Chapter 1
MOTOROLA DSP ASSEMBLER

1.1 INTRODUCTION

The Motorola DSP Assemblers are programs that process assembly language source statements written for Motorola's family of digital signal processors. The Assembler translates these source statements into object programs compatible with other Motorola DSP software and hardware products.

1.2 ASSEMBLY LANGUAGE

The assembly language provides mnemonic operation codes for all machine instructions in the digital signal processor instruction set. In addition, the assembly language contains mnemonic directives which specify auxiliary actions to be performed by the Assembler. These directives are not always translated into machine language. The assembly language enables the programmer to define and use macro instructions which replace a single statement with a predefined sequence of statements found in the macro definition. Conditional assembly also is supported.

1.3 INSTALLING THE ASSEMBLER

The Assembler is distributed on various media and in different formats depending on the host environment. See Appendix G, Host-dependent Information, for details on installing and operating the Assembler on your particular machine.

1.4 RUNNING THE ASSEMBLER

The general format of the command line to invoke the Assembler is:

DSPASM [options] <filenames>

where:

DSPASM

The name of the Motorola DSP Assembler program appropriate for the target processor (see Appendix F, Device-dependent Information). For exam-

ple, for the Motorola DSP56000 processor the name of the Assembler executable is **ASM56000**.

[options]

Any of the following command line options. These can be in any order, but must precede the list of source filenames. Some options can be given more than once; the individual descriptions indicate which options may be specified multiple times. Option letters can be in either upper or lower case.

Command options that are used regularly may be placed in the environment variable **DSPASMOPT**. If the variable is found in the environment the Assembler adds the associated text to the existing command line prior to processing any options. See your host documentation for instructions on how to define environment variables.

Option arguments may immediately follow the option letter or may be separated from the option letter by blanks or tabs. However, an ambiguity arises if an option takes an optional argument. Consider the following command line:

ASM56000 -B MAIN IO

In this example it is not clear whether the file MAIN is a source file or is meant to be an argument to the **-B** option. If the ambiguity is not resolved the Assembler will assume that MAIN is a source file and attempt to open it for reading. This may not be what the programmer intended.

There are several ways to avoid this ambiguity. If MAIN is supposed to be an argument to the **-B** option it can be placed immediately after the option letter:

ASM56000 -BMAIN IO

If there are other options on the command line besides those that take optional arguments the other options can be placed between the ambiguous option and the list of source file names:

ASM56000 -B MAIN **-V** IO

An alternative is to use two successive hyphens to indicate the end of the option list:

ASM56000 -B -- MAIN IO

In this latter case the Assembler interprets MAIN as a source file name and uses the default naming conventions for the **-B** option.

1.5 ASSEMBLER OPTIONS

-A

Indicates that the Assembler should run in absolute mode, generating an absolute object file when the **-B** command line option is given. By default the Assembler produces a relocatable object file that is subsequently processed by the Motorola DSP linker. See Chapter 4, Software Project Management, for more information on Assembler modes.

-B[<objfil>]

This option specifies that an object file is to be created for Assembler output. <objfil> can be any legal operating system filename, including an optional pathname. A hyphen also may be used as an argument to indicate that the object file should be sent to the standard output.

The type of object file produced depends on the Assembler operation mode. If the **-A** option is supplied on the command line, the Assembler operates in absolute mode and generates an absolute object (.CLD) file. If there is no **-A** option on the command line, the Assembler operates in relative mode and creates a relocatable object (.CLN) file.

If a pathname is not specified, the file will be created in the current directory. If no filename is specified, the Assembler will use the basename (filename without extension) of the first filename encountered in the source input file list and append the appropriate file type (.CLN or .CLD) to the basename. If the **-B** option is not specified, then the Assembler will not generate an object file. The **-B** option should be specified only once. **If the file named in the -B option already exists, it will be overwritten.**

Example: **ASM56000 -B**filter main.asm fft.asm fio.asm

In this example, the files MAIN.ASM, FFT.ASM, and FIO.ASM are assembled together to produce the relocatable object file FILTER.CLN.

-D<symbol> <string>

This is equivalent to a source statement of the form:

DEFINE <symbol> <string>

<string> must be preceded by a blank and should be enclosed in single quotes if it contains any embedded blanks. Note that if single quotes are used they must be passed to the Assembler intact, e.g. some host command interpreters will strip single quotes from around arguments. The

-D<symbol> <string> sequence can be repeated as often as desired. See the **DEFINE** directive (Chapter 6) for more information.

Example: **ASM96000 -D** POINTS 16 prog.asm

All occurrences of the symbol POINTS in the program PROG.ASM will be replaced by the string '16'.

-EA <errfil>
-EW <errfil>

These options allow the standard error output file to be reassigned on hosts that do not support error output redirection from the command line. <errfil> must be present as an argument, but can be any legal operating system filename, including an optional pathname.

The **-EA** option causes the standard error stream to be written to <errfil>; if <errfil> exists, the output stream is appended to the end of the file. The **-EW** option also writes the standard error stream to <errfil>; if <errfil> exists it is rewound (truncated to zero), and the output stream is written from the beginning of the file. **Note that there must be white space separating either option from the filename argument.**

Example: **ASM96000 -EW**errors prog.asm

Redirect the standard error output to the file ERRORS. If the file already exists, it will be overwritten.

-F<argfil>

Indicates that the Assembler should read command line input from <argfil>. <argfil> can be any legal operating system filename, including an optional pathname. <argfil> is a text file containing further options, arguments, and filenames to be passed to the Assembler. The arguments in the file need be separated only by some form of white space (blank, tab, newline). A semicolon (**;**) on a line following white space makes the rest of the line a comment.

The **-F** option was introduced to circumvent the problem of limited line lengths in some host system command interpreters. It may be used as often as desired, including within the argument file itself. Command options may also be supplied using the **DSPASMOPT** environment variable. See the discussion of **DSPASMOPT** under [options] at the beginning of this section.

Example: **ASM96000 -F**opts.cmd

Invoke the Assembler and take command line options and source filenames from the command file OPTS.CMD.

-G

Send source file line number information to the object file. This option is valid only in conjunction with the **-B** command line option. The generated line number information can be used by debuggers to provide source-level debugging.

Example: **ASM56000 -B -G** myprog.asm

Assemble the file MYPROG.ASM and send source file line number information to the resulting object file MYPROG.CLN.

-I<pathname>

When the Assembler encounters **INCLUDE** files, the current directory (or the directory specified in the **INCLUDE** directive) is first searched for the file. If it is not found and the **-I** option is specified, the Assembler prefixes the filename (and optional pathname) specified in the **INCLUDE** directive with <pathname> and searches the newly formed directory pathname for the file.

The pathname must be a legal operating system pathname. The **-I** option may be repeated as many times as desired. The directories will be searched in the order specified on the command line.

Example: **ASM56000 -I**\project\ testprog

This example uses IBM PC pathname conventions, and would cause the Assembler to prefix any **INCLUDE** files not found in the current directory with the \project\ pathname.

-L<lstfil>

This option specifies that a listing file is to be created for Assembler output. <lstfil> can be any legal operating system filename, including an optional pathname. A hyphen also may be used as an argument to indicate that the listing file should be sent to the standard output, although the listing file is routed to standard output by default.

If a pathname is not specified, the file will be created in the current directory. If no filename is specified, the Assembler will use the basename (filename without extension) of the first filename encountered in the source input file list and append .LST to the basename. If the **-L** option is not specified, then the Assembler will route listing output to the standard output (usually the console or terminal screen) by default. The **-L** option should be specified

only once. **If the file named in the -L option already exists, it will be overwritten.**

Example: **ASM96000 -L** filter.asm gauss.asm

In this example, the files FILTER.ASM and GAUSS.ASM are assembled together to produce a listing file. Because no filename was given with the **-L** option, the output file will be named using the basename of the first source file, in this case FILTER. The listing file will be called FILTER.LST.

-M<pathname>

This is equivalent to a source statement of the form:

MACLIB <pathname>

The pathname must be a legal operating system pathname. The **-M** option may be repeated as many times as desired. The directories will be searched in the order specified on the command line. See the **MACLIB** directive (Chapter 6) for more information.

Example: **ASM56000 -M** fftlib/ trans.asm

This example uses UNIX pathname conventions, and would cause the Assembler to look in the fftlib subdirectory of the current directory for a file with the name of the currently invoked macro found in the source file.

-O<opt>[,<opt>,...,<opt>]

This is equivalent to a source statement of the form:

OPT <opt>[,<opt>,...,<opt>]

<opt> can be any of the options that are available with the **OPT** directive (see Chapter 6). If multiple options are specified, they must be separated by commas. The **-O**<opt> sequence can be repeated for as many options as desired.

Example: **ASM96000 -OS,CRE** myprog.asm

This will activate the symbol table and cross reference listing options.

-P<proc>

Run the Assembler with the specified processor revision level enhancements. This is for backward compatibility so that the Assembler will flag new constructions as illegal. <proc> can be any of the processor identifiers given

below. Note that if this option is not used the Assembler runs with all latest revision level enhancements on by default.

Processor	Identifier
DSP56001 Rev. C	56001c
DSP56002	56002
DSP56004	56004
DSP56166	56166
DSP96001 Rev. B	96001b
DSP96002	96002

Example: **ASM56000 -P**56001c myprog.asm

Assemble MYPROG.ASM with the DSP56000 Revision C enhancements.

-Q

On some hosts the Assembler displays a banner on the console when invoked. This option inhibits the banner display. It has no effect on hosts where the signon banner is not displayed by default.

Example: **ASM56000 -Q** myprog.asm

Assemble the file MYPROG.ASM but do not display the signon banner on the console.

-R<rev>

Run the Assembler without the specified processor revision level enhancements. This is for backward compatibility so that the Assembler will flag new constructions as illegal. <rev> can be any of the revision specifiers given below, but must be appropriate for the target processor.

This option is superseded by the **-P** option.

Processor	Revision
DSP56001 Rev. C	C
DSP56002	2
DSP56004	4
DSP56166	6
DSP96000 Rev. B	B
DSP96001	1

Example: **ASM56000 -RC** myprog.asm

Assemble MYPROG.ASM without the DSP56000 Revision C enhancements.

-V

This option causes the Assembler to report assembly progress (beginning of passes, opening and closing of input files) to the standard error output stream. This is useful to insure that assembly is proceeding normally.

Example: **ASM56000 -V** myprog.asm

Assemble the file MYPROG.ASM and send progress lines to the standard error output.

-Z

This option causes the Assembler to strip symbol information from the absolute load file. Normally symbol information is retained in the object file for symbolic reference purposes. Note that this option is valid only when the Assembler is in absolute mode via the **-A** command line option and when an object file is created (**-B** option).

Example: **ASM56000 -A -B -Z** myprog.asm

Assemble the file MYPROG.ASM in absolute mode and strip symbol information from the load file created as output.

<filenames>

A list of operating system compatible filenames (including optional pathnames). If no extension is supplied for a given file, the Assembler first will attempt to open the file using the filename as supplied. If that is not successful the Assembler appends .ASM to the filename and attempts to open the file again. If no pathname is specified for a given file, the Assembler will look for that file in the current directory. The list of files will be processed sequentially in the order given and all files will be used to generate the object file and listing.

The Assembler will redirect the output listing to the standard output if the output listing is not suppressed with the **IL** option, or if it is not redirected via the **-L** command line option described above. The standard output generally goes to the console or terminal screen by default, but can be diverted to a file or to a printer by using the I/O redirection facilities of the host operating system, if available. Error messages will always appear on the standard output, regardless of any option settings. Note that some options (**-B**, **-L**) allow a hyphen as an optional argument which indicates that the corresponding output should be sent to the standard output stream. Unpredictable results may occur if, for example, the object file is explicitly routed to standard output while the listing file is allowed to default to the same output stream.

For more details on Assembler operation in a particular machine environment see Appendix G, Host-dependent Information.

1.6 ASSEMBLER PROCESSING

The Motorola DSP Assembler is a two-pass Assembler. During the first pass the source program is read to build the symbol and macro tables. During the second pass the object file is generated (assembled) with reference to the tables created during pass one. It is also during the second pass that the source program listing is produced.

Each source statement is processed completely before the next source statement is read. As each line is read in, any translations specified by the **DEFINE** directive are applied. Each statement is then processed, and the Assembler examines the label, operation code, operand, and data transfer fields. The macro definition table is scanned for a match with the operation code. If there is no match, the operation code and directive tables are scanned for a match with a known opcode.

Any errors detected by the Assembler are displayed before the actual line containing the error is printed. Errors and warnings are accumulated, and a total number of errors and warnings is printed at the end of the source listing. If no source listing is produced, error messages are still displayed to indicate that the assembly process did not proceed normally. The number of errors is returned as an exit status when the Assembler returns control to the host operating system.

1.7 DEFINITION OF TERMS

Since the Motorola DSP architectures are different from normal microprocessors, the programmer may not be familiar with some of the terms used in this document. The following discussion serves to clarify some of the concepts discussed later in this manual.

The Motorola DSP architecture can have as many as five separate memory spaces referred to as the **X**, **Y**, **L**, **P** (**P**rogram), and **E** (EMI - Extended Memory Interface) memory spaces. **L** memory space is a concatenation of **X** and **Y** data memory and is considered by the Assembler as a superset of the **X** and **Y** memory spaces. **E** memory is specific to the DSP56004 processor, and provides for different data representations for various memory hardware configurations. The Assembler will generate object code for each memory space, but object code can only be generated for one memory space at a time.

The memory space and address location into which the object code generated by the Assembler will be loaded are referred to as the **load memory space** and **load address**, respectively. Because the DSP architecture allows data transfers between memory spaces, sometimes object code is loaded into an address of one memory space but will later be transferred to a different memory space and address before the program is run. One example of this might be a program located in an external EPROM that will be transferred into external program RAM before it is run. The transfer of code/data from one memory space/address to a different memory space/address is called an **overlay**.

When the object code for a part of the program is generated that later will be used as an overlay, the load memory space and load address do not correspond to the memory space and address where the program will be run. The memory space and address location where the code/data will be located when the program is run are referred to as the

runtime memory space and **runtime address**, respectively. If the Assembler only used the load address to assign values to labels, then the program would not contain the correct label references when it was transferred to the runtime memory space and the runtime address.

During the assembly process, the Assembler uses location counters to record the addresses associated with the object code. In order to facilitate the generation of object code for overlays, the Assembler maintains two different location counters, the **load location counter**, which determines the address into which the object code will be loaded and the **runtime location counter**, which determines the address assigned to labels. In addition, the Assembler keeps track of the **load memory space**, which is the memory space into which the object code will be loaded, and the **runtime memory space**, which is the memory space to which an overlay will be transferred and the memory space attribute that will be assigned to labels. See Chapter 4, Software Project Management, for a practical discussion of the use of memory spaces and location counters.

The Motorola digital signal processors are capable of performing operations on modulo and reverse-carry **buffers**, two data structures useful in digital signal processing applications. The DSP Assembler provides directives for establishing buffer base addresses, allocating buffer space, and initializing buffer contents. For a buffer to be located properly in memory the lower bits of the starting address which encompass one less than the buffer size must be zero. For example, the lowest address greater than zero at which a buffer of size 32 may be located is 32 (20 hexadecimal). More generally, the buffer base address must be a multiple of 2^k, where 2^k is greater than or equal to the size of the buffer. Buffers can be allocated manually or by using the Assembler buffer directives (see Chapter 6).

The Assembler operates in either **absolute** or **relative** mode, depending on the presence of the command line **-A** option. In relative mode the Assembler creates relocatable object files. These files can be combined and relocated using the Motorola DSP linker. In absolute mode the Assembler generates absolute object files. Absolute files cannot be relocated but can be loaded directly for execution. By default the Assembler runs in relative mode.

1.8 ASSEMBLER SUPPORT FOR DIGITAL SIGNAL PROCESSING

As mentioned previously, the Assembler offers facilities commonly found in other macro Assemblers, such as nested macro capabilities, include files, and conditional assembly. The Assembler must also provide extensions in support of the unconventional architecture of the Motorola digital signal processors, as well as aids for programming DSP-specific applications. Some of these features are discussed briefly below; see the appropriate chapters later in this manual for more information.

The Assembler supports the use of arbitrary algebraic expressions as arguments to various directives and as immediate operands in certain instructions. Terms of these expressions may consist of the Assembler's own built-in functions, which perform data conversion, comparison, and computational operations. In the digital signal processing domain transcendental functions for computing sine, cosine, and natural logarithm are

useful for initializing data values in memory, such as sine/cosine tables for FFT algorithms. Also, there are functions for easily converting values expressed in decimal floating point to their binary or fractional equivalents. This conversion is done automatically for immediate instruction operands and arguments to the **DC** directive (see Chapter 6). See Chapter 3 for more information on Assembler expressions, operators, and built-in functions.

The register set of the Motorola digital signal processors allows for efficient use of modulo and reverse-carry buffers for FFT applications. The Assembler supports this architecture by providing several special-purpose directives for allocating circular buffers. The **BADDR**, **BUFFER**, **DSM**, and **DSR** directives automatically advance the program counter to the next appropriate base address given the buffer size, and perform various boundary and magnitude checks to insure that the buffer is valid. The **BSM** and **BSR** provide for automatic alignment and block initialization of DSP buffers. Since a buffer allocated in this fashion can cause alignment gaps in memory, the **MU** option (see the **OPT** directive, Chapter 6) may be used to generate a full memory utilization report. See Chapter 6 for more information on Assembler directives and options.