KAYPRO

KAYLINK

TECHNICAL MANUAL

TECHNICAL MANUAL FOR KAYLINK

THE FSUCC HASP STATION EMULATOR (HASTE™)
VERSION 1.3

REVISION F

by
Sam Adams
Leslie M. Brooks
Doug Lee
Jim Lyons
Ira B. Margolis
Eric M. Pepke

Computing Center
The Florida State University
Tallahassee, Florida 32306
January, 1984

Important Notice

The program KAYLINK is a version of the HASTE™ HASP Station Emulator. The two versions do not differ in any functional respect and are considered the same program for purposes of copyright. KAYLINK differs from HASTE™ only in cosmetic ways; the word HASTE has been changed to the word KAYLINK, for example. Similarly, this manual does not differ from the HASTE™ manual in any functional respect.

Table of Contents

Figures

The number of each figure begins with the number of the chapter in which it appears.

1.0 How to Use this Manual

This is a **technical manual** and is designed to solve problems of a highly technical nature. If your questions are simply related to ordinary use of KAYLINK, consult the Users' Manual at the front of the binder. If, on the other hand, you have a special problem you want KAYLINK to solve, or if you are just interested in how KAYLINK does things, you have come to the right place.

Since this is a technical manual, it assumes a basic knowledge of communications and of the CP/M operating system as well as a larger vocabulary of technical terms. Also, most of Chapter 4 assumes knowledge of 8080 and Z-80 assembly language, programming concepts, interrupts, and synchronous communications.

Chapter 2 describes facts about KAYLINK which are too technical for the Users' Manual but which might be necessary in trying to get KAYLINK to work under certain circumstances.

Chapter 3 gives a general description of the HASP protocol that KAYLINK uses. Some of the options to customize KAYLINK described in Section 4.1 pertain to concepts discussed in this chapter.

Chapter 4 describes the entire process of installation, the process of taking a distribution copy of KAYLINK and making it useful. It includes a descripion of all the command line arguments used to customize KAYLINK for your application.

Chapter 5 describes in more detail one phase of the installations: installation by code insertion by HIOS. Your copy of KAYLINK already has a HIOS, and this section may help you if you wish to modify it.

Chapter 6 describes another phase of installing KAYLINK: connecting the computer to a communications line.

1.1 Conventions

Most of the manual is in the type style of this paragraph. Another type style, **which looks like this**, is used for special purposes such as the following:

- File names
- Assembly language instructions
- Messages the computer displays
- Typed commands

Numbers which appear in the text are usually in decimal representation. Hexadecimal numbers are followed by an "H." For example, 20 represents decimal 20, while 20H represents hexadecimal two-zero, or decimal thirty-two.

2.0 Facts About KAYLINK

This chapter contains bits of technical information about the KAYLINK program itself. This information includes buffer sizes, communication characteristics, and operating system requirements.

2.1 Communication

The **approximate** maximum speed of KAYLINK is 9600 baud times the clock speed in MHZ for the interrupt version and 2400 baud times the clock speed in MHZ for the non-interrupt version. KAYLINK will communicate well at 9600 baud on a KAYPRO II and up to 19.2 kilobaud on KAYPRO 10.

KAYLINK, as a good follower of the HASP protocol, will send out ENQ's until it gets a response. This means that it really does not matter to KAYLINK when the communications are established. The other computer with which it is communicating, however, may hang up the phone after waiting a certain period of time. For this reason, it may be necessary to bring up KAYLINK before dialing the phone.

If you press the RETURN key alone while using the HASP Station Console, KAYLINK will send a record containing a single null character. The reason for this is that, while null records are sometimes necessary, especially in interactive applications, most host sites are not prepared to accept records of zero length. The compromise of using a single null character causes no problems with most systems.

Lines of zero length in a disk file or the Text Editor are sent as records containing a single space character.

When two microcomputers running KAYLINK are made to communicate with each other and neither is told to be a host, a race will be held to determine who has the honor of hosting the other. One computer will eventually send a signon card, at which point the other will say to itself, "Oh! I'm supposed to be a host. I get it," and will promptly behave as one. If you want one of the computers in particular to be the host, use the command **KAYLINK HOST** to load and run KAYLINK on that computer.

If KAYLINK receives a bad buffer of data, it will ask for that buffer to be transmitted again. After five unsuccessful tries, the message **COMMUNICATIONS PROBLEM** will appear in the Status Window. KAYLINK will continue to ask for the buffer until it receives it properly.

2.2 Buffer Sizes and Magic Numbers

A HASP **buffer** is an entire message sent at once including all protocol bytes. The largest buffer which KAYLINK can receive or send is 800 (decimal) bytes long. When KAYLINK is sending data, this is no problem. However, you should make sure that KAYLINK never receives a buffer of data larger than 800 bytes. Most host sites have provisions for specifying the largest buffer size. If KAYLINK receives a buffer which is too large, it will send a NAK (negative acknowledge) to the other computer.

The largest record of data which KAYLINK can send is 160 (decimal) bytes long. When sending transparent data this is no problem because record boundaries are not significant. When sending non-transparent data such as text all lines larger than 160 characters will be broken up into more than one record, which will be interpreted as more than one line. Lines shorter than 160 characters will be sent as they are, without any space padding.

The HASP Station Console will only allow up to 160 characters to be entered from the keyboard before the line is automatically sent.

2.3 The Text Editor

When you use the Text Editor, you can move the cursor anywhere on the screen without affecting the text. Unlike many other full-screen text editors, the number of lines is not determined until you actually enter some text. The lowest line yet typed on the Text Screen therefore determines the number of lines.

For example, if you type a single line in the Text Editor, there will only be one line, even if you move the cursor to the bottom of the screen. If you then enter a line at the bottom, there will be seventeen lines, of which the middle fifteen will be blank.

Lines on the Text Screen are of various lengths which are determined by what has actually been typed. If you have an application which requires a certain number of spaces at the end of a line, you must type them. It is sufficient for such an application to type a single space in the last position where you want a space, because the editor will fill in the remaining spaces for you.

When text created with the Text Editor is saved on a disk file, hidden text is marked with a CTRL-Q character at the beginning and end of each hidden string. Although this does not secure the hidden text from users of standard CP/M commands, it does ensure that no one may discover it while using KAYLINK. When text is sent over the line, the CTRL-Q characters are discarded.

2.4 Error Trapping

KAYLINK is set up to trap and recover from all BDOS errors. BDOS error messages are **never** displayed on the screen. KAYLINK is also designed to trap BIOS errors for most systems. For some systems, particularly those with custom BIOSes, the BIOS error trapping may not work.

The problem is that KAYLINK traps BIOS error messages by patching the BIOS console output jump vector. If your BIOS error routines display all their messages using this jump vector, there is no problem. If, however, they display messages without going through the jump vector, there is no way that KAYLINK can trap them. The only way to fix this is to rewrite the BIOS so that all displaying of characters is done through the BIOS jump vector.

3.0 The HASP Protocol—An Overview

A protocol is a set of specifications about the ways a device can communicate with another device. Protocols are responsible for defining such things as error checking and retransmission, identification, and device specification.

HASP is the protocol used by the KAYLINK package. Primarily associated with IBM equipment but readily available for many other systems, this protocol is usually used for remote job-entry and printing. Among the systems which use the HASP protocol in some form are JES and POWER. Remember that "HASP" refers to the protocol and not to the KAYLINK package, and phrases such as "HASP provides" refer to the protocol definition.

HASP has a number of features which make it a very useful protocol. For one thing, it uses synchronous communications, which makes it fast. All traits of synchronous communications, such as the fact that SYNC bytes must be sent at certain times to maintain synchronization, are handled by the protocol.

HASP uses a half-duplex line for communications, so data can only go in one direction at a time. However, because of the way that HASP breaks up data, the direction of data flow can switch rapidly enough to give the impression of sending and receiving data at the same time.

HASP allows multi-leaving, which permits several types of data to be sent in one burst of communication. KAYLINK supports receiving multi-leaved data, although very few host sites actually send this type of data. In addition, HASP allows data compression, which KAYLINK fully supports for both incoming and outgoing data.

3.1 The Signon and Signoff Cards

After basic communications have been established and each computer knows that the other exists, the remote will send a **signon card** to identify itself. This is an 80-column card image sent over the communications line along with special function codes. Usually the signon card takes the form

```
/*SIGNON      REMOTEnn pw1
```

where **REMOTE** is a required keyword which starts at column 16, **nn** is the number of the remote, and **pw1** is a password for security which starts at column 25. Sometimes a different signon card is used. JES systems may require an additional password at column 73. HASP on CDC CYBER systems requires a signon card which starts with **/*CONFIG.** Talk to the people at your mainframe site or data center if you aren't sure of the form of the signon card.

KAYLINK allows the signon card to be specified on the command line in two different ways. Say you wanted to run KAYLINK with a signon card which looked like

```
/*SIGNON     REMOTE37 PANJANDRUM
```

One way would be to spell out the entire signon card by typing

```
KAYLINK SO=/*SIGNON     REMOTE37 PANJANDRUM
```

This method works for all signon card formats. Since this particular signon card is a standard format, you can also type

```
KAYLINK RN=37,P1=PANJANDRUM
```

to do exactly the same thing. For more information about command line installation, please turn to Section 4.1.

The signon card also tells whether a remote is **transparent** or **non-transparent**. If the remote is transparent, it can send either transparent or non-transparent data. If it is non-transparent, it can only send non-transparent data. Information about whether a remote is transparent or non-transparent is included in the signon card in a way that is invisible to humans. You can change whether KAYLINK identifies itself as transparent or non-transparent with the **T** and **NT** arguments on the command line. For more information about command line installation, please turn to Section 4.1.

When a microcomputer running KAYLINK receives a signon card, it will realize that it is talking to a remote, and will behave as a host. Since KAYLINK has no use for the signon card, its contents will be ignored. KAYLINK will, however, determine whether the remote is transparent or non-transparent and will not allow transparent data to be sent if the remote is non-transparent.

When the session is over, the remote may send a **signoff card** to the host. Like the signon card, this is a card image. Unlike the signon card, this card has no special function codes associated with it. It is simply a single card sent as card reader data which behaves exactly as if a real punched card containing

/*SIGNOFF

were placed in the card reader. Because some HASP sites do not require a signoff card, KAYLINK allows you to specify on the command line whether to send a signoff card or not. There is no way to change the signoff card. If your system happens to require a different type of card to sign off, use the Send Disk File or Text Editor capabilities of KAYLINK to send one.

3.2 HASP Data Streams

A **data stream** in HASP may be considered a channel for sending and receiving a certain type of data. These data streams are usually associated with a certain device, from which their names are derived. There is a console data stream which goes in both directions, from the host to the remote and back again. This stream is usually used for data typed on the remote keyboard or displayed on the remote console screen. In addition to the console stream, both the host and the remote have seven input and seven output streams each.

The seven output streams for a remote are seven card reader streams, named Reader 1 through Reader 7. The seven input streams may be various combinations of printer and card punch streams, selected from Printer 1 through Printer 7 and Punch 1 through Punch 7. Similarly, a host has seven card reader input streams and seven printer/punch output streams. Printer and punch data only flow from the host to the remote, while reader data only flow from the remote to the host.

KAYLINK will always send printer, punch, or reader data as Printer 1, Punch 1, or Reader 1. Similarly, KAYLINK will treat all incoming data for a certain device, regardless of its number, exactly the same. For example, data received for Printer 1 and data received for Printer 4 are both treated simply as printer data.

Before data can be sent over a stream, that stream must be **open**. The console stream is always open, but all other streams must be specifically opened by commands defined by the HASP protocol. When a computer wants to send data over a specified stream, it will send a request to the other computer to open that stream. The other computer will either open that stream and return an appropriate message or ignore the request.

KAYLINK does not allow two different streams of the same type, for example Printer 2 and Printer 5, to be open at the same time. There are further restrictions on the streams that can be open at once. For example, printer and punch streams cannot be open at the same time.

Once a stream has been opened, HASP provides a way for a computer to signal whether it is ready to receive data for that stream. There is also a way to signal whether it is ready to receive **any** data. These signals are used by KAYLINK to control reception of data. When, for example, KAYLINK is receiving data which is being sent to a disk file, KAYLINK may transmit a signal to suspend reception of the data while it is actually writing to the disk.

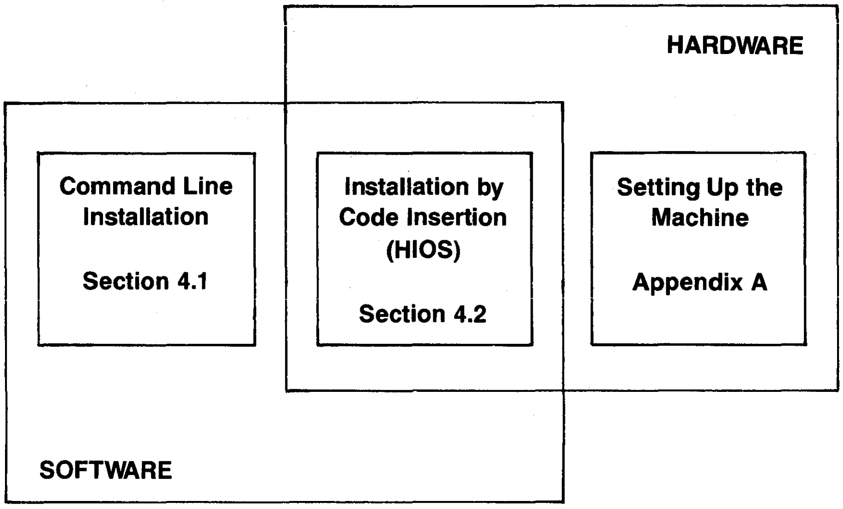


Figure 4.1 The Three Phases of Installing KAYLINK

4.0 Installing KAYLINK

The term **installation** is used to describe the entire process of taking a copy of KAYLINK from the distribution disk and transforming it into a program that runs on your machine and satisfies your requirements. The installation process includes modifying your machine to run KAYLINK as well as modifying KAYLINK to run on your machine.

Figure 4.1 on the opposite page shows the three phases of installing KAYLINK. The boxes labeled **HARDWARE** and **SOFTWARE** roughly indicate the difference between the three types of installation.

Command line installation is explained by Section 4.1. This type of installation is performed by specifying various arguments on the command line when loading and running KAYLINK. All of these arguments are concerned with software and are independent of hardware. In addition, most of the arguments have to do with requirements of the host computer or matters of taste. Any user of KAYLINK may do this type of installation.

Installation by code insertion is explained starting in Section 4.2. For this type of installation, a segment of code is written and inserted into KAYLINK. This code contains all the interfaces to the machine which are not handled by CP/M. Although this type of installation only requires modification of software, it requires intimate knowledge of the hardware as well as experience with assembly language.

Setting up the machine is explained in appendix A. This type of installation consists of all machine modifications which must be done to make the machine function properly with KAYLINK. Depending on the machine, this may be very easy, only involving changing jumpers, or difficult, involving partially rewiring the machine.

4.1 Command Line Installation

KAYLINK is usually loaded and run by typing

```
KAYLINK
```

on the keyboard. Although this is probably the most common way to run it, it is by no means the only way to do so. KAYLINK allows the command line to contain various arguments which specify things about the program.

Readers of the Users' Manual will remember that typing

KAYLINK HOST

loads and runs KAYLINK as a host site rather than as a remote site. This is an example of specifying an argument on a command line (in this case, **HOST** is the one and only argument).

In general, KAYLINK can take any number of arguments separated by commas. These arguments may take the form of a single word, such as **HOST** above, or an equivalence, which is a single word followed by an equal sign (=) which is in turn followed by a string of characters or a number. For example, the command line

KAYLINK LP=66,NP

will load and run KAYLINK with 66 lines per printer page and no password protection on the text editor. **LP=66** is an equivalence, while **NP** is a single word.

Although it is useful to be able to specify arguments for running KAYLINK, typing a long line of arguments every time you want to run KAYLINK can become tiring. Also, it is sometimes necessary to set up a copy of KAYLINK with certain options already changed. For this reason, a special argument is provided. This argument takes the form **O=filename**, where **filename** is replaced by a standard CP/M file name. When this argument is specified on the command line, KAYLINK will save a new copy of KAYLINK with the new setup on file **filename** with secondary name **.COM** and exit immediately. For example, typing

KAYLINK S-,O=KAYLINK2

will save a new copy of KAYLINK on the file **KAYLINK2.COM**. This copy will be the same as the original, except that the new copy, when it is itself loaded and run, will not send a signoff card at the end of a session (**S-** argument).

The following is a list of all legal command line arguments and their meanings. If an argument is described as being the "default," that means that the original copy of KAYLINK on your distribution disk is installed that way.

Argument	Function
HOST	Run KAYLINK as a host. This argument must be the first if more than one is used. It does not work in conjunction with the "**O=**" parameter; thus KAYLINK cannot permanently be installed to be a host.
O = filename	If this argument is present; KAYLINK with the new changed setup will be saved on file **filename** with type **.COM**. If this argument is absent, KAYLINK will be run immediately with the new setup.
I = filename	If this argument is present, KAYLINK will load **filename** as a HIOS **COM** file. See section 4.7 of the Technical Manual for more information about the HIOS.
LP = nn	If this argument is included with **nn** replaced by a decimal number, KAYLINK will use that number as the number of lines per printed page, including all space at the top and bottom of each page. The default is 66 lines per page.
P +	This argument indicates that a form feed be sent to the printer whenever it comes within four lines of the bottom of the page. For example, if there are 66 lines per page, KAYLINK will print 62 lines and then automatically execute a form feed.
P-	This argument disables all automatic form feeds. (Default)
ML = n	If **n** is replaced by a single-digit number, that number will be the minimum length of the Text Editor Password. A length of 0 will allow a null password to be entered when the RETURN key is pressed at the password page. The default minimum length is zero.
NP	This argument specifies that no password is required to use the Text Editor. Do not use in conjunction with **ML = n**.
T	This argument causes KAYLINK to be a transparent remote station. This allows the host to send transparent and non-transparent data to the remote. KAYLINK is by default a transparent station.

NT	This argument sets up KAYLINK as a non-transparent remote station. This prevents the host from sending any transparent data. The "**T**" and "**NT**" options have no effect on KAYLINK operating as a host.
RL	When KAYLINK receives a record to be sent to the console, it must do a carriage return-line feed combination either before or after the record is displayed as a line of text. This argument causes the carriage to return **before** the line and may be used to allow systems to display prompts on the same line they ask for input.
LR	This argument causes the carriage return to occur **after** the line. This is the default.
S +	This argument causes KAYLINK to send a **/*SIGNOFF** card as card reader data before exiting to the operating system when option "7" is selected from the Main Menu. For more information on the signoff card, see Section 3.1 of the Technical Manual. This is the default.
S-	This argument specifies than no signoff card be sent.
SO = string	The signon card is sent over the communications line when KAYLINK is loaded as a remote. This argument changes the signon card to the characters in **string**. If you want to have commas in the signon card, use CTRL-A to represent each comma. For more information on the signon card, turn to Section 3.1 of the Technical Manual.
RN = nn	This argument sets up a standard signon card with remote number **nn**. Since this option builds a signon card, it should not be used at the same time as **SO = string**.
P1 = string	Sets the standard HASP password to **string**. This argument is used in conjunction with **RN = nn** to produce a standard signon card.
P2 = string	Sets the alternate password required by some sites to **string**. This is used in conjunction with **P1 = string**.
BS = nnn	Sets the transmitted block size to nnn. The default is 512 bytes, the maximum is 800 bytes (decimal). The minimum is 160.

M +	Causes the "More Data" question (see the Users' Manual) to be asked whenever card reader data is sent. This is the default.
M-	This causes KAYLINK to not ask the "More Data" question. KAYLINK will assume that there will be more data.

Examples

```
KAYLINK O = KAYLIN80,LP = 80,SO = /*SIGNON
```

will produce a new copy of KAYLINK with 80 lines per page and with the specified signon card. KAYLINK does not run, but a new copy of KAYLINK is written onto file **KAYLIN80.COM**.

```
KAYLINK LP = 80
```

will run KAYLINK with 80 lines per page. No new copy of KAYLINK will be saved.

```
KAYLINK O = KAYLIN37,RN = 37,P1 = WACKAWACKA
```

will produce a new copy of KAYLINK, called **KAYLIN37.COM**, with a standard remote signon card for remote number 37 with password **WACKAWACKA.**

```
KAYLINK O = KAYLINNP,S + ,NP
```

will produce a new copy of KAYLINK, called **KAYLINNP.COM**, with a signoff card and no password for the Text Editor.

If you do not include an argument to change a particular option, the option will remain unchanged. Therefore, the sequence

```
KAYLINK O = KAYLINK,SO = /*SIGNON,S +
KAYLINK O = KAYLINK,LP = 88,ML = 4
```

is effectively equivalent to this sequence

```
KAYLINK O = KAYLINK,SO = /*SIGNON,S + ,LP = 88,ML = 4
```

Notice that the output name is the same as the program name in this example. KAYLINK will write the new copy over the old one without creating an intermediate file. If you try this, make sure you have a backup copy of KAYLINK under another name or on another disk.

This technique of breaking up changes into more than one step can be used when all desired options cannot fit on one line.

4.2 Machine Characteristics and the HIOS

The KAYLINK program uses many features of your machine. Although these features are found on many machines, the ways of using them tend to differ from machine to machine. Synchronous communications hardware, cursor addressing, and interrupts are all things which KAYLINK either must or can use.

KAYLINK is a large program, so its memory requirements are large. Standard KAYLINK requires 48K of available memory (TPA) in a CP/M system. Machines which use large PROM monitors and display memory located in the standard 64K address space may not be able to run KAYLINK.

KAYLINK must be able to use a synchronous port. If a microcomputer running KAYLINK is to be connected to a synchronous modem, the modem must be able to supply the clock signals to the port controller. Many machines can easily be made to receive the clock signals. For example, the Xerox 820 and Xerox 820-II only require changing two jumpers inside the cabinet. On some machines, even some that have synchronous ports, there is no way to allow the modem to supply the clock pulses. These machines may require rewiring to work at all. Consult appendix A for information about your machine.

KAYLINK requires an 80-column screen with at least 24 lines which has direct cursor addressing and clear screen features and is fast enough not to be annoying. KAYLINK also requires a keyboard with a reasonably full ASCII set including control character generation and two special keys called ESC and HELP. These two keys may be any two keys on the keyboard not used for text entry, but to maintain consistency with the manual these keys should be as similar to ESC and HELP as possible (i.e. HOME for HELP). It is most convenient for the user if these keys can be operated by a single key press (no SHIFT or CTRL).

Some machines have features which, although not strictly necessary, can improve the performance of KAYLINK. Both port (modem) and timer (CTC) interrupts, described in Sections 4.4 and 4.6, can multiply the speed of KAYLINK by a factor of four. A feature to erase to the end of the line can speed up screen replots. Special arrow keys can simplify use of the Text Editor.

All these machine-dependent features are accessed by a special section of machine code. This section is called the HIOS, or **H**ASTE **I**nput-**O**utput **S**ystem. KAYLINK is shipped for your machine with the proper HIOS already installed.

If you have special things you want to do with your machine, you must write your own HIOS, or modify the HIOS supplied. Sample HIOS source code files are provided on your disk to help you with the process. Chapter 5 explains how to build a HIOS from scratch.

The HIOS is at most 500H bytes of machine code starting at address 0200H. (Look on the next page to see the beginning of a sample HIOS.) Before address 0200H are six jump vectors which should not be changed by the user but are called by routines in the HIOS. These are described in Section 4.3. Starting at address 0201H is a series of several jump vectors which jump to routines that perform machine-dependent functions and must be changed for different machines. These are described in Section 4.4. After the jump vectors is a series of flags and strings which provide information for KAYLINK. These are described in Section 4.5.

The rest of this chapter is a **guide** for building a HIOS. Chapter 5 gives information on how to build a HIOS completely from scratch. If you have any questions about the HIOS which this manual does not answer, examine the sample HIOSes included on the distribution disk. All have secondary file name **.ASM**.

```
          ORG  200H          ;HIOS EXTENDS TO 6FFH (500H BYTES)

ETOA      EQU  $-18          ;CONVERT EBCDIC TO ASCII
ATOE      EQU  $-15          ;CONVERT ASCII TO EBCDIC
WSTC      EQU  $-12          ;WRITE STRING TO CONSOLE ROUTINE
INTMIN    EQU  $-9           ;MODEM (PORT) INPUT INTERRUPT ROUTINE
INTMOUT   EQU  $-6           ;MODEM (PORT) OUTPUT INTERRUPT ROUTINE
INTIMER   EQU  $-3           ;TIMER INTERRUPT ROUTINE

VERS:     DB   13            ;FOR KAYLINK VERSION 1.3x
HINIT:    JMP  aHINIT        ;ALL INSTALLABLE INITIALIZATION
EXIT:     JMP  aEXIT         ;ANY CLEANING UP BEFORE EXIT
EMINT:    JMP  aEMINT        ;EMULATE INTERRUPTS
ENINT:    JMP  aENINT        ;ENABLE INTERRUPTS
DISINT:   JMP  aDISINT       ;DISABLE INTERRUPTS
ENTXE:    JMP  aENTXE        ;ENABLE MODEM (PORT) TRANSMITTER
ENRCV:    JMP  aENRCV        ;ENABLE MODEM (PORT) RECEIVER
DISTXE:   JMP  aDISTXE       ;DISABLE MODEM (PORT) TRANSMITTER
DISRCV:   JMP  aDISRCV       ;DISABLE MODEM (PORT) RECEIVER
MODOUT:   JMP  aMODOUT       ;SEND BYTE TO MODEM (PORT)
MODIN:    JMP  aMODIN        ;GET BYTE FROM MODEM (PORT)
TIMEND:   JMP  aTIMEND       ;END OF TIMER INTERRUPT ROUTINE
PRSTAT:   JMP  aPRSTAT       ;RETURN PRINTER STATUS
PROUT:    JMP  aPROUT        ;SEND BYTE TO PRINTER
RDRST:    JMP  aRDRST        ;RETURN PHYSICAL CARD READER STATUS
RDRIN:    JMP  aRDRIN        ;GET BYTE FROM CARD READER
AUXIST:   JMP  aAUXIST       ;RESERVED FOR FUTURE EXPANSION
AUXIN:    JMP  aAUXIN        ;    "       "      "       "
AUXOST:   JMP  aAUXOST       ;    "       "      "       "
AUXOUT:   JMP  aAUXOUT       ;    "       "      "       "
CONIST:   JMP  aCONIST       ;RETURN CONSOLE INPUT STATUS
CONIN:    JMP  aCONIN        ;GET BYTE FROM CONSOLE (KEYBOARD)
CONOST:   JMP  aCONOST       ;RETURN CONSOLE OUTPUT STATUS
CONOUT:   JMP  aCONOUT       ;SEND BYTE TO CONSOLE
POSCUR:   JMP  aPOSCUR       ;POSITION CURSOR
CLRSCR:   JMP  aCLRSCR       ;CLEAR SCREEN AND HOME CURSOR
ENRV:     JMP  aENRV         ;ENABLE REVERSE VIDEO
DISRV:    JMP  aDISRV        ;DISABLE REVERSE VIDEO
ETEOLN:   JMP  aETEOLN       ;ERASE TO END OF LINE

ELFLAG:   DB   0FFH          ;ERASE LINE INSTALLED FLAG
INTFLG:   DB   00H           ;INTERRUPT FLAG
TSPRER:   DB   100           ;TENTHS OF SECONDS UNTIL PRINTER ERROR

MAXROW:   DB   24            ;NUMBER OF LINES ON CRT DISPLAY

UPAROW:   DB   'A'-40H       ;ALTERNATE CURSOR CONTROLS FOR TEXT EDITOR
DNAROW:   DB   'B'-40H
LFAROW:   DB   'D'-40H
RTAROW:   DB   'C'-40H

ESCSTR:   DB   'ESC',0,0,0,0,0
HLPSTR:   DB   'HELP',0,0,0,0
RETSTR:   DB   'RETURN',0,0
DELSTR:   DB   'DEL',0,0,0,0,0,0,0
ARRSTR:   DB   'Arrow keys',0
```

Figure 4.2 A Skeletal HIOS

4.3 Accessible KAYLINK Routines

There are six jump vectors before the beginning of the HIOS which may be called by routines within the HIOS. Three of these jump to routines which control various interrupt conditions. The builder of the HIOS must call one of these routines (using an ordinary 8080 **CALL** instruction) when an interrupt occurs. The routines are as follows:

ETOA Convert EBCDIC to ASCII

Entry registers: A contains EBCDIC byte to convert to ASCII.
Return registers: A will contain ASCII equivalent of EBCDIC character.
Saved registers: BC, DE, HL

The jump vector to this routine is found at 0200H-18, or 01EEH. It converts a character in the EBCDIC character set to the ASCII character set. Because KAYLINK assumes that all HIOS routines which use characters work in ASCII, this routine is only provided for special purposes, such as interfacing with an external device that uses the EBCDIC character set.

Because **ETOA** uses the conversion table described in Appendix B, it will return a 0A0H character (ASCII space with the MSB set) when an EBCDIC character has no ASCII equivalent.

ATOE Convert ASCII to EBCDIC

Entry registers: A contains ASCII byte to convert to EBCDIC.
Return registers: A will contain EBCDIC equivalent of ASCII character.
Saved registers: BC, DE, HL

The jump vector to this routine is found at 0200H-15, or 01F1H. It converts a character in the ASCII character set to the EBCDIC character set. Because KAYLINK assumes that all HIOS routines which use characters work in ASCII, this routine is only provided for special purposes, such as interfacing with an external device that uses the EBCDIC character set.

ATOE uses the conversion table described in Appendix B.

WSTC Write String To Console

Entry registers: DE points to 0-terminated string to be sent to the screen.
Return registers: None
Saved registers: BC, HL

The jump vector to this routine is found at 0200H-12, or 01F4H. It sends a 0-terminated string at DE to the console screen. This should be used by various display functions to send a sequence of characters to the terminal. If your machine does not use interrupts, you should always use this routine whenever you want your HIOS routines to output characters. For example, **aCLRSCR** may output an escape sequence to clear the screen.

INTMIN Modem (Port) Input Interrupt Handler

Entry registers: None
Return registers: None
Saved registers: None

The jump vector to this routine is found at 0200H-9, or 01F7H. It should be called when an interrupt (or interrupt emulator) routine in the HIOS determines that a character is ready from the synchronous port. When INTMIN is called, it will get the character and do everything that is necessary with it. **INTMIN** does not check status.

INTMOUT Modem (Port) Output Interrupt Handler

Entry registers: None
Return registers: None
Saved registers: None

The jump vector to this routine is found at 0200H-6, or 01FAH. It should be called when an interrupt (or interrupt emulator) routine in the HIOS determines that the synchronous port is ready to receive an output character. The routine will send a character to the port if that is appropriate. **INTMOUT** does not check status.

INTIMER Timer Interrupt Handler

Entry registers: None
Return registers: None
Saved registers: None

The jump vector to this routine is found at 0200H-3, or 01FDH. It should be called every tenth of a second by the timer interrupt or interrupt emulator routine.

4.4 HIOS Installable Routines

The first byte in the HIOS is a **version number**; for KAYLINK version

1.3x it should be 13 (decimal). This version number must be correct or KAYLINK will reject the HIOS.

This section describes the jump vectors in the HIOS for which the installer is responsible. Each is a 3-byte jump vector containing a single 8080 **JMP** instruction and an address.

Your skeletal HIOS, which is on file **HIOS.ASM**, contains a series of jump vectors which jump to a series of routines. The labels of the routines are exactly the same as the labels of the jump vectors preceded by an "**a**" character. When naming your routines, you must, of course, use the "**a**". However, when you call any of your routines from other routines, call the label of the corresponding jump vector, not the label of the routine itself.

Interrupt routines must save all registers, including flags, and should use the stack as little as possible. All other routines have several stack levels at their disposal but need not save any registers.

The following is a list of routines in the HIOS:

aHINIT KAYLINK Initialization

Entry registers: None
Return registers: None

This routine is probably the most complicated and important routine in the entire HIOS, especially for interrupt versions of KAYLINK. It performs all functions which must be done when KAYLINK is loaded and run. These include such things as initializing the interrupt controller (if any), the serial port, etc.

aHINIT must set up the synchronous port for eight bits, no parity, two byte synchronous mode. It must set the two sync bytes to 32H and 32H. It should also set up the I/O chip to receive external transmit and receive clocks, if this can be done in software. Finally, it should set the DTR (Data Terminal Ready) line on the synchronous port. This routine **must not** enable either the transmitter or the receiver (separate routines exist for these functions).

aHINIT should also set up the terminal characteristics to 80-column mode with wrap-around at the end of the line and scroll at the bottom of the screen. If any code needs to be patched outside KAYLINK, **aHINIT** should do the patching. For example, the Xerox HIOS fixes errors in the monitor and disables the CTRL-ESC key.

aHINIT has complete responsibility to set up and maintain interrupt routines which themselves call routines listed in Section 4.3 to service interrupts. For more information about interrupt routines, see Section 4.6 and **aEMINT** later in this section. Chapter 5 gives examples of typical interrupt applications. **aHINIT** should not enable interrupts, even if it disables interrupts. KAYLINK will enable the interrupts when they are needed.

aEXIT Exit Processing

Entry registers: None
Return registers: None

This routine performs any and all cleanup before KAYLINK exits normally. Before this routine is called, all files will have been closed, and communications will have been terminated.

This routine should, at the very least, disable all interrupts that were set up by KAYLINK and drop the DTR (Data Terminal Ready) line to the modem. If **aHINIT** patched any code external to KAYLINK, **aEXIT** should put it back the way it was.

aEMINT Emulate Interrupts

Entry registers: None
Return registers: None

This routine is used to emulate interrupts for machines which do not have them. It should perform the functions of all interrupts which are not available. KAYLINK calls this routine, on the average, 2400 times a second on a 1 MHZ microprocessor. See Section 4.6 for more information on the interrupt emulator and interrupts in general. See Chapter 5 for a sample interrupt emulator.

If you use real interrupts, make this routine a single **RET** instruction.

aENINT Enable Interrupts

Entry registers: None
Return registers: None

This one can be just a return in a non-interrupt system. In an interrupt system it should re-enable only those interrupts which KAYLINK uses. It should not affect any other interrupts.

aDISINT Disable Interrupts

Entry registers: None
Return registers: None

This routine disables all interrupts that are being used by KAYLINK. Any interrupts that are used by the operating system should be left unchanged. For example, in a multiprocessor or multi-user system there may be system interrupts that are used to switch between users. These interrupts are not used by KAYLINK, and should not be changed by DISINT. In a non-interrupt system, this routine can be just a return.

aENTXE Enable Transmitter

Entry registers: None
Return registers: None

This routine enables the synchronous port transmitter. It should reset error flags, enable DTR (Data Terminal Ready), RTS (Request To Send), and TXEN (Transmitter Enable), and enable transmitter interrupts, if any.

aENRCV Enable Receiver

Entry registers: None
Return registers: None

This routine enables the synchronous port receiver. It should reset error flags, turn on DTR and REN (Receiver Enable), enable receiver interrupts, and cause the I/O chip to enter hunt mode.

The transmitter and the receiver will never both be enabled at the same time. Therefore, **aENRCV** and **aENTXE** do not have to remember what bits have been sent to the I/O chip.

aDISTXE Disable Transmitter

Entry registers: None
Return registers: None

This routine disables the synchronous port transmitter leaving only DTR on and disables transmitter interrupts, if any.

aDISRCV Disable Receiver

Entry registers: None
Return registers: None

This routine disables the synchronous port receiver leaving only DTR on and disables receiver interrupts, if any.

aMODOUT Modem (Port) Output

Entry registers: A contains byte to send to synchronous port.
Return registers: None

This routine sends the byte in register A to the synchronous port. This routine is called by **INTMOUT** which is called only when your interrupt or interrupt emulator routines have already tested status.

aMODIN Modem (Port) Input

Entry registers: None
Return registers: A will contain byte from the synchronous port.

This routine inputs a byte from the synchronous port and returns it in register A. This routine is called by **INTMIN** which is called only when your interrupt or interrupt emulator routines have already tested status.

aTIMEND Timer Routine End

Entry registers: None
Return registers: None

When the KAYLINK **INTIMER** routine (Section 4.3) is called, it will usually return normally to your interrupt routine. Occasionally, in cases such as printer errors, it will take over the stack to recover from errors. For this reason, KAYLINK needs a routine to end the timer interrupt that does not change the stack pointer. **aTIMEND** should signal end of interrupt by whatever method the machine uses **without** changing the stack pointer and then return. On standard Z-80 systems, for example, this routine may be an **EI** instruction followed by a **RETI** instruction. On other systems, this routine may have to send instructions to the CTC timer chip or interrupt controller chip to signal the end of interrupt and then return.

This routine may be an entry point into your timer routine after the stack has been restored. **aTIMEND** may use the stack as long as it exits with the stack pointer unchanged. See Chapter 5 for a sample timer routine.

aPRSTAT Printer Status

Entry registers: None
Return registers: A will contain the status of the printer.

This routine returns the status of the printer. Register A will contain 0FFH if the printer is ready to receive a character or 0 if it is not. This routine may simply contain a call to the appropriate BIOS function. NOTE: If this routine does not return correct printer status, printer error routines will only work properly on an interrupt version of KAYLINK.

aPROUT Printer Output

Entry registers: A contains the byte for the printer.
D contains the current line number.
E contains the current column number.
Return registers: None

This routine sends the character in register A to the printer. When this routine is called, printer status will already have been tested. Therefore, **aPROUT** may or may not test the printer status--it does not matter.

The current line and column number (in registers D and E) are for the line and column where the last character was printed.

PROUT must perform the following functions for the following control codes:

Control Code	**Function**
0DH (CR)	Return the carriage without a line feed.
0AH (LF)	Do a line feed without a carriage return.
0CH (FF)	Go to the upper left-hand corner of the next page. (i.e. do a form-feed and carriage-return)

aRDRST Return Physical Card Reader Status

Entry Registers: None
Return Registers: A and CY will contain status information

This routine, along with its sister routine **aRDRIN**, provides a way of using a physical card reader on a machine running KAYLINK. Operation of the card reader (explained in Section 7.2 of the Users' Manual) essentially requires no interaction with KAYLINK; the user simply puts his cards in the hopper and presses the start button. KAYLINK constantly polls **aRDRST** and when a character is available opens the Card Reader 1 stream and begins sending cards, using **aRDRST** and **RDRIN** to get characters.

The status this routine returns is much more complicated than other routines. Three bits must be set or reset, depending on conditions:

Bit 0 of A register--Set indicates a character is available.
Bit 7 of A register--Set indicates an error was found reading cards.
Carry Flag--Set indicates card reader stopped or hopper empty.

Bit 0 of the A register is simple enough: set this bit if and only if a character is available. If this bit is set, no others should be. In other words, all valid characters should be made available to KAYLINK before an error or stopped condition is reported.

Bit 7 of the A register is a little trickier. When an error occurs, **aRDRST** should stop the card reader and, in any event, **must** set the Carry Flag indicating that the hopper is empty. This implies that an error cannot occur in the middle of a card, or, if one does, it must be reported before any characters on the card are made available to KAYLINK. When KAYLINK recognizes an error, it will display the message **CARD READER PROBLEM** in the Status Window. The user must then fix the problem and punch the START button (or its equivalent) on the card reader.

The Carry Bit is sent if the card reader is stopped or empty. When KAYLINK sees this and there has been no EOF card, it will display **READY TO SEND MORE DATA** in the Status Window. The user then has the option of placing more cards in the card reader and pressing the START button or sending a disk file or the Text Screen as reader data. If there is no more data to send, the user can indicate this by pressing option **"0-NO MORE DATA TO SEND"** on the Text Screen or Send File menu.

aRDRIN   Return Reader Input Character

Entry registers: None
Return registers: A will contain ASCII input character

This routine returns the next character from the physical card reader. In practice, **aRDRIN** must work very closely with **aRDRST** to return the proper codes at the proper times. **aRDRST** will probably read an entire card into a buffer, and **aRDRIN** will simply pick characters out of the internal buffer.

aRDRIN must return ASCII characters. These characters will be interpreted as **non-transparent** data and will be converted to EBCDIC before being sent. If your card reader returns EBCDIC characters, you must convert them to ASCII using the **ETOA** routine described in section 4.3.

There is a special code which **aRDRIN** must return when all cards for a job have been sent. This is the standard CP/M EOF character, CTRL-Z (1AH). **aRDRIN** must recognize the end of the job, for example by a 6-7-8-9 card, and return CTRL-Z. KAYLINK will then close the Card Reader stream. **aRDRIN** should also flush the buffer or otherwise prevent further characters from being sent, because KAYLINK will open the Card Reader stream again if more characters are available.

aAUXIST	**Auxiliary Device Input Status**
aAUXIN	**Auxiliary Device Input**
aAUXOST	**Auxiliary Device Output Status**
aAUXOUT	**Auxiliary Device Output**

These four routines may be used in future versions of KAYLINK but only require place holders for now.

aCONIST Console Input Status

Entry registers: None
Return registers: A will contain console input status.

This routine returns A = 0FFH if the console has a keyboard character waiting or A = 0 if there is no character waiting.

The best way to implement this routine is to use BDOS call 6. First define a single-byte location--call it **CHARIN**--to be a temporary location for the character. At first, **CHARIN** should contain a 0 byte indicating no character. First check to see if **CHARIN** is nonzero and return A = 0FFH if so. If not, load the **C** register with 6, load the **E** register with 0FFH, and execute a **CALL 0005H.** BDOS will return with A=0 if no character is available or A= the character. If A=0, return. If A is not zero, store A in **CHARIN,** set A=0FFH, and return. See section 5.1 for more information on this.

aCONIN Console Input

Entry registers: None
Return registers: A will contain byte from console.

This routine returns one character from the keyboard **without** displaying the character on the console. Since KAYLINK expects certain keycodes, this routine must return these codes for certain keys:

Key	**Returned Code**
ESC	1BH (Standard ESC)
HELP	1EH (Standard HOME)

All other key codes are standard.

If you used the method described in **aCONIST**, this routine could simply load the **A** register with the value in **CHARIN**, zero **CHARIN**, and return.

aCONIN must return only one character per keypress. This means that it is responsible for converting all terminal escape sequences and the like to single characters. This usually involves recognizing an ESC character, waiting a short time, calling **CONIST**, and determining whether an escape sequence was being sent or simply a single ESC. An example of how this may be done appears in Section 5.3.

aCONOST Console Output Status

Entry registers: None
Return registers: A will contain console output status.

This routine returns A = 0FFH if the console is ready to receive an output character or A = 0 if the console is not ready. This routine can speed up the non-interrupt version using an external terminal (as opposed to a memory-mapped display).

aCONOUT Console Output

Entry registers: A contains byte to be displayed.
Return registers: None

This routine sends one character to the console. The following is a list of actions **aCONOUT** must perform with certain control codes.

Control Code	**Function**
0DH (CR)	Return the cursor to the first column.
0AH (LF)	Line feed without returning cursor.

08H (BS)	Do a non-destructive back space.
07H (BELL)	Ring the bell, if any.

In addition, **aCONOUT** must wrap around to the beginning of the next line when at the end of the line and must scroll at the bottom of the screen. If your terminal must be set up to do these things, put the appropriate code in **aHINIT**.

aPOSCUR   Position Cursor

Entry registers: D contains row, E contains column.
Return registers: None

This routine positions the standard console cursor to row D, column E. Rows and columns start from zero at the upper left-hand corner of the screen.

aCLRSCR   Clear Screen

Entry registers: None
Return registers: None

This routine clears the console screen and homes the cursor to the upper left-hand corner.

aENRV   Enable Reverse Video

Entry registers: None
Return registers: None

This routine enables reverse video or whatever attribute mode is used. All characters subsequently sent to **CONOUT** will be displayed in reverse video.

aDISRV   Disable Reverse Video

Entry registers: None
Return registers: None

This routine disables reverse video. All subsequent characters will be displayed normally.

aETEOLN   Erase To End-Of-Line

Entry registers: None
Return registers: None

This routine erases all characters from the cursor to the end of the line inclusive. This routine should never cause the screen to scroll. KAYLINK contains an internal routine for terminals which do not have this feature (see Section 4.5).

4.5 HIOS Flags and Strings

After the table of jump vectors are several labeled **DB** statements which are described here.

ELFLAG: DB 0FFH

This is a single byte which indicates whether or not a function which erases to the end-of-line is available (see the description of **aETEOLN** above). If your terminal does not have this feature, put a 0 byte in this location, and make the **aETEOLN** routine simply return without doing anything. If your terminal does have this feature, put a 0FFH in this location and ensure that your **aETEOLN** routine performs this function using calls to **WSTC**.

INTFLG: DB 80H

If your machine does not use serial port interrupts, put a 0 byte in this location. If your machine uses port interrupts which may occur during disk operations, put an 81H byte in this location. If your machine uses port interrupts, but those interrupts may not occur during disk operations, put an 80H byte here.

The best situation is to have interrupts which may occur during disk operations, such as on an S-100 system with a DMA disk system. This provides the best speed, efficiency, and accuracy.

If you have interrupts which may not occur during disk operations, KAYLINK will wait for a particularly safe time to do a disk access. This has the effect of slowing down disk operations somewhat and occasionally requiring retransmission of a buffer over the communications line. It is assumed that the disk routines (and any routines which do bank switching) disable interrupts before and re-enable them after disk operations. Ensuring that this actually happens properly may require **aHINIT** to patch the disk routines.

If you don't have interrupts, communications might be limited to 4800 baud or so, depending on the speeds of various routines.

Consult Section 4.6 for more information on interrupt routines.

TSPRER: DB 100

This byte contains the number of tenths of a second until a printer error is reported. This time delay gives slow printers time to start up or do form feeds. It should be adjusted as needed for your printer. The delay may not be precise in non-interrupt version of KAYLINK.

If **TSPRER** is equal to 0 (zero), the printer can wait as long as it wishes, and KAYLINK will never report a printer problem. Use this special case with care, especially in systems in which PRSTAT is not correctly implemented.

MAXROW: DB 24

This is a single byte giving the number of **scrollable** rows or lines on the CRT display. This **must** be at least 24 but can be more, depending on the characteristics of your display. For example, although the Heath/Zenith H19/Z19 has 25 lines, the last one is not used in scrolling. Therefore, **MAX-ROW** for the Z19 is 24.

```
UPAROW:  DB   'A'-40H
DNAROW:  DB   'B'-40H
LFAROW:  DB   'D'-40H
RTAROW:  DB   'C'-40H
```

These four bytes give the characters returned for an alternate set of cursor keys for the Text Editor. Remember that CTRL-H, CTRL-J, CTRL-K, and CTRL-L **always** work, even if an entirely new set is specified here. Characters given here are those for the Xerox 820 or Xerox 820-II.

ESCSTR: DB 'ESC',0,0,0,0,0

This is a 0-terminated string giving the name of the ESC key. You may change this message if, for example, your ESC key uses the legend **ESCAPE**. The entire string, including at least one terminating zero, must be **exactly** eight characters long.

HLPSTR: DB 'HELP',0,0,0,0

This is a 0-terminated string which will be displayed as the name of the help key. This may be changed if, for example, you are using a Heath/Zenith H19/Z19 and want the BLUE function key to act as the help key. Including the terminating zero, there must be exactly eight characters.

ESC and HELP are used in the same line in places where the length of the displayed string is critical. Make sure that the sum of the printable characters in **ESCSTR** and **HLPSTR** is **at most** eleven characters. If there are more than eleven characters, some information will not be displayed on the screen.

```
RETSTR: DB     'RETURN',0,0
```

This string gives the name of the RETURN key. For different terminals, you might have to change this to **'ENTER'** or **'NEWLINE'**. Remember to terminate with a zero byte and pad the entire string out to eight characters.

```
DELSTR: DB     'DEL',0,0,0,0,0,0,0
```

This is the string for the DELETE key. Pad it out to ten characters including at least one terminating 0 byte.

```
ARRSTR: DB     'Arrow keys',0
```

This is the string which will be displayed on the Text Editor on-screen help to describe how to move the cursor. Users of different terminals might want to change this to **'CTRL-H,J,K,L'** or whatever is appropriate. The string should have at most thirteen printing characters and should be terminated by a zero.

4.6 Writing Interrupt Routines

KAYLINK can make use of two types of interrupts or "real-time" service. The first is a timer which goes off ten times a second and allows KAYLINK to control such things as HASP transmit and receive timers and printer errors. The second is an immediate interrupt which occurs when the synchronous port transmit buffer can accept a character to transmit (check the I/O chip TXR flag) or the receive buffer has a character to be read. How the interrupts are handled is entirely the responsibility of the installer.

The HIOS should contain interrupt handlers for both types of interrupts. The **aHINIT** routine (see Section 4.4) is responsible for setting up interrupts and ensuring that each type of interrupt results in a call to the appropriate routine. Interrupt routines may not change any registers at all, **including** flags. If they use any registers, they must first save the old values on the stack. To be safe, push as few values on the stack as possible before changing to a new interrupt stack. All other registers should be saved on the new interrupt stack.

Do not change the stack pointer using **DAD SP** before pushing PSW to the old stack. This will affect the carry flag and is guaranteed to do something terrible. If you are using a Z-80, use a **LD (nn),SP** instruction instead.

The **timer interrupt routine** is responsible for calling the **INTIMER** interrupt handler ten times a second. The **port interrupt routine** is responsible for determining whether port input or output needs to be serviced (check "READY" lines, not "BUFFER EMPTY" lines), and calling **INTMIN** or **INTMOUT** respectively. Those routines always know exactly what to do by themselves, so no parameters of any sort need to be passed.

All this is very good, but what if your machine has no interrupts or only one type of interrupt? KAYLINK provides an **interrupt emulator** to solve this problem. Simply provide a routine called **aEMINT**, accessed through jump vector **EMINT**. This routine should perform the activity of both the timer **and** the port interrupts. That is, it should count off the tenths of seconds and check both the input and output statuses of the port. If a character is ready, call **INTMIN**; if the transmitter is ready for a character, call **INTMOUT**. For timing, be aware that the routine will be called, on the average, about 2400 times a second if your CPU clock is running at 1 MHZ.

If your machine has no interrupts, use **aEMINT** to perform all interrupt operations and put a 0 byte in location **INTFLG**.

If your machine has a timer interrupt but no serial port interrupts, you have two choices. If your timer interrupts are faster than the emulator, you can simply use the timer interrupt to handle all interrupt jobs. If they are not very fast, use the timer interrupt for timing, but use the interrupt emulator for handling the port. In any case, put a 0 byte in location **INTFLG**.

If your machine has a port interrupt but no timer interrupt, use the port interrupt as an interrupt and use the interrupt emulator for timing. If disk operations may be interrupted, put an 81H byte in location **INTFLG**. If disk operations may not be interrupted, put an 80H byte there.

If your machine has interrupts, **by all means** use them to their fullest. The interrupt emulator works, but can never match the speed and efficiency of real interrupts. Using real interrupts can speed up KAYLINK by a factor of four.

The operation of the interrupt emulator depends on the speed of console output. If you use the emulator, you should do the following things:

* Make sure that routine **aCONOST** returns correct console output status, if possible.

* If you are using a CPU-driven memory-mapped display, make sure that routine **aCONOUT** is as fast as possible.

* Make sure that all other status routines return correct status.

* Make all routines which do output (**aENRV**, **aPOSCUR**, etc.) use the **WSTC** routine for their output.

* Make the "interrupt" routine as short, fast, and efficient as possible.

4.7 Patching in the HIOS

Once you have written your HIOS, use the following procedure to insert the HIOS into your copy of KAYLINK. You need one disk which contains KAYLINK, the HIOS, an absolute assembler, and the CP/M **LOAD** utility.

IMPORTANT! DO NOT USE YOUR DISTRIBUTION DISK! Section 3.1 of the Users' Manual explains how to make a copy of KAYLINK on one of your disks.

(1) Assemble the HIOS with an assembler which produces a standard absolute **HEX** file. For example, if your HIOS is called **HIOS.ASM** and your assembler is the standard CP/M assembler **ASM**, type the following:

```
ASM HIOS
```

(2) Convert the **HEX** file into a **COM** file by typing the following:

```
LOAD HIOS
```

(3) Type

```
KAYLINK I = HIOS,O = KAYLINKT
```

This will load the HIOS into KAYLINK and write the new KAYLINK to file **KAYLINKT.COM**

(4) When you are satisfied that your new KAYLINK on file **KAYLINKT.COM** works properly, you can rename it to **KAYLINK.COM**. To do this using standard CP/M, type the following two commands (REMEMBER--you should **not** be using the distribution disk):

```
ERA KAYLINK.COM
REN KAYLINK.COM = KAYLINKT.COM
```

You will probably want to use additional command line installation after you are convinced that your HIOS works. Section 4.1 describes command line installation.

```
          ORG  200H         ;HIOS EXTENDS TO 6FFH (500H BYTES)

ETOA      EQU  $-18         ;CONVERT EBCDIC TO ASCII
ATOE      EQU  $-15         ;CONVERT ASCII TO EBCDIC
WSTC      EQU  $-12         ;WRITE STRING TO CONSOLE ROUTINE
INTMIN    EQU  $-9          ;MODEM (PORT) INPUT INTERRUPT ROUTINE
INTMOUT   EQU  $-6          ;MODEM (PORT) OUTPUT INTERRUPT ROUTINE
INTIMER   EQU  $-3          ;TIMER INTERRUPT ROUTINE

VERS:     DB   13           ;FOR KAYLINK VERSION 1.3x
HINIT:    JMP  aHINIT       ;ALL INSTALLABLE INITIALIZATION
EXIT:     JMP  aEXIT        ;ANY CLEANING UP BEFORE EXIT
EMINT:    JMP  aEMINT       ;EMULATE INTERRUPTS
ENINT:    JMP  aENINT       ;ENABLE INTERRUPTS
DISINT:   JMP  aDISINT      ;DISABLE INTERRUPTS
ENTXE:    JMP  aENTXE       ;ENABLE MODEM (PORT) TRANSMITTER
ENRCV:    JMP  aENRCV       ;ENABLE MODEM (PORT) RECEIVER
DISTXE:   JMP  aDISTXE      ;DISABLE MODEM (PORT) TRANSMITTER
DISRCV:   JMP  aDISRCV      ;DISABLE MODEM (PORT) RECEIVER
MODOUT:   JMP  aMODOUT      ;SEND BYTE TO MODEM (PORT)
MODIN:    JMP  aMODIN       ;GET BYTE FROM MODEM (PORT)
TIMEND:   JMP  aTIMEND      ;END OF TIMER INTERRUPT ROUTINE
PRSTAT:   JMP  aPRSTAT      ;RETURN PRINTER STATUS
PROUT:    JMP  aPROUT       ;SEND BYTE TO PRINTER
RDRST:    JMP  aRDRST       ;RETURN PHYSICAL CARD READER STATUS
RDRIN:    JMP  aRDRIN       ;GET BYTE FROM CARD READER
AUXIST:   JMP  aAUXIST      ;RESERVED FOR FUTURE EXPANSION
AUXIN:    JMP  aAUXIN       ;    ''      ''      ''       ''
AUXOST:   JMP  aAUXOST      ;    ''      ''      ''       ''
AUXOUT:   JMP  aAUXOUT      ;    ''      ''      ''       ''
CONIST:   JMP  aCONIST      ;RETURN CONSOLE INPUT STATUS
CONIN:    JMP  aCONIN       ;GET BYTE FROM CONSOLE (KEYBOARD)
CONOST:   JMP  aCONOST      ;RETURN CONSOLE OUTPUT STATUS
CONOUT:   JMP  aCONOUT      ;SEND BYTE TO CONSOLE
POSCUR:   JMP  aPOSCUR      ;POSITION CURSOR
CLRSCR:   JMP  aCLRSCR      ;CLEAR SCREEN AND HOME CURSOR
ENRV:     JMP  aENRV        ;ENABLE REVERSE VIDEO
DISRV:    JMP  aDISRV       ;DISABLE REVERSE VIDEO
ETEOLN:   JMP  aETEOLN      ;ERASE TO END OF LINE

ELFLAG:   DB   0FFH         ;ERASE LINE INSTALLED FLAG
INTFLG:   DB   00H          ;INTERRUPT FLAG
TSPRER:   DB   100          ;TENTHS OF SECONDS UNTIL PRINTER ERROR

MAXROW:   DB   24           ;NUMBER OF LINES ON CRT DISPLAY

UPAROW:   DB   'A'-40H      ;ALTERNATE CURSOR CONTROLS FOR TEXT EDITOR
DNAROW:   DB   'B'-40H
LFAROW:   DB   'D'-40H
RTAROW:   DB   'C'-40H

ESCSTR:   DB   'ESC',0,0,0,0,0
HLPSTR:   DB   'HELP',0,0,0,0
RETSTR:   DB   'RETURN',0,0
DELSTR:   DB   'DEL',0,0,0,0,0,0,0
ARRSTR:   DB   'Arrow keys',0
```

Figure 5.1 A Skeletal HIOS

5.0 Building a HOIS from Scratch

Chapter 4 explained the entire process of installing KAYLINK and described the HIOS. This chapter gives a step-by-step procedure for building a HIOS from scratch.

DO NOT DO ANY HIOS DEVELOPMENT ON YOUR DISTRIBUTION DISK! Make a copy of the entire disk or all files you will need.

Because the examples are presented separately, for the sake of completeness some different routines may contain some of the same statements. For example, several routines may contain the same **EQU**ates. Most assemblers will complain loudly if all these routines are simply typed in without regard for repetition. When putting these routines together, put one copy of each **EQU**ate near the beginning of the HIOS.

5.1 Standard Routines

To build a complete HIOS, first build a partial HIOS which will not communicate but at least will load and run. This section explains how to build such a basic HIOS. After this basic HIOS is running it can be modified to do communication, take advantage of interrupts, and eventually become a complete HIOS. Sections 5.2 through 5.5 explain how to do this.

Figure 5.1 on the opposite page is the top part of a HIOS. Every HIOS included on the distribution disk contains this code, sometimes slightly modified.

At the top of figure 5.1 is an **ORG** statement followed by six addresses, **ETOA**, **ATOE**, **WSTC**, **INTMIN**, **INTMOUT**, and **INTIMER**. **Do not change any of these addresses!** They are the addresses of jump vectors to routines in KAYLINK which are called by routines within the HIOS. **WSTC** is used by routines described in this section; the other five are used later. Also do not change the version number.

Next is a series of jump vectors to HIOS routines. All the routines with names beginning with a lower-case "**a**" must be written by the installer. Since the jump vectors **themselves** should not be changed, let's skip them for a while.

At the bottom are several flags and strings (all **DB** statements). Change these first.

```
ELFLAG: DB  0      ;ERASE LINE INSTALLED FLAG
                   ;00H MEANS TERMINAL HAS NO ERASE TO END-OF-LINE
                   ;0FFH MEANS aETEOLN ERASES TO END-OF-LINE
```

Make the **ELFLAG** byte zero. KAYLINK will handle all erases to the end of the line with its own internal routine. Later on you can change it to take advantage of your terminal's features.

```
INTFLG: DB  0      ;INTERRUPT FLAG
                   ;00H MEANS NO PORT INTERRUPTS
                   ;80H MEANS INTERRUPTS WHICH CAN'T INTERRUPT DISK
                   ;81H MEANS CAN INTERRUPT DISK ACCESS
```

For now, make **INTFLG** a zero byte indicating no interrupts. Most interrupt versions will use 80H. Only a few such as those for DMA disk systems will use 81H.

```
TSPRER: DB  100    ;TENTHS OF SECONDS UNTIL PRINTER ERROR
```

This byte is the number of **tenths** of seconds until a printer error is reported. The number 100, or 10 seconds, is standard.

```
MAXROW: DB 24      ;NUMBER OF LINES ON CRT DISPLAY
```

This is the number of **scrollable** lines on the CRT display. Most systems use 24 lines, but a few, such as the Sanyo MBC-1000, use 25. If this number is not set correctly, KAYLINK will not scroll properly.

See the example for **aHINIT** later on in this section for additional terminal requirements.

```
UPAROW: DB         'L'-40H   ;ALTERNATE CURSOR FOR TEXT EDITOR
DNAROW: DB         'J'-40H
LFAROW: DB         'H'-40H
RTAROW: DB         'K'-40H

ESCSTR: DB         'ESC',0,0,0,0,0
HLPSTR: DB         'HELP',0,0,0,0
RETSTR: DB         'RETURN',0,0
DELSTR: DB         'DEL',0,0,0,0,0,0,0
ARRSTR: DB         'Arrow keys',0
```

Don't bother changing the above listed bytes and strings until later.

Next, plug in the installable routines. All of these go after **ARRSTR**. For purposes of clarity, put them into the HIOS in the order in which they appear in the jump vectors.

aHINIT is probably the most important routine in the HIOS. Along with its partner, **aEXIT**, it can be used for anything from setting up the terminal, to controlling interrupts, to fixing bugs in the operating system. Although most of these functions can be put off until later, setting up the terminal must be done now.

KAYLINK requires an 80-column screen with at least 24 lines. The number of lines above 24 is variable by changing **MAXROW** listed above, but the number of columns is not. If your terminal has more than 80 columns, **aHINIT** should set up the terminal for 80 columns.

The terminal must also wrap around at the end of each line to the beginning of the next line. When the terminal wraps around at the end of the very bottom line on the screen, the screen must scroll up.

Most terminals do all these things by default and do not need to be set up. Some, however, must be set up by sending a string of bytes to the terminal. The following is **sample** code for a Zenith Z-100 system:

```
ESC      EQU  1BH        ;FIRST CHARACTER IN ESCAPE SEQUENCE

aHINIT:  LXI  D,SETUP    ;DE POINTS TO SET UP STRING
         CALL WSTC       ;USE KAYLINK ROUTINE TO SEND STRING TO TERMINAL
         RET

SETUP:   DB   ESC,'y?'   ;DISABLE EXPANDED KEY CODES
         DB   ESC,'v'    ;ENABLE WRAP-AROUND AT END OF LINE
         DB   0          ;A ZERO BYTE TERMINATES THE STRING
```

The next few routines do not need to be written yet but require place holders.

```
aEXIT:                   ;ANY CLEANING UP BEFORE EXIT
         RET
aEMINT:                  ;EMULATE INTERRUPTS
         RET
aENINT:                  ;ENABLE INTERRUPTS
         RET
aDISINT:                 ;DISABLE INTERRUPTS
         RET
aENTXE:                  ;ENABLE MODEM (PORT) TRANSMITTER
         RET
aENRCV:                  ;ENABLE MODEM (PORT) RECEIVER
         RET
```

```
aDISTXE:                ;DISABLE MODEM (PORT) TRANSMITTER
        RET
aDISRCV:                ;DISABLE MODEM (PORT) RECEIVER
        RET
aMODOUT:                ;SEND BYTE TO MODEM (PORT)
        RET
aMODIN:                 ;GET BYTE FROM MODEM (PORT)
        RET
aTIMEND:                ;END OF TIMER INTERRUPT ROUTINE
        RET
```

aPRSTAT and **aPROUT** are the printer status and output routines. **aPRSTAT** should return A = 0FFH if it is O. K. to send a character to the printer and A = 00H otherwise. **aPROUT** should send the character in register A to the printer. Because these are standard CP/M functions, implementing them is very easy. Use the following routines:

```
BDOS    EQU  0005H      ;ADDRESS OF JUMP VECTOR TO BDOS
WBOOTA  EQU  0001H      ;ADDRESS OF POINTER TO WARM BOOT
BIOSPST EQU  3*15       ;DISPLACEMENT TO BIOS PRINTER STATUS ROUTINE

aPRSTAT:                ;RETURN PRINTER STATUS IN THE A REGISTER
                        THIS CODE WILL NOT WORK ON CP/M 3.0 OR LATER
        LHLD WBOOTA     ;GET THE WARM BOOT ADDRESS
        MVI  L,BIOSPST  ;HL NOW POINTS TO THE BIOS PRINTER STATUS ROUTINE
        PCHL            ;DO IT AND RETURN VIA BIOS ROUTINE

aPROUT:                 ;PRINT CHARACTER IN THE A REGISTER
        MOV  E,A        ;PUT THE CHARACTER IN THE E REGISTER
        MVI  C,5        ;PRINTER OUTPUT ROUTINE IS BDOS NUMBER 5 ROUTINE
        CALL BDOS       ;DO IT
        RET
```

The next six routines only require place holders:

```
aRDRST:                 ;PHYSICAL CARD READER STATUS
        XRA  A          ;NO CHARACTER AVAILABLE
        STC             ;CARD READER STOPPED
        RET

aRDRIN:                 ;GET CARD READER CHARACTER
        RET

aAUXIST:                ;RESERVED FOR FUTURE EXPANSION
aAUXIN:
aAUXOST:
aAUXOUT:
        XRA  A
        RET
```

Next implement **aCONIST** and **aCONIN**, the console input status and console input routines. These are also very standard, but must carefully avoid a feature of CP/M. This is that BDOS function 2, console input, traps control-S. This can cause KAYLINK to hang trying to send something to the console. Because of this, you should always use BDOS function 6, direct console I/O. In this version, **aCONIST** actually gets the character from the keyboard and **aCONIN** simply plucks it out of a memory byte. We can do this because KAYLINK **always** calls **aCONIST** before calling **aCONIN**. The routines are as follows:

```
aCONIST:                    ;RETURN CONSOLE STATUS
         LDA  CHARIN        ;CHECK TO SEE IF WE ALREADY HAVE A CHARACTER
         ORA  A
         MVI  A,0FFH        ;SET STATUS TO RETURN REGISTER A =0FFH (TRUE)
         RNZ                ;EXIT WITH CHARACTER AVAILABLE IF SO
         MVI  C,6           ;OTHERWISE, USE BDOS FUNCTION 6
         MVI  E,0FFH        ;TO DO CONSOLE INPUT OR STATUS.
         CALL BDOS          ;DO IT
         ORA  A             ;SEE IF THERE IS A CHARACTER
         RZ                 ;IF NOT, RETURN WITH REGISTER A=0 (FALSE)
         STA  CHARIN        ;OTHERWISE STASH THE CHARACTER
         MVI  A,0FFH        ;MAKE REGISTER A = 0FFH (TRUE)
         RET                ;RETURN WITH CHARACTER AVAILABLE

CHARIN:  DS   1             ;HOLDS LAST CHARACTER OR 0 FOR NONE

aCONIN:                     ;GET CONSOLE INPUT CHARACTER
         LXI  H,CHARIN      ;HL POINTS TO CHARACTER SAVED BY CONIST
         MOV  A,M           ;REGISTER A = CHARACTER
         MVI  M,0           ;CLEAR THE CHARACTER
         RET
```

Now add code in **aHINIT** to set the byte to zero:

```
aHINIT:

          .
          .    preceding aHINIT code goes here
          .
          XRA  A           ;SET THE A REGISTER TO ZERO
          STA  CHARIN      ;PUT IT IN CHARIN
```

The next routine, **aCONOST**, is neither a standard BDOS nor BIOS function, and therefore is very different from machine to machine. For the first version, make **aCONOST** return 0FFH (TRUE) all the time, indicating that the console is ready to receive a character all the time. Unless your machine has a memory-mapped screen, you might later want to have this routine actually check whether the terminal is ready to receive a character. The only advantage would be speed improvements on the non-interrupt version.

```
aCONOST:                ;RETURN CONSOLE OUTPUT STATUS
        MVI   A,0FFH    ;ALWAYS READY
        RET
```

The next routine, **aCONOUT**, is a standard BDOS function and is very easy to implement.

```
BDOS    EQU   0005H     ;ADDRESS OF JUMP VECTOR TO BDOS

aCONOUT:                ;OUTPUT CHARACTER IN REGISTER A TO CONSOLE
        MOV   E,A       ;PUT CHARACTER IN E REGISTER
        MVI   C,6       ;USE BDOS FUNCTION 6 TO SEND CHAR TO CONSOLE
        CALL  BDOS      ;DO IT
        RET
```

The next routine, **aPOSCUR**, is very important. It is used throughout KAYLINK to format text on the screen. KAYLINK will call this routine with the D register set to the number of the row (starting with 0 at the top of the screen) and the E register set to the number of the column (starting with 0 at the left-hand side of the screen). This routine will then move the console screen cursor to that row and column.

Although the method of doing this can vary wildly from terminal to terminal, many terminals use an escape sequence followed by the row and column added to decimal 32. The following routine is for the Zenith Z-19 but can be modified for most terminals:

```
ESC     EQU   1BH       ;FIRST CHARACTER IN ESCAPE SEQUENCE

aPOSCUR:                ;POSITION CURSOR TO ROW D COLUMN E
        XCHG            ;PUT DE INTO HL
        LXI   D,POSSTR  ;DE POINTS TO CURSOR POSITIONING STRING
        CALL  WSTC      ;USE KAYLINK ROUTINE "WSTC" TO DISPLAY STRING
        MOV   A,H       ;A=ROW
        ADI   A,32      ;OFFSET IT BY 32
        STA   ROW       ;PLUG IT INTO STRING
        MOV   A,L       ;A=COLUMN
        ADI   32        ;OFFSET IT BY 32
        STA   COLUMN    ;PLUG IT INTO STRING
        XRA   A         ;SET A TO ZERO
        STA   RCEND     ;PLUG IT INTO STRING
        LXI   D,ROW     ;DE POINTS TO STRING CONTAINING CURSOR POSITION
        CALL  WSTC      ;SEND IT TO THE CONSOLE
        RET

POSSTR: DB    ESC,'Y',0 ;Z-19 USES ESC Y ROW COLUMN
ROW:    DS    1         ;ROW-COLUMN STRING
COLUMN: DS    1
RCEND:  DS    1         ;SPACE FOR TERMINATOR
```

The next routine, **aCLRSCR**, is used to clear the screen and home the cursor. Like **aPOSCUR** above, this routine usually requires sending a string to the console. The following routine is for the Zenith Z-19:

```
aCLRSCR:                    ;CLEAR THE SCREEN AND HOME THE CURSOR
          LXI   D,CLRSTR ;DE POINTS TO CLEAR SCREEN STRING
          CALL  WSTC      ;SEND STRING TO CONSOLE
          RET
CLRSTR:   DB    ESC,'E',0 ;ZENITH USES ESC E
```

When you have made all these changes, use the procedure described in Section 4.7 to assemble your HIOS and insert it into KAYLINK.

Next, try out KAYLINK to see how it flies. Use the standard HOME key (CTRL- SHIFT-6 on most terminals) as the HELP key.

Page through the menus. Try out the Text Editor. Try the Local Disk Functions. List a file to the screen and to the printer.

Don't try doing any communications; they will not work. Also, don't expect the printer error time-out to work properly; until the **aEMINT** routine is written no time-outs can occur.

When you are satisfied that the HIOS so far functions properly, you can go to the next step and install communications.

5.2 Installing Communications (Non-Interrupt)

The next step in building a HIOS is to install non-interrupt communications. To do this you will need detailed documentation for your computer as well as complete specification sheets on the I/O chip for your machine. Examples are given for the 8251 and Z-80 SIO chips. These examples are by no means complete, although they do give an overview of programming those particular chips. Computers with different chips may, of course, require a completely different programming method. Even for computers with the same chips as the examples, any number of things may differ from computer to computer.

Most computers use a port system to address the I/O chip. That is, the chip is accessed with standard 8080 **IN** and **OUT** instructions. The examples to come use labels for the I/O chip addresses, but use no **EQU**ates to give those addresses values. It is your responsibility to determine the proper addresses for your computer. **MIOS** is the I/O chip status port, **MIOC** is the command port (usually the same as the status port), and **MIOD** is the data port.

More examples of routines can be found on the HIOSes included on the distribution disk.

The first routine which must be changed is **aHINIT**. This routine now has the responsibility of programming the I/O chip for synchronous communications with eight data bits, no parity, and two sync bytes: 32H and 32H. Set up the I/O chip to receive external clocks if the chip has that feature; in any event the I/O chip must be set to X1 clock. **aHINIT** must also raise the DTR (Data Terminal Ready) line.

This example is for a Z-80 SIO chip.

```
SYNC    EQU  32H

SIOTBL:                           ;TABLE OF BYTES TO PROGRAM Z-80 SIO
        DB   18H                  ;CHANNEL RESET
        DB   04H                  ;SELECT REGISTER 4
        DB   10H                  ;2-BYTE SYNC, NO PARITY, X1 CLOCK MODE
        DB   03H                  ;SELECT REGISTER 3
        DB   E0H                  ;SET UP AUTO ENABLES
        DB   05H                  ;SELECT REGISTER 5
        DB   80H                  ;RAISE THE DTR LINE
        DB   06H                  ;SELECT REGISTER 6
        DB   SYNC                 ;PROGRAM SYNC BYTE 1
        DB   07H                  ;SELECT REGISTER 7
        DB   SYNC                 ;PROGRAM SYNC BYTE 2
        DB   01H                  ;SELECT REGISTER 1
        DB   00H                  ;NO INTERRUPTS FOR NOW
SIOEND:                           ;END OF SIO TABLE

aHINIT:
        .
        .    preceding aHINIT code goes here
        .
        MVI  B,SIOEND-SIOTBL ;# OF BYTES TO PROGRAM MODEM PORT
        LXI  H,SIOTBL             ;HL POINTS TO TABLE OF BYTES TO SEND
PGMSIO: MOV  A,M                  ;GET A BYTE
        OUT  MIOC                 ;OUTPUT TO MODEM I/O COMMAND
        INX  H                    ;POINT TO NEXT BYTE
        DCR  B                    ;DECREMENT BYTE COUNT
        JNZ  PGMSIO               ;LOOP UNTIL DONE
        RET
```

The following example is for an 8251. Notice the similarity to the previous example.

```
SYNC    EQU  32H

SIOTBL:                           ;TABLE OF BYTES TO PROGRAM 8251
        DB   0,0,0                ;MAKE SURE 8251 IS WAITING FOR A COMMAND
        DB   40H                  ;INTERNAL RESET
        DB   0CH                  ;SET UP SYNCHRONOUS MODE, 8 BITS, 2 SYNC BYTES
        DB   SYNC,SYNC            ;SYNC BYTES
        DB   12H                  ;ERROR RESET, RAISE DTR LINE
SIOEND:                           ;END OF SIO TABLE

aHINIT:
        .
        .    preceding aHINIT code goes here
        .
        MVI  B,SIOEND-SIOTBL ;# OF BYTES TO PROGRAM MODEM PORT
        LXI  H,SIOTBL          ;HL POINTS TO TABLE OF BYTES TO SEND
PGMSIO: MOV  A,M               ;GET A BYTE
        OUT  MIOC              ;OUTPUT TO MODEM I/O COMMAND
        INX  H                 ;POINT TO NEXT BYTE
        DCR  B                 ;DECREMENT BYTE COUNT
        JNZ  PGMSIO            ;LOOP UNTIL DONE
        RET
```

Both of the listed **aHINIT** routines raise the DTR line. When KAYLINK exits, the DTR line must go back down to indicate that KAYLINK is no longer communicating. This is especially important when some form of auto-dial or auto-answer equipment is used. The **aEXIT** routine must therefore drop the DTR line. The following example is for a Z-80 SIO:

```
aEXIT:                  ;CLEANUP BEFORE EXIT
        MVI  A,5        ;SELECT REGISTER 5
        OUT  MIOC
        MVI  A,0        ;DROP DTR
        OUT  MIOC
        MVI  A,18       ;CHANNEL RESET
        OUT  MIOC
        RET
```

The following example is for an 8251:

```
aEXIT:                  ;CLEANUP BEFORE EXIT
        MVI  A,0        ;DROP DTR
        OUT  MIOC
        RET
```

The next routine which must be written is **aEMINT**. The example which follows should work properly providing that the helping routines **MICRDY** and **MOCRDY** (explained later) are written correctly.

```
PSPEED  EQU  50         ;PROCESSOR SPEED IN 100KHZ UNITS (HERE, 5 MHZ)
                        ;PLUG IN THE SPEED OF YOUR PROCESSOR HERE
TENTH   EQU  24*PSPEED  ;NUMBER OF EMULATED INTERRUPTS PER TENTH SECOND

aEMINT:                 ;EMULATE INTERRUPTS
                        ;MODEM (PORT) SECTION
        CALL MICRDY     ;CHECK IF MODEM (PORT) HAS A CHARACTER TO INPUT
        CNZ  INTMIN     ;CALL KAYLINK MODEM INPUT ROUTINE IF SO
        CALL MOCRDY     ;CHECK IF MODEM (PORT) IS READY TO OUTPUT A CHAR
        CNZ  INTMOUT    ;CALL KAYLINK MODEM OUTPUT ROUTINE IF SO
                        ;TIMER SECTION
        LHLD TIMER      ;GET COUNTER FOR INTERRUPT
        DCX  H          ;COUNT DOWN ONE
        SHLD TIMER      ;PUT IT BACK
        MOV  A,H        ;CHECK TO SEE IF COUNTED DOWN TO ZERO
        ORA  L
        RNZ             ;EXIT IF NOT
        LXI  H,TENTH    ;IF DOWN TO ZERO, RESET IT TO THE TOP (TENTH)
        SHLD TIMER
        CALL INTIMER    ;AND CALL THE KAYLINK TIMER INTERRUPT ROUTINE
aTIMEND:                ;END OF TIMER INTERRUPT
        RET

TIMER:  DS   2          ;COUNTER FOR INTERRUPT
```

To initialize TIMER, add the following code to the very end of **aHINIT** before the final **RET**:

```
LXI   H,TENTH
SHLD  TIMER
```

Notice the use of **aTIMEND** in the example. Because this is not an interrupt version, **aTIMEND** does not need to signal the end of the interrupt. However, it is the end of the timer interrupt and belongs here.

Now the helping routines **MICRDY** and **MOCRDY** must be written. **MICRDY** tests to see if there is a character ready to be input from the I/O port and clears the Z flag if so. Similarly, **MOCRDY** tests to see if the I/O port is ready to output a character and clears the Z flag if so.

Notice that the setting of the Z flag is different from the normal HIOS convention of returning A = 0FFH for "true" and A = 0 for "false," as used in routines like **aPRSTAT**. This is because these are only helping routines and are not used by KAYLINK. Also, these routines should be as fast as possible, and it is faster to set a flag than to set a register to one of two values.

Both **MICRDY** and **MOCRDY** may differ greatly from system to system. The following example is for a Z-80 SIO system:

```
MICRDY:                     ;MODEM (PORT) INPUT TEST
          IN    MIOS        ;GET BYTE FROM STATUS REGISTER ON I/O CHIP
          ANI   1           ;TEST RX CHARACTER AVAILABLE BIT
          RET
MOCRDY:                     ;MODEM (PORT) OUTPUT TEST
          IN    MIOS        ;GET STATUS
          ANI   4           ;TEST TX CHARACTER AVAILABLE BIT
          RET
```

The following example is for an 8251:

```
MICRDY:                     ;MODEM (PORT) INPUT TEST
          IN    MIOS        ;GET STATUS
          ANI   2           ;TEST RX RDY BIT
          RET
MOCRDY:                     ;MODEM (PORT) OUTPUT TEST
          IN    MIOS        ;GET STATUS
          ANI   1           ;TEST TX RDY BIT
          RET
```

If you like, you can eliminate the **CALL**s to these routines entirely and put the contents of the routines where the **CALL**s were. This will speed things up but will make the conversion to interrupts harder.

The next two routines, **aENINT** and **aDISINT**, can be just a return for non- interrupt installations. In interrupt driven systems, these routines **must** enable and disable the interrupts used by KAYLINK. They should not affect interrupts used only by the operating system.

```
aENINT:                         ;ENABLE INTERRUPTS
        RET

aDISINT:                        ;DISABLE INTERRUPTS
        RET
```

Enabling and disabling the transmitter and receiver is done by four routines, **aENTXE, aENRCV, aDISTXE,** and **aDISRCV**, all of which are defined in Section 4.4. The transmitter enable routine, **aENTXE**, must reset error flags, set DTR (Data Terminal Ready) and RTS (Request To Send) RS-232 lines and must enable the I/O chip transmitter. The receiver enable routine, **aENRCV**, must reset error flags, set the DTR (Data Terminal Ready) line, clear input registers, enable the I/O chip receiver, and enter hunt mode. Both disable routines, **aDISRCV** and **aDISTXE**, must reset all lines but leave DTR set.

The following example of all four routines is for a Z-80 SIO:

```
aENTXE:                         ;ENABLE TRANSMITTER
        MVI   A,35H             ;ERROR RESET AND POINT TO REGISTER 5
        OUT   MIOC              ;OUTPUT TO I/O COMMAND REGISTER
        MVI   A,0EAH            ;8 BITS, TXEN, DTR, AND RTS
        OUT   MIOC
        RET
aENRCV:                         ;ENABLE RECEIVER
        IN    MIOD
        IN    MIOD              ;CLEAR INPUT REGISTERS
        MVI   A,33H             ;ERROR RESET AND SELECT REGISTER 3
        OUT   MIOC
        MVI   A,0F1H            ;8 BITS, RCEN, DTR, AND ENTER HUNT MODE
        OUT   MIOC
        RET
aDISTXE:                        ;DISABLE TRANSMITTER
        MVI   A,28H$5           ;RESET TXE INTERRUPT PENDING AND SELECT REG 5
        OUT   MIOC
        MVI   A,80H             ;TURN OFF TRANSMITTER BUT KEEP DTR SET
        OUT   MIOC
        RET
aDISRCV:                        ;DISABLE RECEIVER
        MVI   A,3               ;SELECT REGISTER 3
        OUT   MIOC
        MVI   A,20H             ;TURN OFF RECEIVER, KEEPING AUTO ENABLES
        OUT   MIOC
        RET
```

These routines are for an 8251:

```
aENTXE:                         ;ENABLE TRANSMITTER
           MVI  A,33H           ;RESET ERRORS AND TURN ON DTR, RTS, AND TXEN
           OUT  MIOC
           RET
aENRCV:                         ;ENABLE RECEIVER
           MVI  A,96H           ;RESET ERRORS, TURN ON DTR AND RXEN, ENTER HUNT
           OUT  MIOC
           IN   MIOD            ;CLEAR INPUT REGISTERS
           IN   MIOD
           RET
aDISTXE:                        ;TURN OFF TRANSMITTER
           MVI  A,02H           ;TURN OFF EVERYTHING EXCEPT DTR
           OUT  MIOC
           RET
aDISRCV:                        ;TURN OFF RECEIVER
           MVI  A,02H           ;TURN OFF EVERYTHING EXCEPT DTR
           OUT  MIOC
           RET
```

The last two routines to be written are **aMODOUT**, the modem (port) output routine, and **aMODIN**, the modem (port) input routine. These routines, which effectively control all actual communication, are actually the easiest to write. The following example should work for practically all types of I/O chips:

```
aMODOUT:                        ;MODEM OUTPUT ROUTINE
           OUT  MIOD            ;OUTPUT BYTE IN REGISTER A TO I/O DATA REGISTER
           RET
aMODIN:                         ;MODEM INPUT ROUTINE
           IN   MIOD            ;GET BYTE FROM I/O DATA REGISTER
           RET
```

If your routines are written correctly, KAYLINK should now be able to communicate. Use the procedure outlined in Section 4.7 to install the HIOS into KAYLINK.

To determine whether your estimate of the processor speed was close enough, use the procedure in Section 9.1 of the Users' Manual to send a file to the printer, **but switch the printer off-line**. If you used the number 100 for **TSPRER** in Section 5.1, exactly 10 seconds should pass before the message ***** PRINTER PROBLEM ***** appears in the Status Window.

If it takes longer than 10 seconds, many things could be the matter. The speed of the emulated time interrupts depends primarily on the speed of **aCONIST** and **aCONOST** as well as, for the printer error at least, on the speed of the **aPRSTAT** routine. On some machines, such as the Zenith Z-100, these routines can be painfully slow.

For computers with external terminals, you may have to write the

aCONOST routine so that it returns true console output status. This will have no effect on computers with internal memory-mapped screens.

If the printer error never appears and KAYLINK hangs and stops responding to keystrokes, the printer status routine in the BIOS is probably wrong. Since very few programs actually test printer status, many computers of a certain type can be sold before somebody discovers the error. Either fix the printer status routine or update the HIOS to use interrupts. The interrupt version can detect printer errors even if the printer status is wrong.

5.3 Cleaning Up the HIOS

Once KAYLINK is working, it should be made to work as well and as nicely as possible. This chapter explains how to change the way KAYLINK interacts with the user to make its use clear and simple.

Reverse video is used by KAYLINK to display status messages and key names and makes the program much easier to understand. To use reverse video, your terminal must have character-by-character (not line-by-line or block-by-block) reverse video. There must be a way to make all subsequent characters be displayed in reverse video or normal video as the programmer chooses. Also, the method of enabling and disabling reverse video must not take up any space on the screen.

Most terminals that have reverse video meet these requirements. A few, such as some Televideo terminals, which have reverse video cannot be used because they fail one or more of the above requirements.

For those terminals that can be used, the following two routines must be written: **aENRV**, which enables reverse video for all subsequent characters, and **aDISRV**, which disables reverse video for all subsequent characters. This example is for a Xerox 820-II:

```
aENRV:                        ;ENABLE REVERSE VIDEO
         LXI  D,ENSTR
         CALL WSTC
         RET
aDISRV:                       ;DISABLE REVERSE VIDEO
         LXI  D,DISSTR
         CALL WSTC
         RET

ENSTR:   DB   ESC,')',0       ;STRING FOR 820-II TO ENABLE ATTRIBUTE
DISSTR:  DB   ESC,'(',0       ;STRING FOR 820-II TO DISABLE ATTRIBUTE
```

Some terminals must also be set up to do reverse video. For example, the Xerox 820-II must have its attribute set to reverse video rather than blinking, low intensity, or graphics. The place to do this is in **aHINIT**. This example is for an 820-II:

```
aHINIT:                             ;KAYLINK INITIALIZATION
          .
          .    existing code
          .
          LXI  D,RVSTR              ;DE POINTS TO REVERSE VIDEO STRING
          CALL WSTC                 ;SET 820-II ATTRIBUTE TO REVERSE VIDEO
          DI                        ;(XEROX RE-ENABLES INTERRUPTS ON CONSOLE OUTPUT)
          .
          .    more existing code
          .
          RET

RVSTR:    DB   ESC,'7',0
```

KAYLINK can use the erase to end-of-line features of the terminal to speed up replots considerably. What is required is a function that erases all the characters from and including the cursor character up to and including the right-most character in the line.

Most terminals require a special character or escape sequence to be sent to perform this function. The following example is for a Xerox 820 or 820-II:

```
aETEOLN:                            ;ERASE TO END-OF-LINE
          LXI  D,ELSEQ              ;DE POINTS TO ERASE SEQUENCE
          CALL WSTC                 ;SEND IT
          RET

ELSEQ:    DB   'R'-40H,0            ;820 USES CTRL-R
```

Notice that the **WSTC** routine is used, even though only one character is to be output. This is good practice for all routines in the HIOS.

The next thing to do is to change the HELP key. Pick a key on the keyboard. It should be easy-to-find. It should be accessed by a single keypress (no SHIFT or CTRL), especially if KAYLINK will be used by people less experienced than you are. People who do not know that you have to hold down the CTRL key while pressing the other key are by no means rare. And finally, the name should be seven characters or less long.

Function keys, the TAB key, or any HELP key are good choices. The LINE FEED key is **not** a good choice; it is used by the Text Editor.

In using alternate keys for HELP there are two problems to be solved. The first is the fact that KAYLINK expects the character code 1EH to be returned for the HELP key. This is very easy to solve by making minor modifications to **aCONIN**. The following example uses the TAB key as the HELP key:

```
TAB       EQU  'I'-40H          ;TAB KEYCODE
HELP      EQU  1EH              ;HELP KEYCODE

aCONIN:                         ;GET CONSOLE INPUT CHARACTER
          LXI  H,CHARIN         ;THIS CODE IS THE SAME AS
          MOV  A,M              ; THE PRELIMINARY VERSION DESCRIBED
          MVI  M,0              ; IN SECTION 5.1
                                ;THE FOLLOWING CODE IS NEW
          CPI  TAB              ;SEE IF CHARACTER IS A TAB
          RNZ                   ;IF IT ISN'T, JUST RETURN
          MVI  A,HELP           ;OTHERWISE, MAKE IT A HELP KEYCODF
          RET
```

The next problem is not so easy to solve. When a function key is pressed, many terminals send escape sequences instead of single keycodes. Since KAYLINK expects a single keycode, these escape sequences must be converted. The way to do this is to have **aCONIN**, whenever it sees an ESC code, wait a short while to see if another keycode is coming. If so, it is an escape sequence, and can be converted. If not, the code for ESC should be returned.

The following example is one of the many ways to do this. It returns the HELP code whenever the BLUE function key on the Zenith Z-19 is pressed. All other function keys will act as the ESC key.

```
ESC       EQU  1BH              ;ESC KEYCODE
HELP      EQU  1EH              ;HELP KEYCODE

aCONIN:                         ;GET CONSOLE INPUT CHARACTER
          LXI  H,CHARIN         ;THIS CODE IS THE SAME AS
          MOV  A,M              ; THE PRELIMINARY VERSION DESCRIBED
          MVI  M,0              ; IN SECTION 5.1
                                ;THE FOLLOWING CODE IS NEW
          CPI  ESC              ;SEE IF CHARACTER IS AN ESC
          RNZ                   ;IF IT ISN'T, JUST RETURN

          MVI  C,6              ;WAIT FOR THE TIME IT TAKES TO SEND 6 NULLS
STALLC:   PUSH B                ;TO THE CONSOLE (TRY OTHER VALUES, TOO)
          XRA  A                ;A=NULL
          CALL CONOUT           ;SEND IT
          POP  B
          DCR  C                ;SEE IF 6 HAVE BEEN SENT
          JNZ  STALLC           ;LOOP IF NOT

          CALL CONIST           ;SEE IF ANOTHER CHAR IS AVAILABLE
          ORA  A                ;SEE IF CONIST RETURNS TRUE
          MVI  A,ESC            ;SET UP ESC TO BE RETURNED, JUST IN CASE
          RZ                    ;IF NO OTHER CHARACTER, JUST RETURN AN ESC
          CALL CONIN            ;GET THE NEXT CHARACTER IN THE ESCAPE SEQUENCE
          CPI  'P'              ;CHECK FOR AN ESC-P (BLUE FUNCTION)
          MVI  A,HELP
          RZ                    ;IF SO, RETURN A HELP
          MVI  A,ESC            ;IF NOT, RETURN AN ESC—ALL ESCAPE SEQUENCES
          RET                   ;OTHER THAN ESC-P WILL JUST RETURN AN ESC
```

When there is more than one escape sequence to be translated, a table lookup is a better way of doing the translation. Also, there are many trickier and perhaps better ways of determining the existence of an escape sequence, but the one above has the advantages that it is simple and works largely independently of terminal speed.

Depending on your particular terminal, you may have to use the above procedure to change the DEL (Delete) key as well. KAYLINK expects the DEL key to return 7FH.

The Text Editor in KAYLINK uses CTRL-H, J, K, and L to move the cursor. Many terminals have arrow keys which can simplify the use of the Text Editor greatly. For this reason, KAYLINK provides for an alternate set of cursor control keys.

After **MAXROW** near the beginning of the HIOS are four **DB** statements labeled **UPAROW, DNAROW, LFAROW,** and **RTAROW.** Each of these contain a single byte giving the keycode of an alternate cursor control key.

Change these bytes to take advantage of the arrow keys on your terminal, if any. The following example is for a Xerox 820 or 820-II:

```
UPAROW:  DB   'A'-40H          ;UP ARROW KEY
DNAROW:  DB   'B'-40H          ;DOWN ARROW KEY
LFAROW:  DB   'D'-40H          ;LEFT ARROW KEY
RTAROW:  DB   'C'-40H          ;RIGHT ARROW KEY
```

The next things to change are the strings which KAYLINK will display for various keycodes. There are five of these strings, found at the bottom of figure 5.1. They should be changed to match the names on the keys exactly. A program which says **Press the CR key** when the key actually says **RETURN** is sloppy and confusing.

When changing the strings, **be very careful** how many characters you use. Forgetting a single padding zero byte can offset the rest of the strings by one character. Also, the menus and Help Screens are designed to work only with a certain number of characters per string.

```
ESCSTR:  DB   'ESC',0,0,0,0,0    ;ESCAPE STRING—EXACTLY 8 BYTES
```

This string is the name of the ESC or ESCAPE key. It should be changed to use the exact legend on the ESC key. Notice the zero bytes after the string. Only the first is used to terminate the string; the rest are there to pad the entire string out to eight characters. The longest ESC string could therefore have seven letters plus one terminating zero.

```
HLPSTR:  DB    'HELP',0,0,0,0     ;HELP STRING—EXACTLY 8 BYTES
```

This is the string for the HELP key. Because very few terminals have a key named **HELP**, this string will probably need to be changed. Like the ESC string above, this string must have exactly eight characters including a terminating zero. There are further restrictions on the length of the ESC and HELP strings, however. The total number of **printable** characters in both strings may be at most eleven characters. If they have more, the status line at the top of the HASP Station Console will be ruined. The strings **ESC** and **HELP** only add up to seven characters, so there is plenty of room. A combination of strings such as **ESCAPE** and **HERE IS**, however, would be too long, even though each string individually contains seven characters or fewer.

```
RETSTR:  DB    'RETURN',0,0       ;RETURN STRING—EXACTLY 8 BYTES
DELSTR:  DB    'DEL',0,0,0,0,0,0,0 ;DELETE STRING—EXACTLY 10 BYTES
```

Note that the DEL string has space for ten characters, not eight. This is so that the ADM-3A **SHIFT-RUB**, with nine printing characters, just fits.

```
ARRSTR: DB   'Arrow keys',0  ;CURSOR STRING—AT MOST 13 CHARS
```

This string gives the name of the cursor control keys for the Text Editor. Since this is the last string, it does not need to have an exact length and can contain up to 13 **printing** characters. If your terminal has no alternate arrow keys, use the string **CTRL-H,J,K,L**. On terminals such as the Zenith Z-19, for example, you may want to use **SHIFT-Arrows**.

5.4 Installing Port Interrupts

Installing interrupts is by far the most difficult part of writing a HIOS. Only attempt it if you have plenty of skill and experience and **complete** documentation for your system.

The rewards for installing interrupts are as great as the difficulties. Interrupt versions of KAYLINK are faster and generally much nicer than non-interrupt versions.

Because the methods of using interrupts can vary from machine to machine, it is impossible to present a reasonable sample here. Instead, we will go through the procedure of writing interrupts for an imaginary machine.

These are the specifications of the imaginary machine:

* 2 MHZ Z-80 processor
* Standard Z-80 SIO chip, accessed at I/O ports 10H-13H
* Standard Z-80 CTC timer chip, accessed at I/O ports 20H-24H
  Neither the SIO nor the CTC are used by the operating system.
  SIO and CTC are connected in a standard daisy chain.
* A disk system whose reads and writes cannot be interrupted
* No obtrusive bugs in the operating system

Even though a Z-80 microprocessor is used, examples of both 8080 **and** Z-80 code will be given. The 8080 code is given so that you can use the standard CP/M assembler **ASM**. To provide some necessary Z-80 instructions in the 8080 code versions, **DW** statements will be used.

The first thing to do is install port interrupts. This is what will actually speed up communications.

First of all, signal that the HIOS now uses port interrupts. On figure 5.1 you can see a single byte named **INTFLG**. Change this byte to an 80H, indicating port interrupts that cannot interrupt disk access.

The next thing to do is actually to install the interrupts. Because the machine is simplified, so is the task. For one thing, it is given that the operating system does not use the interrupts at all. Also, the operating system has no obtrusive bugs. This is an extremely rare and delightful quality. In the real world, operating systems are often riddled with bugs, especially those that affect interrupts.

However, our machine has none of those problems. It uses the standard Z-80 interrupt system which is very simple. In this system, the Z-80 vectored interrupt mode 2 is used. The CPU is given a byte which will be the high-order byte of an interrupt vector table somewhere in memory. When the device interrupts, it will supply a low-order byte, making a complete 16-bit address. The CPU will then look at the 2-byte word address at that location and effectively do a **CALL** to that address.

Because the interrupt devices are connected in their own daisy-chain configuration, all the problems of interrupt priorities are invisible to the user. As long as a Z-80 **RETI** instruction is executed as the last instruction in the interrupt routine, all priorities will be handled automatically.

Since the operating system does not use the interrupts, we can build the interrupt vector table inside the HIOS. The Z-80 SIO has two channels, A and B. Each channel is set up to access four vectors (for our purposes all alike), or eight bytes per channel. The entire SIO therefore requires an interrupt table 16-bytes long. For consistency with earlier examples we will use SIO channel B and therefore the **second** set of four vectors.

This table must be normalized on a 16-byte boundary, that is, the four least significant bits of the address of the beginning of the table must be zero. One way to ensure that this happens is to put

```
ORG    (($+0FH)/10H)*10H ;PUT TABLE ON 16-BYTE BOUNDARY
```

just before the interrupt table.

The rest of the procedure is relatively straightforward. The high-order byte of the interrupt vector is set with a standard Z-80 instruction, while the low order byte is programmed into the SIO chip.

Several routines must then be changed. The initialization routine **aHINIT** must now set up the interrupt table and routines.

The following 8080 code for **HINIT** is taken directly from the first example in Section 5.2 and has been modified to take advantage of interrupts. Notice that the routine **MODIO** (Modem Input/Output) has been added to the end actually to process the interrupts. This routine is essentially a distilled version of the first part of the interrupt emulator in section 5.2.

```
SIOADD  EQU  10H              ;ADDRESS OF Z-80 SIO
MIOS    EQU  SIOADD+3         ;CHANNEL B STATUS
MIOC    EQU  MIOS             ;COMMAND IS THE SAME
MIOD    EQU  SIOADD+1         ;CHANNEL B DATA

RDA     EQU  1                ;RECEIVER DATA AVAILABLE MASK
TBR     EQU  4                ;TRANSMITTER BUFFER READY MASK

IM2     EQU  5EEDH            ;BACKWARD Z-80 IM   2   INSTRUCTION (USE IN DW)
LDIA    EQU  47EDH            ;     "         "  LD   I,A       "
RETI    EQU  4DEDH            ;     "         "  RETI           "

SYNC    EQU  32H

SIOTBL:                       ;TABLE OF BYTES TO PROGRAM Z-80 SIO
        DB   18H              ;CHANNEL RESET
        DB   04H              ;SELECT REGISTER 4
        DB   10H              ;2-BYTE SYNC, NO PARITY, X1 CLOCK MODE
        DB   03H              ;SELECT REGISTER 3
        DB   E0H              ;SET UP AUTO ENABLES
        DB   05H              ;SELECT REGISTER 5
        DB   80H              ;RAISE THE DTR LINE
        DB   06H              ;SELECT REGISTER 6
        DB   SYNC             ;PROGRAM SYNC BYTE 1
        DB   07H              ;SELECT REGISTER 7
        DB   SYNC             ;PROGRAM SYNC BYTE 2
        DB   02H              ;SELECT REGISTER 2
        DB   MTABLE AND 0FFH ;PROGRAM LOW-ORDER INTERRUPT VECTOR BYTE
        DB    01H                 ;SELECT REGISTER 1
        DB    1AH                 ;INTERRUPTS ON ALL TX AND RX CHARACTERS
SIOEND:

        ORG   (($+0FH)/10H)*10H  ;PUT TABLE ON 16-BYTE BOUNDARY
MTABLE:                           ;MODEM (PORT) INTERRUPT TABLE
        DS    8                   ;DON'T USE THE FIRST FOUR VECTORS
        DW    MODIO               ;ALL FOUR B CHANNEL VECTORS POINT TO
        DW    MODIO               ;PORT INTERRUPT SERVICE ROUTINES
        DW    MODIO
        DW    MODIO

aHINIT:                           ;KAYLINK INITIALIZATION
        CALL  DISINT              ;MAKE SURE WE DON'T GET INTERRUPTED
        .
        .     code to set up the terminal, etc
        .
        DW    IM2                 ;SELECT Z-80 INTERRUPT MODE 2
        MVI   A,MTABLE/256        ;A=TOP BYTE OF INTERRUPT TABLE ADDRESS
        DW    LDIA                ;PUT IT IN THE Z-80 INTERRUPT VECTOR REGISTER

        MVI   B,SIOEND-SIOTBL     ;# OF BYTES TO PROGRAM MODEM PORT
        LXI   H,SIOTBL            ;HL POINTS TO TABLE OF BYTES TO SEND
PGMSIO: MOV   A,M                 ;GET A BYTE
        OUT   MIOC                ;OUTPUT TO MODEM I/O COMMAND
        INX   H                   ;POINT TO NEXT BYTE
        DCR   B                   ;DECREMENT BYTE COUNT
        JNZ   PGMSIO              ;LOOP UNTIL DONE
        RET
```

Now we must write the actual routine that recognizes the port interrupt. All four interrupt vectors point to one place: **MODIO**. **MODIO** must therefore determine whether the transmitter or receiver requested the interrupt.

In the following routine, notice how the stack is preserved. The method uses only one level of the old stack before switching to a new one. Because an interrupt can occur at just about any time, including during a BDOS call, it is very important not to rely on many stack levels.

The examples of 8080 code use one stack level before saving the stack pointer to save the flags before executing a **DAD**. There is a trickier way to save the stack pointer on the 8080 without using any stack levels, but using one stack level works for the HIOS and is simpler. The Z-80 code example given later on shows a simpler way to save the stack pointer without using the old stack.

```
MODIO:                            ;MODEM (PORT) INTERRUPT ROUTINE
        PUSH  PSW                 ;SAVE A AND FLAGS
        SHLD  OLDHL               ;SAVE HL
        LXI   H,0
        DAD   SP                  ;HL=CURRENT STACK POINTER
        SHLD  OLDSTK              ;SAVE IT
        LXI   SP,INTSTK           ;SET STACK TO NEW INTERRUPT STACK
        PUSH  D
        PUSH  B                   ;SAVE OTHER REGISTERS

        IN    MIOS                ;GET STATUS BYTE
        PUSH  PSW                 ;SAVE IT
        ANI   RDA                 ;CHECK IF RECEIVE DATA AVAILABLE
```

```
          CNZ  INTMOUT        ;IF SO, CALL KAYLINK ROUTINE
          POP  PSW            ;GET STATUS BACK
          ANI  TBR            ;CHECK IF READY TO SEND CHARACTER
          CNZ  INTMOUT        ;IF SO, DO IT

          POP  B
          POP  D              ;RECOVER REGISTERS
          LHLD OLDSTK         ;GET OLD STACK POINTER
          SPHL                ;RESTORE IT
          LHLD OLDHL          ;GET HL BACK
          POP  PSW            ;GET A AND FLAGS BACK
          EI                  ;ENABLE INTERRUPTS AGAIN
          DW   RETI           ;Z-80 RETURN FROM INTERRUPT

OLDHL:    DS   2              ;SAVING PLACE FOR HL
OLDSTK:   DS   2              ;OLD STACK POINTER
          DS   2*20           ;20 STACK LEVELS ARE ENOUGH
INTSTK:   DS   2
```

The following is the same code slightly modified and for a Zilog Z-80 mnemonic assembler. Only the actual code is included here, **EQU**ates and **DB** statements are as above, although to follow the Z-80 mnemonics strictly, **DB**, **DW**, and **DS** should be changed to **DEFB**, **DEFW**, and **DEFS**, respectively. Notice the different and better way of saving the stack pointer.

```
aHINIT:                        ;KAYLINK INITIALIZATION
          CALL DISINT          ;MAKE SURE WE DON'T GET INTERRUPTED
          .
          .    code to set up the terminal, etc
          .
          IM   2               ;SELECT Z-80 INTERRUPT MODE 2
          LD   A,MTABLE/100H   ;A = TOP BYTE OF INTERRUPT TABLE ADDRESS
          LD   I,A             ;PUT IT IN THE Z-80 INTERRUPT VECTOR REGISTER

          LD   B,SIOEND-SIOTBL ;# OF BYTES TO PROGRAM MODEM PORT
          LD   HL,SIOTBL       ;HL POINTS TO TABLE OF BYTES TO SEND
PGMSIO:   LD   A,(HL)          ;GET A BYTE
          OUT  (MIOC),A        ;OUTPUT TO MODEM I/O COMMAND
          INC  HL              ;POINT TO NEXT BYTE
          DEC  B               ;ONE FEWER BYTE
          JP   NZ,PGMSIO       ;LOOP UNTIL DONE

          RET

MODIO:                         ;MODEM (PORT) INTERRUPT ROUTINE
          LD   (OLDSTK),SP     ;SAVE OLD STACK POINTER
          LD   SP,INTSTK       ;SET STACK TO NEW INTERRUPT STACK
          PUSH AF
          PUSH HL
          PUSH DE
          PUSH BC              ;SAVE REGISTERS

          IN   A,(MIOS)        ;GET STATUS BYTE
          PUSH AF              ;SAVE IT
          AND  RDA             ;CHECK IF RECEIVE DATA AVAILABLE
          CALL NZ,INTMIN       ;IF SO, CALL KAYLINK ROUTINE
          POP  AF              ;GET STATUS BACK
          AND  TBR             ;CHECK IF READY TO SEND CHARACTER
          CALL NZ,INTMOUT      ;IF SO, DO IT

          POP  BC
          POP  DE
          POP  HL
          POP  AF              ;RECOVER REGISTERS
```

```
LD    SP,(OLDSTK)        ;RESTORE OLD STACK POINTER
EI                       ;ENABLE INTERRUPTS AGAIN
RETI                     ;RETURN FROM INTERRUPT
```

The job is not complete, however. The interrupt emulator routine **aEMINT** should be changed so that it does not check the ports.

Look at the example of **aEMINT** in Section 5.2. The four lines at the beginning of the routine should be deleted. The routine will then look like the following:

```
aEMINT:                          ;EMULATE INTERRUPTS
                                 ;TIMER SECTION
          LHLD TIMER             ;GET COUNTER FOR INTERRUPT
          DCX  H                 COUNT DOWN ONE
          .
          .    the rest of the routine as in Section 5.2
          .
```

Due to the Z-80 SIO architecture, the routines **aENTXE**, **aDISTXE**, **aENRCV**, and **aDISRCV** do not need to be changed. This may or may not be true for other I/O chips.

The next two routines, **aENINT** and **aDISINT**, enable and disable interrupts for KAYLINK. These are critical routines – they must properly enable and disable KAYLINK's interrupts, but they must not mess with any interrupts used by the operating system. This is particularly important in a multi-user environment. In the simple case of a single user machine with no system interrupts, these routines can turn on and off all interrupts for the machine. If this is done, then all of the calls to **DISINT** and **ENINT** in our examples could be changed to **DI** and **EI** instructions.

```
ENINT:    EI                     ;ENABLE INTERRUPTS
          RET

DISINT:   DI                     ;DISABLE INTERRUPTS
          RET
```

That is all that must be done for the imaginary machine to make it run port interrupts.

5.5 Installing Timer Interrupts

The reasons for installing timer interrupts are very different from those for installing port interrupts. Timer interrupts will not appreciably speed up communications.

There are two main reasons for timer interrupts. The first is that much of the operation of KAYLINK, including communication, is based on a timer. The interrupt emulator routine, **aEMINT**, can perform these functions fairly well most of the time. Sometimes, such as when changing menus or displaying help screens, cumulative errors in the timing can occur.

The other reason has to do with the printer status routine **aPRSTAT**. Sometimes, the BIOS for a particular machine will not return the correct printer status but will indicate that the printer is ready when it really is not. This can cause KAYLINK to send a character to the printer which will wait forever to be printed. If the timer interrupt is installed, KAYLINK can detect and recover from such printer errors.

The Z-80 CTC chip in our imaginary machine runs directly off the 2 MHZ system clock. In the timer mode we shall be using, the incoming clock is divided by a prescaler factor of 16 or 256, whichever the programmer chooses. The prescaled signal is used to count down an 8-bit register to zero. The 8-bit register, called the down counter, is loaded from another data register, called the time constant, which can be programmed. Every time that the CTC counts down to zero, an interrupt will be generated. The end result is that regularly-spaced interrupt pulses will occur with a programmable period. The period of the pulses can be determined by the following formula:

Interrupt period = System clock period * Prescaler factor * Time constant

What we ideally would like is an interrupt period of one-tenth of a second (100 milliseconds) for ten interrupts a second. The frequency of the system clock is 2 MHZ, and therefore the period of the system clock is its reciprocal, or 500 nanoseconds. Assuming the maximum prescaler factor of 256 and the maximum time constant of 256, the above formula shows that the period is only about 33 milliseconds, less than one-third of what we need. A better approach, therefore, would be to configure the counter for, say, a 10 millisecond interrupt and count to ten internally in the HIOS.

A little calculation reveals that, for a 10 millisecond interrupt, the prescaler factor is 256 and the time constant is 78. The CTC chip has four channels, numbered 0 through 3. For no particular reason, we shall use channel 0.

Since the CTC uses the standard Z-80 interrupt system, like the SIO, it requires an interrupt vector table. We can simply put the CTC table at the end of the SIO table and it will automatically be on a 16 byte boundary:

```
IM2     EQU  5EEDH              ;BACKWARD Z-80 IM    2   INSTRUCTION (USE IN DW)
LDIA    EQU  47EDH              ;      "        "  LD    I,A        "
RETI    EQU  4DEDH              ;      "        "  RETI              "

        ORG  (($+0FH)/10H)*10H  ;PUT TABLE ON 16-BYTE BOUNDARY

MTABLE:                         ;MODEM (PORT) INTERRUPT TABLE
        DS   8                  ;DON'T USE THE FIRST FOUR VECTORS
        DW   MODIO              ;ALL FOUR B-CHANNEL VECTORS POINT TO
        DW   MODIO              ;PORT INTERRUPT SERVICE ROUTINES
        DW   MODIO
        DW   MODIO
CTABLE:                         ;CTC INTERRUPT TABLE
        DW   TIMINT             ;ONLY CHANNEL 0 IS USED
        DW   DUMMY
        DW   DUMMY
        DW   DUMMY

DUMMY:  EI                      ;ENABLE INTERRUPTS
        DW   RETI               ;JUST IN CASE
```

Now **aHINIT** must be modified to program the CTC. The following example shows elements that must be added to the HINIT routine described in Section 5.4:

```
CTC0    EQU  20H                ;CTC CHANNEL 0 ADDRESS

TC      EQU  78                 ;TIME CONSTANT

CTCTBL:                         ;TABLE OF BYTES FOR CTC
        DB   03H                ;RESET CHANNEL 0
        DB   CTABLE AND 0FFH    ;LOAD JUMP VECTOR
        DB   10110101B          ;LOAD TIME CONSTANT, SELECT TIMER, ENABLE INT.
        DB   TC
CTCEND:                         ;END OF CTC TABLE

aHINIT:                         ;KAYLINK INITIALIZATION
        .
        .    terminal setup, communication setup, etc.
        .

        MVI  B,CTCEND-CTCTBL    ;# OF BYTES TO PROGRAM CTC
        LXI  H,CTCTBL           ;HL POINTS TO TABLE OF BYTES TO SEND
PGMCTC: MOV  A,M                ;GET A BYTE
        OUT  CTC0               ;OUTPUT TO CTC CHANNEL 0
        INX  H                  ;POINT TO NEXT BYTE
        DCR  B                  ;ONE FEWER BYTE
        JNZ  PGMCTC             ;LOOP UNTIL DONE
        MVI  A,10
        STA  COUNTER
        RET
```

Now we must write the **TIMINT** routine. The end of this routine will form the **aTIMEND** routine.

```
TIMINT:                         ;TIMER INTERRUPT ROUTINE
        PUSH PSW                ;SAVE A AND FLAGS
        SHLD OLDHL              ;SAVE HL
        LXI  H,COUNTR           ;HL POINTS TO COUNTER
        DCR  M                  ;DECREMENT IT
        JNZ  TIMIN2             ;IF HAVEN'T WAITED FOR 10 COUNTS, EXIT
        MVI  M,10               ;OTHERWISE, RESET COUNTER
        LXI  H,0
        DAD  SP               ;HL=CURRENT STACK POINTER
        SHLD OLDSTK           ;SAVE IT
        LXI  SP,INTSTK        ;SET STACK TO INTERRUPT STACK POINTER

        PUSH D
        PUSH B                ;SAVE REGISTERS
        CALL INTIMER          CALL KAYLINK TIMER INTERRUPT ROUTINE
        POP  B
        POP  D                ;RESTORE REGISTERS
        LHLD OLDSTK           ;GET OLD STACK POINTER
        SPHL                  ;RESTORE IT
TIMIN2: LHLD OLDHL
        POP  PSW              ;RESTORE REGISTERS

aTIMEND:                      ;END OF TIMER INTERRUPT ROUTINE
        EI                    ;RE-ENABLE INTERRUPTS
        DW   RETI

COUNTR: DS   1                ;COUNTER TO 10
OLDHL:  DS   2                ;OLD VALUE OF HL REGISTER
OLDSTK: DS   2                ;OLD STACK POINTER
        DS   20*2             ;20 STACK LEVELS
INTSTK: DS   2                ;INTERRUPT STACK
```

Now that both port and timer interrupts have been written, the old **aEMINT** routine is extraneous. Get rid of it, including its internal **aTIMEND**. All that should be left is

```
EMINT:                        ;INTERRUPT EMULATOR
        RET
```

Interrupts have now completely been installed on the imaginary machine.

It would be nice if installing interrupts for real machines were this easy, but it is not so. Some machines have the right kind of interrupts but no documentation on how to use them. Still other machines have glaring bugs which do not necessarily affect the operating system but severely affect applications programs that use interrupts. It is as if most designers of machines think that nobody will ever need to use interrupts and therefore make no attempt to describe their use.

As poor as the documentation for a machine may be, it is still the best place to find out information. If that fails, try to obtain schematics for the machine. Usually the required information can be found with a little perseverance. Debugging tools such as **DDT** and **SID** can be used to hunt down bugs, and **aHINIT** can be used to fix them.

Interrupts, especially port interrupts, are almost always worth the effort.

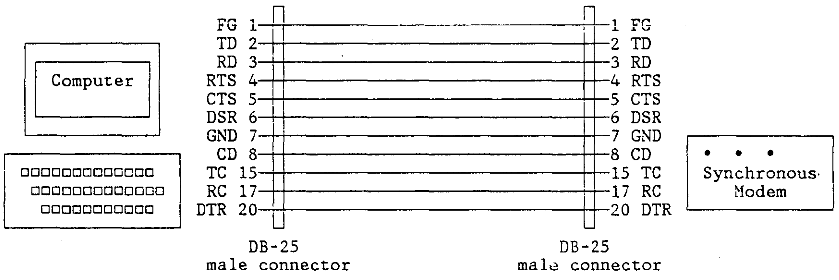


Figure 6.1 Connecting the Microcomputer to a Modem

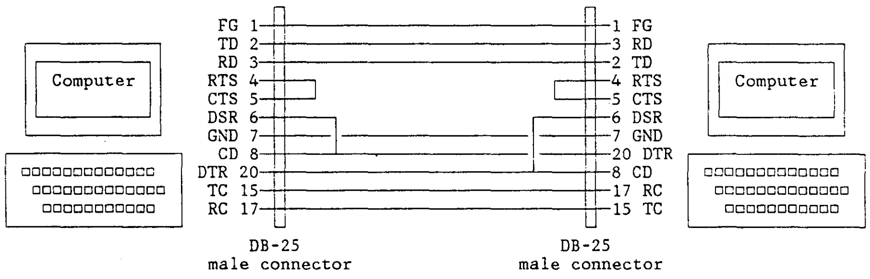


Figure 6.2 Direct Connect

6.0 Connecting KAYLINK to the Outside World

This chapter describes how to hook up your microcomputer so that it can actually communicate with other computers.

6.1 Using a Modem

The most common way of using KAYLINK is to connect the computer to a synchronous modem and connect the other end of the modem to a phone line. If you do this, you will be able to dial up any HASP system within the reach of the phone system, which is just about anywhere.

Since HASP is a synchronous protocol, you must have a synchronous modem. These come in various baud rates and can cost from slightly under $1000 up. (Bell 201 or Bell 208-compatible modems are the most widely used.) Synchronous modems supply clock pulses to keep the computer in step with what is being received. It is for this reason that you don't need to specify the baud rate in KAYLINK.

To connect your computer to the modem, you need a sync cable. This is a standard cable with 25-pin connectors at each end that has at least lines 1-8, 15, 17, and 20 connected. Plug one end of this cable into your computer and plug the other end into your modem.

Figure 6.1 on the opposite page shows the cable required.

Some computers, such as Zenith systems, require a **female** connector on the computer side.

Once you have connected the computer to the modem, you must connect the modem to the phone line. The procedure for this differs depending on what type of phone equipment you have. Consult the manual for your modem for more information.

Most systems involve a single phone line and a switch to connect the phone line to the phone or to the modem. The switch usually has two positions, called **DATA** and **TALK**. The **TALK** position is used to dial the phone, while the **DATA** position is used for data communications.

6.2 Direct Connect

Sometimes it is desirable to connect two microcomputers running KAYLINK directly to each other without using a modem. To do this you must do two things.

First, KAYLINK is designed so that the I/O chip gets its clock signals from the modem. One of the computers must, therefore, provide the clock signals. Some computers cannot be set up to provide clock signals. On those that can, such as the Xerox 820 or Xerox 820-II, set up the machine to supply clock signals through the serial port. Instructions for doing this are in appendix A. Then set the baud rate using instructions in the documentation for your machine. The synchronous baud rate will be 16 times the asynchronous baud rate.

The other computer should be set up to receive clocks as if it were connected to a modem as described in Section 6.1.

Second, a special "lie" cable must be used to fool each computer into thinking that it is talking to a modem. Figure 6.2 on the opposite page shows how to do this.

Note that some of the numbers on the right-hand cable are different from the numbers of the left-hand cable. Be sure to loop back RTS to CTS on each side.

APPENDIX A

Setting Up the Machine

KAYPRO II OR KAYPRO 4

If you have a Kaypro II or a Kaypro 4 computer, you will need to do some rewiring to run HASTE. Only attempt this if you are skilled in repairing computer equipment. Your Kaypro dealer may be able to make these changes for you if you pay a service charge.

To rewire the machine, you will need the following:

* Two (2) 470 pf (or thereabouts) mica or ceramic capacitors
* One double-pole double-throw toggle switch
* One 14-pin DIP socket with solder pins
* One number 1489 line receiver in a 14-pin DIP package
* Solder and wire
* Electronic tools such as soldering iron, screwdrivers, pliers, etc.
* A VOM or other continuity tester (optional)

(1) Remove disks from the system, turn it off, and unplug it.

(2) Unplug the keyboard cord from J3 in the back of the Kaypro and place the Kaypro on a workbench with its back toward you.

(3) Remove the screws from the top shell of the cabinet. There are two screws at the top and four on each side.

(4) Carefully lift the top shell off. The main board will be toward you on the right-hand side.

(5) Remove all plugs from the board including the two small black 4-pin plugs, the white power plug, and the grey disk drive plug.

(6) Remove the screws from the main board. There are two screws in the top of the board toward the front and six through the back of the Kaypro. The six in the back include two hexagonal bolts near J4, two small screws near J2 (be careful with these), and two other screws.

(7) The board should now move freely. Turn the board over so that you can see the bottom. If the back of the board is toward you, you will see a number in the lower right-hand corner of the board, just above the words **MADE IN USA**. There are two numbers separated by a

hyphen and then a single revision letter. In order to make the changes in this appendix, that letter must be **B** if you have a Kaypro II. If you have a Kaypro 4, the letter will probably be **A** .

(8) Locate U70. The part numbers are on the top side of the board. Once you have located U70, turn the board so that the bottom faces up and cut the copper trace between pins 11 and 12. This frees the sync detect of port A from the input of port A.

(9) Again on U70, cut the trace between pins 13 and 14. This separates the transmit and receive clock pins.

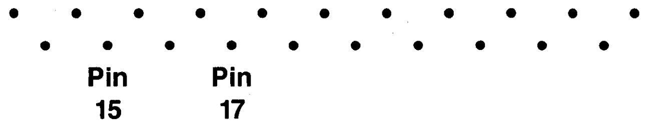
(10) Locate U75. It will be an empty hole for a 16-pin DIP. When you look at the top of the board, there is a copper trace which appears to go physically between pins 13 and 14 of this empty hole. Cut this trace. This separates the clock of U70 from the U78 clock driver.

(11) Insert and solder a 14-pin DIP socket into the U75 hole so that pin 1 of the socket rests in pin 1 of the hole. (As you look at the top of the board from the back, pin 1 will be at the upper right-hand corner of U75).

(12) Look at the new socket at U75 from the bottom of the board. Solder a jumper from pin 7 of the socket to pin 8 of the hole. This connects ground to U75. From now on, pin numbers given for U75 will refer to pin numbers of the socket, not pin numbers of the hole.

(13) Solder one 470 pf capacitor from pin 2 of U75 to pin 7 of U75 (ground). Solder the other 470 pf capacitor from pin 5 of U75 to pin 7 of U75. These provide noise immunity for U75.

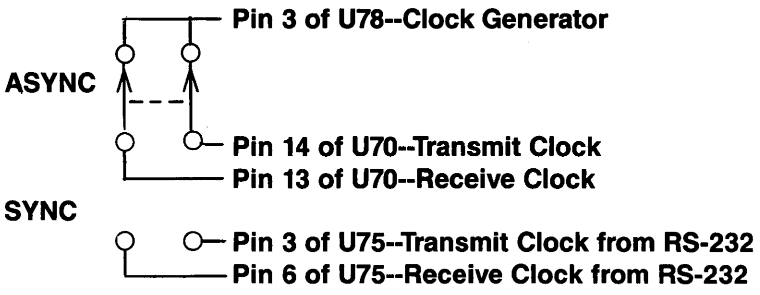
(14) Locate J4, the serial I/O jack. Solder a wire between pin 1 of U75 and pin 15 of J4. The following diagram shows a view of the pins of J4 from the bottom of the board:



(15) Solder a wire between pin 4 of U75 and pin 17 of J4. Now the external clock signals from J4 are buffered through U75.

(16) Connect the DPDT switch to the board according to the following

schematic. The schematic shows the switch in the position used for asynchronous communications. For synchronous communications such as HASTE, flip the switch to the other position. You may need to use a continuity tester to see which position is which on your switch.


Pin 3 of U78--Clock Generator
ASYNC
Pin 14 of U70--Transmit Clock
Pin 13 of U70--Receive Clock
SYNC
Pin 3 of U75--Transmit Clock from RS-232
Pin 6 of U75--Receive Clock from RS-232

(17) Insert a number 1489 line receiver in U75.

(18) Mount the switch in the back of the cabinet. A toggle switch of the proper size can easily be mounted in one of the cooling holes. For convention, mount it so that synchronous communications are selected by flipping the switch up.

(19) Replace the board and screw it in. Don't forget to assemble J2 exactly as it was before.

(20) Replace all plugs into the main board.

(21) Replace the top cover and screw it in.

Your Kaypro II should now be able to run HASTE. Flip the toggle switch up when you want to use HASTE.

The Kaypro II **cannot** be used to provide clock pulses for direct connect.

Setting Up the Machine

KAYPRO 10

If you have a Kaypro 10 computer, the changes needed in order to run HASTE are fairly simple. The Kaypro 10 was designed to support synchronous communications with only a very small modification. Still, you should not attempt it unless you are skilled in repairing computer equipment. Your Kaypro dealer may be able to make these changes for you if you pay a service charge.

To rewire the machine, you will need the following:

* One double-pole double-throw toggle switch
* Solder and wire
* Electronic tools such as soldering iron, screwdrivers, pliers, etc.
* A VOM or other continuity tester (optional)

(1) Remove disks from the system, turn it off, and unplug it.

(2) Unplug the keyboard cord from J5 in the back of the Kaypro and place the Kaypro on a workbench with its back toward you.

(3) Remove the screws from the top shell of the cabinet. There are two screws at the top and four on each side.

(4) Carefully lift the top shell off. The main board will be toward you on the right-hand side.

(5) Remove all plugs from the board including the two small black 4-pin plugs, the white power plug, and the two grey disk drive plugs.

(6) Remove the screws from the main board. There are two screws in the top of the board toward the front and seven through the back of the Kaypro. The seven in the back include two hexagonal bolts each near J3 and J4, two small screws near J6 (be careful with these), and one black screw in the top right hand corner of the back.

(7) The board should now move freely. Turn the board over so that you can see the bottom.

(8) Locate U23, a large 40-pin IC. The part numbers are on the top side of the board. Once you have located U23, turn the board so that the bottom faces up and cut the copper trace between pins 11 and 12. This

frees the sync detect of port A from the input of port A.

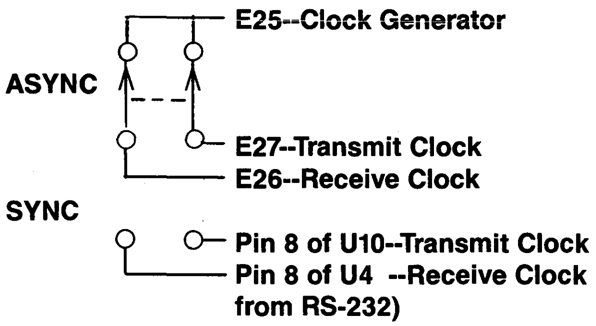
(9) Again on U23, cut the trace between pins 13 and 14. This separates the transmit and receive clock pins.

(10) Just to the right of U23 (from the back side of the board) are three pads right together and all in a line. From the top side of the board these are to the left of U23, and are labeled E25, E26, and E27. A single trace comes from each of these pads. The traces from E25 and E26 go to pads that are above and just to either side of pins 11 and 12 of U23. Locate these two pads. They are connected by a single trace. Cut this trace. **DO NOT** cut the trace from either E25 or E26, just the trace connecting them.

(11) Solder a wire to each of E25, E26, and E27. I use different colored wires for ease of identification. The wire going to E25 brings out the Kaypro's internal clock; this wire goes to both pins on one side of the double pole double throw switch. E26 and E27 bring the clock signals in to U23. They go to the two center pins on the switch.

(12) Solder a wire to pin 8 on each of U4 and U10. These are 1489 RS-232 line receivers; they are between U23 and J3. These wires bring in the external clock signals produced by a synchronous modem. The wire from U4 should go to the same pole (but opposite side) of the switch as the wire from E26. The wire from U10 should go to the same pole (but opposite side) of the switch as the wire from E27. Thus when the switch is thrown to synchronous, E27 connects to U10 pin 8, and E26 connect to U4 pin 8. In the asynchronous position, both E26 and E27 are connected to E25.

(13) Your DPDT switch should now be connected to the board according to the following schematic. The schematic shows the switch in the position used for asynchronous communications. For synchronous communications such as HASTE, the switch should be flipped to the other position. You may need to use a continuity tester to see which position is which on your switch.



(14) Mount the switch in the back of the cabinet. A toggle switch of the proper size can easily be mounted in one of the cooling holes. For convention, mount it so that synchronous communications are selected by flipping the switch up.

(15) Replace the board and screw it in. Don't forget to assemble J6 exactly as it was before.

(16) Replace all plugs into the main board.

(17) Replace the top cover and screw it in.

Your Kaypro 10 should now be able to run HASTE. Flip the toggle switch up when you want to use HASTE. The Kaypro 10 **cannot** be used to provide clocks for direct connect.

APPENDIX B

EBCDIC/ASCII Conversion Tables

The HASP protocol defines EBCDIC as the standard character set for transfer of text data. Because your machine uses the ASCII character set, characters must sometimes be translated from one character set to the other.

KAYLINK uses two tables to perform this translation, one from EBCDIC to ASCII and one from ASCII to EBCDIC. The EBCDIC to ASCII translation table is 256 bytes starting at location 700H (right after the HIOS). To translate a character from EBCDIC to ASCII, determine the number of the EBCDIC character, add it to 700H, and fetch the byte at that address. A value of 0A0H (an ASCII space with the high bit set) indicates no equivalent ASCII character.

The following is the EBCDIC to ASCII translation table as it appears in the distribution copy of KAYLINK.

```
        ORG     700H

SPG     EQU     0A0H    ;SPAGA FOR TRANSLATING BAD EBCDIC CODES

ETOAT:          ;EBCDIC TO ASCII TABLE

;               0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
        DB      00H,SPG,SPG,SPG,SPG,09H,SPG,7FH,SPG,SPG,SPG,SPG,0CH,0DH,SPG,SPG
        DB      SPG,SPG,SPG,SPG,SPG,SPG,08H,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG
        DB      SPG,SPG,1CH,SPG,SPG,0AH,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,BEL
        DB      SPG,SPG,SPG,SPG,SPG,1EH,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG
        DB      ' ',SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,'[','.','<','(','+','!'
        DB      '&',SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,']','$','*',')',';','^'
        DB      '-','/',SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,'|',',','%','_','>','?'
        DB      SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,'`',':','#','@',27H,'=','"'
        DB      SPG,'a','b','c','d','e','f','g','h','i',SPG,SPG,SPG,SPG,SPG,SPG
        DB      SPG,'j','k','l','m','n','o','p','q','r',SPG,SPG,SPG,SPG,SPG,SPG
        DB      SPG,'~','s','t','u','v','w','x','y','z',SPG,SPG,SPG,SPG,SPG,SPG
        DB      SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG,SPG
        DB      '{','A','B','C','D','E','F','G','H','I',SPG,SPG,SPG,SPG,SPG,SPG
        DB      '}','J','K','L','M','N','O','P','Q','R',SPG,SPG,SPG,SPG,SPG,SPG
        DB      '\',SPG,'S','T','U','V','W','X','Y','Z',SPG,SPG,SPG,SPG,SPG,SPG
        DB      '0','1','2','3','4','5','6','7','8','9',SPG,SPG,SPG,SPG,SPG,SPG
```

The following routine will convert the EBCDIC character in register A to its ASCII equivalent. The ETOA jump vector at the beginning of the HIOS jumps to a similar routine.

```
CVETOA: PUSH    H                       ;SAVE REGISTERS
        LXI     H,ETOAT                 ;HL- BEGINNING OF CONVERSION TABLE
        MOV     L,A                     ;HL- ASCII CHARACTER
        MOV     A,M                     ;A = ASCII CHARACTER
        POP     H                       ;RESTORE REGISTERS
        RET
```

The ASCII to EBCDIC translation table begins at address 800H. Because the ASCII character set only uses seven bits, the table is only 128 bytes long. The high bit of the ASCII character should be set to zero before adding to 800H to get the address of the EBCDIC byte.

The following is the ASCII to EBCDIC translation table as it appears in the distribution copy of KAYLINK.

```
        ORG     800H

ATOET:          ;ASCII TO EBCDIC TABLE AT 0800H

;               0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
        DB      00H,00H,00H,03H,00H,00H,00H,2FH,16H,05H,25H,00H,0CH,0DH,00H,00H
        DB      00H,00H,00H,00H,00H,00H,00H,00H,00H,00H,00H,00H,22H,00H,35H,00H
        DB      40H,4FH,7FH,7BH,5BH,6CH,50H,7DH,4DH,5DH,5CH,4EH,6BH,60H,4BH,61H

;               0    1    2    3    4    5    6    7
;               8    9    A    B    C    D    E    F
        DB      0F0H,0F1H,0F2H,0F3H,0F4H,0F5H,0F6H,0F7H
        DB      0F8H,0F9H,07AH,05EH,04CH,07EH,06EH,06FH
        DB      07CH,0C1H,0C2H,0C3H,0C4H,0C5H,0C6H,0C7H
        DB      0C8H,0C9H,0D1H,0D2H,0D3H,0D4H,0D5H,0D6H
        DB      0D7H,0D8H,0D9H,0E2H,0E3H,0E4H,0E5H,0E6H
        DB      0E7H,0E8H,0E9H,04AH,0E0H,05AH,05FH,06DH
        DB      079H,081H,082H,083H,084H,085H,086H,087H
        DB      088H,089H,091H,092H,093H,094H,095H,096H
        DB      097H,098H,099H,0A2H,0A3H,0A4H,0A5H,0A6H
        DB      0A7H,0A8H,0A9H,0C0H,06AH,0D0H,0A1H,007H
```

The following routine will convert the ASCII character in register A to its EBCDIC equivalent. The ATOE jump vector at the beginning of the HIOS jumps to a similar routine.

```
CVATOE: PUSH    H                       ;SAVE REGISTERS
        LXI     H,ATOET                 ;HL- BEGINNING OF CONVERSION TABLE
        ANI     7FH                     ;SET TOP BIT OF ASCII CHARACTER TO ZERO
        MOV     L,A                     ;HL- EBCDIC CHARACTER
        MOV     A,M                     ;A = EBCDIC CHARACTER
        POP     H                       ;RESTORE REGISTERS
        RET
```

APPENDIX C

Versions of HASTE (KAYLINK)

There are several versions of HASTE, each suited to a specific need. The word 1version1 here does not refer to the version number (i.e. 1.11 or 1.20) but rather to completely different types of HASTE.

The following is a list of all versions of HASTE that are available now:

Standard HASTE

This is the version of HASTE for which the manuals were specifically written. All information applies to standard HASTE.

B-only HASTE

This version of HASTE is intended to be used in public environments where security of the disk containing HASTE itself is a problem. What you will have is a HASTE system which only uses drive **A:** to load HASTE and uses drive **B:** for all other purposes. To run this on a machine, several things must be done. Install a startup command on the HASTE disk you will use to immediately run HASTE on power- up or reset. Then physically cut the WRITE line on drive **A:** to prevent damage to the disk on power-up. Finally, insert your HASTE disk into drive **A:**, close the door, and lock it shut so that nobody can open it without first disassembling the microcomputer.

A user of B-only HASTE cannot use drive **A:**. Drive **B:** is the default drive on startup. No questions, Help Screens, or menu options involving selecting a disk drive will appear. Also, all queries for a file name prevent the user from typing a colon (**:**). Because B-only HASTE should be running all the time, option "7" on the Main Menu, the option to exit HASTE, does not appear.

PROM HASTE

PROM HASTE is designed to be burned into 16K of PROM starting at address 0. Another 16K of RAM should occupy address space just above the PROM. PROM HASTE is intended to be used primarily as a printing station. The Text Editor works for data entry, as does the physical card reader (if any is installed), but no disk options of any kind are provided.

Because all machine-specific functions of HASTE are handled by the HIOS, no operating system or monitor is used. Because the HIOS itself is in PROM, the HIOS can no longer store bytes in the HIOS space, as most of the sample HIOSes do. Move all areas which change (**DS** statements) into the address space starting at 4000H and ending at 40FFH. Use the **aHINIT** routine to initialize all these bytes. It is surprisingly easy for even the best programmers to forget a few until the PROM is burned.

Because PROM HASTE is so small, some things had to be removed. Most Help Screens have been severely shortened. PROM HASTE cannot act as a host, and, just as for B-only, there is no way to exit.

Demonstration HASTE

Demonstration HASTE, or DEMO HASTE, is the same as standard HASTE except that all the communications software has been removed. The Status Window says **DEMONSTRATION ACTIVE** instead of **COMMUNICATIONS ACTIVE**. All other features of HASTE may be used at will. DEMO HASTE will allow you to "send" a file, the Text Screen, or a deck of cards (if you have a physical card reader) although, of course, nothing will actually be transmitted.

DEMO HASTE is sold for production and material costs to demonstrate the features of HASTE. Unlike all other versions of HASTE, DEMO HASTE does not include a licensing agreement. We encourage you to make as many copies of a DEMO HASTE (and **ONLY** DEMO HASTE) disk as you want and give them to all your friends, associates, and acquaintances. Don't copy the manuals, however--they are protected by copyright even when they accompany DEMO HASTE. Every HASTE disk contains a DEMO HASTE called **HASTE-D.COM**.

FSU HASTE

FSU HASTE is designed for communication with the FSUCC Cyber computer. It recognizes the signon message sent by the Cyber and, when a password is requested, automatically begins hiding with X's just as if CTRL-Q had been pressed. At this time it also closes all open files and resets the configuration to the startup configuration (see Section 6.1 of the Users' Manual).

FSU HASTE is not really a version in and of itself but rather a modification of other versions. There exist FSU Standard HASTE, FSU B-only HASTE, and FSU PROM HASTE.

KAYLINK

KAYLINK is a special version of HASTE sold for KAYPRO computers. There is no difference between KAYLINK and standard HASTE except for the replacement of the word HASTE by the word KAYLINK and rewording of some of the menus and Help Screens.

APPENDIX D

Revision Record

Each version of this software which is released has a unique revision number. This number is of the form "**a.bc**" where "**a**", "**b**", and "**c**" are single-digit numbers which represent levels of revision.

a	is the major design revision level. This number increases by one whenever major changes in design are made, including major rewrites.
b	is the documentation revision level. This number increases by one whenever a change is made which is large enough to require a change in the manual.
c	is the minor change level. This number increases by one whenever minor changes or bug fixes are made.

Version number 1.23, for example, specifies software with the original design and no major rewrites which has received two changes large enough to cause documentation changes and subsequently three small changes. If a documentation-level change were made, the version number would be 1.30. If the software were rewritten, the version number would be 2.00.

The first release is generally version number 1.00.

When the minor change level is increased, an addendum describing the changes may be added to the manual.

Whenever the documentation revision level is changed, the manual is changed. The revision level will be on the title page and will be a letter from A to Z with I, O, Q, and X omitted. (No letter means revision A, the original.)

DATE	REVISION	VERSION	COMMENTS
01/83	A	1.00	First Release
03/83	B	1.10	HIOS made installable, Technical Manual created
06/83	C	1.20	Card reader added, file closing question changed
10/83	D	1.30	Printer problem handling changed; Statuses added, "No More Data to

			Send" option added, Buffer sizes increased
11/83	E	1.30	Manual improved
01/84	F	1.30	KAYLINK version created

INDEX

INDEX

INDEX

COMMENT SHEET

FSUCC HASP Station Emulator
Version 1.30

____
NAME

____
COMPANY

____
ADDRESS

____
ADDRESS

____________	____________	_______
CITY	STATE	ZIP

______________	__________________
PHONE NUMBER	SOFTWARE SERIAL NUMBER

This form is not intended to be used as an order form. The Florida State University Computing Center welcomes your evaluation of this software product and manual. Please use this form to indicate any errors, suggested additions or deletions, or general comments below (please include page number references and explicit examples whenever appropriate).

________ Please Reply   ________ No Reply Necessary

FOLD

AFFIX
STAMP
HERE

Computer Marketing Associates
1535 Killearn Center Blvd.
Building D
Tallahassee, FL 32308

FOLD