ROS

RESIDENT OPERATING SYSTEM

CROMEMCO INCORPORATED

2400 Charleston Road, Mountain View, California

415-961-7400

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

The Cromemco Resident Operating System (ROS) allows the user to create and edit Z-80 source code, assemble the source code, and produce object code files. ROS resides in 8K bytes of memory space from address A000 to BFFF. 4K bytes of system RAM are required from address D000 to DFFF. User RAM may reside anywhere else in memory space.

ROS is available from Cromemco either on paper tape (model ZA-PT) or in PROM (model ZA-808). PROM may be used in the Cromemco 8K Bytesaver* memory board (model 8KBS) or in the Cromemco 16K PROM board (model 16KPR). Loading instructions for the paper tape are given in Appendix I.

Since you are no doubt anxious to begin using ROS right away, this chapter gives a detailed step-by-step example of the use of ROS in the composition, assembly, and execution of a program. <u>Before attempting to use ROS be sure that you have RAM memory in your computer from location D000 to DFFF</u> (e.g. a Cromemco model 4KZ memory board). This memory space is allocated for ROS system use. You will also need additional RAM in your system for storing your source code and the assembled object code. This is called the user RAM. For the purposes of the example in this chapter it is assumed that there is 4K of user RAM starting at location zero in memory.

Once ROS is resident in your computer, begin program execution at memory location A000. Next follow through the example given in this chapter to learn how to use this powerful software system for program development.

AN ILLUSTRATIVE EXAMPLE

Now let's consider a specific example of writing a Z-80 assembly language program, assembling the program, and executing the resultant machine code.

The title of the program is "ECHO". The purpose of the program is simply to input a character from a keyboard and echo it to a display. The program assumes standard Cromemco I/O convention of data exchange on I/O port 1 with status information on input port 0.

To begin we must execute the ROS program that begins at location A000 in memory. From the Cromemco Monitor this is accomplished by typing:

```
G A000
```

After executing ROS at location A000 depress Carriage Return on your console keyboard until the following response appears on your display:

```
CROMEMCO ROS V.2.1
```

Our assembly language source code will be stored as a "file" in the computer's memory. We must give a name to this file and specify the memory addresses in which the file resides. This is accomplished with the CFIL command. Suppose that we name the file "ECHO" and wish to have the file begin at address 0100 in memory and extend no farther than location 09FF in memory. Such a file can be created by now entering this instruction from the keyboard:

```
CFIL,ECHO,100,9FF
```

After typing this and depressing carriage return on the keyboard, ROS will respond by stating the beginning and ending address of system memory now being used:

```
D000 D1EB
```

When entering our assembly language program from the keyboard we probably would prefer to be prompted with line numbers rather than manually type the line number of each line of the program ourself. It is common to begin with line number 10 and to increment each sucessive line number by 10. For automatic prompting of line numbers we type:

```
AUTO,10,10
```

ROS will then prompt us with the first line number (a 10) and we can proceed to enter the assembly language program. After each carriage return we will be prompted with the next line number. This is shown in the example on the following page.

```
CROMEMCO RDOS 1
;G A000


CROMEMCO ROS V.2.1

CFIL,ECHO,100,9FF
D0C0 D1EB

AUTO,10,10

0010 ;THIS PROGRAM ECHOS THE KEYBOARD
0020 ;
0030 LD SP,0E00H
0040 START: CALL INPUT
0050 CALL OUTPUT
0060 JP START
0070 ;
0080 INPUT: IN A,0
0090 BIT RDA,A
0100 JR Z,INPUT
0110 IN A,1; INPUT CHARACTER
0120 RET
0130 ;
0140 OUTPUT: PUSH AF; SAVE CHARACTER
0150 IN A,0
0160 BIT TBE,A
0170 JR Z,OUTPUT+1
0180 POP AF; RETRIEVE CHARACTER
0190 OUT 1,A
0200 RET
0210 RDA: EQU 6
0220 TBE: EQU 7
0230
```

The above is a transcript of an actual session at a keyboard using ROS. At this point we may wish to have a formatted listing of our file. This can be done first by depressing ESC or ALT-MODE on the terminal keyboard to indicate that we are finished entering the assembly language program. Then we type:

```
FORM
LIST
```

The resultant listing is shown on the next page.

```
FORM
LIST
0010 ;THIS PROGRAM ECHOS THE KEYBOARD
0020 ;
0030          LD    SP,0E00H
0040 START:   CALL  INPUT
0050          CALL  OUTPUT
0060          JP    START
0070 ;
0080 INPUT:   IN    A,0
0090          BIT   RDA,A
0100          JR    Z,INPUT
0110          IN    A,1          ; INPUT CHARACTER
0120          RET
0130 ;
0140 OUTPUT:  PUSH  AF           ; SAVE CHARACTER
0150          IN    A,0
0160          BIT   TBE,A
0170          JR    Z,OUTPUT+1
0180          POP   AF           ; RETRIEVE CHARACTER
0190          OUT   1,A
0200          RET
0210 RDA:     EQU   6
0220 TBE:     EQU   7
```

This formatted listing of the assembly language source code is produced by ROS following the FORM and LIST commands as shown.

The assembly language program shown on the preceding page is composed in the following way. Each line of the assembly language code is made up of as many as five separate items. The first item is the line number. In AUTO mode ROS automatically supplies sucessive line numbers as we enter the program. The second item that may appear on a line is the label. If the line does have a label it is always followed by a colon. The third item that may appear is the instruction mnemonic. The mnemonics for the various Z-80 instructions can be found in the Z-80 CPU TECHNICAL MANUAL published by Mostek and Zilog.* The fourth item that may appear on a line is the operand or operands of the instruction. The first operand to appear must be separated from the instruction mnemonic by at least one space. If there is more than one operand the operands must be separated by commas. The last item that may appear on a line is a comment. A comment must always be preceded by a semi-colon.

Now that we have created a file and entered our assembly language program we are ready to assemble the program. We indicate to ROS that we are finished entering the assembly language program by depressing the ESC or ALT-MODE key on our terminal. To get a formatted assembly output listing we type the command:

FORM

The command to assemble (ASMB) is followed by three parameters to specify: 1)the address at which the machine code is to be executed, 2)the address at which the machine code is to be put after assembly, and 3)an option code. (See Chapter 2 Section 6 of ROS manual for more details). Suppose we wish to have the machine code that results from our assembly be executable beginning at location 0 in memory. Suppose we also wish to have the actual machine code stored at location 0 in memory following assembly. And say we wish a full assembly listing (option 1). Then the command to assemble our assembly language file is given by:

ASMB,0,0,1

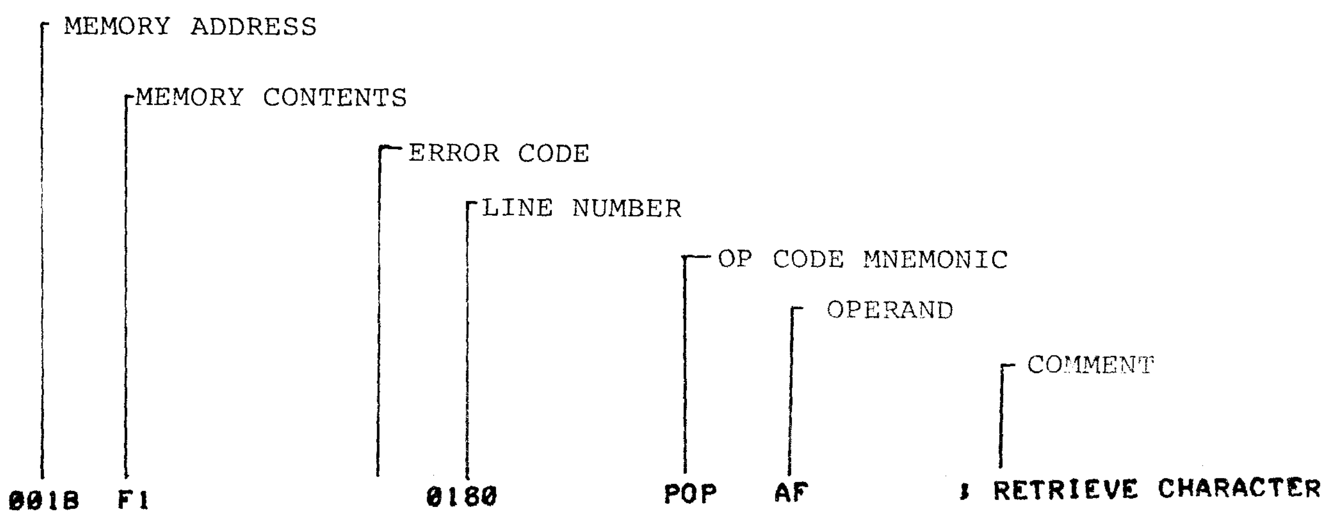
After typing this command from our keyboard the assembly will procede, and an assembly listing will be produced as shown on the following page.

* Note: Some manuals may show the following commands in this way: ADC A,s; ADD A,n; ADD A,r; ADD A,(HL); ADD A,(IX+d); ADD A,(IY+d); SBC A,s; IN A,(n); OUT (n),A. In ROS, shorter versions of these commands are used as listed here: ADC s; ADD n; ADD r; ADD (HL); ADD (IX+d); ADD (IY+d); SBC s; IN A,n; OUT n,A.

```
FORM
ASMB,0,0,1
```

```
CROMEMCO Z-80 ASSEMBLER V.2.0

0000                    0010 ;THIS PROGRAM ECHOS THE KEYBOARD
0000                    0020 ;
0000  31 00 0E          0030          LD     SP,0E00H
0003  CD 0C 00          0040 START:   CALL   INPUT
0006  CD 15 00          0050          CALL   OUTPUT
0009  C3 03 00          0060          JP     START
000C                    0070 ;
000C  DB 00             0080 INPUT:   IN     A,0
000E  CB 77             0090          BIT    RDA,A
0010  28 FA             0100          JR     Z,INPUT
0012  DB 01             0110          IN     A,1          ; INPUT CHARACTER
0014  C9                0120          RET
0015                    0130 ;
0015  F5                0140 OUTPUT:  PUSH   AF           ; SAVE CHARACTER
0016  DB 00             0150          IN     A,0
0018  CB 7F             0160          BIT    TBE,A
001A  28 FA             0170          JR     Z,OUTPUT+1
001C  F1                0180          POP    AF           ; RETRIEVE CHARACTER
001D  D3 01             0190          OUT    1,A
001F  C9                0200          RET
      0006              0210 RDA:     EQU    6
      0007              0220 TBE:     EQU    7
```

ROS produces this assembly listing and stores the machine code object file at the location in memory specified by the ASMB command (in this case location 0). There is a great deal of information on each line of this assembly listing as described on the next page.



This example line from the assembly listing on the previous page shows that there are seven items of information that can reside on each line of the assembly listing. If the assembler detected an error in the composition of the line then an error code would be inserted in the line at the position shown. The following error codes are used by ROS:

Error Code	Description
A	Argument error
D	Double definition
L	Label error
M	Missing label
O	Op-code error
P	Phase error
R	Range error
S	Syntax error
U	Undefined
V	Value error

PROGRAM EXECUTION

After using the ASMB command to assemble this example program, you may wish to execute the program. this can be done by using the ROS command EXEC. Since we put the program at location zero in memory when we used the ASMB command, we would type EXEC,0. This causes an unconditional CALL to location zero saving the ROS return address on the stack. So now let's execute the example program and see if it works:

```
EXEC,0
THIS IS A TEST OF THE PROGRAM "ECHO". AS I TYPE ON THE
KEYBOARD IT IS ECHOED ON THE DISPLAY!!!!
```

SUMMARY

In this Chapter we have given one example of the use of Cromemco's ROS Assembler so that you can start using your copy of ROS right away. The following chapters describe the commands and conventions of ROS in much greater detail, and should answer any questions you had as you worked through this first example.

CHAPTER 2: ROS COMMANDS

Section 1

FILE ORGANIZATION AND FILE COMMANDS

File Organization

Under ROS, user information is organized into files. The files are further subdivided into lines. The user is provided with a set of very comprehensive commands to manipulate his files. Another set of commands - the text editor - allows the user to reorganize the contents of his file, e.g. delete, replace, or insert an individual line. Each line in a memory file must be numbered. The lowest numbered line will always be placed at the beginning of a file, and numbering will continue upward to the end of the file.

File Command Formats

In the formats given for each command the following conventions are used. If an entire word is capitalized, it is a key word and must be used. User supplied information is designated by lower case. A brace { } indicates the user must make a choice. Optional items are enclosed by brackets [ ] . The horizontal ellipsis ... allows the item to be repeated.

Example:

$$\text{CFIL, file-name, beginning-file-address, } \begin{Bmatrix} \text{ending-file-address} \\ \text{file-length} \end{Bmatrix}$$

File Naming

A file name consists of one to six characters and can be any character, (except a control character), for which a code exists. Thus, the file name #@AB41 is legal; however, it is recommended that names descriptive of the file content be used. An entry of a file name longer than six characters results in the first six characters being accepted as the file name. For example, an entry of MYNEWFILE gives MYNEWF as the file name. The number of user files is theoretically only limited by space available in the system RAM area.

Active File - Current File

To avoid both the extra time involved in the user always having to specify which file is being manipulated and the system overhead in searching the RAM area, the concept of a current user file is introduced. Any file may be made current by use of the CURR command. Files are automatically current when they are created.

Optional I/O Drivers

Command, parameters, driver-name

The ROS commands listed below have optional drivers associated with them. If the driver is omitted, then a default is made to SYSØØØ - the system's I/O (console device).

DUMP	LCUS	LIST	TEXT
ECBN	LEAD	RBIN	WBIN
ECHX	LFIL	RCBN	WCBN
ENTE	LIOD	RCHX	WCHX

Example:

```
LIST,10,20,TTY                List on TTY
LIST,10,20,CRT2               List on CRT2
```

Create File

CFIL, file-name, beginning-file-address, $\left\{\begin{array}{l}\text{ending-file-address}\\ \text{file-length}\end{array}\right\}$

The created file is entered in the File Name Table, which resides in the system RAM area. A newly created file becomes the current (active) file. All file commands which do not specifically designate a file default to the new file. After each file creation, the new boundaries of the system RAM are indicated. This reflects an entry into the File Name Table. An attempt to allocate previously assigned memory to a new file will result in the message: "MEMORY ALREADY ALLOCATED".

In several of the files created below some text will be entered. Each text line will be numbered by entering a number followed by a space. Variations on this procedure are given in the section on text commands.

Example:

```
CFIL,A,1000,S1000
D000 D20B
```

A is now the current file. It starts at 1ØØØ hex and is allotted 1ØØØ hex locations; the limits of the file are 1ØØØ to 1FFF. The system responds with the new RAM boundaries DØØØ D2ØB. Next, a new file is created. A is no longer current but remains in the File Name Table and can have data entered at any convenient time by making it current.

Example:

```
CFIL,AIR,2000,2FFF
D000 D217
10 FILE AIR BEGINS AT 2000
20 ENDS AT 2FFF
30 SUBSTITUTION OF S1000 FOR 2FFF
40 GIVES THE SAME RESULT
```

Now an attempt is made to create a new file that extends into a previously allotted area.

```
CFIL,TOMCAT,2500;S300
MEMORY ALREADY ALLOCATED
```

The files previously allotted are intact, and the file TOMCAT is non-existent.

```
CFIL,TOMCATION,3100,31FF
D000 D223
```

The file TOMCAT is now current. The remaining letters ION are ignored.

<u>List File Names</u>

LFIL, [driver-name]

A list of all the files in the File Name Table is provided by entering LFIL. The first file listed is the current (active) file. Each file name is followed by the beginning address of the file, the ending address of the occupied area, and the end of the allocated area. The user can inspect the list of file names to determine if a proposed name is a duplicate. This command provides the user with a map of his files so that the user can decide on future memory allocation via file creates and moves.

Example:

```
LFIL
TOMCAT      3100  3100  31FF
AIR         2000  20FA  2FFF
A           1000  1000  1FFF
```

TOMCAT is the active file, but as yet it is empty. AIR contains FB bytes of information, and its allocated area is from 2ØØØ to 21FF. A, which was allotted 1ØØØ bytes of memory by the swath command S1ØØØ, ends at 1FFF.

Example:

```
LFIL,CRT1                       Files will be listed on CRT1
```

Get Current File

CURR, file-to-be-made-current

Any file may be made current at any time the user is in the command mode. By checking the first line of output from the List File command LFIL, it can be determined which file is current. It usually is faster just to make the desired file current. An attempt to make a non-existent file current will give the message ERROR. After an error message, if it is felt that the syntax of the command was correct, then use LFIL to see if the file already exists.

Example:

```
CFIL,A,1000,S100                A is the current file
DOOO D217
CFIL,B,2000,S1000               B is the current file
DOOO D223
CURR,A                          A is now the current file
```

Delete File

DFIL, file-name

Any file may be deleted by the DFIL command. Files are deleted one at a time. After a DFIL command is issued for the active file, there is no active file. Deletion of a non-existent file gives the message ERROR. Execution of the Delete File command is followed by the system RAM boundaries.

Example:

Assume the files A, AIR, and TOMCAT exist, and that TOMCAT is the current file.

```
DFIL,AIR
D000 D203                       system ram boundaries
DFIL,TOMCAT                     current file is deleted
D000 D1F7
10 SSS                          can not enter text because
NO CURRENT FILE                   no file is current
CURR,A
10 SSS
DFIL,A,AIR,TOMCAT               only A is deleted
D000 D1EB
```

Validate File

VFIL

The validate command performs the following operations:

1) It checks that all lines within a file have a length;
2) It verifies that each line ends with a carriage return;
3) It checks the beginning of each line for a 4 digit line number followed by a space;
4) It certifies that no control characters are part of the text.

When a file passes validation, the name of the file is returned with its starting address, ending address of textual material within the file, and end of the region allocated for the file. If the file contains errors, the byte location of each error is given followed by the message FILE ERROR. It is assumed that typically the user only wishes to validate a current file; therefore, the VFIL is not followed by an operand, a file name.

Example:

```
VFIL
AIR      2000  2072  2FFF            File AIR is ok.
```

Assume at location 2001 the current file contains an error

```
VFIL
2001
FILE ERROR
```

Move File

MFIL, file-name, beginning-address-of-receiving-area

An existing file may be moved to any existing memory location providing the space is not occupied by another file or system information. Attempts to move a file into another file's area will be greeted by MEMORY ALREADY ALLOCATED. After a move, the file will no longer exist at its previous location. The file to be moved need not be active.

Example:

```
MFIL,TOMCAT,0                  The absence of an error message
                               following the move indicates a
                               successful move.
CFIL,STAT,2000,300
D000 D203
MFIL,TOMCAT,2000               Cannot move TOMCAT into area
MEMORY ALREADY ALLOCATED       that is occupied by STAT.
```

Using File Commands

A short demonstration using file commands only follows.

```
CFIL,A,1000,S1000
D000 D1F7                      System ram boundaries
LFIL
A       1000  1000  1FFF       File is empty.
B       2000  200A  2FFF

DFIL,A
D000 D1EB                      system ram decreased by 12 bytes.
CFIL,A,1000,S1000
D000 D1F7
CFIL,B,1000,S1000
DUP. NAMES                     B was already defined.
LFIL
A       1000  1000  1FFF
B       2000  200A  2FFF

CURR,A                         Make A the current file.
DFIL,A
```

```
D000 D1EB
1 ABC
NO CURRENT FILE
CURR,B
1 ABC                              Accepted because there is a current file
CFIL,B,3000,S200
DUP. NAMES                         The file name was already defined
LFIL
B          2000  200A  2FFF

CFIL,T1,3000,S200
D000 D1F7
MFIL,B,0
MFIL,T1,1000               B and T1 are now contiguous at low memory
LFIL
T1         1000  1000  11FF
B          0000  000A  0FFF
```

Section 2

TEXT EDITING COMMANDS

After creation of a file, the text commands allow the user to manipulate the contents of the file. In addition to adding or deleting the lines of a file, each line can be automatically numbered as it enters. Text lines can also be renumbered. Whether a listing will be formatted or unformatted is controlled by a flag in the monitor using the FORM and NFOR commands. To understand the effect of formatting using tabs see the section on the LIST command. Examples of assembly language will be presented in this section. For assembly language conventions see the section on assembly language.

List

LIST, [beginning-line-number], [ending-line-number], [driver-name]

In the absence of parameters the entire contents of the active file are listed when the LIST command is used. If the formatting flag is set, then the list is formatted according to tab settings for the I/O driver. The section on I/O commands covers the setting of the tabs. When tabbing is used in the example below, assume the conventions given below.

Example:

FIELD TYPE	LABEL	OPERATOR	OPERANDS	COMMENTS
COLUMN -	1	9	15	25
CONTENTS -	START:	LD	HL,START	;LOAD HL

When LIST is followed by one line number only, the indicated line is listed. If two parameters - line numbers - follow LIST, all the text lines from the first line number to the second line are listed.

Example:

```
CURR,B
1 SSS
LIST
0001 SSS                         Notice, left zero fill is automatic
```

```
CURR,AIR
1XYZ80                          The presence of a non-numeric character
LIST                            Signals the end of a line number
0001 XYZ80

CURR,B
10 LD A,B
20 START:LD HL,START
30 JP START;JUMP TO START
LIST                            Assume no fromatting
0001 SSS
0010 LD A,B
0020 START:LD HL,START
0030 JP START;JUMP TO START

FORM                            Turn on the formatting switch
LIST
0001            SSS
0010            LD      A,B
0020 START:     LD      HL,START
0030            JP      START        ;JUMP TO START

LIST,10
0010            LD      A,B

LIST,10,10
0010            LD      A,B

LIST,1,20
0001            SSS
0010            LD      A,B
0020 START:     LD      HL,START
```

List Without Numbers

TEXT, [beginning-line-number], [ending-line-number], [driver-name]

TEXT only differs from LIST in that line numbers are not printed.

Example:

```
TEXT,1,20                       Assume no formatting
SSS
LD A,B
START:LD HL,START
```

```
FORM                                   Turn on the formatting switch
TEXT,20,30
START:  LD     HL,START
        JP     START         ;JUMP TO START
```

Format Switch On

FORM

The FORM command turns on the format switch. This switch activates the tabbing associated with each I/O driver. The FORM command affects the LIST and TEXT commands and all assembler commands such as ASMB. The tabs can be changed by using the IODR command. Other selected I/O commands affect the tabbing by resetting tabs, e.g. SYSI. The FORM command is regional, that is it remains in effect until the occurrence of NFOR command. Further discussions of tabbing are covered under Assembly Language commands and the LIST command.

Format Switch OFF

NFOR

The NFOR command deactivates the use of tabbing.

Examples of FORM and NFOR:

```
FORM
LIST,40,60
0040 BRNCH1:  CALL   START
0050 BRNCH2:  JP     START
0060          LD     A,B          ;Z-80

NFOR
LIST
0040 BRNCH1:CALL START
0050 BRNCH2:JP START
0060 LD A,B;Z-80
```

Type Numbers Automatically

AUTO, [lowest-line-number], [increment], [maximum-line-number]

The AUTO command is provided to relieve the user of having to enter line numbers. Four digit line numbers are automatically entered on the left margin by the AUTO command. The user specifies the starting number, the increment size, and the maximum line number. Any numeric value can be entered for any of three parameters. The default parameters are one for the starting number, one for the increment parameters, and 9999 for maximum line number. If the start number exceeds the maximum line, only one line will be printed and wrap around will not occur.

Example:

```
AUTO,3,7,20
0003 LD A,B
0010 START:LD HL,START
0017 JP START
AUTO MODE COMPLETE
```

The line numbers ØØØ3, ØØ1Ø, ØØ17 followed by a blank are printed by the monitor; the user then enters the text.

Example:

```
AUTO,40,10,60
0040 CALL START
0050 JP START
0060 LD A,B
AUTO MODE COMPLETE
```

Line number 6Ø was the limit given in the AUTO command so the monitor message indicated completion. If you wish to leave the auto mode before completion, press the ESC or ALT MODE key.

Renumber File

RENU, [starting-line-number], [increment-size]

Line numbers in the current file are renumbered by the RENU command. The user specifies the starting number and the increment size. The starting number is a line number from 1 to 9999, and the increment size ranges from 1 to 25. When the renumbering reaches 9000, the increment size is 1. Wrap around can occur when the line number reaches 9999; the next line numbers then will be 0, 1, 2, etc. It is possible to have two lines with the same number. Omission of either starting-line-number or increment-size or both causes a default to 1.

Example:

```
LIST
0006 LD A,B
0023 JP AGAIN
0042 CALL SUBX

RENUMBER,20,15
LIST
0020 LD A,B
0035 JP AGAIN
0050 CALL SUBX
```

<u>Delete</u> <u>Lines</u>

DELE, beginning-line-number, ending-line-number

With the DELE command all lines are deleted from the first line number to the second line number, inclusively.

Example:

DELETE,35,95	THE LINES FROM 35 TO 95 INCLUSIVE WILL BE DELETED
DELE,20	LINE NUMBER 20 WILL BE DELETED
DELE,20,15	NO LINE WILL BE DELETED

Section 3

I/O - INPUT/OUTPUT COMMANDS

The majority of commands in this section are related to routines called drivers. These routines contain instructions that allow data to be transferred in or out of the computer memory. The user is able to change selected parameters relating to drivers. The input and output addresses can be modified. The display of text through tabs and page size is alterable by the user.

Three types of data representation are provided for, namely 1) an unmodified binary representation of memory contents, 2) INTEL hexadecimal, and 3) INTEL binary. In many of the examples given below reference is made to the I/O Driver Table and its parameters. These parameters are covered in some detail under LIOD - List I/O Drivers.

List I/O Drivers

LIOD, [driver-name]

A table of I/O assignments is kept in the system RAM area, sometimes referred to as the I/O Driver Table. An entry of LIOD will produce a listing of the table of I/O assignments. The example below explains each parameter.

LIOD

SYSØØØ	6F36	6F3F	Ø	6Ø	6	9	15	25
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)

(1) Driver name; in the example SYSØØØ is the system driver.
(2) Input driver address.
(3) Output driver address.
(4) Number of nulls between each line. This allows time if needed for a line feed to take place before printing the next character on a hard copy device, e.g. teletype.
(5) Number of lines per page - used for assembler paging.
(6) Number of lines between pages - used for assembler paging. If this number is Ø, a form feed is issued to advance to the top of the next page; otherwise, this is the number of line feeds that are issued to advance to the top of the next page.
(7) Beginning column number of operation instruction, pseudo-op, etc. See explanation under LIST command.

(8) Beginning column number of operand.
(9) Beginning column number of comment.

Further examples of LIOD are shown under IODR and SYSI commands.

Define I/O Drivers

IODR, see LIOD for complete description of parameters

Drivers may be either added or modified by the IODR command. Driver names may be from one to six alphanumeric characters. Omitted parameters are indicated by two adjacent commas or terminating the driver definition before all parameter positions are indicated. Immediately following the I/O driver assignment the boundaries of the system RAM are given; this occurs because the assignment of a new driver will expand the system RAM area.

Example:

```
LIOD
SYS000    AF38   AF41    0   60   6   9   15   25
IODR,DISKI,8F00
D000 D22B
```

A new driver DISKI is defined in the example above. The input driver address is 8F00. The absence of parameters between the commas indicates defaulting to the system output driver address of AF41. The number of nulls after a line feed is five. All the remaining parameters will be that of the system. To verify the assignment the I/O assignments are listed below.

Example:

```
LIOD
SYS000    AF38   AF41    0   60   6   9   15   25
DISKI     8F00   AF41    0   60   6   9   15   25
IODR,TTY                 Assign I/O driver TTY, default all
D000 D23B                parameters to system - SYS000.
LIOD
SYS000    AF38   AF41    0   60   6   9   15   25
DISKI     8F00   AF41    0   60   6   9   15   25
TTY       AF38   AF41    0   60   6   9   15   25
```

Delete I/O Driver Name

DIOD, driver name

The DIOD command deletes one driver from the I/O driver table each time it is used. SYSØØØ cannot be deleted. The system RAM boundaries are given after each successful deletion.

Change SYSIO to Name

SYSI, [driver name]

The system I/O driver - the console device whose logical name is SYSØØØ - will have its parameters changed to the I/O driver name following the command SYSI. An entry of only SYSI returns the system I/O driver to the parameters stored in the PROM. The examples given below are a continuation of the results in previous section on Define I/O Drivers.

Example:

SYSI,TTY	SYS000 will have the same parameters as TTY
IODR,TTY,,,,,0	Change number of lines between pages from 6 to 0 for TTY. In affect delete the line feeds and substitute a form feed.

LIOD

SYS000	AF38	AF41	0	60	6	9	15	25
DISKI	8F00	AF41	0	60	6	9	15	25
TTY	AF38	AF41	0	60	0	9	15	25

Note difference between SYS000 and TTY.

SYSI,TTY

LIOD

SYS000	AF38	AF41	0	60	0	9	15	25
DISKI	8F00	AF41	0	60	6	9	15	25
TTY	AF38	AF41	0	60	0	9	15	25

There is now no difference.

SYSI — Reset SYS000 to parameters in the prom.

LIOD

SYS000	AF38	AF41	0	60	6	9	15	25
DISKI	8F00	AF41	0	60	6	9	15	25
TTY	AF38	AF41	0	60	0	9	15	25

Write Leader

LEAD, [driver-name]

Following the issuing of the LEAD command, there is a five or ten second wait, which allows time to turn on the punch; then 60 characters of leader are punched. The wait is five seconds at four magahertz and ten seconds at two megahertz. After punching the leader, control transfers immediately to the system. The user is advised to turn off the punch unit to avoid punching unwanted characters, such as control instructions, on the tape.

Read and Write Commands

In reading a tape the operator places the first character to be read directly over the read sprocket. At the conclusion of all reads the teletype may generate extra characters. These characters can be prevented from becoming a part of a memory file by pressing control X of the teletype.

A five or ten second wait occurs after entry of any write command. This allows the operator time to turn on the punch. The wait is five seconds at four megahertz and ten seconds at two megahertz.

Tape Formats

There are three tape formats: 1) binary without a checksum, 2) binary with a checksum, and 3) hexadecimal. One frame on a binary tape represents one byte from memory; thus, a frame contains two hexadecimal characters. The contents of a binary tape can be read directly into memory and used without conversion. A hexadecimal tape uses two frames per byte of memory. The hexadecimal tape is in ASCII format and can be interpreted off-line by a hard copy device. The contents of this tape cannot be used by a computer without conversion to hexadecimal.

The data on a hexadecimal tape is blocked into discrete records, each record containing record length, record type, memory address, and checksum information in addition to data. A frame-by-frame description is as follows:

Frame 0	Record Mark: Signals the start of a record. The ASCII character colon (":" HEX 3A) is used as the record mark.
Frames 1, 2 (0-9, A-F)	Record Length: Two ASCII characters representing a hexadecimal number in the range of 0 to 'FF' (0 to 255). This is the count of actual data bytes in the record type or check-sum. A record length of 0 indicates end of file.
Frames 3 to 6	Load Address: Four ASCII characters that represent the initial memory location where the data following

	will be loaded. The first data byte is stored in the location pointed to by the load address; succeeding data bytes are loaded into ascending addresses.
Frames 7, 8	Record Type: Two ASCII characters. Currently, all records are type 0. This field is reserved for future expansion.
Frames 9 to 9+2* (Record Length) -1	Data: Each 8 bit memory word is represented by two frames containing the ASCII characters (0 to 9, A to F) to represent a hexadecimal value 0 to 'FF'H (0 to 255).
Frames 9+2* (Record Length) to 9+2* (Record Length) +1	Checksum: The checksum is the negative of the sum of all 8 bit bytes in the record since the record mark (":") evaluated modulus 256. That is, if you add together all the 8 bit bytes, ignoring all carries out of an 8-bit sum, then add the checksum, the result is zero.

For example, if memory locations one through three contain 53F8EC, the format of the hex file produced when these locations are punched is:

:0300010053F8ECC5

A CROMEMCO binary tape with checksum is the same as the hexadecimal tape described above except, as mentioned, each frame contains one memory byte.

The following read-write commands are grouped as follows:

WBIN, RBIN	Binary Format without checksum
WCHX, ECHX, RCHX	Intel Hexadecimal
WCBN, ECBN, RCBN	Binary Format with checksum

Write Binary Tape

WBIN, starting-address-in-memory, {ending-memory-address / S length} [driver-name]

A specific number of bytes starting at a given memory location are written on tape by the WBIN command. Each frame on the tape will be an unmodified image of each byte from memory.

Example:

WBIN, 1000, S20	Write 20H bytes from memory starting at location 1000H.

Read Binary Tape

RBIN, starting-address-in-memory, $\left\{\begin{array}{l}\text{ending-memory-address}\\ \text{S length}\end{array}\right\}$, [driver-name]

A tape written by the WBIN command is read by the Read Binary Tape command. The contents of each frame will be read directly into memory without modification.

Example:

RBIN, 2000, 204F	Read in 50H bytes starting at memory location 204F.

Write Checksummed Binary Tape

WCBN, starting-address-in-memory, $\left\{\begin{array}{l}\text{ending-memory-address}\\ \text{S length}\end{array}\right\}$, [driver-name]

The WCBN command allows the user to place a checksum at the end of a binary tape. The checksum is generated by summing up all the frames after the record mark.

Write EOF on Checksummed Binary Tape

ECBN, [driver-name]

After the WCBN command, an end-of-file is written by ECBN. The message ":" will be typed. A binary end-of-file cannot be interpreted for hard copy.

Read Checksummed Binary Tape

RCBN, starting-address-in-memory, $\left\{\begin{array}{l}\text{ending-memory-address}\\ \text{S length}\end{array}\right\}$, [driver-name]

The RCBN command allows the user to validate the contents of a binary tape. A tape is validated by summing all the frames after the record mark; carries are ignored. This sum is compared to the checksum written previously at the end of the tape. If the two sums do not match, Write Checksum Hex Tape

WCHX, starting-address-in-memory, {ending-memory-address / S length}, [driver-name]

The WCHX command will write an Intel hexadecimal tape with a checksum. The checksum is generated by summing up all the frames after the record mark. The sum ignores carries and is written as the last frame in the record.

Example:

WCHX, 1000, S20

Thirty-two bytes (20H) starting at location 1000H are written onto a tape. The contents are reformatted into Intel hex code.

Write EOF on Checksummed Hex Tape

ECHX, [driver-name]

After the execution of the WCHX command, an end-of-file is written by ECHX. The message ":ØØ" (which is an end-of-file command in hex) will be typed.

Example:

ECHX
:00

Read Checksummed Hex Tape

RCHX, starting-address-in-memory, {ending-memory-address / S length}, [driver-name]

The RCHX command allows the user to validate the contents of an Intel Hexadecimal tape. A tape is validated by summing all the frames after the record mark; carries are ignored. This sum is compared to the checksum written previously at the end of the tape. If the two sums do not match, the message "CS" is printed on the system I/O device.

Section 4

MEMORY ACCESS COMMANDS

Commands which enter, move, delete, or report on the contents of memory without regard to files or lines are classified as memory access commands.

Dump Memory

DUMP, beginning-memory-address, {ending-memory-address / S length}, [driver-name]

The result of a dump is listed in hexadecimal byte by byte. Each printed line contains a maximum of 16 bytes and is preceded by the memory address of the first byte. The two allowable command formats are demonstrated in the example below.

Example:

```
DUMP,0,S10                              Dump 16 bytes of memory starting at
                                        location 0.
0000: 02 00 01 AB FE C3 F8 FE 00 7B FE 5F 54 41 32 54

DUMP,0,1D                               Dump 1DH bytes of memory

0000: 02 00 01 AB FE C3 F8 FE 00 7B FE 5F 54 41 32 54
0010: 20 00 00 80 44 52 49 56 45 52 3F 6F 80 00
```

Enter Memory

ENTE, beginning-memory-address, [driver-name]

The Enter Memory command allows the user to enter hexadecimal data starting at any memory location. A carriage control does not terminate the Enter Memory mode; thus, the user can continue to enter data line after line. The entry of a one to four digit number followed by colon will enter a new memory starting address. An attempt to enter an illegal byte will be reported as an ERROR after a line feed. All bytes up to the incorrect byte will be accepted. When using this command end the data input with a "/".

Single digit entries are filled with a zero on the left side while an entry of more than two digits results in having the two rightmost digits accepted.

Example:

```
ENTE,0              Start entering data at address zero.
12 2 1415/
DUMP,0,3

0000: 12 02 15              The 2 has had a left zero added, and
                            15 was entered in the third byte.


ENTE,0
12 2 1415
23 44
6 3000: 17 20               17 and 20 will be entered in 3000 and 3001.
/

ENTE,1000
12 23 24 55 1G 23 44 55
ERROR                       1G is an illegal entry

DUMP,1000,S8

1000: 12 23 24 55 00 00 00 00
                            55 was the last legal byte
```

Move Memory

MOVE, beginning-sending-address, $\left\{\begin{array}{l}\text{ending-address}\\ \text{S length of move}\end{array}\right\}$, beginning-receiving-address

The MOVE command moves a byte at a time. If either the sending or receiving field exceeds the highest memory location, wrap around will occur to memory location zero. Any character may be propagated through a section of memory having the receiving address one greater than the address location. After a move, the VMEM command is called automatically.

Examples:

```
MOVE,0,S200,1000          Move 200H bytes from location 0 to location 1000h.

ENTE,0                    Enter 30H at location 0.
30/
MOVE,0,6,1                Propragate contents of location 0 for 6 bytes
```

```
DUMP,0,8

0000: 30 30 30 30 30 30 30 F8 FE
```

Verify Memory

VMEM, beginning-sending-address, {ending-address / S length}, beginning-receiving-address

The VMEM command matches the contents of a series of locations on a byte by byte basis. Whenever a mismatch occurs, the first location is given followed by its contents; followed by the contents of the second location followed by its address. At the conclusion of a MOVE the VMEM command is invoked automatically.

Example:

```
MOVE,0,S1000,1000

VMEM,0,S1000,1000

ENTE,444                Change value of 2 bytes, now areas should not verify
23 45/
VMEM,0,S1000,1000
0444 23 FF 1444         Error in validation.
0445 45 E5 1445
```

Print System RAM Area

PRAM

The bounds of the system RAM area are printed the PRAM command.

Example:

```
PRAM
D000 D23B
```

Select Bank(s)

BANK, value

One or more banks on Cromemco memory boards can be selected with the BANK command. (When ROS is initialized, bank Ø is selected). Bank selection can be altered either with the BANK command or by outputting a byte to port 40_{16}. A particular bank n is selected by entering a byte with bit n high as shown in the table below.

BANK	Output byte or value
Ø	Ø1
1	Ø2
2	Ø4
3	Ø8
4	1Ø
5	2Ø
6	4Ø
7	8Ø

More than one bank may be selected at the same time by adding the values for the banks.

Example:

```
BANK,80          Bank 7 is now on, all others are off
BANK,88          Banks 3 and 7 are on
```

Section 5

CUSTOM COMMANDS

Customizing allows the user to use his own set of mnemonic names. Those names may be one to four characters long. The user can equate any allowable name to a memory location. This name, when entered, becomes a command to begin execution at the designated memory location. Either a user routine or a system routine can be evoked. Customizing also allows the user to add his own name to call a monitor command. The list of custom names is dynamic and may be added to or contracted at any time.

Enter Custom Name

ECUS, custom-name, memory-address = transfer address

ECUS and RENA are the two commands that add custom names. ECUS is used to equate a one to four character custom name to a memory location. Whenever a successful entry is made, the system responds with the new system RAM boundaries.

Example:

ECUS,QUIT,AOOC	AOOC is the reenter address.
DOOO D241	
QUIT	When QUIT is now typed in, ROS now transfers to AOOC.
ECUS,PROCES,1000	The user has a process control program starting at location 1000H.
DOOO D247	
PROCESS	The process control program is entered at 1000.

List Custom Name Table

LCUS, [driver-name]

The LCUS command will list the names in the custom command table. Each name is followed by the transfer address associated with the name. For examples of LCUS see the section on DCUS.

Delete Custom Name

DCUS, custom-name

The DCUS command deletes one custom name from the custom table each time it is used. The system RAM boundaries are given each successful deletion.

Example:

```
LCUS
QUIT   A00C   PROC   1000
DCUS,PROCES
D000 D241
LCUS
QUIT   A00C
ECUS,AP,1500
D000 D247
ECUS,MATHX,2000
D000 D24D
DCUS,SUM
ERROR                             SUM was not in the table.
```

Rename System Command

RENA, system-command, custom-name

A duplicate custom name for a system command is obtained by using the RENA command. The RENA command is particularly useful when used to shorten the name of a frequently evoked routine.

Example:

```
RENAM,DUMP,D              Two names now exist for DUMP
D000 D253
D,1,S2                    D now produces a dump.

0001: 30 55

RENAM,DUMP,#              Special characters are excepted.
D000 D259                 There are now three command that will dump
#,3,S2                    memory - DUMP, D, and #.

0003: 45 98
```

Section 6

ASSEMBLER COMMANDS

Assembler system commands enable the user to allocate memory for the symbol table and to control assembly options. An assembly with options allows the user to define the location of his source code and the destination of his object code.

Three assembly options ASMB, ASMO, and ASMU have the same format. The format is: Command, Parameter 1, Parameter 2, Parameter 3. While Parameter 1 is the origin address of the assembly, Parameter 2 is the actual memory location for the assembled code, and Parameter 3 is an assembly option indicated by a digit 1 to 4.

The options available under Parameter 3 are indicated by the following numbers. Option 1 gives a complete assembled listing. Option 2 will list errors only. Option 3 will print a symbol table after the listing. Option 4 provides a cross reference table in addition to assembly listing. The commands FORM and NFOR are used in conjunction with the formatting of an assembled listing.

Print Symbol Table Allocation

PSTA

The PSTA command lists the beginning and end of the symbol table. At the beginning of a program the symbol table is initialized to start at the end of the system RAM. The upper boundary is at address DFFF. Examples of PSTA are given under the STAB command.

Define Symbol Table Location

STAB

The STAB command can reallocate the symbol table in any memory area not occupied by a file. The first parameter following the command is the new beginning address of the symbol table. If the first parameter is omitted, the beginning address will default to the next memory location after the system RAM. The second parameter must always be given and is either the amount of memory allocated or the upper address.

Example:

```
PRAM                          Print system ram boundaries.
D000 D1EB
PSTA                          Print symbol table boundries.
D1EC DFFF
STAB,0,4FFF
MEMORY ALREADY ALLOCATED
STAB,0,S1000
0000 0FFF                     This area available for symbol table.
STAB,,DFFF                    Default of first parameter is end
D1EC DFFF                     end of system ram.
```

Assemble

ASMB, assembly-origin-addr, addr-assembly-code, assembly-option

The ASMB command assembles without user interaction with respect to the source or object allocation. However, the user does have the option of choosing four types of assembly listing. These options are described in the beginning of this section.

Example:

```
ASMB,1000,4000,1                    Complete assembler listing.
CROMEMCO Z-80 ASSEMBLER V.2.0

1000  78              0010          LD    A,B
1001  81              0020          ADD   C
1002  C3 05 10        0030 LAB1:    JP    LAB3
1005  CD 02 10        0040 LAB3:    CALL  LAB1
1008  21 06 00        0050          LD    HL,6
100B  21 06 00        0060          LD    HL,6H
```

```
ASMB,1000,4000,2                    List errors only
CROMEMCO Z-80 ASSEMBLER V.2.0
```

```
ASMB,1000,4000,3                    Assembler listing and symbol table
CROMEMCO Z-80 ASSEMBLER V.2.0
```

```
1000  78               0010          LD     A,B
1001  81               0020          ADD    C.
1002  C3 05 10         0030 LAB1:    JP     LAB3
1005  CD 02 10         0040 LAB3:    CALL   LAB1
100B  21 06 00         0050          LD     HL,6
100B  21 06 00         0060          LD     HL,6H

SYMBOL TABLE

LAB1     1002  LAB3     1005
```

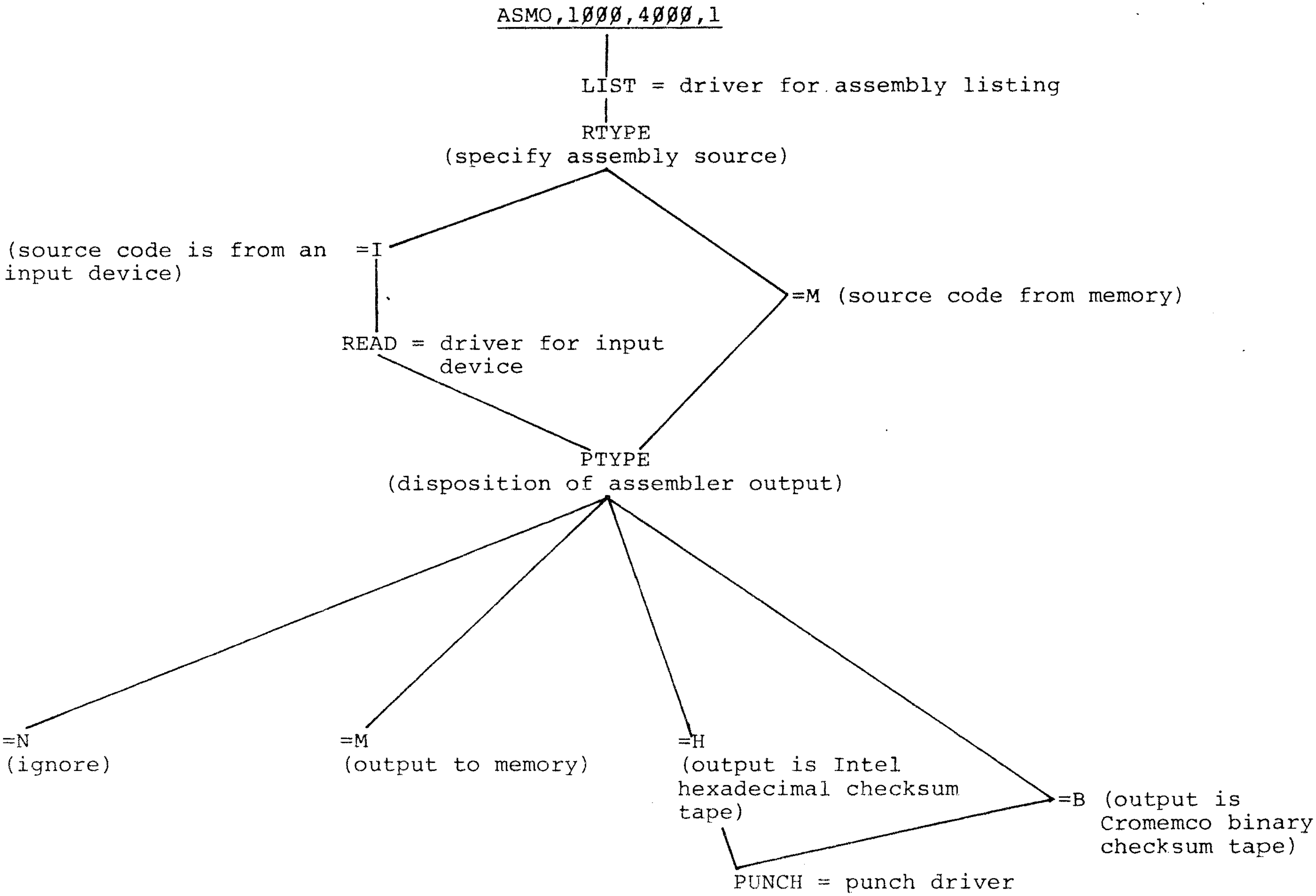
```
ASMB,1000,4000,4                    Assembler listing and crossreference
CROMEMCO Z-80 ASSEMBLER V.2.0

1000  78               0010          LD     A,B
1001  81               0020          ADD    C
1002  C3 05 10         0030 LAB1:    JP     LAB3
1005  CD 02 10         0040 LAB2:    CALL   LAB1
1008  21 06 00         0050          LD     HL,6
100B  21 06 00         0060          LD     HL,6H

CROSS REFERENCE

LAB1     1002     0040
LAB2     1005     0030
```

Assemble with Options

ASMO, assembly-origin-addr, addr-assembly-code, assembly-option

The ASMO command allows the user to specify devices (drivers) for the assembly listing, the assembly source code, and the output driver. The user also specifies the form of the output to a device. The chart on the next page indicates the choices available to the user. The defaults for several options are given below.

Option	Default
LIST=	SYSØØØ
READ=	SYSØØØ
PUNCH=	The driver specified by LIST.


ASMO,1000,4000,1
LIST = driver for assembly listing
RTYPE
(specify assembly source)
(source code is from an input device) =I
READ = driver for input device
=M (source code from memory)
PTYPE
(disposition of assembler output)
=N
(ignore)
=M
(output to memory)
=H
(output is Intel hexadecimal checksum tape)
=B (output is Cromemco binary checksum tape)
PUNCH = punch driver

An attempt to enter an undefined driver will result in the question being repeated. When an output tape is requested by PTYPE, the assembly listing is first listed on the LIST device followed by the punching of the tape.

Example:

ASMO, 4000, 1000, 1	
LIST=TTH	TTH has not been defined as a driver
LIST=TTY	TTY will be the driver for the listing
RTYPE=I	Source will be read from I/O
READ =DISKI	DISKI will be the source driver
PTYPE=N	Object code will not be generated
ASMO, 1000, 4000, 1	
LIST =TTY	
RTYPE=M	Source code is in memory
PTYPE=H	Produce Intel hex tape
PUNCH=TTY	Punch output tape using driver TTY

Assemble Unnumbered I/O File

ASMU, assembly-origin-addr, addr-assembly-code, assembly-option

The ASMU command is identical to the ASMO command except that it will list unnumbered I/O files. Only I/O files may be unnumbered. When an unnumbered file is listed, numbers are placed at the beginning of each line. Numbered files are listed without modification by ASMU.

Section 7

MISCELLANEOUS COMMANDS

Execute at Given Address

EXEC, address

The EXEC command transfers CPU control to the given address by executing an unconditional CALL instruction. A simple return to ROS, the resident operating system, may be made if the user at the end of his subroutine insures 1) that the address popped onto the stack by the CALL is pointed to by the stack pointer, and 2) that the last instruction executed in the subroutine is a return, RET. Performance of the above steps allows execution of the next instruction in the main program.

Burn PROM

PROM, starting-address, $\begin{Bmatrix} \text{end-address} \\ \text{S length} \end{Bmatrix}$ destination address

A 2708 is burned by the PROM command using the Cromemco Bytesaver card. The starting address does not have to begin on a 1K boundary. The resident operating system, ROS, will burn FF_{16} into unused areas. The unused areas are defined to be areas outside of the addressed areas but contained within a 1K block. The FF_{16} and the new data are written to the selected PROM 360 times to insure good programming.

To program a PROM, type the command PROM but do not depress the carriage return. Next, turn the program power switch on the Bytesaver to ON and then type carriage return. The front panel lights will count down. When the light pattern becomes stable, your PROM is programmed. ROS now verifies that the PROM was correctly programmed. Incorrect programming is indicated by displaying the nonverifying addresses and their content in the same format as the VMEM command. Remember to turn the program power switch of the Bytesaver to OFF upon completion of the PROM command.

Example:

```
PROM,1000,S40,6040            Burn 2708 prom

DUMP,6030,S60
```

```
6030: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
6040: 30 30 30 30 30 30 30 30 53 0D 0D 30 30 31 30 20
6050: 4C 44 20 41 2C 42 0D 18 30 30 32 30 20 53 54 41
6060: 52 54 3A 4C 44 20 48 4C 2C 53 54 41 52 54 0D 1D
6070: 30 30 33 30 20 4A 50 20 53 54 41 52 54 3B 4A 55
6080: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

This listing shows that after the PROM command has been executed the PROM memory IC does contain information from address 6040 to address 607F (since in the PROM command it was specified to program a swath just 40 bytes wide). The rest of the PROM has not been programmed at all and thus reads "FF" in each address location, which is the unprogrammed state.

CHAPTER 3: CONVENTIONS AND PSEUDO-OPS

Formats

The most encompassing assembler format is shown below:

```
Formal    Label:    Operation   Operands      ; Comment
Actual    COMP25:   LD          HL,VALUEI     ; Initialize HL
```

The label must be followed by a colon. The colon may be followed immediately by the operation or one or more blanks. Labels need not start in column one.

The maximum accepted length of a label is six alphanumeric characters. All labels must start with an alphabetic character. All labels in the label field must be followed by a colon. A label cannot be a register name.

Correct Labels

```
T12345
A1
T123456  Last character is ignored
```

Incorrect Labels

A	E	SP	HL
B	F	AF	IX
C	H	BC	IY
D	L	DE	R
			I

4A5B Starts with a numeric character

An op-code may be preceded by a label. A space is not required between the label and the op-code. The op-code must be followed by at least one space. The operands must be separated by commas. The length is governed by the type of reference. A reference to a register pair is typically two characters. A label as an operand is up to six alphanumeric characters, and a numeric literal may not exceed FFFF hexadecimal. The op-code of an unlabeled code line may start in column 1.

Example:

```
LABLE:LD HL,14263; 14623 IS BASE TEN AND LESS THAN OFFFFH
```

```
LD HL,14263;OPCODE (OPERATION) STARTS IN COLUMN 1
;           NUMBER IS BASE TEN-DECIMAL
```

All comments must start with a semi-colon. Comments need not be separated from the final operand by a space, although one or more spaces are permitted.

Example:

```
PRT4:OUT   DATA,A;OUTPUT CONTENTS OF ACCUMULATOR
PRT4: OUT  DATA,A      ;SAME EFFECT AS LINE ABOVE
```

Data Representation

Any number is used in assembler code defaults to decimal in the absence of a stated base. A number followed by an H is declared hexadecimal.

Example:

```
LD      A,OFFH          ;LOAD MAXIMUM PERMITTED VALUE - OFFH
                        ;INTO THE ACCUMULATOR
LD      A,255           ;MAXIMUM PERMITTED VALUE IN DECIMAL
LD      HL,OFFFFH       ;MAXIMUM PERMITTED VALUE FOR A
                        ;REGISTER PAIR
```

If a two byte operand exceeds 65,535, then a value of modulus 65,536 is returned without an error flag. Arithmetic expressions are allowed as operands. Computations are performed on both numbers and labels. The operations of addition, subtraction, multiplication, and division are allowed. The expression is evaluated from left to right. The expression 2 + 6 * 2 will evaluate to 16.

Example:

```
LD      B,2+6*2         load b with 16
LD      A,LOC1-LOC2     If LOC1 is twenty locations higher
                        than LOC2 then A is loaded
                        with 20.
```

A "$" references the address of the next instruction

```
        LD      HL,$                 Load HL with address of SETBC
SETBC:  LD      BC,$+15              Load BC with address of COMP+15
```

Assembler Listing Controls

TITLE

Label	Code	Operand
optional:	TITLE	ASCII string

An operand of up to 80 characters will appear as a header on all successive pages until the occurrence of another TITLE command. The ASCII string is not enclosed in quotes. The label field in this command has no effect. TITLE causes an immediate EJECT.

Example:

EJECT

Label	Code	Operand
optional:	EJECT	none

The EJECT command, which advances the paper to the top of the next page, is used for clarity in an assembly listing. A routine can be identified more clearly if preceded by an EJECT.

Data Structure

DEFS - Define Storage

Label	Code	Operands
optional:	DEFS	expression
optional:	DS	expression

The define storage command reserves one or more bytes of storage. The numeric value of the operand determines the number of bytes reserved. Evaluation of the arithmetic expression is from right to left.

Example:

```
BLOCKA: DEFS    20              ;RESERVES 20 BYTES
BLOCKB: DS      20H             ;RESERVES 32 BYTES
```

```
DEFS    LAB1+2*4        ;IF LAB1 IS EQUAL TO 20 THEN
                        ;88 BYTES ARE RESERVED
DEFS    LAB4-LAB3       ;COMPUTE DIFFERENCE OF LAB4 AND LAB3
```

Define Storage (BYTE) DEFB

Label	Code	Operands
optional:	DEFB	expression
optional:	DB	expression

The define byte allows either a numeric expression or an ASCII string to be generated. The numeric expression must be in the range plus or minus 256. However, an ASCII string enclosed in quotes is valid.

Example:

```
DB      'ABCDEF'        ASCII string occupies 6 bytes
DB      'A'+3           The value 44H is generated
DEFB    'ABC'+3         Illegal, arithmetic result too large
DB      'ABC','x'+3     Several fields are allowed
```

Define Storage (WORD) DEFW

Label	Code	Operands
optional:	DEFW	expression
optional:	DW	expression

The define word allows a numeric expression or an ASCII string to be defined. A numeric expression which exceeds 65,535 will be evaluated modulus 65,536; overflow will be ignored. The ASCII string, enclosed in quotes, is limited to one word (two characters). The entire line may be filled with operands.

Example:

```
DW LAB1,LAB2        Labels OK
DW $                Current value of the location counter
```

To clarify the action of the dollar sign, consider the code below:

```
ORG     0
DB      $                    The value of the location counter
                              is zero
```

```
DB      7,$              The second byte has a value of 2

DW      'AA'             Evaluates to 4141H
DW      'A'              Evaluates to 0041H
DW      9,'BD',LAB4      More than one expression permitted
DW      'PDQ'            Illegal, too long
```

Assembly Directives

ORG - Origin

Label	Code	Operand
optional:	ORG	arithmetic expression

The ORG instruction sets the assembler location counter. The counter may be set to a value more than once during assembly.

Example:

```
ORG     100H        ;LOCATION COUNTER IS SET AT 100H
ADD     A           ;THIS INSTRUCTION ASSEMBLED AT 100H
ORG     200H        ;THE LOCATION COUNTER IS NOW 200H
```

EQU - Equate

Label	Code	Operand
label:	EQU	expression

The label field is equated to the operand. An EQU instruction must have a label. The operand, if a label, should be a previously defined label. Any arithmetic expression is allowed. The EQU is global; once a label is defined, it is defined for the entire program.

Example:

```
LAB2:   EQU    LAB1        ;CORRECT IF LAB1 PREVIOUSLY DEFINED
LABLE:  EQU    12*317+4    ;ARITHMETIC EXPRESSION ALLOWED
```

END - End Assembly

Label	Code	Operand
optional:	END	none

The END command simply ends the assembly. A second way to end a program is when reaching the end of a memory file: an end of file terminates assembly. A third way to end a program occurs with I/O files. When an I/O driver is first called, the carry flag is set. The setting of a carry flag indicates a rewind of a file such as a disk file. When the driver reaches the end of the file being processed, it returns a code to set the Z flag (zero flag). When both the carry flag and the zero flag are set, assembly will be terminated.

CHAPTER 4: USEFUL SYSTEM SUBROUTINES

Selected system subroutines can aid the user in his programming. A list of useful system subroutines with short descriptions are given below. Before using any of the routines, index register IX must be loaded with the address of BASE. This address is found under Linkage to Common Routines.

RESTRT

This routine will restart ROS. ROS will be initialized providing that it was not initialized previously. The command mode is entered. This routine does not RETURN.

REENTR

The system is reentered without restarting. This routine is used when the user routines are ended. It does not return to the caller.

CALINT

The resident operating system is initialized and a return is made to the caller.

ACCES

This routine allows program access to all system commands. The HL register points to an input table. Each entry in the table is a command string followed by a carriage return. A byte of zero ends the table. If there is an error, this routine will not return to the caller.

The example below demonstrates a user routine containing two system commands, IODR and EXEC.

Example:

```
CROMEMCO Z-80 ASSEMBLER V.2.0

2000  21 07 20          0001 START:  LD    HL,TABLE
```

```
2003  CD 12 A0       0002          CALL  ACCES
2006  C9             0003          RET
2007                 0004 ;
2007  49 4F 44 52    0005 TABLE:   DB    'IODR,DISKI,FC00',13
      2C 44 49 53
      4B 49 2C 46
      43 30 30 0D
2017  45 58 45 43    0006          DB    'EXEC,A000',13
      2C 41 30 30
      30 0D
2021  00             0007          DB    0
2022                 0008 ;
      A012           0009 ACCES:   EQU   0A012H
```

SYSOUT

This routine is the system output routine. To output a character to the current system output device, load the B register with the character to be sent. Only the AF registers will be altered. This routine will not return if an ESCAPE is read from the input device.

SYSIN

This routine will get a character from the current system input device. The character will be returned in the A and B registers. Only the A, F, and B registers are altered. This routine does not return if an ESCAPE is read from the input device.

P2HEX

The contents of the HL register are printed in hex on the system output device. Only AF and BC are altered.

P1HEX

The contents of the A register are printed in hex on the system output device. Only the AF and BC registers are altered.

P2HEXS

This routine calls P2HEX and then prints a space.

P1HEXS

This routine calls P1HEX and then prints a space.

PRTNUM

This routine will output characters to the system output device. The HL registers are to be loaded with the address of the characters to be printed and the D register loaded with the number of characters to be printed. Only the AF, B, and D registers are altered.

READLN

This routine will read one line from the system input device using all editing features of ROS. The HL register will return pointing to the new line and BC registers will contain the length.

GNAME

This routine gets a six character name from the input line. This routine is used with custom commands to retrieve a name parameter from the input line. On input IY must point to the current position in the command line. This register has already been loaded when the custom command was executed. On output the Z flag will be set if there is a default, DE will point to the six character name padded with blanks, and IY will point to the new position in the line.

SIOTAB

This routine will search the I/O table. The name to be searched for is to be loaded into the DE registers before execution of SIOTAB. On return from the routine the Z flag will be set if found and the HL registers will point to the I/O parameters for the name found.

GTHEXM

This routine is used with the custom commands to retrieve a HEX VALUE parameter from the input line. Before calling the routine, IY must point to the current position in the line. This register has already been loaded when the custom command was executed. On return the Z flag is set if default has occurred. The HL register contains the HEX VALUE. This routine does not return on error.

GTDECM

This routine is the same as GTHEXM except the parameter in the custom command is decimal.

ERROR

This routine prints the word ERROR on system output and then enters the command mode. This routine does not return.

MSGOUT

This routine is used to output a message to the system output device. The HL registers are to contain a pointer to the message. Characters are printed until a carriage return is found. Only the AF, B, HL registers are altered.

PRNTTB

The table pointed to by the HL registers is to be printed on the system output device. The user may want to refer to the section on Table Format.

COMPAR

The HL registers are to be loaded with a pointer to the first argument. The DE registers must point to the second argument. The length of the compare is placed in the B register. When the routine returns, the Z flag will be set if the two arguments were equal. If the first argument was greater than the second argument, the carry flag will be set.

SEARCH

This routine searches the table pointed to by the HL registers. The DE registers point to the name to be found in the table. On return, if the name is found, the Z flag is set and HL points past the name to its parameters. Otherwise, the Z flag is not set and the HL registers point past the last entry in the table.

LOOK

This routine has the same function as SEARCH except that if the name being sought is found, then HL points to the entry in the table.

FILL

Execution of FILL fills each byte in a specified area of memory with the value in the A register. The number of bytes to be filled is given by the value in the BC registers and the starting address is contained in the DE registers.

CLEAR

This routine will clear a specified area of memory by loading spaces into each byte. The number of bytes to be cleared is given by the BC registers. The starting address is contained in the DE register pair.

MBLNK

This routine will move the data starting at a location pointed to by the HL registers to the area pointed to by the DE registers for a length specified by BC or until a delimiter is encountered. The Z flag will be set if a delimiter stops the move. System routine CDILM lists the delimiters.

SBLNK

This routine increments the HL registers until they do not point to a space.

SCHAR

This routine increments the HL registers until a delimiter is encountered. The delimiters are:

; , : + - / *)

plus space and carriage return.

CMBLNK

This routine calls CLEAR and MBLNK.

CNUM

This routine checks the A register for a numeric character. The carry is set if not numeric.

GETHEX

A hexadecimal number is fetched from memory and entered into registers DE. The first byte of the number is pointed to by registers BC and the byte following the number is pointed to by registers HL. If an error occurs, e.g. a number that is not a valid hexadecimal number is encountered, the carry flag is set.

GDECM

A decimal number is fetched from memory and entered into registers DE. The first byte of the number is pointed to by registers BC and the byte following the number is pointed to by registers HL. If an error occurs, e.g. a number that is not a valid decimal number is encountered, the carry flag is set.

LEADER

Seventy nulls are written to the system output device after a five second wait.

EINTEL

In this routine an end of file is written for an INTEL format tape. Carry prime, in the auxiliary flag register - F', must be set for hex tape. If carry prime is not set, then a binary end of file will be generated.

CHKCUR

This routine checks to see if a current input file is present. If no input file is present, the message: "NO CURRENT FILE" will be typed, and control will be returned to ROS. Otherwise, the routine will return to the user.

WINTEL

In this routine an Intel format tape is written. On entry, register D contains the record length, HL contains the address, IY points to stored data, and carry prime is set if hex data is used and reset, Ø, if binary data is used.

PRTONE

In this routine one line of data is printed using assembler tabs. On entry HL points to the line. If the carry bit is set, the text without a line number is printed. When the carry bit is reset, Ø line numbers are printed with the text.

GTSTNG

In this routine a string of characters is obtained by calling CMBLNK. Refer to CLEAR and MBLNK for additional parameter information. HL is then incremented until pointing at a comma or carriage return. Also, the routine puts the contents of HL in IY.

INTTAB

This routine initializes the routine GTENT. When entered, HL must point to the table.

GTENT

In this routine an entry is obtained from a table whose position is pointed to by HL. HL returns pointing to the next entry in the table. The Z flag is set at the end of the table.

FUPACK

In this routine the four packed decimal digits in the DE registers are unpacked into the area pointed to by the HL registers.

AFPACK

The four decimal digits in the DE register are added to the four decimal digits in the HL registers. The result is left in HL and the carry is set if the result is greater than 9999.

FPACK

The four decimal digits pointed to by the HL registers are packed into the DE registers.

CDILM

This routine checks a specific byte to see if it is a delimiter. The delimiters are:

; , : + - / *)

plus space and carriage return. HL is loaded with the pointer to the character to be tested. The Z flag will be set if the character is a delimiter.

ADDAHL

The A register is added to the HL registers. The result is left in the HL registers, and the carry flag will be set if overflow occurred.

SPACnn

This set of routines will print nn spaces to the system output device. Only the AF and B registers are changed.

CHAPTER 5: WRITING I/O ROUTINES

The IODR command may be used to change I/O drivers. The input driver address and output driver address are the first and second parameters following the driver name (see List I/O Drivers). By changing the parameter addresses the user may reference his own I/O drivers.

System Input Drivers

A standard input driver routine first checks to see if a character is ready to read. If a character is not available, the A register is zeroed, the carry flag cleared, and the routine returns. If there is a character available, it will be read into the A register and the carry flag will be set. All registers except AF must be preserved. A return is now made to the system. The example below shows a system input driver.

```
AF38               0001 ;
AF38               0002 ;STANDARD INPUT DRIVER
AF38               0003 ;OUTPUT - CARRY SET IF CHARACTER
AF38               0004 ;         A CONTAINS CHARACTER
AF38               0005 ;
AF38    DB 00      0006 INPUT:  IN     A,0        ;GET STATUS
AF3A    E6 40      0007         AND    40H        ;CHECK FOR CHARACTER
AF3C    C8         0008         RET    Z          ;NO CHARACTER
AF3D    DB 01      0009         IN     A,1        ;INPUT CHARACTER
AF3F    37         0010         SCF               ;SAY GOT CHARACTER
AF40    C9         0011         RET
```

Assembler Input Drivers

An assembly input driver differs from a system input driver in handling flags and in accepting input from an external device as described below. If on entry to the input driver the carry flag is found to be set, a rewind of the input file is to be executed. For example, if the input file is paper tape, the tape will be started over again. The input routine does not return until a character is received. The character is read into the A register, then the Z flag is cleared, and the routine returns. When an end of file is sensed, the Z flag is set before a return. All registers must be preserved.

In the input example below a carry flag is not used. When using the teletype, the operator knows where to start loading the tape; rewinding is not possible on a teletype, so a flag is superfluous. However, a set carry flag could have been used to display a message. On the other hand, if the file were a disk file, the carry flag could be used to rewind the file. In the example a control Z, 1A hexadecimal, is used to indicate the end of the file. If the END pseudo-op code is used in the source code, a control Z is not necessary.

Example:

```
CROMEMCO Z-80 ASSEMBLER V.2.0


1000                 0001 ;
1000                 0002 ;TELETYPE INPUT DRIVER FOR ASSEMBLER
1000                 0003 ;INPUT - CARRY SET TO REWIND FILE
1000                 0004 ;OUTPUT - A CONTAINS CHARACTER
1000                 0005 ;          Z FLAG SET IF END OF FILE
1000                 0006 ;
1000  DB 00          0007 INTTY:  IN    A,0        ;GET STATUS
1002  E6 40          0008         AND   40H        ;CHECK FOR CHARACTER
1004  28 FA          0009         JR    Z,INTTY    ;NOT READY
1006  DB 01          0010         IN    A,1        ;GET CHARACTER
1008  FE 1A          0011         CP    1AH        ;CHECK FOR END OF FILE
100A  C9             0012         RET
```

<u>Output Drivers</u>

The output driver expects the character to be written to be in the B register. When the output driver returns, the A and B registers should both contain the output character. All other registers must be preserved.

Example:

```
AF41                 0001 ;
AF41                 0002 ;STANDARD OUTPUT DRIVER
AF41                 0003 ;INPUT - B CONTAINS CHARACTER
AF41                 0004 ;OUTPUT - A AND B CONTAIN CHARACTER
AF41                 0005 ;
AF41  DB 00          0006 OUTPUT: IN    A,0        ;GET STATUS
AF43  E6 80          0007         AND   80H        ;GET TBE
AF45  28 FA          0008         JR    Z,OUTPUT   ;LOOP UNTIL READY
AF47  78             0009         LD    A,B        ;GET CHARACTER
AF48  D3 01          0010         OUT   1,A        ;OUTPUT CHARACTER
AF4A  C9             0011         RET
```

APPENDIX A

Custom Commands with Parameters

Custom-name, [Parameter-1], [Parameter-2] . . . Input line

Before accessing the contents of the parameters listed above the user first equates his custom-name to the entry point of a routine. When the custom-name is executed, a call is made to the user routine, and register IY will point to the first parameter in the input line. The user may now call system subroutines (see Useful System Subroutines). The system subroutines can perform tasks such a checking the existence of the parameters or retrieving the contents of a parameter. Before using any of the system subroutines, IX must point to BASE. The address of BASE is obtained from Linkage to Common Routines (list is given in Appendix G).

APPENDIX B

Using Parameters in the Command Line

In the code shown below the user ultimately references a routine EXAM which will receive parameters from the command line. EXAM uses several system routines. The first routine attempts to find the location of the parameters pointed to by the IY register. If the parameter is not found, an error message routine is called. The third routine retrieves the contents of the parameter. In the sequence of events the user first loads, perhaps via paper tape, a routine PLOT into the starting memory address 1ØØØH. Then the custom command PLOT is equated to the location 1ØØØ.

ECUS, PLOT, 1ØØØ

Eventually, the user executes the command PLOT

PLOT, X, Y

When PLOT is called, the IY register will point to the first parameter, X. The routine PLOT contains two calls to EXAM.

```
PLOT:   CALL    EXAM            ;IY POINTS TO COMMAND LINE
        LD      (SAVEX),HL      ;SAVE VALUE FOR X
        CALL    EXAM
        LD      (SAVEY),HL      ;SAVE VALUE FOR Y
        CALL    DAZLER          ;PUT DOT ON DAZZLER
        RET                     ;RETURN TO ROS
```

The subroutine EXAM is given below. The circled numbers refer to commentary about each instruction.

```
EXAM:   CALL  GNAME     ;GET NAME PARAMETER      1)
        JR    Z,ERROR   ;DID NOT FIND NAME       2)
        CALL  GTHEXM    ;GET HEX VALUE           3)
```

```
         JR    NZ,EXM300 ;GOT HEX VALUE          4)
         LD    HL,0      ;DEFAULT VALUE          5)
EXM300:  LD    (SAVE1),HL;SAVE VALUE             6)
         RET                                     7)
;
SAVE1:   DS    2         ;SAVE AREA              8)
GNAME:   EQU   0A02DH    ;GET NAME ROUTINE       9)
ERROR:   EQU   0A039H    ;ERROR ROUTINE         10)
GTHEXM:  EQU   0A033H    ;GET HEX ROUTINE       11)
```

1) A call is made to the system routine GNAME. Before the call the user equates 9) GNAME to the address found under Linkage to Common Routines in appendix H. The GNAME subroutine, as described under Useful System Subroutines, will attempt to find the address of the parameter pointed to by IY. If the search is successful, DE will contain the address of the parameter, and the IY pointer is advanced to the next parameter. An unsuccessful call is indicated by the Z flag.

2) When the Z flag is set, a jump is performed to the system subroutine, ERROR. The ERROR subroutine prints or displays the message "ERROR".

3) If a parameter is a hexadecimal value, the GTHEXM will place this value in the HL register. Failure to return the value is indicated by setting the Z flag.

4) Jump to EXM3ØØ if hexadecimal value is returned to HL.

5) A hex value was not returned to set HL to zero to signify failure to user.

6) The HL register is freed for other uses by transferring the hexadecimal parameter to memory location SAVE1.

7) Return to caller.

9), 10), 11) Establish address for all system routines used by using the Linkage to Common Routines table.

APPENDIX C

User Loading Instructions

I/O

ROS has the unique feature of initializing the baud rate of your I/O board. If you have a Cromemco TU-ART serial I/O board, ROS will initialize your I/O for baud rates of 96ØØ, 24ØØ, 3ØØ, 15Ø, or 11Ø. Other manufacturers have I/O boards which have software control of baud rates, consult their user manual to find out if they have this capability. When ROS is initialized, hit the carriage return key several times until the ROS message is printed. This allows ROS to determine the correct baud rate. The I/O board which you use must conform to the drivers which can be found in Chapter 4.

PROM

First load the eight PROMs into your Cromemco BYTESAVER, making sure that you get the PROMs correctly placed. These PROMs have been preprogrammed and contain the Resident Operating System, ROS. Address your BYTESAVER at location ØAØØØH; this is done by using the DIP switch. For technical details refer to the BYTESAVER instruction manual.

On your Cromemco ZPU card install a jumper wire connecting the two pins marked "jump enable". Set the jump address switch to A. By following the instructions in the next paragraph an automatic transfer will be made to the ROS. The jump enable section in your Cromemco ZPU manual gives complete details on the automatic jump feature.

Insert the BYTESAVER and ZPU cards into the computer. Turn the power on and depress the run switch. When either the power is applied to the system or reset is depressed, control will be transferred to the Cromemco ROS. Depress the carriage control several times until the message: "CROMEMCO ROS V.2.0." is displayed.

PAPER TAPE

Appendix I gives the full instructions for loading the Cromemco ROS from a paper tape. The paper tape has been supplied in Cromemco's binary checksummed tape format to insure high reliability.

APPENDIX D

Special Functions of Keys

ESCAPE	When this key is depressed during either input or output, any I/O is ceased and ROS enters the command mode.
ALT MODE	This key has the identical function as ESCAPE.
Control S	This key only has an effect during output. When depressed, the output printing will be stopped. To resume printing, depress <u>any</u> key.
RUBOUT	This key deletes the previous character when inputting. On a TTY a back arrow will be printed. On some CRTs an underline will be printed.
SHIFT O (back arrow)	This key has the same function as RUBOUT.
Control X	This key will delete the line that is being inputted. A carriage return and a line feed will occur.

APPENDIX E

Error Messages

ERROR	This is a general message for any error condition not covered by more specific error messages.
FILE ERROR	This message is given by VFIL command to say that the file contains an error. Example: 3843 FILE ERROR
FILE FULL	This message is given when the current file cannot contain the new line.
NO CURRENT FILE	This message is given when an operation which automatically references a current file is tried when no file has been made current.
FILE TOO LARGE	This message is given by the VFIL command to indicate that the file is larger than the space allocated for it.
DUP. NAMES	This message occurs when trying to create a new file with a name already used by a previous file.
NO MORE ROOM	This message is given when there is no room left in the system RAM.
OK	This message is received after the paper tape is read correctly.
CS	This is received when a checksum error is detected from the paper tape record.

M	This message is received when a memory error occurs while reading checksummed tape.
SYMBOL TABLE FULL	This message is given by the assembler when no space remains for an entry in the symbol table.

APPENDIX F

Table Format

Whenever a system subroutine uses a table, a particular format is followed. It is important for the user to understand this format when using a system subroutine. Some of these subroutines are PRNTTB, SEARCH, LOOK, INTTAB, and GTENT. The first byte of the table contains the length of the compare argument. The second byte of the table contains the length of an entire entry. The table is ended with a byte of zero.

Example:

```
TABLE:    DB  7,8              1)
          DB  'ENTRY 1',7      2)
          DB  'ENTRY 2',8      3)
          DB  Ø                4)
```

1) Each argument is seven bytes long, e.g. 'ENTRY 1' is seven bytes. Each entry is eight bytes. The seven adds one byte.

2), 3) Two seven byte arguments and their corresponding values seven and eight.

4) End of table.

APPENDIX G

System RAM

```
SYSTEM RAM

BFFE               2443          ORG     0D000H
D000   0040        2444 RAM:     DEFS    64          ;STACK AREA
       D040        2445 STACK:   EQU     $
       D040        2446 SYSRAM:  EQU     $           ;START OF SYSTEM RAM
D040   0014        2447 TEMP:    DEFS    20          ;TEMP AREA (LEAVE PRIOR TO RBUF
D054   0005        2448          DEFS    5           ;AREA FOR NUMBER OF LINE
D059   0053        2449 RBUFF:   DEFS    83          ;READ BUFFER
       D0AC        2450 CURIO:   EQU     $           ;CURRENT I/O PARMS
D0AC   0002        2451 IDRIVE:  DEFS    2           ;CURRENT INPUT DRIVER
D0AE   0002        2452 ODRIVE:  DEFS    2           ;CURRENT OUTPUT DRIVER
D0B0   0001        2453 NULLS:   DEFS    1           ;NUMBER OF NULLS
D0B1   0001        2454 LINES:   DEFS    1           ;NUMBER OF LINES/PAGE
D0B2   0001        2455 TERMWD:  DEFS    1           ;TERMINAL WIDTH
D0B3   0001        2456 TAB1:    DEFS    1           ;TABS FOR ASSEMBLER
D0B4   0001        2457 TAB2:    DEFS    1
D0B5   0001        2458 TAB3:    DEFS    1
D0B6   0002        2459 CURADR:  DEFS    2           ;CURRENT ADDRESS IN FILE
D0B8               2460 ;
D0B8   0002        2461 CURLEN:  DEFS    2           ;CURRENT LENGTH
D0BA               2462 ;
       D0BA        2463 BASE:    EQU     $           ;BASE FOR IX
D0BA               2464 ;
       0000        2465 STATUS:  EQU     0           ;STATUS BYTE
D0BA   0001        2466          DEFS    1
D0BB               2467 ;
       0000        2468 BFORM:   EQU     0           ;FORM FLAG
       0001        2469 CURFLE:  EQU     1           ;CURRENT FORM FLAG
       0002        2470 SYM:     EQU     2           ;SYMBOL TABLE FLAG
D0BB               2471 ;
D0BB               2472 ;USER AREA
D0BB               2473 ;
D0BB   0100        2474 USER:    DEFS    256
D1BB               2475 ;
       DFFF        2476 ENDRAM:  EQU     0DFFFH      ;END OF SYSTEM RAM
D1BB               2477 ;
```

```
D1BB   0002             2478 FLTBPT:  DEFS   2          ;PTR TO FILE TABLE
D1BD   0002             2479 IOTBPT:  DEFS   2          ;PTR TO I/O TABLE
D1BF   0002             2480 CUTBPT:  DEFS   2          ;PTR TO CUSTOMER TABLE
D1C1   0002             2481 SZTBPT:  DEFS   2          ;PTR TO ASSEMBLER SYMBOL TABLE
D1C3   0002             2482 SZTEND:  DEFS   2          ;END OF ALLOCATION, SYMBOL TABL
D1C5   0002             2483 TABEND:  DEFS   2          ;END OF TABLES
D1C7                    2484 ;
D1C7   0003             2485 FLTBST:  DEFS   3          ;INITIAL FILE TABLE
D1CA   0013             2486 IOTBST:  DEFS   19         ;INITIAL I/O TABLE
D1DD   0003             2487 CUTBST:  DEFS   3          ;INITIAL CUSTOMER TABLE
D1E0   0003             2488 SZTBST:  DEFS   3          ;INITIAL ASSEMBLER SYMBOL TABLE
D1E3                    2489 ;
D1E3                    2490          ORG    FLTBST+2   ;CURRENT FILE AREA
D1C9   0006             2491 CURFIL:  DEFS   6          ;CURRENT FILE NAME
D1CF   0002             2492 CFSADR:  DEFS   2          ;CURRENT FILE START ADDRESS
D1D1   0002             2493 CFLEND:  DEFS   2          ;CURRENT FILE END ADDRESS
D1D3   0002             2494 CFLALL:  DEFS   2          ;CURRENT FILE ALLOCATION ADDRES
```

APPENDIX H

Linkage to Common Routines

LINK TO SYSTEM

```
D1D5             ;
D1D5             ;LINKAGE TO SYSTEM
D1D5             ;
D1D5                         ORG    START
A000   0003      RESTRT:     DEFS   3        ;RESTART SYSTEM
A003   0009      INIT:       DEFS   9        ;INITIALIZE SYSTEM
A00C   0003      REENTR:     DEFS   3        ;REENTER SYSTEM
A00F             ;
A00F             ;LINKAGE TO COMMON ROUTINES
A00F             ;
A00F   0003      CALINT:     DEFS   3        ;CALL INIT ROUTINE
A012   0003      ACCES:      DEFS   3        ;ACCESS COMMANDEFS
A015   0003      SYSOUT:     DEFS   3        ;SYSTEM OUTPUT ROUTINE
A018   0003      SYSIN:      DEFS   3        ;SYSTEM INPUT ROUTINE
A01B   0003      P2HEX:      DEFS   3        ;PRINT 2 HEX BYTES
A01E   0003      P1HEX:      DEFS   3        ;PRINT 1 HEX BYTE
A021   0003      P2HEXS:     DEFS   3        ;PRINT 2 HEX BYTES AND SPACE
A024   0003      P1HEXS:     DEFS   3        ;PRINT 1 HEX BYTE AND SPACE
A027   0003      PRTNUM:     DEFS   3        ;PRINT CHARACTERS (# IN D)
A02A   0003      READLN:     DEFS   3        ;READ 1 LINE OF INPUT
A02D   0003      GNAME:      DEFS   3        ;GET A NAME PARM
A030   0003      SIOTAB:     DEFS   3        ;LOOK UP IN I/O TABLE
A033   0003      GTHEXM:     DEFS   3        ;GET HEX PARM
A036   0003      GTDECM:     DEFS   3        ;GET A DECIMAL PARM
A039   0003      ERROR:      DEFS   3        ;ERROR ROUTINE
A03C   0003      MSGOUT:     DEFS   3        ;OUTPUT MESSAGE
A03F   0003      PRNTTB:     DEFS   3        ;PRINT TABLE
A042   0003      COMPAR:     DEFS   3        ;COMPARE
A045   0003      SEARCH:     DEFS   3        ;SEARCH TABLE
A048   0003      LOOK:       DEFS   3        ;LOOK THRU TABLE
A04B   0003      FILL:       DEFS   3        ;FILL AREA WITH VALUE
A04E   0003      CLEAR:      DEFS   3        ;FILL AREA WITH SPACES
A051   0003      MBLNK:      DEFS   3        ;MOVE UNTIL DELIMETER
A054   0003      SBLNK:      DEFS   3        ;SKIP BLANKS
A057   0003      SCHAR:      DEFS   3        ;SKIP CHARACTERS UNTIL DELIME
A05A   0003      CMBLNK:     DEFS   3        ;CLEAR AND MBLNK
A05D   0003      CNUM:       DEFS   3        ;CHECK NUMERIC
A060   0003      GETHEX:     DEFS   3        ;GET HEX VALUE
```

```
A063  0003          2534 GDECM:   DEFS  3          ;GET DECIMAL VALUE
A066  0003          2535 LEADER:  DEFS  3          ;WRITE LEADER
A069  0003          2536 EINTEL:  DEFS  3          ;END OF FILE INTEL TAPE
A06C  0003          2537 CHKCUR:  DEFS  3          ;CHECK CURRENT FILE
A06F  0003          2538 WINTEL:  DEFS  3          ;WRITE INTEL FORMAT
A072  0003          2539 PRTONE:  DEFS  3          ;PRINT ONE LINE USING TABS
A075  0003          2540 GTSTNG:  DEFS  3          ;GET A STRING
A078  0003          2541 INTTAB:  DEFS  3          ;INITIALIZE GTENT
A07B  0003          2542 GTENT:   DEFS  3          ;GET AN ENTRY FROM TABLE
A07E  0003          2543 FUPACK:  DEFS  3          ;UNPACK 4 BCD DIGITS
A081  0003          2544 AFPACK:  DEFS  3          ;ADD 4 BCD DIGITS
A084  0003          2545 FPACK:   DEFS  3          ;PACK 4 BCD DIGITS
A087  0003          2546 CDILM:   DEFS  3          ;CHECK FOR DELIMETER
A08A  0003          2547 ADDAHL:  DEFS  3          ;ADD A TO HL
A08D                2548 ;
A08D  0003          2549 SPAC18:  DEFS  3          ;OUTPUT 18 SPACES
A090  0003          2550 SPAC16:  DEFS  3          ;OUTPUT 16 SPACES
A093  0003          2551 SPAC12:  DEFS  3          ;OUTPUT 12 SPACES
A096  0003          2552 SPACE6:  DEFS  3          ;OUTPUT 6 SPACES
A099  0003          2553 SPACE4:  DEFS  3          ;OUTPUT 4 SPACES
A09C  0003          2554 SPACE3:  DEFS  3          ;OUTPUT 3 SPACES
A09F  0003          2555 SPACE2:  DEFS  3          ;OUTPUT 3 SPACES
      A0A2          2556 SPACE:   EQU   $          ;OUTPUT A SPACE
```

APPENDIX I

Paper Tape Loading Instructions

CROMEMCO Z-80 ASSEMBLER V.2.0

```
0000                ;
0000                ;TO LOAD YOUR PAPER TAPE COPY OF
0000                ;CROMEMCO ROS, FOLLOW THESE STEPS:
0000                ;
0000                ;1) BE SURE YOU HAVE 8K OF RAM AT LOCATION 0A000H
0000                ;2) BE SURE YOU HAVE RAM AT LOCATION 0
0000                ;3) KEY IN THIS LOADER AT LOCATION 0
0000                ;4) MOUNT THE PAPER TAPE IN THE READER
0000                ;5) SET THE ADDRESS SWITCHES TO 0
0000                ;6) PRESS STOP
0000                ;7) PRESS EXAMINE
0000                ;8) PRESS RUN
0000                ;9) START THE PAPER TAPE READER
0000                ;
  00                ;WHEN THE PAPER TAPE HAS FINISHED READING,
0000                ;CROMEMCO ROS WILL BE STARTED. DEPRSS
0000                ;CARRIAGE RETURN UNTIL THE MESSAGE
0000                ;'CROMEMCO ROS V.2.0' IS TYPED.
0000                ;
0000                ;IF DURING READING THE PAPER TAPE, A CHECKSUM
0000                ;ERROR OCCURS, A 'C' WILL BE TYPED.
0000                ;START AGAIN AT STEP 4. IF THERE IS BAD MEMORY
0000                ;A 'M' WILL BE TYPED. CHECK YOUR MEMORY!!
0000                ;REPLACE ANY BAD MEMORY AND START AGAIN AT STEP 1.
0000                ;
0000                ;IF YOU ARE USING ANOTHER MANUFACTUREERS I/O BOARD
0000                ;WHICH NEEDS INITIALIZING, CHANGE THE INSTRUCTIONS
0000                ;AT THE LABLE INIT.
0000                ;
0000                         ORG    0
     0001           TTY:     EQU    1          ;TELETYPE DATA PORT
     0000           TTS:     EQU    0          ;TELETYPE STATUS PORT
     0040           DTR:     EQU    40H        ;TELETYPE READY BIT
0000                ;
0000                ;INITIALIZE TELETYPE
0000                ;
0000 97             INIT:    SUB    A          ;SET TO DEVICE A ON CROMEMCO TU
0001 D3 52                   OUT    54H,A
0003 3C                      INC    A          ;RESET TU-ART
```

```
0004  D3 02              OUT   2,A
0006  D3 00              OUT   TTS,A      ;INIT BAUD RATE TO 110
0008  31 00 02           LD    SP,0200H   ;INITIALIZE STACK POINTER
000B                ;
000B                ;START READING TAPE
000B                ;
000B  CD 43 00    WAIT:  CALL  GCHAR      ;GET A CHARACTER
000E  E6 7F              AND   7FH
0010  FE 3A              CP    ':'        ;CHECK FOR A COLON
0012  20 F7              JR    NZ,WAIT    ;NOT FOUND, WAIT FOR A COLON
0014  CD 4C 00           CALL  GTBYT      ;GET COUNT OF CHARACTERS
0017  A7                 AND   A          ;CHECK FOR END OF TAPE
0018  CA 00 A0           JP    Z,0A000H   ;FOUND
001B  47                 LD    B,A        ;SAVE COUNT
001C  5F                 LD    E,A        ;INITIALIZE CHECKSUM
001D  CD 4C 00           CALL  GTBYT      ;GET HIGH BYTE OF ADDRESS
0020  67                 LD    H,A
0021  CD 4C 00           CALL  GTBYT      ;GET LOW BYTE OF ADDRESS
0024  6F                 LD    L,A
0025  CD 4C 00           CALL  GTBYT      ;GET RESERVED BYTE
0028  CD 4C 00    LOOP:  CALL  GTBYT      ;GET DATA BYTE
002B  77                 LD    (HL),A     ;STORE BYTE
002C  BE                 CP    (HL)       ;MAKE SURE STORED
002D  20 0E              JR    NZ,MERROR  ;MEMORY ERROR
002F  23                 INC   HL         ;PT TO NEXT MEMORY LOCATION
0030  10 F6              DJNZ  LOOP       ;COUNT DOWN AND LOOP
0032  CD 4C 00           CALL  GTBYT      ;GET CHECKSUM
0035  7B                 LD    A,E
0036  A7                 AND   A
0037  28 D2              JR    Z,WAIT     ;CHECKSUM OK
0039  3E 43              LD    A,'C'      ;CHECKSUM ERROR
003B  18 02              JR    COUT       ;OUTPUT ERROR CODE
003D                ;
003D  3E 4D       MERROR: LD   A,'M'      ;MEMORY ERROR
003F  D3 01       COUT:  OUT   TTY,A      ;OUTPUT ERROR CODE
0041  18 FE              JR    $-2        ;LOOP UNTIL USER STOPS
0043                ;
0043  DB 00       GCHAR: IN    A,TTS      ;GET TTY STATUS
0045  E6 40              AND   DTR
0047  28 FA              JR    Z,GCHAR    ;LOOP UNTIL CHARACTER
0049  DB 01              IN    A,TTY      ;GET CHARACTER
004B  C9                 RET
004C                ;
004C  CD 43 00    GTBYT: CALL  GCHAR      ;GET A CHARACTER
004F  4F                 LD    C,A        ;SAVE CHARACTER
0050  83                 ADD   E          ;ADD TO CHECKSUM
0051  5F                 LD    E,A        ;SAVE CHECKSUM
0052  79                 LD    A,C        ;RESTORE CHARACTER
0053  C9                 RET
                         END
```

APPENDIX J

Glossary

ASCII	American Standard Code for Information Interchange. A method of encoding bits to represent a character.
Carriage Return Character	When using the teletype for output and a byte containing 13 is sensed, a carriage control will occur, i.e. a return to column one.
Checksum	The checksum is the negative of the sum of all eight bit bytes in the record after the record mark evaluated modulus 256. In other words, if all the eight bit bytes are added together, ignoring carries out of an eight bit sum, and then the checksum is added, the result is zero.
Command String	A series of characters set off by the string symbol (') which contains a system command and its parameters. Example: 'EXEC,AØØØ'
Control Characters	All hexadecimal codes from ØØ to 1F are considered available as control characters, e.g. linefeed.
Delimiter	Any character which will terminate a parameter or string. Frequently, a delimiter functions as a separator, e.g. the comma in EXEC,AØØØ separates EXEC from AØØØ.
Driver	In order to use an Input/Output device, some body of code must: 1) check to see if the device is available, 2) connect the computer to the device, 3) prepare the device for a

transfer of data, 4) properly disconnect the device at the termination of the transfer of data. A driver may do all of the above. A simple device such as a teletype (TTY) has a very simple driver. In contrast a disk driver can be quite complex.

Initialization	Basically, initialization clears all the tables and sets SYSIO to its standard setting.
Linefeed	When using the teletype for output, if a byte containing 1Ø is sensed, a paper advance of one line will occur.
Memory boards	A board on which semiconductor memory modules can be mounted. This board can plug into a master board called a mother board.
Mnemonic Name	A name which the user can easily associate with a desired machine language op-code.
Null	On a paper tape, a null is a frame that will not contain data.
Object Code	The machine readable code which was translated from the user's source code.
Preservation of Registers	When a call is made to a subroutine, the routine or the call may change the contents of several registers. The user may need to preserve the contents of the registers by saving them especially in the stack. Later the registers can be restored from the stack or whatever area they were saved in.
PROM	Programmable Read Only Memory. Once information is written into a PROM by a special burn command, the PROM contents cannot be easily changed. A Cromemco PROM can be erased by radiating the PROM with an ultra-violet source.
Pseudo-op	A command, typically to an assembler, which will not produce any executable

code. For example, a TITLE command will cause a page eject and place a Title on the next page of an assembler listing. A command like TITLE is not like a load instruction which produces code.

RAM	Random Access Memory. An area in main storage which can be both written into and read from.
Region	A logical partition, hunk of memory. The user's file can be said to be assigned the region from 1ØØØH to 15ØØH in memory.
ROS	The Cromemco Resident Operating System once loaded needs no other external routines to operate. In contrast a disk based operating system has a resident portion, the nucleus, and the bulk of the system on a disk.
Source Code	The user written code.
Swath	The number of bytes to be processed.
S length	Swath length.
TTY	A teletype.

APPENDIX K

ASSEMBLER ERROR CODES

There are ten classes of programming errors that can be detected by the Cromemco assembler. If a line of code is in error, this will be indicated by an error code letter just to the left of the line number in the assembly listing. The definitions of these ten error codes are given below:

A	Argument error
D	Double definition
L	Lable error
M	Missing lable
O	Op-code error
P	Phase error
R	Range error
S	Syntax error
U	Undefined
V	Value error