CSC 8-Bit Cross-Assembler

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

Copyright Notice

Limited Warranty Statement

Computer Systems Consultants, Inc., and its agents, makes no express or implied warranties concerning the applicability of the CSC Assembler to a particular purpose. Liability is limited to the original license cost. This warranty is presented expressly in lieu of all other warranties, expressed or implied, including those of merchantability and fitness for use and of all other obligations on the part of Computer Systems Consultants, Inc. and its agents.

Problems and Improvements

Users are encouraged to submit problems and to suggest or to provide improvements for the CSC 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 CSC Assembler. In the case of specific situations requiring extensions to the CSC Assembler or major assistance in its use, consulting is available on a pre-arranged, for-fee basis.

INTRODUCTION TO THE CSC ASSEMBLER

The CSC Assembler is a generic term used in this manual to denote assemblers for several different microprocessors. These assemblers are all so similar that they are all described together, with pertinent differences noted.

The CSC Assembler is written in the C language and thus may be transferred to a variety of systems with far less effort than that required to transfer a program coded in a lower-level language. An description of the C source language modules appears later in this manual.

In addition to the basic ability to recognize the target computer's assembly language and to process symbolic address expressions, it has the ability to support structured assembly programming, the ability to define conditional assembly directive expressions, and the ability to assemble programs stored in modular form.

Since the RCA 1805 instruction set forms a superset of the 1802 instruction set, the CSC 1805 Assembler is capable of assembling programs for both of these microprocessors.

Since the Motorola 6801 instruction set forms a superset of the 6800, 6802, and 6808 instruction sets, and is identical to that of the 6803 instruction set, the CSC 6801 Assembler is also capable of assembling programs written for all of those microprocessors.

The CSC 6801 Assembler is also capable of assembling programs for the Hitachi 6303, as it recognizes the six extra instructions, and for the Motorola 68HC11, as it recognizes the extra instructions and addressing modes. There is one minor inconsistency, in that the XGDX instruction has different opcodes on the 6303 and on the 68HC11. The CSC 6801 Assembler resolves this ambiguity by changing the name of the 6303 XGDX instruction to XDGXH; however, if an instruction recognized only by the 6303 is found, such as AIM, EIM, OIM, etc., the XGDX instruction will be interpreted as a 6303 XGDX instruction (hex 18).

The CSC 6809 Assembler also recognizes 6800 and 6801 mnemonics, generating code for them to simulate the register and memory changes caused by the instructions on the 6800 and 6801. The CSC 6801 Assembler recognizes operators coded in the form 'lda a' or in the form 'ldaa', but the CSC 6809 assembler requires 6800 and 6801 simulated mnemonics to be coded in the latter form.

The CSC 6805 Assembler is also capable of assembling programs written for the Motorola 146805 and 68HC05, as it recognizes the extra instructions implemented by those processors.

The CSC 8048 Assembler is also capable of assembling programs written for the Intel 8022, 80C48, and 80C49, as it recognizes the extra instructions implemented by these processors. It is capable of assembling programs written for the Intel 8020 and 8021, since the 8048 instruction set forms a superset of those instruction sets. It is also capable of assembling programs written for the Intel 8035, 80C35, 8039, 80C39, 8040, 8049, 8050, 8748, and 8749, as their instruction sets are identical to that of the 8048. The CSC 8048 Assembler allows the use of registers in expressions, but all registers must be literally coded when syntactically required, due to ambiguities in the 8048 assembler language.

The CSC 8051 Assembler is also capable of assembling programs written for the Intel 8031 and 8751, as their instruction sets are identical to that of the 8051. The CSC 8051 Assembler allows the use of registers in expressions, but the A and C registers must be literally coded when syntactically required, due to ambiguities in the 8051 assembler language.

The standard Mos 6502 family instruction set and assembler language format is recognized by the CSC 6502 Assembler. Many of the variants, such as those requiring the prefixing of the directives with periods, are also accepted.

Since the Intel 8085 instruction set forms a superset of the 8080 instruction set, the CSC 8085 Assembler is capable of assembling programs for both of these microprocessors.

The Zilog Z8 assembler language format is unlike most others in that it is free-format. Since the CSC Assembler is not free-format, changes, as described below, may be required to convert existing Z8 programs into a format acceptable to the CSC Assembler. In general, these changes may be made quite simply with a text editor.

Only one opcode or directive per line is allowed. Thus the following free-format Z8 assembler language statement:

```
label PROCEDURE ENTRY $ABS %0100
```

should be re-coded as follows:

```
label PROCEDURE
      ENTRY
      $ABS    %0100
```

ARRAY structures are not implemented. They may be re-coded as FCC, FCB, DB, and similar directives.

Opcodes and assembler directives should not start in the first column, as they may be mis-interpreted as labels. A label definition may or may not be followed by a colon, if it starts in the first column, but must be followed by a colon if it does not start in the first column.

Comments may not be continued to successive lines. They may be introduced by either ';' or '!'. Once started, a comment continues to the end of the line.

For the LDC, LDCI, LDE, and LDEI instructions, the first parameter must be '@RRnnn' to generate opcodes hex 92, 93, D2, or D3. This is due to a syntactical ambiguity which the Intel assembler resolves by maintaining a symbol's value and type, but which the CSC Z8 Assembler is unable to otherwise resolve. The ability to maintain a symbol's type is not worth the overhead.

Since the current version of the CSC Assembler produces absolute code, and does not use a linker, many of the assembler directives are accepted, but are ignored. This includes MODULE, $SECTION, $SDEFAULT, CONSTANT, PROCEDURE, ENTRY, GLOBAL, INTERNAL, EXTERNAL, etc. $ABS is interpreted as ASEG, and $REL is interpreted as CSEG. Labels on these declaratives are allowed, but have no special meaning beyond their normal interpretation.

RUNNING THE CSC ASSEMBLER

The command line used to execute the CSC Assembler is provided below.

```
csc***** source-file [+/-options]
```

in which '*****' represents the processor name.

The source file name has a default suffix of ".TXT". The object file name has a suffix of ".MXT" and the file name of the source file without the suffix.

The name selection is limited by the C library and operating system of the host environment. For some systems, the only suffix allowed on the primary source file name is ".TXT"; in this case, the suffix should be omitted on the command line.

Only one primary source file name is allowed, but the assembler directives INCLUDE, LIB, and USE may be used within a source file to specify multiple input files. Library file names have no default prefix or suffix and must be specified as required by the C library and operating system of the host environment.

There is one peculiarity of the use of the CSC Assembler with OS/9. If OS/9 outputs ERROR #207 when the user attempts to run the assembler, try first loading the assembler, then running it. If this does not work, unlink other resident modules before trying again. OS/9 attempts to load some programs twice into memory under certain conditions; however, the CSC Assembler is so large that this is not possible.

CONTROLLING THE CSC ASSEMBLER

There are several option switches, as noted above, which assist the user in controlling the CSC Assembler's actions from the command line. The option fields must be preceded by "+" or "-", although there is no distinction between the two characters, and there is no distinction between upper and lower case in the option letters.

The same options may be set from the OPT directive, using the directive parameters specified in parentheses in the table below. The last column of the table also provides the default value of the options, as the first parameter.

opt	description	OPT parameters
B	Suppress creation of object file	(BIN/NOB/B/-B)
C	List all conditional statements	(NOC/CON/-C/C)
D	List assembly during phase 1	(NOD/DEB/-D/D)
G	List only first line of FCC, DEFM, DB, ...	(GEN/NOG/G/-G)
L	Suppress assembler listing	(LIS/NOL/L/-L)
O	Use separate program and data counters	(NOO/OS9/-O/O)
P	Suppress pagination of assembler listing	(NOP/PAG/-P/P)
S	Generate symbol table listing	(SYM/NOS/S/-S)
U	Suppress symbol name case distinction	(NOU/UPC/-U/U)
W	Truncate listing at column 132, not 80	(NOW/WID/-W/W)

The formatted assembler listing is normally sent to the originating terminal. To create a printer or disk file representing the assembler listing, the terminal output must be redirected in a manner appropriate to the operating system under which the CSC Assembler is being run.

CSC ASSEMBLER DESCRIPTION

The CSC Assembler accepts standard Motorola, MOS, Zilog, Intel, RCA, and other free-format source language statements. Each assembler language statement may contain ASCII characters with hex values between $20 and $7E, inclusive, and must be followed by a carriage return or new line, depending upon the system. Each tab character is interpreted as a single space.

Four fields on the source line are recognized by the CSC Assembler. These consist of the LABEL, the OPERATOR, the OPERAND, and the COMMENT fields, separated by one or more space or tab characters. Tab characters are considered as space characters for field delimiters and for character strings.

The format just described may be represented symbolically as follows:

[label[:]] [operator] [operand] [comment]

The CSC Assembler attempts to format its output listing according to this format. In case syntax errors are encountered, part of the operand field may be output at the beginning of the comment field. This may be helpful in determining the point of the error.

The contents of each of these field are described below.

LABEL FIELD

The label must not be preceded by spaces, unless it is immediately followed by a colon. A colon, if present, is assumed to be a delimiter, and is not considered to be a part of the label itself. The term "label" in this manual refers to the contents of a non-empty label field, not including the optional colon.

If the label field is present, is not preceded by spaces, and starts with '*', ';', or '!' (for Z8), the line is assumed to contain only a comment, which is listed but is not otherwise processed by the assembler.

The label must be unique, unless every occurrence of it is with the SET directive. Except for the EQU, SET, and similar directives, the value of a label is set to the value of the program or data counter at the beginning of the line.

The label should begin with a letter, and must be composed of the characters 'A' - 'Z', 'a' - 'z', '1' - '9', '$', '.', '_', in which upper and lower case letters are unique, unless the UPC option is chosen on the command line or OPT directive.

No label should be the same (case considered unimportant) as a machine register designation or instruction for a specific microprocessor or an assembler directive, nor may a label have the same name as an arithmetic operator, for those assembler languages (Zilog, RCA, and Intel) which use alphabetic arithmetic operator names. If it is necessary to use a label which is the same as a machine instruction or assembler directive, its definition may be followed by a colon.

The label (and optional colon) must be followed by a space or tab character, if an operator follows the label on the line.

The label may be of any length, but only the first ten characters are significant, and only the first eight characters are displayed in the label field on the listing.

OPERATOR FIELD

The operator field follows most of the rules for the label field, with the exceptions that it may be preceded by a period, that it may be composed of '*=' or '=' (for compatibility with the MOS assembler) or ':=' (for compatibility with the Z8 assembler), and that letter case is not significant.

The contents of the operator field may be omitted, in which case no operand or comment fields are allowed in the Motorola and MOS formats. If the label field is not present on a line, the operator field may be preceded by zero or more spaces.

The CSC 6801 Assembler recognizes operators coded in the form 'lda a' or in the form 'ldaa', but the CSC 6809 Assembler requires 6800 and 6801 simulated operators to be coded in the latter form.

OPERAND FIELD

Operands are considered as composed of one or more character strings and expressions, separated by commas. Spaces and tab characters outside of quoted strings are allowed in the Zilog, RCA, and Intel formats, but terminate the operand field in the Motorola and MOS formats. ';' normally separates the operand and comment fields in the Zilog, RCA, and Intel formats, as does '!' in the Z8 format.

Operands are evaluated for the expression type and value and the addressing mode used. Following the 6809 notation, if the operand field is preceded by '<', direct or zero page addressing or 8-bit indexing mode is forced, and, if the operand field is preceded by '>', extended or absolute addressing or 16-bit indexing mode is forced, for Motorola and MOS assembler formats.

8048 and 8051 registers must be referenced as rn, rnn, or rnnn, where n represents a digit.

In case an instruction or directive requires no operand, any operand field is considered part of the comment field.

COMMENT FIELD

The comment field is optional, and is composed of the remainder of the characters on the line after the operator and operand fields. ';' normally starts the comment field in the Zilog, RCA, and Intel formats, as does '!' in the Z8 format.

A total line length of 135 characters is allowed.

EXPRESSIONS

Expressions consist of combinations of symbols, constants, program counters, and data counters, separated by operators. Arithmetic is done using 16-bit unsigned-integer precision.

In the Motorola and MOS formats, expressions may not contain spaces and tabs except within quoted strings. An expression is terminated when a space, comma, tab, carriage return, new line, or other unexpected character is found.

In the Zilog, RCA, and Intel formats, spaces and tabs are ignored except for being used as as internal delimiters. An expression is terminated when ',', ';', '!' (for Z8), carriage return, new line, or other unexpected character is found.

Following is a summary of the formats for constants:

Base	Prefix	Suffix	Characters	
Decimal	none	none	0-9	
Decimal	none	D	0-9	
Decimal	%(10)	none	0-9	(Z8)
Binary	%	B	0-1	(not Z8)
Binary	%(2)	none	0-1	(Z8)
Octal	@	O,Q	0-7	
Octal	%(8)	none	0-7	(Z8)
Hexadecimal	$	H	0-9,A-F,a-f	
Hexadecimal	%	none	0-9,A-F,a-f	(Z8)
Hexadecimal	%(16)	none	0-9,A-F,a-f	(Z8)
ASCII-1	'	none	sp-~	
ASCII-1	'	'	sp-~	
ASCII-2	"	"	sp-~	
n, n<=10	%(n)	none	0-(n-1)	(Z8)

A repeated quote character, the same as the delimiter, is interpreted as one occurrence of the delimiter character within the string. Quoted strings are limited to one or two characters in length within expressions, but have no such length limits as character strings within directives.

Below is a summary of the arithmetic operators supported by the CSC Assembler. They are grouped in descending order of precedence. Binary operators of equal precedence are evaluated left to right. Unary operators of equal precedence are evaluated right to left. The relational operators provide a zero value for false and a nonzero value for true. The operators marked with "(*)" are usable only in the Zilog, RCA, and Intel formats, and should be separated from adjacent operators and symbols with spaces. The other arithmetic operators are usable in all formats.

Parentheses should be freely used to reduce the chance that the assembler will interpret an expression in a different manner from that intended. However, redundant surrounding parentheses may cause mis-interpretation of expressions in the MOS and Zilog assembler formats. Expressions may be parenthesized up to twenty levels.

Operation	Operator	Format
grouping	()	(value)
unary plus	+	+value
unary minus	-	-value
unary not	not	not value (*)
unary not	!	!value (not Z8)
unary not	^	^value
high byte	high	high value (*)
high byte	{	{value

```
low byte          low     low value (*)
low byte          }       }value

multiply          *       value*value
divide            /       value/value
modulus           mod     value mod value (*)
modulus           \       value\value
shift right       shr     value shr count (*)
shift right       >>      value>>count
shift left        shl     value shl count (*)
shift left        <<      value<<count

add               +       value+value
subtract          -       value-value

equal             eq      value eq value (*)
equal             =       value=value
less than         lt      value lt value (*)
less than         <       value<value
greater than      gt      value gt value (*)
greater than      >       value>value
less or equal     le      value le value (*)
less or equal     <=      value<=value
less or equal     =<      value=<value
greater or equal  ge      value ge value (*)
greater or equal  >=      value>=value
greater or equal  =>      value=>value
not equal         ne      value ne value (*)
not equal         <>      value<>value
not equal         ><      value><value

and               and     value and value (*)
and               &       value&value

or                or      value or value (*)
or                |       value|value
or                !       value!value   (not Z8)
xor               xor     value xor value (*)
xor               ~       value~value
```

Symbols are program or data labels, as described above.

The current program counter value is represented by the character '*'. The current data counter is defined by the character '.' after a MOD directive or if the OS9 option is chosen, and the character '*' otherwise.

OBJECT CODE FORMAT

The CSC Assembler generates object code files using the Motorola S-text hex file format. Although this is not the format used by many systems directly, it is very portable and avoids the peculiarities of specific systems. Programs are provided or are available for most common systems to convert S-text files to the binary format required for specific operating systems and to Intel hex format.

The user is responsible for avoiding the generation of files which may not be loaded on the systems of interest; for instance, some operating systems do not support dis-contiguous program segments, and some systems, such as OS9 and UNIFLEX on the 6809, require module header and trailer sequences.

An S-text hex file is formatted by the CSC Assembler as follows:

```
S1        Start of record
xx        Number of bytes to follow, in hex
xxxx      Address at which to place following data, in hex
xx..xx    Data bytes, in hex
xx        Ones-complement checksum of preceding hex bytes
cr        Carriage return or new line
```

An S-text hex file is terminated by an S9 record, which may contain an optional transfer address. This record is formatted as follows:

```
S9        Start of record
xxxx      Optional transfer address
cr        Carriage return or new line
```

The CSC Assembler generates records defining the names and addresses of all symbols defined in the program, by default. These record are formatted as follows:

```
SX        Start of record
xxxx      Symbol address
aaaaaaaa  Symbol name
cr        Carriage return or new line
```

This information may be used by symbolic debugging programs, or it may be sorted by the user for a label definition cross-reference. It should be ignored by the programs used to convert the S-text hex file format to other formats.

CONDITIONAL ASSEMBLY

Assembler language programs may be structured using conditional assembly directives. A very convenient use of conditional assembly capabilities is for customizing programs to a set of parameters, since the parameters may often be used to control the assembly.

Conditional assembly differs from 'IF' statements in BASIC and other languages in that the conditional expression is evaluated at assembly time, not at run time, and the source language statements in the range of the conditional assembly are either included or excluded from the assembly at that point. In BASIC, program statements are always included in the program, but are dynamically executed or skipped depending upon run-time conditions.

Further information on conditional assembly may be found under the IFxx, IFP1, IFP2, and RPT directives.

MODULAR SOURCE LANGUAGE

The LIB and USE directives may be used to introduce additional source language files into a file being assembled. They are very useful for large programs or for source subroutine libraries, and provide many of the capabilities at the source language level normally provided by a link editor at the object language level.

They assist in the use and development of structured modular assembly source language programs. LIB and USE have the same meaning and are simply alternative names for the text inclusion function.

Further information on modular source language may be found under the LIB and USE directives.

CSC ASSEMBLER OPERATORS AND OPERANDS

The CSC Assembler implements essentially the standard microprocessor manufacturer's operator and directive lists and formats.

The appropriate microprocessor manufacturer's software manuals and reference cards should be used as references for the CSC Assembler when used to assemble programs for a given target microprocessor.

The introduction to this manual covers the various assembler languages handled by the CSC Assembler and should be referred to for information on how the CSC Assembler processes specific assembler languages.

The next section of this manual describes the set of directives implemented by the CSC Assembler. This is a combination of the standard microprocessor manufacturer's assembler directives for many popular microprocessors.

Following is a list of 6800/6801 and convenience operators recognized by the CSC 6809 Assembler, in addition to the standard Motorola set of 6809 instructions. In addition, the CSC 6809 Assembler accepts the register designations "P", "PC", and "PCR", as well as "DP" and "Z", interchangeably, to assist in handling programs written for certain non-standard implementations of 6809 assemblers.

```
     operator definition      comments

      aba                     6800/6801
           pshs b
           adda ,s+
      asld                    6801
           aslb
           rola
      bec  ...                convenience (branch if error clear)
           bcc ...
      bes  ...                convenience (branch if error set)
           bcs ...
      cba                     6800/6801
           pshs b
           cmpa ,s+
      clc                     6800/6801
           andcc #$fe
      clf                     convenience (clear firq mask)
           andcc #$bf
      cli                     6800/6801
           andcc #$ef
      clrd                    convenience (clear d register)
           clra
           clrb
      clv                     6800/6801
           andcc #$fd
      clz                     convenience (clear zero flag)
           andcc #$fb
      cpx  ...                6800/6801
           cmpx ...
      des                     6800/6801
           leas -$01,s
      dex                     6800/6801
           leax -$01,x
      ins                     6800/6801
           leas $01,s
      inx                     6800/6801
           leax $01,x
      lbec ...                convenience (long branch if error clear)
```

```
     lbcc ...
lbes ...                convenience (long branch if error set)
     lbcs ...
ldaa ...                6800/6801
     lda ...
ldab ...                6800/6801
     ldb ...
ldad ...                6801
     ldd ...
lsrd                    6801
     lsra
     rorb
negd                    convenience (negate d register)
     coma
     comb
     subd #$ffff
oraa ...                6800/6801
     ora ...
orab ...                6800/6801
     orb ...
psha                    6800/6801
     pshs a
pshb                    6800/6801
     pshs b
pshx                    6801
     pshs x
pula                    6800/6801
     puls a
pulb                    6800/6801
     puls b
pulx                    6801
     puls x
reset                   convenience (reset processor)
     fcb $3e
rhf                     convenience (halt and catch fire)
     fcb $14
sba                     6800/6801
     pshs b
     suba ,s+
sec                     6800/6801
     orcc #$01
sef                     convenience (set firq mask)
     orcc #$40
sei                     6800/6801
     orcc #$10
sev                     6800/6801
     orcc #$02
sez                     convenience (set zero flag)
     orcc #$04
staa ...                6800/6801
     sta ...
stab ...                6800/6801
     stb ...
stad ...                6801
     std ...
tab                     6800/6801
     tfr a,b
     tsta
tap                     6800/6801
     tfr a,cc
tba                     6800/6801
     tfr b,a
```

```
     tsta
tpa                     6800/6801
     tfr cc,a
tstd                    convenience (test d register)
     std -2,s
tsx                     6800/6801
     tfr s,x
txs                     6800/6801
     tfr x,s
wai                     6800/6801
     cwai #$ff
```

CSC ASSEMBLER DIRECTIVES

The CSC Assembler supports most of the standard Motorola, MOS, Zilog, Intel, RCA, and other assembler directives. They are summarized below (in alphabetical order) and are described in detail (in logical groupings) in this section. For MOS and Microsoft assembler compatibility, any of the assembler directives below may optionally be preceded by a period.

directive	description
$ABS	set program counter or data counter (Z8)
$REL	set program counter (Z8)
*=	set program or data counter
*EJECT	start next line on next page (Zilog, Intel, RCA)
*HEADING	specify page subheading (Zilog, Intel, RCA)
*INCLUDE	divert input to source library file (Zilog, Intel, RCA)
*LIST ON	restore listing (Zilog, Intel, RCA)
*LIST OFF	stop listing (Zilog, Intel, RCA)
:=	assign a value to a symbol (Z8)
=	assign a value to a symbol
ASEG	set program counter or data counter
BLK	reserve memory bytes (Zilog, Intel, RCA)
BYTE	form constant byte
CHAR	form constant characters
COND	begin conditional assembly group (Zilog, Intel, RCA)
CSEG	set program counter
DB	form constant bytes (Zilog, Intel, RCA)
DBYTE	form constant double byte
DD	form constant double byte (Zilog, Intel, RCA)
DEFB	form constant bytes (Zilog, Intel, RCA)
DEFD	form constant double byte (Zilog, Intel, RCA)
DEFL	reassign a value to a symbol (Zilog, Intel, RCA)
DEFM	form constant message (Zilog, Intel, RCA)
DEFS	form constant string (Zilog, Intel, RCA)
DEFT	form constant text (Zilog, Intel, RCA)
DEFW	form reversed double byte (Zilog, Intel, RCA)
DM	form constant message (Zilog, Intel, RCA)
DS	form constant string (Zilog, Intel, RCA)
DSEG	set data counter
DT	form constant text (Zilog, Intel, RCA)
DW	form reversed double byte (Zilog, Intel, RCA)
ELSE	reverse conditional assembly sense
EMOD	generate OS9 module trailer
END	provide optional transfer address
ENDC	end conditional assembly group
ENDIF	end conditional assembly group
EQU	assign a value to a symbol
ERR	generate error message

```
EXTERNAL  accepted, but ignored (Zilog, Intel, RCA)
FCB       form constant byte
FCC       form constant characters
FCS       form constant string
FCT       form constant text
FDB       form double byte
FI        end conditional assembly group (Z8)
GLOBAL    accepted, but ignored (Zilog, Intel, RCA)
IFxx      begin conditional assembly group
INCLUDE   divert input to source library file (Zilog, Intel, RCA)
LIB       divert input to source library file
LIST      restore listing (Zilog, Intel, RCA)
LOAD      extend load instruction (RCA)
MOD       generate OS9 module header
NAM       specify page heading
NAME      specify page heading (Zilog, Intel, RCA)
OPT       modify assembler options
ORG       set program or data counter
OS9       generate OS9 call (6809 only)
PAG       start next line on next page
PAGE      start next line on next page (non-Zilog, Intel, RCA)
PAGE      adjust to next logical page (Zilog, Intel, RCA)
PG        start next line on next page
REG       generate push/pull register list (6809 only)
RES       reserve memory bytes
RMB       reserve memory bytes
RZB       reserve and zero memory bytes
RPT       repeat next line a number of times
SET       reassign a value to a symbol (not Z-80)
SPC       insert blank line in the output listing
STTL      specify page subheading
SUBTTL    specify page subheading (Zilog, Intel, RCA)
TEXT      form constant characters (Zilog, Intel, RCA)
TITLE     specify page heading (Zilog, Intel, RCA)
TTL       specify page heading
WORD      form double byte (reversed for all except RCA)
USE       divert input to source library file
XLIST     stop listing (Zilog, Intel, RCA)
ZERO      reserve and zero memory bytes
```

DB, DD, DW, DEFB, DEFD, DEFW, BYTE, DBYTE, FCB, FDB, WORD

All of these directives expect one or more expressions separated by commas and generate the corresponding object code, starting with the current value of the program counter, and adjust the program counter to correspond to the number of bytes of object code generated. The BYTE and FCB directives restrict these expressions to eight bits and generate one byte per expression. The DW, DBYTE, DEFW, FDB, and WORD directives generate two bytes per expression, in high-byte, then low-byte order, except for RCA, in which case they generate the bytes in the opposite order. adjacent commas are interpreted as if a zero were between them, and a leading comma is interpreted as if a zero preceded it.

The format of these directives is as follows:

[label] db expression,...,expression
[label] dd expression,...,expression
[label] dw expression,...,expression
[label] defb expression,...,expression
[label] defd expression,...,expression
[label] defw expression,...,expression
[label] byte expression,...,expression
[label] dbyte expression,...,expression

[label] fcb expression,...,expression
[label] fdb expression,...,expression
[label] word expression,...,expression

CHAR, DB, DEFB, DEFM, DEFT, DM, DT, FCC, FCS, FCT, TEXT

All of these directives expect one or more character strings and expressions to be placed into memory starting with the current value of the program counter. The strings must be delimited by characters which may not appear as operators or in numbers, such as "", """, "#", "?", ".", ":", etc., and each beginning delimiter must match the ending delimiter. If a double adjacent occurrence of a delimiter is found within a character string, it is interpreted as a single occurrence of that character within the string.

CHAR, DB, DEFB, DEFM, DM, FCC, and TEXT are interpreted identically. FCS is similar to the other directives in this group except that the high-order bit is set on the last character generated. DEFT, DT, and FCT are similar to the other directives except that the the length of the string (in binary) is placed into the character before the string, and the string is offset by one character towards higher addresses.

All the directives allow 8-bit expressions to be mixed with the strings to specify character values outside of the range of valid characters which may be placed within strings ($20 thru $7E). If more than one string or expression are coded, they must be separated by commas. Two adjacent commas are interpreted as if a zero were between them, and a leading comma is interpreted as if a zero preceded it, but a trailing comma is ignored.

The format of these directives is as follows:

[label] char "text",...,expression,...
[label] db "text",...,expression,...
[label] defb "text",...,expression,...
[label] defm "text",...,expression,...
[label] deft "text",...,expression,...
[label] dm "text",...,expression,...
[label] dt "text",...,expression,...
[label] fcc "text",...,expression,...
[label] fcs "text",...,expression,...
[label] fct "text",...,expression,...
[label] text "text",...,expression,...

BLK, DEFS, DS, RES, RMB

These directives cause the program or data counter to be incremented by a specified 16-bit amount. The data counter is incremented if the OS9 option is set, and the program counter is incremented if the OS9 option is reset.

The format of these directives is as follows:

[label] blk expression
[label] defs expression
[label] ds expression
[label] res expression
[label] rmb expression

RZB, ZERO

The RZB and ZERO directives generate a specified number (0 to 255) of binary zeroes in the object program and increment the program counter by the corresponding amount.

The format of these directives is as follows:

[label] rzb expression
[label] zero expression

DEFL, EQU, REG, SET, :=, =

The DEFL, EQU, REG, and = directives assign a value to a label. The DEFL and SET directives assign or reassign a value to a label. The SET directive is not available in the Z-80 assembler, since it uses SET to designate a machine instruction. The := directive may be used only in the Z8 assembler language.

Unless every occurrence of the assignment of a value to a label is with the SET or DEFL directive, the assembler will flag every occurrence as a multiple definition, unless the value assigned on every definition is identical. If the value assigned to a label with an EQU, REG, =, or := directive varies from pass 1 of the assembler to pass 2, the assembler will note a phasing error.

The REG declarative (valid only on the 6809) differs from the others in this group in that it expects a 6809 register list and generates the mask corresponding to the push/pull instruction postbyte. The 6809 PSHS, PULS, PSHU, and PULU instructions are allowed by the CSC 6809 assembler to have immediate operands, in order to use such separately-defined masks. The 6809 EXG and TFR instructions are also allowed to have immediate operands by the CSC 6809 Assembler, but the mask defined by the REG directive should not be used with them, as the bit positions are different.

The format of these directives is as follows:

label = expression
label := expression
label defl expression
label equ expression
label reg register-list
label set expression

$ABS, $REL, ASEG, CSEG, DSEG, ORG, *=

These directives set the program or data counter to a specified value. The CSEG and $REL directives sets the program counter and the DSEG directive sets the data counter. The $ABS, ASEG, ORG, and *= directives set the program counter, unless the OS9 option is set, in which case they set the data counter. The DSEG directive also causes '.' in expressions to represent the current value of the data counter, and the other directives cause '.' to represent the current value of the program counter, unless the OS9 option is set. The $ABS and $REL directives may be used only in the Z8 assembler language.

The default value of both counters is zero, and all programs have an assumed ' org 0' at their beginning. A label on one of these directives receives the value of the program or data counter before it is modified by the directive. This facility is used primarily when using the MOS 6502 assembler in statements of the form "label *= *+x" to reserve memory space. The RMB, DS, and similar directives are normally used for this purpose.

The format of these directives is as follows:

[label] $abs expression
[label] $rel expression
[label] *= expression
[label] aseg expression
[label] cseg expression
[label] dseg expression
[label] org expression

PAGE

For the Zilog, Intel, and RCA formats, PAGE causes the program or data counter to be adjusted to the beginning of the next logical page of memory (with address ending with hex 00). If a label is coded, it receives the address of the adjusted page.

The format of this directive is as follows:

```
[label] page
```

END

The END directive has historically been used to indicate the end of the input source file to an assembler. The CSC Assembler uses the END statement only to provide an optional transfer address for those operating systems which support them. If no transfer address appears, no comments should appear on the END directive (in Motorola and MOS formats).

The format of this directive is as follows:

```
[label] end [expression]
```

LOAD

The LOAD directive is used in the RCA 1805 assembler to generate an extended register load operation. It has two operands, separated by commas. The first is expected to be a register designator or expression with value 0 to 15. The second is expected to be a 16-bit expression.

The equivalent code generated is indicated below.

```
ldi high(expr)
phi reg
ldi low(expr)
pho reg
```

The format of this directive is as follows:

```
[label] load register,expression
```

NAM, NAME, STTL, SUBTTL, TITLE, TTL, *HEADING

These directives are used to provide a heading printed at the top of each page if the PAG option is set. No distinction is made between the NAM, NAME, TITLE, TTL, and *HEADING directives. The STTL and SUBTTL directives are listed but have no other meaning. The maximum length of the title is fifty characters.

The format of these directives is as follows:

```
nam [heading]
name [heading]
sttl [heading]
subttl [heading]
title [heading]
ttl [heading]
*heading [heading]
```

PAG, PAGE, PG, *EJECT

PAGE has a different meaning for Zilog, Intel, and RCA formats from the other formats, and is described elsewhere in this manual. For PAG, PG, *EJECT, and for PAGE in the other formats, if the PAG option has been set, the next line of the output listing will be placed at the top of the next page. Vertical formatting of the output listing is performed with form feeds.

The format of these directives is as follows:

```
pag
page
pg
*eject
```

SPC

The SPC directive inserts one blank line in the output listing.

The format of this directive is as follows:

```
spc
```

OPT

The OPT directive modifies the same list of assembler options which may be modified on the command line. The codes to be used as operands of the OPT directive are separated by commas and have all been described in an earlier section.

The format of this directive is as follows:

```
opt opnd,...,opnd
```

LIST, *LIST ON

The LIST and *LIST ON directives are equivalent to the OPT LIS directive.

The format of these directives is as follows:

```
list
*list on
```

XLIST, *LIST OFF

The XLIST and *LIST OFF directives are equivalent to the OPT NOL directive.

The format of these directives is as follows:

```
xlist
*list off
```

ERR

The ERR directive generates an error. It is normally used to indicate that invalid options have been chosen.

The format of this directive is as follows:

[label] err [comments]

COND, IFxx

The COND and IFxx directives cause the lines up to the next matching ENDC, ENDIF, FI, or ELSE directive to be included only if the expression bears some specified relationship to a zero value. No forward references are allowed in the expression. COND and IFNE are processed in an identical manner.

If a matching ELSE directive is found before the matching ENDC, ENDIF, or FI directive, the test condition is reversed for the succeeding lines. IFxx - ELSE - ENDIF groups may be nested up to fifteen levels.

The syntax of these directives is as follows:

cond expression
ifxx expression

in which 'xx' may be one of the following:

suffix	condition for inclusion
(none)	not equal zero
C	equal zero
EQ	equal zero
F	equal zero
GE	not less zero
GT	greater zero
LE	not greater zero
LT	less zero
N	equal zero
NE	not equal zero
T	not equal zero

IF1, IF2, IFP1, IFP2

The IF1, IF2, IFP1, and IFP2 declaratives allow for assembly of the included lines only on pass 1 or on pass 2, respectively. They require no expressions. The included or excluded lines for these directives are terminated by ELSE, ENDIF, and ENDC directives.

The IF1 and IFP1 directives may be used to cause source libraries containing only constant EQU and SET directives to be read only on the first pass, speeding up the assembly process.

The syntax of these directives is as follows:

if1
if2
ifp1
ifp2

IFDEF, IFNDEF

The IFDEF and IFNDEF declaratives allow for assembly of the included lines only if a symbol has already been defined or not defined, respectively. They are similar to the IFxx directives in interpretation and usage.

The syntax of these directives is as follows:

ifdef symbol
ifndef symbol

ELSE

The ELSE directive must match a corresponding IFxx directive, and reverses the state of the conditional assembly, unless the entire group is conditionally excluded.

The syntax of this directive is as follows:

else

ENDC, ENDIF, FI

The ENDC, ENDIF, and FI directives must match a corresponding IFxx directive, and restore the state of the conditional assembly to that before the corresponding IFxx or IFPx directive. FI may be used only in the Z8 assembler language.

The format of these directives is as follows:

endc
endif
fi

RPT

The RPT directive provides for the exclusion of or repetition of the next line in the source file a specified number of times. No forward references are allowed in the expression, and the value is taken modulo 256. RPT directives may not be nested.

The syntax of this directive is as follows:

[label] rpt expression

INCLUDE, LIB, USE, *INCLUDE

The INCLUDE, LIB, USE, and *INCLUDE directives may be used to introduce source language files into a file being assembled. They are usually very useful for large programs or for source subroutine libraries, and provide many of the capabilities at the source language level normally provided by a link editor at the object language level. They assist in the use and development of structured modular assembly source language programs. The INCLUDE, LIB, USE, and *INCLUDE declaratives have the same interpretation.

The syntax of these directives is as follows:

[label] include file-name
[label] lib file-name
[label] use file-name
[label] *include file-name

The file-name must obey all rules of the operating system under which the program is being assembled. No suffix is assumed.

MOD

The MOD directive generates an OS9 module header and initializes the CRC counter for the corresponding module trailer generated by the EMOD directive. It automatically sets the OS9 assembler option to specify the maintenance of separate data and program counters, as required in OS9 assembler programs. It should not be preceded by an ORG directive unless the OS9 option is on the command line.

The MOD directive expects an operand composed of four or more expressions separated by commas. These expressions are used in the construction of an OS9 module header. Review the "Microware OS9 System Programmer's Manual" for further information on the MOD and EMOD directives.

The format of this directive is as follows:

```
[label] mod program-length,name-offset,type/lang,
        attribute/revision,[execution-offset,
        permanent storage size,...]
```

EMOD

The EMOD directive generates an OS9 module trailer corresponding to an OS9 module header. The 24-bit CRC of the module is output as object code correctly only under OS9.

The format of this directive is as follows:

```
[label] emod
```

OS9

The OS9 directive generates the code to perform OS9 calls for the 6809. It requires one 8-bit expression as an operand. It generates the following equivalent 6809 instruction sequence:

```
swi2
fcb expression
```

The format of this directive is as follows:

```
[label] os9 expression
```

UTILITY PROGRAMS

Miscellaneous utility programs are normally included with the assembler programs. 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 are usually included to provide the capabilities of translating object file formats.

ERROR MESSAGES

The CSC Assembler issues error messages to indicate problems with processing the source program. Following is a complete list of the error messages and a brief explanation of each.

Can't open object
: assembler unable to create output object file

Can't open source
: assembler unable to find input source file

xxxxxxxx Could not open
: assembler unable to find indicated input library file

Error
: ERR directive

Internal error
: badly-formatted statement or assembler error

Invalid addressing mode
: inappropriate addressing mode for instruction

Invalid conditional structure
: unmatched ELSE or ENDC/ENDIF directive or bad nesting

Invalid forward reference
: unresolved or illegal forward reference in operand

Invalid label
: use of label on a directive which does not allow a label, lack of a label on a directive which requires a label, etc.

Invalid numeric constant
: illegal characters in a numeric constant

Invalid operator
: contents of the operator field not recognized

Invalid quoted string
: too-long quoted string ('x or 'x' or "xx") in an expression

Invalid register specification
: non-register in a register list

Library nest error
: too-deep nesting of library files

Phasing error
: symbol has different value on the second pass of assembler than it had on first pass

Symbol table overflow
: assembler is unable to add another symbol

Symbol table space full
: assembler is unable to add another symbol

Syntax error
: badly-formed expression or illegally-formatted operand

Unbalanced parentheses
: an expression has unmatched parentheses

Undefined symbol
: a symbol never appears as a label

Value out of range
: the value of an expression exceeds its limit