UniFLEX DYNAMITE+ Disassembler User's Manual

Table of Contents

UniFLEX DYNAMITE+ Disassembler User's Manual

1. INTRODUCTION

A disassembler is a program which, as the name implies, provides a function opposite that of an assembler. An assembler takes text files which you type in, and converts them to object files which can be used by the computer. These object files, while perfect for a computer, are not suitable for human consumption. Thus, the job of a disassembler is to take an object file, and from it create a text file which can be understood. Assemblers and disassemblers are therefore complementary programs.

A disassembler can be useful for many jobs. You might have a program which needs to be modified, but only have an object file of the code. With a disassembler, you can translate this object into a text file, which you can then update as required. The disassembler will not do the whole job, ' 't a good disassembler will make the job infinitely easier. A good way to ,ecome a better programmer is to study the programs of others, to encounter new methods of programming. With a disassembler, you can do this with very little trouble. Finally, this disassembler in particular can come in very handy with a specific problem encountered by 68ØØ users who are upgrading to the 68Ø9, but don't want to leave behind existing programs for the 68ØØ. With the disassembler, those programs can be disassembled, then reassembled under the 68Ø9, after making whatever changes need to be made.

Disassembling can be a difficult job, which can be simplified by a powerful disassembler. A simple disassembler may do nothing more than translate object code into absolute source code, without supplying labels so that the program can be easily moved about in memory. While adequate for small jobs, this type of disassembler will not suffice for larger jobs, where the existence of labels can be a decided advantage, if not a necessity. Another needed ability of a good disassembler is the ability to specify portions of a program as data instead of code which is meant to be executed by the computer. Such data areas occur as strings of ASCII characters for output or tables of byte or word data for reference by the ɔrogram. If a disassembler attempts to disassemble these data areas, incorrect code can be produced, or labels can be generated where they need not be. Finally, a useful addition to a disassembler might be the ability to specify that any reference to certain locations should be labeled not by an internally generated label, but by some standard symbol name. Often, a program will reference some location in an operating system which is normally known by a certain standard name. If the disassembler uses this standard name instead of a less meaningful label, a program becomes much easier to understand on first reading.

Obviously, a disassembler cannot perform all of these functions unaided. For example, a disassembler cannot know where all the data areas in a program are located, but it can make it easier to find them. Disassembling a program becomes a cooperative effort between you, the user, and the disassembler. A well designed disassembler can make this interaction relatively easy, maybe even fun.

DYNAMITE+ is such a disassembler. It has many abilities not usually found in a disassembler, and can make your disassembly jobs much more agreeable. All of the above mentioned functions are available, as well as many others. A partial list of DYNAMITE+'s features and abilities are:

-Runs under UniFLEX(TM) - also available for 6809 FLEX.

-Can disassemble both 6809 and 6800 object code.

-Produces source code which, when reassembled, is completely compatible with the original binary file.

-Automatically produces labels for any memory references within a program. External labels are defined by EQU statements grouped together at the start of the text file.

-Any label can be given a standard name, which is printed instead of the program-generated label name. For example, suppose the program starts execution at address $0000. Address $0000 can be given the standard label name 'begin', so that the end statement will appear as

```
                    end  begin
```

You can define standard names for labels yourself, or when disassembling code originally written for another operating system, you can use standard names for certain addresses in the operating system (i.e. WARMS for $CD03 in FLEX).

-Any block of code within a program can be treated as data. Such an area can be defined as any of four different varieties:

-ASCII data areas will produce fcc's, with strings automatically delimited, and ASCII control characters referred to by their standard names.

-Byte data areas will produce fcb's of hex data bytes.

-Label data areas will produce fdb's of labels references to the program. Many programs have data tables in them which are used for indirect addressing or for performing multi-direction jumps. Such tables are best disassembled as a series of labels.

-Word data areas will produce fdb's of hex data double bytes.

-You can specify the boundaries for data areas within a program in two ways. The data area addresses can be input interactively at the beginning of a disassembly, or can be read from a disk text file which has been set up previously.

-The output from DYNAMITE+ can be directed to the output device or to a disk text file. The text file created has all extra spaces deleted to save disk space.

-The output listing can be line numbered or paginated.

UniFLEX and FLEX are trademarks of Technical Systems Consultants, Inc.

-To help in finding the data areas in a program, the ASCII equivalent of the code being translated can be printed alongside the source line.

-All calls to the operating system via the 'sys' opcode can be automatically disassembled using the standard system names, with the proper number and types of parameters. For instance, a write command can be disassembled (using the standard label capability) as

```
                    sys  write,wrtbuf,$0200
```

This automatic generation of 'sys' calls can be turned off if desired.

-UniFLEX DYNAMITE+ can disassemble executable object code files, in the absolute, segmented text, or segmented no-text formats.

-A utility (convert.bin) is supplied to convert binary files from FLEX to UniFLEX format.

-A cross-reference utility (xref) is provided to generate label and/or opcode cross-reference listings. This utility is very useful during the disassembly (and commenting) process, but may also be used to aid in the documentation of programs for which the original source is available.

2. TERMS

2.1 FILES

DYNAMITE+ uses several files to perform disassembly jobs. Initial definitions for these files follow (more complete definitions are given in the instructions):

INPUT FILE is the executable binary object file which DYNAMITE+ is directed to disassemble.

OUTPUT FILE is the text file which is optionally created by DYNAMITE+ to hold the disassembled code for further editing.

COMMAND FILE is an optional text file which contains line specifying the address boundaries and types of data areas within a program. The command file may also contain option lines such as those normally sent via the UniFLEX command line.

LABEL FILE is a text file which sets up the equivalence between memory addresses and standard symbol names for those addresses. As many label files as desired may be specified in the command line or the command file.

SYSTEM NAME FILE is a text file which sets up the names of the various 'sys' functions, as well as the number and types of parameters for each function. DYNAMITE+ defaults to using the file '/lib/sysnames' for the system name file. You can specify another file to be used in place of this one, or you may choose to disable the disassembling of the 'swi3' opcode as 'sys'.

2.2 CONVENTIONS

As is standard in most 6809 program documentation, certain conventions will be used in declaring the format of user directions to DYNAMITE+. Angle brackets (<>) will be used to enclose essential elements of a statement. Square brackets ([]) will be used to enclose optional elements.

3. THE UniFLEX COMMAND LINE

The full syntax for calling DYNAMITE+ from UniFLEX is as follows:

```
     ++ dynamite <input file> [+<options>]
```

All files default to the working directory.

The <input file> is the file from which the object code is read. This is the only required command line parameter.

If no output file is specifically named (see the +o option below) then the name of the input file, with '.d' appended, will be used as the output file name, with the output going to the working directory. If the output file already exists, then DYNAMITE+ will ask if the existing file should be deleted:

```
     ++ dynamite /usr/me/prog
     Output file exists - Delete it (y/n)?
```

If n is typed, then DYNAMITE+ will stop and return to UniFLEX. If y is typed, the file will be deleted and a new text file created. If anything else is typed, the question will be reasked.

<options> specify a list of disassembly options on the command line. There may be as many <options> parameters as desired, but each must start with a plus sign ('+'). The option parameters do not need to be in any particular order.

There are three different kinds of options. The first are simple characters, which result in the enabling or disabling of certain operating characteristics. These flag options are:

+a	Print the ASCII equivalent of the code on the source line. These characters will be printed between the code bytes is hex and the label field. Illegal ASCII characters will be replaced by a period ('.') in this field.
+b	Prompt the user for the data area boundaries. This directs DYNAMITE+ to enter an interactive mode, requesting data area types and addresses from the user. The prompting format is detailed in the next section.

+d Do not create a text file on the disk. The output file will not be created even if a name is specified.

+e Expand disassembler output by adding a blank line before each labeled code line. This is a 'prettyprinting' option to improve the look of the output.

+g Generate extra lines of code bytes when more than four bytes are in an fcc, fcb, or fdb. Some data area disassembled code may normally require more than one line to list all of the code bytes in hex. If this option is enabled, then as many lines as are required to print all of the code will be output.

+l Do not send a disassembly listing to the output device.

+m Allow the 'Motorola' format for the zero offset indexed mode. In some assemblers, such as that from Motorola, the indexed operand 'Ø,r', r being a register name, will be disassembled as a 5 bit offset rather than the Ø offset mode. If this option is enabled, then any 5 bit offsets of Ø will be output as proper code. If the option is not enabled, then 5 bit offsets of Ø will be disassembled as illegal code and output as an fcb.

+n Generate line numbers in the output listing. The line number is a decimal number printed at the start of the source line.

+p Enable pagination. You will be prompted for a title, which will be printed at the top of every page. The title can be up to 32 characters long. Any excess over 32 characters will be ignored. If pagination is enabled, the listing output is formatted for 66 line pages, with a heading line, several blank lines, 57 code lines, and a form feed at the bottom of each page.

+t Disable the printing of the time and date in the page heading when pagination is enabled.

+z Disassemble the program for the 68ØØ, not the 68Ø9, which is the default.

The second kind of option consists of a single character followed by '=' and a filename. This type of option is used whenever a file other than the input file is required. Any filename option must be the last in any particular option parameter.

+c=file Specify the command file name. The command file is the text file holding the declarations for data area boundaries and for any standard options. Its format is described later. Only one command file may be specified in the command line.

+f=file Specify the name of the substitute system name file. This is the file used to give the names by which the various 'sys' functions are to be used. Without the +f option, the file '/lib/sysnames' will be used as the system name file. If a different set of sys function definitions is desired, then the +f option should be used with another filename. If the disassembling of 'sys' functions should be disabled, so that the 'swi3' instruction is disassembled instead, then the +f option should be used without a filename. Note that the option still requires the equals sign, i.e. '+f='. Only one system name file may be specified in the command line.

+o=file Specify the name for the output text file. If this option is not used, and disk output is enabled, then the output file will be given the name of the input file with '.d' appended, and written to the working directory. If the name is too long for '.d' to be appended, then the name is first truncated to 12 characters. Only one output file may be specified in the command line.

+s=file Specify the name for the label file. The s is for 'standard labels'. This is the file used to specify special names for certain program addresses, rather than the normal generated names. There may be as many +s options as desired.

The third kind of option consists of a single character followed by '=' and a value. This value may be either a decimal number or '$' followed by a hex number. The value must be between 0 and 255. This kind of option is used to set those options which may take a numeric value.

+x=val Set the maximum number of source lines to be printed per page when pagination is enabled. This does not include the heading lines. The default value is 57.

+y=val Set the value for the first ASCII character past $20 which is not printable. This value is referenced during the printing of fcc's and also when printing the ASCII information to the side of the source text when enabled by the +a option. The default value is $7F, the ASCII DEL. This option would be used, for an example, on a Hazeltine terminal. On the Hazeltine, all characters from $7E forward are unprintable, so any attempt to print the character for $7E will cause the terminal output to appear incorrect. Thus, the option '+y=$7E' should be used.

What follows are several examples of calls to DYNAMITE+, with explanations of the effect of each call:

```
++ dynamite proga +dan
```

Disassemble program 'proga', from the working directory. Do not create any disk file, but do list the disassembly. The listing should be line-numbered, and have the ASCII equivalent of the code printed in the code line. This option list is normally used to determine where in a program the data areas, especially the ASCII strings, are located, so that a more accurate disassembly can be made.

```
     ++ dynamite +dn +g objfile.b +b
```

Disassemble program 'objfile.b', read from the working directory. Do not create a disk file. The output listing should have line numbers, and fdb's, fcb's, and fcc's may run to extra lines. Before the disassembly starts, prompt the user for the addresses of data areas within the program.

```
     ++ dynamite /usr/me/testfile +o=srcfile +zl
```

Disassemble program '/usr/me/testfile' and create a text file 'srcfile' in the working directory for the source text. '/usr/me/testfile' should be disassembled as a 6800 program, not 6809. Do not print an output listing.

```
     ++ dynamite filea +pnc=filea.cmd +s=filea.lbl
```

Disassemble the program 'filea' from the working directory, and store the text output under the name 'filea.d' in the working directory. The output listing should be paginated and line numbered. Before disassembly begins, the command file 'filea.cmd' and the label file 'filea.lbl' should be processed.

4. USER INPUT BEFORE THE DISASSEMBLY

The options +b and +p require user input before the disassembly can begin. The format of that input will be described below.

4.1 THE TITLE INPUT

The first user input, if pagination is enabled, will be the title to be printed at the top of every page. DYNAMITE+ will print the prompt

```
     Title?
```

and wait for input from you. Now type in the title you want, then press RETURN. The title can be at most 32 characters long, so any excess typed in is simply ignored.

4.2 THE DATA AREA BOUNDS INPUT

The next user input is that required by the B option. DYNAMITE+ will enter an interactive mode enabling you to specify the addresses of data areas within the program to disassemble. DYNAMITE+ will print the prompt

```
     Data segment type: ascii, byte, label, word, reset, or proceed?
```

and wait for your response. Now type in the command for the type of the next data area. Only the first letter of the type need be typed, as that is all that is checked. The meaning of the commands in the type prompt are:

a This declares a data area to be ASCII, so that it will be disassembled as fcc's. ASCII control characters will be generated using their standard names, with EQU statements collected at the start of the program. Other unprintable ASCII characters will be printed as two digit hex bytes, preceded by a dollar sign, '$'. For the 6809, all of these elements will be generated together in fcc's, separated by commas, since this syntax is allowed by the TSC Assembler. For the 6800, though, this format is illegal, so unprintable ASCII characters will be generated on fcb lines within the fcc data area.

b This declares a data area to be byte data, so that it will be disassembled as fcb's. Each byte in the data area will be printed with two hex digits preceded with a dollar sign. A maximum of eight bytes will be generated in each fcb.

l This declares a data area to be label data, so that it will be disassembled as fdb's of labels. Each double byte in the data area will be printed as a label, and that label will be defined elsewhere in the output. This is useful for jump tables or tables of indirect addresses.

w This declares a data area to be word data, so that it will be disassembled as fdb's of constants. Each double byte in the data area will be printed as four hex digits preceded with a dollar sign. For both the label and word segments, a maximum of four words will be generated in each fdb.

r This does not specify a data area type, but instead directs DYNAMITE+ to erase all data area specifications input so far, and start over on the bounds input from scratch. Due to the structure of the data area bounds processing, it is not possible to back up a single step in the prompting sequence, so the only recourse if an error is made is to reset everything to the start and reenter all the data area bounds.

[illegible] This directs DYNAMITE+ to proceed from the prompting sequence to the next action. In other words, stop prompting for data area addresses and continue with the disassembly.

Any illegal type character will be ignored, and the prompt reissued.

The a, b, l, and w type characters require an pair of addresses to be input. These are the first and last address in the data area. DYNAMITE+ will first prompt with

```
     Starting address?
```

to which you should respond with the hex address of the start of the data area. Next, DYNAMITE+ prompts

```
     Ending address?
```

to which you should respond with the hex address of the end of the data area.

If the type character entered was not p, proceed, then the prompt will be reissued so that the next data area can be specified.

5. THE COMMAND FILE

While you can always specify the addresses of a program's data areas by invoking the +b option, this requires that you enter all of the data area bounds every time you disassemble the program. This is acceptable for small programs, but can be a substantial job for any large programs. The entry of the specifications may take several minutes each time, and a single error would require starting over from scratch. To make your job easier, DYNAMITE+ has the ability to read the data area specifications fro a text file, as well as from the terminal at run time. This is the mai ction of the command file. The command file can also be used to specify options so that they need not be declared in the UniFLEX command line for every disassembly.

The command file consists of a series of command lines. There are two types of command lines, the options line and the data boundary line. The options line consists of any number of option parameters, each beginning with '+', on a single line. The options are the same as those on the UniFLEX command line. The options command line is used to specify options to be used on every disassembly, and to save typing these options in at the UniFLEX command line. For instance, if a file consisted of 6800 code and a label file 'p6800.lbl' should always be referenced, then a command line with the single option parameter '+zs=p6800.lbl' would be used. All options are legal in the command file except for the options +b, +c, +d, +o, and +p. These are not allowed because, by the time DYNAMITE+ processes the command file, these options have already been processed.

The second type of command line is the data boundary line. This lir is used to declare the addresses of data boundaries, with the same effect the +b option. The format of the boundary declaration, though, is much more powerful here, and can be much more complicated. The general format of the specification line is

```
     <rpc> <type><range> [';'<type><range>...]
```

<rpc> is a line repeat count, so that an entire line may be repeated a number of times. <type> is the data area type character, which is one of the letters 'a', 'b', 'c', 'l', or 'w'. These letters have the same meaning as the data type code in the +b option (a=ascii, b=byte, l=label, w=word), with 'c' referring to a code area, which is used to specify a normal program area, used in conjunction with boundary lines using the repeat count. <range> specifies the range of addresses to be used, and is composed of two parts, the starting address and the ending address. In its simplest form, the <range> takes the form 'addr1-addr2' where addr1 and addr2 are hex numbers (without '$') specifying the address range. In its full form, the starting address may be either a hex number or null, in which case the starting address will be one more than the previous ending

address (which must be well-defined), and the ending address may take any of four forms:

null	- The data area is only 1 byte long (2 bytes for w or l data areas).
- addr2	- The data area extends up to the address specified by the hex number addr2.
-	- This is just a dash by itself. In this case, the ending address will be one less than the starting address of the next boundary declaration. For this to work, the next boundary declaration must exist, and have a starting address which is not null.
/length	- length is a decimal number, specifying the number of bytes in the data area. The ending address is computed as starting address + length - 1.

You may also include blank lines or lines beginning with an asterisk, '*', in the command file wherever desired. These lines are ignored by the ommand file processor, and can be used as comment lines or to format the c and file for easy viewing and editing.

As an example of a command file, consider the effect of the following command lines:

```
        * 68ØØ, use label file, no 'sys'
        +z +s=lblfile
        * Data areas within the code
        b a1Ø2/3 ; w/4
        l a233-a24Ø
        a -a27f
```

The first command line is a comment, and is ignored by DYNAMITE+. The second line is an option line, and directs DYNAMITE+ to disassemble the input file as 68ØØ code and to use the standard label file 'lblfile'. The third line is another comment. The fourth line consists of two boundary declarations. The first of these declares a byte area, starting at $A1Ø2 and going for 3 bytes, so that it extends from $A1Ø2 to $A1Ø4. The second declaration declares a four byte word area, with no specified starting a ress. The address used, then, will be one more than the previous ending a ress. The range specified is thus $A1Ø5 to $A1Ø8. The fifth line simply specifies a label area from $A233 to $A24Ø. The sixth line, with no specified starting address, will start at the next address, $A241. The sixth line thus specifies an ASCII area from $A241 to $A27F.

For another example, suppose there is a data table in a program, and you have determined that the table is 25 entries long, each entry consisting of a single ASCII character followed by 3 byte values and a label. For each entry, then, a boundary line, without addresses, would be 'a;b/3;l'. If the table starts at address $1ØØØ, then there are two major ways to specify the boundary declarations for the entire table. First:

```
        c Øfff
        25 a ; b /3 ; l
```

and second:

```
        a 1ØØØ ; b/3 ; l
        24 a ; b /3 ; l
```

ither way, to declare the boundaries, you must first specify the starting ddress of the table. In the first method, this is done with a line giving he address just before the table begins, as a code type area, and then eclaring the table itself with a single line. In the second method, the able is declared in two parts, with the first line actually giving the ddress of the first entry, and the second line giving the declarations for he remainder of the table.

6. THE LABEL FILE

DYNAMITE+ label files are used to supply an equivalence between an address in a program being disassembled and a name to be used in the source listing produced. In the absence of a label file, or of a label definition for a particular address, a label for an address will consist of the letter 's' followed by the 4 hex digits of the address. Thus, a reference to address $1234 in the program will produce the label 's1234'. To change this label to something which reflects the function of the label in the program, a standard name can be used.

Standard name label files are text files which contain a list of equate statements setting up this equivalence of name and address. The text file is read, and any line having the format

```
    label  equ  value
```

where 'label' is a legal assembler name (up to 8 characters long) and 'value' is either a decimal number number or '$' followed by a hex number, are processed to assign standard label 'label' to address 'value'. Any lines not having this strict 'equ' format are ignored, so the label file may contain comments, blank lines, or any assembly code whatever. Thus, normal lib or asmb text files may be used for label files.

As an example, suppose that you wish to refer to address $0000 as 'begin', $1000 as 'datal', $2000 as 'bssbeg', and $2FFF as 'codeend'. The label file might then look like

```
    begin    equ    0
    datal    equ    $1000
    bssbeg   equ    $2000
    codeend  equ    $2fff
```

Label files are useful when disassembling code which was produced under some operating systems which are referenced through fixed locations. For instance, under FLEX, calls to the operating system are made via addresses from $CD00 to $CD4E. To facilitate disassembling under certain operating systems, the UniFLEX DYNAMITE+ package includes a set of standard label files for each. 'dflex09.lbl' is the standard label file for use with FLEX 9.0, while 'dflex00.lbl' is the standard label file for use with FLEX 2.0, the 6800 version of FLEX. 'miniflex.lbl' is the standard label file for MINIFLEX(TM), a predecessor of the FLEX operating system on the 6800. 'swtbug.lbl' is the standard label file for SWTBUG(TM), the system

----

MINIFLEX is a trademark of Technical Systems Consultants, Inc.
SWTBUG and S-BUG are trademarks of Southwest Technical Products Corp.

monitor in SWTPc 6800 computer systems. 'sbug.lbl' is the standard label file for S-BUG(TM), the system monitor in SWTPc 6809 computer systems.

7. THE SYSTEM NAME FILE

The system name file is used by DYNAMITE+ to supply the proper names to the function bytes used in UniFLEX 'sys' calls. For instance, the UniFLEX call to end a program, 'sys term', is actually a 'swi3' followed by the byte value $05 or 5. To properly disassemble this code as 'sys term', DYNAMITE+ must know that, after the 'swi3', 5 means 'term'. This is the first function of the system name file. The second function concerns the use of parameters in UniFLEX 'sys' calls. For instance, the call to 'write' requires two parameters, the first being the address of a buffer, the second the count of bytes to output. Thus, in the terminology of DYNAMITE+ data areas, the first parameter is a label, while the second is a [illegible] d.

The system name file, much as for the label file, is a text file which contains a number of 'equ' lines. As might be expected, the equivalence between name and function value is set up by a 'name equ value' line. To set up the number of types of parameters, the comment field of the 'equ' line is reserved. In this comment field, there should be a series of letters, w and l, one letter for each parameter. The particular letter to be used depends on the parameter type, i.e. w for word parameter, l for label parameter. The order of letters in the comment field follows the order of parameters in the 'sys' call. If a 'sys' function requires no parameters, then the comment field of that particular line in the system name file should be blank.

As an example, consider the lines found for the 'term' and 'write' functions:

```
term     equ   5
write    equ   13 lw
```

The system name file defaults to the file '/lib/sysnames'. If you wish to add or change a system function, then this file should be edited. If you wish to replace this file by another for a single disassembly, then you should use the '+f' option. For more examples of the format of the system name file, just list '/lib/sysnames'.

8. DISASSEMBLY PARTICULARS

There are several particular attributes of DYNAMITE+ which are useful to keep in mind when disassembling.

DYNAMITE+ is a three pass disassembler. On the first pass, the object code is searched for memory references, which are to be translated as labels. As well as the obvious memory references, such as the direct or extended addressing modes, DYNAMITE+ regards the immediate addressing mode using the x, y, u, or s registers as memory references. Thus, a ldx # (load x immediate) encountered within the object code will generate a label for the immediate value. This ensures that the vast majority of labels will be picked up. Unfortunately, this also means that an immediate load of those registers with data will also generate labels, when this was not what was desired. For instance, in some code, x is loaded with a constant to control a loop count, as in ldx #256. This will be disassembled as ldx #s0100, where s0100 is the label assigned a reference to address 256, $100 in hex. Thus, there will usually be some labels generated which are actually data values instead of memory locations. An exception to the generation of labels from immediate addressing is in the immediate mode with accd, as in addd #1. Since accd is almost exclusively used for data manipulation instead of addressing, immediate references are always disassembled as data. While this may, very rarely, miss a label which should have been generated, it will prevent the generation of many unneeded labels.

The second pass of DYNAMITE+ does not actually read the code from the [illegible]k. Instead, this pass flags all labels found in pass 1 as internal or external. Internal labels are those whose value is the address of code within the program, while external labels are those not occurring within the program as code addresses. If the program being disassembled is a segmented file, then pass 2 will also determine which of the external labels are part of the program's bss area. This pass is required so that all external labels can be defined using equate statements at the beginning of the disassembly listing, and all bss labels can be grouped together at the end of the listing.

The third pass of DYNAMITE+ creates the actual disassembly code, both the disk file and the output listing. The output has six main parts. First, any info statements found in the object file are output. Second, all 'sys' function codes which are found in the code are grouped together and defined using equates. Third, all standard labels, whose names where read from the label files, where referenced within the code, and are external to the program, are grouped together and defined using equates. Fourth, all ASCII equates are grouped together. If any ASCII data areas where defined, and control characters were encountered within these areas, t[illegible]n the standard names for the control characters are used in the fcc statements. These standard control character names are defined in this section of the output. Fifth, all external labels without standard names are defined by equates. This output section should be studied carefully, because any generated labels which should really be data will normally show up in this equate segment. If a label is defined which does not appear to be a true legal memory reference, check in the actual disassembly for all references to the label and determine for yourself if it should actually be a data value. The sixth output section is the main section, the actual disassembly. This will be a series of assembler-compatible source lines. If any lines are referenced in the code, then the internally generated label will be placed in the label field of the source line. All label references appear either as a standard name, if the value was found in a label file, or as an 's' followed by a four digit hex number, whose value is the reference address. Thus, references to address $1234 will generate a label of s1234.

The code produced by DYNAMITE+ should always reassemble into code which is identical with the original source file. Some addressing modes, though, can generate either 8 bit references or 16 bit references, especially on the 6809. To ensure that such references will assemble to code with the same reference bit length, DYNAMITE+ will generate forced addressing where required. This can be done only for the 6809, and is compatible with the forced addressing method expected by TSC's assembler. Thus, all direct references, as well as all 8 bit pcr relative addressing will have the 'force 8 bit addressing' character, the less than sign, '<', before the label. For example, a direct load into acca from address $55 will be disassembled as lda <s0055. Also, all 16 bit references for which 8 bits suffice will have the 'force 16 bit addressing' character, the greater than sign, '>', before the label. An extended store of IX to address $33 will be disassembled as stx >s0033.

DYNAMITE+ for UniFLEX can properly disassemble three types of object files, absolute, segmented text, and segmented no-text. If an absolute file is disassembled, then code block addresses will be addressed properly with 'org' statements, as is normal. If a segmented file is disassembled, then the text segment is assumed to start at address $0000, any data segment is assumed to start the first 4K boundary past the text segment, and any bss segment is assumed to start directly following the data segment.

9. A DISASSEMBLY EXAMPLE

To help you understand the procedure for disassembling a program, a fairly routine example which uses many of DYNAMITE+'s abilities will be discussed. The program to be disassembled is a simple filter program, which accepts input from the standard input channel, filters out all $0C characters (form feeds), and outputs to the standard output channel.

The process of disassembling a program with DYNAMITE+ is usually a multi-step procedure. First, you need to determine where the data areas occur within a program. To do this, disassemble the program, suppressing disk output and generating the ASCII equivalent of code in the listing. By studying this listing, all of the ASCII strings can be easily found, and some idea of the location of other types of data areas can be formed. Once an idea is formed of where the data areas occur, the program can be disassembled again, this time with the +d and +b options, so disk output is again suppressed, but this time prompting you for the data areas discovered in the first disassembly. This disassembly will be much closer to the final version, and any data areas missed the first time can usually be found now. Finally, you can create a command file with the final version of the data boundaries, and a label file to give meaningful names to the various program labels, and disassemble the program once more, this time using the command file and creating a disk text file.

The example disassembly will follow this method. All of the files referred to below exist on the disk on which DYNAMITE+ was shipped, so follow along.

The first step, as outlined above, is to disassemble the program with ASCII generated, to find the data areas. First, change the directory to '/dynamite.demo'. Then, enter the following command (the file being disassembled is named 'fl') :

```
++ dynamite fl +dan
```

This will disassemble the program without disk output, and add line numbers and ASCII code equivalents to the listing. The listing produced can be found on the next page.

```
                                      * disassembly by dynamite+ of fl

                                               info    Filter to block ^L's
                                               info    (ie, after assembly)
                                               info    from clearing screen.
                                               info    by Scott Schaeferle
                                               info    June 21,1981

                                      * system name equates

                              0005    term     equ     5
                              000C    read     equ     12
                              000D    write    equ     13

                                      * external label equates

                              0040    s0040    equ     $0040

0000 CC   0000                ...     s0000    ldd     #$0000
0003 113F 0C                  .?.              sys     read,s0040,$0001
000A 1025 0021                .%.!             lbcs    s002f
000E 1083 0000                ....             cmpd    #$0000
0012 26   03                  &.               bne     s0017
0014 113F 05                  .?.              sys     term
0017 B6   0040                ..@     s0017    lda     >s0040
001A 81   0C                  ..               cmpa    #$0c
001C 26   05                  &.               bne     s0023
001E 86   07                  ..               lda     #7
0020 B7   0040                ..@              sta     >s0040
0023 CC   0001                ...     s0023    ldd     #$0001
0026 113F 0D                  .?.              sys     write,s0040,$0001
002D 24   D1                  $.               bcc     s0000
002F 34   06                  4.      s002f    pshs    a,b
0031 CC   0001                ...              ldd     #$0001
0034 113F 0D                  .?.              sys     write,s0041,$0006
003B 35   06                  5.               puls    a,b
003D 113F 05                  .?.              sys     term

0041                                           org     $0041

0041 45                       E       s0041    fcb     $45
0042 72                       r                fcb     $72
0043 72                       r                fcb     $72
0044 6F   72                  or               clr     -14,s
0046 21                       !                fcb     $21

                               0000            end     s0000
```

By scanning the ASCII equivalent area of the listing, it is obvious that the code from $0041 to $0046 is an ASCII string. The remainder of the program seems to have disassembled properly, so all of the data areas have probably been found.

Now, you can proceed with the second disassembly, this time entering the data boundaries to DYNAMITE+. Since there is little chance that a third disassembly will be required, the output can be written to disk at this time. The command sent to UniFLEX is now:

```
++ dynamite fl +bn
```

This command will have DYNAMITE+ request the data area addresses found earlier from you via the keyboard. Also, the output listing will be line numbered. Note that the ASCII equivalent area of the listing is turned off, since it is no longer required.

After DYNAMITE+ has begun execution, the prompt for the entry of the first data area type will be sent. You want to enter the area from $0041 to $0046 as ASCII data, so type an 'a', then RETURN. DYNAMITE+ will ask for the starting address of the data area. Type the characters '41', then RETURN. The dollar sign is not required, since hex is assumed. Next, DYNAMITE+ will ask for the ending address of the data area. To this prompt, answer '46', then RETURN.

DYNAMITE+ reissues the prompt for data area type. If you made an error in the data area specification, type 'r' to reset, and restart the entry. Otherwise, you are finished with the data area boundaries entry. Type a 'p' to indicate that DYNAMITE+ should proceed to the rest of the disassembly process.

DYNAMITE+ will finish up its initial processing before beginning its three passes. At the start of pass one, a RETURN, LINE FEED is sent, and disassembly begins. After pass three is complete, the output listing produced should be the same as the listing on the next page.

```
                   * disassembly by dynamite+ of fl

                           info   Filter to block ^L's
                           info   (ie, after assembly)
                           info   from clearing screen.
                           info   by Scott Schaeferle
                           info   June 21,1981

                   * system name equates

             0005  term    equ    5
             000C  read    equ    12
             000D  write   equ    13

                   * external label equates

             0040  s0040   equ    $0040

0000 CC   0000     s0000   ldd    #$0000
0003 113F 0C               sys    read,s0040,$0001
000A 1025 0021             lbcs   s002f
000E 1083 0000             cmpd   #$0000
0012 26   03               bne    s0017
0014 113F 05               sys    term
0017 B6   0040     s0017   lda    >s0040
001A 81   0C               cmpa   #$0c
001C 26   05               bne    s0023
001E 86   07               lda    #7
0020 B7   0040             sta    >s0040
0023 CC   0001     s0023   ldd    #$0001
0026 113F 0D               sys    write,s0040,$0001
002D 24   D1               bcc    s0000
002F 34   06       s002f   pshs   a,b
0031 CC   0001             ldd    #$0001
0034 113F 0D               sys    write,s0041,$0006
003B 35   06               puls   a,b
003D 113F 05               sys    term

0041                       org    $0041

0041 45 72 72 6F   s0041   fcc    "Error!"

             0000          end    s0000
```

This latest disassembly is the final one required. All of the data areas are formatted correctly, and the output on disk is ready for any editing you may wish to do, under the name 'fl.d'.

The DYNAMITE+ demo files, in '/dynamite.demo', include some additional files for you to use in the disassembly of the example filter program. The file 'fl.cmd' is a command file, which directs DYNAMITE+ to disassemble the ASCII string properly, without the +b option. The command file also includes an option line which directs DYNAMITE+ to print a blank line before each labeled line. The file 'fl.lbl' is a label file, which gives meaningful names to the various labels in 'fl'. For instance, address Ø is labeled 'start', and the ASCII string at $ØØ41 is labeled 'errmsg'. To use these two files, and generate disk output, the following line can be used:

```
++ dynamite fl +n +c=fl.cmd +s=fl.lbl
```

This can also be done using the included shell script file, 'd'. Simply enter 'd' as a command to UniFLEX. As a further example, suppos p ination is desired, but you wish to run the program in background. Since pagination requires the input, at run time, of a title from the standard input channel, the input must be redirected to come from a disk file. The command to accomplish this is found in the shell script file, 'dp':

```
++ dynamite fl <title +dnp +c=fl.cmd +s=fl.lbl &
```

The file 'title' is simply a text file holding the title to be used for pagination.

This completes this example of using DYNAMITE+. Try out the various commands given here, and try modifying them to see what happens. For instance, the option which specifies the label file to use, '+s=fl.lbl', need not go in the command line. Instead, put it in the command file, so that the UniFLEX command becomes shorter by this one option.

1 DYNAMITE+ ERROR MESSAGES

There are two varieties of errors detected by DYNAMITE+. The first of these is disk errors, as returned by FMS. The second is non-disk errors, caused by some form of illegal user input detected by DYNAMITE+.

Disk errors are at least two lines long. The first line gives the particular file involved. The line has the form

```
Type file error
```

where 'Type' is one of Input, Output, Command, or Label. The last line of the error is a description of the particular disk error encountered (open, close, seek, etc.).

Non-disk errors print error messages found internal to DYNAMITE+. These errors give a description of the condition causing the program to be aborted or interrupted. All non-disk errors are detected in the option processing portion of DYNAMITE+, before the three passes begin.

Both types of errors may occur while the command file is being processed. If an error occurs in the command file, then another line is included in the error output. This line has the form

    Command file line #nnn

where nnn is the line number within the command file on which the error occurred. This line is the second line in a disk error, the first in a non-disk error.

Disk errors always cause DYNAMITE+ to stop execution. Control is immediately returned to UniFLEX, with a returned nonzero status. Non-disk errors will normally result in the program being halted also. If the error is caused by illegal input during the user input portion of DYNAMITE+, though, the error message is not usually a terminating one, but just a request for reinput of the unaccepted data.

The errors detected by DYNAMITE+ are:

Illegal option switch or syntax
: Some unrecognizable character sequence was found in the UniFLEX command line or in an option line in the command file.

Start > End, Reenter both addresses
: When entering a pair of addresses, either data area boundaries or the segment to disassemble, the starting address was greater than the ending address. The input is ignored, and you will be reprompted for both addresses.

Illegal entry, re-enter
: An illegal hex number was typed. Retype the input.

Command syntax error
: A line was found in the command file which could not be recognized as a legal command line.

Illegal segment address specification
: The starting-ending address pair in a data area address command line was illegal in some way.

Word or label segment has odd length
: The addresses specified for a data area of type word or label was an odd number of bytes long. Double byte data areas must have an even length. If this error occurred during user prompting of data areas, the input is ignored. In the command file, this error halts DYNAMITE+.

ata segments overlap
: The latest data area occupied some of the same addresses as a previously specified data area. Data areas must be mutually exclusive, in that any address can be in at most one data area. If this error occurred during user prompting of data areas, the input is ignored. In the command file, this error halts DYNAMITE+.

nsufficient memory available
: DYNAMITE+ has insufficient memory to build all of the required tables.

o input file named
: No input file was specified on the UniFLEX command line.

llegal format in input file
: The input file specified was not a proper binary object file. Binary object files, under UniFLEX, are expected to begin with a 24 byte header, the first two bytes of which should be $Ø2, $11 or $Ø2, $12.

[illegible] specified more than once
: A certain type of file was named more than once. The label file is the only file which may be multiply defined. All others can occur only once.

rror during read file open
: An error was encountered while attempting to open one of the files which DYNAMITE+ reads (input, command, label, or system name). Probably due to a non-existant file.

rror during output file open
: An error was encountered while attempting to open the output file.

Error during closing
: An error was encountered while attempting to close one of the open files.

Error during seek
: An error was encountered while attempting to seek on the input file.

Error during read
: An error was encountered while attempting to read from an open read file.

Error during write
: An error was encountered while attempting to write to the output file.

Error during std output
: An error was encountered while attempting to write to the standard output channel.

Error on std input channel
: An error was encountered while attempting to read from the standard input channel.

Unexpected eof on input channel
    End of file was encountered during a read from the standard input channel.

11. Disassembling FLEX programs

While UniFLEX DYNAMITE+ is primarily intended as a tool for disassembling UniFLEX programs, it can also be used to disassemble programs originating on the FLEX operating system. One reason for doing this would be to duplicate the function of a FLEX utility in the UniFLEX environment, without having to write the program from scratch.

Before you can disassemble a FLEX program, you must first get a copy of it into a UniFLEX file. This can be done with the standard utility 'flex'. Just don't forget to specify the 'raw mode' (+r) when you run 'flex', so that your file is copied byte-for-byte, without tab expansion. Otherwise you will find that your program is so much garbage.

Once the FLEX binary file is copied into a UniFLEX file, you must then use the utility 'convert.bin' (supplied as a part of DYNAMITE+) to convert the FLEX binary format into the UniFLEX binary format required by the disassembler. The syntax for using this utility is:

```
++ convert.bin <FLEXfile> <UniFLEXfile>
```

After this utility is finished, you can use the DYNAMITE+ disassembler as usual. You may want to disable 'sys' pseudo-op generation (+f=) and call up a FLEX label file with +s=/lib/dflex09.lbl or similar option.

12. DYNAMITE+ Cross Reference utility (XREF)

XREF, a cross reference generator, takes an assembly language program and generates from it a listing giving for each label in the program the numbers of all lines on which that label occurs. XREF can optionally produce a cross reference for all opcodes in the program as well. If the program being processed contains LIB statements which call in other files for assembly, then XREF can process those files also, or optionally, suppress processing. XREF can be used on 6800 or 6809 assembly language programs, and will properly ignore the operand field in lines with opcodes which do not have operands, such as CLRA or NEGB.

XREF is supplied as a part of DYNAMITE+, since a cross reference generator is a valuable tool in the disassembly process. However, XREF can also be used with assembly-language programs for which the original source listing is available. The cross reference listing can be helpful in documenting such programs, and assisting in their maintenance.

The syntax of the command line to execute XREF is as follows:

```
     ++ xref <input file> [+<options>]
```

<input file> is the name of the file to be processed, and defaults to the current directory. The file should be in the format required by the TSC Mnemonic Assemblers - 'asmb' or 'relasmb'. +<options> is a list of letters specifying the options to be activated, and is optional if none of the options is required. The six legal single character options, with their meanings, are:

+l	Do not process LIB statements. Under normal conditions, LIB statements will cause source input to switch to the file named in the LIB operand field. If the l option is used, then this action is suppressed.
+n	Do not produce a cross reference of names. This might be used if a cross reference of opcodes only is desired.
+o	Do produce a cross reference of opcodes. Normally, the opcode cross reference is not generated.
+p	Paginate the output listing. If option p is used, you will be prompted for a title. The output will then be formatted (by default) for 66 line pages, with form feeds issued at the bottom of each page.
+s	Use short labels. For compatability with asmb, the absolute assembler, labels can be limited to only 6 significant characters. XREF defaults to 8 character labels.
+t	Disable printing of time and date in page heading.

There is also a series of options which require a value to be specified. For these, the option letter must be followed by '=' and the value, as either a decimal number or '$' followed by a hex number. The value cannot be greater than 255. Any option requiring a value must be the last option in a particular option parameter. If another one is needed, use another '+'. The value options are:

+d=val	set the maximum depth for lib nesting. The default value is 12. This value may not exceed 2Ø.
+m=val	set the maximum number of lines per page. The default value is 57, and does not include the heading lines.
+w=val	set the page width. The default value is 8Ø.

If no options are specified, then a label name cross reference will be produced, and LIB statements will be processed. Note the opposite effect of the n and o options. Option n will suppress the name list, while option o will enable the opcode list. The listing produced is sent to the standard output channel, but may be redirected to any file or device.

The name cross reference includes all names found within the program in either the label or operand fields. The opcode cross reference includes all names which are found within the opcode field, as well as all macros defined within the file. The entry for each label or opcode consists of four parts:

cc 1-8	The name of the label or opcode is placed in this first portion of each entry.
col 10	This column is normally blank. If the label is the name of a macro, then an 'm' will be found here. If, in the opcode list, the name is not that of a standard opcode or pseudo-op, and not the name of a macro, then a 'u' will be found here.
cc 12-17	For all labels in the name list, or for macro names in the opcode list, this portion of the entry consists of the number of the line on which the label is defined by placement in the label field. If the label is never defined in this way, then the field will be blank. If the label is found more than once in the label field, then it is multiply defined, and an '*' will be found in column 15. XREF will not flag as multiply defined those labels which are redefined by the SET pseudo-op.
cc 19+	This final portion of each entry consists of the numbers of all lines, other than the initial definition, on which the label or opcode is found. If a label occurs more than once on a line, then that line number will appear once for each usage on the line. If a label is defined but never used, then this field will be blank.

13. UPDATE POLICY

The DYNAMITE+ (UniFLEX version) license fee includes one year of maintenance service. Your name, address, and serial number are on our files, and you will be notified by mail whenever an updated version of DYNAMITE+ is available. The new version will then be supplied to you if you return your original diskette, postpaid, to CSC at the address on the cover of this manual.

You will be billed for additional maintenance (optional) when your first year's maintenance is about to expire. The maintenance fee is currently $60.00 per year.

This update policy applies only to corrections and minor enhancements of DYNAMITE+. Completely new versions, such as for a different computer or operating system, are considered separate products and must be purchased outright.

This manual was prepared using the TSC 6809 Text Processor and printed by a DIABLO model 1640 daisy-wheel printer.