CAS11 ASSEMBLER SUMMARY

1. INTRODUCTION

This summary of the Motorola MC68HC11 assembler (cas11.exe) was adapted from *Freeware 8-bit Cross Assemblers User's Manual* edited by Kevin Anderson.

Programs written in assembly language consist of a sequence of source statements. Each source statement consists of a sequence of ASCII characters ending with a carriage return.

2. SOURCE STATEMENT FORMAT

Each source statement may include up to four fields: a label (or '*' for a comment line), an operation (instruction mnemonic or assembler directive), an operand, and a comment.

Label Field
The label field occurs as the first field of a source statement. The label field can take one of the following forms:

1. An asterisk (*) as the first character in the label field indicates that the rest of the source statement is a comment. Comments are ignored by the Assembler, and are printed on the source listing only for the programmer's information.
2. A ***whitespace*** character (blank or tab) as the first character indicates that the label field is empty. The line has no label and is not a comment.
3. A symbol character as the first character indicates that the line has a label. Symbol characters are the upper or lower case letters a-z, digits 0-9, and the special characters dollar sign ($), period (.), question (?) and underscore (_). Symbols consist of one to at least 20 significant characters, the first of which must be alphabetic or the special characters period (.), question (?) or underscore (_). Upper and lower case letters are distinct.

A symbol may occur only once in the label field. If a symbol does occur more than once in a label field, then each reference to that symbol will be flagged with an error.

With the exception of some directives, a label is assigned the value of the location counter of the first byte of the instruction or data being assembled. The value assigned to the label is absolute. Labels may optionally be ended with a colon (:). If the colon is used it is not part of the label but merely acts to set the label off from the rest of the source line. Thus the following code fragments are equivalent:

```
Fragment 1          Fragment 2
here:  deca         here   deca
       bne  here           bne  here
```

A label may appear on a line by itself. The assembler interprets this by setting the value of the label to the current value of the program counter.

The size of the symbol table is dependent upon available memory.

Operation Field
The operation field occurs after the label field, and must be preceded by at least one whitespace character. The operation field must contain a legal opcode mnemonic or an assembler directive. Upper case characters in this field are converted to lower case before being checked as a legal mnemonic. Thus 'nop', 'NOP', and 'NoP' are recognized as the same mnemonic. Entries in the operation field may be one of two types:

1. ***Mnemonic***: These correspond directly to the machine instructions. The operation code includes any register name associated with the instruction. These register names must not be separated from the opcode with any whitespace characters. Thus 'clra' means clear accumulator A, but 'clr a' means clear memory location identified by the label 'a'.
2. ***Directive/Pseudo-op***: These are special operation codes known to the Assembler which control the assembly process rather than being translated into machine instructions (see section 3).

Operand Field
The operand field's interpretation is dependent on the contents of the operation field. The operand field, if required, must follow the operation field, and must be preceded by at least one whitespace character. The operand field may contain a symbol, an expression, or a combination of symbols and expressions separated by commas.

The operand field of machine instructions is used to specify the addressing mode of the instruction, as well as the operand of the instruction. The following table summarizes the available operand field formats. NOTE: Angle brackets "<>" denote an expression is inserted. These syntax elements are present only for clarification of the format and are not inserted as part of the actual source program. All other characters are significant and must be used when required.)

Operand Format	Addressing Mode
no operand	Accumulator and Inherent
<expr>	Direct, Extended, or Relative
#<expr>	Immediate
<expr>,X	Indexed with X register
<expr>,Y	Indexed with Y register
<expr1> <expr2>	bit set or clear
<expr1> <expr2> <expr3>	bit test and branch

In the second format, the force size characters, '<' and '>', may be used before the expression to force direct or extended addressing respectively. The bit manipulation instruction operands are separated by spaces since the HC11 allows bit manipulation instructions on indexed addresses. Thus a ',X' or ',Y' may be added to the final two formats above to form the indexed effective address calculation. Except for the last two formats, spaces are otherwise not allowed in the operand field.

Expressions. An expression is a combination of symbols, constants, algebraic operators, and parentheses. The expression specifies a value which is to be used as an operand. Expressions may consist of symbols, constants, or the character '*' (denoting the current value of the program counter) joined together by an operator. Expressions are evaluated left to right and there is no provision for parenthesized expressions. Arithmetic is carried out in signed two's-complement integer precision using 16 bits.

Operators. The operators are as follows:

~	unary one's complement
+	add
-	subtract / unary minus
*	multiply
/	divide
%	remainder after division
&	bitwise AND
|	bitwise OR
^	bitwise EOR (exclusive or)
!<	shift left
!>	shift right

Symbols. Each symbol is associated with a 16-bit integer value which is used in place of the symbol during the expression evaluation. The asterisk (*) used in an expression as a symbol represents the current value of the location counter (the first byte of a multi-byte instruction).

Constants. Constants represent quantities of data that do not vary in value during the execution of a program. Constants may be presented to the assembler in one of five formats: decimal, hexadecimal, binary, or octal, or ASCII. The programmer indicates the number format to the assembler with the following prefix characters:

$	hexadecimal
@	octal
%	binary
'	ASCII

Unprefixed constants are interpreted as decimal. The assembler converts all constants to binary machine code and are displayed in the assembly listing as hex.

A decimal constant consists of a string of numeric digits (0-9). The value of a decimal constant must fall in the range 0-65535, inclusive. Hexadecimal constants consist of up to four hex characters (0-9 and A-F), preceded by a '$' prefix, and must be in the range $0000 to $FFFF. Binary constants consist of a maximum of 16 ones or zeros preceded by a '%' prefix. Octal constants consist of a maximum of six octal digits (0-7), preceded by a '@' prefix, and must be in the range @0 to @177777. A single ASCII character can also be used as a constant in expressions. ASCII constants are preceded by a single quote ('). Any character,

CAS11 ASSEMBLER SUMMARY

including the single quote, can be used as a character constant. If additional characters are given after the first, they are ignored. Examples of both valid and invalid constants are shown in the following table:

Type	Valid	Invalid	Reason Invalid
Decimal	12	123456	>5 digits
	12345	12.3	invalid character
Hex	$12	ABCD	no preceding '$'
	$ABCD	$G2A	invalid character
	$001F	$2F018	>4 characters
Octal	@17634	@2317234	>6 digits
	@377	@277272	out of range
	@177600	@23914	invalid character
Binary	%00101	1010101	missing % prefix
	%1	%10011000 101010111	>16 digits
	%10100	%210101	invalid digit
ASCII	'*	'VALID	too long

Comment Field
The last field of an Assembler source statement is the comment field. This field is optional and is only printed on the source listing for documentation purposes. The comment field is separated from the operand field (or from the operation field if no operand is required) by at least one whitespace character. The comment field can contain any printable ASCII characters. Blank lines are considered comment lines as are lines that begin with asterisk (*) in column one.

3. PSEUDO-OPS

<u>Assembly Control</u>

END	Ends assembly, sets transfer address
INCLUDE	Includes another source file
ORG	Origin program counter
OFFSET	Set S-record load offset

<u>Symbol Definition</u>

EQU	Equate symbol with value
SET	Temporary (changeable) equate

<u>Data Definition/Storage Allocation</u>

BSC/FILL	Initialize a block of memory to a constant
BSZ/ZMB	Block storage of zeros; single bytes
DB/FCB	Form constant byte
DW/FDB	Form constant double byte
DS/RMB	Reserve memory; single bytes
FCC	Form constant character string

<u>Listing Control</u>

EJECT/PAGE	Advance to top of next page
INDENT	Set listing indent column
OPT c/noc	Enable/disable cycle counting
OPT contc	Resume cycle counting
OPT l/nol	Print/don't print source listing from this point
OPT s/nos	Print/don't print symbol table
OPT x/nox	Print/don't print cross reference table
OPT nopag	Don't paginate listing
SBTTL/STTL	Set/reset subtitle line
TITLE/TTL	Set/reset title line

4. ASSEMBLER OUTPUT

The assembler output includes an optional listing of the source program and an object file in the Motorola S-Record format (`filename.s19`). The assembler will normally suppress the printing of the source listing, but this condition (and others) can be overridden via options supplied on the command line or within the source code via the OPT pseudo-op. Each line of the listing contains a reference line number, the address and bytes assembled, and the original source input line. If an input line causes more than 6 bytes to be output (e.g. a long FCC directive), additional bytes (up to 64) are listed on succeeding lines.

The assembly listing may optionally contain a symbol table and/or a cross reference table of all symbols appearing in the program. These are always printed at the end of the assembly listing if either the symbol table or cross reference table options are in effect. The symbol table contains the name of each symbol, along with its defined value. The cross reference table additionally contains the assembler-maintained source line numbers of both definition of and all references to every symbol.

5. ASSEMBLER INVOCATION

To run the assembler (cas11.exe), enter the following command line:

```
cas11 [-option1 -option2…] file
```

where `file` is the name of the source file you want to assemble. The source filename may have an extension to override the default `.asm` (however, do not use the extension `.S19` since that is the extension of the object file created by the assembler -- its creation would overwrite the source file when it is written to the disk!).

The options given on the command line are one or more of the following:

h,?	display command line help
c	enable cycle counting
l	enable output listing
s	generate a symbol table
x	generate a cross reference table

The minus sign preceding the option(s) must be preceded by a space. These options may also be indicated to the assembler by the use of the OPT directive within the source file.

The object file created is written to disk and given the name `filename.s19` where `filename` is taken from the source filename (minus extension) specified on the command line. The optional listing (if specified) along with errors and symbol tables are written to a `.lst` file. The errors are also displayed on the screen along with an assembly summary.

Command line examples:

#1: `cas11 prog1.asm`
runs the M68HC11 assembler on the source file `prog1.asm`. The object file would be written to `prog1.s19`, no listing would be produced and any errors would appear on the screen.

#2: `cas11 -l main.asm`
would assemble `main.asm`, generating the object output to `main.s19` and a listing to `main.lst`. The errors would also appear on the screen.

#3: `cas11 -lsc prog2 >prog2.sum`
assembles `prog2.asm` to `prog2.s19` and writes a listing file with cycles and symbol table along with any errors to `prog2.lst`. The assembly summary is redirected to `prog2.sum`.

6. ERROR MESSAGES

Error diagnostic messages are placed in the listing file just before the line containing the error. The format of the error line is:

Error in file_name, line_number: Description of error

or

Warning in file_name, line_number: Description of warning

Errors of the first type in pass one cause cancellation of pass two and no object file is generated. Warnings do not cause cancellation of pass two but are indications of a possible problem. Description messages are meant to be self-explanatory.

Some errors are classed as fatal and cause an immediate termination of the assembly. Generally this happens when there is a file access problem or an internal assembler error.

7. FILES

filename.asm	recommended source filename extension
filename.lst	default listing filename extension
filename.S19	S-record output file
STDOUT	summary and errors (use redirection for file)

8. PROBLEMS

The author would appreciate reports by e-mail on any problems, complaints or suggestions on this assembler to *sumey@cup.edu*.