø2 133 14Ø         JNZ  PTBØ1Ø       NO - DO NEXT
 2Ø6   Ø6Ø144    .   .   .   *
 2Ø7   Ø6Ø144    .   .   .   *  BUFFER FINISHED - CLEAR ALT I/O BIT IN STATUS
 2Ø8   Ø6Ø144    .   .   .   *  AND RETURN SUCCESS
 2Ø9   Ø6Ø144    .   .   .   *
 21Ø   Ø6Ø144   Ø23  .   .          INX  D            D,E -> TYPE
 211   Ø6Ø145    .   .   .   PTB1ØØ EQU  *            (ENTRY POINT FOR EOF, EVD)
 212   Ø6Ø145   Ø23  .   .          INX  D            D,E -> STATUS
 213   Ø6Ø146   Ø32  .   .          LDAX D
 214   Ø6Ø147   346 357  .          ANI  -1-2ØB       TURN OFF BIT
 215   Ø6Ø151   Ø22  .   .          STAX D
 216   Ø6Ø152   Ø76 123  .          MVI  A,123B       SET IOCERR = S
 217   Ø6Ø154   Ø62 117 377         STA  IOCERR
 218   Ø6Ø157   267  .   .          ORA  A            NC => NO ERROR
 219   Ø6Ø16Ø   311  .   .          RET
```

```
==========================================================================
 ITEM     LOC    OBJECT CODE  SOURCE STATEMENTS                      PAGE    6
==========================================================================
  221   060161     .   .   .  *
  222   060161     .   .   .  * * * * * * * * * * * * * * * * * * * * * * * * *
  223   060161     .   .   .  *
  224   060161     .   .   .  *      GETBUF - GET A BUFFER OF DATA FROM THE
  225   060161     .   .   .  *           ALTERNATE I/O PORT
  226   060161     .   .   .  *
  227   060161     .   .   .  *      ENTRY:  DON'T CARE
  228   060161     .   .   .  *
  229   060161     .   .   .  *      EXIT :  D,E -> BUFFER STATUS
  230   060161     .   .   .  *              STATUS = 20B (FOR ALTERNATE I/O)
  231   060161     .   .   .  *              IOCERR = S
  232   060161     .   .   .  *              NC (NO ERROR RETURN)
  233   060161     .   .   .  *
  234   060161     .   .   .  GETBUF EQU  *
  235   060161     .   .   .  *
  236   060161     .   .   .  *  SEE WHETHER USER HIT RETURN (USER INTERRUPT)
  237   060161     .   .   .  *
  238   060161   315 005 110       CALL ZGETKY    USER HIT KEY?
  239   060164   302 203 140       JNZ  GTB010    NO - FIND BUFFER
  240   060167   376 015   .       CPI  15B       YES - IS IT RETURN?
  241   060171   302 161 140       JNZ  GETBUF    NO - CHECK FOR MORE KEYS
  242   060174     .   .   .  *
  243   060174     .   .   .  *  USER HIT RETURN - ABORT
  244   060174     .   .   .  *
  245   060174   076 125   .       MVI  A,125B    SET IOCERR = U
  246   060176   062 117 377       STA  IOCERR
  247   060201   067   .   .       STC            RETURN C => ERROR
  248   060202   311   .   .       RET
  249   060203     .   .   .  *
  250   060203     .   .   .  *  NO USER INTERRUPT - FIND AN EMPTY BUFFER
  251   060203     .   .   .  *
  252   060203     .   .   .  GTB010 EQU  *
  253   060203   072 072 377       LDA  B1STAT    IS BUFFER 1 FREE?
  254   060206   267   .   .       ORA  A
  255   060207   302 223 140       JNZ  GTB020    NO - CHECK BUFFER 2
  256   060212   021 072 377       LXI  D,B1STAT  YES - SET D,E -> STATUS
  257   060215   041 000 374       LXI  H,IOBUF1  SET H,L -> FIRST BYTE OF BUF
  258   060220   303 240 140       JMP  GTB030
  259   060223     .   .   .  GTB020 EQU  *       IS BUFFER 2 FREE?
  260   060223   072 067 377       LDA  B2STAT
  261   060226   267   .   .       ORA  A
  262   060227   302 161 140       JNZ  GETBUF    NO - WAIT
  263   060232   021 067 377       LXI  D,B2STAT  YES - SET D,E -> STATUS
  264   060235   041 000 375       LXI  H,IOBUF2  SET H,L -> FIRST BYTE OF BUF
  265   060240     .   .   .  *
  266   060240     .   .   .  *  EMPTY BUFFER FOUND - CLAIM IT
  267   060240     .   .   .  *
  268   060240     .   .   .  GTB030 EQU  *       ALTERNATE I/O BIT (20B)
  269   060240   076 020   .       MVI  A,20B       CLAIMS BUFFER
  270   060242   022   .   .       STAX D
  271   060243     .   .   .  *
  272   060243     .   .   .  *  FILL BUFFER UNTIL 100 CHARS REC'D OR LF REC'D
```

```
=================================================================================
 ITEM     LOC    OBJECT CODE  SOURCE STATEMENTS                        PAGE   7
=================================================================================
  273   Ø6Ø243    .   .   .   *
  274   Ø6Ø243    .   .   .   GTB1ØØ EQU  *
  275   Ø6Ø243   315 Ø54 14Ø         CALL GETCHR     GET A CHARACTER
  276   Ø6Ø246   167  .   .          MOV  M,A        PUT IN BUFFER
  277   Ø6Ø247   Ø54  .   .          INR  L          INCREMENT POINTER
  278   Ø6Ø25Ø   376 Ø12  .          CPI  12B        IS CHARACTER A LINE FEED?
  279   Ø6Ø252   312 263 14Ø         JZ   GTB12Ø     YES - QUIT
  28Ø   Ø6Ø255   175  .   .          MOV  A,L        NO - 1ØØ CHARS REC'D
  281   Ø6Ø256   376 144  .          CPI  1ØØ
  282   Ø6Ø26Ø   3Ø2 243 14Ø         JNZ  GTB1ØØ     NO - CONTINUE FILLING BUFFER
  283   Ø6Ø263    .   .   .   *
  284   Ø6Ø263    .   .   .   *  BUFFER FULL - SET LENGTH, TYPE
  285   Ø6Ø263    .   .   .   *
  286   Ø6Ø263    .   .   .   GTB12Ø EQU  *
  287   Ø6Ø263   Ø33  .   .          DCX  D
  288   Ø6Ø264   Ø33  .   .          DCX  D          D,E -> LENGTH
  289   Ø6Ø265   175  .   .          MOV  A,L        STORE LENGTH
  29Ø   Ø6Ø266   Ø22  .   .          STAX D
  291   Ø6Ø267   Ø23  .   .          INX  D          D,E -> TYPE
  292   Ø6Ø27Ø   Ø76 377  .          MVI  A,-1       -1 => DATA
  293   Ø6Ø272   Ø22  .   .          STAX D
  294   Ø6Ø273   Ø23  .   .          INX  D          D,E -> STATUS FOR EXIT
  295   Ø6Ø274   Ø76 123  .          MVI  A,123B     SET IOCERR = S
  296   Ø6Ø276   Ø62 117 377         STA  IOCERR
  297   Ø6Ø3Ø1   267  .   .          ORA  A          NC => NO ERROR
  298   Ø6Ø3Ø2   311  .   .          RET
```

```
================================================================================
   LOC    OBJECT CODE  SOURCE STATEMENTS                                 PAGE  8
================================================================================
 060303    .   .   .   *
 060303    .   .   .   * * * * * * * * * * * * * * * * * * * * * * * * *
 060303    .   .   .   *
 060303    .   .   .   *      CONTRL - HANDLE CONTROL OPERATIONS
 060303    .   .   .   *
 060303    .   .   .   *      ENTRY:  IOCTYP = OPERATION TYPE
 060303    .   .   .   *                0 => OUTPUT <P> 1'S
 060303    .   .   .   *                ANYTHING ELSE => DO NOTHING
 060303    .   .   .   *
 060303    .   .   .   *      EXIT :  IOCERR = S
 060303    .   .   .   *              NC (NO ERROR POSSIBLE)
 060303    .   .   .   *
 060303    .   .   .   CONTRL EQU  *
 060303   072 330 377         LDA  IOCTYP    WHAT TYPE OPERATION?
 060306   267  .   .          ORA  A
 060307   302 070 141         JNZ  CNT200    ANYTHING BUT 0 - QUIT
 060312   072 334 377         LDA  IOPSGN    PARAM SUPPLIED (NO SIGN)?
 060315   376 200  .          CPI  200B
 060317   312 053 141         JZ   CNT100    YES - DO OUTPUT
 060322    .   .   .   *
 060322    .   .   .   *  GET PARAMETER FROM KEYBOARD
 060322    .   .   .   *
 060322   076 015  .          MVI  A,15B     SEND CR TO DISPLAY
 060324   117  .   .          MOV  C,A
 060325   315 202 000         CALL CHINT
 060330   076 012  .          MVI  A,12B     SEND LF TO DISPLAY
 060332   117  .   .          MOV  C,A
 060333   315 202 000         CALL CHINT
 060336   041 077 141         LXI  H,PARMSG  DISPLAY PARAMETER REQUEST
 060341   042 361 377         SHLD MSGPT1
 060344   267  .   .          ORA  A         ADD TO DISPLAY
 060345   315 100 000         CALL DSPMSG
 060350   041 000 000         LXI  H,0       CLEAR ACCUMULATOR
 060353   042 336 377         SHLD IODATA
 060356   076 012  .          MVI  A,10      SET RADIX FOR DECIMAL
 060360   062 324 377         STA  RADIX
 060363    .   .   .   *
 060363    .   .   .   *  PROCESS NUMERIC INPUT UNTIL CR
 060363    .   .   .   *
 060363    .   .   .   CNT010 EQU  *
 060363   315 024 110         CALL ZBELL     RING BELL
 060366    .   .   .   CNT020 EQU  *
 060366   315 005 110         CALL ZGETKY    KEY HIT?
 060371   302 366 140         JNZ  CNT020      NO - WAIT
 060374   376 015  .          CPI  15B       IS IT RETURN?
 060376   312 033 141         JZ   CNT040      YES - ECHO CR, DO OUTPUT
 060401   376 060  .          CPI  60B       IS IT LESS THAN ASCII 0?
 060403   332 363 140         JC   CNT010      YES - RING BELL, WAIT
 060406   376 072  .          CPI  72B       IS IT GREATER THAN ASCII 9?
 060410   322 363 140         JNC  CNT010      YES - RING BELL, WAIT
 060413   062 210 377         STA  CHAR      DIGIT RECEIVED -
 060416   315 106 000         CALL DCNUM       ACCUMULATE VALUE
 352   060421   072 210 377           LDA   CHAR          ECHO TO DISPLAY
 353   060424   117  .   .            MOV   C,A           (CHINT WANTS CHAR IN C, A)
 354   060425   315 202 200           CALL  CHINT
 355   060430   303 366 140           JMP   CNTØ20      WAIT FOR MORE INPUT
 356   060433    .   .   .      *
 357   060433    .   .   .      *   ECHO CR TO DISPLAY
 358   060433    .   .   .      *
 359   060433    .   .   .      CNTØ40 EQU   *
 360   060433   117  .   .            MOV   C,A         CHINT WANTS CHAR IN C AND A
 361   060434   315 202 000           CALL  CHINT
 362   060437   076 012  .            MVI   A,12B       SEND LF TO DISPLAY
 363   060441   117  .   .            MOV   C,A
 364   060442   315 202 000           CALL  CHINT
 365   060445    .   .   .      *
 366   060445    .   .   .      *   MOVE ACCUMULATOR TO PARAMETER LOCATION
 367   060445    .   .   .      *
 368   060445   052 336 377           LHLD  IODATA
 369   060450   042 325 377           SHLD  IOCCNT
 370   060453    .   .   .      *
 371   060453    .   .   .      *   PARAMETER RECEIVED - OUTPUT 1'S
 372   060453    .   .   .      *
 373   060453    .   .   .      CNT100 EQU   *
 374   060453   072 325 377           LDA   IOCCNT      USE LOW BYTE OF IOCCNT
 375   060456   107  .   .            MOV   B,A         B IS COUNTER
 376   060457   076 061  .            MVI   A,61B       A = ASCII 1
 377   060461    .   .   .      CNT120 EQU   *
 378   060461   315 070 140           CALL  PUTCHR      OUTPUT 1
 379   060464   305  .   .            DCR   B           ALL FINISHED?
 380   060465   322 061 141           JNZ   CNT120
 381   060470    .   .   .      *
 382   060470    .   .   .      *   OPERATION FINISHED - RETURN SUCCESS
 383   060470    .   .   .      *
 384   060470    .   .   .      CNT200 EQU   *
 385   060470   076 123  .            MVI   A,123B      SET IOCERR = S
 386   060472   062 117 377           STA   IOCERR
 387   060475   267  .   .            ORA   A           NC => SUCCESS
 388   060476   311  .   .            RET
 389   060477    .   .   .      *
 390   060477    .   .   .      *   MSG REQUESTING PARAMETER: 202B = INVERSE VIDEO
 391   060477    .   .   .      *
 392   060477   202 105 116   PARMSG DEF   202B,'ENTER NUMBER OF ONES:'
 393   060525   316  .   .            DEF   316B        END OF MESSAGE FLAG
 394   060526    .   .   .            END
   0   ERRORS FOUND IN ASSEMBLY CODE .
```

```
SYMBOL  VALUE  REFERENCED ON
=================================================================================
ALSTRT Ø6ØØØØ     79,   81
B1LEN  17747Ø     46,  193
B1STAT 177472     44,  253,  256
B1TYPE 177471     45
B2LEN  177465     51
B2STAT 177467     49,  26Ø,  263
B2TYPE 177466     5Ø
CHAR   17761Ø     71,  35Ø,  352
CHINT  ØØØ2Ø2     67,  324,  327,  354,  361,  364
CNTØ1Ø Ø6Ø363    339,  347,  349
CNTØ2Ø Ø6Ø366    341,  343,  355
CNTØ4Ø Ø6Ø433    359,  345
CNT1ØØ Ø6Ø453    373,  318
CNT12Ø Ø6Ø461    377,  38Ø
CNT2ØØ Ø6Ø47Ø    384,  315
CONTRL Ø6Ø3Ø3    312,   88
DATPRS ØØØØØ1    117,  144
DCNUM  ØØØ1Ø6     68,  351
DSPMSG ØØØ1ØØ     65,  331
DUMMY  Ø6ØØ41    112,   89
GETBUF Ø6Ø161    234,   86,  241,  262
GETCHR Ø6ØØ54    142,  145,  275
GTBØ1Ø Ø6Ø2Ø3    252,  239
GTBØ2Ø Ø6Ø223    259,  255
GTBØ3Ø Ø6Ø24Ø    268,  258
GTB1ØØ Ø6Ø243    274,  282
GTB12Ø Ø6Ø263    286,  279
INIT   Ø6ØØ43    127,   82
INIT2  Ø6ØØ32     93,   83
INTRET Ø6ØØ34     99,   84
IOBUF1 176ØØØ     43,  194,  257
IOBUF2 1764ØØ     48,  196,  264
IOCCNT 177725     6Ø,  369,  374
IOCERR 177517     55,  217,  246,  296,  386
IOCNTL 1075ØØ    121,  129
IOCTYP 17773Ø     59,  313
IODATA 177736     7Ø,  333,  368
IOINPT 1074ØØ    119,  146
IOOUTP 10754Ø    12Ø,  165
IOPSGN 177734     61,  316
IOSTAT 10744Ø    116,  143,  161
MSGPT1 177761     66,  329
PARMSG Ø6Ø477    392,  328
PCHØ1Ø Ø6ØØ71    16Ø,  163
PTBØ1Ø Ø6Ø133    2ØØ,  195,  205
PTB1ØØ Ø6Ø145    211,  185
PUTBUF Ø6Ø1Ø6    181,   87
PUTCHR Ø6ØØ7Ø    158,  2Ø3,  378
RADIX  177724     69,  335
RDYSND ØØØØØ2    118,  162
RETURN Ø6ØØ4Ø    1Ø7,   85
ZBELL  Ø44Ø24     76,  34Ø
ZGETKY Ø44ØØ5     75,  238,  342

    53 SYMBOLS,    127 REFERENCES,      2 WORK TRACKS
```

```
===========================================================================================
ITEM     LOC    OBJECT CODE  SOURCE STATEMENTS                                  PAGE    1
===========================================================================================
   1   000000     .    .    .          ASB  BIN        ALEX2 - INTERRUPTING INPUT
   2   000000     .    .    .    *
   3   000000     .    .    .    *
   4   000000     .    .    .    *   ALTERNATE I/O EXERCISE 2:  WRITE A ROUTINE TO
   5   000000     .    .    .    *   ACCEPT INPUT FROM AN RS-232 PORT AT 9600 BAUD.
   6   000000     .    .    .    *   USE INTERRUPTS.
   7   000000     .    .    .    *
   8   000000     .    .    .    *   THIS CODE TALKS TO THE ASYNC MULTIPOINT CARD
   9   000000     .    .    .    *   STRAPPED FOR MODULE 17, 9600 BAUD.
  10   000000     .    .    .    *
  11   000000     .    .    .    *   STRAPPING:
  12   000000     .    .    .    *       INT CLOSED   (INTERRUPT ON ATN2)
  13   000000     .    .    .    *       PL6 CLOSED   (RESPOND TO POLL ON BIT 6)
  14   000000     .    .    .    *       2SB CLOSED   (ONE STOP BIT)
  15   000000     .    .    .    *       ALL OTHER STRAPS OPEN
  16   000000     .    .    .    *
  17   000000     .    .    .    *   NOTE THAT CHARACTERS ARE NOT SENT TO THE
  18   000000     .    .    .    *   DISPLAY BY THE INTERRUPT ROUTINE OR THE USUAL
  19   000000     .    .    .    *   MONITOR ROUTINE CALLED BY THE TIMER INTERRUPT
  20   000000     .    .    .    *   ROUTINE.  EITHER OF THESE SOLUTIONS COULD
  21   000000     .    .    .    *   CAUSE CHINT TO BE CALLED WHILE AN INTERRUPTED
  22   000000     .    .    .    *   INVOCATION OF CHINT IS WAITING TO BE RESUMED.
  23   000000     .    .    .    *   THIS COULD RESULT IN INCORRECT LINKS IN THE
  24   000000     .    .    .    *   DISPLAY, LOST CHARACTERS, ETC.  INSTEAD, THE
  25   000000     .    .    .    *   INTERRUPT ROUTINE PUTS CHARACTERS INTO A
  26   000000     .    .    .    *   CIRCULAR BUFFER.  ANOTHER ROUTINE (MONITR) IS
  27   000000     .    .    .    *   CALLED BY THE WAIT LOOP VIA SCNVEC TO SEND
  28   000000     .    .    .    *   CHARACTERS FROM THE BUFFER TO THE DISPLAY.
  29   000000     .    .    .    *
```

```
=================================================================================================
 ITEM     LOC    OBJECT CODE  SOURCE STATEMENTS                                        PAGE    2
=================================================================================================
   31   000000     .    .    .   ************************************
   32   000000     .    .    .   *  USEFUL VARIABLES, ENTRY POINTS  *
   33   000000     .    .    .   ************************************
   34   000000     .    .    .   *
   35   000000     .    .    .   *  ERROR RETURN VARIABLE, MESSAGE POINTER
   36   000000     .    .    .   *
   37   177517     .    .    .   IOCERR EQU  177517B
   38   177761     .    .    .   MSGPT1 EQU  177761B     POINTER TO STRING
   39   000000     .    .    .   *
   40   000000     .    .    .   *  MAIN CODE ROUTINES
   41   000000     .    .    .   *
   42   000202     .    .    .   CHINT  EQU  202B       PROCESS CHARACTER
   43   000000     .    .    .   *
   44   000000     .    .    .   *  VECTOR CALLED BY WAIT LOOP
   45   000000     .    .    .   *
   46   110550     .    .    .   SCNVEC EQU  110550B
   47   000000     .    .    .   *
   48   000000     .    .    .   *  ALTERNATE I/O VARIABLES
   49   000000     .    .    .   *
   50   177146     .    .    .   BUFADR EQU  177146B   ADDRESS OF CIRCULAR BUFFER
   51   177144     .    .    .   FILLPT EQU  177144B   FILL POINTER
   52   177142     .    .    .   EMPTY  EQU  177142B   EMPTYING POINTER
   53   000000     .    .    .   *
   54   000000     .    .    .   *  ADDRESS FOR ASYNC MULTIPOINT CARD
   55   000000     .    .    .   *
   56   107400     .    .    .   IOINPT EQU  107400B   INPUT DATA
   57   107440     .    .    .   IOCMND EQU  107440B   OUTPUT CONTROL
```

```
=====================================================================================
 ITEM     LOC    OBJECT CODE  SOURCE STATEMENTS                             PAGE   3
=====================================================================================
   59   000000    .   .   .          ORG  60000B
   60   060000    .   .   .   ALSTRT EQU  *
   61   060000  120   .   .          DEF  120B       VERSION, CODE PRESENT FLAGS
   62   060001  140   .   .          DEF  ALSTRT/256  CODE PRESENT FLAG
   63   060002  303 101 140          JMP  INIT       INITIALIZATION
   64   060005  303 105 140          JMP  INIT2
   65   060010  303 141 140          JMP  INTRPT     INTERRUPTS
   66   060013  303 032 140          JMP  RETURN     NO MONITOR
   67   060016  303 033 140          JMP  DUMMY      NO INPUT ROUTINE
   68   060021  303 033 140          JMP  DUMMY      NO OUTPUT ROUTINE
   69   060024  303 033 140          JMP  DUMMY      NO CONTROL
   70   060027  303 033 140          JMP  DUMMY      NO STATUS
   71   060032    .   .   .   *
   72   060032    .   .   .   *  DUMMY MONITOR
   73   060032    .   .   .   *
   74   060032    .   .   .   RETURN EQU  *
   75   060032  311   .   .          RET
   76   060033    .   .   .   *
   77   060033    .   .   .   *  DUMMY INPUT, OUTPUT, CONTROL, STATUS
   78   060033    .   .   .   *
   79   060033    .   .   .   DUMMY  EQU  *
   80   060033  041 050 140          LXI  H,NODRVR   SET ERROR MESSAGE
   81   060036  042 361 377          SHLD MSGPT1
   82   060041  076 106   .          MVI  A,106B     SET IOCERR = F
   83   060043  062 117 377          STA  IOCERR
   84   060046  067   .   .          STC             C => ERROR
   85   060047  311   .   .          RET
   86   060050  116 117 040   NODRVR DEF  'NO DEVICE DRIVER ROUTINE',316B
```

```
=========================================================================================
 ITEM    LOC    OBJECT CODE  SOURCE STATEMENTS                                   PAGE
=========================================================================================
   88  060101    .   .   .  *
   89  060101    .   .   .  * * * * * * * * * * * * * * * * * * * * * * * * * *
   90  060101    .   .   .  *
   91  060101    .   .   .  *      INIT, INIT2 - INITIALIZATION
   92  060101    .   .   .  *
   93  060101    .   .   .  *      GET AND INITIALIZE 256-BYTE CIRCULAR BUFFER
   94  060101    .   .   .  *      SET UP MONITOR ROUTINE
   95  060101    .   .   .  *
   96  060101    .   .   .  *
   97  060101    .   .   .  INIT   EQU  *
   98  060101   001 000 001        LXI  B,256        ASK FOR 256-BYTE BUFFER
   99  060104   311  .   .         RET
  100  060105    .   .   .  INIT2  EQU  *
  101  060105   353  .   .         XCHG              H,L => BUFFER
  102  060106   042 146 376        SHLD BUFADR       SAVE BUFFER ADDRESS
  103  060111   042 144 376        SHLD FILLPT       INIT FILL POINTER
  104  060114   042 142 376        SHLD EMPTY        INIT EMPTY POINTER
  105  060117   076 303  .         MVI  A,303B       SET UP MONITOR ROUTINE
  106  060121   062 150 221        STA  SCNVEC
  107  060124   041 174 140        LXI  H,MONITR
  108  060127   042 151 221        SHLD SCNVEC+1
  109  060132   076 377  .         MVI  A,377B       SET BAUD RATE, PARITY
  110  060134   062 040 217        STA  IOCMND
  111  060137   267  .   .         ORA  A            NC => NO ERROR
  112  060140   311  .   .         RET
```

```
==============================================================================
ITEM     LOC    OBJECT CODE   SOURCE STATEMENTS                         PAGE   5
==============================================================================
       060141    .   .   .   *
       060141    .   .   .   * * * * * * * * * * * * * * * * * * * * * * * * *
       060141    .   .   .   *
       060141    .   .   .   *      INTERRUPT ROUTINE
       060141    .   .   .   *
       060141    .   .   .   *      ENTRY:  PSW, H & L PUSHED
       060141    .   .   .   *
       060141    .   .   .   *      EXIT :  INTERRUPT CLEARED, RET FROM INT
       060141    .   .   .   *
       060141    .   .   .   INTRPT EQU  *
       060141   052 144 376         LHLD FILLPT     GET FILL POINTER
       060144   072 000 217         LDA  IOINPT     GET CHARACTER
       060147   346 177  .          ANI  177B       CLEAR HIGH BIT
       060151   167  .   .          MOV  M,A        PUT INTO CIRCULAR BUFFER
       060152   043  .   .          INX  H          INCREMENT FILL POINTER
       060153   072 146 376         LDA  BUFADR     REACHED END OF BUFFER?
       060156   275  .   .          CMP  L
       060157   302 165 140         JNZ  INT020     NO -
       060162   052 146 376         LHLD BUFADR     YES - POINT TO BEGINNING
       060165    .   .   .   INT020 EQU  *
       060165   042 144 376         SHLD FILLPT     STORE NEW FILL POINTER
       060170   341  .   .          POP  H          RETURN FROM INTERRUPT
       060171   361  .   .          POP  PSW
       060172   373  .   .          EI
       060173   311  .   .          RET
```

```
****************************************************************************************
 ITEM    LOC   OBJECT CODE  SOURCE STATEMENTS                                     PAGE
****************************************************************************************
 140   060174    .   .   .  *
 141   060174    .   .   .  * * * * * * * * * * * * * * * * * * * * * * * * * *
 142   060174    .   .   .  *
 143   060174    .   .   .  *      MONITR - DISPLAY ANY CHARACTERS REC'D
 144   060174    .   .   .  *
 145   060174    .   .   .  *      ENTRY:  DON'T CARE
 146   060174    .   .   .  *
 147   060174    .   .   .  *      EXIT :  EMPTY POINTER = FILL POINTER
 148   060174    .   .   .  *
 149   060174    .   .   .  MONITR EQU  *
 150   060174   052 142 376        LHLD EMPTY      GET EMPTY POINTER
 151   060177   072 144 376        LDA  FILLPT     SAME AS FILL POINTER?
 152   060202   275  .   .         CMP  L
 153   060203   310  .   .         RZ              YES - QUIT
 154   060204   176  .   .         MOV  A,M        NO - DISPLAY CHARACTER
 155   060205   117  .   .         MOV  C,A
 156   060206   345  .   .         PUSH H            (SAVE EMPTY POINTER)
 157   060207   315 202 000        CALL CHINT
 158   060212   341  .   .         POP  H
 159   060213   043  .   .         INX  H          INCREMENT EMPTY POINTER
 160   060214   072 146 376        LDA  BUFADR     HIT END OF BUFFER?
 161   060217   275  .   .         CMP  L
 162   060220   302 226 140        JNZ  MON020
 163   060223   052 146 376        LHLD BUFADR     YES - POINT TO BEGINNING
 164   060226    .   .   .  MON020 EQU  *
 165   060226   042 142 376        SHLD EMPTY      STORE NEW EMPTY POINTER
 166   060231   303 174 140        JMP  MONITR     AND CHECK FOR ANY MORE CHARS
 167   060234    .   .   .         END
   0  ERRORS FOUND IN ASSEMBLY CODE .
```

```
SYMBOL  VALUE  REFERENCED ON
================================================================================
ALSTRT 060000      60,    62
BUFADR 177146      50,   102,   129,   132,   160,   163
CHINT  000202      42,   157
DUMMY  060033      79,    67,    68,    69,    70
EMPTY  177142      52,   104,   150,   165
FILLPT 177144      51,   103,   124,   134,   151
INIT   060101      97,    63
INIT2  060105     100,    64
INT020 060165     133,   131
INTRPT 060141     123,    65
IOCERR 177517      37,    83
IOCMND 107440      57,   110
IOINPT 107400      56,   125
MON020 060226     164,   162
MONITR 060174     149,   107,   166
MSGPT1 177761      38,    81
NODRVR 060050      86,    80
RETURN 060032      74,    66
SCNVEC 110550      46,   106,   108

    19 SYMBOLS,     53 REFERENCES,       1 WORK TRACKS
```

Section X. DATA COMM MODULE

The 2640 Series terminals offer both multi-point and basic point-to-point data communications capabilities. The data comm firmware is separate from the other main code modules and normally resides in locations 20K-24K. This physical independence from the other main code modules means that you may alter or expand the data comm code without affecting the operation of the terminal's other main code firmware.

Before you consider doing so, however, you should become thoroughly familiar with the existing data comm capabilities by studying chapter five of the 2645A/S Reference Manual (part number 02645-90005).

DATA COMM/MAIN CODE INTERFACE

If you'll think back to the memory map illustrated in figure 1-3, the area from 20K to 24K (decimal) is allocated to data comm code.

The first location of this area (50000 octal) must contain the code for the ASCII character "P" to tell the main code firmware that a data comm code module is present.

The second location must contain the value 20K/256. This value is used as an address check by the main code firmware.

The next six locations are used for defining certain control characters (such as the transfer trigger, record separator, and block separator) and for inhibiting the programmatic alteration of keyboard I/F jumpers S through Z.

The 33 bytes starting at location 050010 (octal) are used as an entry vector table that allows the main code firmware to pass control to the various routines in the data comm module. All access from main code to the data comm routines is achieved by CALLs to the appropriate entry point in this vector table. Control returns from the data comm routines to the main code via subroutine RETurn calls.

Each entry in the vector table consists of three bytes. The first byte contains a JMP instruction (303 octal or C3 hex) and the next two bytes contain the address of the particular data comm routine. If a certain data comm environment does not need to use a particular routine, you must issue a RETurn instruction instead of a JMP in the vector table. The main code firmware expects the various entries in the vector table to be as follows:

Octal Location	Data Comm Routine
050010	Initialization Routine
050013	Initialization Continuator
050016	Monitoring Routine
050021	Control Routine
050024	Data Comm Self-Test Routine
050027	Character Input Routine
050032	Character Output Routine
050035	Binary Input Routine
050040	Start Binary Output Routine
050043	End Binary Output Routine
050046	Data Comm Interrupt Handler

Descriptions of all of these routines, including the register contents when the routine is called and the expected register contents when control is returned to the main code, are presented under data communications/main code interface in the firmware portion of the 13255A Technical Information Package.

GENERAL OPERATION

To give you a general feel for how the existing data comm firmware operates in conjunction with the other main code modules, let's look at a typical input (receive) and output (transmit) operation.

Receiving Data

When the data comm interface receives one or more characters from the remote device it causes an interrupt. In response to the interrupt, the main code passes control to location 40B (refer to table 1-4 in Section I). This location contains a JMP instruction to location 050046B which is the entry point for the data comm interrupt handler in the data comm module's entry vector table. This interrupt routine accepts the incoming characters directly from the data comm interface board and stores them in the data comm buffer (updating certain buffer pointers in the process). When all the incoming characters have been stored, the routine sets a context flag to specify that there is data in the buffer and then issues a RETurn instruction.

The data comm input routine is called regularly from the wait loop. This routine examines the context flags to see if there is any incoming data in the data comm buffer yet to be processed. If a context flag indicates that there is, then the data comm input routine processes the data appropriately.

The Character Input Routine (entry location 050027B) fetches 8-bit characters from the data comm buffer, masks out the 8th bit, and passes on a 7-bit character code. The Binary Input Routine (entry location 050035B) fetches an 8-bit character from the data comm buffer and passes on the full 8-bit code. Like the interrupt handler routine, these two routines also manipulate the buffer pointers and context flags appropriately. Each time one of these input routines is called, one character is removed from the data comm buffer and passed back to the main code. Thus, each time through the wait loop, one incoming character from the data comm buffer can be processed. When all the data has been extracted from the data comm buffer, the input routine resets the context flags to indicate that the buffer is empty.

Transmitting Data

When a main code module wants to transmit a character via the data comm interface, it places the character in the A-register and transfers control to the Character Output Routine (entry location 050032B). The calling routine uses the Carry Bit to indicate whether or not the character is the last one in the block. The Character Output Routine extracts the character from the A-register, sends it directly to the data comm interface board and then issues a RETurn instruction. (If the character was specified as being the last one in a block, the routine transmits the appropriate record and/or block separator characters before returning control to main code.)

Appendix A. PROGRAM REFERENCE TABLES

ASCII-Hex-Octal Conversion

ASCII	HX	OCT	ASCII	HX	OCT	ASCII	HX	OCT	ASCII	HX	OCT
NULL	00	000	@	40	100		80	200		C0	300
STH	01	001	A	41	101		81	201		C1	301
STX	02	002	B	42	102		82	202		C2	302
ETX	03	003	C	43	103		83	203		C3	303
EOT	04	004	D	44	104		84	204		C4	304
ENQ	05	005	E	45	105		85	205		C5	305
ACK	06	006	F	46	106		86	206		C6	306
BELL	07	007	G	47	107		87	207		C7	307
BS	08	010	H	48	110		88	210		C8	310
HT	09	011	I	49	111		89	211		C9	311
LF	0A	012	J	4A	112		8A	212		CA	312
VT	0B	013	K	4B	113		8B	213		CB	313
FF	0C	014	L	4C	114		8C	214		CC	314
CR	0D	015	M	4D	115		8D	215		CD	315
SO	0E	016	N	4E	116		8E	216		CE	316
SI	0F	017	O	4F	117		8F	217		CF	317
DLE	10	020	P	50	120		90	220		D0	320
DC1	11	021	Q	51	121		91	221		D1	321
DC2	12	022	R	52	122		92	222		D2	322
DC3	13	023	S	53	123		93	223		D3	323
DC4	14	024	T	54	124		94	224		D4	324
NACK	15	025	U	55	125		95	225		D5	325
SYN	16	026	V	56	126		96	226		D6	326
ETB	17	027	W	57	127		97	227		D7	327
CAN	18	030	X	58	130		98	230		D8	330
EM	19	031	Y	59	131		99	231		D9	331
SUB	1A	032	Z	5A	132		9A	232		DA	332
ESC	1B	033	[	5B	133		9B	233		DB	333
FS	1C	034	\	5C	134		9C	234		DC	334
GS	1D	035	]	5D	135		9D	235		DD	335
RS	1E	036	^	5E	136		9E	236		DE	336
US	1F	037	_	5F	137		9F	237		DF	337
Space	20	040	`	60	140		A0	240		E0	340
!	21	041	a	61	141		A1	241		E1	341
"	22	042	b	62	142		A2	242		E2	342
#	23	043	c	63	143		A3	243		E3	343
$	24	044	d	64	144		A4	244		E4	344
%	25	045	e	65	145		A5	245		E5	345
&	26	046	f	66	146		A6	246		E6	346
'	27	047	g	67	147		A7	247		E7	347
(	28	050	h	68	150		A8	250		E8	350
)	29	051	i	69	151		A9	251		E9	351
*	2A	052	j	6A	152		AA	252		EA	352
+	2B	053	k	6B	153		AB	253		EB	353
,	2C	054	l	6C	154		AC	254		EC	354
-	2D	055	m	6D	155		AD	255		ED	355
.	2E	056	n	6E	156		AE	256		EE	356
/	2F	057	o	6F	157		AF	257		EF	357

ASCII	HX	OCT	ASCII	HX	OCT	ASCII	HX	OCT	ASCII	HX	OCT
0	30	060	p	70	160		B0	260		F0	360
1	31	061	q	71	161		B1	261		F1	361
2	32	062	r	72	162		B2	262		F2	362
3	33	063	s	73	163		B3	263		F3	363
4	34	064	t	74	164		B4	264		F4	364
5	35	065	u	75	165		B5	265		F5	365
6	36	066	v	76	166		B6	266		F6	366
7	37	067	w	77	167		B7	267		F7	367
8	38	070	x	78	170		B8	270		F8	370
9	39	071	y	79	171		B9	271		F9	371
:	3A	072	z	7A	172		BA	272		FA	372
;	3B	073	{	7B	173		BB	273		FB	373
<	3C	074	|	7C	174		BC	274		FC	374
=	3D	075	}	7D	175		BD	275		FD	375
>	3E	076	~	7E	176		BE	276		FE	376
?	3F	077	DEL	7F	177		BF	277		FF	377

Main Routine Entry Vectors

Name	Description	Entry Location
DSPMSG	Display message	40
RSTDSP	Restore normal display	43
DCNUM	Accumulate digit for escape sequence	46
DCPLUS	Add plus sign to parameter	49
DCMNUS	Add minus sign to parameter	4C
ESCEND	Terminate escape sequence	4F
CHKLIM	Check parameter limits	52
CLBLXF	Clear pending multi-character transfer flag	55
SBLXF0	Set pending flag for escape sequence initiated multi-character transfer	58
SBLXFA	Set pending flag for non-block mode keyboard initiated multi-character transfer	5B
STRTBL	Initialize for display transmission	5E
CURPH	Home cursor (exclude transmit-only fields)	61
CURPHD	Home down cursor	64
FRECNT	Check number of free display blocks	67
PTBLK	Add display block to free list	6A
CLEARL	Clear line	6D
CLEARS	Clear display	70
FNDTB2	Set bit in byte	73
SDTERM	Send block terminator and end transfer	76
SDTRM1	Send block terminator only	79
XPUTDC	Transmit character	7C
TRMTST	Perform terminal self-test	7F
CHINT0	Perform character function	82
INITD0	Initialize for display tear-apart	85
GETDSP	Get next display character for output	88
LNFEED	Perform Line-Feed	8B
EXPAND	Expand display control byte	8E
NXTCHR	Get next display character in display chain	91
GETDCM	Process data communications input	94
MLKSC0	Locate first unlocked row	97
MLKOF0	Turn off MEMORY LOCK	9A
HANGU0	Hang terminal on fatal error	9D
BUFMSG	Pointer to buffer overflow message	A0
DCTEST	Perform data communications self-test	A2
IORMGO	Execute code in optional ROM	A5
BN2DEC	Convert 16-bit binary to decimal	A8
BN2DE0	Convert 8-bit binary to decimal	AB
RCADRA	Locate current cursor position	AE
GIMODE	Check for page mode	B1

Keyboard Routine Entry Vectors

Name	**Description**	**Entry Location**
INITKB	Initialize keyboard	4802
GTKEY	Get keyboard input	4805
KBCTL	Keyboard control	4808
KBMON	Monitor keyboard	480B
SETMD1	Set terminal Mode 1 flags	480E
CLRMD1	Clear terminal Mode 1 flags	4811
BELL	Sound the keyboard bell	4814
SETXMT	Turn on the TRANSMIT indicator	4817
CLRXMT	Turn off the TRANSMIT indicator	481A
STJMPR	Set the Jumper Escape Sequence processor	481D
STLKYS	Set the Latching Key Escape Sequence processor	4820
ALPCHK	Alpha field check	4823
NUMCHK	Numeric field check	4826

Keyboard Control Routines (Firmware)

KBCTL (ZKBCTL) A = Control Code

1 = Lock keyboard	8 = Check for I/O key down
2 = Unlock keyboard	9 = Stop key repeat
3 = Repeat last key	10 = Check for Break key
4 = Set permanent block mode	11 = Switch character set
5 = Set Self-Test start mode	12 = Set foreign mode
6 = End Self-Test start mode	13 = Set bi-lingual mode
7 = Reset keyboard	14 = Set foreign mode 1

Alternate I/O Entry Vectors

Name	Description	Entry Location
IN1ALT	Initialization routine	6002
IN2ALT	Initialization continuator	6005
ALTINT	Interrupt processor	6008
ALTMON	Monitoring routine	600B
ALT2BF	Input routine	600E
BF2ALT	Output routine	6011
ALTCTL	Control routine	6014
STAALT	Status routine	6017
MSGALT	Device name message	601A

Data Communications Entry Vectors

Name	Description	Entry Location
INITDC	Initialization routine	5008
INI2DC	Initialization continuator	500B
DCMON	Monitoring routine	500E
DCCTL	Control routine	5011
DCTST	Self-test routine	5014
GETDC	Character input routine	5017
PUTDC	Character output routine	501A
GETBIN	Binary input routine	501D
STBIN	Start binary output routine	5020
ENDBIN	End binary output routine	5023
DCINTR	Data comm interrupt handler	5026