68000 Assembler

by Paul McKee

User's Manual

Contents

1 Introduction

The program described here, 68000 Assembler, is a basic two-pass assembler for the 68000 and 68010 microprocessors. It supports the complete instruction set of both processors as well as a modest but capable set of assembler directives. The program produces formatted listing files as well as object code files in S-record format.

The program was written in VAX-11 C by Paul McKee during the fall semester, 1986. The program should be portable (with reasonable changes) to any C language implementation that supports 32-bit integers.

1 Source Code Format

2.1 Source Line Format

The input to the assembler is a file containing instructions, assembler directives, and comments. Each line of the file may be up to 256 characters long. It is recommended, however, that the source lines be no longer that 80 characters, as this will guarantee that the lines of the listing file do not exceed 132 characters in length. The assembler treats uppercase and lowercase identically.

Each line of the source code consists of the following fields:

```
LABEL   OPERATION    OPERAND,OPERAND,...   COMMENT
```

For example,

```
LOOP    MOVE.L       (A0)+,(A1)+           Sample source line
```

The fields may be separated by any combination of spaces and tabs. Except for the comment field and quoted strings, there must be no spaces or tabs within a field.

2.1.1 Label Field

Legal labels follow the rules for forming symbol names described in section 2.2. Labels may be distinguished in one of two ways: (1) They may begin in column 1, or (2) they may end in a colon, which does not become part of the label but simply serves to mark its end. A line may consist of a label alone. When a label is encountered in the source code, it is defined to have a value equal to the current location counter. This symbol may be used elsewhere is the program to refer to that location.

2.1.2 Operation Field

The operation field specifies the instruction that is to be assembled or the assembler directive that is to be performed. A size code (.B, .W, .L, or .S) may be appended to the operation code if allowed, to specify Byte, Word, Long, or Short operations, respectively. The operation field must not begin in the column 1, because the operation would be confused with a label.

2.1.3 Operand Field

The operand field may or may not be required, depending on the instruction or directive being used. If present, the field consists of one or more comma-separated items with no intervening

spaces or tabs. (There may be spaces or tabs within an item, but only within quoted strings.)

2.1.4 Comment Field

The comment field usually consists of everything on a source line after the operand field. No special character is needed to introduce the comment, and it may contain any characters desired.

A comment may also be inserted in the source file in another way: An asterisk ("*") at the beginning of the line or after the label field will cause the rest of the line to be ignored, i.e., treated as a comment.

2.2 Symbols

Symbols appear in the source code as labels, constants, and operands. The first character of a symbol must be either a letter (A-Z) or a period ("."). The remaining characters may be letters, dollar signs ("$"), periods ("."), or underscores("_"). A symbol may be of any length, but only the first 8 characters are significant. Remember that capitalization is ignored, so symbols which are capitalized differently are really the same.

2.3 Expressions

An expression may be used in the source program anywhere a number is called for. An expression consists of one or more operands (numbers or symbols), combined with unary or binary operators. These components are described below. The value of the expression and intermediate values are always computed to 32 bits, with no account being made of any overflow that may occur. (Division by zero, however, will cause an error.)

2.3.1 Operands in Expressions

An operand in an expression is either a symbol or one of the following sorts of constants.

Decimal Numbers A decimal number consists of a sequence of decimal digits (0-9) of any length. A warning will be generated if the value of the number cannot be represented in 32 bits.

Hexadecimal Numbers A hexadecimal number consists of a dollar sign ("$") followed by a sequence of hexadecimal digits (0–9 and A–F) of any

length. A warning will be generated if the value of the number cannot be represented in 32 bits.

Binary Numbers A binary number consists of a percent sign ("%") followed by a sequence of binary digits (0 and 1) of any length. A warning will be generated if the number consists of more that 32 digits.

Octal Numbers An octal number consists of a commercial at sign ("@") followed by a sequence of octal digits (0-7) of any length. A warning will be generated if the value of the number cannot be represented in 32 bits.

ASCII Constants An ASCII constant consists of one to four ASCII characters enclosed in single quote marks. If it is desired to put a single quote mark inside an ASCII constant, then two consecutive single quotes may be used to represent one such character.

If the ASCII constant consists of one character, then it will be placed in the bottom byte of the 32 bit value; two characters will be placed in the bottom word, with the first character in the higher-order position. If four characters are used, then all four bytes will contain characters, with the first in the highest-order location. However, if three characters are used, then they will be placed in the three highest-order bytes of the 32-bit value, with 0 in the low byte (this is to accommodate the high-byte-first addressing used on the 68000).

Note that ASCII constants in expressions are different from strings in DC directives, as the latter may be of any length.

2.3.2 Operators in Expressions

The operators allowed in expressions are shown in the following table, in order of decreasing precedence. Within each group, the operators are evaluated in left-to-right order (except for group 2, which is evaluated right-to-left).

Operators in Expressions

1.	()	Parenthesized subexpressions
2.	−	Unary minus (two's complement)
	˜	Bitwise not (one's complement)
3.	<<	Shift left (x«y produces x shifted left by y bits and zero filled)
	>>	Shift right
4.	&	Bitwise and
	!	Bitwise or
5.	∗	Multiplication
	/	Integer division
	\	Modulus (x\y produces the remainder when x is divided by y)
6.	+	Addition
	−	Subtraction

2.4 Addressing Mode Specifications

The 68000 and 68010 provide 14 general addressing modes. The formats used to specify these modes in assembly language programs are listed in the table below. The following symbols are used to describe the operand formats:

Operand symbols

Dn	=	Data Register
An	=	Address Register (SP may used instead of A7)
Xn	=	Data or Address register
.s	=	Index register size code, either .W or .L (.W will be assumed if omitted)
<ex8>	=	Expression that evaluates to an 8-bit value (may be empty, in which case 0 will be used)
<ex16>	=	Expression that evaluates to a 16-bit value (may be empty, in which case 0 will be used)
<ex>	=	Any expression
PC	=	Program Counter

Addressing Mode Specifications

Mode	Assembler Format
Data Register Direct	Dn
Address Register Direct	An
Address Register Indirect	(An)
Address Register Indirect with Predecrement	-(An)
Address Register Indirect with Postincrement	(An)+
Address Register Indirect with Displacement	<ex16>(An)
Address Register Indirect with Index	<ex8>(An,Xn.s)
Absolute Short or Long (chosen by assembler)	<ex>
Program Counter with Displacement	<ex16>(PC)
Program Counter with Index	<ex8>(PC,Xn.s)
Immediate	#<ex>

In addition to the general addressing modes, the following register names may be used as operands in certain instructions (e.g. MOVEC or EORI to CCR):

SR	=	Status Register
CCR	=	Condition Code Register
USP	=	User Stack Pointer
VBR	=	Vector Base Register (68010)
SFC	=	Source Function Code Register (68010)
DFC	=	Destination Function Code Register (68010)

2 Assembly Details

3.1 Branch Instructions

The branch instructions (Bcc, BRA, and BSR) are unique in that they can take a ".S" size code. This suffix directs the assembler to assemble these as short branch instructions, i.e., one-word instructions with a range to -128 to +127 bytes. If the ".S" size code is used, and the destination is actually outside this range, then the assembler will print an error message. If the ".L" size code is used, the assembler will use a long branch, which is a two-word instruction with a range of -32768 to +32767 bytes. If neither size code is specified, then the assembler will use a short branch if possible (the branch destination must be known on the first pass to be within the short branch range); otherwise it will use long branch.

3.2 MOVEM Instruction

The MOVEM instruction, which is used for saving and restoring sets of registers, has one the following two forms:

```
MOVEM   <register_list>,<effective_address>
MOVEM   <effective_address>,<register_list>
```

The register list may be an explicit register list of the form described in Section 4.2.3. On the other hand, if a particular set of registers is to be saved and restored repeatedly, the REG directive (Section 4.2.3) can be used to define a register list symbol that specifies the registers. For example, if the register list symbol WORKSET is defined as follows:

```
WORKSET  REG     A0-A4/D1/D2
```

then the following instructions will perform the same function:

```
MOVEM.L  WORKSET,-(SP)
MOVEM.L  A0-A4/D1/D2,-(SP)
```

If a register list symbol is used, it must be defined before it appears in any MOVEM instructions.

3.3 Quick Instructions (MOVEQ, ADDQ, SUBQ)

The MOVE, ADD, and SUB instructions have one-word "quick" variations which can be used certain addressing modes and operand values. The assembler will use these faster variations automatically when possible, or they may be specified explicitly by writing the mnemonic as MOVEQ, ADDQ, or SUBQ.

The MOVEQ instruction may be used for moving an immediate value in the range -128 to +127 into a data register. The assembler will assemble a MOVE.L #<value>,Dn as a MOVEQ if the value is known on the first pass.

The ADDQ (SUBQ) instruction adds (subtracts) an immediate value from 1 to 8 to (from) any alterable destination. The assembler will use the quick form if the value is known on the first pass to be in the range 1 to 8.

3 Assembler Directives

4.1 ORG - Set Origin

The assembler maintains a 32-bit location counter, whose value is initially zero and which is incremented by some amount whenever an instruction is assembled or a data storage directive is carried out. The value of this location counter may be set with the ORG directive. This is typically done at the start of a program and at appropriate places within it. The format of the ORG directive is

```
<label> ORG     <expression>
```

where <expression> is an expression containing no forward references, i.e., its value must be known on the first pass at the point where the ORG directive appears. An error will result if an attempt is made to set the location counter to an odd value; in this case the location counter will be set to the specified value plus one. The <label> is optional and, if present, the specified symbol will be set to the new value of the location counter.

4.2 Symbol Definition Directives

4.2.1 EQU - Equate Symbol

The equate directive is used to define symbols whose value will not change within the program. The format of this directive is

```
<label> EQU     <expression>
```

where <expression> is an expression containing no forward refer ences, i.e., its value must be known on the first pass at the point where the EQU directive appears. The <label> must be specified, since it tells what symbol is being defined. If <label> is omitted, an error will result. If an attempt is made to redefine a symbol that was defined with EQU, either as a label or using any symbol definition directive, an error message will be printed.

4.2.2 SET - Set Symbol

The SET directive is similar in function and format to the equate directive, with one important difference: symbols defined using SET may be redefined later using another SET directive (but not using a EQU or REG directive). The format of this directive is

```
<label> SET     <expression>
```

4.2.3 REG - Register Range

Register ranges consist of lists of registers separated by slashes ("/"). Each register range may be either a single register ("An" or "Dn") or a range of registers ("An-Am" or "Dn-Dm"), which denotes all the registers between the two registers listed (they may be given in either order). For example, the following register list specifies that D0, D1, D2, D3, D7, A1, A2, and A3 are to be saved (or restored):

```
D3-D0/D7/A1-A3
```

The registers and ranges may be specified in any order. The same format for register lists may be used with the MOVEM instruction directly. In order to avoid confusion, it is best to avoid specifying a range that includes both an address register and a data register, although the assembler will not treat this as an error.

4.3 Data Storage Directives

4.3.1 DC - Define Constant

The define constant directive is used to store strings and lists of constants in memory. The format of a DC directive is

```
<label> DC.<size>  <item>,<item>,...
```

The label will be defined to equal the address of the start of the list of data. The size code specifies that a list of bytes (.B), words (.W), or longwords (.L) is being defined; if omitted, word size is used.

A list of items follows the directive; each item may be an expression or a string. If an item is an expression, the expres sion is evaluated and stored as the size indicated, i.e., a byte, a word, or a longword. An error is generated if the value will not fit as either a signed or unsigned value in the specified size. If an item is a string, delimited by single quotes, then the string will be stored in successive entities of the size specified; if words or longwords are being generated, and the string does not fit into an whole number of words or longwords, then the string will be padded with zeros at the end to make a

whole number of words or longwords. Strings and expressions may intermixed in a single DC directive.

If words (DC.W) or longwords (DC.L) are being generated, then the start of the list of constants will be aligned on a word boundary by increasing the location counter by one, if necessary. This is not performed for DC.B directives, so that strings of bytes may be contiguous in memory. If an instruction follows a DC.B directive, the assembler will automatically adjust the location counter (if necessary) to place the instruction on a word boundary.

An example of a DC directive that defines a null-terminated string:

```
TEXT    DC.B    'DC Example',$0D,$0A,0
```

This directive results in the following data at location TEXT:

```
44 43 20 45 78 61 6D 70 6C 65 0D 0A 00  (hexadecimal)
```

4.3.2 DCB - Define Constant Block

The define constant block directive generates a block of bytes, words, or longwords that are all initialized to the same value by the assembler. The format of the directive is

```
<label>  DCB.<size>  <length>,<value>
```

The label will be defined to equal the address of the start of the block. The size code specifies that a block of bytes (.B), words (.W), or longwords (.L) is being set up; if omitted, word size is used.

The length argument is an expression that tells the number of bytes, words, or longwords that are to be in the block. This value must be known on the first pass at the point where the DCB directive appears, and it must be non-negative. The value argument is an expression whose value is to be placed in each data item in the block; it needn't be known on the first pass. An warning message will be printed if the value will not fit (as a signed or unsigned number) in the data size selected.

If word or longword size is selected, then the start of the block will be placed on a word boundary by increasing the location counter by one, if necessary. If an instruction follows a DCB.B directive, the assembler will automatically adjust the location counter (if necessary) to place the instruction on a word boundary.

4.3.3 DS - Define Storage

The define storage directive generates an uninitialized

block of bytes, words, or longwords. The format of the directive is

```
<label>  DS.<size>  <length>
```

The label will be defined to equal the address of the start of the block. The size code specifies that a block of bytes (.B), words (.W), or longwords (.L) is being set up; if omitted, word size is used.

The length argument is an expression that tells the number of bytes, words, or longwords that are to be in the block. This value must be known on the first pass at the point where the DCB directive appears, and it must be non-negative. The effect of the DS directive is basically to increase the value of the location counter by <length> times one (if DS.B is used), two (if DS.W is used), or four (if DS.L is used)

If word or longword size is selected, then the start of the block will be placed on a word boundary by increasing the loca tion counter by one, if necessary. Thus, DS.W 0 can be used to force the location counter to be aligned on a word boundary without allocating any space. However, if an instruction follows a DS.B directive, the assembler will automatically adjust the location counter (if necessary) to align the instruction on a word boundary.

4.4 END - End of Source File

The end directive is used to mark the end of the source file. It is purely optional. The format is simply

```
        END
```

The assembler will ignore anything in the source file after the END directive.

4 Usage

5.1 Command Line

The 68000 Assembler is run by typing a command line of the following form:

```
ASM <options> <filename>
```

The options are a string of letters, preceded by a dash, which alter the behavior of the assembler. The following option letters are allowed:

C	=	Show all the Constants produced by DC directives (see Section 5.2)
L	=	Produce a Listing file
N	=	Produce No object code file

If these options are not specified, the defaults are to show only one line of data from a DC directive, to produce no listing, and to produce an object file.

The filename is the name, including directory specifica tions, of the file to be assembled. No default file extension is applied. The names of the listing and object code files, if generated, are constructed by using the source file name with an extension of ".LIS" (for the listing) or ".H68" (for the object file); the output files are always placed in the user's default directory.

The program will print "68000 Assembler by PGM" when it begins work. If any errors occur, the program will print "ERROR in line X" or "WARNING in line X" (this information is also placed in the listing file). Upon conclusion, it will print the number of errors encountered and warnings generated during the assembly.

If there is an error in the command line, e.g., if no file name is specified, then the assembler will print a brief usage summary and abort.

5.2 Listing File Format

The assembler produces a listing file which shows the source code alongside the the object code produced by the assembler. A typical listing file line might appear as follows (not to scale):

```
0000102E  22D8           200  LOOP  MOVE.L (A0)+,(A1)+    Sample
```

The eight digit hexadecimal number at the left is the assembler's location counter; the generated instruction word, $22D8, is placed at that address. The next number is the source file line number, and the remainder of the line simply repeats the source line. Remember that if the source lines are no longer than 80 columns, then the listing file lines will not exceed 132 columns.

If an error is encountered or a warning is generated for a given source line, then in the listing that line will be followed by a line that describes the error. At the end of the listing, the program prints the total number of errors and warnings.

There is only limited space to list the object code in this format. There is sufficient space for the longest possible instruction, but the DC directive poses a problem, since it may generate, e.g., dozens of longwords from a single source line. The assembler's -C command line option controls the assembler's actions when the object code exceeds the space available on one line. If -C is not specified, then the assembler will print only one listing line with an ellipsis ("...") at the end of the object code field, indicating that some of the data produced by the directive was omitted from the listing. If -C is included on the command line, then the assembler will use as many source lines as are needed to print all the data produced by the directive. Each line after the first will contain only the location counter and the object code field (the source line is not repeated).

5.3 Object Code File Format

The 68000 Assembler produces an object code output file in S-record format. The object file name is the source file name, with the extension changed to ".H68". The object file and the listing file are always placed in the user's default directory.

The S-record format is designed to allow files containing any data to be interchanged in text file format with checksum error detection. The format of these files will not be described here, but the following technical information will be provided: The first line of the object file is an S0 (header) record and the last line is an S9 (termination) record. The lines in between are S1, S2, or S3 records, depending on the whether the address of the first byte in a record requires 2, 3, or 4 bytes to be represented. No record is longer than 80 characters.