Introduction

The HSW12ASM is a simple multi-pass assembler which has been written in Perl code. Some features of this assembler are:

- It handles 16MB of address space
- It generates linear and paged S-Record files
- It uses two program counters (linear and paged) to contol the S-Record output
- It exports source code symbols into the **HSW12 IDE**

This assembler is normally invoked from the **HSW12 IDE**, however it can be used stand alone with the command line interface **hsw12asm.pl**. **hsw12asm.pl** is invoked as follows:

```
perl hsw12asm.pl <src files> [-L <library paths>] [-D <defines: name=value or name>] [-S19|-S28]
     <src files>       source code files(*.s)
     <library paths>   directories to search for include files
     <defines>         assembler defines
     S19,S28           S-Record format
```

The following sections give some insight to the assembler's source code format and it's outputs.

Comments

All code following ";" to the end of the line is interpreted as a comment by the HSW12 assembler. Comments may also begin with an "*" if it is the first non-whitespace character in the line.

Expressions

Expressions consist of symbols, constants and operators. They are used as operands for the HC(S) opcodes and and the assembler pseudo opcodes.

Symbols

Symbols represent integer values.

User Defined Symbols

Symbols can be defined through various pseudo-opcodes or through the use of labels. A symbol name must comply to these rules:

- The symbol name must consist of alpha-numeric characters, and underscores (^[A-Z0-9_]+$)
- The symbol name must begin with a letter (^[A-Z])
- The symbol name may not contain any whitespaces
- The symbol name may not be any of the keywords: A, B, D, X, Y, SP, CCR, PC, TMP2, TMP3, UNMAPPED

Predefined Symbols

The HSW12 assembler knows a set of predefined symbols:

```
@  Represents the current value of the linear program counter
*  Represents the current value of the paged program counter
```

Automatic Symbol Extensions

The HSW12 assembler supports the automatic generation of symbol name extensions. If a symbol name ends with a "", this character will be substituted by the contents of the **LOC** counter variable. This counter may be incremented by the **LOC** pseudo-opcode.

Constants

Integer Constants are of the following format:

```
%...   binary constant      (^%[01]+$)
 ...   decimal constant     (^[0-9]+$)
$...   hexadecimal constant (^\$[0-9A-H]+$)
"..."  ascii strings        (^["']+["']$)
```

Operators

The HSW12 assembler supports the operaters that are listed below (from highest to lowest precedence). Expressions may be nested in parenthesis.

```
   &  bitwise AND
   |  bitwise OR
   ^  bitwise XOR
  >>  leftshift
  <<  rightshift
   *  multiplication
   /  integer division
   %  modulus
   +  addition
   -  subtraction
```

Labels

Labels assign the current value of the paged program counter to a symbol. The syntax is:

```
SYMBOL
```

or

```
SYMBOL:
```

(The symbol name must be the first characters in the line.)
To assign the current value of the linear program counter to a symbol, use the following syntax:

```
SYMBOL  EQU  @
```

Precompiler Directives

The HSW12 assembler knows the following precompiler directives:

- #DEFINE
- #UNDEF
- #IFDEF
- #IFNDEF
- #ELSE
- #ENDIF
- #INCLUDE
- #MACRO
- #EMAC

All precompiler directives must comply to the following syntax rules:

```
line starts with
a hash, directly
followed by the
directive
|
V
#<directive> <arg> <arg> ...
            ^     ^     ^
            |     |     |
            spaces,  tabs
```

#DEFINE

Sets an assembler define for conditional code compilation. All assembler defines will be exported into compiler symbols at the end of the precompile step.
"#DEFINE" requires two arguments:

1. a define name
2. a value the define is set to (optional)

To make the HSW12 assembler behave a little more like the AS12, all lables and pseudo-opcode symbol assignments will be considered as precompiler defines as well.

#UNDEF

Undefines an assembler define.
"#UNDEF" requrires one argument:

1. a define name

#IFDEF

Starts a section of conditional code. This code will only be compiled if the define is set.

#IFNDEF

Starts a section of conditional code. This code will only be compiled if the define is not set.

#ELSE

Ends a section of conditional code that has been initiated with **"#IFDEF"** or **"#IFNDEF"** and starts a new one that requires the opposite condition.

#ENDIF

End a section of conditional code.

#INCLUDE

Includes a source code file at the current position.

#MACRO

Starts a macro definition. This directive requires two arguments:

1. The macro name
2. The number of arguments which are to be passed to the macro

A macro definition ends with an #EMAC directive. Inside the macro, the strings "\1", "\2", ... will be replaced by the macro arguments. All lables will be defined in a local name space. Nested macro calls are possible.

Example:

```
#MACRO MAC0 2
       MAC1  \1
LOOP   CPD   0,\2
       BEQ   LOOP
#EMAC

#MACRO MAC1 1
LOOP   CPD   0,\1
       BNE   LOOP
#EMAC

       CPU   S12
       ORG   $0000
       MAC0  X, Y
```

Result:

```
??????         S12 CODE:                       CPU   S12
000000 0F4000                                  ORG   $0000
000000 0F4000 MACRO                            MAC0  X, Y
000000 0F4000 MACRO                            MAC1  \1                                                (MAC0)
000000 0F4000 AC 00                    LOOP    CPD   0,\1                                             (MAC0/MAC1)
000002 0F4002 26 FC                            BNE   LOOP                                              (MAC0/MAC1)
000004 0F4004 AC 40                    LOOP    CPD   0,\2                                             (MAC0)
000006 0F4006 27 FC                            BEQ   LOOP                                              (MAC0)
```

#EMAC

Ends a macro definition.

Pseudo-Opcodes

The following pseudo-opcodes are supported by the HSW12 assembler:

- ALIGN
- CPU
- DC.B (DB, FCB)
- DC.W (DW, FDW)
- DS.B (DS, RMB)
- DS.W (RMW)
- EQU
- FCC
- FCS
- FILL
- LOC
- ORG
- UNALIGN
- SETDP

All pseudo-opcodes must comply to the following syntax rules:

```
symbol name
must start at            arguments must
the begin of             be separated
the line                 by a comma
|                           |
V                           V
<symbol> <psudo-opcode> <arg>, <arg>, ...
        ^               ^
        |               |
        +-spaces, tabs-+
```

ALIGN

Increments both program counters until PC & mask == 0. If a second argument is given, then all memory locations in between are filled with the lower eight bit of this integer.
Syntax:

```
        ALIGN <mask>
```

or

```
        ALIGN <mask> <pattern>
```

CPU

Switches to a different opcode table. Supportd CPUs are:

- HC11 (untested)
- HC12
- S12
- S12X
- XGATE

Syntax:

```
CPU <processor>
```

DC.B (DB, FCB)

Writes a number of constant bytes into the memory.
Syntax:

```
DC.B <byte>, <byte>, ...
```

DC.W (DW, FDW)

Writes a number of constant words into the memory.
Syntax:

```
DC.W <word>, <word>, ...
```

DS.B (DS, RMB)

Advances both program counters by a number of bytes.
Syntax:

```
DS.B <#bytes>
```

DS.W (RMW)

Advances both program counters by a number of words.
Syntax:

```
DS.W <#words>
```

EQU

Directly assigns a value to a symbol.
Syntax:

```
<symbol> EQU <expression>
```

FCC

Writes an ASCII string into the memory. The string must be surrounded by a delimeter which can be any character.
Syntax:

```
FCC <delimeter><string><delimeter>
```

FCS

Writes an ASCII string into the memory, which is termitated by a set MSB in the last character. The string must be surrounded by a delimeterwhich can be any character.
Syntax:

```
FCS <delimeter><string><delimeter>
```

FILL

Fills a number of memory bytes with an 8-bit pattern.
Syntax:

```
FILL <pattern>, <#bytes>
```

LOC

Increments the **"LOC"** counter that is used for automatic symbol name extensions.
Syntax:

```
LOC
```

ORG

This pseudo-opcode can be used to set the program counters to a certain value. If "ORG" is called with two arguments, then the paged program counter will be set to the value of the first argument. The linear program counter will be set to the value of the second argument. If only one argument is passed to the pseudo-opcode, then this one will be the new value of the paged program counter. The value of the linear program counter is determined by the following table.

Paged Program Counter	Linear Program Counter
xx0000 to xx3FFF	F4000 to F7FFF
xx4000 to xx7FFF	F8000 to FBFFF
xx8000 to xxBFFF	(xx*4000) to (xx*4000+3FFF)
xxC000 to xxFFFF	FC000 to FFFFF

Syntax:

```
ORG <paged PC>
```

or

```
ORG <paged PC>, <linear PC>
```

UNALIGN

Same as ALIGN, except that the program counters are incremented until PC & mask == mask.
Syntax:

```
UNALIGN <mask>
```

or

```
UNALIGN <mask>, <pattern>
```

SETDP

Selects the 256 byte address range in which direct address mode can be applied for S12X MCUs.
Syntax:

```
SETDP <direct page>
```

HCS12 Opcodes

For a description of the HC(S)12 instruction set, please refer to the HCS12 Core Guide.
All opcodes must comply to the following syntax rules:

```
label name
must start at         operands must
the beginning         be separated
of the line           by a comma
|                      |
V                      V
<label> <opcode> <operand>, <operand>, ...
        ^       ^
        |       |
      spaces, tabs
```

Output Files

The HSW12 assembler can generate tree output files:

A Code Listing
The Code Listing shows the assembler source together with the associated hex code. The entries are sorted by their paged address.

A Paged S-Record File
The hex code of the paged address domain (paged program counter) in Motorolas S-Record format.
Paged addresses consist of an 8-bit page value (PPAGE register, MSB) and a 16-bit address value (PC register, LSB) =>PPAGE:ADDR.

A Linear S-Record File
The hex code of the linear address domain (linear program counter) in Motorolas S-Record format.
Linear addresses are equivalent to the physical address space of the NVMs inside the HC(S)12 MCUs.