INSTANT ASSEMBLER

for TRS-80 Model I Level II

Mumford Micro Systems - Box 400 - Summerland, California 93067

Corrections to the INTASM Instruction Manual for the TRS-80 Model III

Page 1 (Final paragraph) The Model III version of INTASM occupies 8330 (decimal) bytes starting at 444CH and uses most of RAM between 4360H and 444BH for stack and scratch-pad storage space. In a 16K machine you will have more than 6900 bytes remaining.

Page 2 (Loading Instructions) The tape supplied is 500 baud, so select the low speed cassette option. Also, operate INTASM in caps lock mode only.
(Entry Point) The entry point is 24323 (5F03H).

Page 5 (Item (6)) For economical entry of DEFB and DEFW, use SHIFT-1 (!) and SHIFT-2 (") respectively.

Page 9 (Tape Commands) All tape commands of the Model III Instant Assembler use the 500 baud cassette speed to ensure reliable recordings.

Page 13 (End of Item (A11)) To transfer to Instant Assembler after the SYSTEM prompt "*?", type "/24323" and press ENTER.
(Item (A13)) The Model III addresses are as follows:
(1) The number of spaces of indentation is kept in 54B2H.
(2) The number of printed lines per page, plus one, is kept in 5497H.
(3) The number of blank lines between pages is kept in 549CH. This number and the number in 5497H must be changed together.
For recording the altered INTASM, use a FIRST ADDRESS of 441CH, a FINAL ADDRESS of 64D6H, and an ENTRY ADDRESS of 5F03H.
(Item (A14)) Last sentence — The place to store the 0AH character is in location 549EH.

Page 14 Line 19 of the listing should be changed to

```
        LD      A,3CH
```

Page 15 Line 83 of the listing should be changed to

```
SCAN    CALL    2BH
```

Page 15 Line 91 of the listing should be changed to

```
        CP      1
```

Line 92 of the listing should be changed to

```
        ;RETURN TO BASIC ON BREAK
```

Page 16 The third sentence of text following the program listing should read:
(Did you use "SHIFT-1" for the DEFB pseudo-ops in lines 34 and 36?)

Page 17 (Explanation of the hex-to-decimal converter's ROM calls) The reference to "The subroutine at 3E3H" should instead be to "The subroutine at 2BH".

Page 22 (RG and MM commands) The Model III does not have a BACK ARROW in its character set. Instead, a LESS THAN symbol is used to prompt a register change (with the RG command), or a memory change (with the MM command).

23 (MM command) The Model III has no UP ARROW or DOWN ARROW in its character set. The actual characters displayed when you use these keys to back up or advance the memory display will be (respectively) an upward pointing caret and a broken vertical line that may be considered to be pointing down.

Page 23 (FN command) The FN command in the Model III version of INTASM will not accept a FINAL ADDRESS of FFFF (or 65535). If you wish to search through FFFF for a single byte number you will have to inspect the final location (FFFF) with the MM command.

Page 30 (Section 7.2) The entry point of the Top-Down-Loader for the Model III is 17597 (44BDH).

Pages 33-34 (TP and VF commands) The commands record and verify at 500 baud only.

Page 35 (Next-to-last Paragraph) After executing the hex-to-decimal conversion program, return to Basic with the BREAK key.

INSTANT ASSEMBLER

a Program by J. Blattner

TABLE OF CONTENTS

INTRODUCTION

(TRS-80 Is a Trademark of the Tandy Corporation.)

Instant Assembler is a powerful, tape-based assembly system for the TRS-80 Model I (Level II). It features:

(1) Immediate detection of most potential assembly errors as the lines of symbolic assembly language code are entered.

(2) In-memory assembly and debugging. Entered code may be assembled-in-memory, debugged, and corrected without changing programs and without recording on -- or reading from -- tape.

(3) A built-in, single-stepping debugger with full register displays, including flags displayed as separate registers. The debugger accepts addresses in either decimal or hex, allows either forward or backward movement in examining memory, and provides for disassembly of Z-80 hex code, display of blocks of ASCII characters, hex-to-decimal and decimal-to-hex conversions, and location of all references to any address. Other special features of the debugger include the ability to fast-step to any designated terminal address, and to step with the target program in control of the video display.

(4) A compactly encoded source format that provides a 3 to 1 storage advantage (both in memory and on tape) over the standard source code format.

(5) Production of relocatable code modules.

(6) The ability to link-load independently written relocatable code modules.

(7) Numerous operational features, including:
- Built-in keyboard debounce.
- Error prevention through field editing of keyboard input.
- Single-stroke entry of DEFB and DEFW pseudo-ops for rapid table building.
- Continuous editing of successive lines, including ability to move backward through the source buffer.
- Pinpoint control of video listings, including the ability to move backward in listings.
- Alphabetic listing of symbol table.
- Separate commands for listing error lines and for listing the symbol table.
- Indentation, pagination, and clear, open formatting of printed listings.
- A block move command for rearranging source code.
- Ability to address a line by its label -- or by the label of one of its neighbors -- as well as by its line number. This feature is useful in listing, editing, and block movement.
- A command for finding all references to any label.
- Verification of recorded source tapes.
- Recording of EDTASM-compatible source tapes.

The Instant Assembler tape has three separate programs -- the Instant Assembler proper, and two versions of a Linking Loader that implements feature (6) above. The Instant Assembler proper comprises two major subdivisions -- the assembly subsystem and the debugging subsystem. The assembly subsystem again has the title "Instant Assembler", while the debugging subsystem is named "MicroMind". Together these programs form a unit that occupies 8378 (decimal) bytes of memory, starting at location 434CH (hexadecimal). (The system also utilizes most of RAM between 4260H and 434BH for stack and scratch-pad storage space.) In a 16K RAM you will have remaining more than 7100 bytes -- enough to write an assembly language program of around 2000 bytes, or to write, assemble-in-memory, and debug, an assembly language program of about 1500 bytes. The Model III version of INTASM occupies 8330 (decimal) bytes starting at 444CH and uses most of RAM between 4360H and 444BH for stack and scratch-pad storage space. In a 16K machine you will have more than 6900 bytes remaining.

PART I. THE ASSEMBLER AND DEBUGGER

LOADING INSTRUCTIONS

To load the Instant Assembler program, turn on your computer and answer the "MEMORY (or MEM) SIZE?" query by pressing ENTER. With the computer in the READY state, position the tape for reading Instant Assembler (first on the tape), and depress the PLAY key of the recorder. Now type "SYSTEM" and press ENTER. In response to the "*?" prompt, type "INTASM" and press ENTER. After the tape has been read, another "*?" prompt will appear. Type "/" and press ENTER. Instant Assembler will now be in control of your computer. The "?" prompt at the top left of the screen is a request for entering a command. (Model III select low speed for cassette option, 500 baud). Operate in the caps lock mode only.

ENTRY POINT

The entry point of Instant Assembler is 24112 (5E30H). You may need this address if you wander away from Instant Assembler and want to return. Model III entry point is 24323 (5F03H).

SUMMARY OF COMMANDS

Learning to use Instant Assembler requires understanding its command structure. There are 44 two-letter commands -- 25 in the assembly subsystem and 19 in the debugging subsystem. These commands will be fully explained in the following two sections. Here is a list of the commands with summaries of their functions.

COMPOSING COMMANDS
CP is for ComPosing an assembly language program.
CC is for Continuing Composition of a program.

EDITING COMMANDS
ED is for EDiting lines of source code.
IS is for InSerting additional lines of source code.
DL is for Deleting a Line of source code.
DM is for Deleting Multiple (consecutive) lines of code.
MB is for Moving a Block of source code to a new position.

LISTING COMMANDS
LC is for Listing Completely to the screen.
PC is for Printing a Complete listing on the line printer.
LL is for Listing to the Last line from any starting line.
PL is for Printing to the Last line from any starting line.
PR is for Printing a specified Range of lines.
LE is for Listing all Error lines.
PE is for Printing all Error lines.
LS is for Listing the Symbol table.
PS is for Printing the Symbol table.

TAPE COMMANDS
WS is for Writing the Source code on tape.
VS is for Verifying the recording of the Source code.
WO is for Writing Object code on tape.
WE is for Writing an Edtasm-compatible source tape.
RS is for Reading Source code from tape.

MISCELLANEOUS ASSEMBLER COMMANDS
AM is for Assembling the source program into Memory.
RO is for Resetting the Origin of the source program.
FR is for Finding all References (in the source code) to any label.
MD is for transferring to microMinD (the debugger).

DEBUGGING COMMANDS

SP is for StePping through a machine language program.
XC is for eXeCuting an entire subroutine while stepping.
BD is for Blanking the Display while stepping -- to permit viewing the video output of the program under examination.
RN is for RuNning (at about 140 instructions per second) to a specified terminal address, which may even be in ROM.
BK is for setting a BreaKpont anywhere in RAM.
RB is for Restoring a Breakpoint previously set.
SB is for restoring a breakpoint and then Stepping from the restored Breakpoint.
JP is for JumPing to any location in memory (including ROM).

DEBUGGING ACCESSORY COMMANDS

CL is for CalLing (and executing) any closed subroutine in ROM or RAM.
RG is for displaying and changing the ReGisters of the program under examination.
MM is for displaying and changing MeMory.
P1 is for displaying 19 (consecutive) bytes of memory as Page 1.
P2 is for displaying another 19 bytes of memory as Page 2.

UTILITY AND TRANSFER COMMANDS

FN is for Finding all occurrences of a Number (or address) in a block of memory.
DS is for DiSassembling and displaying (in standard Zilog mnemonics) any Z-80 instructions in ROM or RAM.
AS is for displaying the contents of memory cells as AScii characters.
HD is for Hex-to-Decimal conversions.
DH is for Decimal-to-Hex conversions.
IA is for transferring to Instant Assembler.

SECTION 1. THE ASSEMBLY SUBSYSTEM

It is assumed that you have had previous experience with an assembler (probably EDTASM -- the TRS-80 Editor/Assembler), and that you have a table of the Z-80 mnemonic instructions, together with a description of their functions -- information such as that provided in the EDTASM manual. Composing lines of assembly language source code with Instant Assembler is quite similar to the same operation with EDTASM; for the most part, the formats and procedural rules are the same.

CHAPTER 1. COMMANDS

Instant Assembler has a command structure like that of EDTASM. The assembly subsystem has 25 two-letter commands, which will be fully described in the five subsections of this chapter. The "?" displayed by Instant Assembler is the request for entering a command. The following general remarks apply to the entry of all commands -- both in the assembly and debugging subsystems.

(1) The rule for the entry of any command is that the entry is completed with the typing of the second letter -- the ENTER key does not have to be pressed to enter the command. If you enter an unrecognizable command, Instant Assembler will ask for the command again with another prompt.
(2) The BREAK key is universally recognized as a demand for escape; Instant Assembler will terminate the operation in progress and then request the next command.

1.1. COMPOSING COMMANDS

CP (for ComPose)
Enter "CP" in response to the "?" prompt to commence the composition of an assembly language program. If the source buffer is empty, you will be given line number 1 to start your program. If the source buffer is not empty, Instant Assembler will respond "CONFIRM:"; enter "CP" again to override this protective feature, or else press BREAK to avoid wiping out the source buffer. Here are the rules for entering source code.

(1) A composition line normally has three fields: Label, Opcode, Operand. Use the RIGHT ARROW key to tab to the next field; you cannot tab farther than the operand field. Use the LEFT ARROW key to backspace and erase the previous character (including backspacing to the previous field if necessary). Each field is edited with respect to the number and type of characters that it will accept:
    The label field accepts only a letter or the ampersand (&) as its first character, only letters or digits for its subsequent characters, with 6 characters maximum.
    The opcode field accepts only letters, with 4 characters maximum.
    The operand field accepts anything except spaces, with 17 characters maximum. (If a single quote is entered at the beginning of the operand field, then this field will accept spaces, too.)
    It is not possible to enter more characters in a field than the field limits just given. The ampersand as the first character of the label field is used to designate an external label -- one whose value can be made available to other modules by the Linking Loader.
(2) By entering a semicolon as the first character of a line, you override the three-field format given in (1) and convert the entire line to a comment line, with 40 characters maximum, including the semicolon. No comments are allowed on the lines with instructions.
(3) Instant Assembler does not recognize the ORG, END, or DEFL pseudo-ops. ORG and END are supplied automatically when a program is listed or recorded. Use the RO command (subsection 1.5) to set or change the origin.
(4) Symbols follow the rules for labels -- 6 characters maximum, first character either a letter or ampersand, subsequent characters either letters or digits. Symbols may be postfixed with decimal offsets in the range of -31 to +31, inclusive. (An offset gives the number of bytes of displacement, just as in EDTASM.) Any other combinations involving symbols are not legal. A symbol may represent an address or a 16-bit

constant, but may not be used for an 8-bit constant; thus, JR THERE and LD HL,NUMBER-10 are valid, but LD A,SPACE is not.

(5) Numerical constants may be entered in either decimal or hexadecimal -- though the first character must be a digit -- and may be prefixed with a minus sign. Hex constants and addresses must also bear the postfix "H". Legal entries include:

```
LD    A,0CFH
LD    B,-5
LD    HL,23586
INC   (IX-0BH)
ADD   A,(IY+100)
LD    DE,-9
LD    BC,0A5A5H
```

Note particularly that the increment (or decrement) to an index register may be in either decimal or hex and must lie in the range of -128 (-80H) to +127 (+7FH), inclusive.

(6) The pseudo-ops DEFB and DEFW may be entered economically by using SHIFT-B and SHIFT-W, respectively, before tabbing to the opcode field. (These abbreviations still work after a label has been entered on the line.) Entering SHIFT-B is equivalent to entering the sequence TAB (or RIGHT ARROW), "DEFB", TAB, and places you immediately in the operand field for entering the value of the byte; similarly for SHIFT-W. These features are provided for convenience in assembling tables. Model III use SHIFT-1 (!) and -2 (")

(7) The operand for a DEFS pseudo-op is restricted to the range 1 to 255 (decimal), inclusive. To reserve more than 255 bytes of storage, use multiple DEFS's. DEFS 0 is illegal.

(8) The operand for an EQU pseudo-op must be an absolute address. As examples, HERE EQU 823BH is legal, but HERE EQU THERE+1 is not. (Note also that an EQU pseudo-op must always have a label.)

(9) Because of the 17 character limit in the operand field, a DEFM pseudo-op cannot define a string of more than 15 characters. To define a longer string, use multiple DEFM's.

(10) All relative jumps (JR, JR NZ, JR Z, JR NC, JR C, and DJNZ) must refer to symbolic target addresses. Any relative jump to an absolute address will be rejected. Examples of legal relative jump instructions are:

```
DJNZ  LOOP
JR    NZ,THERE-3
```

Instant Assembler does not recognize "$" as a reference to the memory location of the present instruction. Hence, the following are illegal:

```
JR    $-12
JR    5024H
```

The above restrictions are designed to force you to use labels to indicate the destinations of relative jump instructions -- a sound programming practice in any case.

(11) When a line of source code is complete, enter it by pressing ENTER. Instant Assembler will immediately assemble it (except for a possible reference to an as-yet-undefined symbol). If there is any error, Instant Assembler will reject the instruction, give you an appropriate error message, and present you with the same line number for entering a corrected version of the instruction. Possible error messages at this stage are:

```
BAD LABEL          (Label is a Z-80 operand.)
DBLY DFND LABEL    (Label has already been used.)
MISSING OPCODE
BAD OPCODE         (Not a Z-80 opcode.)
MISSING OPERAND
BAD OPERAND        (Many possible reasons, including field overflow and improper
                    punctuation.)
OUT OF RNG         (Backward relative jump is too long.)
```

(12) If Instant Assembler accepts your instruction, it will present you with the next line number in sequence for continuation. To end composition, press BREAK.

CC (for Continue Composition)
After composition has been ended with the BREAK key, it may be continued by entering the CC command. You will be given a line number one larger than that of the last line of code in the source buffer to continue your program. (CC may also be used to add to a source program that has been read in from tape.)

1.2. EDITING COMMANDS

General
The editing commands all require a starting line number, and two of them (DM and MB) also require an ending line number. (MB requires yet a third line number -- see the description of MB below.) These line numbers may be entered in either of two ways:

(1) As decimal numbers. This method requires that you know your targeted lines by number. Use of the listing commands can help you to find these line numbers.
(2) As labels (with optional decimal offsets in the range of -31 to +31, inclusive). For example, asking for the line "EXIT" would cause Instant Assembler to find the line of source code with the label "EXIT". Asking for "EXIT+10" would direct Instant Assembler to the line whose line number is 10 larger than that of the line with the label "EXIT". Note that the offset number here is the number of lines of offset from the specified label. This feature of being able to address a line by its label (or label plus offset) makes it easy to find lines in a large program if you have a rough (hand-written) copy of the listing, or a printed listing of an earlier version of the program.

If you enter a line number that is less than 1 or larger than that of the last line in the source buffer, Instant Assembler will respond "BAD" and ask for the line number again. The same is true if you enter a label that does not correspond to any line of source code, or a label plus offset that defines a line number outside the range of actual line numbers.

ED (for EDit)
The ED command is used for continuous editing of successive lines of source code and provides for intermixed changes, deletions, and insertions. After you have entered the ED command, you will be asked for a "FIRST LINE#?" Answer this question with the line number (as a decimal number, or as a label plus offset) of the first line that you wish to edit. The instruction at this line number will then be listed (in assembled form).

(1) If you wish to change the line, press the C key. The line will then be deleted, and its line number will be presented for you to enter a corrected version, which you should now do, completing the job by pressing ENTER. (If you BREAK before pressing ENTER, the deleted line will be gone from the source buffer.)
(2) If you wish to delete the line that is presented for editing, press the D key, and the line will be deleted.
(3) If you wish to insert a new line immediately after the line displayed on the screen, press the I key, and you will be given a line number for composing the new line.

After you have changed, deleted, or inserted, the next sequential line will be displayed, and you may treat it as the previous one, with "C", "D", or "I". (Note that insertion or deletion of a line in ED mode affects the line numbers of all following lines -- see also the descriptions of the IS, DL, and DM commands below.) When you press BREAK, you will leave the ED mode.

ADDITIONAL FEATURE: When a line is displayed for editing, you may move backward or forward one line by pressing the UP ARROW or DOWN ARROW key, respectively. (Of course, you may do this repeatedly to move more than one line.) Neither key alters the line that was just displayed; these keys serve to move you to where you want to make your next edit. (Use of any key except UP ARROW, DOWN ARROW, C, D, or I will effect an exit from ED mode. Also, when you edit the final line in the buffer, Instant Assembler will automatically exit from ED mode.)

IS (for InSert)
When you use the IS command, Instant Assembler will ask for a "LINE#?" Respond with a decimal number, or a label plus offset. The insertion will be immediately before the line whose number you specify.

EXAMPLE: If you insert at LINE# 69, the inserted line will then have the line number 69, while previous line 69 will become line 70, previous line 70 will become line 71, etc.

After you have composed the new line, it is inserted, and Instant Assembler will give you the next line number for continued insertion. You may insert as many instructions as you please. Use the BREAK key to exit from IS mode.

DL (for Delete Line)
Use DL to delete a single line. Instant Assembler will ask for the "LINE#?" The line numbers of all lines following the deleted line will be decreased by 1.

DM (for Delete Multiple)
Use DM to delete a block of lines. Instant Assembler will ask for the "FIRST LINE#?" and the "FINAL LINE#?" of the block in two separate questions. (Deletion of a large block takes some time, since the deletion proceeds one line at a time. And, of course, multiple deletion reduces the line numbers of all lines following the deleted block.)

MB (for Move Block)
Use MB to move a block of source code from one position to another. Instant Assembler will ask for three line numbers (with separate queries). "FIRST LINE#" and "FINAL LINE#" designate the first and last lines of the block to be moved, while "INSRT LINE#" is the line number at which the block is to be inserted. (It will be inserted just ahead of the line whose line number is the INSRT LINE#.) The INSRT LINE# must either be less than the FIRST LINE# or greater than the FINAL LINE# plus one; otherwise, you will get a "BAD" message and a request for reentry of this line number. MB will obviously have a drastic effect on many line numbers.

To move a block to the end of the program, first add a NOP at the end (using the CC command), move the block to just in front of the NOP, then delete the NOP.

1.3. LISTING COMMANDS

General
In the listing commands, a first letter of "L" directs the listing to the screen, while a first letter of "P" directs the listing to the line printer. (If printer output is selected, have the printer turned on and ready.) The LL, PL, and PR commands require one or two line numbers to be input; enter these as explained in the General remarks of 1.2 above.

In a listing to the screen, only 12 lines are presented at a time for your inspection; when you are ready for the next 12 lines, press ENTER (or any key except BREAK, SPACE BAR, or UP ARROW). This 12-lines-at-a-time progress of screen listings may be overridden by depressing the SPACE BAR. Holding the SPACE BAR down will cause continuous scrolling of the listing; this scrolling will stop instantly when you release the SPACE BAR. Rapid depression and release of the SPACE BAR will effect the listing of one or two additional lines of the program. With the SPACE BAR released, ENTER will act in its usual fashion to cause the listing of another 12 lines. After a pause in a video listing, use of the UP ARROW key (in either the LC or LL mode) will cause the listing to move backward about ten lines, so that you can review it. Thus, the ENTER, SPACE BAR, and UP ARROW keys give you pin-point control of listings to the screen.

In a listing to the line printer, Instant Assembler indents each line 10 spaces to provide a left margin for binding. (Since a line of listing does not exceed 64 characters, there will still be an ample right margin.) After each 59 lines of printed listing, Instant Assembler supplies 7 line feeds for pagination in the standard (66 lines per page) printer format. (The 59-counter is reset to zero each time a new listing command is entered.) Depressing the BREAK key and holding it down until the printer completes a line will terminate any listing to the printer.

All source code listings are assembly listings; the format is essentially that of EDTASM, with the following exceptions:

(a) Bytes of hex code are separated by spaces for readability.
(b) Assembled bytes for a DEFM instruction are displayed as for other instructions -- four to a line, with up to four lines of display for the one DEFM.
(c) The memory addresses for all intructions except EQU's are displayed at the extreme left of the listing. (EDTASM does not show the starting memory address for a DEFS instruction.) No memory address is displayed for an EQU pseudo-instruction.

LC (for List Completely)
LC causes a complete listing to be posted to the screen (with a pause after each 12 lines unless the SPACE BAR is held down). A complete listing consists of the ORG line (supplied by Instant Assembler), the assembled source code (with an error message at the right of each line in error), the END line (also supplied by Instant Assembler), the error count, and the symbol table. The symbol table is in alphabetic order and is printed four symbols to a line for compactness. The possible error messages in a listing are just these two:

```
OUT OF RNG      (Relative jump is too long.)
UNDEF SYMB      (Undefined symbol.)
```

CAUTION: If you use the UP ARROW key with the LC command to review the listing, errors that you pass over will be counted again in the forward listing, so that the final error count may be too large.

PC (for Print Completely)
PC is like LC, except that the output is to the line printer.

LL (for List to the Last line)
After the LL command is entered, Instant Assembler will ask for a "FIRST LINE#?" Respond with the number (as a decimal, or as a label plus offset) of the first line to be listed. The listing will commence there and continue (with a pause after each 12 lines) to the last source line, or until the BREAK key is used to terminate the LL mode. No symbol table will be printed.

PL (for Print to the Last line)
PL is like LL, except that the output is to the line printer.

PR (for Print a Range of lines)
After the PR command is entered, Instant Assembler will ask for a "FIRST LINE#?" and a "FINAL LINE#?" These are the numbers of the first and last lines of source code to be listed, and may be entered either as decimal numbers or as labels plus offsets. Output is to the line printer.

LE (for List Error lines)
LE causes all lines with errors to be listed to the screen (with a pause after each 12 lines). The UP ARROW key will not reverse the direction of this listing. Listing of error lines will be slow if the program is large and there are few lines in error.

PE (for Print Error lines)
PE is like LE, except that the output is to the line printer.

LS (for List Symbols)
LS causes the symbol table to be listed to the screen. This listing is in alphabetic order, four symbols to a line. (External symbols are listed first.) The defined value of each symbol is displayed next to the symbol. (Actually, what is listed is a label table. Undefined symbols are not listed.)

PS (for Print Symbols)
PS is like LS, except that the output is to the line printer.

1.4. TAPE COMMANDS (Model III: use 500 baud for reliable recordings.)

WS (for Write Source)
The WS command is used to record a source tape (in Instant Assembler format) of the program in the source buffer. (These source tapes are also the relocatable code modules that the Linking Loader operates on. Instant Assembler source is essentially object code plus a symbol table, with pointers connecting the two.) After the WS command is entered, you will be asked for a "TITLE?" The title is restricted to 6 characters, the first of which must be a letter; subsequent characters must be either letters or digits. Have the tape ready for recording, with the PLAY and RECORD keys of the cassette depressed. As soon as you press ENTER after typing the title, the recording will begin. Although an Instant Assembler source tape is recorded with a nominal origin (which may be set with the RO command), this origin has no effect upon where the program can be loaded with Linking Loader.

VS (for Verify Source)
After recording an Instant Assembler source tape with the WS command, rewind the tape and use the VS command to verify it. (Have the tape ready for reading before entering the VS command.) VS requires no arguments and returns either "GOOD" or "BAD" in reporting on the verification. In case of a "BAD" verify, try adjusting the volume before repeating the VS command; as a last resort, record the tape again and verify it.

WO (for Write Object)
The WO command is used to record an object tape (in SYSTEM format) of the program in the source buffer. After the WO command is entered, you will be asked for an "ORIGIN?", an "ENTRY ADDRESS?", and a "TITLE?" (The entry address is the point at which the program will be entered after a SYSTEM load and a response of "/" to the following "*?" prompt.) The origin and entry address may be entered in either decimal (five digits) or hexadecimal (four or fewer hex digits) -- see the RULE FOR ENTRY OF ADDRESSES in subsection 4.1 of Chapter 4. When the title has been entered (by pressing ENTER), recording will commence.

NOTE: Object code recorded with the WO command is in one contiguous block; there are no skips for DEFS pseudo-ops. In fact, a DEFS instruction causes the specified number of bytes to be recorded as zeroes on the object tape.

WE (for Write Edtasm source tape)
The WE command is for recording a source tape (of the program in the source buffer) that can be read and edited by EDTASM. You will be asked for an "ORIGIN?" and a "TITLE?"; recording commences as soon as the latter is entered. The line numbers for a source tape produced with the WE command start with 00000 (for the ORG line) and proceed in steps of 10.

While Instant Assembler is a complete assembly system, the WE command has been provided so that you may use Instant Assembler in conjunction with EDTASM.

RS (for Read Source)
The RS command causes a source tape (recorded in Instant Assembler format with the WS command) to be read into the source buffer for editing, assembling, or debugging. You will be asked for a "TITLE?" before reading commences. Have the tape ready for reading as you complete the entry of the title. Instant Assembler will report on the read with either a "GOOD" or a "BAD" message. In the case of a BAD read, rewind the tape and try again (perhaps adjusting the volume before the second try).

1.5. MISCELLANEOUS COMMANDS

AM (for Assemble-in-Memory)
The AM command permits you to assemble a source program directly into memory. Once assembled, the program may be debugged with the debugging subsystem (MicroMind). After the AM command has been entered, Instant Assembler will respond "1ST FREE MEM: XXXX", where the XXXX is the hexadecimal address of the first memory location available for the assembly. You will then be asked for an "ORIGIN?", which may be entered in either decimal or hex -- see the RULE FOR ENTRY OF ADDRESSES in subsection 4.1 of Chapter 4. The origin must be at least as high as the number announced in the 1ST FREE MEM report; otherwise, Instant Assembler will respond "BAD" and ask for the origin again. Also, the origin must be low enough to allow the assembly to take place in the remainder of RAM; if it is not, Instant Assembler will respond "OUT OF MEM" and ask for the origin again.

As the assembly progresses, any line in error will be listed to the screen, with the error reported at the right end of the line. The listing (and the assembly) will pause when 12 error lines have been listed; press ENTER to continue the assembly. When assembly is complete, the error count will be reported on the screen. If you now list the source program with the LC command, you will find that it has the origin specified for the assembly; thus, the listing will correspond exactly to the assembled program.

RO (for Reset Origin)
Use the RO command to define (or redefine) the origin for a listing or recording. After the RO command is entered, you will be asked for an "ORIGIN?" Enter this in either decimal (5 digits) or hexadecimal (4 or fewer hex digits) -- see subsection 4.1 of Chapter 4 for the RULE FOR ENTRY OF ADDRESSES.

FR (for Find References)
The FR command enables you to find all instructions in the source program that reference any specified label. Instant Assembler responds to the FR command with the query "FIND?" Answer this by entering the label of any line in the source program. (If you enter a nonexistent label, Instant Assembler will merely repeat the "FIND?" question.) Instant Assembler will then report the line numbers -- eight to a video display line -- of all instructions that refer to this label, including any reference to the label with an offset. Following this report, Instant Assembler will repeat the "FIND?" question. Use the BREAK key to terminate the FR mode.

MD (for transfer to microMinD)
Use the MD command to transfer control to MicroMind -- the debugging subsystem of Instant Assembler.

CHAPTER 2. QUESTIONS AND ANSWERS ON USE OF THE ASSEMBLER

(Question 1) Why do line numbers change when I perform insertions, deletions, or block movements?
(Answer 1) The line number of an instruction -- instead of being fixed as it is in EDTASM -- is determined by the relative position of the instruction in the source buffer, which may change as a result of any of the operations mentioned. This implicit line numbering has at least three advantages over fixed line numbering:

(1) It allows continuous insertion of new lines of code without periodic pauses for line renumbering.
(2) The MB command is easy to implement with implicit line numbering. (Moving a large block of code while maintaining a fixed line numbering would be impossible.)
(3) It saves a great deal of space in the storage of the source code -- both in memory and on tape. (If each line of code had a two-byte line number attached to it, Instant Assembler's buffer space requirement would be about 50% higher than it is.)

Of course, implicit line numbering has its disadvantages, too. One of these becomes apparent when you want to delete two or more nonadjacent lines of code. For example, suppose that you wish to delete lines 69 and 85; if you first delete line 69, you will find that the other line is now number 84. One way to handle this problem is to delete from the top down. (In the example, delete line 85 first.) Another way to handle it is to address each line by its label (or label plus offset). In fact, this ability to address a line by its label (plus offset) frees you from dependence upon absolute line numbers. (As a last resort, you can always find the up-to-date line number of any instruction by using the LL command.)

(Q2) When a listing to the screen pauses, why can't I do an immediate edit on one of the displayed lines?
(A2) Because you are still in a listing mode. If you type in "ED", the "E" will merely effect an exit from the listing mode -- it will not be recognized as the first letter of a new command.

(Q3) Why can't I use the ED command to move a labeled line by first inserting it at its new position and then deleting it from its old position?
(A3) Because Instant Assembler will view this attempt as a "doubly defined label" error and abort it. To succeed, you must perform the deletion first; or, you could use the MB command to move the single line. (The problem that this question addresses does not arise if the line of code has no label.)

(Q4) Why does Instant Assembler convert my decimal constants and addresses to hexadecimal?
(A4) Because Instant Assembler keeps no record of the form in which you entered these numbers. In its listings Instant Assembler adheres to one fixed format for numbers other than offsets, and that is hexadecimal. (Offsets are listed in decimal, just as you entered them.) Also, Instant Assembler does not remember minus signs -- except on offsets, and in index register instructions -- after it has converted the negative numbers to their positive equivalents. For example, if you enter the instruction

```
LD    B,-11
```

Instant Assembler will list it as

```
LD    B,0F5H
```

This can be a bit disconcerting at first, but you will quickly get used to it.

(Q5) Why don't origins and entry addresses entered in hexadecimal require a terminating H, or a leading zero if the first hex digit is a letter?
(A5) Because Instant Assembler is smart enough to distinguish between the (required) 5-digit entry of a decimal address and the 4- (or fewer) digit entry of a hex address. This feature is for the convenience of the user.

(Q6) Then why -- in composing lines of source code -- do I have to terminate hex constants and addresses with an H, and why do I have to prefix a zero to these if they would otherwise

commence with a letter?
(A6) Because that is the standard established by EDTASM, and Instant Assembler dared not change it.

(Q7) What does "NO CODE" signify?
(A7) That the source buffer is empty. You will get this message in response to a number of commands when there is no code in the buffer.

(Q8) When may I use external labels?
(A8) External labels are those that commence with an ampersand. You may use external labels whenever you feel like it; all your labels may be external if you choose. However, you need to use external labels only when the program module that you are composing will be loaded together with other modules that reference it. Then every instruction and storage location in this module that will be referenced by another module must be given an external label. Later, Linking Loader will be able to assemble and link all the modules. (Linking Loader does not check for doubly defined external labels, so you must be careful that you use each external label in only one module. Any nonexternal label, however, may be used in as many modules as you please.)

(Q9) How can I assemble an instruction equivalent to

```
LD    HL,STORE+512      ?
```

(A9) In any reasonable program in which this instruction occurred, STORE+512 would not refer to a line of code, but rather to a memory location outside the program area. You will need to give this location a label by using DEFS pseudo-ops. For example, if STORE is the label of the beginning of a buffer area of 512 bytes, the following lines of code would suffice to define the required new label:

```
STORE   DEFS  250
        DEFS  250
        DEFS  12
STOR2   DEFS  1
```

Then, the instruction of the question becomes simply

```
LD    HL,STOR2.
```

(Q10) How can I assemble an instruction equivalent to

```
LD    BC,END-BEG+1      ?
```

(A10) This is a little harder, and requires the expenditure of a few additional bytes of code. The following coding will always suffice, and may be shortened (by deleting the PUSH and POP) if the HL register pair is free at the time the above instuction is needed.

```
PUSH  HL
LD    HL,END+1
LD    BC,BEG
OR    A
SBC   HL,BC
LD    B,H
LD    C,L
POP   HL
```

(Q11) How easily can I wipe out my source buffer?
(A11) Instant Assembler has been carefully designed to make this difficult. Here are three ways to do it:

(1) Use the RS command to read in a new source tape, thus destroying the previous source program.
(2) Use the CP command to try to commence a new source program. Instant Assembler will respond "CONFIRM:". If you now enter "CP" again, you will destroy the old source code.
(3) Enter MicroMind (with the MD command), and then use its MM command to store some random numbers in your source buffer.

Besides the CONFIRM feature of (2) above, Instant Assembler has a safeguard against loss of

the source buffer through any required reinitialization of the computer. For example, in using MicroMind to debug a program, you might inadvertently jump to 0 (thus triggering the ROM startup procedure), or get trapped in an unending loop. In the latter case, simply press the reset button at the rear of your computer. In either case, after your computer comes to the READY state, enter "SYSTEM" and transfer to Instant Assembler by responding "/24112" to the "*?" prompt. You will then find your source buffer intact unless the program that you were debugging wrote into that source buffer.

Model III To transfer to Instant Assembler after the SYSTEM prompt "*?", type "/24323" and press ENTER.

(Q12) What happens if my program is too big for my RAM?
(A12) You will get an "OUT OF MEM" report, followed by an exit from CP (or CC) mode, when you enter a source line that exhausts your available memory. If this occurs, you will either have to shorten your program by deleting some comments or break the program into two or more segments. (The latter procedure is greatly facilitated by the linkage loading feature.) Running out of memory is unlikely, however; with a 16K RAM you should be able to compose an assembly language program of about 2000 bytes, while with a 32K RAM you should be able to compose a 7000-byte program.

(Q13) How can I change the print parameters for listings?
(A13) By knowing the secret locations where these parameters are kept and using the MM command of MicroMind to alter them. There are three print parameters:

(1) The number of spaces of indentation from the left margin (normally 10) is stored in location 53DCH.
(2) The number of printed lines per page, plus one, is stored in location 53C2H. For example, if you wish to reduce the number of lines per page to 54, store 55 (37H) in 53C2H.
(3) The number of blank lines between pages is stored in location 53C7H. This number and the number in 53C2H must be changed together; their total should be 67 for standard printers and paper.

After you have changed any of these parameters, you can load the bottom-up version of Linking Loader and use its TP command to make a copy of your revised Instant Assembler. (Refer to Chapter 8 and subsection 9.3 of Chapter 9. The FIRST ADDRESS, FINAL ADDRESS, and ENTRY ADDRESS for this recording will be 431CH, 6406H, and 5E30H, respectively.)

(Item (A13)) The Model III addresses are as follows:
(1) The number of spaces of indentation is kept in 54B2H.
(2) The number of printed lines per page, plus one, is kept in 5497H.
(3) The number of blank lines between pages is kept in 549CH. This number and the number in 5497H must be changed together.
For recording the altered INTASM, use a FIRST ADDRESS of 441CH, a FINAL ADDRESS of 64D6H, and an ENTRY ADDRESS of 5F03H.

(Q14) What is wrong if my printer does not correctly execute a top-of-form after each page of a listing?
(A14) Instant Assembler creates top-of-form by outputting several line feeds. The line feed character that it transmits to the printer is 8AH, which works on most printers. However, Radio Shack's Line Printer III will advance only one line when it receives several consecutive 8AH characters. If this is your problem, store a 0AH character in location 53C9H in place of the 8AH that is normally there.

Model III The place to store the 0AH character is in location 549EH.

CHAPTER 3. EXAMPLE OF THE ASSEMBLER IN ACTION

Load the Instant Assembler program, run it, and enter the CP command. Then compose the following source code. (The line numbers are supplied by Instant Assembler, of course.)

```
0001 ;HEX-TO-DECIMAL CONVERTER
0002 ;
0003 &BEGIN  CALL  1C9H
0004         LD    HL,3C14H
0005         LD    (4020H),HL
0006         LD    HL,TITLE
0007         CALL  &VIDOT
0008         CALL  &CARET
0009 INPLP   CALL  &CARET
0010         LD    HL,HEXNM
0011         CALL  &VIDOT
0012         CALL  &KBINP
0013         JR    Z,&BEGIN
0014         CALL  &CONVT
0015         LD    A,(4020H)
0016         AND   0C0H
0017         ADD   A,0BH
0018         LD    (4020H),A
0019         LD    A,5DH           ( LD    A,3CH   for Model III)
0020         CALL  33AH
0021         LD    A,20H
0022         CALL  33AH
0023         EX    DE,HL
0024         CALL  0A9AH
0025         XOR   A
0026         CALL  1034H
0027         OR    (HL)
0028         CALL  0FD9H
0029         LD    HL,4131H
0030         CALL  &VIDOT
0031         JR    INPLP
0032 TITLE   DEFM  'HEX-TO-DECIMAL'
0033         DEFM  ' CONVERTER'
0034         DEFB  0
0035 HEXNM   DEFM  'HEX#? '
0036         DEFB  0
0037 ;
0038 ;PART 2 (NEXT) CONTAINS VIDEO OUTPUT AND
0039 ;          CONVERSION ROUTINES
0040 ;
0041 &VIDOT  LD    A,(HL)
0042         OR    A
0043         RET   Z
0044         CALL  33AH
0045         INC   HL
0046         JR    &VIDOT
0047 &CARET  LD    A,0DH
0048         JP    33AH
0049 &CONVT  LD    DE,0
0050 NXTHX   PUSH  HL
0051         LD    H,D
0052         LD    L,E
0053         ADD   HL,HL
```

```
0054          ADD   HL,HL
0055          ADD   HL,HL
0056          ADD   HL,HL
0057          EX    DE,HL
0058          POP   HL
0059          LD    A,(HL)
0060          SUB   30H
0061          CP    0AH
0062          JR    C,DIGIT
0063          SUB   7
0064 DIGIT    OR    E
0065          LD    E,A
0066          INC   HL
0067          DJNZ  NXTHX
0068          RET
0069 ;
0070 ;PART 3 (NEXT) CONTAINS KEYBOARD INPUT
0071 ;               ROUTINE
0072 ;
0073 &KBINP   LD    BC,404H
0074          LD    HL,BUFFR
0075          PUSH  HL
0076          LD    A,0EH
0077 POST     CALL  33AH
0078          LD    A,B
0079          OR    A
0080          JR    Z,CRET
0081 NXTCH    PUSH  BC
0082          PUSH  HL
0083 SCAN     CALL  3E3H            (SCAN   CALL   2BH    for Model III)
0084          OR    A
0085          JR    Z,SCAN
0086          POP   HL
0087          POP   BC
0088          LD    (HL),A
0089          CP    0DH
0090          JR    Z,CRET
0091          CP    65H             (CP     1    for Model III)
0092 ;RETURN TO BASIC ON "SHIFT-E"  (---- ON BREAK     for Model III)
0093          JP    Z,BASIC
0094          CP    8
0095          JR    Z,BKSPC
0096          CP    30H
0097          JR    C,NXTCH
0098          CP    3AH
0099          JR    C,GOOD
0100          CP    41H
0101          JR    C,NXTCH
0102          CP    47H
0103          JR    NC,NXTCH
0104 GOOD     INC   HL
0105          DEC   B
0106          JR    POST
0107 BKSPC    LD    A,B
0108          CP    C
0109          JR    Z,NXTCH
0110          LD    A,(HL)
0111          DEC   HL
```

```
0112          INC   B
0113          JR    POST
0114 CRET     LD    A,0FH
0115          CALL  33AH
0116          LD    A,C
0117          SUB   B
0118          LD    B,A
0119          POP   HL
0120          RET
0121 BUFFR    DEFS  4
0122 BASIC    EQU   1A19H
```

(Model III: "SHIFT-1")

In entering the above program you may need to refer frequently to the procedures detailed under the CP command in subsection 1.1 of Chapter 1. When you have finished, you will have obtained a working knowledge of most of these procedures. (Did you use "SHIFT-B" for the DEFB pseudo-ops in lines 34 and 36?) After the last line has been entered, press the BREAK key and type "LE". If you have done your work correctly, you should get (after a brief pause) the response "ERR COUNT: 000". (If not, use the ED command to correct all listed errors.) Now type "LS" and take a look at the symbol table. (Note that the values of the symbols are low because an origin of 0 has been assumed for your program. You may use the RO command to change the origin to anything you wish.) Next type "LC" and use the ENTER key to go through the entire source program 12 lines at a time, checking it carefully against the above listing. If you have a printer, turn it on, make it ready, and enter the PC command to obtain a printed listing of the program.

Now use the WS command to make a tape of this program for later use. Give it the title "HDCONV". Rewind the tape, and verify it with the VS command. Then, with the DM command, delete lines "HEXNM+2" through "BASIC". If you have done this correctly, only lines 1-36 (and the ORG and END lines) will remain, and there will now be 7 "UNDEF SYMB" errors in the residual program. Since all of these errors are references to external labels that will ultimately be resolved by Linking Loader, they are acceptable. Tape and verify (with WS and VS) this first segment of the program, giving it the title "HDCNV1".

Next, use the RS command to read the HDCONV tape. With the original program in the source buffer again, delete (with the DM command) lines 1 through "HEXNM+1" and lines "DIGIT+5" through "BASIC", retaining Part 2 of the program. (Part 2 by itself should show no errors when listed.) Tape and verify this segment, using the title "HDCNV2". Finally, load the HDCONV tape once more (with the RS command), delete lines 1 through "DIGIT+4", and tape and verify Part 3 (which also should have no errors), giving it the title "HDCNV3". Save these four source tapes for later practice with MicroMind and Linking Loader.

By this time you have exercised many of the commands of Instant Assembler. To practice using the rest of the commands, read in the HDCONV tape again. Enter the AM command, answer the "ORIGIN?" query with "7000", and press ENTER. Your program will be assembled into memory starting at 7000H, and Instant Assembler should report "ERR COUNT: 000". Transfer to MicroMind by typing "MD". Then enter the JP command, respond to the query "ADDRESS?" with "7000", and press ENTER. Your hex-to-decimal conversion program will now execute. Enter any hex number of up to four digits (pressing ENTER if the number of digits is less than four), and the number will be instantly converted to its decimal equivalent. When you tire of this, press the SHIFT and E keys together, and control will be transferred to the Level II monitor. From there you may reenter Instant Assembler by typing "SYSTEM" (ENTER), then "/24112" (ENTER). You will find the source buffer intact.

(Since you may be interested in the inner workings of the hex-to-decimal converter, a few words are in order to clarify some of its more mysterious instructions. The program uses several ROM subroutines; otherwise, it would be much longer than it is. The ROM subroutine at 1C9H clears the screen. The one at 33AH displays a character at the cursor position and updates the cursor. The subroutines at 0A9AH, 1034H, and 0FD9H act to convert a 16-bit

(Model III subroutine is at 2BH.) binary number to a string of decimal digits. The subroutine at 3E3H scans the keyboard and decodes the input characters. Locations 4020H-4021H contain the address of the video memory cell in which the cursor resides. Any remaining mysteries could be solved by using MicroMind to step through the program.)

At this point you are back in Instant Assembler, and the hex-to-decimal converter is still in the source buffer. Use the WO command to write an object tape of the program. Give it an origin of 7000, an entry address of 7000, and a title of "HDCONV"; later, you may load and execute this tape through the SYSTEM command. Also, make an EDTASM source tape with the WE command; save this for later review when you have EDTASM in your computer.

To see how the block move command operates, type "MB", and then move Part 3 of the program to just in front of Part 2. (FIRST LINE# = DIGIT+5, FINAL LINE# = BASIC, and INSRT LINE# = HEXNM+2.) Next use the FR command to find all lines that reference the "&VIDOT" label. End this session with Instant Assembler by practicing inserting (IS), deleting (DL), and editing (ED).

SECTION 2. THE DEBUGGING SUBSYSTEM

Instant Assembler's debugging subsystem is named MicroMind and is reached via the MD command from the assembly subsystem. In addition to a full range of utilities, MicroMind provides powerful debugging support by its ability to single-step through programs with prominent register displays. In stepping, a subroutine may be executed in full with a single command, and a special command permits fast stepping to any designated terminal address.

CHAPTER 4. COMMANDS

MicroMind has 19 two-letter commands, which will be fully explained in the three subsections of this chapter. As in the assembly subsystem, response to your command follows immediately upon typing the second letter of the command. The command prompt in MicroMind is "*".

4.1. DEBUGGING COMMANDS

SP (for SteP)

This command puts MicroMind into the step mode, which allows you to step through a machine language program one instruction at a time. This mode with its register displays provides a nearly infallible tool for debugging. After you have entered the SP command, you will be asked for a "FIRST ADDRESS?", which is the address at which you will begin to execute the machine language program step-by-step. This address may be entered in either decimal or hexadecimal. The rule for the entry of all addresses (in both subsystems of Instant Assembler) is this:

RULE FOR ENTRY OF ADDRESSES

> If you enter five digits, the address is in decimal. If you enter four or fewer digits (possibly including A, B, C, D, E, or F), the address is in hex; a terminating H is not required (or even allowed) for hex addresses.

To step from 1000 decimal, say, enter "01000" as the FIRST ADDRESS. Since nearly all stepping will be at addresses above 16384 (decimal), the five-digit rule for decimal addresses is only slightly restrictive. An address field (for a prompt) is always five characters long. If you enter five digits, the last digit will trigger the next action; if you enter four or fewer digits, you will have to press ENTER to cause the next action.

If you make an error in entering an address, the error can be corrected (before the fifth digit is typed) by use of the LEFT ARROW key for erasure. An uncorrected error will likely result in an address that MicroMind cannot interpret, in which case the "FIRST ADDRESS?" query will be repeated.

Once you are in step mode, the registers will be displayed at the top right of the screen, and you may fetch and execute instuctions merely by pressing ENTER. As each instruction is fetched, it is displayed at the top left of the screen. The format for this display is:

- Line 1: Memory address (in hex), followed by the first hex byte of the instruction at that address, followed by the Zilog mnemonics for that instruction.
- Subsequent lines (as needed): Memory address, followed by the hex byte at that address, repeated for as many lines as the instruction displayed has additional bytes. (For example, a three-byte instruction has two of these extra lines in its display.)

ENTER actually half-steps through the target machine language program. You see the FETCH and EXECUTE cycles as separate half-steps, each activated by pressing ENTER. After the EXECUTE cycle, another register display at the bottom right of the screen will show the contents of the registers after execution of the instruction; the BEFORE register display remains in the upper right corner of the screen, so that the effect of the instruction just executed upon the registers can be clearly seen. When the next instruction is fetched, the BEFORE display will change to contain the information of the AFTER display from the previous execution, and the AFTER display will be erased.

The register displays are largely self-explanatory. Each double register is presented with a designator followed by the (hex) contents of the register. The most important flags are also displayed as separate bits; their designators are CY for Carry, Z for Zero, S for Sign, and PV for Parity/oVerflow. Following the BC, DE, HL, IX, and IY displays are single hex bytes in parentheses; these represent the contents of the memory locations pointed to (respectively) by these 16-bit registers. For example, if you see DE: 48FB (E6) in the display, you know that the DE register pair contains 48FBH and that memory location 48FBH contains 0E6H. The two-byte hex number in parentheses following the SP (stack pointer) display is the number (or address) on the top of the stack. This number is presented in the natural form of high-order byte first, even though the high-order byte is in the memory cell whose address is one more than the contents of the SP register. The PC display shows the contents of the program counter, which points (in the BEFORE display) to the memory location from which the instruction has been fetched, or (in the AFTER display) to the memory location from which the next instruction is to be fetched.

In the SP mode you may fast-step through several (up to 99) instructions by typing a two-digit number in response to the "*" prompt. For example, entering "03" will cause three complete instructions to be executed in rapid succession. The register displays show the register contents before the first instruction is executed and after the last instruction has been executed. (This feature of MicroMind is especially helpful in working through a short loop that has to be executed many times.)

NOTE 1. Once you are in SP mode, you will not permanently exit from it except with a JP (JumP), CL (CalL), or IA (transfer to Instant Assembler) command. However, you can reinitialize it with another SP command.

NOTE 2. When you are in SP mode and the "*" prompt is displayed, pressing ENTER will always cause the next half-step (FETCH or EXECUTE) to be carried out. This is true even if you have made extensive use of other commands since your last step. To clear the left side of the screen so that the FETCH or EXECUTE cycle can be clearly observed, merely press BREAK.

NOTE 3. Not all hex numbers can be decoded as legitimate Z-80 instructions. In the unlikely event that MicroMind encounters such an indecipherable combination in the instruction stream while stepping, it will treat each byte as a NOP for execution (and print no mnemonic for that byte) until it arrives at the next recognizable instruction. Further discussion of this point will be found under the DS command in subsection 4.3 of this chapter.

XC (for eXeCute)
XC is operative only in step mode, and only if the last instruction fetched is a CALL, conditional CALL, or RST (restart). Its effect is to execute the called subroutine as a whole, without stepping. It is useful when the target subroutine has already been debugged. The register displays show the contents of the registers before the subroutine is called and after it has been executed. If the XC command is entered when it is inapplicable, it is simply ignored, and another "*" prompt is issued. If the fetched instruction is a conditional call, and if the condition is not met, the XC command merely has the effect of stepping through the conditional call instruction without executing the subroutine.

BD (for Blank Display)
This command is operative only in step mode. Its effect is to clear the screen (except for less than half a line in the extreme lower left corner), to transfer an abbreviated (to one line) display of the fetched instruction to the lower left corner of the screen, and to permit continued stepping with the target program in control of the screen (again, except for less than half a line in the lower left corner). Pressing ENTER now causes execution of the fetched instruction and fetching of the next instruction in sequence (full-stepping rather than the half-stepping of SP mode.) No register displays are available in BD sub-mode; the target program can be traced, but a detailed examination of its workings is no

longer possible.

To return from BD sub-mode to regular SP mode, merely press the BREAK key; there will be no loss of place or of continuity in stepping through the target program.

The BD sub-mode has been designed to allow you to see a target program print on the video screen. Since many instructions are normally required to post even a single character to the screen, single-stepping through a video display routine can be distressingly slow. Therefore, MicroMind provides three sub-commands in BD sub-mode to speed up the action:

(a) Depressing the R (for Run) key while in BD sub-mode causes stepping to occur at about 140 instructions per second. Releasing the R key terminates the fast-stepping and returns you to the single-stepping (via the ENTER key) of BD sub-mode.

(b) The S (for Seek subroutine) key has the fast-stepping effect of the R key, except that fast-stepping is terminated whenever a CALL, conditional CALL, or RST is encountered in the instruction stream. Often keyboard input requests are mixed in with video output, and stepping (even fast-stepping) through an input routine can be an aggravation. (You probably won't get the input in; hence, you'll probably never get out of the subroutine.) Use of the S key in BD sub-mode allows you to fast-step until a subroutine is reached, then pause long enough to use the X key.

(c) The X (for eXecute) key permits execution of the subroutine as a whole. The X key is to BD sub-mode what the XC command is to SP mode.

Here's an example of the use of the BD command and the R, S, and X sub-commands. With MicroMind running, enter the SP command and type in a FIRST ADDRESS of 0 (pressing ENTER to complete the entry of this one-digit address). You will then see that the DI instruction at (ROM) address 0 has been fetched -- you are in the TRS-80 startup procedure. Now enter the BD command. The screen is blanked and the DI instruction display is transferred to the lower left corner. Press the R key and watch the fast-stepping action as mirrored in the lower left corner display. The screen remains otherwise blank, but be patient. In about half a minute, the "MEMORY (or MEM) SIZE?" query will begin to appear on the screen, one character at a time. Now release the R key (before the question mark appears) and depress the S key. Alternately press the S and X keys until the cursor appears after the question mark. At this point you are executing the input subroutine; type "32500" (for the memory size) and press ENTER, returning control to MicroMind. Continue by depressing the R key and watch the continuation of the startup procedure, through the printing of "READY". Then press the BREAK key to return to SP mode.

RN (for RuN)

RN is also operative only in SP mode; this command enables fast-stepping (with a blank screen) to any designated terminal address. After you have entered the RN command, you will be asked for a "FINAL ADDRESS?". This is the address of the last instruction to be fetched. (The run will be ended when this instruction is fetched but not executed. The run starts, of course, from the instruction that you have reached in SP mode.) The FINAL ADDRESS should be entered in accordance with the RULE FOR THE ENTRY OF ADDRESSES given under the SP command.

The RN command permits you to execute rapidly a portion of the target program without leaving or reinitializing the SP mode, and with the assurance that your designated terminal point (which may even be in ROM) will be honored. When the final address has been reached in RN sub-mode, the fast-stepping will stop, and you will be in BD sub-mode.

If you wish to exit from the RN sub-mode before the terminal address is reached, press the BREAK key. You will then be in BD sub-mode, from which you may continue with single-stepping, or use the R, S, and X keys, or return to SP mode by pressing BREAK once more. Note, however, that if you now use the R key, it will return you to RN sub-mode (running to the designated final address); that is, releasing the R key now will not terminate the (still unsatisfied) run -- only the BREAK key can do that in this instance.

Here's an example of the use of the RN command. With MicroMind running, enter the SP command. Give a FIRST ADDRESS of 0, and then enter the RN command. To the "FINAL ADDRESS?" query, respond with "00187". (Note the two leading zeroes to indicate a decimal address -- "187" would be 187 hex, which is 391 decimal.) MicroMind will then fast-step through the ROM startup procedure, ending at the subroutine that accepts the input for the "MEMORY SIZE?" question. Press the X key to cause execution of that input routine. Type "32500" and press ENTER. Continue by pressing the R key, which now operates normally, since the run to 00187 has been satisfied. When you tire of this, press BREAK to return to SP mode and the facilities of MicroMind.

BK (for BreaKpoint)
The BK command allows you to set a breakpoint in RAM. After "# BK", you will be asked for an "ADDRESS?". This should be the address of the first byte of some instruction in the target machine language program. (MicroMind will not accept any address lower than 4000H.) MicroMind will then replace three bytes of the machine language program with a jump to a MicroMind entry point and will confirm the breakpoint with the message "BREAK AT (address)". (The three replaced bytes are saved for later restoration.) If the BK command is entered when a breakpoint is already in effect, the same message will appear (with the old breakpoint address) in rejecting the command.

Upon return from a breakpoint, an AFTER register display will appear in the lower right corner of the screen, showing the contents of the registers at the completion of the program segment terminated by the breakpoint.

RB (for Restore Breakpoint)
RB undoes the effect of BK. It may be used if you change your mind about the breakpoint that you have set, or it may be used after returning from a breakpoint. Since only one breakpoint may be in effect at any one time, RB might be used to restore an old breakpoint so that a new one may be set. When the RB command is entered, the confirmation "BREAK AT (address) RESTORED" will appear, assuming that a breakpoint was actually in existence; if there was no breakpoint, the legend "NO BREAK" will be printed on the screen.

SB (for Step from Breakpoint)
SB combines the effects of RB and SP, with the latter commencing at the address of the breakpoint. That is, the breakpoint will be restored, and the instruction at the address of the breakpoint will be the first one fetched for step-wise execution. SB is useful after a return from a breakpoint, at which time the registers will be in exactly the condition in which the program segment just executed has left them. If the SB command is entered when no breakpoint is in effect, the legend "NO BREAK" will appear on the screen.

JP (for JumP)
The JP command allows you to transfer control to any point in memory, including ROM. After "# JP", you will be asked for an "ADDRESS?". (To change your mind at this point, use the BREAK key to cancel the JP command.) When this address has been entered, the registers will be loaded with the values shown in your last register display (if you were in SP mode), and the jump will be taken to the specified address. JP is useful for returning to the Level II monitor (JP to 1A19H) and also in conjunction with a breakpoint. (After setting a breakpoint with BK, use JP to execute part of a machine language program and then return to MicroMind.)

It is obvious that caution must be exercised in the use of JP, since control is taken out of the hands of MicroMind. In particular, if the jump address is five digits (decimal), be certain that you have entered it correctly before typing the last digit, for the fifth digit automatically triggers the jump.

4.2. DEBUGGING ACCESSORY COMMANDS

CL (for CalL)
The CL command allows independent execution of any closed subroutine in ROM or RAM. After entering the CL command, you will be asked for an "ADDRESS?". Respond with the entry point address of the subroutine to be executed. As soon as the address entry is completed, the screen is cleared, the registers are loaded with the values shown in your last full register display (if you were in SP mode), and control is transferred to the target subroutine. (If you wish to adjust the contents of the registers before calling the subroutine, use the RG command, which is explained next.) The return will be to MicroMind. You may inspect the contents of the registers after execution of the subroutine by using the SP command (with any FIRST ADDRESS) to call up a full register display.

RG (for ReGisters) (on Model III use "LESS THAN" symbol for back arrow)
The RG command allows you to inspect and change the contents of a target program's registers. Following "* RG", you will be asked to name a register by the query "REG?". You may answer this question with "A", "CY" (for Carry), "Z" (for Zero), "S" (for Sign), "PV" (for Parity/oVerflow), "BC", "DE", "HL", "IX", "IY", or "SP" (for Stack Pointer). A two-letter name automatically triggers the display of the contents of the named register; if your request is for the A, Z, or S register, you must press ENTER to trigger the display. Note that the flags can be inspected and changed as conveniently as any of the other registers.

When the register name has been entered, its contents will be displayed to the right of the name, followed by a BACK ARROW prompt. To change the contents of the register, enter a new byte (in hex) for the A register, either "0" or "1" for a flag, or a new number (in address format -- either 5 decimal digits, or four or fewer hex digits) for a double register. (If you enter a new byte for the A register with only one hex digit, you will have to press ENTER to complete the entry.)

If you were in SP mode before using the RG command, and if you were between FETCH and EXECUTE cycles, then all register changes will be immediately reflected in the BEFORE register display. Such changes will not be shown, however, after an EXECUTE cycle -- they will appear after the next FETCH cycle.

If you don't wish to change the contents of a displayed register, just press ENTER. Completing an entry on one line (either by entering a change for the register or by pressing ENTER) will cause a new "REG?" query to appear on the next line. To exit RG mode, use the BREAK key, either in response to the BACK ARROW that prompts the register change or in response to the "REG?" query. You will be returned to SP mode if you were there before using RG, and you may then continue stepping through a program.

The RG command is useful in debugging for the following reason (among others): Often you will discover an error that adversely affects register contents. Rather than having to abort the debugging procedure to make a change in the program, you may make a note of the discovered error, use RG to set the registers right, and then continue stepping.

MM (for MeMory) (on Model III use "LESS THAN" symbol for back arrow)
MM allows you to inspect and change the contents of memory locations. (ROM may be inspected too, but not changed.) After "* MM", you will be asked for a "FIRST ADDRESS?", which again may be in either decimal or hex. After a memory cell is displayed (one hex byte following the hex address of the cell), a BACK ARROW prompt will appear, and you may change the contents of the cell by typing your new byte (which must be in hex). (If you type a single character here, the transaction must be completed by pressing ENTER. A two-character entry will automatically trigger the change.) If you don't wish to change the contents of the cell, just press ENTER. In either case, completion of the entry advances you to the next memory location.

(Model III displays a caret pointing up for UP ARROW and a broken vertical line for the DOWN ARROW.)

If you want to back up, or to advance more than one address, use the UP ARROW key, or the DOWN ARROW key, respectively, followed by the number of addresses that you wish to back up or advance. (These two characters are typed in lieu of a hex byte for changing the contents of the memory cell.) Following the UP or DOWN ARROW, digits 0-9 will have their natural effect, while letters A, B, C, ..., Z will back up or advance the address by 10, 11, 12, ..., 35 locations (respectively).

To exit from the MM mode, use the BREAK key. If you were previously in SP mode, you will be returned precisely to where you were before you used the MM command -- except, of course, that some memory cells may have been changed. (If you are between FETCH and EXECUTE cycles, the instruction fetched will remain the same even if you changed it in memory during the MM operation.) Memory changes will be immediately reflected in the BEFORE register display if (and only if) you are between FETCH and EXECUTE. (Remember that the register display shows the contents of those memory locations that are pointed to by the double registers BC, DE, HL, IX, IY, as well as the two-byte number on the top of the stack.)

P1 (for Page 1)

The P1 command is used to display a small block of memory. After "* P1", you will be asked for a "FIRST ADDRESS?". When you have responded with the address, 19 consecutive memory cells starting at this address will be displayed at the next-to-bottom line of the screen. (Page 2 will simultaneously be displayed at the bottom line of the screen.) The two-byte hex number at the extreme left of the P1 display line is the address of the first memory cell of the display.

When MicroMind has posted the P1 display, it will wait until you press another key; any key except ENTER will end the pause and cause another "*" prompt to appear. Pressing ENTER during this pause will advance the starting address of the P1 display by 10H (16 decimal), and this may be repeated for as long as desired. Thus, you may quickly scan a large section of memory.

When MicroMind is first activated, Page 1 will not be displayed until either the P1, P2, or SP command is invoked, or memory is changed with the MM command. When this occurs, both Page 1 and Page 2 will appear on the screen, and their displays will remain in evidence until you exit from MicroMind with a JP, CL, or IA command. (Each of these commands has the effect of turning off the page displays for later reentry of MicroMind.)

P2 (for Page 2)

P2 is exactly like P1, except that the display is on the bottom line of the screen. P1 and P2 may be set independently, so that you can keep an eye on two different regions of memory, which can be extremely helpful in debugging.

4.3. UTILITY AND TRANSFER COMMANDS

(for final location FFF use the MM command.)

FN (for Find Number) (MODEL III will not accept final address of FFFF or 65535)

The FN command allows you to find all occurrences of either a one- or two-byte number in a block of memory. (This can be especially useful for finding all program references to a certain address.) After "* FN", you will be asked for a "FIRST ADDRESS?". Respond with the starting address of the block of memory to be searched. When this has been accepted, MicroMind will ask you for a "FINAL ADDRESS?". Respond with the last address of the block of memory to be searched. Then the question "FIND?" will appear. Answer this with the number that you wish to locate in the specified memory block. (Entry of this number follows the RULE FOR THE ENTRY OF ADDRESSES.) If you enter three, four, or five characters here, MicroMind will search for a two-byte number (or address); if you enter only one or two characters, MicroMind will search for a one-byte (hex) number. (In the latter case, expect a lot of matches unless the memory block is small.) The addresses of all occurrences of your search number in the specified memory block will then be displayed, four to a line. (For a two-byte search number, the address shown will be that of the low-order byte.)

After MicroMind has reported all addresses corresponding to your search number, it will post another "FIND?" query. By entering another number here, you may continue searching the same block of memory, and you may repeat this for as many numbers as you please. To exit from FN (perhaps only to set new first and last addresses for another search), press BREAK.

DS (for DiSassemble)
The DS command enables you to see instructions in memory (including ROM) in their Zilog mnemonics. After "* DS", you will be asked for a "FIRST ADDRESS?". Respond with the starting address of the program that you wish to disassemble. The instruction at that address will then be disassembled and displayed in the format explained under the SP command. MicroMind will then pause, awaiting your next directive. Pressing the ENTER key here will cause the next instruction in memory to be displayed, and this may be continued for as long as you wish. Pressing any other key will end the DS mode and cause another "*" prompt to appear.

NOTE: Not all hex numbers can be decoded as legitimate Z-80 instructions. For example, the byte 0DDH by itself is meaningless -- it requires at least one following byte to give it meaning. And not all following bytes are legal; DD01 is not the beginning of any valid Z-80 instruction. When the disassembler encounters such a combination, it reports only the first byte (with no mnemonic) on a line, and then proceeds (as you press ENTER) one byte at a time until it finds a combination that it can decode. In disassembling data, this type of ambiguity can easily arise. When you have worked through the data, the MicroMind disassembler will quickly get back into "sync", though an instruction or two following the data may be misreported. In stepping through a program (with SP), the same impasse is possible, though far less likely than in random disassembly. In this unlikely event, MicroMind treats the unidentifiable code byte-by-byte -- turning each byte into a NOP for execution -- until it can get back into "sync".

AS (for AScii)
The AS command allows you to decode blocks of memory as ASCII characters. After "* AS", you will be asked for a "FIRST ADDRESS?". Respond with the starting address of the block of memory that you wish to examine. MicroMind will then display five lines of ten characters each and pause, awaiting your next directive. Pressing the ENTER key will cause another five lines of ten characters each to be displayed. (The starting address of each line appears at the left side of the display.) Pressing any key except ENTER will end the AS mode and bring forth another "*" prompt.

Here's an example of the use of the AS command: With MicroMind running, enter the AS command and give it an address of 1650. You will then be reading the start of the BASIC command table in ROM. Press ENTER repeatedly to scan this table. Note the small graphics block at the upper left corner of the initial letter of each command. This block indicates that the character in memory has bit 7 set. A graphics block at the left middle of a character indicates that it is actually a lower case character. A two-wide graphics block where a character should be indicates that the number in memory cannot be interpreted as an ASCII character. And, finally, a three-tall graphics block where a character should be indicates a carriage return.

HD (for Hex-to-Decimal)
After "* HD", you will be asked for a "HEX#?". Enter any hex number of not more than four (hex) digits, completing the entry by pressing ENTER if the number of digits typed is less than four. MicroMind will respond with the decimal equivalent of that hex number. It will then ask for another "HEX#?". Terminate this mode by pressing BREAK. (The hex-to-decimal converter of Chapter 3 is similar to this MicroMind routine.)

DH (for Decimal-to-Hex)
This conversion works like the HD above, except that now you will enter a decimal number of up to five digits, and the response will be its hex equivalent. (In either type of

conversion, if MicroMind can't decipher your input, it will simply ask for it again.)

IA (for transfer to Instant Assembler)
Use the IA command to transfer control to the assembly subsystem of Instant Assembler.

CHAPTER 5. QUESTIONS AND ANSWERS ON USE OF MICROMIND

(Question 1) When do I need to supply a leading zero with a numeric entry?
(Answer 1) A leading zero (or zeroes) is required when entering a decimal address of less than five digits; otherwise, MicroMind will treat the entry as a hexadecimal address. Also, when using the FN command to search for a two-byte hex number whose value is less than 100H, a leading zero will be necessary to tell MicroMind that it is a two-byte number (rather than a single byte) that you are searching for; if your two-byte search number is less than 10H, two leading zeroes will be required. Note that the "H" postfix is never required (or even allowed) for hexadecimal numbers.

(Q2) What are the character limits for the various responses to MicroMind querys?
(A2) MicroMind doesn't permit you to enter more characters than the field limit of the latest prompt or query. These field limits are:

- 5 for all address fields, 5 for the search number in FN, 5 for the fields to change the contents of double registers, and 5 for the decimal number in DH.
- 4 for the hex number in HD.
- 2 for the field to change the contents of the A register, 2 for the field to change the contents of a memory cell, and 2 for the field to enter a command after the "*" prompt.
- 1 for the field to change a flag bit.

In all cases, when the field limit has been reached, the entry is complete, and the subsequent action is immediately triggered.

(Q3) How can I correct an error in my entry?
(A3) Use of the BACK ARROW key allows you to erase and correct an error. EXCEPTION: A mistake in a character that satisfies the field limit cannot be recalled. Fortunately, such a mistake rarely has irrevocable consequences, except when using the JP (or CL) command. Be careful when JumPing!

(Q4) How can I exit from any command mode?
(A4) Use the BREAK key. (However, in SP mode BREAK merely clears the left side of the screen.)

(Q5) What does it mean when MicroMind repeats a prompt or query?
(A5) That MicroMind didn't recognize your response to its previous prompt or query.

(Q6) Why doesn't MicroMind accept any spaces in its input?
(A6) Because no spaces are needed in any of the formal responses to MicroMind's queries.

(Q7) Will use of other commands confuse the stepping process?
(A7) No. MicroMind will never forget where it is in the stepping process until you do a JP, CL, IA, SB, or another SP, and exiting from any of the other commands will automatically return you to where you were in the SP mode.

(Q8) How can I view all the registers at once?
(A8) By entering the SP command to call up a full register display. It isn't necessary to do any stepping to use this feature.

(Q9) How can I inspect the alternate register set?
(A9) You can't. However, there is no need to know what is in the alternate registers unless your target program first loads some of these registers, then does an EXX (or EX AF,AF'), and later exchanges registers again. By stepping through the first exchange, you will know what has been saved in the alternate registers; MicroMind will not change their contents.

(Q10) How can I change the contents of the Add/Subtract or Half-Carry flag?
(A10) You can't -- at least not easily. By inspecting the AF register (in a full register display), you can at least ascertain the state of these flags. It is highly unlikely that you will need to change that state.

(Q11) When are the P1 and P2 displays updated?
(A11) After each use of the MM command to change memory, and after each EXECUTE cycle in SP mode.

(Q12) Where may my target program put its stack?
(A12) MicroMind initializes a stack in low RAM for use of the target program; this stack suffices for nearly all debugging. If your program must set up its own stack, be sure that it does not employ any memory locations below the address given by Instant Assembler in its "1ST FREE MEM" report when the AM command is used. (This last remark applies to any storage areas that your program may establish.)

CHAPTER 6. EXAMPLE OF MICROMIND IN ACTION

The goal of this chapter is to encourage you to become familiar with MicroMind by practicing its operations on the hex-to-decimal conversion program that you composed with Instant Assembler in Chapter 3. To that end, a program of action is described, but detailed instructions for carrying out all of the steps are not given; refer to the explanations of Chapter 4 for any required additional help.

Load and run Instant Assembler. Use the RS command to read in the HDCONV tape that you recorded with the directions of Chapter 3. Assemble-into-memory (with the AM command), giving the program an origin of 7000. If you have a printer, make a listing of the assembled program with the PC command; this will be helpful in following the steps outlined below.

Now transfer to MicroMind; the command for this is "MD". Enter the SP command and an address of 7000. You are ready to commence stepping through the hex-to-decimal converter.

For a first run, a blank screen will allow you to follow the highlights of the action. Enter the BD command, and the screen will be cleared except for the CALL 1C9H instruction displayed at the lower left corner. Press the X key to execute this subroutine (which clears the screen). Then press ENTER three times to step through the instructions at 7003, 7006, and 7009. Type "X" to execute the video output routine called at 700C, and the program title will appear on the screen. Press the X key twice more to execute the next two subroutines (which merely move the screen display line down). Then step through the instruction at 7015, and type "X" to execute the next subroutine, which will display the "HEX#?" prompt. Press the X key again, and the cursor will appear -- you are now in the input routine. Enter a hex number of four digits, and you will be back in MicroMind, ready to step through the instruction at 701E. Continue stepping -- using the X key to execute each CALL that you encounter -- until the decimal equivalent of your hex entry has appeared and the instruction at 7012 has been fetched again.

Press the X key to execute the subroutine called at 7012, and step through the instruction at 7015. Now, instead of executing the video output routine called at 7018, step through it. That is, continue to press the ENTER key until you reach the instruction at 706E (CALL 33AH). Use the X key to execute this subroutine, then step (with the ENTER key) until you return to 706E. Continue in this fashion, using the X key each time you reach the instruction at 706E. You should see the "HEX#?" query take form on the screen one character at a time. When this has occurred, press BREAK to return to regular SP mode.

Now enter the SP command again, with an address of 7000. Step through the program with full register displays; use the ENTER key to fetch each instruction, and also to execute it -- unless it is a CALL, in which case use the XC command to execute the subroutine as a whole. Note that the screen displays of the hex-to-decimal converter will now flash and be gone; MicroMind needs the screen for its own displays. When you execute the keyboard input routine (called by the instruction at 701B), however, you will be able to see your input until the fourth digit is entered. That is because MicroMind is suspended while the input subroutine is executing. In stepping with full register displays, proceed very deliberately, observing how the registers are affected by each command.

It is instructive to step through the program again with a slight modification along the way. Enter the SP command and an address of 7000. Execute (XC) the instruction at 7000, and step through the instruction at 7003 (LD HL,3C14H). Now use the RG command to change the contents of the HL register pair to 3E14. Exit from RG mode with the BREAK key, enter the BD command, and continue stepping (using the X key to execute all CALLs). The effect of the register change is to move the initial display half way down the screen. When you have observed this, press the BREAK key.

As a final exercise, use the BK command to set a breakpoint at 70AC. (This is inside the keyboard input routine). Then use the JP command to transfer control to 7000. The hex-to-decimal converter will now execute without external support. However, as soon as you enter a first character in response to the "HEX#?" prompt, the breakpoint will take effect, and MicroMind will be in control again. (Do not be alarmed by the appearance of multiple cursors on the screen. The keyboard input routine has turned on the cursor, and one will appear at the end of nearly every line of display.) Now enter the SB command, and you can step through the processing of this input character. If you want to see how different input characters are processed, you can reinitialize the SP mode with an address of 70AE, then use the RG command to enter a character directly into the A register. Step from this point, and you will be able to observe the handling of the character. (This last procedure points up the fact that it is easy to back up -- or advance -- the stepping process by simply reinitializing the SP mode; frequently this needs to be accompanied by use of the RG command to set theregisters right for the new starting point.)

It should be evident that, in the process of single-stepping through a program, you will almost surely discover any error that exists. If the program resides in Instant Assembler's source buffer, it is a simple matter to transfer to Instant Assembler (with "IA"), correct the error in the source code, assemble-in-memory again, and return to MicroMind (with "MD") to continue debugging.

PART II. THE LINKING LOADER

GENERAL

Linking Loader is a machine language program that will load a module produced with Instant Assembler into any RAM location outside its own program and storage areas. It will also load and link multiple modules. (The modules that Linking Loader operates on are the source code modules recorded with the WS command of Instant Assembler.) In addition, Linking Loader will record and verify an object tape of the loaded program.

Linking Loader occupies approximately 2100 bytes at one end of RAM and requires a certain amount of memory (which is dynamically allocated as needed) adjacent to its program area for the storage of labels and their values. Object code is assembled and placed in memory in real time (as the input tape is read), so that no buffer space is required for this operation. In loading multiple modules, Linking Loader proceeds from the specified starting point toward its own storage area. It will stop -- with an "OUT OF MEM" report -- if it runs out of room. In a 16K RAM there is enough space to load a multi-segment program of 9000-10000 bytes. (The exact upper limit depends upon the sizes of the individual modules, the number of cross references between modules, and even the order in which the modules are loaded.)

Linking Loader is supplied in two versions. The Top-Down Loader resides at the bottom of RAM and loads programs downward from the specified top address. The Bottom-Up Loader occupies the top of RAM and loads programs upward from the specified bottom address.

CHAPTER 7. THE TOP-DOWN LOADER

7.1. LOADING INSTRUCTIONS

Turn on your computer and answer the "MEMORY (or MEM) SIZE?" query by pressing ENTER. With the computer in the READY state, position the program tape for reading the Top-Down Loader (immediately after Instant Assembler on your tape), and depress the PLAY key of the recorder. Type "SYSTEM" and press ENTER. In response to the "*?" prompt, type "LNKLDT" and press ENTER. After the tape has been read, another "*?" prompt will appear. Type "/" and press ENTER. Linking Loader will now be in control; the "#" prompt at the top left of the screen is a request for entering a command.

7.2. ENTRY POINT

The entry point of the Top-Down Loader is 17341 (43BDH). (Model III entry 17597, 44BDH)

7.3. LOADING SOURCE MODULES -- THE LD COMMAND

With the Top-Down Loader running, answer the "#" prompt with "LD" (for LoaD). (Linking Loader's four commands are, like Instant Assembler's, all two-letter commands.) Linking Loader will respond with the legend "LOAD-" and then ask for a "FINAL ADDRESS?". Answer with the memory address of where you want your program to end; it will load (without gaps) from this end point toward the bottom of RAM. (It follows that the higher you set the final address, the more room you will have to load the program.) The final address may be entered either in decimal or hexadecimal -- see the RULE FOR ENTRY OF ADDRESSES in subsection 4.1 of Chapter 4.

After you have specified the final address, Linking Loader will ask for a "TITLE?". Respond with the title of the first module to be loaded; have this module positioned in the cassette recorder for reading, and depress the PLAY key. Linking Loader will start to read the module as soon as you have completed the entry of the title (by pressing ENTER). If the load is error-free, Linking Loader will display "LOAD-" and ask for another "TITLE?". Position the next module for reading, enter its title, and press ENTER to cause it to be loaded. When the last module has been successfully loaded, press BREAK. At this point Linking Loader will report all assembly errors that it discovered. The format of this error report

and the procedure to be followed next will be explained in Chapter 9.

If any module fails to load properly, Linking Loader will issue the report "BAD", followed by the legend "LOAD-" and another request for a title. (If this was the first module, the request for a title will be preceded by a repeat request for the final address.) Rewind the tape, enter its title again, and try once more to load it. Be alert for this "BAD" message; if you fail to observe it, your loaded program might be a module short.

CHAPTER 8. THE BOTTOM-UP LOADER

8.1. ASSEMBLY AND LOADING INSTRUCTIONS

The Bottom-Up Loader is supplied in the form of an Instant Assembler source tape to facilitate loading it into the top of a RAM of any size. First load and run the Top-Down Loader, then use it to load the source module for the Bottom-Up Loader. The FINAL ADDRESS for the latter load should be 7FFD, BFFD, or FFFD, depending upon whether you have a 16K, 32K, or 48K RAM, respectively; the title for the load is "LNKLDB". After completion of this load, use the TP and VF commands (as explained in subsection 9.3 of Chapter 9) to record and verify an object tape of this program. Subsequently you will load the Bottom-Up Loader with the SYSTEM command of Level II.

8.2. ENTRY POINT

The entry point of the Bottom-Up Loader is 30651 (77BBH) in a 16K RAM, 47035 (0B7BBH) in a 32K RAM, or 63419 (0F7BBH) in a 48K RAM.

8.3. LOADING SOURCE MODULES -- THE LD COMMAND

Loading source modules with the Bottom-Up Loader is almost identical to the same operation with the Top-Down Loader, except that the load will commence with the question "ORIGIN?" (instead of "FINAL ADDRESS?"). Answer this question with the desired starting address of the assembled program. The Bottom-Up Loader will load the program from the specified origin toward the top of RAM. When all modules have been loaded, press BREAK in response to the last request for a title. At this point Linking Loader will report all errors that it found. The format of this error report and the procedure to be followed next will be explained in Chapter 9.

CHAPTER 9. ADDITIONAL FEATURES

9.1. ERROR REPORTS

After the last source module has been loaded and you have pressed the BREAK key, Linking Loader will give an error report in the following format:

```
INT ERRS: 001
EXT UNDEF SYMBS: 002
&MULT
&SRCE
```

Here, "INT ERRS" (for internal errors) gives the total of all errors resulting from undefined nonexternal symbols, relative jumps out of range, and relative jumps with targets that are labels defined externally (that is, not in the same module as the relative jump). Linking Loader will not try to link a jump of the last type even if it is within the allowed range of a relative jump. All of these internal errors should have been eliminated before the modules were recorded, since they would all be reported by the LE command.

"EXT UNDEF SYMBS" gives the number of external symbol references that have no corresponding labels to define them.. All these symbols are then listed below the count. In the above example, "&MULT" and "&SRCE" should have appeared as labels in some module (or modules), but did not. Obviously, you will have to correct these errors eventually.

There is one type of error that Linking Loader will not detect, and that is an external label that appears in more than one module -- a doubly defined external label. In supplying an address for an instruction that references such a label, Linking Loader will use the latest defined value -- that is, the value of the label in the most recently loaded module in which it appears. Thus, an error of this type may or may not result in an actual error in the assembled program. (Of course, the way to avoid the possibility of a real error of this type is to use each external label in only one module.)

9.2. FINDING SYMBOL VALUES

Following the report of errors, Linking Loader will ask you for a "SYMBOL?". If you respond with the name of any external label in the program that has just been loaded, Linking Loader will report the absolute memory address (in hex) of the instruction at that label and then ask you for another "SYMBOL?". (If you name a nonexternal label, or one that does not exist in the program, Linking Loader will respond "BAD" and then ask you for another "SYMBOL?".) You may thus learn the memory addresses of as many external labels as you please. When satisfied, press BREAK, and Linking Loader will post the "#" prompt, requesting another command.

9.3. OTHER COMMANDS

Besides the LD function, Linking Loader provides these commands:

EX (for EXit)
The EX command allows transfer of control to any point in memory. It works just like the JP command of MicroMind. After "# EX", you will be asked for an "ADDRESS?". Respond with the address to which you want control to be transferred. (This address may be in either decimal or hex, just as in MicroMind.) You may use the EX command to exit to the Level II monitor (EX to 1A19) or to execute a program that you have loaded with Linking Loader.

TP (for TaPe)
The TP command allows you to record a machine language program (in SYSTEM format) on cassette tape. You may use it to make an object tape of a program that you have loaded and linked with Linking Loader, or to record any other machine language program that is in memory.

After "# TP", Linking Loader will request a "FIRST ADDRESS?". This is the lowest memory address of the program that you wish to record, and it may be entered in either decimal (5 decimal digits) or hex (four or fewer hex digits). There is also a default option available for this address: If you are taping a program that you have just loaded with Linking Loader, you may press ENTER in response to the "FIRST ADDRESS?" query, and the correct beginning address will be automatically supplied. Next, Linking Loader will request a "FINAL ADDRESS?". This is the highest memory address of the program to be recorded, and it may be entered in either decimal or hex. Again, there is a default option for this address: If you are taping a program that you have just loaded with Linking Loader, you may press ENTER in response to the "FINAL ADDRESS?" query, and the correct end address will be automatically supplied. Linking Loader will then ask for an "ENTRY ADDRESS?", which is the address at which execution of the machine language program is to start when later it is loaded with the SYSTEM command. An address must be entered here (in either decimal or hex) even if it is meaningless. (If Linking Loader does not like any address that you enter, it will repeat its request for that address.)

Once the three addresses have been established, Linking Loader will request a "TITLE?"; type in any name of six or fewer characters. Before this title entry is completed (by pressing ENTER), be sure that the tape is correctly positioned in the recorder and that the RECORD and PLAY keys are depressed, for Linking Loader will begin recording immediately. (After the title has been entered, the contents of memory between -- and including -- the locations specified by your first and final addresses will be dumped to tape.)

After you have used the Top-Down Loader to load the source tape for the Bottom-Up Loader, you will want to record (with the TP command) the latter in SYSTEM format. The following table gives the necessary addresses for this recording.

RAM SIZE	FIRST ADDRESS	FINAL ADDRESS	ENTRY ADDRESS
16K	77BB	7FCD	77BB
32K	B7BB	BFCD	B7BB
48K	F7BB	FFCD	F7BB

VF (for VeriFy)

The VF command allows you to verify a machine language tape that you have just recorded with the TP command. Rewind the tape to the beginning of the recorded segment and type "VF" in response to the "#" prompt. Linking Loader will immediately start to try to verify the recording, so press the PLAY key on the recorder. If anything is wrong, "BAD" will be displayed, and you may re-record the program and try again to verify it. If the recording is all right, "GOOD" will be displayed when the verification is complete.

(Model III requires recording at 500 baud for TP and VF commands.)

CHAPTER 10. EXAMPLE OF LINKING LOADER IN ACTION

In this final chapter you will use the two versions of Linking Loader to load and link the 3-segment hex-to-decimal conversion program that you constructed in Chapter 3. The first part of that program calls subroutines in each of the other two parts; thus, Linking Loader will need to determine the addresses of those subroutines as it loads them and then plug those addresses into the calling instructions of Part 1 of the program.

Load and run the Top-Down Loader (using the SYSTEM command of Level II). Enter the LD command and type in "70E6" in answer to the "FINAL ADDRESS?" query. Now place the source tape for Part 3 of the hex-to-decimal converter in the cassette recorder and depress the PLAY key. In response to the "TITLE?" prompt, type in "HDCNV3" and press ENTER. Part 3 will then be loaded, and Linking Loader will request another "TITLE?". Place the source tape for Part 2 of the hex-to-decimal converter in the recorder, depress the PLAY key, and enter the title "HDCNV2". After Part 2 has been loaded, repeat with Part 1, which has the title "HDCNV1". When Linking Loader then asks for another title, press the BREAK key. The error report that appears should show no errors. Linking Loader will now ask you for a "SYMBOL?". Type "&BEGIN" -- the label of the starting instruction of the hex-to-decimal converter. The response from Linking Loader should be "ADDRESS: 7000". (If you were to make a tape of the program just loaded (using the TP command), you would need this address to respond to the "ENTRY ADDRESS?" query. You now see the reason for using an external label at the entry point of the hex-to-decimal converter; although &BEGIN is not referenced by either Part 2 or Part 3 of the program, it is referenced by the Linking Loader. Linking Loader cannot give you the value of any nonexternal label.)

Respond to the next "SYMBOL?" request by pressing the BREAK key. Then enter the EX command and transfer control to the hex-to-decimal converter (EX to 7000). After satisfying yourself that the program has been properly loaded and linked, use SHIFT-E to return to the Level II monitor. (Model III use BREAK)

The program that you have just loaded could have been located in any region of memory above the Top-Down Loader; you might wish to reload it into another area. Also, the three modules of the hex-to-decimal converter can be loaded in any order; you might want to repeat the linkage-loading procedure with a different order of loading. However, it is frequently true in top-down loading that one module defines the end of a storage area of indeterminate size that lies below the program area; in such a case, care must be exercised to insure that this module is the last one loaded. There are also circumstances in which one particular module must be the first one loaded. The main fact to keep in mind is that -- in top-down loading -- successive modules are loaded into successively lower regions of memory. (It does not follow, though, that a single module is loaded from higher to lower addresses; indeed, the reverse is true.)

If you have made an object tape of the Bottom-Up Loader, now is the time to load and run it. Enter the LD command, and give an origin of 5000. Then load the three modules of the hex-to-decimal converter in the order HDCNV1, HDCNV2, HDCNV3. (Any other order would work as well.) Again the error report should show no errors. Use the SYMBOL? feature to learn the entry point address (the value of &BEGIN), which should be 5000. Transfer to this entry point with the EX command, and finally return to the Level II monitor with SHIFT-E. (Model III use BREAK)

In bottom-up loading, successive modules are loaded into succesively higher memory locations; keep this fact in mind if the successful operation of the total program is dependent upon the relative positions of the segments in memory. It should also be clear that, the lower you set the origin for the load, the more memory space there will be for the program that you are loading.