68010 ASSEMBLER

by Edgar M. (Bud) Pass, Ph.D.

Copyright Notice

Limited Warranty Statement

Problems and Improvements

Users are encouraged to submit problems and to suggest or to provide improvements for the 68010 ASSEMBLER. Such input will be processed on a best effort basis. Computer Systems Consultants reserves the right to make program corrections or improvements on an individual or wholesale basis, as required. The company is under no obligation to provide corrections or improvements to all users of the 68010 ASSEMBLER. In the case of specific situations requiring extensions to the 68010 ASSEMBLER or major assistance in its use, consulting is available on a pre-arranged, for-fee basis.

Introduction

This assembler is used to convert 68010 assembler language mnemonics and expressions to 68010 machine language. The syntax of the assembler language which it accepts is compatible with the standard language as implemented in the Motorola 68000 and 68010 Resident Structured Assemblers. In order to be somewhat more compatible, it accepts several directives implemented only by the TSC 68000 Assembler.

It may also be used to produce machine language for any of the other processors in the 680xx family, although with some limitations. For those processors with instruction sets which are supersets of the 68010 instruction set, such as the 68020 and 68030, only those instructions common to the 68010 and to the superset processor may be generated directly by the 68010 assembler. For those processors with instruction sets which are subsets of the 68010 instruction set, such as the 68000 and 68008, only those instructions common to the 68010 and to the subset processor may be correctly generated by the 68010 assembler; those instructions not in the subset will be treated as invalid instructions by the subset processor. The -m0 option may be specified on the command line to inhibit the incorrect generation of code for subset processors.

This assembler provides an absolute assembler capability, and produces object code directly, without the use of a link-editor. It supports conditional assembly, the use of complex arithmetic expressions, and the inclusion of auxiliary source files.

It optionally produces an formatted listing of the program being assembled. On other than minimum systems, its output includes a sorted cross-reference of the symbols declared and used in the program, and English error messages, as appropriate. A macro pre-processor is available with the full package.

Invocation of The 68010 Assembler

The 68010 assembler is invoked with a command line formatted as follows:

```
csc68k [-/+options] source-file[.ext] [object-file]
```

in which the following symbols are used:

options
represents optional assembler options, as described below; multiple options may be separated or may be joined together; the '+' or '-' symbol must be used to start each group of options.

+B, +b, -B, -b
inhibits the construction of the output object file, which is produced by default; this file normally has the same file name as the main program source file, except that it has suffix ".mxt" (formatted as S-records) or ".com" or ".bin" (formatted as binary records), the details of which are described later in this manual.

+F, +f, -F, -f
inhibits the generation of word-alignment bytes generated when directives which cause odd alignment are followed by directives requiring even alignment.

+Hxxx, +hxxx, -Hxxx, -hxxx
sets listing pagination mode and sets page length to decimal xxx, which may not be less than 24 or greater than 130.

+L, +l, -L, -l
inhibits the generation of the formatted source listing; the source listing may be redirected to disk or printer with command line parameters, as supported by the operating system; vertically formatted and paginated listings may be created with the H option.

+Mx, +mx, -Mx, -mx
indicates type of processor for which code is to be generated: 0 for 68000, 1 for 68010, 2 for 68020, 3 for 68030 (1, 2, and 3 cause the same actions).

+Ox, +ox, -Ox, -ox
specifies name of object file to be produced; it will be suffixed as described above, except for minimum systems.

+Pxxx, +pxxx, -Pxxx, -pxxx
sets width of formatted output lines to decimal xxx; the default is 80, but it may be set to any value between 50 and 132; however, setting the output line width less than 64 will inhibit output formatting.

+S, +s, -S, -s
specifies the generation of the formatted and sorted symbol table listing at the end of the source listing on other than minimum systems.

+Tx, +tx, -Tx, -tx

indicates type of object file to be produced: 0 for S3, 1 for S2, 2 for 02, 3 for 03, 4 for 03+21, 5 for 05, 6 = 05+22 (either 0 or 3 is usually the default option, depending upon the system, but minimum systems allow only 0).

+W, +w, -W, -w
causes the assembler to ignore .L, .W, .l, and .w suffixes on labels within expressions, for compatibility with other compilers.

+Zxx, +zxx, -Zxx, -zxx
sets max symbol length to xx bytes; the default for minimum systems is 8 bytes, for PC's is 16 bytes, and for other systems is 31 bytes.

source-file
specifies the main program source file name; it must meet all requirements for a legal file name for the operating system under which the assembler is used.

ext
represents an optional file extension; by default, this assembler expects a source file name to have extension ".txt", if none is specified.

object-file
specifies name of object file to be produced; it will be suffixed as described above, except for minimum systems.

If the assembler detects missing or invalid information specified on the command line, it will output a prompt message indicating the correct format of the command line. On other than minimum systems, this prompt provides summary information on valid command lines.

Assembler Syntax

The input to this assembler is one text file containing a number of lines formatted according to the requirements of the 68010 assembler language. Because of the ability of this assembler to include auxiliary text files, source text libraries may be readily established and used. Whatever the source, the lines may be automatically formatted to separate the label, instruction, and operand fields, whenever the output line is specified to be at least 64 characters in width.

The statements acceptable to this assembler are free-field, one statement per line. Intermediate fields are separated by at least one space or tab, and each line is terminated by a carriage return.

There are three types of statements in the assembler language, as follows:

instruction
directive
comment

Each of these statement types is discussed below.

Instruction Statement

An instruction statement consists of zero to four fields, as follows:

[label[:]] [instruction [operand] [comment]]

Label Field

The label field is optional. It is normally used to provide a symbolic name for the address of the machine code generated by the following assembler instruction. However, labels may also be used as parts of arithmetic expressions to assist in the process of assembling the program. If the label field is omitted, it must be represented by a space. Otherwise, the instruction field would be misinterpreted as the label field.

Labels are composed of characters of which all are significant up to the default number of 8 for minimum systems, 16 for PC's, or 31 for other systems. Only the first eight characters are displayed on the formatted listing.

The first character of a label must be a letter or period, and the remaining characters must be letters, periods, digits, underlines, and dollars. Lower case and upper case letters are considered unique. Label definitions starting at other than the left margin must end with a colon; however, the colon is not considered to be a part of the label name.

No alignment is normally forced before or after a label's value is set. However, alignment may be forced before a label definition by instructions and by certain types of declaratives, if the label is coded on the same line as the instruction or declarative. In order to ensure label alignment (in cases such as labels not coded on the same line as an instruction), an DS 0 or EVEN directives may be inserted on the line before a label definition, or a label may also be coded on the line with one of these directives.

The following symbols may not be used as labels, since to do so would create ambiguous register references:

```
    A0 thru A7, CCR, D0 thru D7, DFC, PC, SFC, SP, SR, USP, VBR,
    a0 thru a7, ccr, d0 thru d7, dfc, pc, sfc, sp, sr, usp, vbr.
```

These represent the following 68010 registers:

```
    A0 thru A7  32-bit address registers A0 thru A7,
    CCR         8-bit condition code register (low 8 bits of SR),
    D0 thru D7  32-bit data registers D0 thru D7,
    DFC         Destination function code register,
    PC          32-bit program counter,
    SFC         Source function code register,
    SP          32-bit stack pointer (same as A7),
    SR          16-bit status register,
    USP         32-bit user stack pointer,
    VBR         Vector base register.
```

Instruction Field

The instruction field contains the 68010 assembler instruction or directive. It will normally always be present, but if it is omitted, the operand and comment fields must also be deleted. If the instruction field consists only of an asterisk, a comment field may follow the instruction field. Lines containing labels only are allowed, as are totally-blank lines. The program counter is always aligned to an even address before a 68010 assembler instruction.

An instruction name contains no more than seven characters, with upper case and lower case non-unique. The complete set of 68010 assembler language instructions is described in the Motorola 68010 Resident Structured Assembler manual. The set of directives accepted by this assembler is described later in the current manual.

Those 68010 instructions which require data size specification may be suffixed with length modifiers. These include ".b" for 8-bits, ".w" for 16-bits, and ".l" for 32-bits. If no data length modifier is specified, a ".w" suffix is normally assumed, except for branch instructions, for which a ".l" suffix is assumed. The ".w" suffix default also applies to most directives, except for ORG and RORG, for which a ".l" suffix is assumed.

The suffix ".s" is used to indicate short branch instructions. This assembler will not change short branches to long branches or vice versa, and a short branch to the next instruction will not be changed to a long branch, but will be noted as an illegal instruction.

This assembler will not automatically change add, subtract, or move instructions to their equivalent quick immediate form. Although this would produce faster and smaller object code, it may also produce undesirable results in some cases. The programmer is thus responsible for coding the best form of a desired operation.

Operand Field

The operand field contains zero or more parameters of the instruction. If multiple sub-fields are present, they must be separated by a comma. Only the MOVEM instruction allows more than two operands, but the register list of that instruction must be separated with slashes and hyphens, even if only one register is used. Many instructions (such as MOVE) have two operands, which are designated as source and destination sub-fields, from left to right. The DB types of declarative may also contain multiple sub-fields.

This assembler supports the standard Motorola notation for the 68010 effective addressing modes. They are summarized below.

CCR, DFC, SFC, SR, USP, VBR
Control register

An
Address register direct

Dn
Data register direct

(An)
Address register indirect

(An)+
Address register indirect with postincrement

-(An)
Address register indirect with predecrement

expression(An)
Address register indirect with displacement

expression(An,Xn)
expression(An,Xn.W)
expression(An,Xn.L)
Address register indirect with index

expression (in ORG)
*-expression
*[+expression] (in ORG)

Address short or long

expression (in RORG)
*[+expression] (in RORG)
expression(PC)
Program counter with displacement

expression(PC,Xn)
expression(PC,Xn.W)
expression(PC,Xn.L)
Program counter with displacement and index

#expression
Immediate

Comment Field

The comment field consists of any text following the fields described above, but preceding the terminating carriage return on the line. It may contain characters with hex values from $20 thru $7E, plus tab characters, which are each considered equivalent to one space.

Directive Statement

A directive statement consists of one to four fields, as follows:

[label[:]] directive [operand] [comment]

Label Field

The label field of a directive follows the same rules provided above. However, a label may be used only with the directives indicated below.

Directive Field

The directive field provides information for the assembler, rather than instructions for the 68010 processor. This information includes such items as the base address of the program, the establishment of symbol values, the allocation of storage, the specification of additional source files to be read by the assembler, etc. A list of the directives implemented by this assembler is provided later in this manual. Lower case and upper case letters in the directive field are not considered unique.

Operand Field

The operand field consists of zero or more sub-fields. Multiple sub-fields are separated with commas.

Comment Field

The comment field consists of any text following the fields described above, and preceding the terminating carriage return. The comment field may be composed of characters with ASCII hex values from $20 thru $7E and tab characters.

Comment Statement

A line which starts with an asterisk is ignored by the assembler except for being optionally listed (along with the remainder of the program). A comment statement may be composed of characters with ASCII hex values from $20 thru $7E and tab characters. A comment statement may be labelled, since the asterisk appears to the assembler in this case to be in the instruction field.

Directives

[label[:]] DC   operand[,operand[,...]]
[label[:]] DC.B operand[,operand[,...]]
[label[:]] DC.L operand[,operand[,...]]
[label[:]] DC.W operand[,operand[,...]]
[label[:]] FCB operand[,operand[,...]]
[label[:]] FCC operand[,operand[,...]]
[label[:]] FDB operand[,operand[,...]]
DC specifies the assignment of one or more values into successive words, bytes, or long words. The suffixes are interpreted as described above for instruction suffixes. A DC.B directive not followed immediately by another DC.B directive will normally have a filler byte inserted, if necessary, to force word alignment. FCB and FCC are equivalent to DC.B and FDB is equivalent to DC.W. Strings must be enclosed in single or double quote symbols.

[label[:]] DS   expression
[label[:]] DS.B expression
[label[:]] DS.L expression
[label[:]] DS.W expression
[label[:]] EVEN
[label[:]] RMB   expression
[label[:]] SPACE expression
DS indicates the reservation of a specified number of words, bytes, or long words, usually without placing values into the locations. No word alignment is forced after any DS directive, although word alignment is forced before DS, DS.L, DS.W, and EVEN directives. RMB is equivalent to DS.B, and EVEN is equivalent to DS 0. SPACE is equivalent to DS.B except that the contents of the locations are initialized to binary zeroes.

END [label]
END indicates the logical end of the assembler language unit, although not necessarily the end of the program, as the assembler will continue to process input until it finds physical end of file. Although the END statement may not be labelled, an optional label may appear as an operand indicating the starting address of the program, and will be placed into the output object file as the transfer address.

ENDC
ENDIF
ENDC and ENDIF indicate the end of the range of an IFxx or IFDEF declarative.

label[:] EQU expression
EQU defines a symbol and sets its value to that of the expression. This operand may contain complex expressions, but not forward references, and once defined by an EQU statement or as a program label, a symbol's value may not be changed.

IFxx expression
IFDEF label
IFP1

IFxx or IFDEF indicates the beginning of a sequence of assembler statements which may be logically conditionally included or excluded from the input to the assembler. The xx indicated above must be replaced by EQ, GE, GT, LE, LT, or NE, in order to indicate the condition (expression xx zero) for the inclusion of the assembler statements, which are terminated by a corresponding ENDC/ENDIF directive. IFDEF includes the sequence of statements if the indicated label is defined. IFP1 includes the sequence on the first pass of the assembler, but excludes it on the second pass. IFxx, IFDEF, and IFP1 statements may be nested to 32 aggregate levels.

INCLUDE filename
LIB filename
USE filename
INCLUDE, LIB, and USE provide the name of an auxiliary source file which is logically inserted into the current source file in place of the INCLUDE, LIB, or USE directive. The file name must follow the conventions required by the host operating system.

LIST
LIST allows the source listing to be produced, unless it is suppressed by an option on the command line.

NOLIST
NOLIST suppresses source listing production.

NOPAGE
NOPAGE suppresses source listing pagination.

ORG expression
ORG.L expression
ORG.W expression
ORG specifies the starting absolute memory base of the program counter. ORG and ORG.L specify long absolute addresses representing the full range of the address space. ORG.W specifies short absolute addresses restricted to the address ranges $00000000-$00007FFF and $FFFF8000-$FFFFFFFF.

PAGE
PAGE causes the next line on the source listing to appear on the next page.

RORG expression
RORG.L expression
RORG.W expression
RORG specifies the starting absolute memory base of the program counter, and the generation of program-counter-relative effective addresses. RORG and RORG.L specify long absolute addresses representing the full range of the address space. RORG.W specifies short absolute addresses restricted to the address ranges $00000000-$00007FFF and $FFFF8000-$FFFFFFFF.

RPT expression
RPT specifies the number of times the next line in the same source file will be repeated. The next line is always included at least once, even if the value of the expression is zero.

label[:] SET expression
label[:] = expression
SET defines or redefines a symbol and sets its value to that of the expression. This operand may contain complex expressions, but not complex forward references, and this symbol's value may be changed only by another SET or = directive.

SPC
SPC places a blank line in the source listing.

TTL title

NAM title
TTL and NAM specify the title to be placed at the top of each page in the source listing when pagination is active.

Arithmetic Expressions

Expressions consist of combinations of symbols, constants, operators, and parentheses. Symbols must normally be defined before being used in complex arithmetic expression evaluations. All arithmetic and logical operations are performed using 32-bit two's complement integer arithmetic; division is truncated, and overflows are normally lost. Expressions may be forced into 16-bit mode by enclosing them with '(' ... ').W', or by masking them with $ffff.

Symbols used in expressions include the following:

- program labels
- symbols defined with DC, DS, EQU, SET directives
- numeric constants
- '*' (the program counter)

Program labels and symbols are composed of one to 31 characters, starting with a letter or period, as described earlier in this manual.

Constants may be expressed in decimal, hexadecimal, binary, octal, or ASCII, and are constructed as follows:

- decimal constants are composed of one or more digits, optionally followed by D or d. hexadecimal constants start with a dollar symbol or end with H or h, and are composed of one or more digits or letters, restricted to the ranges a thru f and A thru F.

- binary constants start with a percent symbol or end with B or b, and are composed of one or more digits 0 and 1.

- octal constants start with an at symbol or end with O or o, and are composed of one or more digits in the range 0 to 7.

- ASCII constants start and end with single or double quote symbols, and contain characters with hex values $20 thru $7E; quote symbols within the constants, matching the delimiters, are represented by two adjacent quote symbols; the number of characters in ASCII constants within expressions is restricted to four except in DC and equivalent types of directives.

Operators used in expressions may include the following, in decreasing order of priority:

unary minus	-
left/right logical shifts	<< and >>
logical and/or	& and !
multiplication/division	* and /
addition/subtraction	+ and -

Object File Formats

One of the object file types produced by this assembler follows the format of the standard Motorola S-records file. The format of each type of S-record is described below.

The record type field is an 'S' and a digit, interpreted as follows:

- S1 indicates a record containing data, starting at the value in the 16 bit (2 byte) address field.
- S2 indicates a record containing data, starting at the value in the 24 bit (3 byte) address field.
- S3 indicates a record containing data, starting at the value in the 32 bit (4 byte) address field.
- S7 indicates a record containing an optional 32 bit (4 byte) xfer address, and terminates a group of S3 records.
- S8 indicates a record containing an optional 24 bit (3 byte) xfer address, and terminates a group of S2 records.
- S9 indicates a record containing an optional 16 bit (2 byte) xfer address, and terminates a group of S1 records.

The number of bytes to follow is indicated as two hexadecimal digits. If an address or value is to appear in this record, it follows the length field just described; it is represented in hexadecimal. If data appears in the record, it follows the starting address, and is represented in hexadecimal. The one's-complement checksum of all of the items is indicated as two trailing hexadecimal digits.

Another set of the object file types produced by this assembler is the SK*DOS (copyright Star-Kits Software Systems) binary load records 02, 03, and 05. These formats are described below:

leadin	address	length	body	comments
$02	2-byte-offset	1-byte	data	
$03	4-byte-offset	2-byte	data	default
$05	4-byte	2-byte	data	
$16	2-byte-offset	none	none	xfer address
$18	4-byte-offset	none	none	xfer address, default
$19	4-byte	none	none	xfer address

The naming of the object file produced by this assembler is dependent upon the type of object file and the system on which it is run. The following table provides the suffixes:

type	system	suffix
S3/S2	all	.mxt
02/03/05	SKDOS	.com
02/03/05	others	.bin

68010 Instruction Syntax

The syntax of the assembler language accepted by this assembler is compatible with the standard language as implemented by the Motorola 68010 Resident Structured Assembler.

Upper and lower case letters in instructions are not considered unique. For those instructions with length modifiers, the suffix ".w" will usually be assumed if none is provided in the instruction field. Branch instructions must be coded with ".s" to indicate short branches or with ".l" or without suffix to indicate long branch instructions. A short branch to the next instruction is invalid and will not be changed to a long branch. For unsized instructions, length modifiers are generally ignored.

Certain instructions (ADD, AND, CMP, EOR, MOVE, NEG, OR, and SUB) have variations in their basic operation codes beyond length modification. The following variations are normally handled by this assmebler, but may be coded explicitly, as follows:

```
    ....A  address register operation
    ....I  immediate operation
```

Other variations must be coded explicitly (as the assembler has no manner in which to determine that they were desired), as follows:

```
    ....M  memory operation
    ....Q  quick immediate operation
    ....X  extended carry operation
```

For the MOVEM instruction, if only one register is present in the register list, it must appear as Dx-Dx, Ax-Ax, Dx/Dx, or Ax/Ax. The syntax of the MOVEM requires a register list, not a register.

The BKPT, MOVEC, MOVEM, and RTD instructions and the MOVE CCR,... and MOVE ...,CCR variants of the MOVE instruction are invalid when the 68000 option (-m0) is chosen on the command line, as these are all 68010 instructions.

Utility Programs

Miscellaneous utility programs are normally included with the 68010 assembler. Their functions may generally be determined by executing them with no arguments, as they will provide a usage prompt. In particular, utility programs with names starting with s1, s2, and s3 are usually included to provide the capabilities of translating object file formats.

Error Messages

Most error messages are self-explanatory; however, all of them are briefly explained in the list below. For minimum systems, only error numbers are output by the assembler.

19 Assembler error
: An internal error in the assembler has been detected, probably caused by syntax errors in the statement.

07 Bad 16 bit displacement
: The displacement value is less than -32768 or greater than +32767.

25 Bad 16 bit displacement range
: The displacement value is less than -32768 or greater than +32767.

11 Bad 16 bit extension
: The extension value is less than -32768 or greater than +32767.

02 Bad 3 bit value

The quick immediate data value is not in the range 1 to 8, or the specified breakpoint number is invalid.

03 Bad 32 bit field specifier
The bit field value specifies a bit outside of a long word.

31 Bad 4 bit value
The specified vector number is invalid.

08 Bad 8 bit displacement
The displacement value is less than -128 or greater than +127.

26 Bad 8 bit displacement range
The displacement value is less than -128 or greater than +255.

12 Bad 8 bit extension
The extension value is less than -128 or greater than +255.

15 Bad 8 bit field specifier
The bit field value specifies a bit outside of a byte.

22 Bad 8 bit operand
The operand value is less than -128 or greater than +255.

01 Bad address displacement
The displacement is out of range for the effective address type.

06 Bad count operand
The immediate shift count is less than 1 or greater than 8.

10 Bad destination effective address
The destination effective address type is invalid for this type of instruction.

09 Bad destination indexed address
The indexed destination effective address type is invalid for this type of instruction.

13 Bad effective address
The effective address type is invalid for this type of instruction.

14 Bad expression format
The expression is badly-formed or contains a space.

27 Bad expression range
The address expression has a value outside the short addressing range.

18 Bad instruction field
The item in the instruction field cannot be recognized.

20 Bad label usage
The directive statement may not be labelled.

xx Bad library file ...
The indicated library file could not be found.

21 Bad multiple destination
: The operand specifies registers not allowed in a multiple destination list.

xx Bad object file ...
: The indicated object file could not be created.

16 Bad operand 1 for instruction type
: The first operand is improper for this instruction.

04 Bad operand 1 formated incorrectly.
: The first operand has been coded incorrectly.

17 Bad operand 2 for instruction type
: The second operand is improper for this instruction.

05 Bad operand 2 format
: The second operand has been coded incorrectly.

23 Bad operand format
: The instruction operand field is badly-formed or contains an unintended space or other illegal character.

xx Bad source file ...
: The indicated source file could not be found.

xx Library file nest error ...
: The indicated library file could not be included, as the nest level is too deep (usually 3 for minimum systems and 12 for others).

24 Phasing error
: The non-redefinable symbol has been found to have a different value on the second pass as it had on the first pass of the assembler; This is often due to conditional assembly including different text on the two passes or due to other syntax errors in the assembly text.

28 Redefined symbol
: The symbol defined in other than a SET statement has been redefined.

30 Symbol table overflow
: The table which contains the program labels has been filled to capacity. Attempt to reduce its size with the Zxx command line option, reduce the number of labels used in the program, reduce their lengths, or divide the program into smaller parts.

29 Undefined symbol
: The symbol is used in an expression or in a context which does not allow forward references or is not defined in the program.