

N1asm

- [Introduction](#)
- [Comments](#)
- [Expressions](#)
- [Labels](#)
- [Precompiler Directives](#)
- [Pseudo-Opcodes](#)
- [N1 Opcodes](#)
- [Output Files](#)

Introduction

The N1ASM is a simple multi-pass assembler which has been written in Perl code. It is based on the [HSW12 assembler](#). This assembler can be run as follows:

```
perl N1asm.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
```

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.

Expressions

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

Predefined Symbols

The N1 assembler knows a set of predefined symbols:

- @ Represents the current value of the linear program counter
- * Represents the current value of the local program counter

Automatic Symbol Extensions

The N1 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 N1 assembler supports the operators 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 local 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 global program counter to a symbol, use the following syntax:

SYMBOL EQU @

Precompiler Directives

The N1 assembler knows the following precompiler directives:

- [#DEFINE](#)
- [#UNDEF](#)
- [#IFDEF](#)
- [#IFNDEF](#)
- [#IFMAC](#)
- [#IFNMAC](#)
- [#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 N1 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" requires 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.

#IFMAC

Starts a section of conditional code. This code will only be compiled if the macro is defined.

#IFNMAC

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

#ELSE

Ends a section of conditional code that has been initiated with "#IFDEF", "#IFNDEF", "#IFMAC", or "#IFNMAC" 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      :      2
HEADER_START DW    ((\2&$FF)<<8) | (CODE_START - *)
NAME        FCS    \1
CODE_START  EQU    *
```

```
ORG      $8000, $F000
:        "XOR", $00
XOR      ;

:        "2ROT", $00
2ROT     ;

:        "LSHIFT", $00
LSHIFT   ;
```

Result:

008000	00F000		ORG	\$8000, \$F000	
008000	00F000	MACRO	:	"XOR", \$00	
008000	00F000	0003	HEADER_START	DW	((\$2&\$FF)<<8) (CODE_START - *)
008001	00F001	584F D2FF	NAME	FCS	\$1
008003	00F003	-> \$8003	CODE_START	EQU	*

008003 00F003 8EA0		XOR	;	
008004 00F004 MACRO		:	"2ROT", \$00	
008004 00F004 0003	HEADER_START	DW	((\$2&\$FF)<<8) (CODE_START-*)	(:)
008005 00F005 3252 4FD4	NAME	FCS	\$1	(:)
008007 00F007 -> \$8007	CODE_START	EQU	*	(:)
008007 00F007 06AB 0580 06AB 0598		2ROT	;	
0755 0598 0755 0598				
8460				
008010 00F010 MACRO		:	"LSHIFT", \$00	
008010 00F010 0004	HEADER_START	DW	((\$2&\$FF)<<8) (CODE_START-*)	(:)
008011 00F011 4C53 4849 46D4	NAME	FCS	\$1	(:)
008014 00F014 -> \$8014	CODE_START	EQU	*	(:)
008014 00F014 8F20		LSHIFT	;	

#EMAC

Ends a macro definition.

Pseudo-Opcodes

The following pseudo-opcodes are supported by the N1 assembler:

- [ALIGN](#)
- [CPU](#)
- [DC.W \(DW, FDW\)](#)
- [DS.W \(RMW\)](#)
- [ERROR](#)
- [EQU](#)
- [FCC](#)
- [FCS](#)
- [FCZ](#)
- [FILL](#)
- [LOC](#)
- [ORG](#)
- [UNALIGN](#)
- [SETDP](#)

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

symbol name must start at the begin of the line	arguments must be separated by a comma
↓	↓
<symbol>	<pseudo-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. Supported CPUs are:

- N1

Syntax:

```
CPU <processor>
```

DC.W (DW, FDW)

Writes a number of constant words into the memory.

Syntax:

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

DS.W (RMW)

Advances both program counters by a number of words.

Syntax:

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

ERROR

Triggers an intentional compile error. The string must be surrounded by a delimiter which can be any character.

Syntax:

```
ERROR <delimiter><string><delimiter>
```

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 delimiter which can be any character.

Syntax:

```
FCC <delimiter><string><delimiter>
```

FCS

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

Syntax:

```
FCS <delimiter><string><delimiter>
```

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 local program counter will be set to the value of the first argument. The global 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 both program counters.

Syntax:

```
ORG <local/global PC>
```

or

```
ORG <local PC>, <global 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>
```

N1 Opcodes

For a description of the N1 instruction set, please refer to the [N1 manual](#).

All opcodes must comply to the following syntax rules:

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

Output Files

The N1 assembler can generate two output files:

A Code Listing

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

A Memory File

The hex code of the global address domain (global program counter) in Verilog \$readmemh format.
Global addresses represent the physical address space.