REVAS

Release 3

Z80/8080 DISASSEMBLER

USER'S MANUAL

by

A.E. Hawley

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

For several years a disassembler named REVAS has been available for systems which use a Z80 central processor. It produces the mnemonics originated by Technical Design Labs: an extension of the Intel 8080 mnemonic set. It was originally designed to operate under a monitor (TDL's ZAPPLE and functionally similar ones) and was completely relocatable so that it could disassemble code resident anywhere in the CPU's memory space. It was modified to operate under CP/M, making elementary use of CP/M file handling capabilities. REVAS is by now a mature and effective program that is being used all over the world. It is good, but I felt that an even better disassembler could be written.

Users of the original REVAS have asked for a disassembler that handles either TDL or ZILOG mnemonic sets; they also wanted one that would disassemble a program too large for available memory space. Incorporation of these and many other suggestions from users would have resulted in major surgery to the architecture of the original REVAS program. Such surgery usually incurs the risk of unacceptable compromises, bugs, and inefficient code. The best approach seemed to be a complete rewrite, incorporating new data and control structures along with the proven functional features of the old REVAS.

REVAS3 is the result. The rest of this manual describes its use and, as seems necessary, the algorithms employed to implement some of the main functions.

The manual is arranged in three major sections: a tutorial section (OVERVIEW, GETTING STARTED, and DISASSEMBLY), a command description section, and an interface and configuration section (CHANGING REVAS). Ready reference is provided by the table of contents, Appendices, an index, and page headings.

The tutorial section introduces you to the language defined by REVAS3 commands and to the concepts involved in using REVAS3. The OVERVIEW chapter tells you how to invoke and exit from REVAS3, introduces you to the syntax and editing of REVAS3 commands, and defines many of the terms used in the rest of the manual. GETTING STARTED introduces you to some of the utility commands and leads you through a simple disassembly. DISASSEMBLY continues with more advanced tutorial examples which include the disassembly of a real program whose source code is not available.

In the command description chapters, a complete description of the function and forms of each command is presented. The emphasis is on how commands work so that you can make full and effective use of their power and flexibility.

The configuration chapter (CHANGING REVAS) is small, because it is not necessary to change REVAS3 to make it run in a standard CP/M system. You may have preferences regarding screen format parameters, list device characteristics, or the label and comment delimiters in a disassembly listing. If so, you can consult that chapter.

The appendix contains a formal command syntax description and lists of command words, reserved words, and parameter names. There is a memory map which you will find useful in understanding the organization of REVAS and its table structure. Lastly, there are some sample disassemblies which are referenced in the tutorial sections.

REVAS3 was written in assembly language. Assembly language programming for all but trivial programs has been described as an "exercise in masochism". To my wife Stephanie, I owe a debt of gratitude for her love and forbearance during the long programming and debugging period; I am sure that she looks on assembly language programming as an "exercise in sadism"!

What would life be like without friends? I am grateful for the encouragement and intellectual support that Bob Doolittle, Tom Gallant, and Al Krug extended during the development of REVAS3. Many of the capabilities included in the program were selected as a result of their observations and advice. The many discussions about programming styles, techniques, and architecture eased the task of selecting 'the right way' to implement REVAS3. Thanks, Tom, for the critical proofreading that improved the organization of this manual, made me stick to good english usage, and protected the reader from inaccuracies in the command descriptions. And thanks to Al Krug for the hours of frustration he spent running developmental versions with their inevitable bugs.

Excitement exists in my relationship with Echelon and the Z-System with REVAS3 becoming a Z-Tool.

It is my intent to supply software and documentation that is as useful and free of errors as possible. All known bugs have been exterminated from REVAS3. I am interested in improving the quality of REVAS3 and its documentation wherever possible. Thus, I will welcome and respond to comments and recommendations sent to the address below.

Al Hawley
2 February 1985
Echelon Team Member
6032 Chariton Avenue
Los Angeles, CA 90056

CHAPTER 2

REVAS3 OVERVIEW

The purpose of this chapter is to make you comfortable with the operation of the REVAS3 disassembler. The main concepts involved in disassembly are given here along with an introduction to some of the commands from REVAS3's rich set. Some of the terms used in later chapters are defined here. You want to know what the programs on your distribution disk are for and how to execute REVAS3. As you progress through the manual, the concepts and commands are expanded on in increasing detail. This chapter forms the foundation of understanding that will help you keep your bearings in a sea of detail. Study it before going on to actual examples.

Your distribution disk contains a file named REVAS.COM, a number of files with the extension ".MNE", and a "READ.ME" file. The ".MNE" files are overlays that provide interactive selection of mnemonic sets during disassembly. One of the sets (normally TDL.MNE) is already present in REVAS. Other files may also be present. All are described in the READ.ME file, which can be listed using the Z-System and CP/M 'TYPE' command.

REVAS3 executes under the Z-System and CP/M operating systems. (CP/M is a registered trademark of Digital Research, Inc.) Here are some of the features that you will find in REVAS3 as you become familiar with it:

* disassembly of programs ORG'd <u>anywhere</u>
* disassembly of memory or disk resident programs
* disassembly of programs too large to fit into memory
* automatic symbol generation
* automatic data type assignment
* assignmnet & control <u>real</u> labels
* include comments in the disassembly
* interactive, dynamic choice of mnemonic sets
* undocumented Z80 opcodes
* easy-to-remember command words & abbreviations
* symbolic command arguments
* multiple commands on a command line
* command macros
* control over source, destination, and names of files
* a calculator that operates in your choice of radix
* a HELP command reviews commands, parameters for you

In order to make effective use of a program like REVAS, you must know what functions it can perform. You must also know how to request those functions (the Command Set). The next few pages will introduce you to many of the basic functions and commands.

A SIMPLE DISASSEMBLY ALGORITHM

Let's review just what it is that we expect a disassembler to do. You are probably familiar with the 'L' command of Z-Tool ZDM or DDT CP/M debugger; it produces a list of assembly language mnemonics. Other debuggers are available which do about the same. We expect more from a disassembler, though. We expect a disassembler to display the address of each instruction, the code itself, and the entire instruction; and we expect to see labels and symbols where they are appropriate. We expect to see something like the PRN file that most assemblers produce, even including data ('DB' or '.BYTE') entries where the code does not represent instructions.

How does a disassembler produce such a result? The answer to that question will be a disassembly algorithm. The algorithm on page 2-4 is a simplified version of that used by REVAS. A good way to understand the process is to perform the disassembly manually, then relate the experience to the REVAS command set. The paragraphs which follow set up the logical environment within which the algorithm operates.

First, there must be some object code to disassemble, and it must be accessible for 'reading' on a byte-by-byte basis. We'll call it the Target Program. (Use XD.OBJ if you want a concrete example. Your debugger or monitor can produce a HEX listing for manual use. Also, see the example in the Appendix, page C-1.)

Bytes will be accessed serially from the target program code and supplied on demand to the disassembler. Associated with each byte is its memory address. A logical register (the Program Counter, or PC) contains the address of each byte as it is accessed.

The algorithm requests exactly the number of bytes required to produce a line of disassembled code. The line of code is translated into appropriate mnemonics, register names, numbers, and other symbols. The line is then sent to an output device for display (with appropriate formatting). The arguments of the instruction are recorded in the symbol table when appropriate. The process is repeated for each line of disassembly to be produced.

To execute the algorithm manually, you will need three sheets of paper. The first sheet is titled DISASSEMBLY LISTING and will contain the disassembled code. The second contains a single line divided into 7 fields. It is the 'scratch-pad' working space for the algorithm. Here's how it looks:

DISASSEMBLY LINE

ADDRESS:	CODE:	LABEL:	OPR:	OPA1:	OPA2:	COMMENT:

Here is the third sheet format. It provides for an indefinitely large number of lines. (In REVAS the number of lines is limited by the amount of available memory space.) Everything that the algorithm is required to "remember" is stored in the Symbol Table. The simplified REVAS algorithm does not use all the fields; they are included here for later reference.

SYMBOL TABLE

VALUE:	SYMBOL:	SYMBOL-TYPE:	DATA-TYPE:	MODE-LOCK:

Here are some typical entries for the first two fields:

0100	BEGIN	(BEGIN & COUNTR are symbols)
1A4C	COUNTR	(assigned by a human)

Here are the allowed entries for the other fields:

		SYNTHETIC	INSTRUCTION	UNLOCKED
		REAL	BYTE(or DB)	LOCKED
		NONE	WORD(or DW)	

There is one further requirement. The disassembler algorithm must have access to a table of op-codes which contains relevant information on each instruction: number of bytes, the equivalent mnemonic, register(s) referenced, and location of byte or word data bytes. This table is derived from information supplied by the manufacturer of your 8080/Z80 CPU. It will be called the op-code table. For manual disassembly, you must use the manufacturer's data, your assembler manual, or one of the several books available at computer suppliers.

Step 3 of the algorithm calls for generation of a synthetic symbol. Such a symbol is generated by concatenating one or two letters and the hex ASCII value field contents. For now, use the letter 'S' (for Symbol). A typical synthetic symbol is 'S12F8'.

Manual Disassembly Algorithm

1. Erase the contents of the disassembly line, then copy the value of the PC into the disassembly line address field.

2. Search the symbol table for a value field which matches the address field of the disassembly line. If none exists, skip to step 4. Otherwise, fill in the label field by continuing with step 3.

3. A symbol table entry has been found. If a symbol has been assigned, transfer it to the field; else make a synthetic symbol and transfer it to the field.

4. Transfer the next byte of code from the target to the code field and increment the PC.

5. Look up the byte in a table of op codes. If it is not found there, go to step 4.

6. An op code has been found. From data in the opcode table, determine the length of the instruction and finish transferring bytes from the target to the code field. The code field now contains the entire instruction.

7. Get the mnemonic for the instruction from the opcode table; enter it in the OPR field of the disassembly.

8. If OPA field entries are not required, go to step 9. Else fill the fields as required. If one of the fields is a 16 bit quantity which matches a value entry in the symbol table, then use the procedure of step 3 to replace that quantity with a Real or Synthetic symbol.

9. Place a comment field delimiter (';') in the comment field and then transfer the code bytes to the comment field, translating them into their ASCII equivalents. Ignore the high bit of each byte, and replace non-printing characters with ".".

10. Copy the line to the DISASSEMBLY LISTING.

11. If there are no 16 bit values in the operand fields, go to step 12. Otherwise, a 16 bit argument is present in one of the OPA fields. Search the symbol table for a record containing that value. If none is found, then write the 16 bit argument in the next empty value field of the symbol table list.

12. end of algorithm (Repeat from step 1 for each line)

It doesn't take long to become disenchanted with a manual disassembly; if you've tried it previously, you probably used a different procedure. This algorithm is worthwhile to study and understand because it brings to light a number of important observations about the process.

Notice the unconventional use of the comment field; it isn't being used for comments at all! The characters entered there are really a different way of interpreting the code from the target program: as ASCII characters rather than as instructions for the 8080 or Z80 CPU to execute. After all, some code bytes represent strings of ASCII data in real programs. This alternate interpretation in the comment field helps you to spot those areas easily. But there's a more important observation to make. THERE ARE A NUMBER OF POSSIBLE WAYS TO INTERPRET THE CODE. Disassembly is _not_ just a matter of translating op codes into mnemonics! We can classify segments of code into Data Types, each with its own MODE of interpretation. The four modes of interpretation that we will consider are INSTRUCTION, BYTE, WORD, and ASCII.

You will already have noticed that the simple algorithm makes no provision for 8-bit (BYTE or DB) and 16-bit (WORD or DW) data segments. That's not fatal for an assembler, but it sure is inconvenient for people!

In step 3 of the algorithm, the symbol table is searched to see if a label belongs in the label field. A similar search is made at step 8 in order to use any symbol found to replace a 16 bit argument. A symbol table is _necessary_ to store such information for later use. More significantly, notice that new symbol table information is only available for _later_ use. An argument whose value is less than the current address is called a backward reference. Only on a subsequent pass through the code will new backward references result in a symbol being generated in the label field.

In step 11, all 16-bit arguments are entered in the symbol table. Is that really what we want done? Some 16-bit arguments are really being used as numbers rather than addresses; such numbers should not result in symbol generation.

Although a field was reserved in the symbol table for a label, there is no provision for entering one there. The synthetic label of step 3 is not actually stored; it is generated as needed from the value field of the table. Similarly, no use was made of the data-type and mode-lock fields in the symbol table.

Clearly, a better algorithm is needed for execution by REVAS. There are at least two functions being performed by the simple algorithm; DISPLAY the disassembly, and BUILD the symbol tables. The display function includes the actual disassembly according to an INSTRUCTION mode algorithm in steps 4 through 9, after which the completed line is sent to an output device represented by the DISASSEMBLY LISTING. The Symbol Table BUILDing function occurs in step 11.

DISPLAY and BUILD are two of the principle commands in REVAS. The principle which solves the backward reference problem is simple: DISPLAY does not make symbol table entries, and BUILD does not produce disassembled output.

We'll get to the other problems by giving the REVAS algorithm. For this algorithm, REVAS uses an output buffer which is logically organized to contain one line of output with the same fields as illustrated for the "disassembly line", except that there are up to four OPA fields instead of two. The symbol table is a table in memory (initially empty) which is built and maintained by REVAS with your help. The SYMBOL-TYPE, DATA-TYPE, and MODE-LOCK fields are used to store information about interpretation of the code which starts at the address value in the symbol record. In the algorithm, the contents of these fields are called "attributes".

THE REVAS DISASSEMBLY ALGORITHM

1. Fill the line buffer with spaces, then enter the current program counter contents in the address field.

2. Search the symbol table for an entry whose value field is the same as the PC. If not found, go to step 5.

3. Table entry exists. Fetch attributes which specify:
   a) DATA-TYPE, i.e. which mode of interpretation is appropriate for the code which follows.
   b) SYMBOL-TYPE, Real or Synthetic Symbol or none at all

4. Change the current MODE to agree with data type from step 3a. If SYMBOL-TYPE is 'none', then go to step 5. Otherwise, enter a Real or synthetic symbol in the label field.

5. Using the current MODE (of interpretation), fill in the CODE, OPR, COMMENT, and OPA fields of the line buffer.

6. If DISPLAY, send line buffer to output device(s)
   If BUILD, make entries or changes to symbol tables.

7. end of algorithm

As before, the algorithm is for the disassembly of only one line, and the starting address must be supplied. Furthermore, 'current MODE' is not necessarily defined within the algorithm, so it too must be supplied. These parameters are variables within REVAS. Their values are initially determined by default and respecified in commands. The MODE is specified by the I,B, or W prefixes permitted with the DISPLAY, PRINT, and BUILD commands; the address range for disassembly is determined from arguments supplied with the commands.

The REVAS algorithm requires, in step 5, that there be several different algorithms available for interpretation of code, and that a means be provided for selection of the relevant algorithm. The selection mechanism is the MODE parameter which is in turn controlled both by keyboard entry and dynamically by information stored in the symbol table. Only the INSTRUCTION, BYTE, and WORD modes are implemented, and the ASCII interpretation is left in the comment field. The INSTRUCTION algorithm is equivalent to steps 4 through 9 of the first algorithm.

The BYTE algorithm is simpler; it need only supply a pseudo-op for the OPR field and reformat the data from the code field for the OPA fields. Output format restrictions limit the number of bytes per line to 4. The WORD algorithm is somewhat more complex, because it is dealing with words which must be replaced with symbols when appropriate.

How about the handling of 16 bit arguments which are not really to be interpreted as addresses? Two approaches are employed by REVAS to handle the problem. First, REVAS checks to see if the value is within the range of the target program's addresses, as determined by the values of the PS (Program Start) and PE (Program End) parameters. Automatic entry in the symbol table is skipped if the value is out of range. The second method involves the use of the KILL command. KILL removes entries from the symbol table. In subsequent disassemblies, a 16 bit quantity which has been KILLed will be shown as a hex value.

If REVAS stopped and waited for you to assign a symbol of your choice (a 'Real' symbol in the algorithm) each time it encountered a new 16 bit argument during BUILD, you would be aggravated beyond belief. You need time to analyze the disassembled code in order to assign meaningful symbols. The solution is to let REVAS go ahead and use synthetic labels during BUILD. Later, after you have displayed the resulting disassembly, you can use the EQUALS command to insert Real symbols. In fact, you don't even have to wait for BUILD to make a symbol table entry; you can use the EQUALS command at any time to assign a symbol to <u>any</u> 16-bit quantity. With the EQUALS command, you have absolute control over the contents of that 'symbol' field in the symbol table. For examples see the disassemblies in Appendix C.

During the BUILD portion of step 6, allusion is made to 'changes' to the symbol tables. Such changes are modification of the data type information stored there, and occur because of blind application of an algorithm to ambiguous code constructs. What is needed is a method to control such situations. One of the attributes in the symbol table is a "mode-lock". When locked, it prevents changes to the data type attribute during execution of the BUILD function. The MARK command controls the mode-lock.

In summary, BUILD, EQUALS, and KILL determine the main contents of the symbol tables; MARK gives you keyboard control over the effect of the DATA-TYPE in the tables. And DISPLAY gives you the main disassembly output.

Metalanguage

Throughout this manual, and particularly in the command description sections, certain symbols are used to avoid excessively wordy and confusing descriptions. These symbols are not a literal part of the command but serve to express alternatives and choices in command construction. In the case of CR and SP, the symbols express an entity which may be included in a command, but is otherwise difficult to describe. This collection of symbols is called a Metalanguage, and its definitions follow so that you can refer to them as necessary while reading the manual. The same set of metalanguage definitions will also be found in the Appendix.

!	= logical OR
[ ]	= optionally present
[...]	= present any number of times
< >	= defines syntactic unit
( )	= establishes logical grouping
CR	= ASCII CR (carriage return)
SP	= ASCII SP (input from space bar)
^	= next character is ASCII control character

Word

There are two definitions of "word" used in this manual. Which one is applicable depends on, and is usually apparent from, the context.

a) In a command string, a word is defined as a string of alphanumeric characters terminated by a space, comma, semicolon, or carriage return.

b) In a DATA context, a word is taken to mean 16 binary bits of data and is equivalent to two bytes. This is the definition used in most assemblers for 8 bit computers.

Number

A number is a word which is composed entirely of characters from the currently active radix. If the first character of a word entered from the keyboard is a digit (0 to 9), then that word will be interpreted by REVAS as a number. If the first character is not a digit, then REVAS will first attempt to identify the word as a command, label, reserved word, parameter name, or macro name before interpretation as a number.

Argument

An argument is a word distinguished primarily by its lexical position in a command string. Once the command word (defined below) is identified, all other words in a command string are considered arguments.

ENTRY: INVOKING REVAS

REVAS is invoked in the same manner as other utilities under the Z-System or CP/M operating system: by typing its name after the command prompt. Called in this manner, the target code that REVAS disassembles is assumed to be in some portion of your systems memory space. Naturally, the target code cannot overlay REVAS or the operating system.

As you might expect, an unambiguous file name may be included on the calling line from the system level:

```
        A> revas xd.obj
    or  A> revas xd         (REVAS assumes a .COM extension)
```

In such cases, the code from the named file is disassembled as if it were in memory. The file, however, never overlays memory outside the bounds of REVAS. The file may be as long as 64K bytes! This is the virtual memory mode of access. These forms, as well as other extensions will be discussed in section 8. For now, the simplest invocation will suffice:

```
        A> revas
```

Once loaded, REVAS will print a sign-on message, a status report, and finally a prompt (M#). The prompt indicates that the program is waiting for a line of input from the console: a command line.

Once a command line has been entered, REVAS begins command execution. During command execution, a few single-character commands are recognized. These are called instant commands. Their function is to permit stopping/restarting command execution and aborting further execution by returning to command mode.

EXIT FROM REVAS

The next most important thing to know about a program like REVAS is how to return gracefully to the operating system. There are two ways to leave REVAS; they are completely equivalent. The first, as your intuition has already suggested, is to type a ^C from the keyboard. This method only works when the ^C is the first character typed following REVAS' prompt; at other times ^C is completely ignored. The second method is to enter the command word QUIT.

RE-ENTRY

Provided that no other system transient commands have been executed, it is possible to re-enter REVAS and pick up where you left off prior to the '^C' or 'QUIT' exits from REVAS. Just save a file with zero length and '.COM' extension ('@.COM', for example) and execute it. REVAS, whose code is still at location 100, will be restarted without reinitializing files or tables.

COMMAND EDITING

For all but instant commands, REVAS uses the operating system buffered console input; the command buffer will hold 70 characters. The normal editing functions are active until the CR is entered at the end of the command line. You can correct typing errors using the system editing functions. For editing details, refer to your operating system manual.

There is one exception. Control-C (^C), as the first character of the input line, results in a return to the operating system without a warm boot operation. Since REVAS does not disturb the operating system routines, the warm boot is not necessary.

WORDS REVAS KNOWS

REVAS contains a built-in "dictionary" which contains three kinds of words: command words, reserved words, and parameter names. A list of the words in each category is given in Appendix A. In addition, user defined words are recognized once they are defined as Macro Names or Labels.

COMMAND FORMAT

A command is a string of words and/or numbers terminated by a semicolon or carriage return. Each command element (word or number) must be delimited by spaces or a comma. The rules for well formed commands are more flexible than just implied; more detailed descriptions are given in the command descriptions and in Appendix A (Formal Command Syntax, Page A-1).

A command contains exactly one command word, and as many other words or numbers as may be appropriate. One number (or other word) may precede the command word. The other words follow the command word. A Macro Name is treated as and may replace a command word once the Macro is defined. The 'other words' are called arguments. A number is a numeric argument; anything else can be called a symbolic argument. The argument which precedes a command word is referred to as the RPT argument. RPT stands for the word 'repeat'; the name reflects the fact that in many of the commands this argument specifies the number of times that disassembly of the next instruction is to be executed.

If you refer to the Formal Command Syntax in the Appendix, you will note that no reference is made to command words; a command is simply defined as a delimited sequence of arguments, or a macro name. Syntactically, that is correct. Functionally, either the first or second argument must be a command word.

Arguments not expected by the current command are ignored. For instance, the SAVE command requires no arguments; if arguments are supplied following the command word, they are ignored. If one argument precedes the command word (SAVE), that one will also be ignored. There is a limit, however, to the number of arguments which may precede a command word: no more than one. REVAS expects one of the first two words in a command string to be a command word; if neither word is, then the entire string is ignored and an irreverent reminder will be displayed on your console.

RESERVED WORDS

Reserved words are symbols whose value is predefined. ON and OFF are typical reserved words. Reserved words are much easier to use instead of their equivalent numeric value because they make mnemonic sense in the command. In fact, in any command that expects to find a numeric value as part of the command, a reserved word or user assigned label may be used instead of the number. Command words, Macro Names, and parameter names do not have associated numeric values, and cannot be used that way.

PARAMETER NAMES

Parameter names are used in only one context: as one of the arguments of the SET command. In any other context they will be unrecognized unless they have been assigned a value as a label. (That might confuse you, but REVAS will distinguish between the two usages of the same name. You can assign command names as labels, too, without conflict.)

SYMBOLS AND LABELS

A symbol is a string of six or less characters. A string (see <string> in the FORMAL COMMAND SYNTAX in Appendix A.) starts with an alphabetic character, and may contain all but a few of the printable alphanumeric set of characters. A symbol has an equivalent 16 bit numeric value. When a symbol is used as a label, its value is a memory address (location).

A label is a symbol which may appear in the label field of an assembly listing with proper termination.

There are two kinds of symbols: Real and Synthetic. A Real symbol is one which YOU create by a keyboard entry; it is stored and used literally. A Synthetic symbol is created by REVAS. It is created by concatenation of one or two letters and the Hex ASCII value of the symbol. Only the hex part of a synthetic symbol will be recognized in a command string.

During execution of the BUILD command, REVAS identifies 16 bit quantities in the operand field of machine language instructions. Each such newly identified quantity is added to a table in free memory above REVAS' disk buffer areas. On subsequent DISPLAY of a disassembly, a Synthetic Symbol (Synthetic Label) is generated and displayed wherever its 16 bit value would have occurred. If you have created a real label with the EQUALS command, then that label will be displayed instead of the synthetic label. Labels and Symbols are thus associated each with a specific numeric value; that value is usually an address.

MACRO NAMES

A macro is a command line which has been assigned a MACRO NAME using the MACRO command. A macro name may contain from one to six alphanumeric characters. Once assigned, the macro name may be used as a command word in a command string, where it results in execution of all the commands included in its definition.

ABBREVIATIONS

Command words may be abbreviated by truncation. For example, the word DISPLAY may be shortened to any of the following:

```
        D, DI, DIS, DISP, DISPL, DISPLA
```

The names of parameters in the SET (or TURN) command may be similarly abbreviated. Reserved words may be abbreviated to 2 letters, but will not be recognized if abbreviated to 1 letter.

Labels, Symbols, and Macro names may not be abbreviated. These symbols are all assigned by the user and are treated literally by REVAS. If you assign a macro name which is a possible abbreviation of a command word, the macro will not be executed when you include it in a command string; instead, the command will be executed.

Here is the algorithm REVAS uses to recognize an abbreviation. The command, parameter, or reserved word list is searched in the order given in the HELP listing, and the first match found is the one used by REVAS. Thus, you must include enough letters of the word to avoid ambiguity.

Appendix A contains lists of all the key words that REVAS uses. In those listings, the smallest unambiguous abbreviation is shown in parentheses after each word. The HELP command produces similar lists on your console or printer. In the HELP listings, the key words are shown in mixed upper and lower case; the upper case portion of each word is the shortest possible abbreviation.

COMMAND PARSING & INTERPRETATION

a) The first <arg> in a command is compared with entries in the command word list. If a match (possibly abbreviated) is found, then that argument is identified as the command word, and the command list is excluded from any further searches.

If a match is not found, then
b) the symbol table and the reserved word table are searched (in that order). If a match is found, then the argument has been identified and the parsing is continued with the next argument. If a match is not found, then the argument is assumed to be a number, and it is converted to binary according to the current input radix; parsing is continued with the next argument.

If the first argument was a command word, then all lexically subsequent arguments are identified by the same procedure as in (b). If the first argument was NOT a command word, the second is assumed to be; the command word list is searched. If no match is found, then an error condition exists and parsing is terminated with an error message. If a match is found, then the command is identified, and subsequent arguments are identified as in (b).

Parsing is terminated when a semicolon or end-of-line is encountered.

When Macros have been defined, the list of Macro names is searched as if it were appended to the command word list.

MEMORY USAGE

Appendix B contains a memory map which will help you to visualize the organization of the major parts of REVAS. Many of the addresses given on the map are symbolic because they are different in the various versions and revisions of REVAS that may follow the printing of this manual. You will not need these addresses to configure REVAS for your system. When REVAS is first invoked without any arguments, one of the reports issued by the implicit STATUS display is the address of the first byte of free memory. That address is where the first byte of the symbol tables will be stored; it immediately follows the 128 byte buffer reserved for the symbol table header (HDR, on the memory map). Later, after symbol table entries have been generated, the 'free memory' report is simply an indicator of the extent to which you have utilized your available memory space.

When you ERASE or KILL symbol table entries, the space so liberated is added to a list of empty records that is maintained in the symbol table area. Subsequent new symbol table entries will utilize space from the empty record list if possible; only when insufficient space exists in the empty record list will new entries use memory at the 'free' location and thereby increase the size of the symbol tables. If you use the STATUS command frequently you will be able to observe the incremental changes in symbol table size. There is no mechanism provided for reducing the size of the symbol tables short of complete annihilation (the REINIT command).

MNEMONICS

Since Z80 code is a superset of that used by the 8080 CPU, REVAS 3 disassembles code intended for either. The mnemonic sets supported by REVAS 3 include the set introduced by TDL and the ZILOG set. Several .MNE files will be found on your distribution disk. TDL.MNE is the overlay file which produces TDL mnemonics; Z80.MNE produces ZILOG mnemonics. When the TDL mnemonics are chosen, the pseudo-ops produced are consistent with the TDL assembler (.BYTE, .WORD, and .END). When ZILOG mnemonics are chosen the corresponding pseudo-ops are DB, DW, and END. You can interactively select among the mnemonic sets which are present on the default disk drive using the SET MNE ... command. If you prefer, you can use your system debugger to change the default MNE set to one of the others. Just follow the directions in the READ.ME file or in chapter 10 (CHANGING REVAS) of this manual.

You may have occasion to disassemble Z80 code that makes use of the undocumented Z80 opcodes. Or, you may encounter them during DISPLAY of a code segment that contains data. REVAS 3 detects and displays these opcodes using the 5 new mnemonics listed below. The undocumented opcodes were described by Daniel R. Lunsford in Dr. Dobb's Journal, Number 44, April 1980 (Vol 5, Issue 4), p 47. See also a more recent discussion of them in Microcomputing, April 1981, page 58, "Secret Codes Revealed" by Edwin E. Freed.

HX, HY	=	high order bytes of IX and IY registers
LX, LY	=	low order bytes of IX an IY registers
SLLR	=	shift left, set low bit to one (Freed calls this one RLO)

SUMMARY

In this chapter, you have been introduced to the strategy that REVAS employs in performing a disassembly. The terms 'command', 'word', 'number', 'argument', 'symbol', and 'label' have been defined. The concepts of command words, reserved words, and parameters have been described. Some notes about memory usage have been presented, and the command parsing algorithm has been given. Finally, the unique opcodes produced by REVAS were introduced.

I hope that this list of diverse subjects serves its intended purpose: establishing a common ground of comprehension so that the more detailed treatment in following chapters will be clear and unambiguous. In the next chapter, we will go through a simple set of operations which will give you a more intuitive feel for REVAS operation. You will have an opportunity to experience the practical aspect of some of the concepts presented so far. So, turn on your computer and try out REVAS as you read Chapter 3.

CHAPTER 3

GETTING STARTED

Before trying to execute REVAS, make sure that you have a copy of the distribution disk (marked with the copyright notice, of course) as a working master, and that the distribution disk is safely archived. If you are using Z-System or CP/M Version 2.2, all the files on both disks should be marked R/O (READ ONLY). Now transfer a copy of REVAS.COM onto a blank disk along with your operating system and any system utilities you might need. You may also wish to copy one or more of the .MNE files (You will be using PIP; be sure to include the [O] switch, since these files contain binary code!). On the Master Disk is a file named XD.OBJ. If you wish to duplicate the examples that follow, you should also transfer that file to the Test Disk. If your system uses more than one disk drive, make sure that there is nothing irreplaceable on the alternate drives. These are the precautions that I always follow when first trying out new software, and apply to ANY program. REVAS does not eat disks, destroy files, or cause system crashes; after initial trial it can be used like any other utility program under Z-System and CP/M.

REVAS is executed just like any other transient program under Z-System or CP/M. Assuming drive A: is logged in, type

```
        A>revas            ('A>' is the system prompt)
```

On your terminal you should now see the copyright notice, a message to type HELP for assistance, and a display of STATUS information. On the last line will be the REVAS prompt:

```
        M#
```

The second important command you need to know is how to get out of REVAS and back to the operating system level; there are two ways. Either type "quit" or "^c" (control-c). They are exactly equivalent in function, but ^c only works when it is the first character following the prompt. Try one, reinvoke REVAS, try the other, reinvoke REVAS, and then try the HELP command:

```
        M# QUIT
        A> REVAS
        .
        .
        M# ^C
        A> REVAS
        .
        .
        M# help
```

Now try the three commands suggested by the main HELP command listing. Count the number of lines displayed before the continuation message. You can change that number to anything that you desire by preceding the "H" by the required number. Try the following sequence of commands:

```
M# 5 h c     (sets screen paging to 5 lines)
M# h p       (See? REVAS remembers!)
M# 8 h
M# h p
```

You now know how to invoke REVAS and exit from it, and how to get the HELP functions. Did you notice that after each command word in the HELP listing is the shortest abbreviation that you can use for that word? You have also introduced yourself to the idea of an argument that precedes the command word in REVAS commands. For the HELP command, this argument updates the parameter HLINES which you saw in the parameter listing when you typed "h p".

Now let's explore the STATUS command. Try the following command sequence:

```
M# status
M# s all
M# s hlines
M# s c
```

This is the command that lets you check on the state of REVAS' internal parameters. You can, if you wish, change the values assigned to these parameters by using the SET command. Try the following commands:

```
M# set echo on; set hlines 5
M# stat argrad; h p
```

Notice that you can issue multiple commands in a command line by separating them with a ';'. Also notice the connection between 'argrad', the number following 'hlines' above, and the number of lines displayed by the 'H' command.

REVAS considers the command words SET and TURN to be completely equivalent; you can use the one which seems most natural in the command string context. To activate the command-echo feature above, you could have typed:

```
M# turn echo on; set hlines 5
```

You have seen the effect of the 'ECHO' parameter. You should be able to turn the command echo feature off now. Do it, and in the same line set HLINES to the value that seems right for your screen display.

Now that you're getting the feel of the REVAS command format, let's do some disassembling. We'll disassemble the start of REVAS itself, but for now don't bother with trying to analyse the code itself. (You can feel free to do that later, after you have mastered the instruction set.) Try the following command sequence:

```
        M# d 100
```

How many lines were displayed? You can control the number of lines displayed by including a REPEAT (abbreviated RPT) argument in front of the command, as in the following:

```
        M# 4 d 100       (the '4' is the RPT argument)

        M# d
        M# d 100 110
        M# d
        M# 10 d
        M# d 100
```

Experiment with the commands (particularly the DISPLAY command just demonstrated) we've covered so far until they become familiar. Use different arguments. Look at the STATUS display after each command and see the relation to the displayed values and the commands you have executed. You will notice that the RPT argument is not included in the parameter list. That's because there's never any reason to SET it independently; its value is always controlled by being specified as the argument which precedes a DISPLAY class command.

If you specify a value of 0 (zero) for the RPT argument, then the number of lines displayed will be essentially infinite. How do you stop the process? That's where the 'instant commands' come in. Instant commands are recognized at any time except during entry of a command string:

^S!s!S Stops current display activity. Display resumes following any other keyboard input (except the instant commands).

^E!e!E Aborts the current command execution and continues with the next (if any) command.

^X!x!X Aborts the entire command line and prompts for another command. (If the command line only contains one command, then 'E' and 'X' are equivalent.)

Note: These commands use direct input, bypassing the system input editing procedures (^C has no effect).

Now issue the following command and use 'S' to stop the disassembly, and any key except E, S, or X to restart; use 'E' or 'X' to get back to the REVAS command level.

```
        M# 0 d 100 114
        M# d      (here's where instant commands are needed)
```

This use of the RPT argument is good for scanning a long section of code rapidly. It can cause trouble, however, if the console parameter is OFF (as it is during the BUILD command) because you can't see what is happening. It's a good idea to re-establish the RPT argument at some finite value (such as a screen full of lines) by issuing another display command like:

```
        M# 15 d
```

Are you ready now for a real file? OK, let's try the XD.OBJ file. Try the following:

```
        M# pgm xd
```

OOPS! If the filename extension is not given, REVAS expects it to be 'COM', and then can't find xd.com on the disk! OK, then, let's try it again:

```
        V# pgm xd.obj
```

By the way, did you notice the change in prompt character? The 'V' means that REVAS is now in virtual memory mode; it reads segments of code from the disk file (as required by the disassembly) instead of from absolute memory locations. If you disassemble REVAS, you will find how it's done among the disk file handling routines.

OK, back to business. Use the STATUS command to see the values of PS(program start) and PE(pgm end) in the line starting with V#. How do they relate to the length of the file XD.OBJ on disk? (Hint: subtract PE from PS; that's HEX arithmetic, and you can do it with the CALC command) Also note that STATUS informs you of the file names currently active.

Now use the parameter setting command (T or SE) to turn on the PRINTER; then give the STATUS command again. If your system doesn't have a list device (or if you don't want printed output) turn the Printer back off. Use the STATUS command to be sure you were successful.

Go ahead and use the DISPLAY command to scan the code. Start with:

```
        V# d   (you did set the RPT argument, didn't you?)
```

Notice that the display started at address 100H. That's because the default starting address was automatically set to the value of PS. It does that in Memory mode, too.

As you disassemble beyond address 500H, you will notice a disk access. Don't panic! REVAS just read in the next code segment. Now stop the disassembly (instant command if necessary), and try:

```
        V# d 120    (or some other address less than 4ffH)
```

There was another disk access, as REVAS automatically read in the appropriate code segment.

Now try disassembling code outside the range of addresses defined by PS and PE. What happens? Why does it make no sense to disassemble outside the PS-PE range? (Hint: where does the next code segment come from?)

So far, it's just a little better than using the disassembly feature of your debugger program. Right? Don't give up yet; read the section of this manual on the DISPLAY commands. Experiment with BDisplay and WDisplay to see what they do. After using each form, display a few lines with the D command (no prefix). Now try the IDisplay command, then try Display again. Experiment until you feel comfortable with the commands discussed up to this point. Don't be tempted to try other commands until you are familiar with these.

At this point, you have become acquainted with the basic commands and command format that REVAS understands. You can invoke REVAS and return to the operating system. You can perform rudimentary disassemblies of memory resident code and of code from a disk file. You have been introduced to the parameters that control many of the functions in REVAS. You have used two of the Reserved Words (ON and OFF) to supply values (0ffh and 00) for the SET command. And, for those (rare?) times when you need it, you can use the HELP command to refresh your memory.

Now you are ready to explore the other commands and features. While you are exploring, make frequent use of the Status command; the parameters that it displays tell you what is going on. Start with the MARK, KILL, and EQUALS commands and use Display frequently to see their effects. Once Mark or Equals has been used, a symbol table will have been started, and you can save it with the SAVE command. Read about each command and understand it before going to the next. The next chapter, DISASSEMBLY, will be of particular help with the BUILD command. You have already met with some of the Utility commands. Learn the rest at your leisure. As you get acquainted with REVAS, you will find that "tearing apart" some mysterious code (or entire programs) produces disassemblies that are more complete and understandable than with anything you have seen before!

CHAPTER 4

DISASSEMBLY WITH REVAS

FIELD NAMES

As we have seen in 'GETTING STARTED', REVAS disassembles code to produce a listing which contains one instruction per line. One such disassembled instruction is shown below. It is the first instruction from the XD.OBJ file. The line is composed of seven fields; the name of each field appears just above its contents.

```
ADDR CODE        LABEL   OPR     OPA1,OPA2              CMT
0100  21 0000             LXI     H,0000H               ;!..
```

The ADDR field contains the address in hexadecimal of the memory location of the first byte of the code shown in the CODE field. The LABEL field is blank because no label has been assigned to the instruction at this address. The code in the CODE field has been translated into the instruction which appears in the next three fields: the Operator (OPR) mnemonic, and the two operands in the OPA1 and OPA2 fields. For many instructions one or both of the operand fields will be blank. The comment (CMT) field contains a different kind of translation: the ASCII equivalent of the hex values in the code field. For this purpose, the high order bit of each byte is ignored and non-printing characters are replaced with periods('.').

These field name assignments correspond to the terminology used for the listing produced by many assemblers. The source text for an assembler includes only the label, operator, operand, and comment fields. In order to use the output of REVAS for re-assembly, the address and code fields must be suppressed; the comment field is of no use to the assembler, so it can be suppressed also.

DESTINATION OF THE DISASSEMBLY

The disassembly can be sent to the console, the printer, the punch, or a disk file by turning the appropriate switch parameter(s) on. In all cases, suppression of the address, code, and comment fields is controlled by the ASMFLAG switch parameter; when ASMFLAG is turned ON, the fields are suppressed and will not appear in the output. If, instead, only the CMT switch is turned OFF then all but the comment field will be displayed or sent to the .LST file. User supplied comments, each of which is on a line by itself, are never suppressed. If the ECHO switch has been turned ON then each command will be repeated on the output just before it is executed.

Disassembly is sent to the .LST file by turning ON the LSTFILE switch. When the switch is turned OFF, transmission to the file ceases, but the file is not closed. Subsequent output can be appended to the file by turning the switch back on. The file will be closed when you exit from REVAS. You may also close the file with the CLOSE command. The file manipulation commands are more fully explained in chapter 8. See also the LSTFILE parameter description in chapter 9.

Another way to send a disassembly to the .LST file is by means of the WRITE command. The WRITE command opens the file, sends the disassembly to the file with the ASMFLAG ON, closes the file, then turns the ASMFLAG OFF. Any previous contents of the file are overwritten and lost.

CODE INTERPRETATION

The remark above that the comment field is a different translation of the code introduces an important concept in disassembly. Any given segment of code in memory could have been generated by a number of different types of source statements in the assembler source, and the code itself carries no information about how it was generated. The three bytes of code at address 100 could have been generated exactly as shown by an assembler using Intel mnemonics, but exactly the same code would have been generated if an entirely different assembler using Zilog mnemonics were used. Likewise, to emphasize the point, it could have been the result of a .BYTE or .WORD (equivalent to DB and DW pseudo-ops in many assemblers) data area in which the contents of the memory locations are defined at assembly time, or of an assembler pseudo-op that simply sets aside memory locations without initialization. In the latter case, any random bytes present at that location would have been included. The point is that there are a number of possible ways to translate the raw code: to instructions or to one of several kinds of data. REVAS must "know" at all times which translation algorithm to use in order to supply the disassembly that makes the most sense to you. REVAS can translate to straight ASCII (in the comment field), to Instructions, to byte oriented data, and to word oriented data.

The IDISPLAY command tells REVAS to use the Instruction translation algorithm until a new mode is requested or becomes appropriate. The BDISPLAY and WDISPLAY commands similarly request disassembly to byte and word type data modes. The DISPLAY command requests disassembly in the most recently specified mode.

DISASSEMBLY OF XD.OBJ

An initial disassembly of a portion of XD.OBJ is shown in Appendix C. It was produced by the following command line (after invoking REVAS):

```
        M# PGM XD.OBJ ; 55 PRINT
```

Notice that the "most recently specified" mode must have been Instruction. That's because until otherwise requested, the default disassembly mode is "Instruction". Notice also that the listing is broken up by blank lines inserted after each unconditional branch instruction. That's because such an instruction breaks the flow of program instructions; following instructions are almost certainly not related to those preceding the hard branch. The blank line helps you to visualize program structure.

Look at the first five instructions. Clearly, the CPU stack pointer is being saved at location 06BBH in the first three, then a new stack is being designated in the fourth instruction. Finally, there is an unconditional jump to location 0135H. Nothing strange here! Scanning on down the listing, the code "looks" sensible until we come to address 0160. Why would anyone load the accumulator twice from the same location? And what sense does it make to have 8 JRNZ instructions in a row? A look at the comment field doesn't help; all the bytes are non-printing. But look at the code itself; ØD is a carriage return in ASCII and ØA is a line-feed. And the JRNZ's are actually a string of ASCII spaces. This must be a data area! But where does it end? You can probably figure it out by listing XD.OBJ yourself until you find code that makes sense, but let's instead consider a much more powerful tool that REVAS has for you.

LABELS

Each of the 16 bit operands encountered so far is potentially a pointer to a segment of code; in other words, they could be used as labels. Assemblers don't like labels that start with a number, and they generally accept a label that is six characters long. If REVAS could put one or two letters in front of the 16 bit number (in hex) and supply a suitable label delimiter, that entity could be used as a label in the label field at the appropriate address. REVAS would also have to 'remember' to consistently use such a synthetic label in other operand fields where appropriate. If such assignments were made indiscriminately during the DISPLAY command, a whole bunch of incorrect labels would have been generated at the string of JRNZ commands at location 0163 to 0171, though. We need a special command so the process can be controlled. That command is the BUILD command, whose other forms (IBUILD, BBUILD, AND WBUILD) are analogous to the similar forms of the DISPLAY command.

SYMBOL TABLES

Here's how it works. We have decided that all the code from 0100 through 015D looks like valid instructions. Use the following command:

```
        V#ibuild 100 15d
```

All the code in the range specified is disassembled. The disassembly is not sent to any output device; instead, a slash is displayed on the console for each 60 lines of disassembly processed as a 'working' indication. (see the BUILD command description.) Each 16 bit argument found in an operand field is examined to see if it is within the range of the program. If it is, then it is stored as an entry in the Symbol Table in memory above REVAS. In the same record, mode information is stored which identifies the nature of the instruction which referenced this value. Certain instructions characteristically reference data areas; others always reference an instruction. Also, this table entry contains a marker which indicates that a symbol is to be generated.

The 16 bit value mentioned above is the value associated with a symbol or label. The marker which indicates that a symbol is to be generated also indicates whether the symbol is to be synthetic or is one which you have designated with the EQUALS command. When a synthetic symbol is required during a disassembly, it is generated from the 16 bit value. The synthetic label itself is not stored here; only its 16 bit value is stored. If you have assigned a symbolic name, then that name is the symbol which will be used by REVAS.

The table in which the address record is stored is a binary tree which shares high memory with other binary trees. The key on which the tree is organized is the 16 bit argument referenced above. Another tree contains user assigned labels (discussed below); the records in these two trees are cross-referenced by appropriate pointers. A third tree stores command macros when they are defined by the user. There is no connection between this tree and the first two.

The collection of tables is referred to as 'Symbol Tables' in this manual. In some cases where the context permits, the word 'Tables' is used and should cause no confusion with other tables in REVAS such as command or parameter lists.

After the BUILD command has been executed, we can look at the results with the DISPLAY command. Do it, displaying a screenfull at a time, starting with address 100 and stopping at address 18F. The disassembly you get will be similar to the second listing in the appendix. Notice that the zero argument at 100 was not assigned as a symbol, but remains as a hex constant. That's because it is outside the bounds of the target code (XD.OBJ). Likewise, the argument of the instruction at 011C has also been treated as a constant for the same reason. The CALL 0005H instruction at 0137 is another instance.

The load and store type instructions which have numeric operands are now displayed with synthetic symbols starting with the letters 'UT' and ending with the value of the argument (in hex). The arguments of branch type instructions (as at 010A and 0123) start with the letter 'S'.

These letters all have meanings. The 'S' stands for 'symbol', and implies that the code at the address represented by the symbol is an instruction. The letter 'T' stands for 'table', and implies that as a label its associated code is to be interpreted as BYTEs of data. The letter 'W' implies a table of data organized as WORDs.

When a synthetic label in the label field contains an 'S', the DISPLAY procedure will automatically switch to the instruction mode of disassembly; when a synthetic label contains a 'T', the switch is to byte mode disassembly. For a 'W', the switch is to WORD mode.

The letter 'U' stands for 'uncommitted', 'uncertain', or 'unlocked' (take your choice!). A label is 'unlocked' if REVAS can automatically interchange 'T','S', and 'W'. THE ABSENCE OF A 'U' IN A SYNTHETIC LABEL(SYMBOL) MEANS THAT REVAS CAN NO LONGER MODIFY THE MODE INFORMATION ALGORITHMICALLY; ANY CHANGE IN THE MODE MUST BE BY USER REQUEST. You can specify (with the MARK command) which data type the label will imply; and you may do so again if you change your mind. When you specify a data mode using the MARK command, that mode is locked. It can be changed by another MARK command, and unlocked by the UNLOCK command. The MARK, LOCK, and UNLOCK commands give you absolute control of the disassembly process.

At the start of a line of disassembly, the Program Counter (PC) contains the address of the first byte of code. The first responsibility of the disassembly algorithm is to search the Tables for an entry whose value equals that of the PC. If an entry is found, then the disassembly (display) mode is set to correspond to that recorded in the tables. If a synthetic label is called for, it is generated and placed in the label field; if a user assigned label is present, then it is used instead of the synthetic label.

Now let's get back to that suspected data area at address 0160 in the disassembly. There are two courses of action possible. a) We could use the MARK command to designate the first byte (at 160H) as BYTE. Or, b) we could use the BUILD command starting where we think instruction code starts after the data segment.

Using the DISPLAY command, we observe that there is now a synthetic label at 018C, and that it is an instruction. So, let's use the BUILD command to convert some more arguments into symbol table entries; to keep things under control we'll just BUILD over a small range of code that looks like valid instructions, then display the results:

```
        V# b 18c 19b ; dis 100 19b
```

The results so far are shown in the second listing in Appendix C. REVAS has detected (at addr 18C) a data reference to address 160, and assigned a synthetic label to the code at that address. The label is UT0160. The 'U' tells us that the label is unlocked so that subsequent references could change the type from 'T' to 'S' or 'W'. The subsequent code is now listed in byte format, in accord with the 'T' in the label.

Although the data area at 160 appears to be quite valid (it's a buffer with some ASCII already present, and terminated with the '$' that the 'print buffer' function of CP/M uses), let's use it to experiment with the other forms of the BUILD command. Try the following:

```
        V# ib 160 160; d 15d 18c
```

The effect is to change the label to US0160 and, in accord with the 'S', disassemble all the code up to the next label as instructions. If the range of the 'build' above had been from 160 to 18b, then all of the 'jrnz' instructions would have generated locked synthetic labels. Now try:

```
        V# wb 160 161; d 15d 18c
```

This changes the label to UW0160, and all the code up to the next label is now displayed in 'WORD' format. OOPS! The last line in the block of data is .BYTE format, and there is just one byte of code there. What happened? The word format starts at address 160 and continues through 018B; that's an odd number of bytes. Since each word in the data area requires 2 bytes, the one byte left at the end cannot be used to construct a word, and REVAS automatically switches to byte format display for that byte.

It is wise to be careful in specifying the range, because the WBUILD function will (by default) assign all words within its range as symbols of type 'US' (unlocked instruction). If the first word in the list had been within the program address limits, that would have happened here. Also observe that the address range included 2 bytes rather than only one as in the previous example. That's because it doesn't make sense to process one byte as if it were a (2 byte) word!

Now return to the original byte format display with:

```
        V# bb 160 160; d 15d 18c
```

There is a general principal to be observed about the conduct of a disassembly from these examples. If you simply use the BUILD command in the instruction mode to initially build tables (and thus assign labels) over a range of addresses that contain tables of data, the result will be a bunch of false labels. Even more of them would be generated if you used the WBUILD command indiscriminately. The best way to start a disassembly is to use the DISPLAY commands only to identify ranges of code that are instructions. Use the BUILD command to extract the argument information from the instruction areas as thoroughly as possible. After doing that, most of the data areas will have been identified (by synthetic labels starting with 'UT').

IBUILD is usually used to ensure instruction mode after one of the other modes has been used. WBUILD is used principally after you have decided that the code segment of interest really is addresses (typically a jump table). And BBUILD might be used when you want to recover from one of the other forms.

TARGET CODE ANALYSIS

The object of complete disassembly is to gain a thorough understanding of the function, logic flow, and data organization of the target program. Perhaps you wish to generate a source file for modification and subsequent reassembly. Or perhaps you wish to learn the techniques used, so you can add them to your own arsenal of programming expertise. For such purposes both heuristic and analytic approaches are required; REVAS supplies the tools for analysis and you supply the heuristic intelligence. You will use every clue you can get to piece together a complete analysis of the program.

We will continue to refer to XD.OBJ for examples. Rename it to XD.COM and run it, if you haven't already done so. Here is what we know about it now:

XD is a directory listing program that operates under the CP/M operating system. It is invoked with an optional drive designator argument, and you will observe that it is not necessary to include the usual colon after the drive name. The directory display produced is sorted and formatted on the console, and includes the length of each file and finally the number of kilobytes of space left on the disk. Such a statement of program function furnishes you with clues about the kinds of routines that must be present in the program.

During the initial disassembly and table building processes, additional clues became evident. The code has been organized into segments, each of which can be analysed to determine its function. The data areas have been identified, and those that contain ASCII give a further clue as to function. You will have already started the analysis by observing at address 0104 that the caller's (CP/M) stack pointer is being stored at a location with the symbolic label UT06BB and that a local stack is established starting at symbolic address UT06F9. As you decipher the intent of the code, you can use the AT command to insert short comments; then you won't have to decipher it again! Be careful, though; the comments do take up memory space! You could easily over-do it and use up available space.

REAL SYMBOLS & LABELS

We know without even trying that we will not be able to remember more than a half-dozen or so such labels and what they mean. The trick is to rename such symbols with a mnemonic that will be meaningful. So, we can use the EQUALS command to assign such a mnemonic:

```
        V# 6bb eq OLDSP; 6f9 eq STACK; d 100
```

This command string assigns two labels (OLDSP and STACK), then displays a screenfull of disassembled code starting at 0100. Notice that only the address portion of the synthetic label is used, and that it is not necessary to include the leading zero. REVAS will also recognize a synthetic label (such as S6bb or UT6f9) as one of its command arguments; however, only the address portion is actually stored in the symbol table or used for a search in the tables. You should avoid assigning REAL symbols that look like synthetic ones in order to avoid confusion.

Suppose you change your mind about the name 'STACK' and would like to change it to 'NEWSP'. Either of the following commands will work:

```
        V# 6f9 equal NEWSP
or      V# STACK e NEWSP
```

Notice, in the second example, the symbolic reference to the value(06f9H). THIS IS THE FIRST OBVIOUS EXAMPLE OF REVAS' ABILITY TO USE SYMBOLIC REFERENCES. It's a lot easier for us humans to reference a subroutine by its name than by its numeric address!

For illustrative purposes, the above examples have used several abbreviations of the 'equals' command. REVAS doesn't care which you use, as long as no ambiguity results. The HELP list of the commands shows the minimum unambiguous abbreviation of each command. Also note that the labels assigned in the examples are upper case. That's only for emphasis. If you wish to use lower case, do so. REVAS, however, converts all console input to upper case, so the labels you assign will appear that way in listings. LABELS WHICH YOU ASSIGN WITH THE EQUALS COMMAND ARE RECORDED AND RECOGNIZED LITERALLY. If you try to abbreviate a user assigned label, it will not be recognized.

Incidentally, REVAS won't let you assign a symbol which duplicates an existing one. If you try it, you will be reminded.

Mode information stored in the symbol table is not altered by user label assignment. In particular, the mode lock bit is unchanged. Assume that disassembly has shown a data area at hex address 160, and there is a synthetic label (UT0160) resulting from a previous BUILD command execution. The leading 'U' implies that the mode is not permanently defined, and can still be affected by subsequent BUILD commands whose address range includes the label. If you want the data type (mode) to be permanent, then use the LOCK command. For example, to prevent inadvertent changes in the buffer at 0160, you could use:

```
        V# lock 160 byte
```

Or, if you had assigned the label 'BUFFER' to that address, you could use the alternate form:

```
        V# lock buffer byte
```

REMOVING SYMBOLS & LABELS

Suppose that, during the initial table building process, some erroneous synthetic labels were generated. If they were generated as a result of code that looked like branch-type instructions, then their modes will be locked. How do you get rid of them? The answer is to use the KILL command, which completely removes all reference to its argument from the symbol tables. If you have assigned a label to an address, the KILL command can use that label as its argument. Using as an example the same address as above, either of the following will work:

```
        V# kill 160
or      V# kill buffer
```

On a subsequent display listing, the code at 160 will now be interpreted as instructions because the most recently encountered label (at 15d) is flagged as instruction mode. Address 160 and its assigned symbol have been "forgotten".

The EQUALS command and the MARK command can be used even when no synthetic label has been generated. For example, it's convenient to have a label on the first executable statement of the program. So far, there has been none generated there, so let's put a label there and mark it as instruction mode:

```
        V# 100 eq xd;mark xd instr
or      V# mark 100 inst; 100 eq xd
```

The first option clearly creates a table entry which associates the label 'XD' with address 0100 and then marks it as instruction mode. Not so obvious is the fact that the mode is also locked against inadvertant change. The second option illustrates a subtle characteristic of the mark command. At the time it is invoked here, there is no symbol table entry for address 0100; there is nothing to mark! In such a case MARK creates a table entry, marks it as directed, locks the mode, but does not record the presence of a symbol of any kind. The subsequent EQUALS command then finds this table entry and enters the symbol without disturbing the mode information.

CONTROL ENTRIES

Earlier, when discussing disassembly of the buffer data area at address 0160, it was pointed out that the MARK command could be used. (MARK produces a control record in REVAS' tables.) If the command:

```
V# mark 160 byte
```

had been issued at that time, then subsequent display would have shown the code from 0160 through 018b in byte format and any attempt to change that display with the DISPLAY or BUILD commands would fail, EVEN THOUGH NO SYNTHETIC SYMBOL IS DISPLAYED. However, once the BUILD command encounters a reference to address 0160 (as at address 18C), the generation of a synthetic label (T0160:) will be enabled with the data mode locked in byte format. There are cases of data areas for which the first byte of the area is never referenced; a push-down stack is an example of such an area. Once you have found such an area, the only way to get it to disassemble properly is to MARK the first byte appropriately.

Control entries can be modified with the MARK, LOCK, and UNLOCK commands; can be removed with the KILL command; and can be displayed with the SHOW INDEX command.

CROSS REFERENCING

During analysis of the disassembled code, a logical approach is to functionally identify as many routines and data storage locations as possible by assigning meaningful labels. As these labels appear in operand fields during subsequent displays, they help in figuring out what the more complicated routines are doing. You can use the FIND command to locate all the places a particular routine or memory location is referenced. Or, once symbol tables have been built, you can use the XREF command to produce a similar listing for a group of such references.

At this point, you should be able to complete the disassembly of XD. It will be a good way to gain facility with the REVAS command set usage, and you may wish ultimately to modify the .LST FILE and reassemble it to create your own customized version. As you will see during analysis of the program, it was written for version 1.4 of CP/M and assumes an 8" single density disk, so it is likely that you would want to change it if your system differs.

SAVING THE SYMBOL TABLES

Seldom is a disassembly completed at one sitting. Even if it were complete, you want the option of saving the symbol tables that you and REVAS have generated for the disassembly so you won't have to repeat it next time you want to examine the target code. The SAVE command saves the symbol tables in a file whose name is shown in the STATUS display after 'TBL::'. If you SAVE the tables periodically during a disassembly/analysis session, then you will not lose what you have done so far when the 'inevitable' system crash occurs.

There are other commands and parameters that have not been discussed in this section. Also, many of the commands discussed here possess additional capabilities. With the background presented here, you should have little difficulty in understanding their capabilities and use from the descriptions given in the following sections. I strongly recommend that you experiment with each command while you are disassembling the XD program (or some other that you are interested in).

COMMENTS

Comments up to about 62 characters long may be inserted in the disassembly with the AT command. The comment will appear just before the address with which it is associated. The AT command will replace an existing comment with a new one if the location specified is the same. If the AT command is given without a comment string, then any existing comment will be removed.

So, you can insert, replace, and remove comment lines. This facility is sufficient to document functions for which a six character label would be too cryptic. Longer elaborations will have to be inserted in the .LST file later with your system editor.

COMMAND DESCRIPTIONS

The following chapters contain the formal descriptions of the commands, parameters, and reserved words used by REVAS.

A command may follow one of REVAS' prompts (V# or M#), or it may be part of a command string. For this reason, no indication of a prompt or command delimiters is given in the explanatory examples which follow. From the introductory sections, you will have recognized that a command may only be entered after a prompt or as part of a command string or macro.

CHAPTER 5

UTILITY COMMANDS

INSTANT COMMANDS

A few single-letter commands are recognized at any time except during entry of a command string. They permit you to suspend and restart output activity, and to abort execution at any time.

^S!s!S Stops current display class activity. Activity resumes following any other keyboard input except instant commands.

^E!e!E Aborts the current command execution and continues with the next (if any) command.

^X!x!X Aborts the current command, ignores the remainder of the command line, and prompts for another command. (If the command line only contains one command, then 'E' and 'X' are equivalent.)

Note: These commands use direct input, bypassing the system input editing procedures (^C has no effect).

CALC <arg1> <arg2>

<arg1> and <arg2> are added, subtracted, multiplied, and divided using integer arithmetic. The arguments may be numbers, symbols which have already been defined, or reserved words.

Addition and subtraction are performed modulo 0FFFF(HEX); if a sum is greater than the modulus, then the quantity displayed is equivalent to the 16 least significant bits. If the result of a subtraction is less than 0 (i.e., negative), then the quantity displayed is equivalent to the 1's complement of the result.

Note that the product is displayed as a 32 bit result (4 bytes), while the other results are displayed as 16 bit quantities. Also displayed is the remainder from the division; this quantity is identified as "X MOD Y".

The arguments are entered in the currently defined input radix (see the ARGRAD parameter); the results are displayed in the currently defined output radix (see the OUTRAD parameter). Thus, in addition to the arithmetic functions, this command can be used to perform radix conversions.

CALL <address>

Transfers execution control to the subroutine starting at <address>. <address> may be a number, symbol, or reserved word, as long as it has a value which is a real memory address at which executable machine code starts. If the called routine ends with a 'RET', then REVAS will continue execution if the called routine has not altered the code in REVAS or its tables. Naturally, the routine must preserve the CPU stack pointer and stack contents in order to return to REVAS. During execution of such external code, you will have to establish a separate stack if the routines use more than 20 words of stack space; that's the amount available on REVAS' stack after a CALL command.

ERASE <macro name>

Macros occupy space in the symbol table area of memory. The ERASE command deletes the macro and returns the space to REVAS for other use, such as storing symbols. This command works only on macros; deletion of other types of user assigned names is accomplished with the KILL or EQUALS commands. <macro name> must be spelled out in full; abbreviations are not recognized.

[n] HELP [<option>]

This command produces one of three listings on the active output device(s). Each list contains brief reminders of the functions of the key words in the list. Some of the lists are long enough to scroll off the screen. To prevent the consequent loss of information, <n> lines are displayed and then REVAS pauses for permission to continue. Instant commands may be used during the pause to abort the listing if you wish. When REVAS is first invoked, n has a default value which is suitable for most terminals; n, if present, redefines the default value. The parameter HLINES is identical to <n> specified in the HELP command; thus, an alternate method of specifying the number of lines to display is to use the SET command (see the section on parameters).

If <option> is present, it defines which of the HELP lists is to be displayed. If <option> is not present, HELP explains the options. The recognized options are COMMANDS, PARAMETERS, and RESERVED. Abbreviations can be used for the command word and for the options. For example, the command string "H C;H P;H R" will cause the display of all three listings.

MACRO <macro name>; <command string> CR

This is the command by which Command Macros are defined. The macro name may be up to 6 characters long, and must not start with a digit or ")". The semicolon command separator must follow the macro name to define the start of the macro command string. The command string may contain as many commands (delimited by semicolons) as required to define the function you want. A carriage return terminates the macro definition. In practice, the length of a Macro is limited to 60 characters by the size of the input buffer. Macros may be nested as deeply as you wish; that is, the command string may include macro calls. Recursive use of macros is not supported; i.e, the command string may not contain a macro call to its defining name at any level of nesting.

A macro is invoked (called) by including its name in a console generated command string or in the command string of another macro. Appendix D contains an example of a macro and its use.

MEMORY and VIRTUAL

REVAS is able to disassemble code either directly from a disk file or from program code that is resident in memory (but does not overlay REVAS or its table space). The type of access currently active is indicated by the prompt issued at the start of each command line: M# or V#. You may switch from one type of access to the other by issuing one of these commands.

EXAMPLE (including prompt):

```
        M#VIR
        V#       (REVAS issues new prompt)
```

During virtual program access, target code is transferred from disk to a buffer as required for the disassembly process. Logical addresses for the PC are calculated continuously based on the value assigned to PS (program start or origin). The default value of PS is 100H, consistent with most CP/M compatible programs that are designed to run at 100H. REVAS will not attempt to disassemble outside the range of PS to PE, since that would involve access to completely undefined disk areas.

REINIT

Reinitializes all file control blocks, symbol tables, and macro definitions. Any title for the list device page heading is erased. Does not flush buffers or close files. Used to abort a disassembly and start over.

QUIT (no arguments)

Flushes the .LST buffer and closes the .LST file if necessary, gives you an opportunity (when it makes sense) to save symbol tables, then returns control to the CP/M Console Command Processor (CCP). Within REVAS, control-C as the first character of the command line has the same effect as the QUIT command. If exit from REVAS is via cold boot or system reset, then no buffer flush or file closure takes place.

SHOW <name> [<start> [<end>]]

The SHOW command displays information from REVAS' tables. <name> has four possible symbolic values: Comment, Index, Macro, and Symbol. If the <start> and <end arguments are absent, then data from the entire table referenced by <name> will be displayed. If <name> is MACRO or SYMBOL, then the <start> and <end> arguments are ignored, and the entire table is displayed. For the COMMENT and INDEX forms of the command, <start> represents the value (usually an address) of the first item to be displayed, and <end> represents the value of the last item to be displayed. If <end> is missing, then only the item represented by <start> is displayed. <name> may be abbreviated to as little as one letter, and the <start> and <end> arguments may be any numeric or symbolic value.

SHOW COMMENT Lists the comments stored in the tables, one per line, along with their locations in the disassembly. The list is sorted by location (address).

EXAMPLES:

SH C 43A   might produce the following display:

```
043A ;THIS IS A COMMENT AT 43A IN THE DISASSEMBLY
```

SH COM 43A PE

Produces a list of all comments from address 43A to the end of the disassembly

SHOW INDEX tabulates entries for which synthetic symbols or disassembly controls will be generated during disassembly. This is the only command which displays the control entries produced by the MARK command. Such entries are distinguished by a leading '*' on the synthetic symbol in this display. A typical row from the table might look like this:

```
XS0000   S0100   UT0132  *W0156  *US0211  S026E  T0271   ..., etc.
```

The 'X' prefix on the first symbol indicates that this is an external symbol (outside the limits set by PS and PE). The data types specified at 0000, 0100, 026E, and 0284 are locked instruction; the data type at 0132 is unlocked byte and that at 0271 is locked byte. Control records are present for adresses 0156 and 0211 which specify locked word and unlocked instruction data types. Note that the display is sorted on the address value.

SHOW MACROS Lists currently defined Macros, one per display line. The list will be in alphabetic order, sorted on the MACRO name. Each macro is enclosed in brackets for display purposes. The brackets are not part of the macro.

SHOW SYMBOLS Lists user assigned symbols and their values in tabular form. The table is arranged with the symbols in alphabetic order. The value is displayed as a synthetic symbol so that you can see the extant MODE and MODE LOCK assignments. If OUTPUT is a locked symbol which you have assigned to the subroutine at address 1A43, then it will be shown in the table as follows:

```
 ...(other symbols).... OUTPUT =S1A43 .............
```

SET <parameter name> <new value>
or TURN <parameter name> <new value>

These two commands are completely equivalent and interchangeable. The only reason for the two command words is to permit command entry in the form that seems most natural for each parameter.

EXAMPLES:

set pe 0a110	tells REVAS where pgm ends
se pr 1	turns on list device
set pr on	turns on list device
se con 0	turns off console
turn con off	turns off console

Most of the functions within REVAS are controlled by parameters stored within the program. Many can be changed at will by using the SET (or TURN) command. For a list of parameters, type HELP PARAMETERS (or an abbreviation of it such as H P).

STATUS [<parameter name> ! ALL]

The STATUS command without arguments produces a display of some of the current parameter values, file assignments, and the start of unused memory space. The abbreviations PC, PS, PE, DS, and DE in the Status display stand for Program Counter, Program Start, Program End, Display Start, and Display End, respectively. They are discussed below in the Parameters section. This form of the STATUS command is automatically executed during initial start-up of REVAS

The STATUS command with an argument reports the current values of parameters. If a parameter name is given as the argument, then that parameter name and its value is displayed. If the word ALL is the argument, then all of the parameters are listed with their values. PE and ORG are always displayed in Hex; the switches, ECHO through TOP, are displayed with ON/OFF values; and the remaining parameters are ALWAYS displayed with DECIMAL values.

As in other command strings, unambiguous abbreviations are recognized.

TITLE <string>

When output is sent to the printer and pagination has been requested by turning ON the PAGER switch parameter, a 5-line space is left at the top of each page. The second line in this vertical tabulation can be used for a title which will appear on each subsequent page. The TITLE command is used to define the content of that page heading line. <string> may contain any characters except semicolon or carriage return. Either of the latter will terminate the string. <string> starts with the first non-blank character; after that one, blanks (spaces) may be included in the string. The maximum length of the title string is 60 characters. If there is no string specified (first non-blank character is ';' or carriage return), then any previously entered title is erased.

TURN

This command is identical to SET. See SET/TURN, on the preceding page.

VIRTUAL

This command directs REVAS to access code from a disk file instead of directly from memory space addresses. See the description under MEMORY/VIRTUAL on page 5-5.

CHAPTER 6

DISPLAY CLASS COMMANDS

<cmd word>=	DISPLAY ! PRINT ! BUILD
format:	[n] [<prefix>]<cmd word> [arg1] [arg2]

n, if present, is a symbol or number which defines the default number of lines of output to be processed. If n=0, the process is repeated for an unlimited number of lines. The default value of n (before it is specified in a command) is 15. n is 'remembered' until it is respecified, so it need not be included in each new command string. If n is input as a number, then its radix is determined by the 'RPTRAD' parameter. The default radix is decimal.

<prefix>, if present, is one of the letters I,B, or W. The effect of the prefix is to establish the default mode of interpretation of subsequently disassembled code as Instruction, Byte, or Word, respectively. The default mode is shared by all three DISPLAY class commands; once specified for DISPLAY, for example, PRINT and BUILD will also use the same default.

arg1, if present, defines the starting address in the target program for the appropriate action by setting the program counter (PC) to the value of arg1. If arg1 is absent or blank, then the current value of the PC is used.

arg2, if present, defines the last address to be processed. The range defined by (arg2 - arg1) temporarily over-rides the default number of lines specified by n. If arg2 is absent or blank, then the current value of PE is used.

arg1 and arg2, if present as numbers, are interpreted in the current input radix. The input radix for arguments is specified by the ARGRAD parameter, whose default setting is HEX (16 decimal).

DISPLAY, IDISPLAY, BDISPLAY, WDISPLAY

Disassembles and displays code in the form (or mode) specified by one of the prefixes I,B, or W.

DISPLAY (no prefix) uses the most recently specified (default) prefix. When REVAS is first started, the default value of the prefix is 'I', so code is interpreted and DISPLAYed as instructions.

If symbol tables are active, then the display MODE is controlled by table entries. Each time a label is encountered, its recorded MODE is noted and the display changes automatically to the new mode. A flow chart of this process is shown on the next page.

EXAMPLES:

DIS	displays disassembled code, starting at the current PC. The number of lines is determined by the current default value of n
21 D 12FF	displays 21 lines starting at 12FFH and sets the default value of n to 21 (decimal)
D PS	displays the default number of lines, starting at the logical program origin. For CP/M transient programs, PS normally has a value of 100H.
BD	Displays n lines of BYTE data, 4 bytes per line, starting at the current PC.
WD ,,PE	Displays data in WORD format, 4 words per line, starting at PC and continuing to the end of the program as indicated by PE.
0 ID	Displays disassembled code as Instructions until interrupted by an Instant Command or the end of the program (PE)

```
           (DISPLAY or PRINT command)
         (Entry from Command Processor)
                        :
                        :
->--------------------->:
^                       ↓
^                    DONE YET?
^                       :
^                       *----YES--------->.
^                       :                 ↓
^                       NO            RETURN TO
^                       :          COMMAND PROCESSOR
^                       :
^                       ↓
^                 SYMBOL PRESENT?
^                       :
^                       *----YES--------->.
^                       :                 ↓
^                       NO            COPY MODE FROM
^                       :             SYMBOL RECORD
^                       :                 :
^                       :<---------------<⊥
^                       :
^                       ↓
^               PROCESS CODE FOR ONE
^               DISPLAY LINE IN THE
^                   CURRENT MODE
^                       :
^                       ↓
^               SEND THE LINE TO THE
^               ACTIVE OUTPUT DEVICES
^                       :
^                       ↓
^                    UPDATE THE
^                 PROGRAM COUNTER
^                       :
^                       ↓
^-<--------------------<-⊥
```

FLOW CHART FOR DISPLAY AND PRINT COMMANDS

PRINT, IPRINT, BPRINT, WPRINT

Same as the DISPLAY command, except that output is also sent to the list device. For examples, see DISPLAY command, Page 6-2.

Some elementary page formatting is available for output sent to the list device (the Printer) if the PAGER switch is turned ON. In this mode of output, 5 lines are reserved for a header at the top of the output page. The second line of the header area may contain a brief title of your choice. You may designate such a title by using the TITLE command. (See the description of this command in the UTILITY COMMANDS chapter.) The number of text lines to be printed per page and the number of blank lines to leave as a bottom margin are parameters which you may interactively specify with the SET command. (See PLINES and BOTMAR in the Parameter Description chapter). Total page length is, of course, the sum of PLINES, BOTMAR, and the 5 header lines. If you choose to change these from the keyboard, you will probably find it convenient to SET ARGRAD DECIMAL first. Don't forget to change ARGRAD back to HEX if you're used to using hex addresses!

If you are using a printer which allows single page feed (like a typewriter) then you will want the disassembly output to pause while you insert a fresh page in the machine. Turning the PAUSE switch ON will cause that to happen. The PAUSE switch is described in the PARAMETER DESCRIPTIONS chapter.

Finally, you must have some way of telling the REVAS routines that you have just set the printer to top-of-page. That is accomplished by SETting the TOP parameter. The only time it is necessary to SET TOP is after you have manually readjusted the position of the paper in your printer and want the new position to be considered top-of-page. It is not necessary to SET TOP when inserting a new single page during a PAUSE for that purpose, since REVAS already knows it is at top-of-page.

Although all of the parameters may be set and the TITLE command may be executed at any time, none of them will take effect until the PAGER switch is turned ON. Also note that these parameters are physically located in the block of values at the start of REVAS where you may, if you wish, change their default values to those which are most generally useful in your system. (See the CHANGING REVAS section.)

BUILD, IBUILD, BBUILD, WBUILD

The main function of the BUILD command is to create symbols and store them in the symbol tables. The BUILD function also generates and stores disassembly control information with each symbol. The control information comprises the data type (I,B,W) associated with the symbol and a flag (the mode-lock) that tells whether the data type can be changed again without specific direction from you.

IBUILD (or BUILD if the default MODE is already 'I') is used to process the code in address ranges that you believe to be filled with instruction type code. For most disassemblies, this form will produce the majority of synthetic labels. Because of ambiguities in code interpretation, some of the symbols produced will have an incorrect data type assignment. (You'll see them in subsequent DISPLAY of the disassembly.)

BBUILD does not create symbol table entries. It's main use is to change all labels within its argument range to type BYTE (if permitted by the mode-lock flag).

WBUILD creates a table entry for each 16 bit word in its argument range. It's main use is to process data segments which you have decided contain address-type data.

BUILD operates by algorithmic creation and modification of address records in the symbol tables. Code is disassembled line-by-line, as if it were to be displayed. Instead of sending the code to the console buffer, however, it is searched for 16 bit arguments. Each argument detected is used to create a new symbol table record or update an existing one.

The interpretation of a line of code (as Instruction, Byte, or Word) is controlled by the current content of the MODE parameter. If a symbol table record exists for the address of the current line of code, that record also contains a mode specification. (Such a record also indicates that a label would occur in the label field for this line.) When a table entry exists REVAS has a decision to make: does it interpret the line according to the MODE parameter, or according to the mode specified in the symbol table record? The decision depends on the status of the mode-lock ('U' or blank) in the symbol record. If the symbol record mode is Unlocked, then it is changed to agree with MODE; if the symbol record mode is locked, then MODE is changed to agree with the mode specified in the symbol record.

The algorithm used for BUILD is illustrated by the flow chart on page 6-7; the actions taken during processing a line of code are summarized in the table on page 6-8.

Those 16 bit arguments that fall outside the range of the target program (as defined by PS and PE) are treated as constants, and are not automatically entered in the symbol tables.

If the range of addresses over which the BUILD command is executed is very large, significant time elapses before completion of the process. During that time, there is no disassembly output on the console. In order to let you know that REVAS is still alive and well (and busy), a series of slashes is sent to the console at a rate of about one for every 60 lines of disassembly processed. Actually, this mechanism comes into play any time the console is turned OFF, as it is during BUILD.

Each address record in the symbol tables contains a flag which specifies whether the data type (Instruction, byte, or word) associated with that symbol may be changed automatically. This flag bit is called the mode-lock. When the mode is locked by setting the flag the data type cannot be changed by the BUILD command. When the data type is unlocked, it may be changed during execution of a BUILD command. The only time that REVAS automatically sets the mode lock is during the creation of symbol table entries for the arguments of branch-type instructions such as 'call' or 'jmp' That's because such arguments almost invariably address instruction type code, and the instruction data type assigned should not be changed by inadvertent use of the BUILD command. Once you have identified a segment of disassembly as INSTRuction, BYTE, or WORD, you may use the MARK command (refer to its description in section 7) to lock the mode against inadvertent changes.

```
                   (BUILD Command)
              (Entry from Command Processor)
                           :
                           :
->--------------------->:
^                          v
^                      DONE YET?
^                          :
^                          *----YES---------->.
^                          :                  v
^                          NO            RETURN TO
^                          :          COMMAND PROCESSOR
^                          :
^                          v
^                    SYMBOL PRESENT?
^                          :
^                          *---->NO--------------->:
^                         YES                      :
^                          v                       :
^                     SYMBOL MODE                  :
^                       LOCKED?                    :
^                          v                       :
^        NO----------------*-------------YES       :
^        v                                v        v
^ COPY DEFAULT MODE                   COPY MODE    :
^ TO SYMBOL RECORD                       FROM      :
^   AND TO MODE                     SYMBOL RECORD  :
^        v                                v        v
^        '->------------>:<---------------------<-'
^                          :
^                          v
^                DISASSEMBLE CODE FOR ONE
^                  DISPLAY LINE IN THE
^                      CURRENT MODE
^                          :
^                          v
^                    MAINTAIN TABLES
^                      AS REQUIRED
^                          :
^                          v
^                       UPDATE THE
^                    PROGRAM COUNTER
^                          :
^                          v
^-<------------------<-'
```

FLOW CHART FOR THE BUILD COMMAND

THE BUILD ALGORITHM

MODE:	Action:
INSTR	a) The data type of any Label encountered is changed to INSTR if its data type is not locked. b) 16 bit arguments are entered in the tables with DATA TYPE determined by the nature of the instruction. If the argument already exists in the tables, the entry is checked to see if its recorded mode is locked; if not, then the mode is set to INSTR data type for arguments of a CALL or JMP type instruction, and to BYTE data type for all others. If the argument belongs to a CALL or JMP type instruction, then the data type is locked to prevent future change.
BYTE	a) The data type of any Label encountered is changed to BYTE if its data type is not locked. b) No table entries are generated
WORD	a) The data type of any Label encountered is changed to WORD if its data type is not locked. b) Each 16 bit word of data which follows is considered to be a valid address if it is within the program range defined by parameters PS and PE. A symbol table entry is made for each such word and a data type (mode) is assigned to the entry. The data type assigned is the value of the AMODE (Argument MODE) parameter, whose default value is Ø(INSTR). This default value can be changed,of course, with the parameter setting command.

CHAPTER 7

DISASSEMBLY COMMANDS

AT <arg> [<string>]

The AT command provides the means for adding comments to the disassembly and for removing comments. <arg> may be an address, a symbol, or a reserved word; it is the location at which the comment will be inserted or removed. <string> is the comment you wish inserted. If <string> is absent, then any comment already present at location <arg> will be removed. If both arguments are missing, then nothing is done.

<string> may be up to 62 characters long and may contain any printable characters except ';'. The argument string is terminated by ';' or a carriage return. It will also be terminated automatically if the console input buffer becomes full.

<arg> EQUALS <symbol>

The EQUALS command is used to assign, replace, and erase user assigned symbols. <arg> may be an address or a label, and its value may be anything from 0000 to 0ffffH. <symbol> is a string of up to six printable characters. (See <string> in the Formal Command Syntax in Appendix A.) The string should not start with a digit and may not contain spaces, commas, or semicolons.

<addr> EQUALS <symbol>	assigns a new symbol
<old symbol> EQ <new symbol>	replaces a symbol with another
<addr> EQ	erases any symbol at <addr>
EQU <symbol>	erases the symbol

EXAMPLES:

```
        100 EQ START
```

enters the symbol START in the symbol tables so that "START" will appear in the label field of the instruction at location 100(HEX) and in the argument field of any instruction that references location 100H. If the same symbol is already in use, REVAS will notify you and refuse to enter the duplicate symbol.

```
        START EQUALS BEGIN
```

replaces the label START with BEGIN.

If either one of the arguments of the EQUALS command is missing, then any label associated with the argument given is erased; in subsequent disassembly a synthetic label will be displayed. A symbol table entry still exists; only the user assigned symbol is erased. Note the difference between this use of the EQUALS command, and the KILL command.

```
EXAMPLES:       100 EQU
                EQUALS BEGIN
```

Assuming that, as above, the label "BEGIN" has been assigned to the instruction at location 0100H; either of these commands would erase the label "BEGIN" from memory, leaving only a synthetic label with its MODE information. The KILL command, on the other hand, removes both the real and synthetic labels, leaving only a HEX constant.

If both arguments are missing, then the command is ignored.

[n] FIND <srch addr> [<start>] [<end of srch>]

This command searches a range of target program code for those instructions that reference the <srch addr> argument as a 16 bit quantity. Each line containing a 16 bit reference to the <srch addr> will be output to the currently assigned output device(s).

The range searched is determined the same way as in the DISPLAY class commands; if <start> is not given, the current PS is used as the place to start searching. If <end of srch> is not given, then the last specified value of <n> is used to define the range. If n=0 and <end of srch> is not given, then the search will end at the current value of PE (logical program end).

Examples:
('STRING', 'BEGIN', and 'END' have been assigned as labels with the EQUALS command)

```
        FIND 100 140 92F
```

displays all lines in which 100H is an argument, within the address range of 140H to 92fH.

```
        0 FIND 100,140
```

displays all lines in which 100H is an argument, within the address range of 140H to the end of the program.

```
        FIND STRING, BEGIN, END
```

displays all lines in which 'STRING' is an argument, within the range starting with the label 'BEGIN' and ending with the label 'END'.

```
        T printer on; fi string, begin, end;t pr off
```

Same as the previous example, except that output also goes to your system printer.

KILL <arg1> [-<arg2>]

Removes all reference to addresses within the range of <arg1> to <arg2> from the symbol tables. In subsequent disassembly listings, these addresses will be shown as constant(s). Reassignment as a symbol requires a BUILD or EQUALS command.

If <arg2> or its minus sign prefix is absent, then only one record (the one that represents <arg1>) will be deleted. If both arguments are missing, then nothing is done.

LOCK [<arg1> [-<arg2>]]
or
UNLOCK [<arg1> [-<arg2>]]

The effect of these commands is to prohibit (LOCK) or permit (UNLOCK) automatic data type assignment during execution of the BUILD command. They operate on symbol table entries within the address range represented by <arg1> and <arg2>.

<arg2>, if present, must be prefixed by a minus sign as shown. If <arg2> is not present, then only the symbol address <arg1> is affected. The command is ignored if no arguments are present.

These commands do not create symbol table entries; they operate only on the control bits of existing table entries.

MARK [<arg1> [-<arg2>] [<mode>]]

This command establishes the data type associated with addresses in the range specified by <arg1> and <arg2>. If no table entry exists for <arg1>, then one is created; entries are NOT created for other addresses in the range. Beyond the first argument, the effects of MARK occur only for already existing table entries. <arg2>, if present, MUST be prefixed with a minus sign as shown. If you forget to include the minus sign, <arg2> will be interpreted as if it were a <mode> argument.

As with other commands, the arguments may be numbers, reserved words, or defined labels. In particular, the reserved words INSTR, BYTE, and WORD are useful with the MARK command; and have the values 0, 1, and 2, respectively.

If the <mode> argument value is 0, 1, or 2, then the data type stored in each symbol table entry (the mode assignment) within range is set accordingly (i.e., to INSTRuction, BYTE, or WORD data type) and the mode is locked.

If the <mode> argument is blank or if its value is greater than two then a default value of 0 (INSTruction) is used.

Although a symbol table entry may be created, no label is generated; labels are generated only as a result of other commands such as BUILD or EQUALS.

EXAMPLES:

```
    MARK OUTSUB INSTR
```

Establishes the data starting at label OUTSUB as instructions and locks the assignment against automatic change during subsequent BUILD command operation.

```
    MARK CONDTA -PE BYTE
```

All labels present in the range from CONDTA to the end of the program carry the BYTE attribute and associated code will display in the BYTE (or DB) format.

<u>XREF <arg1> <arg2> <arg3> <arg4></u>

The cross-reference command repeatedly searches the address range defined by arg3 to arg4 for instructions that reference a label within the range of values defined by arg1 to arg2. Each such instruction found is printed out on the currently active output device(s). The instructions that reference the smallest value label in the label range are printed first, followed by groups of instructions which reference successively higher label values. This procedure is actually a repeated execution of the FIND command, in which the <srch addr> is obtained by a sequential search of the symbol tables. If the symbol tables contain no entries (i.e., no BUILD or symbol-generating command has been executed), then XREF will fail. XREF only works after synthetic or real symbols have been generated. The arguments all have default values which will be used if the argument is left blank (two successive commas):

arg1:	current value of DS	(Display Range Start)
arg2:	current value of DE	(Display Range End)
arg3:	current value of PS	(Pgm Start address)
arg4:	current value of PE	(Pgm End address)

EXAMPLES:

```
XREF 100 200
```

Produces a listing of all the references to labels in the range of 100 to 200. References come from the entire program defined by PS and PE

```
turn console off;turn lst on;xref PS PE;turn lst off;close
```

produces a cross reference listing for the entire program with output going only to the file shown in the STATUS display after 'LST:'. This will take a while and will produce a large file, so be prepared with lots of disk space and time!

```
T printer on;XREF sub1 sub2;T pr off
```

Produces a cross reference listing for code in the range of sub1 to sub2, with output going to console and your system printer. By default, the entire program is scanned for references to labels in the code range.

CHAPTER 8

DISK FILE ACCESS COMMANDS

REVAS can access three files associated with the disassembly process. The following four commands (FILES, PGMFILE, TBLFILE, and LSTFILE) provide the means of telling REVAS what file names you wish to specify. REVAS then fills in appropriate file control blocks, accesses files as required, and sets up internal parameters to work with those files.

The three types of file are referred to here as the PGM file, the TBL file, and the LST file.

The PGM file contains the object code which you wish to disassemble. REVAS reads from the PGM file, but never writes to it.

The TBL file contains the symbol tables for the disassembly; the tables are written into the file when you invoke the SAVE command. This file, if present, is loaded during a FILES or TBLFILE command; it will also be loaded during the implied FILES command at the time REVAS is invoked if the <filename> argument list is present.

The LST file contains the Assembly format output which you create with the WRITE command. The LST file also has a second use which will conflict with its primary function as an assembly format output file: if the LSTFILE switch is turned on, then all REVAS output is copied into that file forming a record of the current session. (See the LSTFILE parameter description.)

File names must be unambiguous, and follow the conventions of the CP/M operating system. You may specify a disk drive in the usual way. Default entries are present in the file control blocks. If you omit the drive specification, then the current default disk is assumed. If you omit the filename or the extension, then the default values are used. If only the extension is specified in the filename argument, you must include the "." that separates the filename and extension just as you do for other system utilities. The STATUS command reports the current assignment of the three file control blocks. If none of the four commands has been executed, then the STATUS command will display the default file assignments.

The files which REVAS is to use may be specified on the invoking command line (at the CP/M prompt level) by appending the file name argument list (see FILES command below for sample argument lists)

FILES [<FN1>] [<FN2>] [<FN3>]

The FILES command takes from one to three CP/M filenames as arguments. Its simplest form (FILES <FN>) is usually all that is needed for a disassembly. The TBL and LST files produced will have the same name as the PGM file; only the extensions will be different. The following table shows what assignments will be made for the possible permutations of the argument list. Note that any one of the arguments may be no more than a file name extension you wish to specify; it must be of the form: '.<EXT>' in order to be distinguished from a name. An argument of the form 'B:' is also acceptable and properly specifies or changes the drive assignment for the appropriate file name. If no arguments are given, file assignments are displayed on active output devices.

ARGUMENT LIST:			ASSIGNMENTS:			ACTION:
			PGM	TBL	LST	
-	-	-	-	-	-	4
FN1	-	-	FN1	FN1	FN1	1,2,3
FN1	FN2	-	FN1	FN2	FN2	1,2,3
FN1	FN2	FN3	FN1	FN2	FN3	1,2,3
FN1	( )	FN3	FN1	FN1	FN3	1,2,3
( )	FN2	-	*	FN2	FN2	2
( )	FN2	FN3	*	FN2	FN3	2
( )	( )	FN3	*	*	FN3	none

Notes:
- - = missing terminal argument(s)
- ( ) = empty argument (n+1 commas for n arguments)
- * = uses default or most recent assignment

Actions:
1. Initialize tables to null.
2. Load TBL file if present on disk (see TBLFILE)
3. Access FN1 (see PGMFILE)
4. Display file assignments

Examples:

```
FILES B:FN1                (last 2 args missing)

FILES B:FN1, ,B:NF3        (2nd arg empty)

FILES ,,A:FN2              (1st arg empty, last missing)

FILES , , A:FN2            (1st arg empty, last missing)

FILES ,,A:FN2,FN3          (1st arg empty)

FILES,,,B:FN3              (1st two args empty)

FILES , , ,B:FN3           (1st two args empty)
```

PGMFILE <CP/M filename>

Open the PGM file and access it if present. Switch to Virtual access mode if not already there. This command does not affect assignments or access to TBL or LST files.

Example:

```
        PGMFILE B:XD.OBJ
```

TBLFILE <FN>

Establishes <FN> as the active TBL file, without affecting PGM or LST file assignments. The disk is searched for <FN>. If found, <FN> is loaded into the symbol table area and data from the file header is used to update PS, PE, PC, and the memory/virtual access mode. If the file is not found, REVAS waits until a SAVE command is executed to create the file and save the current symbol tables in it.

Example:

```
        TBLFILE FN2     (creates FCB for FN2 and current drive)
```

LSTFILE <FN>

Establishes <FN> as the active LST file, without affecting PGM or TBL file assignments. The file is not opened until REVAS attempts to write to the LST file. There are two ways to write to the file: by TURNing LSTFILE ON, and via the WRITE command (see next page). The first method sends everything that would display on the console to the LST file without editing, while the second sends only a disassembly in assembler input format to the file and then automatically closes the file. (See LSTFILE parameter on Page 9-3, and the CLOSE command below.)

Example:

```
        LSTFILE B:FN3     (creates FCB for FN3 using drive B:)
```

This command is exactly equivalent to the following form of the FILES command:

```
        FILES,,,B:FN3             (1st two args empty)
```

The reason for having the two equivalent forms is that the FILES command is implicitly executed when REVAS is first invoked by CPM; the assembly format output file may be specified at that time by:

```
        A>REVAS ,,,B:FN3
```

After REVAS has been invoked, the more convenient method of specifying the assembly format output file is by use of the LSTFILE command.

WRITE [<arg1>] [<arg2>]

Writes the label, operator, and operand fields of the disassembly to the .LST file shown in the STATUS display. The same data is displayed on the console and the list device if they are turned on. For .BYTE format portions of the disassembly, the auto-comment field is included. The disassembly range is specified only by the two arguments which follow the command word ([n] is ignored). If arg1 is empty, PS is used; if arg2 is empty, PE is used in its place. The file is automatically opened, written, terminated with the ".END" or "END" pseudo-op and closed during the execution of this command. Any existing file with the same name is over-written and lost.

SAVE

Automatically opens the .TBL file, writes the symbol tables into it, and closes the file. Any arguments are ignored. Any extant file with the same name is over-written.

CLOSE

The text in the LST file buffer is terminated with the End-Of-File character, then the buffer is flushed to disk and the file closed. Any previous file with the same name will have been over-written and lost. This command is used to close the LST file after it has been enabled by the LSTFILE command and written to. An example of the use of the CLOSE command is given under XREF on Page 7-5.

HLINES

This is the number of lines of information to be displayed on your console by the HELP command before pausing for permission to continue. When changing this parameter, remember that the desired number of lines must be expressed in the current ARGRADix.

LSTFILE

This parameter controls writing to the file designated for disassembly output (NOT the symbol table file). Its default value is OFF. When turned ON, all text displayed on the console is sent to the .LST file. It is NOT closed by returning the value of LSTFILE parameter to OFF; OFF simply inhibits writing to the file. The file must be closed in order to ensure retention of the last buffer-full of text and the proper end-of-file mark. Closure is automatic during execution of the WRITE and QUIT commands. If you are using the .LST File to record XREF output, then you will want to explicitely CLOSE the file in order to reassign the .LST output to another filename for other listing purposes. You will use the CLOSE command.

MNE

The MNE parameter controls the use of Instruction Data Tables (see the memory map in appendix B). These tables are present on the REVAS distribution disk as a set of files whose extension is ".MNE" and whose <FN> field defines the allowed words that may be used as an argument in the 'SET MNE <FN>' command. These files are used to overlay the instruction data tables in REVAS with the net effect of changing the mnemonic set that is used to display a disassembly. If the appropriate files are not present on the default disk when the 'SET MNE' command is issued, an error message will be generated. They do not have to be present if you will not be using that command, but if present they must retain their as-distributed name in order to be accessed. The possible names are listed in the Reserved Words list in Appendix A and can be displayed using the HELP command. You can change mnemonic sets as often as you like at any time during a disassembly with commands like:

```
SET MNE Z80     (for ZILOG standard mnemonics)
SET MNE I85     (for INTEL 8085 mnemonics)
SET MNE MAC     (for Digital Research's MACRO Z80 set)
SET MNE TDL     (for the TDL set initially in REVAS)
```

MODE

The MODE parameter controls the interpretation of source code during disassembly. At the start of execution of each DISPLAY CLASS command, its value is reset to agree with DMODE. During disassembly, the mode control information associated with each label (in the label field) encountered is used to update the value of MODE. MODE controls the interpretation of code until the next label or MARKed control entry is encountered. The value of MODE cannot be SET from the keyboard. It is described here for reference only. The possible values of MODE and DMODE are:

MODE:	Interpretation:
0	INSTRuction mnemonics
1	BYTE data
2	WORD data (i.e., addresses)

Note that capitalized portions of the interpretations above are RESERVED WORDS which have the values 0, 1, and 2, respectively.

ORG

This parameter specifies the origin of the code in a disk file. Such code might not originate at the CP/M standard 100H (the default value); the actual origin must then be specified by setting the value of ORG either before or after the PGM file has been loaded. This value will appear in the STATUS listing as the value of PS for the VIRTUAL access mode. When ORG is specified, PE is automatically recalculated also. The use of ORG allows you to disassemble code intended to run anywhere in memory space, even if you have no memory there!

OUTRAD

Determines the number base to be used when sending data to output devices from the CALC command. The default radix is HEX.

Examples:

```
        SET OUT 8         sets radix to OCTAL from HEX or DEC
        SET OUTRAD 20     sets radix to HEX from OCTAL
        SET OUT DEC       sets radix to DECIMAL
```

Note that, as in the last example, you can always get back to a familiar radix by using one of the familiar words: HEX, BINARY, OCTAL, or (as shown) DECIMAL.

CHAPTER 9

PARAMETER DESCRIPTIONS

Parameters are internal variables within REVAS whose numeric or symbolic value is reported by one of the forms of the STATUS command. All parameters have initial default values. A new value can be assigned to most of the parameters by use of the SET or TURN commands. SETting of parameters is a method of controlling the disassembly and display functions.

AMODE

The AMODE parameter controls the mode assigned to synthetic symbols generated while building symbol tables in WORD mode (WB command or equivalent). Possible values for AMODE are limited to 0,1,2, and 255. The default value is 255, which causes all the values in the operand field of WORD type data to be treated as constants; they are not entered in the symbol tables. A value of 00 (INSTRuction), results in assignment of symbols of the form "USxxxx" where xxxx is the Hex value of the symbol. A value of 01(BYTE) causes symbols to by assigned in the form "UTxxxx" implying that the data at address xxxx is a list of bytes. A value of 02(WORD) causes symbols to be assigned in the form "UWxxxx", implying a list of words at address xxxx. The value of AMODE is determined by keyboard entry using the SET command. When SETting the value, any attempt to assign a value other than the ones listed above will result in assignment of the default value (255). AMODE retains its value until a new value is SET.

ARGRAD

Determines the number base for input of arguments which follow the command word. The default radix is Hex.

Examples:

SET ARG 8	sets radix to Octal from Hex or Dec
SET ARGRAD 20	sets radix to HEX from Octal
SET ARG DEC	sets radix to Decimal

ASMFLAG

This switch controls format of disassembly display. Its default value is OFF, and results in a display that simulates an assembler PRN output listing. When ASMFLAG is turned ON, only Label, Operator, and Operand fields are displayed for Instruction or Word type displays. For Byte type displays, the comment field is also transmitted since it may contain convenient ASCII information. This switch is automatically turned ON while the LST file is being written and is responsible for format in that file.

BOTMAR

The number of blank lines to be left at the bottom of a page during output to the list device. Since the number of lines in the top margin is fixed at 5, the total page length is 5+PLINES+BOTMAR. The default value of BOTMAR is 6, making the total page length the standard 66 lines (at 6 lines per inch for 8.5x11 inch paper). If BOTMAR is assigned a value greater than 127 (decimal), then REVAS will send a form feed character to the printer instead of a defined number of line feeds after the page full of output is printed. Some printers will feed faster in this mode than with discrete line feeds. It also saves you the bother of calculating page length.

CMT

Controls display of the automatic comment field generated by REVAS. The default value is ON. When CMT is turned OFF, the automatic comment field is not generated for lines that display instructions.

CONSOLE

Controls the display of disassemblies on the console device. The default (initial) value is ON. When the console is OFF, commands may still be entered normally, since the console echo is an operating system function.

DMODE

This parameter is the default mode setting. Its value is automatically set by the I,B, and W prefixes of the DISPLAY CLASS commands (such as, for example, BDISPLAY, which sets the default mode to BYTE). DMODE cannot be SET from the keyboard. It is described here for reference only.

ECHO

This parameter is initially OFF. When it is turned ON, each command in the command line is repeated on the console and printer (if they are currently ON) before execution of the command. This function is useful when multiple commands are to be given in the command line; you don't have to look back to see which commands and arguments were used to produce the current output.

PAGER

PAGER is a switch that controls the printer paging function. Its value is zero or not-zero (OFF/ON). Its default value is 'OFF' and the paging function is not active. Output to the printer will be continuous with no top or bottom margins. The command:

```
TURN PAGER ON
```

will activate the paging function. Subsequent printout on the printer will be formatted into a top margin, lines of disassembly text, and a bottom margin.

PAUSE

If you are using a printer that accepts single sheets of paper, you want printing to stop while you insert a new sheet. PAUSE is a switch that allows this. Its value may be zero or not-zero (OFF/ON). Its default value is OFF. When it is turned ON, output is suspended at the end of each page until you depress any key at your console. Printing will then continue with the top margin (including any Title you have specified), the disassembly text, and the bottom margin. Although you may turn PAUSE on or off any time with the SET/TURN command, the pause function will only be active when PAGER is also ON. You will also find PAUSE handy if you are using continuous form paper and are near the end of the box!

PE

Program End address. For complete description, see PS/PE on page 9-6.

PLINES

The number of lines of disassembly which you wish to have displayed per page on the printer. When this number of lines has been printed, enough line feeds will be issued to simulate a form feed. A value of 0FFFF(HEX) results in continuous listing (no page breaks) after an initial top margin. The default value is 37H (55 Decimal) lines.

PRINTER

Controls the display of disassemblies on the system printer. The default value is OFF. When turned ON, output is sent to the printer without affecting other I/O devices. Each is controlled by its own switch parameter.

PS, PE
Program Start, Program End addresses

These parameters are used to define the limits of the program being disassembled, and values are maintained for both the Memory (M#) and Virtual (V#) access modes. Default values are built in, with values visible in the STATUS command. Arguments encountered during disassembly which fall outside the range defined by PS and PE are treated as constants and not assigned as symbols during BUILD. When a disk file is first accessed for disassembly, PE is reset to correspond with the file length in the disk directory; i.e., at the end of the last segment of the file. The values of PS and PE are stored with the symbol tables in the .TBL file, so on the next disassembly of the same file, they are automatically reset to the values last used.

PUNCH

Controls output to the device assigned to the CP/M punch channel. The default value is OFF.

RPTRAD

Determines the number base (radix) which REVAS uses to interpret the RPT argument preceding a command word. The default radix is decimal. You may SET this parameter to another radix by specifying the new base numerically in terms of the current one, or by using one of the four Reserved Words (BINARY, OCTAL, DECIMAL, HEX) to specify the base. When the reserved word method is used, you don't have to worry about getting the numeric argument right; it works just like you expect it to.

Examples:

SET RPTRAD 16	sets radix to HEX from DECIMAL
SET RPT 0A	sets radix to DECIMAL from HEX
SET RPT DECIMAL	sets radix to DECIMAL

TABLE

Access to the symbol tables is controlled by the TABLE switch parameter. Its initial value is ON, permitting entry and retrieval of records from the tables. The status of this switch is sensitive to the kind of target program access (indicated by the REVAS prompt character, M# or V#) in effect when the first entry is made in the symbol table. Thereafter, the switch will be ON for that program access mode, and OFF for the other. When the switch is OFF, the BUILD, EQUALS, and MARK commands will do nothing, and the tables will appear to be empty to all disassembly functions. If table access is permitted (TABLE is ON) for VIRTUAL access (V#), a switch to MEMORY access mode (M#) will cause the TABLE parameter to be OFF, prohibiting access to the symbol tables. In those rare cases where this is inconvenient, the TABLE switch can be turned ON/OFF as required.

TOP

When sending paginated text to the printer it is necessary to adjust the paper so that the printer will start printing at the top of the page; and then to so inform REVAS. TOP is the switch parameter which records the fact that the printer is at the top of a page. It has the same zero/not-zero (OFF/ON) values as other switches. When you have adjusted the printer to top-of-page, then you must type one of the following commands to let REVAS know:

```
        SET TOP
or      TURN TOP ON
```

The TOP parameter is automatically maintained by REVAS; its value is not altered while PAGER is OFF, and is automatically SET at top-of-page when PAGER is ON and a page of print has been output. The only time that it is necessary for you to SET TOP is when printing has stopped at other than a page end, and you wish to advance the paper and continue on a new page.

CHAPTER 10

CHANGING REVAS

DEFAULT PARAMETER VALUES

Some of the "permanent" control parameters in REVAS may need to be changed to suit your particular desires or system configuration. For this purpose, there is a block of constants and pointers near the beginning of the program whose values may be appropriately modified using your system debugger or monitor. The contents of that block are shown in the listing below. In order to make changes, load REVAS at location 0100 with your system debugger. Make a note of the length of REVAS, then make the changes you desire. If necessary, consult your debugger manual for instructions on how to load a file, make changes, and resave it. You will need the length of the REVAS file for use when saving the modified copy. DO NOT EXECUTE REVAS UNTIL AFTER IT HAS BEEN RESAVED WITH THE NEW PATCH VALUES. REVAS is designed to run properly only as a transient program under CP/M.

Address	Hex contents	Description
0100	C3 XX XX	Jump around CONSTANTS section Do not change the values at XX XX!
0103	0A	Default number of lines to display before pausing during HELP
0105	37	Printer text lines/page
0107	06	Bottom margin on print page
0109	00	paging control byte: 0FFH enables
010B	00	0FFH enables pause at top-of-page
010D	3A	ASCII value of label delimiter
010E	3B	ASCII value of comment delimiter
010F	2F	ASCII value of BUILD action char.
.	.	
011E	00	40H free bytes for user patches
.	.	
015D	.	

SYNTHETIC LABEL CHARACTER

The initial letter for locked synthetic labels is obtained from a list of three ASCII values ("STW") in memory. The location of this list can be found by subtracting 4 from the 16 bit pointer value at address 0118H in REVAS. You can use your debugger to change these characters to suit your fancy. The default characters were selected such that they would not be visually confused with HEX digits in a printout.

"FORBIDDEN" OUTPUT CHARACTERS

REVAS does not send control characters to the console, printer, punch, or ASCII files (Except for the standard EOF character, 1AH). The automatic Comment field in a line of disassembly is a literal transmission of the code being disassembled, except that non printing code bytes are replaced with a period. Control characters fall into this category, and are handled automatically within the body of REVAS. 'RUBOUT' is another such character which could cause strange results if allowed to be sent in an ASCII string. Your system may not be able to display certain characters, or may have special responses to certain characters. If you need to avoid having such characters sent to your peripherals, you can patch them into the data string near the beginning of REVAS as shown below. Up to six such 'forbidden' characters will then be replaced with '.' in the comment field, avoiding any unwanted side effects.

Address	Hex contents	Description
0111	7F 00 00	List of up to six values for
0114	00 00 00	ASCII characters which must be
0117	00	replaced with '.' in comment
		field. (See CMT parameter.)

Note that control characters are already so treated, and 7FH (rubout) is in the list already. Other typical candidates for inclusion are 5EH, 5FH, and 7EH. A value of zero terminates the list, so you may add 5 more character values (Hex ASCII) starting at location 112H by replacing successive null entries. Do not modify the (null) value at location 117H, since that is the last available list terminator!

USER PATCH AREA

Starting at HEX address 011E is a 64-byte (40 HEX) region of memory that is currently not used by REVAS. This area is available for user patches or routines that would be available to the CALL command. One such patch might be a minor revision that does not warrant reassembly and redistribution of a new version.

INTERUPTS

REVAS disables interupts during certain functions, then re-enables interupts using the code (EI, RET) at location 015FH. If your system requires that interupts not be enabled by REVAS, then you must change the EI instruction at 015FH to either NOP or DI.

FINDING THE MNEMONIC TABLES

Address	Hex contents	Description
0118	(PNTR)	Address at which mnemonic-dependant tables begin

The value at 0118H, (PNTR), is the address of the list of vectors at the beginning of the instruction data tables (IDT). The actual value is entered at the indicated location during assembly and is present in your copy. It is present to help you find the table of mnemonics in order to make minor changes such as changing '.END' to 'END' or '.BYTE' to 'DB', etc. You will find the MEMORY MAP in Appendix B-1 helpful in understanding the organization of the tables. (PNTR) is the address of the first word in the list. The 22nd word (WORD22) is the address of the list of mnemonics.

To find the tables, execute REVAS in MEMORY mode (M# prompt). First, display the word value at 118H to see PNTR. Then display at least 22 words at that address. The 22nd one is the starting address of the mnemonics. If you display at <u>that</u> address using byte format, you will see the ASCII for the mnemonics in the comment field of the disassembly. Whew! Record the address for future reference. Exit REVAS (QUIT command or CNTL-C), then bring in a copy of REVAS from disc with your system debugger. The object is to make any changes to an <u>unexecuted</u> copy of REVAS. Use the address you recorded to locate the mnemonics. They are in alphabetic order except for a few at the end of the list. Consult the paragraph on page 10-1 for additional information.

If you elect to make changes to any of the mnemonics, observe that:

a) the total number of bytes in the table must be unchanged,
b) the last byte of each mnemonic entry must have the high bit set (=1) or may be null or '$'.
c) there must be no more than ONE byte with high bit set for each mnemonic entry, and no nulls or '$' in the mnemonic.

If the new mnemonic is shorter than that in the table, the entry must be filled with spaces (20H). The last byte may be 0A0H (20H with high bit set), or it may be null (00) or '$'. For example, to change '.BYTE' to 'DB',

```
WAS:            2E 42 59 54 C5
CHANGE TO:      44 42 20 20 A0
```

The terminal byte marks the end of a mnemonic entry. The next byte is assumed to be the beginning of the next mnemonic in the list. If the terminal byte is null or '$', it is not printed out as part of the mnemonic. If the terminal byte has its high bit set, then the seven low order bits are considered to be printable ASCII to be included in the mnemonic.

CHANGING THE DEFAULT MNEMONIC SET

REVAS as distributed disassembles using the TDL mnemonic set. You may wish to change to one of the others that are present on your distribution disk as the default set. (You have your choice of one of the .MNE files.) To do so, bring an unexecuted REVAS.COM into memory at its normal execution address under CP/M (100H), locate the address at which the mnemonic-dependant tables begin, load the desired .MNE file at that address, and then re-save REVAS.COM with the CP/M SAVE command. Here are the steps:

1). Use STAT to determine the length of REVAS.COM. Make a note of the length for future reference.

2). Use your system debugger to load REVAS at 100H, and find the destination address (PNTR) as described on the preceding page.

3). Examine the code at the target address. For the TDL mnemonic set, the first two bytes will be 00H. For other mnemonic sets, the first byte will be non-zero; the second will always be 00H.

4). Use your debugger to load the xxx.MNE file at the destination address. If you're unsure about the offset (bias) to use, consult your debugger documentation and make trial runs until you are satisfied that you have the offset properly defined. Then reload REVAS and the xxx.MNE file.

WARNING!! DO NOT EXECUTE REVAS FROM THE DEGUGGER PRIOR TO SAVING IT!! REVAS WILL BE ALTERED AND THE SUBSEQUENTLY SAVED IMAGE WILL NOT WORK PROPERLY!!

5). Exit the debugger (control-C) and immediately SAVE the newly created version with the length you recorded in step 1, giving it a name like TEMP.COM.

6). Test the new version in the usual way by typing its name after the Z or CP/M prompt. If the DISPLAY command works properly, everything else is probably OK.

7). Rename TEMP.COM to REVAS.COM (or whatever else you like) on your working disk(s), but do not replace the master copy. If the overlay was not done accurately, you can always retreat to the master copy and try again..

BUFFER SIZES

The size of I/O buffers is allocated during initialization of REVAS just before sending the sign-on message to the console. You can change the buffer size if you wish by changing the values stored at the following addresses. Note carefully that these are 16 bit values stored in standard low-byte-first order. Unless you have an absolutely incredible amount of memory to squander, the high byte will always be zero!

Address	Hex contents	Description
011A	08 00	size of target program buffer
011C	04 00	size of the .LST buffer

Both of the above sizes are expressed as the number of 128 byte records in the buffer. Larger buffers mean less time is required for disk access during a disassembly session, but at the expense of less memory available for symbol tables.

FORMAL COMMAND SYNTAX

Metalanguage Definitions:

```
        !        = logical OR
        [  ]     = optionally present
        [ ...]   = present any number of times
        <  >     = defines syntactic unit
        (  )     = establishes logical grouping
        CR       = ASCII CR (carriage return)
        SP       = ASCII SP (input from space bar)
```

Command Syntax:

```
<command line>  =: <command>[;<command>...] CR!<null command>

<command>       =: [<macro name>] ! [ [<arg>] [<del><arg>]...]

<null command>  =: [<del>...] CR ! (; [;...])

<del>           =: [SP...] (SP ! ',') [SP...]

<num>           =: [<digit>] [<rdigit>...]

<digit>         =: 0!1!2!3!4!5!6!7!8!9

<rdigit>        =: <any char in current input radix>

<arg>           =: <num> ! <string>

<string>        =: <alpha>[ (<alpha> ! <digit>)...]

<alpha>         =: any printable char except (<digit>!<special>)

<special>       =: ',' ! SP ! CR ! ';'
```

Notes:

1) A null command terminates parsing of the command line; any following commands on that line will be ignored.

2) Commas have a special delimiter function:
 a) one comma (see <del>) is a delimiter
 b) n commas define n-1 blank arguments

3) At least one delimiter must separate elements of a command

4) Numbers must start with a decimal digit if there is a possibility of confusion with a symbol, command word, or abbreviation.

COMMAND LIST

Command	Abbr.	Function	Page
AT	(A)	Enter/Delete comments	7-1
BBUILD	(BB)	BYTE mode build	6-5
BDISPLAY	(BD)	BYTE display	6-2
BPRINT	(BP)	BYTE display and print	6-4
BUILD	(B)	BUILD SYMBOL TBLS	6-5
CALC	(CA)	A+B, A-B, A*B, A DIV B, A MOD B	5-3
CALL		Call a user subroutine	5-4
CLOSE	(C)	Close the .LST file	8-4
DISPLAY	(D)	MAIN DISASSEMBLY COMMAND	6-1
EQUALS	(E)	Assign or change a label	7-1
ERASE	(ER)	Remove a MACRO	5-4
FILES	(F)	Set up PGM, TBL, and LST files	8-2
FIND	(FIN)	Find <LABEL> or <ADDR ARG>	7-2
HELP	(H)	Print HELP lists	5-4
IBUILD	(IB)	INSTR mode build	6-5
IDISPLAY	(ID)	INSTR mode display	6-2
IPRINT	(IP)	INSTR mode display and print	6-4
KILL	(K)	Remove table entry	7-3
LOCK	(L)	Prohibit automatic mode change	7-3
LSTFILE	(LS)	Establish <FN> for LST file	8-3
MACRO	(MAC)	Define following cmds as MACRO <NAME>	5-5
MARK	(M)	Set data type at named address	7-4
MEMORY	(ME)	Switch to MEMORY mode	5-5
PGMFILE	(PG)	Establish <FN> for PGM file	8-3
PRINT	(P)	Display on printer & console	6-4
REINIT	(REINIT)	erase memory & restart	5-6
QUIT	(Q)	exit to CPM	5-6
SAVE	(SA)	Save symbol tables	8-4
SET	(SE)	Set parameters	5-7
SHOW	(SH)	Display symbols, macros, comments	5-6
STATUS	(S)	List current parameters	5-8
TBLFILE	(TB)	Establish <FN> for TBL file	8-3
TITLE	(TI)	define page header for PRINT	5-8
TURN	(T)	same as SET	5-7
UNLOCK	(U)	Allow REVAS to reassign mode	7-3
VIRTUAL	(V)	Switch to DISK FILE mode	5-5
WBUILD	(WB)	WORD mode build	6-5
WDISPLAY	(WD)	WORD display	6-2
WPRINT	(WP)	WORD display and print	6-4
WRITE	(WR)	Send disassy to lstfile	8-4
XREF	(X)	Print cross reference table	7-5
E,X,S		INSTANT COMMANDS	5-3

PARAMETER LIST

Parameter	Abbr.	Value/Function	Page
AMODE	(AM)	argument mode for WORD data	9-1
ARGRAD	(AR)	input radix for numeric arguments	9-1
ASMFLAG	(A)	ON/OFF, ASM/PRN format	9-1
BOTMAR	(BOT)	bottom margin, lines	9-2
CMT	(CM)	ON/OFF, auto comments	9-2
CONSOLE	(C)	ON/OFF	9-2
ECHO	(E)	ON/OFF, command repeat	9-2
HLINES	(H)	lines per screen for HELP	9-3
LSTFILE	(L)	ON/OFF	9-3
MNE	(M)	Select new Mnemonic Set	9-3
ORG	(O)	Origin of PGM file	9-4
OUTRAD	(OU)	display radix for CALC	9-4
PAGER	(PA)	ON/OFF, paging on list device	9-5
PAUSE	(PAU)	ON/OFF, pause at page top	9-5
PE	(P)	PGM file end	9-5
PLINES	(PL)	text lines per page	9-5
PRINTER	(PR)	ON/OFF	9-6
PUNCH	(PU)	ON/OFF	9-6
RPTRAD	(R)	input radix for (n)	9-6
TABLE	(T)	ON/OFF, access in current mode	9-7
TOP	(TO)	list device is at top of page	9-7

RESERVED WORD LIST

Res. Wd.	Abbr.	HEX value	Page
OFF	(OF) =	0000	2-11
ON	(ON) =	0FFFF	2-11
INSTR	(IN) =	0000	2-11
BYTE	(BY) =	0001	2-11
WORD	(WO) =	0002	2-11
TPA	(TP) =	0100	2-11
PSTART	(PS) =	current PS or ORG	2-11
PEND	(PE) =	current pgm end	2-11
BINARY	(BI) =	0002	2-11
OCTAL	(OC) =	0008	2-11
DECIMAL	(DE) =	000A	2-11
HEX	(HE) =	0010	2-11

The following words are arguments for SET MNE ....

TDL	(TDL)=	0000 selects TDL 8080/Z80 mnemonics	9-3
Z80	(Z80)=	0001 selects Zilog Z80 mnemonics	9-3
MAC	(MAC)=	0002 selects Z80 mnemonics per RMAC	9-3
I80	(I80)=	0080 selects Intel 8080 mnemonics	9-3
I85	(I85)=	0081 selects Intel 8085 mnemonics	9-3

(The values shown are the first word in the MNE table file.)

```
              MAIN MEMORY MAP                           INSTRUCTION DATA TABLES

0000      :--------------------------------:                 ::                              ::
          :  CP/M POINTERS,BUFFERS, ETC    :      --(PNTR)   ::------------------------------::
          :                                :                 ::  23 WORD VECTOR LIST         ::
0100     ::================================::                ::  WORD01                      ::
         ::  CONSTANTS & POINTERS          ::                ::     .                        ::
         ::                                ::                ::     .                        ::
         ::--------------------------------::                ::     .                        ::
         ::  MAIN BODY OF REVAS            ::                ::  WORD23                      ::
         ::  APPROX 13K BYTES              ::                ::                              ::
         ::                                ::       WORD02   ::------------------------------::
         ::                                ::                ::  REGISTER NAMES              ::
         ::                                ::                ::                              ::
(PNTR)   ::--------------------------------::--              ::                              ::
         ::  INSTRUCTION DATA TABLES       ::                ::                              ::
         ::  APPROX 1K BYTES               ::       WORD08   ::------------------------------::
         ::                                ::                ::  INSTRUCTION DECODE DATA     ::
BUF1     ::================================::--              ::                              ::
          : PGMFILE BUFFER                 :                 ::                              ::
          : 1024 BYTES (DEFAULT)           :                 ::                              ::
          :                                :        WORD21   ::------------------------------::
BUF2      :--------------------------------:                 ::  MNEMONIC LIST               ::
          : LSTFILE BUFFER                 :                 ::                              ::
          : 512 BYTES (DEFAULT)            :                 ::                              ::
HDR       :--------------------------------:                 ::                              ::
          : SYMBOL TABLE HEADER            :        WORD22   ::------------------------------::
          :                                :                 ::  'END' PSEUDO-OP + CRLF      ::
          :--------------------------------:      --WORD23   ::==============================::
          : SYMBOL TABLES                  :                  :  (BUF1 STARTS HERE)          :
          :     .                          :                  .
                .
                .
CCP       :--------------------------------:
          :  CP/M CONSOLE COMMAND          :
             PROCESSOR & OPERATING
                 SYSTEM
          :--------------------------------:
          :  TOP OF SYSTEM MEMORY          :
          :--------------------------------:
```

Initial Disassembly of XD.OBJ

```
0100 21 0000        LXI     H,0000H        ;!..
0103 39             DAD     SP             ;9
0104 22 BB06        SHLD    06BBH          ;";.
0107 31 F906        LXI     SP,06F9H       ;1y.
010A C3 3501        JMP     0135H          ;C5.

010D 2600           MVI     H,00H          ;&.
010F 3A B606        LDA     06B6H          ;:6.
0112 6F             MOV     L,A            ;o
0113 54             MOV     D,H            ;T
0114 29             DAD     H              ;)
0115 29             DAD     H              ;)
0116 29             DAD     H              ;)
0117 3A B706        LDA     06B7H          ;:7.
011A 5F             MOV     E,A            ;_
011B 19             DAD     D              ;.
011C 11 E6FF        LXI     D,0FFE6H       ;.f.
011F 3E01           MVI     A,01H          ;>.
0121 19             DAD     D              ;.
0122 3C             INR     A              ;<
0123 DA 2101        JC      0121H          ;Z!.
0126 11 1907        LXI     D,0719H        ;...
0129 19             DAD     D              ;.
012A 32 B106        STA     06B1H          ;21.
012D 7E             MOV     A,M            ;~
012E 32 B206        STA     06B2H          ;22.
0131 32 B306        STA     06B3H          ;23.
0134 C9             RET                    ;I

0135 0E19           MVI     C,19H          ;..
0137 CD 0500        CALL    0005H          ;M..
013A 32 B006        STA     06B0H          ;20.
013D 3A 8100        LDA     0081H          ;:..
0140 B7             ORA     A              ;7
0141 CA 5D01        JZ      015DH          ;J].
0144 3A 8200        LDA     0082H          ;:..
0147 FE41           CPI     41H            ;~A
0149 C2 5301        JNZ     0153H          ;BS.
014C AF             XRA     A              ;/
014D 32 B006        STA     06B0H          ;20.
0150 C3 5D01        JMP     015DH          ;C].

0153 FE42           CPI     42H            ;~B
0155 C2 4D05        JNZ     054DH          ;BM.
0158 3E01           MVI     A,01H          ;>.
015A 32 B006        STA     06B0H          ;20.
015D C3 8C01        JMP     018CH          ;C..
```

```
0160 0D              DCR     C              ;.
0161 0A              LDAX    B              ;.
0162 0A              LDAX    B              ;.
0163 2020            JRNZ    0185H          ;..
0165 2020            JRNZ    0187H          ;..
0167 2020            JRNZ    0189H          ;..
0169 2020            JRNZ    018BH          ;..
016B 2020            JRNZ    018DH          ;..
016D 2020            JRNZ    018FH          ;..
016F 2020            JRNZ    0191H          ;..
0171 2020            JRNZ    0193H          ;..
0173 2020            JRNZ    0195H          ;..
0175 2020            JRNZ    0197H          ;..
0177 2020            JRNZ    0199H          ;..
0179 44              MOV     B,H            ;D
017A 49              MOV     C,C            ;I
017B 52              MOV     D,D            ;R
017C 45              MOV     B,L            ;E
017D 43              MOV     B,E            ;C
017E 54              MOV     D,H            ;T
017F 4F              MOV     C,A            ;O
0180 52              MOV     D,D            ;R
0181 59              MOV     E,C            ;Y
0182 2044            JRNZ    01C8H          ;.D
0184 52              MOV     D,D            ;R
0185 49              MOV     C,C            ;I
0186 56              MOV     D,M            ;V
0187 45              MOV     B,L            ;E
0188 202D            JRNZ    01B7H          ;.-
018A 2024            JRNZ    01B0H          ;.$
018C 11 6001         LXI     D,0160H        ;.`.
018F 0E09            MVI     C,09H          ;..
```

Disassembly after Building Tables

```
0100 21 0000                LXI     H,0000H             ;!..
0103 39                     DAD     SP                  ;9
0104 22 BB06                SHLD    UT06BB              ;";.
0107 31 F906                LXI     SP,UT06F9           ;1y.
010A C3 3501                JMP     S0135               ;C5.

010D 2600                   MVI     H,00H               ;&.
010F 3A B606                LDA     UT06B6              ;:6.
0112 6F                     MOV     L,A                 ;o
0113 54                     MOV     D,H                 ;T
0114 29                     DAD     H                   ;)
0115 29                     DAD     H                   ;)
0116 29                     DAD     H                   ;)
0117 3A B706                LDA     UT06B7              ;:7.
011A 5F                     MOV     E,A                 ;_
011B 19                     DAD     D                   ;.
011C 11 E6FF                LXI     D,0FFE6H            ;.f.
011F 3E01                   MVI     A,01H               ;>.
0121 19         S0121:      DAD     D                   ;.
0122 3C                     INR     A                   ;<
0123 DA 2101                JC      S0121               ;Z!.
0126 11 1907                LXI     D,UT0719            ;...
0129 19                     DAD     D                   ;.
012A 32 B106                STA     UT06B1              ;21.
012D 7E                     MOV     A,M                 ;~
012E 32 B206                STA     UT06B2              ;22.
0131 32 B306                STA     UT06B3              ;23.
0134 C9                     RET                         ;I

0135 0E19       S0135:      MVI     C,19H               ;..
0137 CD 0500                CALL    0005H               ;M..
013A 32 B006                STA     UT06B0              ;20.
013D 3A 8100                LDA     0081H               ;:..
0140 B7                     ORA     A                   ;7
0141 CA 5D01                JZ      S015D               ;J].
0144 3A 8200                LDA     0082H               ;:..
0147 FE41                   CPI     41H                 ;~A
0149 C2 5301                JNZ     S0153               ;BS.
014C AF                     XRA     A                   ;/
014D 32 B006                STA     UT06B0              ;20.
0150 C3 5D01                JMP     S015D               ;C].

0153 FE42       S0153:      CPI     42H                 ;~B
0155 C2 4D05                JNZ     S054D               ;BM.
0158 3E01                   MVI     A,01H               ;>.
015A 32 B006                STA     UT06B0              ;20.
015D C3 8C01    S015D:      JMP     S018C               ;C..

0160 0D0A 0A20  UT0160:     .BYTE   0DH,0AH,0AH,20H     ;....
0164 2020 2020              .BYTE   20H,20H,20H,20H     ;....
0168 2020 2020              .BYTE   20H,20H,20H,20H     ;....
016C 2020 2020              .BYTE   20H,20H,20H,20H     ;....
0170 2020 2020              .BYTE   20H,20H,20H,20H     ;....
0174 2020 2020              .BYTE   20H,20H,20H,20H     ;....
```

```
0178 2044 4952              .BYTE    20H,44H,49H,52H            ;.DIR
017C 4543 544F              .BYTE    45H,43H,54H,4FH            ;ECTO
0180 5259 2044              .BYTE    52H,59H,20H,44H            ;RY.D
0184 5249 5645              .BYTE    52H,49H,56H,45H            ;RIVE
0188 202D 2024              .BYTE    20H,2DH,20H,24H            ;.-.$

018C 11 6001       S018C:   LXI      D,UT0160                   ;.`.
018F 0E09                   MVI      C,09H                      ;..
0191 CD 0500                CALL     0005H                      ;M..
0194 3A B006                LDA      UT06B0                     ;:0.
0197 B7                     ORA      A                          ;7
0198 C2 AE01                JNZ      S01AE                      ;B..
019B C3 A301                JMP      S01A3                      ;C#.

019E 41                     MOV      B,C                        ;A
019F 0D                     DCR      C                          ;.
01A0 0A                     LDAX     B                          ;.
01A1 0A                     LDAX     B                          ;.
01A2 24                     INR      H                          ;$
01A3 11 9E01       S01A3:   LXI      D,019EH                    ;...
01A6 0E09                   MVI      C,09H                      ;..
01A8 CD 0500                CALL     0005H                      ;M..
01AB C3 C101                JMP      01C1H                      ;CA.
01AE C3 B601       S01AE:   JMP      01B6H                      ;C6.

01B1 42                     MOV      B,D                        ;B
01B2 0D                     DCR      C                          ;.
01B3 0A                     LDAX     B                          ;.
01B4 0A                     LDAX     B                          ;.
01B5 24                     INR      H                          ;$
01B6 11 B101                LXI      D,01B1H                    ;.1.
01B9 0E09                   MVI      C,09H                      ;..
01BB CD 0500                CALL     0005H                      ;M..
01BE CD 8004                CALL     0480H                      ;M..
01C1 AF                     XRA      A                          ;/
01C2 32 B706                STA      UT06B7                     ;27.
```

Disassembly With Real Labels and Comment Examples

```
;REVAS INSERTS COMMENTS LIKE THIS
0100 21 0000     XD:     LXI     H,0000H                       ;!..
0103 39                  DAD     SP                            ;9
0104 22 BB06             SHLD    OLDSP                         ;";.
0107 31 F906             LXI     SP,STACK                      ;1y.
010A C3 3501             JMP     START                         ;C5.

010D 2600        DSKADR: MVI     H,00H                         ;&.
010F 3A B606             LDA     GROUP                         ;:6.
0112 6F                  MOV     L,A                           ;o
;FREE ZERO FROM H
0113 54                  MOV     D,H                           ;T
;GROUP*8=LOGICAL SECTORS
0114 29                  DAD     H                             ;)
0115 29                  DAD     H                             ;)
0116 29                  DAD     H                             ;)
0117 3A B706             LDA     REMSEC                        ;:7.
011A 5F                  MOV     E,A                           ;_
011B 19                  DAD     D                             ;.
;DE= -26 (SECTORS/TRACK)
011C 11 E6FF             LXI     D,0FFE6H                      ;.f.
011F 3E01                MVI     A,01H                         ;>.
0121 19          NXTRK:  DAD     D                             ;.
0122 3C                  INR     A                             ;<
0123 DA 2101             JC      NXTRK                         ;Z!.
0126 11 1907             LXI     D,DIRBUF                      ;...
0129 19                  DAD     D                             ;.
012A 32 B106             STA     TRACK                         ;21.
012D 7E                  MOV     A,M                           ;~
012E 32 B206             STA     SECTOR                        ;22.
0131 32 B306             STA     RECORD                        ;23.
0134 C9                  RET                                   ;I

;FROM HERE ON USE YOUR OWN COMMENTS
0135 0E19        START:  MVI     C,19H                         ;..
0137 CD 0500             CALL    BDOS                          ;M..
013A 32 B006             STA     REQDRV                        ;20.
013D 3A 8100             LDA     0081H                         ;:..
0140 B7                  ORA     A                             ;7
0141 CA 5D01             JZ      DODIR                         ;J].
0144 3A 8200             LDA     0082H                         ;:..
0147 FE41                CPI     41H                           ;~A
0149 C2 5301             JNZ     NOT.A                         ;BS.
014C AF                  XRA     A                             ;/
014D 32 B006             STA     REQDRV                        ;20.
0150 C3 5D01             JMP     DODIR                         ;C].

0153 FE42        NOT.A:  CPI     42H                           ;~B
0155 C2 4D05             JNZ     JDRERR                        ;BM.
0158 3E01                MVI     A,01H                         ;>.
015A 32 B006             STA     REQDRV                        ;20.
015D C3 8C01     DODIR:  JMP     DOHDR                         ;C..
```

```
0160 0D0A 0A20  HDRTXT: .BYTE   0DH,0AH,0AH,20H         ;....
0164 2020 2020          .BYTE   20H,20H,20H,20H         ;....
0168 2020 2020          .BYTE   20H,20H,20H,20H         ;....
016C 2020 2020          .BYTE   20H,20H,20H,20H         ;....
0170 2020 2020          .BYTE   20H,20H,20H,20H         ;....
0174 2020 2020          .BYTE   20H,20H,20H,20H         ;....
0178 2044 4952          .BYTE   20H,44H,49H,52H         ;.DIR
017C 4543 544F          .BYTE   45H,43H,54H,4FH         ;ECTO
0180 5259 2044          .BYTE   52H,59H,20H,44H         ;RY.D
0184 5249 5645          .BYTE   52H,49H,56H,45H         ;RIVE
0188 202D 2024          .BYTE   20H,2DH,20H,24H         ;.-.$

018C 11 6001    DOHDR:  LXI     D,HDRTXT                ;.`.
018F 0E09               MVI     C,09H                   ;..
0191 CD 0500            CALL    BDOS                    ;M..
0194 3A B006            LDA     REQDRV                  ;:0.
0197 B7                 ORA     A                       ;7
0198 C2 AE01            JNZ     JDRVB                   ;B..
019B C3 A301            JMP     DRVA                    ;C#.

019E 410D 0A0A  HDRA:   .BYTE   41H,0DH,0AH,0AH         ;A...
01A2 24                 .BYTE   24H                     ;$

01A3 11 9E01    DRVA:   LXI     D,HDRA                  ;...
01A6 0E09               MVI     C,09H                   ;..
01A8 CD 0500            CALL    BDOS                    ;M..
01AB C3 C101            JMP     INIT                    ;CA.
01AE C3 B601    JDRVB:  JMP     DRVB                    ;C6.

01B1 420D 0A0A  HDRB:   .BYTE   42H,0DH,0AH,0AH         ;B...
01B5 24                 .BYTE   24H                     ;$

01B6 11 B101    DRVB:   LXI     D,HDRB                  ;.1.
01B9 0E09               MVI     C,09H                   ;..
01BB CD 0500            CALL    BDOS                    ;M..
01BE CD 8004            CALL    LOGREQ                  ;M..
01C1 AF         INIT:   XRA     A                       ;/
01C2 32 B706            STA     REMSEC                  ;27.
01C5 32 B606            STA     GROUP                   ;26.
01C8 32 B806            STA     FCOUNT                  ;28.
01CB 21 1907            LXI     H,DIRBUF                ;!..
01CE 0E83               MVI     C,83H                   ;..
01D0 1E00               MVI     E,00H                   ;..
01D2 73         ..FILL: MOV     M,E                     ;s
01D3 23                 INX     H                       ;#
01D4 0D                 DCR     C                       ;.
01D5 C2 D201            JNZ     ..FILL                  ;BR.
01D8 21 9B07            LXI     H,SORTBF                ;!..
01DB 22 FD06            SHLD    SBFPTR                  ;"}.
01DE 21 1907            LXI     H,DIRBUF                ;!..
01E1 22 AD06            SHLD    DIRPTR                  ;"-.
01E4 3A B006            LDA     REQDRV                  ;:0.
01E7 5F                 MOV     E,A                     ;_
```

A MACRO DEMONSTRATION

This appendix illustrates the result of execution of the MACro, Help, Status, and SHow commands. Paragraphs typed in upper case only were produced by REVAS; paragraphs which contain lower case text (like this one) were inserted later with an editor. The file into which REVAS wrote was named XD.DOC. You will see it reported in the 'FILES IN USE' portion of the STATUS command output. All of the output was produced with REVAS by execution of a command macro named DEMO, which is itself displayed as a result of the SHow Macro command.

Here are the steps used to produce the listing:

```
     a) disassembly of XD.OBJ to produce XD.TBL
     b) generate the DEMO macro with:
     MACRO DEMO;T E ON;T L ON;SH M;H;S;SH I;SH S;T L OFF;T E OFF
     c) assign the name XD.DOC to the LST file:
          command used: LST  .DOC
     d) execute the macro with the command: DEMO
```

And here is the result:

The SHow Macro command displays all currently defined command macros. The command (preceded by '###') is displayed before execution because the Echo parameter was turned on by 'T E ON'. The first two commands in the macro are not displayed here because output to the XD.DOC file was not initiated until after execution of the second command, 'T L ON' (Turn Lstfile ON).

```
###SH M
DEMO    [T E ON;T L ON;SH M;H;S;SH I;SH S;T L OFF;T E OFF]
```

This is what you get if you type HELP, HEL, HE, or H. As you see, it tells you how to get more specific assistance.

```
###H
FOR BRIEF DESCRIPTIONS OF:
COMMANDS , TYPE: HELP C
PARAMETERS , TYPE: HELP P
RESERVED WORDS, TYPE: HELP R
```

The Status command gives information on the current program counter (PC), the logical first address of the program (PS), the program end (PE), the most recent range operated on (DS, DE), current file assignments, and an indication of memory used.

```
###S

V#  PC=0100 PS=0100 PE=07FF DS=0100 DE=FFFF
M#  PC=0000 PS=0000 PE=FFFF DS=0000 DE=0000
DISPLAY MODE: INSTR, DEFAULT: INSTR

FILES IN USE:
PGM:: XD.COM
TBL:: XD.TBL
LST:: XD.DOC

FREE MEM AT: 4A8D
```

The SHow Index command displays synthetic symbols for table entries to which real symbols have not been assigned. Each symbol is a concatenation of a variable length attribute prefix and a 4 digit hex value. If the prefix contains '*', a control entry is implied. If a 'U' is present, then the data type can be changed only by specific operator command. The letters 'S', 'T', and 'W' identify the data type of the code at this address as Instruction, Byte, or Word, respectively.

```
###SH I
*S0103      UT020E      UT0223      UT022E      S0257       S0271
S0272       S027C       S0293       S0297       S02A2       S02AC
S02DF       S02F3       S0300       S0306       S031E       S0323
S032F       UT034A      S0351       UT037D      S0384       S03C2
UT03CD      S03CF       S03FA       S0408       UT040E      S0411
S041C       UT041F      S0423       S042B       S0439       S0442
S045F       UT0465      S0467       S0470       UT04AE      S04B0
UT04C3      S04C5       UT04CE      UT04D9      S04DC       UT04DF
S04EF       UT0502      S050D       UT0524      S0540       S0548
UT0550      S056E       S059B       S05C0       S05C7       S05D5
S05DC       S05DF       S05EA       S060D       S0631       S0656
S0664       *T069A      UT069F      UT06A1      UT06A3      UT06A5
UT06A7      UT06A9      UT06B4      UT06B5      UT06BA      *T06BD
*T06FF
```

The SHow Symbol command displays real (user assigned) symbols and their equivalent synthetic ones. Use of this command is a way to review the value and attributes associated with a real symbol, since only the symbol itself is displayed during disassembly.

```
###SH S
DIRBUF=UT0719    DIRPTR=UT06AD    DODIR =S015D     DOHDR =S018C
FENCE =UT06B9    GROUP =UT06B6    HDRA  =UT019E    HDRB  =UT01B1
HDRTXT=UT0160    INIT  =S01C1     JDRERR=S054D     JDRVB =S01AE
LOGREQ=S0480     MOVSEC=S0245     NOT.A =S0153     NXTRK =S0121
REMSEC=UT06B7    REQDRV=UT06B0    RHLR  =S05E2     SBFPTR=W06FD
STACK =W06F9     START =S0135     TESTMT=S0230     TRACK =UT06B1
```

Turn Lstfile OFf terminates output to XD.DOC

```
###T L OFF
```

The last command in the DEMO macro (T E OFF) does not appear here because output to the Lstfile has already been terminated.