ASSEMBLY LANGUAGE NOTES (AS11 Assembler for the 68HC11)

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.

SOURCE STATEMENT

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

A symbol starting in the first column is a label and may optionally be ended with a colon (:).

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 programming 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, period (.), dollar sign ($), and the underscore (_). Symbols consist of one to fifteen characters, the first of which must be alphabetic or the special characters period or underscore. All characters are significant and 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 program counter (PC) of the first byte of the instruction or data being assembled. The value assigned to the label is absolute. Labels may be ended with a colon (:), although the colon is not part of the label (it only 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 as "set the value of the label equal to the current value of the PC." For example:

```
Label   EQU   *
```

The symbol table has room for at least 2000 symbols of length 8 characters or less. Additional characters up to 15 are permissible at expense of decreasing the maximum number of symbols possible in the table.

Operation Field

The operation field occurs after the label field, and must be preceded by at least one white space 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.

Opcode (Mnemonic): These correspond directly to the machine instructions. Note that register names sometimes appear at the end of a mnemonic (e.g. `NEG`**`A`** or `STA`**`B`**) and therefore must not be separated by any whitespace characters. Thus, '`CLRA`' means clear accumulator 'A', but that '`CLR A`' means clear memory location 'A'.

Directive: These are special operation codes known to the assembler which controls the assembly process rather than being translated into machine instructions.

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 operand field for the M68HC11 processor.

Operand Format	Addressing Mode
no operand	Accumulator and Inherent
<expression>	Direct, Extended, or Relative
#<expression>	Immediate
<expression>,X or Y	Indexed with X or Y Register

NOTE: Parenthesis '()' signify optional elements and '<> 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.

Expression: An expression is a combination of symbols, constants, algebraic or logical operators, and parentheses. The expression specifies a value which is to be used as an operand. Expressions may consist of symbols or constants joined together by one of the following operators: `+ - * / % & | ^`. They are evaluated left to right and there is no provision for parenthesized expressions. Arithmetic is carried out in signed twos-complement integer precision (16 bits on the IBM PC).

Operator: The operators are the same as in C programming language:

+	add
-	subtract
*	multiply
/	divide
%	remainder after division
&	bitwise AND
|	bitwise OR
^	bitwise XOR (exclusive-or)

Symbol: 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 PC.

A symbol is a string of characters with a non-initial digit. The string of characters may be from the set:

```
[a-z][A-Z]_.[0-9]$
```

('`.`' and '`_`' count as non-digits). The `$' counts as a digit to avoid confusion with hexadecimal constants. All characters of a symbol are significant, with upper and lower case characters being distinct. The maximum number of characters in a symbol is currently set at 15.

The symbol table has room for at least 2000 symbols of length 8 characters or less.

Constant: A constant represents a quantity of data that does not vary in value during the executing of a program. Constants may be presented to the assembler in one of five formats: decimal, hex, binary, octal, or ASCII. The default format is decimal. The assembler converts all constants to binary and displays them in hex. The programmer indicates the number format to the assembler with the following prefixes:

'	ASCII
$	hexadecimal
@	octal
%	binary
(None)	decimal

Decimal constants consist of up to 5 valid digits (0-9) and must be in the range 0-65535, inclusive. Hex constants consist of up to four hex characters

(0-9 and A-F), are preceded by a $ and must be in the range $0000 to $FFFF. Binary constants consist of up to 16 ones and zeros preceded by a %. Octal constants consist of up to six valid numeric digits (0-7), are preceded by an @, and must be in the range @0 to @177777. A single ASCII character can be used as a constant in expressions. ASCII constants are preceded by a single quote (`'`). Any character, including the single quote, can be used as a character constant. If too many ASCII characters are given, the assembler assembles the first character and ignores the remainder.

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
Binary	%00101	1010101	missing %
	%1	%10011000 101010111	> 16 digits
	%10100	%210101	invalid character
Octal	@17634	@2317234	> 6 digits
	@377	@277272	out of range
	@177600	@23914	invalid character
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 operation is required) by at least one whitespace character.

A comment is any text following the operands for a given mnemonic up to the end of a line. A comment may also be either a line beginning with '*' or an empty line. Comments are ignored by the assembler, and are printed on the source listing only for programming information.

Continuations

If a line ends with a backslash (\) then the next line is fetched and added to the end of the first line. This continues until a line is seen which doesn't end in \ or until MAXBUF characters have been collected (MAXBUF ≥ 256).

AS11 ASSEMBLER

The assembler output includes an optional listing of the source program and an object file which is 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 program via pseudo opcodes (OPT). Each line of the listing contains a reference line number, the address and bytes assembled, and the original source input line.

The assembler for the M68HC11 is named `as11.exe`. Command line arguments specify the filenames to assemble.

Assembler Invocation

To run the assembler enter the following command line:

```
as11 file1 [file2] … [- option1 option2 …]
```

where `file1`, `file2`, etc. are the names of the source file(s) you want to assemble. The source filenames may have extensions but the assembler does not check for any particular extension (however, do not use the `.S19` extension 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 assembler accept options from the command line to be included in the assembly. These options are the following:

l	enable output listing.
nol	disable output listing (default).
cre	generate cross reference table.
s	generate a symbol table.
c	enable cycle count.
noc	disable cycle count.

If this method of passing commands to the assembler is used rather than the OPT pseudo opcode , a space should separate the minus sign from the last file name and the first option. For example:

```
as11 EX_FILE -l cre
```

This command assembles file `EX_FILE` with an output listing and a cross reference table.

The object file created is written to disk and given the name `filename.S19` where `filename` is the name of the first source file specified on the command line. Any errors and the optional listing (if specified) are written to the standard output (normally displayed on the screen). The listing and/or error messages may be saved to a file for later examination or printing by appending an i/o redirection ('>') command to the command line.

If multiple files are assembled, the 'S1' file will be placed under the first file's name (`file1.S19`).

The example command line

```
as11 myfile
```

runs the M68HC11 assembler on the source file `myfile`. The object file would be written to `myfile.s19` and any errors would appear on the screen. The example command line

```
as11 test1.asm test2.s -l
```

runs the HC11 assembler on the source files `test1.asm` and '`test2.s`'. The object file would be written to `test1.s19` and any errors and the assembly listing would appear on the screen. The example command line

```
as11 test1.asm test2.s -l cre s>test.lst
```

runs the HC11 assembler on the source files `test1.asm` and `test2.s`. The object file would be written to `test.s19`. A listing would be created followed by a cross-reference and symbol table which would all be written to the file `test.lst`.

The listing file contains the address and bytes assembled for each line of input followed by the original input line (unchanged, but moved over to the right some). 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 with no address preceding them.

Equates cause the value of the expression to replace the address field in the listing. Equates that have forward references cause Phasing Errors in Pass 2.

Error Messages

Error diagnostics are placed in the listing file just before the line containing the error. Format of the error line is:

`Line_number:` Description of error

or

`Line_number:` `Warning` --- Description of error

Errors of the first type in pass one cause cancellation of pass two. Warnings do not cause cancellation of pass two but should cause you to wonder where they came from. Error messages are meant to be self-explanatory.

If more than one file is being assembled, the file name precedes the error:

`File_name,Line_number:` Description of error

Finally, some errors are classed as fatal and cause an immediate termination of the assembly. Generally, these errors occur when a temporary file cannot be created or is lost during the assembly. Consult your local guru if this happens.

ASSEMBLY LANGUAGE NOTES (AS11 Assembler for the 68HC11)

Pseudo Opcodes

The OPT pseudo-opcodes allows the following operands:

nol	Turn off output listing
l	Turn on output listing (default)
noc	Disable cycle counts in listing (default)
c	Enable cycle counts in listing (clear total cycles)
contc	Re-enable cycle counts (don't clear total cycles)
cre	Enable printing of a cross reference table
s	generate a symbol table

Some of the more common pseudo-ops are not present. The below 4 pseudo-ops are recognized, but ignored:

SPC	Use blank lines instead
END	The assembly ends when there is no more input
TTL	Use `pr' to get headings and page numbers
NAM[E]	Did you ever use this one anyway?

Differences From Other Assemblers

- For indexed addressing, the comma is required before the register; e.g., `INC X` and `INC ,X` are not the same.
- Macros are not supported.
- Bit manipulation operands are separated by blanks instead of commas since the 68HC11 has bit manipulation instructions that operate on indexed addresses.
- The only pseudo-ops supported are:
  ORG, FCC, FDB, FCB, EQU, RMB, BSZ, ZMB, FILL, PAGE and OPT.

Files

filename.S19	S-record output file extension
STDOUT	listing and errors (use redirection for listing file)
Fwd_refs	Temporary file for forward references
filename.ASM	Suggested assembly language source file extension
filename.LST	Suggested list file extension

Implementation Notes

This is a classic 2-pass assembler. Pass 1 establishes the symbol table and pass 2 generates the code.

ASSEMBLER DIRECTIVES SUMMARY

Assembly Control

ORG	Origin program counter

Symbol Definition

EQU	Assign permanent value

Data Definition/Storage Allocation

BSZ	Block storage of zero; single bytes
FCB	Form constant byte
FCC	Form constant character string
FDB	Form constant double byte
FILL	Initialize a block of memory to a constant
RMB	Reserve memory; single bytes
ZMB	Zero memory bytes; same as BSZ

Listing Control

OPT	c	Enable cycle counting
OPT	cre	Print cross reference table
OPT	1	Print source listing from this point
OPT	nol	Inhibit printing of source listing from this point
OPT	s	Print symbol table
PAGE	<xxx>	Print subsequent statements on top of next page