MACHINE LANGUAGE DEBUG PACKAGE

USER'S MANUAL

for

INTECOLOR 3621 AND COMPUCOLOR II

FOR V6.78 & V8.79 SOFTWARE

Intelligent Systems Corp.

Intecolor Drive • 225 Technology Park/Atlanta • Norcross, Georgia 30092 • Telephone 404/449-5961 • TWX 810-766-1581

MACHINE LANGUAGE DEBUG PACKAGE

USER'S MANUAL

for

INTECOLOR 3621 AND COMPUCOLOR II

FOR V6.78 & V8.79 SOFTWARE

SEE PAGE 15

999332

7/23/80

Intelligent Systems Corp.
Intecolor Drive
225 Technology Park/Atlanta
Norcross, Georgia 30092
Telephone: 404/449-5961
TWX: 810/766-1581

TABLE OF CONTENTS

MLDPIN.TXT 12(D)
from WINDER 1/86
not saved !

SUMMARY OF COMMANDS FOR MLDP.

DUMP D	Displays specified range in Hex. e.g. DBG>D AF00,40	
DISPLAY DS	Displays spec. range in assembly. e.g. DBG>DS 9800,40	
FILL F	Fills spec. range with a value. e.g. DBG>F 9000:900A:20	
MOVE M	Moves spec. memory to spec addr. e.g. DBG>M E000,2000 to C000	
AT	Assign a breakpoint at spec.address. e.g. DBG>AT 9000	
LIST L	Displays all breakpoints & opcodes. e.g. DBG>LIST	
CLEAR C	Clears a breakpoint at spec address. e.g. DBG>CLEAR 9408	
RESUME R	Resumes execution after a breakpoint. e.g. DBG>R #8600	
INTERPRET I	Interprets a spec. number of instructions. e.g. DBG>I A000,20	
STEP S	Displays one instr., start at current PC. e.g. DBG>S	
@	The '@' transfers control to EDIT address. e.g. DBG>9000 DBG>@9000	9800 A0 // ANA B MEM>
%	Used to make assignments to psuedo reg./mem.	
%%	Displays contents of 8080 regs and status.	
=	Calculates and displays HEX/DEC equiv. or will calculate an equation.	
/	Exits the MEM> mode and or the MLDP program. e.g. MEM>/ (to DBG>), or DBG>/ (to CRT MODE.)	
EXIT	Exits DBG> mode to CCII CRT mode. e.g. DBG>EXIT	

The MLDP manual is required to make full use of the program, size is twenty-three pages.

INTRODUCTION TO MLDP

MLDP is the abbreviated name for **M**achine **L**anguage **D**ebug **P**ackage.

The Compucolor II MLDP program is a set of routines that facilitates the coding and debugging of machine language programs. It contains commands to set breakpoints (up to 8 at one time), manipulate machine registers, obtain a hex dump of memory, alter memory to a numeric or character value, disassemble memory contents to mnemonics, enter opcodes which are immediately assembled and stored in memory, move a memory range, fill a memory range with a constant, execute a machine language program, interpret a program, and single step a program. MLDP is an extremely versatile tool in that execution of a program may be halted at any point, the states of the registers and memory noted and/or altered, and the program itself corrected or displayed in assembler mnemonics. Included in all commands that accept numeric parameters is the ability to calculate the value of any expression involving constants, register or memory locations and algebraic numeric, logical, and comparative operations.

A Compucolor with at least 16K user memory is required for MLDP. It is neccessary to have the Compucolor 8080 Assembler and Compucolor Text Editor or Compucolor Screen Editor diskettes to completely create, debug and assemble an 8080 source file. The Compucolor Screen Editor is the most efficient way to create and edit an 8080 source file. and requires the 117-key keyboard.

On the following pages are the MLDP commands, syntax, and usage. The 'Summary of Commands' page will outline the MLDP commands in general for use as a quick reference. The 'Explanation of Commands' pages will contain the command keywords and abbreviated keywords, along with the syntax of the parameters. After a description of the command syntax and output, there are several examples that may aid the user in understanding the purpose and general implementation of the command.

The MLDP program has a built-in error checking routine which monitors the user command input. If invalid syntax is used, an easy to understand error message will be displayed and a red marker will indicate where the syntax error occurred within the command input.

SUMMARY OF COMMANDS

DUMP **D**	Displays a specified memory range in hexadecimal format. e.g. DBG>D AF00,40
DISPLAY **DS**	Displays a specified memory range in disassembled 8080 mnemonics. e.g. DBG>DS 9800,40
FILL **F**	Fills a specified memory range with a value. e.g. DBG>F 9000:900A:20
MOVE **M**	Moves a specified block of memory to a specified address. e.g. DBG>M E000,2000 to C000
AT	Assign a breakpoint at specified memory address. e.g. DBG>AT 9000
LIST **L**	Displays all assigned breakpoints and corresponding op codes. e.g. DBG>LIST
CLEAR **C**	Clears a breakpoint at a specified address. e.g. DBG> CLEAR 9408
RESUME **R**	Resumes program execution after the occurrence of a breakpoint. e.g. DBG>R #8600
INTERPRET **I**	Interprets a specified number of 8080 mnemonic instructions. DBG>I A000,20
STEP **S**	Displays one instruction at a time, beginning at the current Program Counter address. eg. DBG>S
@	The symbol '@' transfers control to the Memory Edit Mode with the editor set to the specified address. e.g DBG>@9000
%	The symbol '%' is used to make assignments to the psuedo registers and/or memory locations.
%%	The symbols '%%' display the contents of the 8080 registers and status.
=	The symbol '=' calculates and displays either a decimal/hexadecimal equivalent or evaluates an equation.
/	The symbol '/' exits the MEM> mode and/or the MLDP program and returns control to the Compucolor CRT MODE.
EXIT	Exits DBG mode to Compucolor CRT MODE.

DUMP <memory range>
D <memory range>

The DUMP command accepts a memory range and displays it to the screen in hexadecimal with color coded character equivalents. The output is organized as 16 locations per line displayed in hex followed by the corresponding 16 character interpretations. The color of the character indicates the state of the upper two bits:

Red indicates a control character.
Green indicates normal ASCII graphics and upper case.
Yellow indicates the Compucolor special graphic character set.

Note that for the Red and Yellow ranges the characters displayed will be the corresponding upper case ASCII symbols. If the most significant bit of the location is set (bit 7) then the character color will change to Magenta, Cyan and White for the Red, Green and Yellow ranges respectively.

The memory range supplied to the DUMP command (and anywhere else the DUMP or D <memory range> is used) may be entered as a starting address, a comma, and a length or as a starting address, a colon, and an ending address.
All of the parameters may be entered as either constants or expressions. The default length for the DUMP command is 256 (100 hex).

If the Break key is depressed the display will pause until another key is entered. If that key is a linefeed then execution of the command is terminated otherwise the Dump continues.

Example Input	Result
DBG>**DUMP** 0,FFFF<cr>	Displays 0 hex thru FFFF hex in hexadecimal format.
DBG>**D** 0:100<cr>	Displays 0 hex thru 00FF hex in hexadecimal format.
DBG>**D** 8200,1000<cr>	Displays 8200 hex thru 91FF hex in hexadecimal format.
DBG>**D** 9000:9000+500<cr>	Displays 9000 hex thru 94FF hex in hexadecimal format.

DISPLAY <memory range>
DS <memory range>

The Display command accepts a memory range and produces a disassembed listing to the screen composed of the following fields:

AAAA:XXXXXX 'CCC' MNEMONICS

where AAAA is the address of the code, XXXXXX is the hex value of the code, CCC is the color coded character representation (See DUMP command) and MNEMONICS are the corresponding assembly language mnemonics. The Break key responds in the same manner as in the DUMP command and the parameters are the same except the default length is 1 instruction.

Example Input	Result
DBG>**DISPLAY** 0,40<cr>	Disassembles and displays 0 hex thru 3F hex.
DBG>**DS** B000,100<cr>	Disassembles and displays B000 hex thru B0FF hex.

FILL <memory range> WITH <value>
F <memory range> WITH <value>

The FILL command accepts a memory range and a value and proceeds to fill the indicated block of memory with the low order 8 bits of the value. The value may be any valid expression with a legal byte value (-128 to 255). If the comma is used in place of WITH then an explicit length must be entered else the parser will accept the value field as the length and get a syntax error when it looks for the value. MLDP will malfunction if its memory space is FILLed with a value.

Example Input	Result
DBG>**FILL** 8200:DFFF WITH FF<cr>	Fills 8200 hex thru DFFF hex with FF hex.
DBG>**F** 6000:6FFF WITH 55<cr>	Fills 6000 hex thru 6FFF hex with 55 hex.
DBG>**FILL** 8200:A000:0<cr>	Fills 8200 thru 9FFF with 0.

MOVE <memory start>,<memory end> TO <address>
M <memory start>:<length in bytes>,<address>

The MOVE command accepts a memory range and a destination address and moves the block of memory specified to the destination address. The same holds true for the use of comma as described above. MLDP will malfunction if only part of its memory space is MOVEd or if other memory is moved to the MLDP's memory space.

Example Input	Result
DBG>**MOVE** E000,2000 TO C000<cr>	Moves the memory range from E000 hex thru FFFF hex to C000 hex.
DBG>**M** 0:04FF TO B000<cr>	Moves the memory range from 0 hex thru 04FF hex to B000 hex.

AT <address>

The AT command sets a breakpoint at the address specified. The breakpoint will remain active until cleared by a CLEAR command or the MLDP program is restarted. The memory location at the address specified will not be altered in any way until a RESUME command is issued, whereupon RST 1 instructions will be inserted at all active breakpoints after the first instruction is interpreted.

When an RST 1 instruction is executed and the address it occurs at is in the table of breakpoints, it will be acknowledged as a breakpoint. If a RST 1 instruction is executed in a user program that the debugger did not set it will be acknowledged as a checkpoint and the RST 1 instruction will not be altered. In either case, all active breakpoints will have their instructions restored to the original codes.

Example Input	Result
DBG>**AT** 9000<cr>	Sets a breakpoint at 9000 hex.
DBG>**AT** 1800<cr>	Sets a breakpoint at 1800 hex.
DBG>**AT** %PC+50-#11<cr>	If the program counter(PC) was equal to 8200 hex then a breakpoint at 8245 hex would be set. The program counter (PC), plus 50 hex, minus 11 decimal.

LIST
L

The LIST command accepts no parameters and displays a table of all active breakpoints to the screen in the form:

XXXX:DD

where XXXX is the address of the breakpoint and DD is the instruction code at that location. No more than 8 breakpoints may be active concurrently.

Example Input	Result
If breakpoints were previously set at 1200,4000,8200 hex using the AT command then the following would be listed to the screen after the LIST command:	
DBG>LIST<cr> 1200:DD 4000:DD 8200:DD	The breakpoints at 1200 hex, 4000 hex and 8200 hex will be displayed.
DBG>L<cr>	Same result as above.

CLEAR <address>
C <address>

The CLEAR command accepts an address expression and clears the breakpoint entry for that address if a breakpoint is set there.

Example Input	Result
DBG>CLEAR 3000<cr>	Clears the breakpoint at 3000 hex.
DBG>C 8300<cr>	Clears the breakpoint at 8300 hex.

RESUME [<address>]
R [<address>]

The RESUME command accepts an address if given and stores it in the pseudo PC register. If no parameter is given then the PC is not changed. Subsequently, the first instruction is interpreted, all active breakpoints are setup with RST 1 instructions, the machine registers are loaded from the pseudo registers, and execution of the user program is resumed. Upon execution of a RST 1 instruction the debugger will regain control of the system, store the machine registers in the pseudo registers. replace all breakpoint instructions and inform the user of the type of execution check that occurred (either a breakpoint or a checkpoint). The first instruction to be interpreted is the instruction after a RESUME address. A breakpoint may be placed at the current address which if executed would immediately return to the debugger, without ever executing a single instruction from the user program.

Example Input	Result
DBG>**RESUME** %PC+3<cr>	Begins execution at the current program counter (PC) address plus 3 hex.
DBG>**R** #3000<cr>	Begins execution at 3000 decimal (668 hex).

INTERPRET [<address>][,<cycles>]
I [<address>][,<cycles>]

The INTERPRET command accepts an optional starting address which replaces the current PC if given and an expression that represents the number of instructions to be interpreted. If the cycles expression is omitted or given as zero then the interpreter will process instructions indefinitely until either an address in the breakpoint table matches the PC or an illegal instruction is interpreted (an RST 1, a HLT instruction or any instruction not recognized by the interpreter) or the Break key is depressed.

Example Input	Result
DBG>**INTERPRET** ,7<cr>	Interprets seven instructions beginning with the current PC address.
DBG>**I** A000,20<cr>	Interprets 32 instructions beginning with A000 hex thru A01F hex and displays it to the screen.

STEP
S

The STEP command interprets one instruction at the current PC address, then performs a register dump and disassembles the new PC address.

Example Input	Result
DBG>STEP<cr>	Interprets the instruction beginning at the current PC.
DBG>S<cr>	Same as above.

@<address>

The '@' command transfers control to the memory edit routine with the editor set to the address specified. Note that a space does not follow the '@' symbol. After entering the memory edit mode. the code at the address will be disassembled and the user will be prompted with 'MEM>'. If an '=' sign followed by an expression is entered, then the byte value of the expression will be stored in memory and the address incremented. If a single quote is entered followed by a string of characters (including quotes) they will be entered into memory as their ASCII representation and the memory address will be incremented correspondingly. If an '@' sign followed by an address expression is entered then the address will be changed to the value of the expression. If a '-' sign is entered then the address will be decremented by one. If a '+' sign is entered then the address will be incremented by one. If a '/' sign is entered then memory edit mode will be terminated and control will return to the debug mode. If the Return key is depressed with no text preceding it then the address will be incremented by the length of the instruction at the current address. If none of the above conditions hold, then the input line will be interpreted as an assembly language mnemonic and will be assembled and stored at the current address which is then updated by the length of the instruction. A space should separate the mnemonic field from the parameters if there are any. Expressions may be used as constant values but they will be computed at assembly time. not run time.

Example Input	Result
DBG>@8200<cr>	Enters the memory edit mode with the edit address set at 8200 hex.

%<destination>=<expression>

The '%' command is used to make assignments to the pseudo registers and / or memory locations. The destination field may be a register name (any one of A,SW,B,C,D,E,H,L,M,PSW,BC,DE.HL,SP,PC followed by one of Z<C<X<S or P to represent a bit in the status word) or a memory reference (MW or MB followed by an expression in parentheses to denote a memory word or a memory byte respectively). The destination is then followed by an '=' sign and an expression whose value is calculated and stored in the destination.

Example Input	Result
DBG>%PC=?W(%HL+8)<cr>	Sets the pseudo program counter (PC) to the value of the word whose address is the contents of the HL register pair plus eight.

%%

The '%%' command causes the pseudo registers to be dumped to the console along with the top four stack entries.

Example Input

DBG>%%<cr>

```
A  B C  D E  H L  M  SW: (SZXPC) PC   SP (SP+0,SP+2,SP+4,SP+6)
XX XXXX XXXX XXXX XX XX   XXXXX AAAA AAAA AAAA AAAA AAAA AAAA
```

Result

Dumps all registers to the screen in the above format where XX is the hex value in the register and AAAA is the hex address of the program counter (PC), stack pointer, and 4 top stack entries.

=<expression>

The '=' command calculates the value of the expression entered and then displays it in hexadecimal and decimal.

Example Input	Result
DBG>=%A000+4*7<cr>	Displays the value of A000 hex plus 3 times 7, in hex and decimal.
DBG>=#33265<cr>	Calculates the hex equivalent of 33265 decimal.

/

The character symbol '/' terminates the memory edit mode if previously entered by the @ command, otherwise the '/' command will exit the MLDP program and return control to the Compucolor II CRT MODE.

Example Input	Result
MEM>/<cr> DBG>	While in the Memory Edit Mode (MEM>), the '/' symbol will exit back to the DBG> mode.
DBG>/<cr> (screen clears) CRT MODE VX.XX	While in the DBG> mode, the '/' symbol will exit the MLDP program and reset the Compucolor to the CRT MODE.

EXIT

The EXIT command allows the MLDP program to be terminated from the DBG> mode and returns control to the Compucolor CRT MODE.

Example Input	Result
DBG>EXIT<cr> (screen clears) CRT MODE VX.XX	Exits the MLDP program and enters the Compucolor CRT MODE.

GENERAL EXPRESSION SYNTAX

Expressions are entered in standard algebraic form with the exception that precedence is right to left rather than left to right. This means that 5-3-2 is interpreted as 5-(3-2)=4 rather than (5-3)-2=0. Otherwise, all operators have reasonably standard precedence.

The following is a list of operators in order of their precedence.

<table>
<tr><th>OPERATOR</th><th>PRIORITY</th><th>FUNCTION</th></tr>
<tr><td>-</td><td>0</td><td>Negative of Operand</td></tr>
<tr><td>~</td><td>0</td><td>Bitwise Logical NOT of Operand</td></tr>
<tr><td>?L</td><td>0</td><td>Low order Bits of Operand</td></tr>
<tr><td>?H</td><td>0</td><td>High order Bits of Operand</td></tr>
<tr><td>?S</td><td>0</td><td>Sign Extend of Operand Low Order Bits</td></tr>
<tr><td>?B</td><td>0</td><td>Byte at Memory addressed by Operand</td></tr>
<tr><td>?W</td><td>0</td><td>Word at Memory addressed by Operand</td></tr>
<tr><td>?U</td><td>1</td><td>Operand 1 Shifted Left (Up) Operand 2 Bits</td></tr>
<tr><td>?D</td><td>1</td><td>Operand 1 Shifted Right (Down) Operand 2 Bits</td></tr>
<tr><td>*</td><td>2</td><td>Unsigned Product of Operands</td></tr>
<tr><td>/</td><td>2</td><td>Unsigned Quotient of Operands</td></tr>
<tr><td>\</td><td>2</td><td>Remainder of Operand 1 Divided by Operand 2</td></tr>
<tr><td>+</td><td>3</td><td>Operand 1 Plus Operand 2</td></tr>
<tr><td>-</td><td>3</td><td>Operand 1 Minus Operand 2</td></tr>
<tr><td>=</td><td>4</td><td>Operand 1 Equal to Operand 2</td></tr>
<tr><td>></td><td>4</td><td>Operand 1 Greater Than Operand 2</td></tr>
<tr><td><</td><td>4</td><td>Operand 1 Less Than Operand 2</td></tr>
<tr><td>>=</td><td>4</td><td>Operand 1 Greater Than or Equal to Operand 2</td></tr>
<tr><td><=</td><td>4</td><td>Operand 1 Less Than or Equal to Operand 2</td></tr>
<tr><td><></td><td>4</td><td>Operand 1 Not Equal to Operand 2</td></tr>
<tr><td>&</td><td>5</td><td>Operand 1 Bitwise AND Operand 2</td></tr>
<tr><td>!</td><td>5</td><td>Operand 1 Bitwise OR Operand 2</td></tr>
<tr><td>|</td><td>5</td><td>Operand 1 Bitwise XOR Operand 2</td></tr>
</table>

Operands may be entered as hexadecimal values (default radix), as decimal if prefixed with a '#' sign, as a register name prefixed with a "%" sign or as an expression enclosed in parentheses.

EXPLANATION OF MLDP ERROR CODES

One of the following error codes are generated by MLDP if incorrect syntax and/or invalid parameters are used.

E000 ??? SYSTEM ERROR ???

REASON FOR ERROR
Detected error in MLDP program. The MLDP program in memory was altered causing the program to malfunction.

CORRECTIVE ACTION NEEDED
Re-load the MLDP program into the computer and begin again.

E001 INVALID SYNTAX.

REASON FOR ERROR
Invalid or improper use of parameter expression.

CORRECTIVE ACTION NEEDED
Refer to MLDP instructions for proper parameter expression syntax.

E002 INVALID COMMAND.

REASON FOR ERROR
Invalid or improper use of command keywords.

CORRECTIVE ACTION NEEDED
Refer to MLDP instructions for proper command expression syntax.

E003 BREAKPOINT PREVIOUSLY SET.

REASON FOR ERROR
An attempt was made to assign a address for use as a breakpoint twice.

CORRECTIVE ACTION NEEDED
Use the LIST command to verify the status of all currently assigned breakpoints.

E004 BREAKPOINT NOT SET AT ADDRESS.

REASON FOR ERROR
An attempt to RESUME execution at an address not specified as a current breakpoint.

CORRECTIVE ACTION NEEDED
Use the LIST command to obtain the status of all currently assigned breakpoints.

E005 TOO MANY BREAKPOINTS SET.

REASON FOR ERROR
An attempt was made to assign more than 8 breakpoints concurrently.

CORRECTIVE ACTION NEEDED
Since only 8 breakpoints may be assigned concurrently then the only solution is to use the CLEAR command and free one of the currently assigned breakpoints.

EXPLANATION OF MLDP PROMPTS

INSTRUCTION COUNT EXHAUSTED AT AAAA

This prompt will be displayed whenever the number of instructions specified in the INTERPRET command have been executed.

BREAKPOINT AT AAAA

This prompt will be displayed whenever a breakpoint is encountered. A display of the registers will take place at the breakpoint address showing their current content.

INTERUPTED BY USER AT AAAA

This prompt will be displayed whenever the user aborts an executing instruction such as INTERPRET, by depressing the ATTN/BREAK key.

ILLEGAL OPCODE EXECUTED AT AAAA

Displayed whenever an attempt is made to execute an illegal 8080 Op Code.

CHECKPOINT AT AAAA

Displayed whenever a RST 1 instruction is executed in a user program that the MLDP did not set. It will be acknowledged as a checkpoint and the RST 1 instruction will not be altered.

EXECUTION HALTED AT AAAA

Displayed whenever a HLT instruction is executed in a user program.

RELOCATING PRG-TYPE FILES

Nine (9) programs are included on the MLDP disk:

MENU.BAS;01 BAS (BASIC file) Relocate program that creates PRG files.

MLDP Debug Files

MLDP.LOW;01 LDA (Assembler Object File) ORGed at 0000H.
MLDP.HGH;01 LDA (Assembler Object File) ORGed at 0100H.
MLDP.PRG;01 PRG (Machine Code File) that runs at A000H.
MLDP.PRG;02 PRG (Machine Code File) that runs at E000H.

PRINT Printer Handler Files

PRINT.LOW;01LDA (Assembler Object File) ORGed at 0000H.
PRINT.HGH;01LDA (Assembler Object File) ORGed at 0100H.
PRINT.PRG;01PRG (Machine Code File) that runs at A000H.
PRINT.PRG;02PRG (Machine Code File) that runs at E000H.

The PRINT.PRG files are Compucolor RS232C printer drivers. By RUNning PRINT in the FCS mode, text files can be listed out to a RS232C equipped printer using the Compucolor RS232C port.

Two MLDP LDA-type files (.LOW & .HGH) and MENU, allow the user to relocate and create MLDP.PRG files at locations specified by the user. MLDP.LOW and MLDP.HGH have been previously assembled at 0000 and 0100 hex respectively.

Two PRINT LDA-type files (.LOW & .HGH) and MENU, allow the user to relocate and create PRINT.PRG files at locations specified by the user. PRINT.LOW and PRINT.HGH have been previously assembled at 0000 and 0100 hex respectively.

LDA-type files may be used with the MENU to relocate and create PRG-type files. Initially, two source (.SRC) files must be created using the Compucolor Text Editor or Screen Editor. The source file cannot end with a DS instruction. One source (.SRC) file ORGed at 0000 hex and the other source (.SRC) file ORGed at 0100 hex. The Compucolor Assembler is used to assemble the two source (.SRC) files which produce two different LDA-type files. The LDA file ORGed at 0000 hex is renamed using FCS.

Example: FCS COMMAND   FCS>REN MLDP.LDA TO MLDP.LOW

RESULT   MLDP.LOW

The LDA file ORGed at 0100 hex is renamed using FCS to HGH.

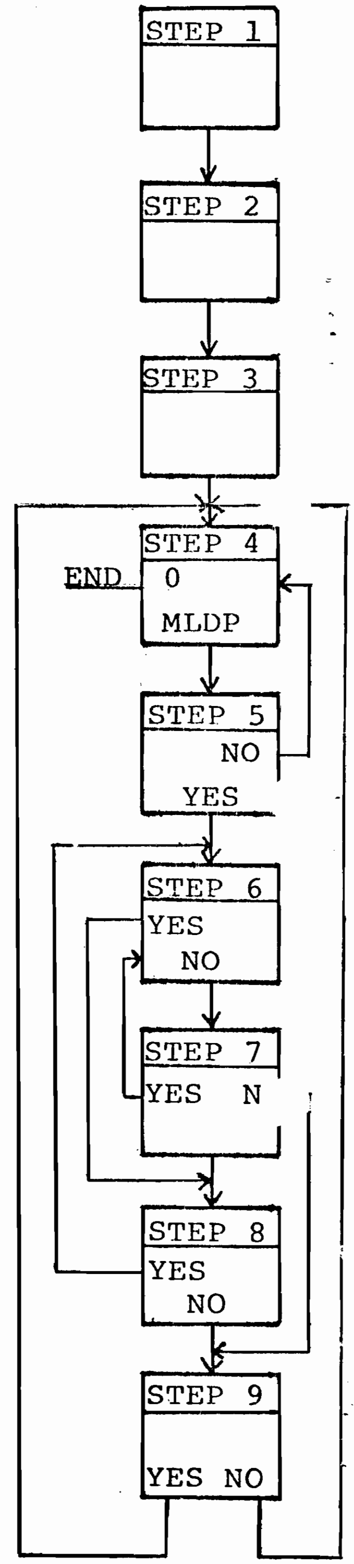
Example: FCS COMMAND   FCS>REN MLDP.LDA TO MLDP.HGH

RESULT   MLDP.HGH

The MENU can now be used with the two (.LOW & .HGH) type files to relocate and create a PRG-type file. The LOW & HGH files created, will allow a PRG file to be relocated in user memory.

⊗

DECISION FLOWCHART



STEP	INSTRUCTIONS & COMMENTS
1	Insert MLDP diskette into default disk drive
2	Initialize BASIC (Press ESC, then W, then RETURN)
3	Press AUTO and the MENU will LOAD and RUN

The following will be displayed:

4 MEMORY ENDS AT AAAA
FILE NAME (ENTER '0' TO END):

(AAAA is the end-of-memory)

Enter 0 to end MENU program
-OR-
Enter MLDP for file name

If **MLDP** is entered in STEP 4, the disk drive will access for a period of time and then the following will be displayed:

5 FILE ALREADY EXISTS: DO YOU WISH TO CONTINUE (Y/N)?

Entering YES will continue to STEP 6
-OR-
Entering NO will return to STEP 4

* If YES was entered in STEP 5, then the following will be displayed:

6 FILE SPECS ARE -
LOAD ADDRESS: AAAA
FILE SIZE: 1EE6
START ADDRESS: AAAA
DO YOU WISH TO OVERIDE LOAD ADDRESS?

(AAAA is the memory address)

The loading address, file size, and starting address are calculated by the computer.

Enter NO if the loading address is desirable.
-OR-
Enter **YES** if loading address is not desirable.

Enter YES or NO.

* If NO was input then go to STEP 8.
* If YES was entered in STEP 6, then the following will be displayed:

7 INPUT YOUR NEW LOAD ADDRESS:

Enter a hex address of your choosing between 8200 hex and E100 hex. Any address outside this boundary will result in an error message. The program will round off to the lowest 256 page boundary (example E150 will be E100).

Enter a Hex Address (For this example 9000 will be used)

8 FILE SPECS ARE --
LOAD ADDRESS: 9000
FILE SIZE: XXXX
START ADDRESS: 9000
DO YOU WISH TO OVERIDE LOAD ADDRESS?

If **NO** is entered then go to STEP 9
If YES is entered then go to STEP 6

Entering **YES** will overide the designated load address, then go to STEP 6

-OR-

Entering **NO** will go to STEP 9

* If NO was entered in STEP 8, the disk drive will access and display the following:

THIS SHOULD TAKE ABOUT 10 TO 11 MINUTES...

After a 10 to 11 minute wait, the following is displayed:

MLDP.PRG HAS BEEN CREATED.

9 DO YOU WISH TO RESET THE CURRENT END-OF-MEMORY (Y/N)?

Entering **YES** will change the BASIC end-of-memory. The initial end of memory will be changed to one byte less than the loading address of the PRG file just created.
Entering **NO** will not change end-of-memory and return to STEP 4

Enter **YES** or **NO**.

* If **YES** was entered in STEP 9, then go to STEP 4
* If **NO** was entered in STEP 9, then go to STEP 4

NOTE: The steps are the same when creating any (.PRG) type file.

VERSION 6.78 SYSTEM SOFTWARE MEMORY MAP

All addresses are hexadecimal.

0000	-	**003F**	Restart Vectors and Initial Values
0040	-	**211B**	BASIC MASK ROM
211C	-	**3FFF**	FCS and CRT MASK ROM
4000	-	**5FFF**	Future Space / User Space
6000	-	**6FFF**	High Speed Screen Refresh RAM
7000	-	**7FFF**	Low Speed Screen Refresh RAM
8000	-	**81FF**	System Scratch Pad Memory
8200	-	**FFFF**	Dynamic User RAM

VERSION 8.79 SYSTEM SOFTWARE MEMORY MAP

All addresses are hexadecimal.

0000	-	**003F**	Restart Vectors and Initial Values
0040	-	**1F25**	FCS and CRT MASK ROM
1F26	-	**3FFF**	BASIC MASK ROM
4000	-	**5FFF**	Future Space / User Space
6000	-	**6FFF**	High Speed Screen Refresh RAM
7000	-	**7FFF**	Low Speed Screen Refresh RAM
8000	-	**81FF**	System Scratch Pad Memory
8200	-	**FFFF**	Dynamic User RAM

ESCAPE SEQUENCE TABLE

Keybd Hex Address

ESC I => 9000
ESC P => 4000
ESC S => A000
ESC T => 8200
ESC Z => 4800
ESC \ => 5000
ESC] => 5800

8080 OP CODE TABLE

The table contains all the 8 bit numbers from 0 to 255 in decimal and hexadecimal so the table can also be used as a base conversion chart. The following format is used for the mnemonics:

One byte instructions are shown in capitol letters only. Two and three byte instructions have operands in angle brackets signifying the additional bytes following the mnemonics. LO is the low order byte and HI is the high order byte of an address or immediate data. DB = one byte of immediate data and DV = device code. Unimplemented codes are signified by '--'.

DEC	HEX	MNEMONIC	DEC	HEX	MNEMONIC	DEC	HEX	MNEMONIC
000	00	NOP	040	28	- -	080	50	MOV D,B
001	01	LXI B<LOHI>	041	29	DAD H	081	51	MOV D,C
002	02	STAX B	042	2A	LHLH <LOHI>	082	52	MOV D,D
003	03	INX B	043	2B	DCX H	083	53	MOV D,E
004	04	INR B	044	2C	INR L	084	54	MOV D,H
005	05	DCR B	045	2D	DCR L	085	55	MOV D,L
006	06	MVI B<DB>	046	2E	MVI L<DB>	086	56	MOV D,M
007	07	RLC	047	2F	CMA	087	57	MOV D,A
008	08	- -	048	30	- -	088	58	MOV E,B
009	09	DAD B	049	31	LXI SP<LOHI>	089	59	MOV E,C
010	0A	LDAX B	050	32	STA <LOHI>	090	5A	MOV E,D
011	0B	DCX B	051	33	INX SP	091	5B	MOV E,E
012	0C	INR C	052	34	INR M	092	5C	MOV E,H
013	0D	DCR C	053	35	DCR M	093	5D	MOV E,L
014	0E	MVI C<DB>	054	36	MVI M<DB>	094	5E	MOV E,M
015	0F	RRC	055	37	STC	095	5F	MOV E,A
016	10	- -	056	38	- -	096	60	MOV H,B
017	11	LXI D<LOHI>	057	31	DAD SP	097	61	MOV H,C
018	12	STAX D	058	32	LDA <LOHI>	098	62	MOV H,D
019	13	INX D	059	33	DCX SP	099	63	MOV H,E
020	14	INR D	060	34	INR A	100	64	MOV H,H
021	15	DCR D	061	35	DCR A	101	65	MOV H,L
022	16	MVI D<DB>	062	36	MVI A<DB>	102	66	MOV H,M
023	17	RAL	063	37	CMC	103	67	MOV H,A
024	18	- -	064	38	MOV B,B	104	68	MOV L,B
025	19	DAD D	065	39	MOV B,C	105	69	MOV L,C
026	1A	LDAX D	066	3A	MOV B,D	106	6A	MOV L,D
027	1B	DCX D	067	3B	MOV B,E	107	6B	MOV L,E
028	1C	INR E	068	3C	MOV B,H	108	6C	MOV L,H
029	1D	DCR E	069	3D	MOV B,L	109	6D	MOV L,L
030	1E	MVI E<DB>	070	3E	MOV B,M	110	6E	MOV L,M
031	1F	RAR	071	3F	MOV B,A	111	6F	MOV L,A
032	20	- -	072	40	MOV C,B	112	70	MOV M,B
033	21	LXI H<LOHI>	073	41	MOV C,C	113	71	MOV M,C
034	22	SHLD <LOHI>	074	42	MOV C,D	114	72	MOV M,D
035	23	INX H	075	43	MOV C,E	115	73	MOV M,E
036	24	INR H	076	44	MOV C,H	116	74	MOV M,H
037	25	DCR H	077	45	MOV C,L	117	75	MOV M,L
038	26	MVI H<DB>	078	46	MOV C,M	118	76	HLT
039	27	DAA	079	4F	MOV C,A	119	77	MOV M,A

8080 OP CODE TABLE
(continued)

DEC	HEX	MNEMONIC
120	78	MOV A,B
121	79	MOV A,C
122	7A	MOV A,D
123	7B	MOV A,E
124	7C	MOV A,H
125	7D	MOV A,L
126	7E	MOV A,M
127	7F	MOV A,A
128	80	ADD B
129	81	ADD C
130	82	ADD D
131	83	ADD E
132	84	ADD H
133	85	ADD L
134	86	ADD M
135	87	ADD A
136	88	ADC B
137	89	ADC C
138	8A	ADC D
139	8B	ADC E
140	8C	ADC H
141	8D	ADC L
142	8E	ADC M
143	8F	ADC A
144	90	SUB B
145	91	SUB C
146	92	SUB D
147	93	SUB E
148	94	SUB H
149	95	SUB L
150	96	SUB M
151	97	SUB A
152	98	SBB B
153	99	SBB C
154	9A	SBB D
155	9B	SBB E
156	9C	SBB H
157	9D	SBB L
158	9E	SBB M
159	9F	SBB A
160	A0	ANA B
161	A1	ANA C
162	A2	ANA D
163	A3	ANA E
164	A4	ANA H
165	A5	ANA L
166	A6	ANA M
167	A7	ANA A
168	A8	XRA B
169	A9	XRA C
170	AA	XRA D
171	AB	XRA E
172	AC	XRA H
173	AD	XRA L
174	AE	XRA M
175	AF	XRA A
176	B0	ORA B
177	B1	ORA C
178	B2	ORA D
179	B3	ORA E
180	B4	ORA H
181	B5	ORA L
182	B6	ORA M
183	B7	ORA A
184	B8	CMP B
185	B9	CMP C
186	BA	CMP D
187	BB	CMP E
188	BC	CMP H
189	BD	CMP L
190	BE	CMP M
191	BF	CMP A
192	C0	RNZ
193	C1	POP B
194	C2	JNZ <LOHI>
195	C3	JMP <LOHI>
196	C4	CNZ <LOHI>
197	C5	PUSH B
198	C6	ADI <DB>
199	C7	RST 0
200	C8	RZ
201	C9	RET
202	CA	JZ <LOHI>
203	CB	- -
204	CC	CZ <LOHI>
205	CD	CALL <LOHI>
206	CE	ACI <DB>
207	CF	RST 1
208	D0	RNC
209	D1	POP D
210	D2	JNC <LOHI>
211	D3	OUT <DV>
212	D4	CNC <LOHI>
213	D5	PUSH D
214	D6	SUI <DB>
215	D7	RST 2
216	D8	RC
217	D9	- -
218	DA	JC <LOHI>
219	DB	IN <DV>
220	DC	CC <LOHI>
221	DD	- -
222	DE	SBI <DB>
223	DF	RST 3
224	E0	RPO
225	E1	POP H
226	E2	JPO <LOHI>
227	E3	XTHL
228	E4	CPO <LOHI>
229	E5	PUSH H
230	E6	ANI <DB>
231	E7	RST 4
232	E8	RPE
233	E9	PCHL
234	EA	JPE <LOHI>
235	EB	XCHG
236	EC	CPE <LOHI>
237	ED	- -
238	EE	XRI <DB>
239	EF	RST 5
240	F0	RP
241	F1	POP PSW
242	F2	JP <LOHI>
243	F3	DI
244	F4	CP <LOHI>
245	F5	PUSH PSW
246	F6	ORI <DB>
247	F7	RST 6
248	F8	RM
249	F9	SPHL
250	FA	JM <LOHI>
251	FB	EI
252	FC	CM <LOHI>
253	FD	- -
254	FE	CPI <DB>
255	FF	RST 7

BINARY, OCTAL, DECIMAL, HEXADECIMAL, ASCII REFERENCE CHART

This chart contains the range from 0 thru 127 decimal.
Hexadecimal value denoted by **bold** print.

BINARY 10_2	OCT 10_8	DEC 10_{10}	HEX 10_{16}	ASCII CHARACTER
00000000	000	000	**00**	NULL
00000001	001	001	**01**	AUTO
00000010	002	002	**02**	PLOT
00000011	003	003	**03**	CURSOR X,Y
00000100	004	004	**04**	– –
00000101	005	005	**05**	– –
00000110	006	006	**06**	CCI
00000111	007	007	**07**	– –
00001000	010	008	**08**	HOME
00001001	011	009	**09**	TAB
00001010	012	010	**0A**	LINE FEED
00001011	013	011	**0B**	ERASE LINE
00001100	014	012	**0C**	ERASE PAGE
00001101	015	013	**0D**	RETURN
00001110	016	014	**0E**	A7 ON
00001111	017	015	**0F**	BLINK/A7 OFF
00010000	020	016	**10**	BLACK KEY
00010001	021	017	**11**	RED KEY
00010010	022	018	**12**	GREEN KEY
00010011	023	019	**13**	YELLOW KEY
00010100	024	020	**14**	BLUE KEY
00010101	025	021	**15**	MAGENTA KEY
00010110	026	022	**16**	CYAN KEY
00010111	027	023	**17**	WHITE KEY
00011000	030	024	**18**	XMIT
00011001	031	025	**19**	CURSOR RIGHT
00011010	032	026	**1A**	CURSOR LEFT
00011011	033	027	**1B**	ESCAPE
00011100	034	028	**1C**	CURSOR UP
00011101	035	029	**1D**	FG ON/FLAG OFF
00011110	036	030	**1E**	BG ON/FLAG ON
00011111	037	031	**1F**	BLINK ON
00100000	040	032	**20**	SPACE
00100001	041	033	**21**	!
00100010	042	034	**22**	"
00100011	043	035	**23**	#
00100100	044	036	**24**	$
00100101	045	037	**25**	%
00100110	046	038	**26**	&
00100111	047	039	**27**	'
00101000	050	040	**28**	(
00101001	051	041	**29**	)
00101010	052	042	**2A**	*
00101011	053	043	**2B**	+
00101100	054	044	**2C**	,
00101101	055	045	**2D**	–
00101110	056	046	**2E**	.
00101111	057	047	**2F**	/
00110000	060	048	**30**	0
00110001	061	049	**31**	1

BINARY, OCTAL, DECIMAL, HEXADECIMAL, ASCII REFERENCE CHART
(continued)

Hexadecimal value denoted by **bold print.**

BINARY 10_2	OCT 10_8	DEC 10_{10}	HEX 10_{16}	ASCII CHAR
00110010	062	050	**32**	2
00110011	063	051	**33**	3
00110100	064	052	**34**	4
00110101	065	053	**35**	5
00110110	066	054	**36**	6
00110111	067	055	**37**	7
00111000	070	056	**38**	8
00111001	071	057	**39**	9
00111010	072	058	**3A**	:
00111011	073	059	**3B**	;
00111100	074	060	**3C**	<
00111101	075	061	**3D**	=
00111110	076	062	**3E**	>
00111111	077	063	**3F**	?
01000000	080	064	**40**	@
01000001	081	065	**41**	A
01000010	082	066	**42**	B
01000011	083	067	**43**	C
01000100	084	068	**44**	D
01000101	085	069	**45**	E
01000110	086	070	**46**	F
01000111	087	071	**47**	G
01001000	090	072	**48**	H
01001001	091	073	**49**	I
01001010	092	074	**4A**	J
01001011	093	075	**4B**	K
01001100	094	076	**4C**	L
01001101	095	077	**4D**	M
01001110	096	078	**4E**	N
01001111	097	079	**4F**	O
01010000	100	080	**50**	P
01010001	101	081	**51**	Q
01010010	102	082	**52**	R
01010011	103	083	**53**	S
01010100	104	084	**54**	T
01010101	105	085	**55**	U
01010110	106	086	**56**	V
01010111	107	087	**57**	W
01011000	110	088	**58**	X
01011001	111	089	**59**	Y
01011010	112	090	**5A**	Z
01011011	113	091	**5B**	[
01011100	114	092	**5C**	/
01011101	115	093	**5D**	[
01011110	116	094	**5E**	^
01011111	117	095	**5F**	_
01100000	120	096	**60**	\
01100001	121	097	**61**	a
01100010	122	098	**62**	b
01100011	123	099	**63**	c

BINARY, OCTAL, DECIMAL, HEXADECIMAL, ASCII REFERENCE CHART
(continued)

Hexadecimal value is denoted by **bold** print.

BINARY 10_2	OCT 10_8	DEC 10_{10}	HEX 10_{16}	ASCII CHAR
01100100	124	100	**64**	d
01100101	125	101	**65**	e
01100110	126	102	**66**	f
01100111	127	103	**67**	g
01101000	130	104	**68**	h
01101001	131	105	**69**	i
01101010	132	106	**6A**	j
01101011	133	107	**6B**	k
01101100	134	108	**6C**	l
01101101	135	109	**6D**	m
01101110	136	110	**6E**	n
01101111	137	111	**6F**	o
01110000	140	112	**70**	p
01110001	141	113	**71**	q
01110010	142	114	**72**	r
01110011	143	115	**73**	s
01110100	144	116	**74**	t
01110101	145	117	**75**	u
01110110	146	118	**76**	v
01110111	147	119	**77**	w
01111000	150	120	**78**	x
01111001	151	121	**79**	y
01111010	152	122	**7A**	z
01111011	153	123	**7B**	{
01111100	154	124	**7C**	|
01111101	155	125	**7D**	}
01111110	155	126	**7E**	~
01111111	157	127	**7F**	DELETE

SUGGESTED REFERENCE BOOKS

These suggestions for reference manuals are available from the Intel Technical Library.

1) **8080/8085 Assembly Language Programming Manual** (98-301)

2) **MCS-80 User's Manual** (98-153)

3) **8080/8085 Reference Card** (98-438)

    Intel Corporation
    Literature Department
    3065 Bowers Avenue
    Santa Clara, California 95051
    (408) 987-6475

These suggestions for reference and educational books are available from Adam Osborne and Associates, Inc.

1) **An Introduction to Microcomputers**
   Volume 0: The Beginner's Book (6001)

2) **An Introduction to Microcomputers**
   Volume 1: Basic Concepts (2001)

3) **8080 Programming For Logic Design** (4001)

4) **An Introduction to Microcomputers**
   Volume 2: Some Real Products (3001)

4) **8080A/8085 Assembly Language Programming** (31002)

    Adam Osborne and Associates, Inc.
    P.O. Box 2036
    Berkley, California 94702
    (415) 548-2805 [Pacific Coast Time]
    TWX 910-366-7277

Z80 and 8080 Assembly Language Programming (5167-0)
by Kathe Spracklen

    Hayden Book Company, Inc.
    Rochelle Park, New Jersey

The 8080A Bugbook
Microcomputer Interfacing and Programming (21447)
by Peter R. Rony, David G. Larsen, and Johnathan A. Titus

    Howard W. Sams & Co., Inc.
    4300 West 62nd Street
    Indianaplois, Indiana 46268
