32000 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 32000 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 32000 ASSEMBLER. In the case of specific situations requiring extensions to the 32000 ASSEMBLER or major assistance in its use, consulting is available on a pre-arranged, for-fee basis.

Introduction

This 32000 assembler is used to convert 32000 assembler language mnemonics and expressions to 32000 machine language. The syntax of the 32000 assembler language which it accepts is compatible with the standard language as implemented in the National Semiconductor Series 32000 Assembler.

The 32000 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 produces an optional formatted listing of the program being compiled, including a sorted cross-reference of the symbols declared and used in the program, and generates English error messages. A macro pre-processor is available with the full package.

Invocation of The 32000 Assembler

The 32000 assembler is invoked with a command line formatted as follows:

```
asm32k program[.ext] [+/-options]
```

in which the following symbols are used:

program
specifies the main program source file name; it must meet all requirements for a legal file name for the operating system under which the 32000 assembler is being invoked.

ext
represents an optional file extension; by default, the 32000 assembler expects a source file name to have extension ".txt", if none is specified.

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; for example, the following string could represent a properly-formed set of options:

```
+blt4000d -M1
```

+B, +b, -B, -b
inhibits the construction of the output object file, which is produced by default; this file 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
forces the generation of word-alignment bytes generated when DC.B directives are not followed immediately by DC.B directives.

+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 or with auxiliary programs.

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

+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; setting the output line width less than 64 will inhibit output formatting.

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

+Txxxx, +txxxx, -Txxxx, -txxx
sets symbol table size to decimal xxxx bytes, to attempt to force a smaller or larger symbol table size; since the maximum size of the symbol table in a given case depends upon the size of memory available to the assembler, if insufficient space is available to process a program, the number of program labels and symbols must be decreased, or the program must be segmented and separately assembled.

+X, +x, -X, -x
inserts the formatted symbol table at the end of an S3/S2 object file.

If the assembler finds 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 the valid command-line arguments (file names and options).

Assembler Syntax

The input to this assembler is one text file a number of lines formatted according to the requirements of the 32000 assembler language. Because of the ability of this assembler to include auxiliary text files, source text libraries may be established and used. Whatever the source, the output lines are automatically formatted to separate the label, instruction, and operand fields, whenever the output line is 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 32000 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 code generated by the following assembler instruction. If the label field is omitted, it must be replaced with a space.

Labels are composed of one to 30 characters, of which all characters are significant. 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. Minimum systems limit the significant label length.

None of the 32000 register names may be used as labels, in either upper or lower case. These represent the following 32000 registers:

```
bcnt        32-bit breakpoint count register,
bpr0        32-bit breakpoint 0 register,
bpr1        32-bit breakpoint 1 register,
eia         32-bit error/invalidate register,
f0-f7       32-bit floating-point data registers,
fp          32-bit frame pointer,
```

fsr	32-bit floating-point status register,
intbase	32-bit interrupt base register,
mod	16-bit module register,
msr	32-bit memory management status register,
pc	32-bit program counter,
pf0	32-bit program flow 0 register,
pf1	32-bit program flow 1 register,
psr	16-bit processor status register,
ptb0	32-bit page table base 0 register,
ptb1	32-bit page table base 1 register,
r0-r7	32-bit general registers,
sb	32-bit static base register,
sc	32-bit sequential count register.
sp	32-bit stack pointer,
sp0	32-bit interrupt stack pointer,
sp1	32-bit user stack pointer,

Instruction Field

The instruction field contains the 32000 assembler instruction. It will normally always be present, but if it is omitted, the operand and comment fields must also be deleted. Lines containing labels only are allowed, as are totally-blank lines.

A 32000 instruction name contains no more than seven characters, with upper case and lower case non-unique. The complete set of 32000 assembler language instructions is described in the National Semiconductor Series 32000 Assembler manual.

Operand Field

The operand field contains zero or more parameters of the instruction. If multiple sub-fields are present, they must be separated by commas.

This assembler supports the standard National Semiconductor notation for the 32000 effective addressing modes. The standard format of 32000 assembler language operands is described in the National Semiconductor Series 32000 Assembler manual.

The determination of the type of an operand which may be either PC-relative or absolute addressing is made by inspecting the first character of the operand. If it is an asterisk, the operand is assumed to be PC-relative, otherwise it is assumed to be absolute. Expressions involving the program counter may be forced to absolute addressing by enclosing them in parentheses. Expressions not directly involving the program counter may be forced to PC-relative addressing by prefixing the expressions with *-*+. Thus, the expressions *, *+1, *-1, *-*+(...), and *-*+label are PC-relative, and the expressions (*), (*+1), (*-1), (...), and label are absolute. Operands of instructions which allow only relative addressing are always interpreted as PC-relative, regardless of the operand format.

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 32000 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 the 32000 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.

Directives

```
[label[:]] DC   operand[,operand[,...]]
[label[:]] DC.B operand[,operand[,...]]
[label[:]] DC.D operand[,operand[,...]]
[label[:]] DC.L operand[,operand[,...]]
[label[:]] DC.W 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. Strings must be enclosed in single or double quote symbols.

```
[label[:]] DS   expression
[label[:]] DS.B expression
[label[:]] DS.D expression
[label[:]] DS.L expression
[label[:]] DS.W expression
[label[:]] EVEN
```

DS indicates the reservation of a specified number of words, bytes, or long words, without placing values into the locations; no word alignment is forced after any DS directive, although word alignment is forced before DS and DS.W directives, and double-word alignment is forced before DS.D and DS.L directives. EVEN is equivalent to DS.W 0.

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 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 single operand; 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.

LIB filename
USE filename
LIB and USE provide the name of an auxiliary source file which is logically inserted into the current source file in place of the 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.D expression ORG.L expression
ORG specifies the starting absolute memory base of the program counter, assuming long absolute addresses representing the full range of the address space.

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

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 single operand; 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 location counter)

Program labels and symbols are composed of one to 30 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 represented by the digits 0 thru 9; they may not contain decimal points.

- hexadecimal constants start with the dollar symbol, which must be followed by one or more digits and/or letters, restricted to the ranges a thru f and A thru F; alternately, hexadecimal constants may start with a digit and end with h or H.

- binary constants start with the percent symbol, which must be followed by one or more 0's and 1's; alternately, binary constants may start with a digit and end with b or B.

- octal constants start with the at symbol, which must be followed by one or more digits, restricted to 0 to 7; alternately, octal constants may start with a digit and end with q or Q.

- 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 the 32000 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 the 32000 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 the 32000 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
```

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 format
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 or has been specified as too large; attempt to change its size with the 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.