九齊科技股份有限公司

Nyquest Technology Co., Ltd.

User Manual

NYASM

Nyquest MCU Assembler

Version 5.5

Aug. 27, 2025

Table of Contents

1 General Information

This first chapter contains general information that will be useful to know before working with *NYASM*.

1.1 About This Guide

1.1.1 Document Layout

This document describes how to use *NYASM* to develop code for Nyquest micro-controller applications. The user's guide layout is as follows.

2. *NYASM* Preview: Defines *NYASM* and describes what it does and how it works with other tools.

3. *NYASM* Installation and Getting Started: Describes how to install *NYASM* and gives an overview of operation.

4. Using *NYASM* with Windows: Describes how to use *NYASM* with Microsoft Windows via a Windows shell interface.

5. Directive Language: Describes the *NYASM* programming language including statements, operators, variables, and other elements.

6. Macro Language: Describes how to use *NYASM*'s built-in macro processor.

7. Expression Syntax and Operation: Provides guidelines for using complex expressions in *NYASM* source files.

Appendix A - Quick Reference: Lists Nyquest MCU device instruction sets, *NYASM* quick reference, and 4-bit MCU list.

Appendix B - *NYASM* Errors/Warnings: Contains a descriptive list of the errors, and warnings generated by *NYASM*.

1.1.2 Conventions Used in This Guide

This manual uses the following documentation conventions:

Directive	Description	Syntax
Arial Font	User-entered code or sample code.	#define BITWIDTH
Angle Brackets: <>	Variables. Text user supplied.	<label>, <exp>
Curly Brackets and Pipe Character: { | }	Choice of mutually exclusive arguments.	an OR selection error level { 0 | 1 }
Square Brackets: []	Could be omit.	[<label>] db <expr>[,<expr>,...,<expr>]
Ellipses: ...	Used to imply, but not show, additional text that is not relevant to the example.	List "list_option", ...,"list_option"
0xnn	Represents a hexadecimal number where n is a hexadecimal digit.	0xFF, 0x3B

1.1.3 Updates

All documentation becomes dated, and this user's manual is no exception. Since *NYASM*, and other Nyquest tools are constantly evolving to meet customer needs, some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site to obtain the latest documentation available.

1.2 Recommended Reading

This user's guide describes how to use *NYASM*. The user may also find the data sheets for specific micro-controller devices informative in developing firmware.

- **RevisionHistory.TXT**
  For the latest information on using *NYASM*, read the REVISIONHISTORY files (ASCII text files) included with the *NYASM* software. The REVISIONHISTORY files contain update information that may not be included in this document.
- **Interface**
  In-text Bold Characters Designates a button **OK, Cancel.**
  Uppercase Characters in Angle Brackets: < > Delimiters for special keys. <TAB>, <ESC>.
- **Microsoft Windows Manuals**
  This manual assumes that users are familiar with Microsoft Windows operating system. Many excellent references exist for this software program, and should be consulted for general operation of Windows.

1.3 The Nyquest Internet Web Site

Nyquest provides on-line support on the Nyquest World Wide Web (WWW) site. The web site is used by Nyquest as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Microsoft® Internet Explorer®.

• Connecting to the Nyquest Internet Web Site

The Nyquest website is available by using your favorite Internet browser to attach to: **http://www.nyquest.com.tw**

The website provides a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, and Articles.

Other data available for consideration is:

• Latest Nyquest Press Releases.

• Product Information.

1.4 Development Systems Customer Notification Service

Nyquest provided the customer notification service to help our customers keep current on Nyquest products with the least amount of effort. You will receive email notification whenever we change, update, revise or have errata related to that product family or development tool.

1.5 Customer Support

Users of Nyquest products can receive assistance through several channels:

• Distributor or Representative.

• Field Application Engineer (FAE).

• Hot line.

Customers should call their distributor, representative, or field application engineer (FAE) for support.

2 *NYASM* Preview

NYASM Windows-based PC application provides a platform for developing assembly language code for Nyquest's microcontroller (MCU) families

Content:

2.1 System Requirements

• Pentium 1.3GMHz CPU or above, Microsoft Windows operating system (7, 8, 10, 11).
• At least 1G of DRAM.
• At least 2G free space on hard disk.
• A display card and monitor with resolution of 1366x768 or higher.
• Microsoft .Net Framework 4.0 installed.

2.2 What *NYASM* Does

NYASM provides a universal solution for developing assembly code for all of Nyquest's 8-bit and 4-bit micro-controllers. Notable features include:

• All MCU Instruction Sets.
• Window Interfaces.
• Rich Directive Language.

2.3 Compatibility Issues

NYASM is compatible with all Nyquest development systems currently in production. This includes Q-Code and NYIDE. *NYASM* supports a clean and consistent method of specifying radix (see Chapter 4). You are encouraged to develop new code using the methods described within this document.

3 *NYASM* Installation and Getting Started

This chapter provides instructions for installation of *NYASM* on your system, and an overview of the assembler (*NYASM*).

Content:

3.1 Installation

Current version of *NYASM* is for Windows XP/7/8 version, NYASM.EXE has a Windows GUI interface. NYASM.EXE may be used with Windows XP/7/8. You can obtain *NYASM* from our website or sales. *NYASM* will be in a zip file.

To install:

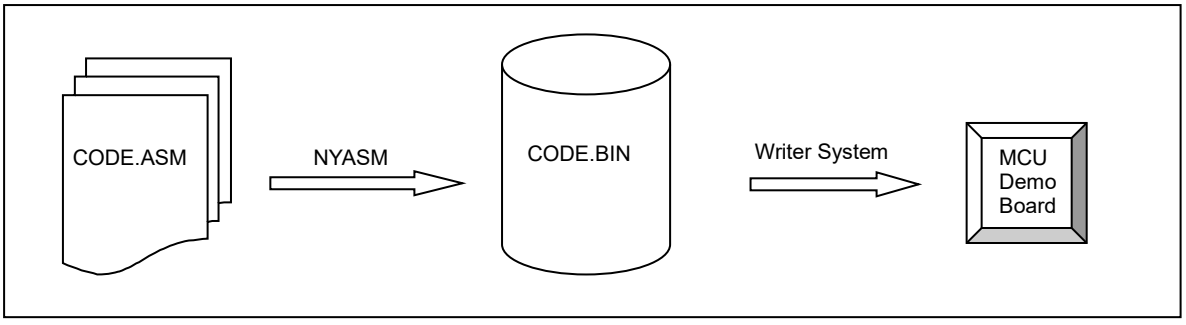
• Create a directory in which to place the files.

• Unzip the *NYASM* files using either WinZip®.

3.2 Overview of Assembler

NYASM can be used to generate binary code that can be executed directly by a micro-controller. Binary code is the default output from *NYASM*. This process is shown in Figure 3.1. When a source file is assembled in this manner, all values used in the source file must be defined within that source file, or in files that have been explicitly included. If assembly proceeds without errors, a BIN file will be generated, containing the executable machine code for the target device. This file can then be used in conjunction with a device programmer to program the micro-controller for function verification.

Figure 3.1: Generating binary code for function verification



3.3 Assembler Input/Output Files

These are the default file extensions used by *NYASM* and the associated utility functions.

Table 3.1: *NYASM* Default Extensions

Extension	Purpose
.ASM	Default source file extension input to *NYASM*: <source_name>.ASM
.LST	Default output extension for listing files generated by *NYASM*: <source_name>.LST
.ERR	Output extension from *NYASM* for Warning/error files: <source_name>.ERR
.BIN	Output extension from *NYASM* for the machine code of an application program in binary form: <source_name>.BIN
.HEX	Output extension from *NYASM* for representing BIN file in hexadecimal form: <source_name>.HEX
.DBG	Output extension from *NYASM* for the symbol and debug file. This file is created for AMCIDE debug mode: <source_name>.DBG

3.3.1 Source Code Format (.ASM)

The source code file may be created using any ASCII text file editor. It should conform to the following basic guidelines. Each line of the source file may contain up to four types of information:

- labels
- mnemonics
- operands
- comments

The order and position of these are important. Labels must start in the first non-blank position of a line. Mnemonics may start in the first non-blank position of a line, or follow a label. Operands follow the mnemonic. Comments may follow the operands, mnemonics or labels. The maximum column width is 255 characters. A colon must separate the label and the mnemonic, one or more spaces, or tabs must separate the mnemonic and its operand(s). Multiple operands must be separated by a comma. For example:

For example:

Sample *NYASM* Source Code (Shows multiple operands)

```
; sample NYQUEST assembler source code
;
list    p= ny5c640b , c=off ,r=hex
ORG_OFF             equ     0x30
ORG_SUBOFF          equ     0x00
SUBPPTRADDR         equ     ORG_SUBOFF+ORG_OFF
#include   "2102.h"
org         0x10
mvma        0x20
jmp         start
org         0x30
start:
mvma        0x30
mvat        0x12
end
```

3.3.1.1 Labels

A label must start in the first non-blank position of a line. It must be followed by a colon (:). Labels must begin with an alphabetic character or an under bar (_) and may contain alphanumeric characters, the under bar and the '@' symbol. By default they are case insensitive, but case sensitivity may be enabled through the command option of *NYASM*.

3.3.1.2 Mnemonics

Assembler instruction mnemonics, assembler directives and macro calls can begin in any column. If there is a label on the same line, instructions must be separated from that label by a colon, or by one or more spaces or tabs.

3.3.1.3 Operands

Operands must be separated from mnemonics by one or more spaces, or tabs. Multiple operands must be separated by commas.

3.3.1.4 Comments

NYASM treats anything after a semicolon as a comment. All characters following the semicolon are ignored through the end of the line.

3.3.2 Listing File Format (.LST)

For example:

Sample *NYASM* Listing File (.LST)

Nyquest Technology Co., Ltd.

NYASM 1.00 Copyright(c) Nyquest Technology Co., Ltd. [Build:Dec 20 2007]

File=E:\MyProjects\Build\asm\NYASM\Sample\NYASMSample.asm

Date=2007/12/20, 06:22:21 pm

```
ADDR     OPCODE/VALUE   LINE    TAG SOURCE STATEMENT                PAGE:1
                        0-0001   ; sample NYQUEST assembler source cod
                        0-0002   ;
                        0-0003   list   p=ny5c640b , c=off ,r=hex
         000000030      0-0004   ORG_OFF          equ    0x30
         000000000      0-0005   ORG_SUBOFF       equ    0x00
         000000030      0-0006   SUBPPTRADDR      equ    ORG_SUBOFF+ORG_OFF
                        0-0007   #include   "2102.h"
                        1-0001   ;
         000000010      0-0008   org      0x10
000010   D020           0-0009   mvma     0x20
000011   6030           0-0010   jmp      start
         000000030      0-0011   org      0x30
         000000030      0-0012   start:
000030   D030           0-0013   mvma     0x30
000031   0112           0-0014   mvat     0x12
                        0-0015   end
```

NYASM 1.00 Copyright(c) Nyquest Technology Co., Ltd. [Build:Dec 20 2007]

File=E:\MyProjects\Build\asm\NYASM\Sample\NYASMSample.asm

Date=2007/12/20, 06:22:21 pm

```
SYMBOL TABLE          TYPE        VALUE          PAGE:2
__NY5C640B            Constant    00000001
ORG_OFF               Constant    00000030
ORG_SUBOFF            Constant    00000000
Start                 Label       00000030
SUBPPTRADDR           Constant    00000030
```

SOURCE FILE TABLE

000 E:\MyProjects\Build\asm\NYASM\Sample\NYASMSample.asm

001 E:\MyProjects\Build\asm\NYASM\Sample\2102.h

PROCESSOR = NY5C640B (4 bits)

PROGRAM ROM = 0x00000000 - 0x000FFFFF

DATA ROM = 0x00000000 - 0x000FFFFF

SRAM / SFR = 0x00000000 - 0x000000FF

The listing file format produced by *NYASM* is straightforward:

The product name and version, the assembly date and time, and the page number appear at the top of every page. The first column of numbers contains the base address in memory where the code will be placed. The second column displays the 32-bit value of any symbols created with the EQU, VARIABLE, CONSTANT, or CBLOCK directives. The third column is reserved for the machine instruction. This is the code that will be executed by the Nyquest MCU. The fourth column lists the associated source file line number for this line. The remainder of the line is reserved for the source code line that generated the machine code. Errors, warnings, and messages are embedded between the source lines, and pertain to the following source line. The symbol table lists all symbols defined in the program.

3.3.3 Error File Format (.ERR)

NYASM by default generates an error file. This file can be useful when debugging your code. The format of the messages in the error file is:

[<type>] <file> (<line>) <number> <description>

For example:

[Error] C:\PROG.ASM 7 (133) w001: Symbol not previously defined (start).

Appendix B describes the error messages generated by *NYASM*.

3.3.4 Hex File Formats (.HEX)

NYASM is capable of producing different hex file formats.

3.3.5 Symbol and Debug File Format (.DBG)

When *NYASM* is evoked by NYIDE, it produces a DBG file for use in ICE debugging of code.

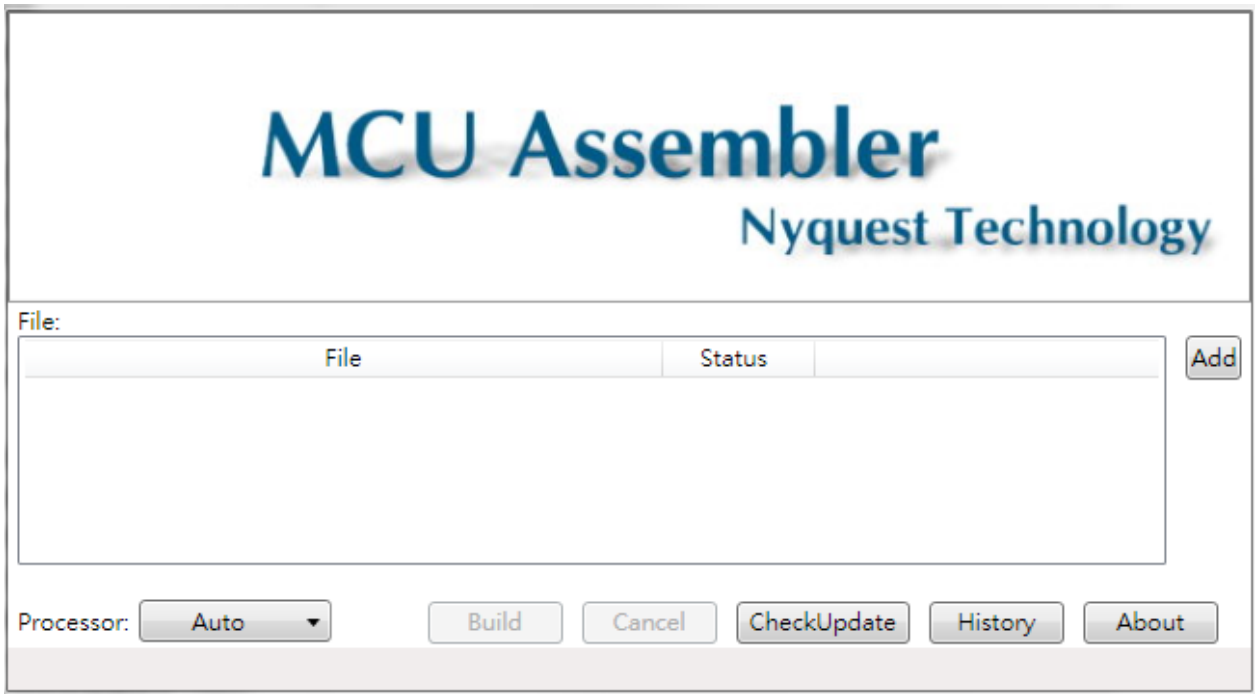
4 Using *NYASM* with Windows

This chapter is dedicated to describing the version of *NYASM* for Windows. This version may be run stand-alone, or within other Nyquest development tools. e.g. *Q-Code* and *NYIDE*.

4.1 User Interface

NYASM for Windows provides a graphical interface for setting assembler options. It is invoked by executing NYASM.EXE while in Windows.

Figure 4.1: *NYASM* Windows User Interface



Select a source file by dragging it into the window or using the **Add** button. Set the various options as described below. Then click **Build** to assemble the source file.

Note: When NYASM for Windows is invoked through other Nyquest development tools, the options screen is not available. Options are passed from specific tools in the form of arguments.

Table 4.1: Assembler Options

Option	Usage
Processor	Override any source file processor settings. Please refer to A.2 MCU List.

4.2 Introduction

NYASM provides UI with graphics and text mode. User can call *NYASM* in text mode by command script and then make execution automation. The executable file of user interface is *NYASM*.exe of installation directory. The available parameters are listed below.

Table 4.2: Available options

Option	Usage
/o=<file>	Import the specified asm file
/p=<icbody>	Replace the source file processor settings. Please refer to A.2 MCU List.
/f=<file>	Specify hardware configuration block file.
/bypass	By pass graphic interface. End program after completing configuration.
/unlockrsvmem	Allow the programming right in reserved memory area.
/nocfgblk	Ignore the assembly time check for the existence of configuration block file

5 Directive Language

This chapter describes the *NYASM* directive language. Directives are assembler commands that appear in the source code but are not translated directly into opcodes. They are used to control the assembler: its input, output, and data allocation.

5.1 Highlights

There are five basic types of directives provided by *NYASM*:

- Control Directives – Control directives permit sections of conditionally assembled code.
- Data Directives – Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, that is, by meaningful names.
- Listing Directives – Listing Directives are those directives that control the *NYASM* listing file format. They allow the specification of titles, pagination, and other listing control.
- Macro Directives – These directives control the execution and data allocation within macro body definitions.

5.2 NY4, NY5, NY7, NY8A, NY9

5.2.1 Directive Summary

Table 5.1 contains a summary of directives supported by *NYASM*. The remainder of this chapter is dedicated to providing a detailed description of the directives supported by *NYASM*.

Table 5.1: Directive Summary

Directive	Description	Syntax
BREAK	Escape from a **FOR**, **WHILE** or **REPEAT-UNTIL** loop, or Jump to the end of a **SWITCH** block.	break [<Boolean expression>]
CASE	Part of a **SWITCH** block; must use **CASE** with **SWITCH.**	switch <expression> case <expression 1>[,<expression 2>] <statements>
CBLOCK	Define a block of constants.	cblock [<expr>]
CONSTANT	Declare symbol constant.	constant <label>[=<expr>,...,<label>[=<expr>]]
CONTINUE	Jump to the begin of **FOR**, **WHILE** or **REPEAT-UNTIL** loop that contains **CONTINUE** directive. All statements behind **CONTINUE** in a loop are ignored.	continue [<Boolean expression>]
DEFAULT	Part of a **SWITCH** block; must use **DEFAULT** with **SWITCH.** Begin default assembly block to	default <statements>

Directive	Description	Syntax
	SWITCH.	
#DEFINE	Define a text substitution label.	#define <name> [<value>] #define <name> [<arg>,...,<arg>]
DW	Declare data of one word.	[<label>] dw <expr>[,<expr>,...,<expr>]
ELSE	Begin alternative assembly block to **IF.**	else <statements>
END	End program block.	end
ENDC	End an automatic constant block.	endc
ENDFOR	End a **FOR** loop.	endfor
ENDIF	End conditional assembly block.	endif
ENDM	End a macro definition.	endm
ENDS	Directive for coding convenience: presenting **ENDFOR, ENDW, ENDSW, ENDIF.**	ends
ENDSW	End conditional switching assembly block.	endsw
ENDW	End a **WHILE** loop.	endw
EQU	Define an assemble constant.	<label> equ <expr>
ERROR	Issue an error message.	error "<text_string>"
EXITM	Exit from a macro.	exitm
EXPAND	Expand macro listing.	expand
EXTERN	External symbol.	extern <label>
FOR	Perform counting loop **FOR.**	for <iterator> = <expr1> to <expr2> [step <expr3>]
IF	Begin conditionally assembled code block.	if <expr>
IFDEF	Execute if symbol has been defined.	ifdef <label>
IFNDEF	Execute If symbol has not been defined.	ifndef <label>
#INCLUDATA	Include binary data file.	#includata "<data_file>" [,<address>]
#INCLUDE	Include additional source file.	#include "<include_file>"
LINES	Re-declare line-per-page.	lines <value>
LIST	Listing options.	list [<list_option>,...,<list_option>]
LOCAL	Declare local macro variable.	local <label>[,<label>]
MACRO	Declare macro definition.	<label> macro [<arg>,...,<arg>]
MAXMACRO DEPTH	Setup the maximum depth of macro expansion.	Maxmacrodepth [=] <expr>
MESSG	Create user defined message.	messg "<message_text>"

Directive	Description	Syntax
NEWPAGE	Re-declare line-per-page.	Newpage <value>
NOEXPAND	Turn off macro expansion.	noexpand
ORG	Set program origin.	[<label>:] org <expr>
ORGALIGN	Set program origin with alignment.	[<label>:] orgalign <expr>, <align>
RADIX	Specify default radix.	radix <default_radix>
REPEAT	Begin at-least-one-time loop.	Repeat <statements> until <Boolean expression>
SUBTITLE	Specify program subtitle.	subtitle "<sub_text>"
SWITCH	Begin conditional switching assembly block.	switch <expr>
TITLE	Specify program title.	title "<title_text>"
#UNDEFINE	Delete a substitution label.	#undefine <label>
UNTIL	End at-least-one-time loop if condition is true.	Repeat <statements> until <Boolean expression>
VARIABLE	Declare symbol variable.	variable <label>[=<expr>,...,<label>[=<expr>]]
WHILE	Perform loop **WHILE** condition is true.	while <expr>
.ALIGN2	Align the starting address of the program.	.align2 <expr>, <bit>

5.2.2 BREAK – Jump Out Point in a Logic Block

◆ **Syntax**

Syntax 1:

```
<for|while|repeat – loop begin>
    [<statements>]
    break [<Boolean expr>]
    [<statements>]
<for|while|repeat – loop end>
```

Syntax 2:

```
switch <expr>
    case <expr1>[,<expr2>]
    [<statements>]
    break [<Boolean expr>]
    [<statements>]
    [<case-statements>]
Endsw
```

◆ **Description**

Set the logical point in a program which will escape the running flow from a WHILE, FOR, or REPEAT-UNIT loop. break also is used in switch block to achieve the purpose of switching among conditional branches.

◆ **Example**

Example 1:

```
for i =0 to 4
nop
break i==2
halt
endfor
```

Example 2:

```
a=1
switch a
case 1, 2
   nop
   break
   case 1
halt
endsw
```

◆ **See Also**

FOR, WHILE, REPEAT, SWITCH

5.2.3 CASE – Define an Option Item of SWITCH

◆ **Syntax**

```
switch <expr>
      case <expr1>[,<expr2>]
      [<statements>]
      :
      :
      default
      [<statements>]
endsw
```

◆ **Description**

Define an option of selection statement. Once the <exprN> matched one of the conditions after case, running flow will branch into that case item. case is part of a switch block, and must be used with switch.

◆ **Example**

```
a=1
switch a
    case 1, 2
    nop
    break
    case 1
    halt
endsw
```

◆ **See Also**

DEFAULT, SWITCH

5.2.4 CBLOCK – Define a Block of Constants

◆ **Syntax**

```
cblock [<expr>]
   [<label>[=<increment>][,<label>[=<increment>]]]
endc
```

◆ **Description**

Define a list of named constants. Each <label> is assigned a value of one higher than the previous <label>. The purpose of this directive is to assign address offsets to many labels. The list of names end when an ENDC directive is encountered. <expr> indicates the starting value for the first name in the block. If no expression is found, the first name will receive a value one higher than the final name in the previous CBLOCK. If the first CBLOCK in the source file has no <expr>, assigned values start with zero. If <increment> is specified, then the next <label> is assigned the value of <increment> higher than the previous <label>. Multiple names may be given on a line, separated by commas. cblock is useful for defining constants in program and data memory.

◆ **Example**

```
cblock 0x20                ; name_1 will be assigned 20
   name_1, name_2          ; name_2 is 21
   name_3 =0x30, name_4  ; name_4 is assigned 30,name_4 is assigned 31.
endc
```

◆ **See Also**

ENDC

5.2.5 CONSTANT – Declare Symbol Constant

◆ **Syntax**

```
constant <label>=<expr> [...,<label>=<expr>]
```

◆ **Description**

Creates symbols for use in *NYASM* expressions. Constants may not be reset after having once been initialized, and the expression must be fully resolvable at the time of the assignment. This is the principal difference between symbols declared as CONSTANT and those declared as VARIABLE. Otherwise, constants and variables may be used interchangeably in expressions.

◆ **Example**

```
variable RecLength=64                    ; Set Default RecLength
constant BufLength=512                   ; Init BufLength
                :                        ; RecLength may
:                                        ; be reset later
:                                        ; in RecLength=128
:                                        ;
constant MaxMem=RecLength+BufLength ;CalcMaxMem
```

◆ **See Also**

VARIABLE

5.2.6 CONTINUE – Ignore Statements Afterward and Start Next Loop

◆ **Syntax**

```
<for|while|repeat – loop begin>
    [<statements>]
    continue [<Boolean expr>]
    [<statements>]
<for|while|repeat – loop end>
```

◆ **Description**

Set a logical point in a program which will ignore statements after continue in WHILE, FOR, or REPEAT-UNITL looping block, and jump to the begin of looping block containing continue directive.

◆ **Example**

```
for i =0 to 4
            nop
            continue i==2
            halt
endfor
```

◆ **See Also**

FOR, WHILE, REPEAT

5.2.7 DEFAULT – Define an Unconditional Item of SWITCH

- **Syntax**

```
switch <expr>
    case <expr1>[,<expr2>]
    [<statements>]
    :
    :
    default
    [<statements>]
    endsw
```

- **Description**

Define an unconditional item of selection statement. Once no condition after case items matched <expr>, running flow will go into default item. default is part of a switch block, and must be used with switch.

- **Example**

```
a=1
switch a
case 1, 2
        nop
        break
        case 1
        halt
        default
nop
endsw
```

- **See Also**

CASE, SWITCH

5.2.8 #DEFINE – Define a Text Substitution Label

- **Syntax**

#define <name> [<string>]

- **Description**

This directive defines a text substitution string. Wherever <name> is encountered in the assembly code, <string> will be substituted. Using the directive with no <string> causes a definition of <name> to be noted internally and may be tested for using the IFDEF directive.

- **Example**

```
#define length 20
```

```
#define control 0x19, 7
:
:
srbr control ; set bit 7 in 0x19
```

- **See Also**

  #UNDEFINE, IFDEF, IFNDEF

5.2.9 DW – Declare Data of One Word

- **Syntax**

  [<label>:] dw <expr>[,<expr>,...,<expr>]

- **Description**

  Reserve program memory words for data, initializing that space to specific values. Values are stored into successive memory locations and the location counter is incremented by one. Expressions may be literal strings and are stored as described in the DATA directive.

- **Example**

```
dw 39, (d_list*2+d_offset)
dw diagbase-1
```

5.2.10 ELSE – Begin Alternative Assembly Block to IF

- **Syntax**

  else

- **Description**

  Used in conjunction with an IF directive to provide an alternative code block should the IF evaluate to false. ELSE may be used inside a regular program block or macro.

- **Example**

```
speed macro rate
   if rate < 50
   dw slow
   else
   dw fast
   endif
endm
```

- **See Also**

  ENDIF, IF

5.2.11 END – End Program Block

- **Syntax**

  end

- **Description**

  Indicates the end of the program.

- **Example**

```
list p= ny4b095a
:       ; executable code
:       ;
end ; end of instructions
```

5.2.12 ENDC – End an Automatic Constant Block

- **Syntax**

  endc

- **Description**

  ENDC terminates the end of a CBLOCK list. It must be supplied to terminate the list.

- **See Also**

  CBLOCK

5.2.13 ENDFOR – End a For Loop

- **Syntax**

  endfor

- **Description**

  ENDFOR terminates a FOR loop. As long as the looping counter specified by the FOR directive went over the conditional boundary, the source code between the FOR directive and the ENDFOR directive will be repeatedly expanded in the assembly source code stream. This directive may be used inside a regular program block or macro.

- **See Also**

  FOR

5.2.14 ENDIF – End Conditional Assembly Block

- **Syntax**

  endif

- **Description**

  This directive marks the end of a conditional assembly block. ENDIF may be used inside a regular program block or macro.

◆ **See Also**

ELSE, IF

5.2.15 ENDM – End a Macro Definition

◆ **Syntax**

endm

◆ **Description**

Terminates a macro definition begun with MACRO.

◆ **Example**

```
make_table macro arg1, arg2
    dw arg1, 0      ; null terminate table name
    res arg2  ; reserve storage
endm
```

◆ **See Also**

MACRO, EXITM

5.2.16 ENDS – Coding Convenience

◆ **Syntax**

ends

◆ **Description**

Present ENDFOR, ENDW, ENDSW, ENDIF

◆ **See Also**

ENDFOR, ENDW, ENDSW, ENDIF

5.2.17 ENDSW – End a Switch Block

◆ **Syntax**

endsw

◆ **Description**

Terminates a SWITCH block definition begun with SWITCH.

◆ **Example**

See the example for SWITCH

◆ **See Also**

SWITCH

5.2.18 ENDW – End a While Loop

◆ **Syntax**

endw

- **Description**

  ENDW terminates a WHILE loop. As long as the condition specified by the WHILE directive remains true, the source code between the WHILE directive and the ENDW directive will be repeatedly expanded in the assembly source code stream. This directive may be used inside a regular program block or macro.

- **Example**

  See the example for WHILE

- **See Also**

  WHILE

5.2.19 EQU – Define an Assembler Constant

- **Syntax**

  <label> equ <expr>

- **Description**

  The value of <expr> is assigned to <label>.

- **Example**

  four equ 4 ; assigned the numeric value of 4 to label four

5.2.20 ERROR – Issue an Error Message

- **Syntax**

  error "<text_string>"

- **Description**

  <text_string> is printed in a format identical to any NYASM error message. <text_string> may be from 1 to 80 characters.

- **Example**

```
error_checking macro arg1
    if arg1 >= 55        ; if arg is out of range
    error "error_checking-01 arg out of range"
    endif
endm
```

- **See Also**

  MESSG

5.2.21 EXITM – Exit from a Macro

- **Syntax**

exitm

- **Description**

  Force immediate return from macro expansion during assembly. The effect is the same as if an ENDM directive had been encountered. This directive can only be used in NY5+ and NY6.

- **Example**

```
test macro filereg
    if filereg == 1  ; check for valid file
    exitm
    else
    error "bad file assignment"
    endif
endm
```

- **See Also**

  ENDM MACRO

5.2.22 EXPAND – Expand Macro Listing

- **Syntax**

  expand

- **Description**

  Expand all macros in the listing file. This directive is roughly equivalent to the “Macro Expansion” assembler option, but may be disabled by the occurrence of a subsequent NOEXPAND.

- **See Also**

  MACRO, NOEXPAND

5.2.23 EXTERN – External Symbol

- **Syntax**

  extern <label>

- **Description**

  Define the symbol as a public symbol when it needs to be accessed across different modules. This is required when, for example, two independently compiled assembly files call functions defined in each other. This feature is only available in NY8 C language projects.

5.2.24 FOR – Perform For Loop While Iterator Meets the Condition

- **Syntax**

```
for <iterator>=<expr1> to <expr2> [step <expr3>]
:
```

:

endfor

◆ **Description**

The lines between the FOR and the ENDFOR are assembled as long as <iterator> evaluates in the range of <expr1> to <expr2>. A FOR loop can be repeated a maximum of 256 times.

◆ **Example**

```
for I=0 to 5
        nop
endfor
```

◆ **See Also**

ENDFOR

5.2.25 IF – Begin Conditionally Assembled Code Block

◆ **Syntax**

if <expr>

◆ **Description**

Begin execution of a conditional assembly block. If <expr> evaluates to true, the code immediately following the IF will assemble. Otherwise, subsequent code is skipped until an ELSE directive or an ENDIF directive is encountered. An expression that evaluates to zero is considered logically FALSE. An expression that evaluates to any other value is considered logically TRUE. The IF and WHILE directives operate on the logical value of an expression. A relational TRUE expression is guaranteed to return a nonzero value, FALSE a value of zero.

◆ **Example**

```
if version == 100; check current version
    :    ;executable cod
    :    ;executable cod
else
    :    ;executable cod
    :    ;executable cod
endif
```

◆ **See Also**

ELSE, ENDIF

5.2.26 IFDEF – Execute If Symbol has Been Defined

◆ **Syntax**

ifdef <label>

◆ **Description**

If <label> has been previously defined, usually by issuing a #DEFINE directive or by setting the value on the NYASM command line, the conditional path is taken. Assembly will continue until a matching ELSE or ENDIF directive is encountered.

◆ **Example**

```
#define testing 1 ; set testing "on"
:
:
ifdef testing
   <execute test code> ; this path would be executed.
Endif
```

◆ **See Also**

#DEFINE, ELSE, ENDIF, IFNDEF, #UNDEFINE

5.2.27 IFNDEF – Execute If Symbol has not Been Defined

◆ **Syntax**

ifndef <label>

◆ **Description**

If <label> has not been previously defined, or has been undefined by issuing an #UNDEFINE directive, then the code following the directive will be assembled. Assembly will be enabled or disabled until the next matching ELSE or ENDIF directive is encountered.

◆ **Example**

```
#define testing1 ; set testing on
:
:
#undefine testing1 ; set testing off
ifndef testing ; if not in testing mode
:      ; execute this path
:
endif
end ; end of source
```

◆ **See Also**

#DEFINE, ELSE, ENDIF, IFDEF, #UNDEFINE

5.2.28 #INCLUDATA – Include Binary Data File

- **Syntax**

  #includata "<binary_data_file>"[, address]

- **Description**

  The specified file is read in as binary data. The effect is the same as if the entire text of the included file were inserted into the file at the location of the #includata statement. If the includes data file needs to be inserted at a specific location, users can specify the location by address. #includata must be the last statement before end directive. <binary_data_file> must be enclosed in quotes. If a fully qualified path is specified, only that path will be searched. Otherwise, the search path is: source file directory. <binary_data_file> will becomes a label after assembled.

- **Example**

  #includata "c:\music\s02.sog", 0x2000 ; insert data file at 0x2000

5.2.29 #INCLUDE – Include Additional Source File

- **Syntax**

  #include "<include_file>"

- **Description**

  The specified file is read in as source code. The effect is the same as if the entire text of the included file were inserted into the file at the location of the include statement. Upon end-of-file, source code assembly will resume from the original source file. <include_file> must be enclosed in quotes. If a fully qualified path is specified, only that path will be searched. Otherwise, the search path is: source file directory.

- **Example**

  #include "c:\sys\sysdefs.inc" ; system defs

  #include “regs.h” ; register defs

5.2.30 LINES – Reset Line Count per Listing Page

- **Syntax**

  lines <value>

- **Description**

  Set the maximum line count per page when generating listing file.

- **See Also**

  NEWPAGE

5.2.31 LIST – Listing Options

- **Syntax**

list [<list_option>, ..., <list_option>]

◆ **Description**

Occurring on a line by itself, the LIST directive has the effect of turning listing output on, if it had been previously turned off. Otherwise, one of the following list options can be supplied to control the assembly process or format the listing file:

Table 5.2: List Directive Options

OPTION	DEFAULT	DESCRIPTION
c	Off	Enable/Disable case sensitivity c=on Enable c=off Disable
p	None	Set the processor type: /p=<processor_type> where <processor_type> is an Nyquest MCU device. For example, NY5A005A.
unlockrsvmem	Locked	/unlockrsvmem For 4-bit MCU only. Allow the programming right in reserved memory area.
nocfgblk	Configuration Block required	/nocfgblk For 4-bit MCU only. Ignore the assembly time check for the existence of configuration block file.

◆ **Example**

list p=ny5c640b, c = off

5.2.32 LOCAL – Declare Local Macro Variable

◆ **Syntax**

local <label>[,<label>...]

◆ **Description**

Declares that the specified data elements are to be considered in local context to the macro. <label> may be identical to another label declared outside the macro definition; there will be no conflict between the two. If the macro is called recursively, each invocation will have its own local copy.

◆ **Example**

<main code segment>

:

:

len equ 10 ; global version

size equ 20 ; note that a local variable may now be created and modified

test macro size

```
    local len, label ; local len and label
    len set size ; modify local len
    label res len ; reserve buffer
    len set len-20 ;
endm ; end macro
```

◆ **See Also**

ENDM, MACRO

5.2.33 MACRO – Declare Macro Definition

◆ **Syntax**

<label> macro [<arg>, ..., <arg>]

◆ **Description**

A macro is a sequence of instructions that can be inserted in the assembly source code by using a single macro call. The macro must first be defined, then it can be referred to in subsequent source code. A macro can call another macro, or may call itself recursively.

◆ **Example**

```
Read macro device, buffer, count
mvma device
mvma buffer
mvma count
endm
:
:
read 1,2,3
```

◆ **See Also**

ELSE, ENDIF, ENDM, EXITM, IF, LOCAL

5.2.34 MAXMACRODEPTH – Define Maximum Macro Depth

◆ **Syntax**

maxmacrodepth[=]<expr>

◆ **Description**

MAXMACODEPTH defines the maximum valid macro depth to <expr>. <expr> must be less than or equal to the maximum depth 256. MAXMACODEPTH can be used more than once in a source file. Each use redefines the maximum valid macro depth.

◆ **Example**

```
list p=ny5c640b
maxmacrodepth 0x10
:
```

:

5.2.35 MESSG – Create User Defined Message

- **Syntax**

  messg "<message_text>"

- **Description**

  Causes an informational message to be printed in the listing file. Issuing a MESSG directive does not set any error return codes.

- **Example**

```
mssg_macro macro
    messg "mssg_macro-001 invoked without argument"
endm
```

- **See Also**

  ERROR

5.2.36 NEWPAGE – Insert Listing Page Eject

- **Syntax**

  newpage <value>

- **Description**

  Inserts a page eject into the listing file.

- **See Also**

  LINE

5.2.37 NOEXPAND – Turn off Macro Expansion

- **Syntax**

  noexpand

- **Description**

  Turns off macro expansion in the listing file.

- **See Also**

  EXPAND

5.2.38 ORG – Set Program Origin

- **Syntax**

  [<label>:] org <expr>

- **Description**

Set the program origin for subsequent code at the address defined in <expr>. If <label> is specified, it will be given the value of the <expr>. If no ORG is specified, code generation will begin at address zero.

- **Example**

int_1: org 0x20
; Vector 20 code goes here
int_2: org int_1+0x10
; Vector 30 code goes here

5.2.39 ORGALIGN – Set Program Origin With Address Alignment

- **Syntax**

[<label>:] orgalign <expr>,<align>

- **Description**

Set the program origin for subsequent code at the address defined in <expr>|<align>. If <label> is specified, it will be given the value of the <expr>|<align>. If no ORGALIGN is specified, code generation will begin at address zero.

- **Example**

int_1: orgalign 0x20,0x7

- **See Also**

.align2

5.2.40 RADIX – Specify Default Radix

- **Syntax**

radix <default_radix>

- **Description**

Sets the default radix for data expressions. The default radix is dec. Valid radix values are: hex, dec, oct, or bin.

- **Example**

radix dec

- **See Also**

LIST

5.2.41 REPEAT – Begin a Repeat-Until Loop Block Definition

- **Syntax**

repeat
:

:
until <expr>

◆ **Description**

Begin a REPEAT-UNTIL block definition.

◆ **Example**

```
test_mac macro count
    variable i
    i = 0
    repeat
    i += 1
    until i > count
endm
:
:
End
```

◆ **See Also**

WHILE, UNTIL

5.2.42 SUBTITLE – Specify Program Subtitle

◆ **Syntax**

subtitle "<sub_text>"

◆ **Description**

<sub_text> is an ASCII string enclosed in double quotes, 60 characters or less in length. This directive establishes a second program header line for use as a subtitle in the listing output.

◆ **Example**

```
subtitle "diagnostic section"
```

◆ **See Also**

TITLE

5.2.43 SWITCH – Begin Conditional Switching Assembly Block

◆ **Syntax**

```
switch <expr>
    case <expr1>[,<expr2>]
    [<statements>]
    case < exprM>[,<exprN>]
    :
    :
```

```
        default
        [<statements>]
    endsw
```

◆ **Description**

Begin execution of a conditional switching assembly block. If <expr> evaluates to matching any <exprX> after cases , the code immediately following that matched case will assemble. Otherwise, subsequent code is skipped until a default directive or an ENDSW directive is encountered.

◆ **Example**

```
a=1
switch a
    case 1, 2
    nop
    break
    case 1
    halt
    default
endsw
```

◆ **See Also**

BREAK, DEFAULT

5.2.44 TITLE – Specify Program Title

◆ **Syntax**

title "<title_text>"

◆ **Description**

<title_text> is a printable ASCII string enclosed in double quotes. It must be 60 characters or less. This directive establishes the text to be used in the top line of each page in the listing file.

◆ **Example**

title "operational code, rev 5.0"

◆ **See Also**

SUBTITLE

5.2.45 #UNDEFINE – Delete a Substitution Label

◆ **Syntax**

#undefine <label>

◆ **Description**

<label> is an identifier previously defined with the #DEFINE directive. It must be a valid NYASM label. The symbol named is removed from the symbol table.

- **Example**

```
#define length 20
:
:
#undefine length
```

- **See Also**

#DEFINE, IFDEF, #INCLUDE, IFNDEF

5.2.46 UNTIL – Perform Loop Until Condition is True

- **Syntax**

```
repeat
:
:
until <expr>
```

- **Description**

The lines between the REPEAT and the UNTIL are assembled at least once, and as long as <expr> evaluates to FALSE. A REPEAT loop can be repeated at maximum of 256 times.

- **Example**

```
test_mac macro count
    variable i
    i = 0
    repeat
    i += 1
    until i < count
endm
:
:
end
```

- **See Also**

WHILE, REPEAT

5.2.47 VARIABLE – Declare Symbol Variable

- **Syntax**

variable <label>[=<expr>][,<label>[=<expr>]...]

- **Description**

Creates symbols for use in NYASM expressions. Variables and constants may be used interchangeably in expressions. Note that variable values cannot be updated within an operand. You must place variable assignments, increments, and decrements on separate lines.

- **Example**

  Please refer to the example given for the CONSTANT directive.

- **See Also**

  CONSTANT

5.2.48 WHILE – Perform Loop While Condition is True

- **Syntax**

```
while <expr>
:
:
endw
```

- **Description**

  The lines between the WHILE and the ENDW are assembled as long as <expr> evaluates to TRUE. An expression that evaluates to zero is considered logically FALSE. An expression that evaluates to any other value is considered logically TRUE. A relational TRUE expression is guaranteed to return a non-zero value; FALSE a value of zero. A WHILE loop can contain at most 100 lines and be repeated a maximum of 256 times.

- **Example**

```
test_mac macro count
variable i
i = 0
while i < count
movlw i
i += 1
endw
endm
start
test_mac 5
end
```

- **See Also**

  ENDW IF

5.2.49 .ALIGN2 – AlignThe Staring Address of Program

- **Syntax**

.align2 <expr>, <bit>

- **Description**

  Align the starting address of program with <bit>, the low bit of address is <expr>.

  When the demand addresses are 0x41, 0x141, 0x241, 0x341, and so on, user could use .align2 to align the 8 bits, and set the low bit as 0x41. But the ORGALIGN command cannot specify the number of bit, 0xC1will still be generated.

  This command only be supported nu NY5+ and NY6

- **Example**

  .align2 0x41, 8

- **See Also**

  ORGALIGN

5.3 NY8L

The directives for NY8L are different from other IC series. The following are the descriptions of directives.

5.3.1 Directive Summary

Table 5.3 contains a summary of directives supported by *NYASM*. The remainder of this chapter is dedicated to providing a detailed description of the directives supported by *NYASM*.

Table 5.3: Directive summary

Directive	Description	Syntax
.and	Boolean and operation	<expr> .and <expr>
.bankbyte	Access bank byte	.bankbyte(<expr>)
.bitand	Bit and operation	<expr> .bitand <expr>
.bitnot	Bit not operation	.bitnot <expr>
.bitor	Bit or operation	<expr> .bitor <expr>
.bitxor	Bit xor operation	<expr> .bitxor <expr>
.blank	Check blank symbol	.blank(<symbol>)
.byte	Low byte	.byte(<expr>)
.ceil	Unconditional carry	.ceil(<expr>)
.code	The abbreviation of .segment “code”	.code

Directive	Description	Syntax
.data	The abbreviation of .segment “data”	.data
.define	Definition	.define <symbol> <expr>
.defined	Check whether the symbol is defined	.defined(<symbol>)
.else	Begin alternative assembly block to IF	.else
.elseif	Begin alternative assembly block after IF and the specified condition is true	.elseif(<expr>)
.endif	End conditional assembly block	.endif
.endmacro	End macro defined block	.endmacro
.endrepeat	End the repeating scope	.endrepeat
.endscope	End variable scope	.endscope
.endstruct	End structure block	.endstruct
.equ	Define constant	<symbol> .equ <expr>
.error	Issue an compilation error message	.error “<text>”
.export	Export symbol	.export <symbol>
.exportzp	Export zero page symbol	.exportzp <symbol>
.extern	Declare external symbol	.extern <symbol>
.externzp	Declare global zero page symbol	.externzp <symbol>
.floor	Uncoditional round down	.floor(<expr>)
.hibyte	High byte	.hibyte(<expr>)
.if	Conditional assembly	.if(<expr>)
.ifblank	Conditional assembly if parameter is blank	.ifblank(<symbol>)
.ifdef	Conditional assembly If defined	.ifdef(<symbol>)
.ifnblank	Conditional assembly If parameter isn’t blank	.ifnblank(<symbol>)
.ifndef	Conditional assembly If undefined	.ifndef(<symbol>)
.import	Import symbol	.import <symbol>
.importzp	Import zero page symbol	.importzp <symbol>

Directive	Description	Syntax
.incbin	Insert binary file	.incbin “<file>”
.include	Include file	.include “<file>”
.lobyte	Low byte	.lobyte(<expr>)
.local	Declare local macro variable	.local <symbol>
.macro	Define macro	.macro <name> <arg1>, <arg2>, …
.mod	Remainder operation	<expr> .mod <expr>
.not	Boolean reverse operation	.not <expr>
.or	Boolean or operation	<expr> .or <expr>
.org	Set program origin	.org <expr>
.repeat	Begin a repeat-until loop block definition	.repeat <expr>
.res	Reserve space	.res <expr>, <expr>
.round	Round	.round(<expr>)
.scope	Start variable scope	.scope <symbol>
.segment	Program segment	.segment “<symbol>”
.setcpu	Setup CPU	.setcpu <ic_body>
.shl	Left shift	<expr> .shl <expr>
.shr	Right shift	<expr> .shr <expr>
.string	Access string	.string(<symbol>)
.word	Word	.word <expr>
.xor	Boolean exclusive or	<expr> .xor <expr>

5.3.2 .And – Boolean AND Operation

- **Syntax**

  <symbol> = <expr1> .and <expr2>

- **Description**

  Calculate expr1 & expr2

- **Example**

```
.if(0 .and 1)
    .error
.endif
```

5.3.3 .BANKBYTE – Access Bank Byte

- **Syntax**

  .bankbyte(<expr>)

- **Description**

  Obtain bank byte (bit 16~23) of <expr> high byte.

- **Example**

  Label1_bank = .bankbyte(label1)

5.3.4 .BITAND - Bit AND Operation

- **Syntax**

  <symbol> = <expr1> .and <expr2>

- **Description**

  Calculate expr1 & expr2

- **Example**

  Ans = 1 .bitand 3

  ; Ans = 1

5.3.5 .BITNOT – Bit NOT Operation

- **Syntax**

  <symbol> = .not <expr>

- **Description**

  Reverse every bit of expr. The bit width is 32bit. If 0 is reversed the result will be 1 of 32bit.

- **Example**

  Ans = .bitnot 3

  ; Ans = 0xFFFFFFFC

5.3.6 .BITOR – Bit XOR Operation

- **Syntax**

  <symbol> = <expr1> .bitor <expr>

- **Description**

  Calculate expr1 exclusive or expr2

- **Example**

```
Ans = 3 .bitor 6
; Ans = 7
```

5.3.7 .BITXOR – Bit XOR Operation

- **Syntax**

  <symbol> = <expr1> .xor <expr2>

- **Description**

  Calculate expr1 exclusive or with expr2

- **Example**

```
Ans = 1 .xor 3
; Ans = 2
```

5.3.8 .BLANK – Check Blank Symbol

- **Syntax**

  .blank(<symbol>)

- **Description**

  The returned Boolean value will indicate the parameter <symbol> is blank or not. It can be used for checking the assigned parameter of caller if applied in macro.

- **Example**

```
.macro M1 arg1
   .if ( .blank(arg1) )
      .error
   .endif
.endmacro
```

5.3.9 .BYTE – Low Byte

- **Syntax**

  <symbol> = .byte(<expr>)

- **Description**

  Access the low byte of expr.

- **Example**

```
Ans = .byte(0x1234)
; Ans = 0x34
```

5.3.10 .CEIL – Unconditional Carry

- **Syntax**

  <symbol> = .ceil(<expr>)

- **Description**

  If <expr> is a float, it will be rounded upward to the nearest integer.

- **Example**

  Ans = .ceil(1.2)

  ; Ans = 2

5.3.11 .CODE - The abbreviation of .segment “code”

- **Syntax**

  .code

- **Description**

  Equivalent to .segment “code”

- **See Also**

  .SEGMENT

5.3.12 .DATA - The abbreviation of .segment “data”

- **Syntax**

  .data

- **Description**

  Equivalent to .segment “data”

- **See Also**

  .SEGMENT

5.3.13 .DEFINE – Definition

- **Syntax**

  .define <symbol> <expr>

- **Description**

  Define an expression to symbol which make the symbol represented expression then.

- **Example**

  .define AAA 1 + 2

  Ans = AAA

  ;Ans = 3

- **See Also**

  .DEFINED

5.3.14 .DEFINED – Check Whether the Symbol Is Defined

- **Syntax**

.defined(<symbol>)

- **Description**

If <symbol> has been defined, the outcome will be true(1) otherwise false(0). In general conditions, use .ifdef <symbol> to check the defined symbols. However, if numbers of symbol have to be checked if defined, a nested .ifdef is necessary. Instead, user can use .defined to check multiple defined symbols, as below example:

```
.ifdef (symbol1)
    .ifdef(symbol2)
        <statements>
    .endif
.endif
```

It can be rewritten:

```
.if (.defined(symbol1) && .defined(symbol2) )
    <statements>
.endif
```

- **Example**

```
.if ( .defined(def_name))
    .error
    ; Because def_name symbol isn't defined, the program won't be assembled
.endif
```

- **See Also**

.DEFINE

5.3.15 .ELSE – Begin Alternative Assembly Block to IF

- **Syntax**

```
.if(<expr>)
    <statements>
.else
    <statements>
.endif
```

- **Description**

Used in conjunction with an IF directive to provide an alternative path of assembly code should the IF evaluate to false. ELSE may be used inside a regular program block or macro.

- **Example**

```
.if( 0 )
    .error
.else
```

```
    ; do something
.endif
```

◆ **See Also**

.IF, .ELSEIF

5.3.16 .ELSEIF –Begin Alternative Assembly Block After IF And The Specified Condition Is True

◆ **Syntax**

```
.if(<expr>)
    <statements>
.else if (<expr>)
    <statements>
.endif
```

◆ **Description**

Assemble a program block if <expr> is true if the previous .if or .elseif evaluate to false.

◆ **Example**

```
TempVar = 1
.if( TempVar < 1 )
    .error
.elseif(TempVar < 2)
    ; do something
.endif
```

◆ **See Also**

.IF, .ELSE

5.3.17 .ENDIF – End Conditional Assembly Block

◆ **Syntax**

```
.if(<expr>)
    <statements>
.else if (<expr>)
    <statements>
.endif
```

◆ **Description**

End the conditional assembly block started by .iF.

◆ **See Also**

.IF, .ELSE, ELSEIF

5.3.18 .ENDMACRO – End Macro Defined Block

- **Syntax**

  .macro <symbol> [<arg1> [,<arg2>…]]

  <statements>

  .endmacro

- **Description**

  End the conditional assembly block started by .macro.

- **See Also**

  .macro

5.3.19 .ENDREPEAT – End the Repeating Scope

- **Syntax**

  .repeat <expr>

  <statements>

  .endrepeat

- **Description**

  End the repeating block that .repeat starts off.

- **See Also**

  .repeat

5.3.20 .ENDSCOPE – End Variable Scope

- **Syntax**

  .scope <symbol>

  <statements>

  .endscope

- **Description**

  End the repeating scope that .repeat starts off.

- **See Also**

  .scope

5.3.21 .ENDSTRUCT – End Structure Block

- **Syntax**

  .endstruct

- **Description**

  the repeating scope that . struct starts off.

- **See Also**

  .struct

5.3.22 .EQU – Define an Assembler Constant

- **Syntax**

  <symbol> .equ <expr>

- **Description**

  Define an constant <symbol> and assignment <expr>. <expr> must be an constant that can be calculated at this moment. When the constant is defined, it's value cannot be changed.

- **Example**

  MyInteger .equ 1

5.3.23 .ERROR –Issue A Compilation Error Message

- **Syntax**

  .error ["<message>"]

- **Description**

  Generate an error message. The message should be quoted by a double quotation.

- **Example**

  .error "argument out of range"

5.3.24 .EXPORT – Export Symbol

- **Syntax**

  .export <symbol>

- **Description**

  The effect is same with .extern. The directive is established as an alias for compatibility.

- **See Also**

  .extern

5.3.25 .EXPORTZP – Export Zero Page Symbol

- **Syntax**

  .exportzp <symbol>

- **Description**

  The effect is same with.externzp. The directive is established as an alias for compatibility.

- **See Also**

  .externzp

5.3.26 .EXTERN – Declare External Symbol

- **Syntax**

  .extern <symbol>

- **Description**

  Declare symbol as a global symbol. The external symbol can be defined by its module or called from other module. When numbers of module is linked, the global symbols cannot named the same name or un-assigned.

- **Example**

  This directive is meaningful when modules are linked. The following are explanations of this directives in 3 different files.

```
;----- header.h------
.ifndef HEADER_H
.define HEADER_H
.extern GLOBAL_LABEL
.endif

;------ module1.s ------
.include “header.h”
    jmp GLOBAL_LABEL ;jump to module2

;------ module2.s -----
.include “header.h”
GLOBAL_LABEL:
    nop
```

- **See Also**

  .externzp

5.3.27 .EXTERNZP – Declare Global Zero Page Symbol

- **Syntax**

  .externzp <symbol>

- **Description**

  Declare symbol as a global symbol. When using this symbol, the zero page addressing mode has high priority. While assigning a value, user must limit the value in the range of zero page(0x00 ~ 0xFF). The exceeded value could result in error. The .externzp is for ROM addressing definition , the .extern is for global subroutine definition.

- **See Also**

  .extern

5.3.28 .FLOOR – Unconditional Round Down

- **Syntax**

  <symbol> = .floor(<expr>)

- **Description**

  If <expr> is a float, rounding down unconditionally.

- **Example**

```
Ans = .floor(1.2)
; Ans == 1
```

5.3.29 .HIBYTE – High Byte

- **Syntax**

  <symbol> = .hibyte(<expr>)

- **Description**

  Access a byte from high byte of expr. (bit 8~15)

- **Example**

```
Ans = .hibyte(0x1234)
; Ans == 0x12
```

5.3.30 .IF – Conditional Assembly

- **Syntax**

```
.if(<expr>)
    <statements>
.endif
```

- **Description**

  If <expr> is true then assembles this block.

  <expr> is Boolean type operation , for example, a==b.

- **Example**

```
Tmp = 1 + 2 * 3
.if(Tmp != 7)
    .error
.endif
```

5.3.31 .IFBLANK – Conditional Assembly If Parameter Is Blank

- **Syntax**

```
.ifblank(<symbol>)
    <statements>
```

.endif

- **Description**

  This directive is the abbreviation of .if(.blank(<symbol>))

- **See Also**

  .blank

5.3.32 .IFDEF – Conditional Assembly If Defined

- **Syntax**

  .ifdef(<symbol>)

  <statements>

  .endif

- **Description**

  If <symbol> is defined, assembling this block.

- **Example**

  .ifdef(UNDEFINE_SYM)

  .error

  .endif

- **See Also**

  .if, .defined

5.3.33 .IFNBLANK – Conditional Assembly If Parameter Isn't Blank

- **Syntax**

  .ifnblank(<symbol>)

  <statements>

  .endif

- **Description**

  This directive is the abbreviation of .if(! .blank(<symbol>)).

- **See Also**

  .blank

5.3.34 .IFNDEF – Conditional Assembly If Undefined

- **Syntax**

  .ifndef(<symbol>)

  <statements>

  .endif

- **Description**

  If <symbol> is undefined, assembling the block.

◆ **Example**

```
.ifdef(UNDEFINE_SYM)
        .error
.endif
```

◆ **See Also**

.if, .defined

5.3.35 .IMPORT – Import Symbol

◆ **Syntax**

.import <symbol>

◆ **Description**

The effect is same with .extern. The directive is established as an alias for compatibility.

◆ **See Also**

.extern

5.3.36 .IMPORTZP – Import Zero Page Symbol

◆ **Syntax**

.importzp <symbol>

◆ **Description**

The effect is same with .externzp. The directive is established as an alias for compatibility.

◆ **See Also**

.externzp

5.3.37 .INCBIN – Insert Binary File

◆ **Syntax**

.incbin “<file>”

◆ **Description**

Insert the content of <file> as binary data. The .include directly uses the content of target file, whereas .include the target file as text data. The .incbin usually is applied for binary files such as sound and graphic files.

◆ **Example**

```
L_RES_Voice1:
.incbin “d:\abc\voice1.v8lx”
```

5.3.38 .INCLUDE – Include File

◆ **Syntax**

.include “<file>”

- **Description**

  The <file> must be another original assembly file.The assembler will stop assembling the current file and starting to assemble the included <file>. When the assembly of include file finished, the assembler will return to previous position of previous file.

- **Example**

```
---- a1.h ----
.ifndef A1_H
.define A1_H
; content
.extern G_Func1

.endif

---- a1.s ----
.include “a1.h”
G_Func1:
ret
```

5.3.39 .LOBYTE – Low Byte

- **Syntax**

  <symbol> = .lobyte(<expr>)

- **Description**

  Access one byte from expr low byte. (bit 0~7)

- **Example**

```
Ans = .lobyte(0x1234)
; Ans == 0x34
```

5.3.40 .LOCAL – Declare Local Macro Variable

- **Syntax**

  .local <symbol>

- **Description**

  Declares that the specified symbol is to be considered in local context to the macro. <label> may be identical to another label declared outside the macro definition; there will be no conflict between the two. If the macro is called recursively, each invocation will have its own local copy.

- **Example**

```
.macro M_x1
```

```
    .local LL_exit
    jmp LL_exit
LL_exit:
.endmacro
```

5.3.41 .MACRO – Declare Macro

- **Syntax**

  .macro <symbol> [<arg1>, <arg2>, …]

  <statement>

  .endmacro

- **Description**

  A macro is a sequence of instructions that can be inserted in the assembly source code by using a single macro call. The macro must first be defined, then it can be referred to in subsequent source code. A macro can call another macro, or may call itself recursively.

- **Example**

```
.macro M_LDXY arg_value_x, arg_value_y
    LDX #arg_value_x
    LDY #arg_value_y
.endmacro
```

5.3.42 .MOD – Remainder Operation

- **Syntax**

  <symbol> = <expr1> .mod <expr2>

- **Description**

  Calculate the remainder of expr1 / expr2.

- **Example**

```
ans = 5 .mod 3
; ans == 2
```

5.3.43 .NOT – Boolean Reverse Operation

- **Syntax**

  <symbol> = .not <expr1>

- **Description**

  Calculate the reverse value of expr1.

- **Example**

```
ans = .not 1
; ans == 0
```

5.3.44 .OR – Boolean Or Operation

- **Syntax**

  <symbol> = <expr1> .or <expr2>

- **Description**

  Calculate expr1 || expr2

- **Example**

  ans = 0 .or 1

  ; ans == 1

5.3.45 .ORG – Set Program Origin

- **Syntax**

  .org <expr>

- **Description**

  Set the program origin for subsequent code at the address defined in <expr>. If <label> is specified, it will be given the value of the <expr>. If no ORG is specified, code generation will begin at address zero.

- **Example**

```
.org 0x7e0
    .word L_TM2_INT
.code
L_TM2_INT:
    RTI
```

5.3.46 .REPEAT - Begin a Repeat-Until Loop Block Definition

- **Syntax**

```
.repeat <expr>
    <statement>
.endrepeat
```

- **Description**

  Repeat assembly <statement>, the number of times is assigned by <expr>.

- **Example**

```
.org 0x7e0
    .word L_TM2_INT
.code
L_TM2_INT:
    RTI
```

5.3.47 .RES – Reserve Space

- **Syntax**

  .res <expr1>, <expr2>

- **Description**

  Reserve size of <expr1> in memory and fill in with the value <expr2>.

- **Example**

  ; Reserve 12 bytes of memory with value $AA

  .res 12, $AA

5.3.48 .ROUND – Round

- **Syntax**

  .round(<expr>)

- **Description**

  Round the <expr> up to the nearest value.

5.3.49 .SCOPE – Start Variable Scope

- **Syntax**

  .scope <symbol>

  <statements>

  .endscope

- **Description**

  Start a variable scope. In the range of .scope to .endscope, the new defined symbol can be directly accessed. When the symbol is asccessed outside, it must add a prefix word of scope. The name of scope cannot conflict with the rest of symbols.

  Example

```
.scope Error       ; Start new scope named Error
    None = 0
    File = 1
    Parse = 2
.endscope          ; close scope

LDA #Error::File   ; use symbol from scope Error
```

- **See Also**

  .endscope

5.3.50 .SEGMENT – Program Segment

- **Syntax**

  .segment “<symbol>”

◆ **Description**

Switch to another program segment. The .segment directive has to be named with a string. The available names is relative to the selected IC. Please refer to IC document.

◆ **Example**

```
.segment “tm0_int”
        .word L_tm0_int
.code
L_tm0_int:
```

◆ **See Also**

.code

5.3.51 .SETCPU – Setup CPU

◆ **Syntax**

.setcpu <symbol>

◆ **Description**

Label the IC Body in front of the file. This directive can only declare once.

◆ **Example**

```
.setcpu NY8L030A
```

5.3.52 .SHL – Left Shift

◆ **Syntax**

<expr1> .shl <expr2>

◆ **Description**

Calculate the result as <expr1> left shifts by <expr2>.

◆ **Example**

```
Result = 2 .shl 1
; Result = 4
```

5.3.53 .SHR – Right Shift

◆ **Syntax**

<expr1> .shr <expr2>

◆ **Description**

Calculate the result as <expr1> right shifts by <expr2>.

◆ **Example**

```
Result = 2 .shr 1
; Result = 1
```

5.3.54 .STRING – Access String

- **Syntax**

  .string(<symbol>)

- **Description**

  Get the defined string of<symbol>. It's applied to macro usually, user can get the string that it defined.

- **Example**

```
.macro M_inc_v8lx name
    .incbin .string(name)
.endmacro
```

5.3.55 .WORD - Word

- **Syntax**

  .word <expr1>

- **Description**

  Write a data of one word (2-bit) at current location, the content is <expr1>.

- **Example**

```
.word 0x12EF
```

5.3.56 .XOR – Boolean Exclusive Or

- **Syntax**

  <expr1> .xor <expr2>

- **Description**

  Calculate <expr1> exclusive or <expr2> outcome.

- **Example**

```
Result = 0 .xor 1
; Result = 1
```

6 Macro Language

Macros are user defined sets of instructions and directives that will be evaluated in-line with the assembler source code whenever the macro name is invoked. Macros consist of sequences of assembler instructions and directives. They can be written to accept arguments, making them quite flexible. Their advantages are:

• Higher levels of abstraction, improving readability and reliability.

• Consistent solutions to frequently performed functions.

• Simplified changes.

• Improved testability.

Applications might include creating complex tables, frequently used code, and complex operations.

6.1 Macro Syntax for NY4, NY5, NY7, NY8A, NY9

NYASM macros are defined according to the following syntax:

<label> macro [<arg1>,<arg2> ..., <argn>]

:

:

endm

Where <label> is a valid *NYASM* label and <arg> is any number of optional arguments supplied to the macro. The values assigned to these arguments at the time the macro name is invoked will be substituted wherever the argument name occurs in the body of the macro. The body of a macro may be comprised of *NYASM* directives, or *NYASM* Macro Directives (LOCAL for example). Refer to Chapter 5.2. *NYASM* continues to process the body of the macro until an EXITM or ENDM directive is encountered.

Note: Forward references to macros are not permitted.

6.1.1 Macro Directives

There are directives that are unique to macro definitions. They cannot be used out of the macro context (refer to Chapter 5.2.1 for details concerning these directives):

• MACRO

• LOCAL

• EXITM

• ENDM

When writing macros, you can use any of these directives PLUS any other directives supported by *NYASM*.

6.1.2 Text Substitution

String replacement and expression evaluation may appear within the body of a macro. Arguments

may be used anywhere within the body of the macro.

Command	Description
<arg>	Substitute the argument text supplied as part of the macro invocation.

```
define_table macro num_of_entry
        local a = 0
        while a < num_of_entry
        dw 0
        a += 1
        endw
endm
```

when invoked, would generate:

```
dw 0 ; 1st
dw 0 ; 2nd
   :
   :
dw 0 ; (num_of_entry-1)-th
dw 0 ; (num_of_entry)-th
```

6.1.3 Macro Usage

Once the macro has been defined, it can be invoked at any point within the source module by using a macro call, as described below:

<macro_name> [<arg>, ..., <arg>]

where <macro_name> is the name of a previously defined macro and arguments are supplied as required. The macro call itself will not occupy any locations in memory. However, the macro expansion will begin at the current memory location. Commas may be used to reserve an argument position. The EXITM directive (see Chapter 5) provides an alternate method for terminating a macro expansion. During a macro expansion, this directive causes expansion of the current macro to stop and all code between the EXITM and the ENDM directives for this macro to be ignored. If macros are nested, EXITM causes code generation to return to the previous level of macro expansion.

6.2 Macro Syntax for NY8L

6.2.1 MACRO Syntax

NY8L macros are defined according to the following syntax:

```
.macro <label> [<arg1>,<arg2> ..., <argn>]
:
```

:

.endmacro

Where <label> is a valid *NYASM* label and <arg> is any number of optional arguments supplied to the macro. The values assigned to these arguments at the time the macro name is invoked will be substituted wherever the argument name occurs in the body of the macro. The body of a macro may be comprised of *NYASM* directives, or *NYASM* Macro Directives (LOCAL for example). Refer to Chapter 5.3. *NYASM* continues to process the body of the macro until an EXITM or ENDM directive is encountered.

Note: Forward references to macros are not permitted.

6.2.2 Macro Directives

There are directives that are unique to macro definitions. They cannot be used out of the macro context (refer to Chapter 5.3 for details concerning these directives):

.MACRO – Declare Macro

.ENDMACRO – End Macro Defined Block

.LOCAL – Declare Local Macro Variable

.IFBLANK – Conditional Assembly If Parameter Is Blank

.IFNBLANK – Conditional Assembly If Parameter Isn't Blank

.BLANK – Check Blank Symbol

When writing macros, you can use any of these directives PLUS any other directives supported by *NYASM*.

6.2.3 Text Substitution

String replacement and expression evaluation may appear within the body of a macro. Arguments may be used anywhere within the body of the macro.

Command	Description
<arg>	Substitute the argument text supplied as part of the macro invocation.

◆ **Example**

```
.macro CAJE value, label
        cmp #value
        jz     label
.endmacro
```

6.2.4 Macro Usage

Once the macro has been defined, it can be invoked at any point within the source module by using a macro call, as described below:

<macro_name> [<arg>, ..., <arg>]

where <macro_name> is the name of a previously defined macro and arguments are supplied as required. The macro calls will not occupy any locations in memory. However, the macro expansion will begin at the current memory location. Commas may be used to reserve an argument position. The used parameters of macro calls can be less than the definied parameters, users can check the designated parameters whether are delivered from caller by .blank.

7 Expression Syntax and Operation

This chapter describes various expression formats, syntax, and operations used by *NYASM*.

7.1 NY4, NY5, NY7, NY8A, NY9

Content:

7.1.1 Numeric Constants and Radix

NYASM supports the following radix forms: hexadecimal, decimal, octal and binary. The default radix is decimal the default radix determines what value will be assigned to constants in the object file when a radix is not explicitly specified by a base descriptor. *NYASM* only supports unsigned constants and the values are assumed to be positive.

The following table presents the various radix specifications:

Table 7.1: Radix Specifications

Type	Syntax	Example
Decimal	D'<digits>'	D'100'
Hexadecimal	H'<hex_digits>' 0x<hex_digits> <hex_digits>h	H'9f' 0x9f 9fh
Octal	O'<octal_digits>'	O'777'
Binary	B'<binary_digits>' <binary_digits>b	B'00111001' 00111001b

Table 7.2: Arithmetic Operators and Precedence

Operator		Example
$	Current/Return program counter	goto $ + 3
(	Left Parenthesis	1 + (d * 4)
)	Right Parenthesis	(Length + 1) * 256
!	Item NOT (logical complement)	if ! (a == b)
-	Negation (2's complement)	-1 * Length
~	Complement	flags = ~flags
high	Return high byte of a 24-bit value	mvma high(0x121314) ;accumulator will contain 0x12

Operator		Example
mid	Return mid byte of a 24-bit value	mvma mid(0x121314) ;accumulator will contain 0x13
low	Return low byte of a 24-bit value	mvma low(0x121314) ;accumulator will contain 0x14
high0	Return low nibble of high byte of a 24-bit value	mvma high0(0x123456) ;accumulator will contain 0x2
high1	Return high nibble of high byte of a 24-bit value	mvma high1(0x123456) ;accumulator will contain 0x1
mid0	Return low nibble of middle byte of a 24-bit value	mvma mid0(0x123456) ;accumulator will contain 0x4
mid1	Return high nibble of middle byte of a 24-bit value	mvma mid1(0x123456) ;accumulator will contain 0x3
low0	Return low nibble of low byte of a 24-bit value	mvma low0(0x123456) ;accumulator will contain 0x6
low1	Return high nibble of low byte of a 24-bit value	mvma low1(0x123456) ;accumulator will contain 0x5
*	Multiply	a = b * c
/	Divide	a = b / c
%	Modulus	entry_len = tot_len % 16
+	Add	tot_len = entry_len * 8 + 1
-	Subtract	entry_len = (tot - 1) / 8
<<	Left shift	flags = flags << 1
>>	Right shift	flags = flags >> 1
>=	Greater or equal	if entry_idx >= num_entries
>	Greater than	if entry_idx > num_entries
<	Less than	if entry_idx < num_entries
<=	Less or equal	if entry_idx <= num_entries
==	Equal to	if entry_idx == num_entries
!=	Not equal to	if entry_idx != num_entries
&	Bitwise AND	flags = flags & ERROR_BIT
^	Bitwise exclusive OR	flags = flags ^ ERROR_BIT
|	Bitwise inclusive OR	flags = flags | ERROR_BIT
&&	Logical AND	if (len == 512) && (b == c)
||	Logical OR	if (len == 512) || (b == c)
=	Set equal to	entry_index = 0
+=	Add to, set equal	entry_index += 1
-=	Subtract, set equal	entry_index -= 1
*=	Multiply, set equal	entry_index *= entry_length

Operator		Example
/=	Divide, set equal	entry_total /= entry_length
%=	Modulus, set equal	entry_index %= 8
<<=	Left shift, set equal	flags <<= 3
>>=	Right shift, set equal	flags >>= 3
&=	AND, set equal	flags &= ERROR_FLAG
|=	Inclusive OR, set equal	flags |= ERROR_FLAG
^=	Exclusive OR, set equal	flags ^= ERROR_FLAG
++	Increment	i ++
--	Decrement	i --

7.1.2 High/Mid/Low

- **Syntax**

  high <operand>

  mid <operand>

  low <operand>

- **Description**

  These operators are used to return one byte of a multi-byte label value. This is done to handle dynamic pointer calculations as might be used with table read and write instructions.

7.1.3 Increment/Decrement (++/--)

- **Syntax**

  <variable>++

  <variable>--

- **Description**

  Increments or decrements a variable value. These operators can only be used on a line by themselves; they cannot be embedded within other expression evaluation.

- **Example**

```
LoopCount = 4
while LoopCount > 0
   nop
   LoopCount --
Endw
```

7.2 NY8L

Content:

7.2.1 Numeric constants and Radix

NYASM supports the following radix forms: hexadecimal, decimal and binary. The default radix is decimal the default radix determines what value will be assigned to constants in the object file when a radix is not explicitly specified by a base descriptor. *NYASM* only supports unsigned constants and the values are assumed to be positive.

Table 7.3 Radix Specifications

Type	Syntax	Example
Decimal	<digits>	100
Hexadecimal	$<hex_digits>’ 0x<hex_digits>	$9f 0x9f
Binary	%<binary_digits>’	%00111001

Table 7.4 Arithmetic Operators and Precedence

Operator		Example
(	Left Parenthesis	1 + (d * 4)
)	Right Parenthesis	(Length + 1) * 256
!	Item NOT (logical complement)	if ! (a == b)
-	Negation (2’s complement)	-1 * Length
~ .bitnot	Complement	flags = ~flags
.bankbyte	Return high byte of a 24-bit value	mvma high(0x121314) ;accumulator will contain 0x12
.hibyte	Return mid byte of a 24-bit value	mvma mid(0x121314) ;accumulator will contain 0x13
.lobyte	Return low byte of a 24-bit value	mvma low(0x121314) ;accumulator will contain 0x14
*	Multiply	a = b * c
/	Divide	a = b / c
%	Modulus	entry_len = tot_len % 16
+	Add	tot_len = entry_len * 8 + 1
-	Subtract	entry_len = (tot - 1) / 8

Operator		Example
<<	Left shift	flags = flags << 1
>>	Right shift	flags = flags >> 1
>=	Greater or equal	if entry_idx >= num_entries
>	Greater than	if entry_idx > num_entries
<	Less than	if entry_idx < num_entries
<=	Less or equal	if entry_idx <= num_entries
==	Equal to	if entry_idx == num_entries
!=	Not equal to	if entry_idx != num_entries
& .bitand	Bit AND	flags = flags & ERROR_BIT
^	Bit mutex OR	flags = flags ^ ERROR_BIT
| .bitor	Bit OR	flags = flags | ERROR_BIT
&& .and	Logical AND	if (len == 512) && (b == c)
|| .or	Logical OR	if (len == 512) || (b == c)
.round	Round	.round(2.345)
.ceil	Unconditional carry	.ceil(2.345)
.floor	Unconditional round down	.floor(2.345)

7.2.2 High/Mid/Low

- **Syntax**

  .bankbyte <operand>

  .hibyte <operand>

  .lobyte <operand>

- **Description**

  These operators are used to return one byte of a multi-byte label value. This is done to handle dynamic pointer calculations as might be used with table read and write instructions.

8 Revision History

Version	Date	Description	Modified Page
1.00	2007/12/20	The first version.	-
1.01	2009/10/12	Revision.	-
1.1	2010/01/11	Add NY4 series MCU.	75
1.2	2010/07/20	1. Add NY4/NY5 series new bodies. 2. Add NYASM Errors/Warnings.	75 61
1.3	2010/08/17	Windows 7 complied.	11
1.4	2012/02/29	Modify NY5B/5C series MCU.	75
1.5	2013/06/25	1. Please use Windows XP or above operating system version. 2. Add NY4(B) and NY7 series to MCU List.	11 75
1.6	2013/08/16	Modify NY7 series of MCU List.	59
1.7	2014/02/24	1. Modify the example of MACRO. 2. Modify the default of radix as decimal. 3. Modify Errors/Warnings messages. 4. Add “Forward reference” to Glossary.	35 37, 65, 87 79 86
1.8	2014/05/16	Add NY8 series MCU.	78
1.9	2014/11/14	Change the IC bodies of MCU List: NY4B018C, NY4B038C, NY4B058C, NY5C158C, NY5C185C, NY5C345C.	75
2.0	2015/01/29	1. Add binary representation 2. Modify the examples of arithmetic operators.	65
2.1	2015/07/27	Modify UI description.	17
2.2	2015/11/20	Add NY9UB series MCU.	79
2.3	2016/01/27	1. Remove NY4xxxxA/NY5xxxxA series, keep NY5AxxxA series. 2. Add NY8 series MCU.	- 61

Version	Date	Description	Modified Page
2.4	2016/05/20	1. Add NY6 series MCU. 2. Add NY8A051C/51D MCU.	60 62
2.5	2016/08/22	Add NY8A53D MCU.	63
2.6	2016/11/18	1. Remove NY5AA series. 2. Add NY9UP01A MCU.	- 79
2.7	2017/05/23	1. Support NY8L series MCU. 2. Add NY8A054A MCU. 3. Add NY8L series MCU.	42, 62, 66 78 78
2.8	2017/08/09	Add NY8A054D MCU.	78
2.9	2017/11/17	1. Modify List Directive Options and NYASM Assembler Options. 2. Add NY8A051E MCU.	34, 74 81
3.0	2018/02/08	Add NY8B062D MCU.	81
3.1	2018/08/27	1. Remove DT command. 2. Remove NY8A057A, NY8B073A, NY8B074A MCU. 3. Remove NY6C450A ~ NY6C720A MCU.	- - -
3.2	2018/11/21	Add NY8B062A MCU.	80
3.3	2019/02/19	Add NY8A051F, NY9UP08A MCU.	78, 79
3.4	2019/05/28	Add NY5P025J, NY5P055J, NY5P085J, NY5B035C, NY5B045C, NY8A050D, NY8AE51D and NY8B062B MCU.	76
3.5	2019/08/22	Remove NY8L005A, NY8L010A and add NY8LP10A, NY8LP11A.	-
3.6	2019/11/14	Add NY5P series, NY5A018C, NY5A025C and NY8BM72A.	76
.3.7	2020/03/16	Add NY5AC and NY5BC MCU.	77
3.8	2020/08/18	1. Add the command description of.bitor and .word. 2. Add NY6 series, NY8A054E, and NY8B061D IC.	44, 59 77

Version	Date	Description	Modified Page
3.9	2020/11/12	1. Remove NY4B018B / NY5AxxxB / NY5BxxxB / NY5C112B / 132B / 158B / 185B/ 225B / 265B / 305B / 345B IC. 2. Add NY8A053E / NY9UP02A IC.	- 82
4.0	2021/01/27	1. Remove NY6A003A / NY6A005A / NY8L050A IC. 2. Add NY8B062E / NY8TM52D IC.	- 83
4.1	2021/05/18	Add NY5QxxxA / NY8B060E / NY8BE62D IC.	80
4.2	2021/09/10	1. Remove DATA, DB and DN commands. 2. Add NY5Q020A.	- 79
4.3	2021/11/11	Add NY8TE64A IC.	82
4.4	2022/02/22	Add NY5Q026A, NY5Q046A, NY5Q080A, NY5Q160A, NY8A051H, NY8AE51F。	79
4.5	2022/05/19	1. System requirement adds Microsoft Win11. 2. Add NY8B060D.	9 82
4.6	2022/08/24	Add NY8B061E.	82
4.7	2022/11/28	Add NY4P045C, NY8B062F.	77
4.8	2023/02/15	Add NY4P018C, NY4P065C, NY4P085C, NY4P105C and NY8A050E.	77
4.9	2023/05/15	Fix incorrect description.	-
5.0	2023/08/21	Add NY8A052E.	83
5.1	2024/02/22	1. Add the .align2 command. 2. Add NY8BM61D and NY8BM62D. 3. Remove NY8L020A and NY8L030A.	39 84 -
5.2	2024/08/22	Remove NY5P520J, NY5P720J, NY5P1K0J, NY5P1K2J, NY5C450B, NY5C520B, NY5C640B, NY5C720B, NY7C450A, NY7C520A, NY8A051A, NY8A051C, NY8A051E, NY8A053A, NY8AE51D, NY8B060E, NY8B061D and NY8B071A.	-

Version	Date	Description	Modified Page
5.3	2025/02/27	Add NY5Q019A, NY5Q039A, NY5Q079A, NY5Q159A, NY8A051J, NY8A051K, NY8A051L, NY8LP08.	80, 82, 83
5.4	2025/05/27	1. Add an assembler directive. 2. Add NY8A051H1, NY8B062F1, NY8BM84A.,	28 82, 83
5.5	2025/08/27	Add NY8A054E1, NY8F2481.	83, 84

Appendix A - Quick Reference

This appendix lists abbreviated information on *NYASM* and MCU instruction sets for use in developing applications using *NYASM*.

Content:

A.1 *NYASM* Quick Reference

The following Quick Reference Guide gives all the instructions, directives, and command list options for *NYASM* Assembler.

Table A.1: *NYASM* Directive Language Summary

Directive	Description	Syntax
CONTROL DIRECTIVES		
CONSTANT	Declare symbol constant.	constant <label>[=<expr>,...,<label>[=<expr>]]
#DEFINE	Define a text substitution label.	#define <name> [<value>] #define <name> [<arg>,...,<arg>]
END	End program block.	end
EQU	Define an assemble constant.	<label> equ <expr>
ERROR	Issue an error message.	error "<text_string>"
#INCLUDATA	Include binary data file.	#includata "<data_file>" [,<address>]
#INCLUDE	Include additional source file.	#include "<include_file>"
LIST	Listing options.	list [<list_option>,...,<list_option>]
MESSG	Create user defined message.	messg "<message_text>"
ORG	Set program origin.	[<label>:] org <expr>
LINES	Re-declare line-per-page.	lines <value>
NEWPAGE	Re-declare line-per-page.	Newpage <value>
RADIX	Specify default radix.	radix <default_radix>
SUBTITLE	Specify program subtitle.	subtitle "<sub_text>"
TITLE	Specify program title.	title "<title_text>"
#UNDEFINE	Delete a substitution label.	#undefine <label>
VARIABLE	Declare symbol variable.	variable <label>[=<expr>,...,<label>[=<expr>]]

Directive	Description	Syntax
CONDITIONAL ASSEMBLY		
BREAK	Escape from a **FOR**, **WHILE** or **REPEAT-UNTIL** loop, or Jump to the end of a **SWITCH** block.	break [<Boolean expression>]
CASE	Part of a **SWITCH** block; must use **CASE** with **SWITCH**.	switch <expression> case <expression 1>[,<expression 2>] <statements>
CONTINUE	Jump to the begin of **FOR**, **WHILE** or **REPEAT-UNTIL** loop that contains **CONTINUE** directive. All statements behind **CONTINUE** in a loop are ignored.	continue [<Boolean expression>]
DEFAULT	Part of a **SWITCH** block; must use **DEFAULT** with **SWITCH**. Begin default assembly block to **SWITCH.**	default <statements>
ELSE	Begin alternative assembly block to **IF**.	else <statements>
ENDFOR	End a **FOR** loop.	endfor
ENDIF	End conditional assembly block.	endif
ENDS	Directive for coding convenience: presenting **ENDFOR, ENDW, ENDSW, ENDIF**.	ends
ENDSW	End conditional switching assembly block.	endsw
ENDW	End a **WHILE** loop.	endw
FOR	Perform counting loop **FOR**.	for <iterator> = <expr1> to <expr2> [step <expr3>]
IF	Begin conditionally assembled code block.	if <expr>
IFDEF	Execute if symbol has been defined.	ifdef <label>
IFNDEF	Execute If symbol has not been defined.	ifndef <label>
REPEAT	Begin at-least-one-time loop.	Repeat <statements> until <Boolean expression>
SWITCH	Begin conditional switching assembly block.	switch <expr>
UNTIL	End at-least-one-time loop if condition is true.	Repeat <statements> until <Boolean expression>
WHILE	Perform loop **WHILE** condition is true.	while <expr>
DATA		
CBLOCK	Define a block of constants.	cblock [<expr>]
DW	Declare data of one word.	[<label>] dw <expr>[,<expr>,...,<expr>]
ENDC	End an automatic constant block.	endc
MACRO		
ENDM	End a macro definition.	endm
EXITM	Exit from a macro.	exitm
EXPAND	Expand macro listing.	expand

Directive	Description	Syntax
LOCAL	Declare local macro variable.	local <label>[,<label>]
MACRO	Declare macro definition.	<label> macro [<arg>,...,<arg>]
MAXMACRODEPTH	Setup the maximum depth of macro expansion.	Maxmacrodepth [=] <expr>
NOEXPAND	Turn off macro expansions.	noexpand

Table A.2: NYASM Assembler Options:

OPTION	DEFAULT	DESCRIPTION
c	Off	Enable/Disable case sensitivity c=on Enable c=off Disable
p	None	Set the processor type: /p=<processor_type> where <processor_type> is an Nyquest MCU device. For example, NY5A005A.
unlockrsvmem	Locked	/unlockrsvmem For 4-bit MCU only. Allow the programming right in reserved memory area.
nocfgblk	Configuration block required	/nocfgblk For 4-bit MCU only. Ignore the assembly time check for the existence of configuration block file.

Table A.3: Radix Types Supported

Type	Syntax	Example
Decimal	D'<digits>'	D'100'
Hexadecimal	H'<hex_digits>' 0x<hex_digits> <hex_digits>h	H'9f' 0x9f 9fh
Octal	O'<octal_digits>'	O'777'
Binary	B'<binary_digits>'	B'00111001'

Table A.4: NYASM Arithmetic Operators

Operator		Example
$	Current/Return program counter	goto $ + 3
(	Left Parenthesis	1 + (d * 4)
)	Right Parenthesis	(Length + 1) * 256
!	Item NOT (logical complement)	if ! (a == b)
-	Negation (2's complement)	-1 * Length
~	Complement	flags = ~flags

Operator		Example
high	Return high byte of a 24-bit value	mvma high 0x121314 ;accumulator will contain 0x12
mid	Return mid byte of a 24-bit value	mvma mid 0x121314 ;accumulator will contain 0x13
low	Return low byte of a 24-bit value	mvma low 0x121314 ;accumulator will contain 0x14
high0	Return low nibble of high byte of a 24-bit value	mvma high0 0x123456 ;accumulator will contain 0x2
high1	Return high nibble of high byte of a 24-bit value	mvma high1 0x123456 ;accumulator will contain 0x1
mid0	Return low nibble of middle byte of a 24-bit value	mvma mid0 0x123456 ;accumulator will contain 0x4
mid1	Return high nibble of middle byte of a 24-bit value	mvma mid1 0x123456 ;accumulator will contain 0x3
low0	Return low nibble of low byte of a 24-bit value	mvma low0 0x123456 ;accumulator will contain 0x6
low1	Return high nibble of low byte of a 24-bit value	mvma low1 0x123456 ;accumulator will contain 0x5
*	Multiply	a = b * c
/	Divide	a = b / c
%	Modulus	entry_len = tot_len % 16
+	Add	tot_len = entry_len * 8 + 1
-	Subtract	entry_len = (tot - 1) / 8
<<	Left shift	flags = flags << 1
>>	Right shift	flags = flags >> 1
>=	Greater or equal	if entry_idx >= num_entries
>	Greater than	if entry_idx > num_entries
<	Less than	if entry_idx < num_entries
<=	Less or equal	if entry_idx <= num_entries
==	Equal to	if entry_idx == num_entries
!=	Not equal to	if entry_idx != num_entries
&	Bitwise AND	flags = flags & ERROR_BIT
^	Bitwise exclusive OR	flags = flags ^ ERROR_BIT
|	Bitwise inclusive OR	flags = flags | ERROR_BIT
&&	Logical AND	if (len == 512) && (b == c)
||	Logical OR	if (len == 512) || (b == c)

Operator		Example
=	Set equal to	entry_index = 0
+=	Add to, set equal	entry_index += 1
-=	Subtract, set equal	entry_index -= 1
*=	Multiply, set equal	entry_index *= entry_length
/=	Divide, set equal	entry_total /= entry_length
%=	Modulus, set equal	entry_index %= 8
<<=	Left shift, set equal	flags <<= 3
>>=	Right shift, set equal	flags >>= 3
&=	AND, set equal	flags &= ERROR_FLAG
|=	Inclusive OR, set equal	flags |= ERROR_FLAG
^=	Exclusive OR, set equal	flags ^= ERROR_FLAG
++	Increment	i ++
--	Decrement	i --

A.2 MCU List

Table A.5: MCU List

No.	IC type	PROG ROM size	DATA ROM size	Reserved Memory	I/O Pin Count
1	NY4P018C	16K x 10	48K x 10	0x001F--0x07FF	8 I/O
2	NY4P045C	16K x 10	112K x 10	0x001F--0x07FF	8 I/O
3	NY4P065C	16K x 10	160K x 10	0x001F--0x07FF	8 I/O
4	NY4P085C	16K x 10	208K x 10	0x001F--0x07FF	8 I/O
5	NY4P105C	16K x 10	256K x 10	0x001F--0x07FF	8 I/O
6	NY4A003B	12K x 10	12K x 10	0x001F--0x07FF	4 I/O
7	NY4A005B	16K x 10	16K x 10	0x001F--0x07FF	4 I/O
8	NY4A008B	16K x 10	24K x 10	0x001F--0x07FF	4 I/O
9	NY4A011B	16K x 10	32K x 10	0x001F--0x07FF	4 I/O
10	NY4B003B	12K x 10	12K x 10	0x001F--0x07FF	8 I/O
11	NY4B005B	16K x 10	16K x 10	0x001F--0x07FF	8 I/O
12	NY4B008B	16K x 10	24K x 10	0x001F--0x07FF	8 I/O
13	NY4B011B	16K x 10	32K x 10	0x001F--0x07FF	8 I/O
14	NY4B018C	16K x 10	48K x 10	0x001F--0x07FF	8 I/O
15	NY4B025B	16K x 10	64K x 10	0x001F--0x07FF	8 I/O
16	NY4B038C	16K x 10	96K x 10	0x001F--0x07FF	8 I/O
17	NY4B045B	16K x 10	112K x 10	0x001F--0x07FF	8 I/O
18	NY4B058C	16K x 10	144K x 10	0x001F--0x07FF	8 I/O
19	NY4B065B	16K x 10	160K x 10	0x001F--0x07FF	8 I/O
20	NY4B075B	16K x 10	184K x 10	0x001F--0x07FF	8 I/O
21	NY4B085B	16K x 10	208K x 10	0x001F--0x07FF	8 I/O
22	NY4B095B	16K x 10	232K x 10	0x001F--0x07FF	8 I/O
23	NY4B105B	16K x 10	256K x 10	0x001F--0x07FF	8 I/O
24	NY4B115B	16K x 10	280K x 10	0x001F--0x07FF	8 I/O
25	NY4B125B	16K x 10	304K x 10	0x001F--0x07FF	8 I/O
26	NY4B145B	16K x 10	352K x 10	0x001F--0x07FF	8 I/O
27	NY4B165B	16K x 10	400K x 10	0x001F--0x07FF	8 I/O
28	NY5P025B	16K x 10	64K x 10	0x001F--0x0BFF	16 I/O
29	NY5P055B	16K x 10	136K x 10	0x001F--0x0BFF	16 I/O
30	NY5P085B	16K x 10	208K x 10	0x001F--0x0BFF	16 I/O
31	NY5P185B	16K x 10	448K x 10	0x001F--0x0BFF	16 I/O
32	NY5P025J	16K x 10	64K x 10	0x001F--0x0BFF	16 I/O
33	NY5P055J	16K x 10	136K x 10	0x001F--0x0BFF	16 I/O

No.	IC type	PROG ROM size	DATA ROM size	Reserved Memory	I/O Pin Count
34	NY5P085J	16K x 10	208K x 10	0x001F--0x0BFF	16 I/O
35	NY5P185J	16K x 10	448K x 10	0x001F--0x0BFF	16 I/O
36	NY5P345J	16K x 10	832K x 10	0x001F--0x0BFF	16 I/O
37	NY5A003C	12K x 10	12K x 10	0x001F--0x0BFF	7+1 I/O
38	NY5A005C	16K x 10	16K x 10	0x001F--0x0BFF	7+1 I/O
39	NY5A008C	16K x 10	24K x 10	0x001F--0x0BFF	7+1 I/O
40	NY5A011C	16K x 10	32K x 10	0x001F--0x0BFF	7+1 I/O
41	NY5A018C	16K x 10	48K x 10	0x001F--0x0BFF	7+1 I/O
42	NY5A025C	16K x 10	64K x 10	0x001F--0x0BFF	7+1 I/O
43	NY5A035C	16K x 10	88K x 10	0x001F--0x0BFF	7+1 I/O
44	NY5A045C	16K x 10	112K x 10	0x001F--0x0BFF	7+1 I/O
45	NY5A055C	16K x 10	136K x 10	0x001F--0x0BFF	7+1 I/O
46	NY5A065C	16K x 10	160K x 10	0x001F--0x0BFF	7+1 I/O
47	NY5B005C	16K x 10	16K x 10	0x001F--0x0BFF	14+1 I/O
48	NY5B008C	16K x 10	24K x 10	0x001F--0x0BFF	14+1 I/O
49	NY5B011C	16K x 10	32K x 10	0x001F--0x0BFF	14+1 I/O
50	NY5B018C	16K x 10	48K x 10	0x001F--0x0BFF	14+1 I/O
51	NY5B025C	16K x 10	64K x 10	0x001F--0x0BFF	14+1 I/O
52	NY5B035C	16K x 10	88K x 10	0x001F--0x0BFF	14+1 I/O
53	NY5B046C	16K x 10	112K x 10	0x001F--0x0BFF	14+1 I/O
54	NY5B055C	16K x 10	136K x 10	0x001F--0x0BFF	14+1 I/O
55	NY5B065C	16K x 10	160K x 10	0x001F--0x0BFF	14+1 I/O
56	NY5B075C	16K x 10	184K x 10	0x001F--0x0BFF	14+1 I/O
57	NY5B085C	16K x 10	208K x 10	0x001F--0x0BFF	14+1 I/O
58	NY5B112C	16K x 10	272K x 10	0x001F--0x0BFF	14+1 I/O
59	NY5B132C	16K x 10	320K x 10	0x001F--0x0BFF	14+1 I/O
60	NY5B158C	16K x 10	384K x 10	0x001F--0x0BFF	14+1 I/O
61	NY5B185C	16K x 10	448K x 10	0x001F--0x0BFF	14+1 I/O
62	NY5C112C	16K x 10	272K x 10	0x001F--0x0BFF	19+1 I/O
63	NY5C132C	16K x 10	320K x 10	0x001F--0x0BFF	19+1 I/O
64	NY5C158C	16K x 10	384K x 10	0x001F--0x0BFF	19+1 I/O
65	NY5C185C	16K x 10	448K x 10	0x001F--0x0BFF	19+1 I/O
66	NY5C225C	16K x 10	544K x 10	0x001F--0x0BFF	19+1 I/O
67	NY5C265C	16K x 10	640K x 10	0x001F--0x0BFF	19+1 I/O
68	NY5C305C	16K x 10	736K x 10	0x001F--0x0BFF	19+1 I/O
69	NY5C345C	16K x 10	832K x 10	0x001F--0x0BFF	19+1 I/O

No.	IC type	PROG ROM size	DATA ROM size	Reserved Memory	I/O Pin Count
70	NY5Q019A	48K x 10	48K x 10	0x001F—0x07FF	8 I/O
71	NY5Q020A	48K x 10	48K x 10	0x001F—0x07FF	8 I/O
72	NY5Q026A	64K x 10	64K x 10	0x001F—0x07FF	4 I/O
73	NY5Q039A	64K x 10	96K x 10	0x001F—0x07FF	8 I/O
74	NY5Q040A	64K x 10	96K x 10	0x001F—0x07FF	8 I/O
75	NY5Q046A	64K x 10	112K x 10	0x001F—0x07FF	12 I/O
76	NY5Q060A	64K x 10	144K x 10	0x001F—0x07FF	16 I/O
77	NY5Q079A	64K x 10	192K x 10	0x001F—0x07FF	12 I/O
78	NY5Q080A	64K x 10	192K x 10	0x001F—0x07FF	12 I/O
79	NY5Q092A	64K x 10	224K x 10	0x001F—0x07FF	16 I/O
80	NY5Q159A	64K x 10	384K x 10	0x001F—0x07FF	12 I/O
81	NY5Q160A	64K x 10	384K x 10	0x001F—0x07FF	12 I/O
82	NY5Q172A	64K x 10	416K x 10	0x001F—0x07FF	16 I/O
83	NY5Q342A	64K x 10	832K x 10	0x001F—0x07FF	20 I/O
84	NY6P025A	64K x 10	64K x 10	0x001E—0x03FF	9 I/O
85	NY6P025J	64K x 10	64K x 10	0x001E—0x03FF	16 I/O
86	NY6P055J	136K x 10	136K x 10	0x001E—0x03FF	16 I/O
87	NY6P085J	208K x 10	208K x 10	0x001E—0x03FF	16 I/O
88	NY6P185J	448K x 10	448K x 10	0x001E—0x03FF	24 I/O
89	NY6P345J	832K x 10	832K x 10	0x001E—0x03FF	24 I/O
90	NY6A008A	24K x 10	24K x 10	0x001E—0x03FF	8 I/O
91	NY6A011A	32K x 10	32K x 10	0x001E—0x03FF	8 I/O
92	NY6A018A	48K x 10	48K x 10	0x001E—0x03FF	8 I/O
93	NY6A025A	64K x 10	64K x 10	0x001E—0x03FF	8 I/O
94	NY6A035A	88K x 10	88K x 10	0x001E—0x03FF	8 I/O
95	NY6A045A	112K x 10	112K x 10	0x001E—0x03FF	8 I/O
96	NY6A055A	136K x 10	136K x 10	0x001E—0x03FF	8 I/O
97	NY6A065A	160K x 10	160K x 10	0x001E—0x03FF	8 I/O
98	NY6B005A	16K x 10	16K x 10	0x001E—0x03FF	16 I/O
99	NY6B008A	24K x 10	24K x 10	0x001E—0x03FF	16 I/O
100	NY6B011A	32K x 10	32K x 10	0x001E—0x03FF	16 I/O
101	NY6B018A	48K x 10	48K x 10	0x001E—0x03FF	16 I/O
102	NY6B025A	64K x 10	64K x 10	0x001E—0x03FF	16 I/O
103	NY6B035A	88K x 10	88K x 10	0x001E—0x03FF	16 I/O
104	NY6B045A	112K x 10	112K x 10	0x001E—0x03FF	16 I/O

No.	IC type	PROG ROM size	DATA ROM size	Reserved Memory	I/O Pin Count
105	NY6B055A	136K x 10	136K x 10	0x001E—0x03FF	16 I/O
106	NY6B065A	160K x 10	160K x 10	0x001E—0x03FF	16 I/O
107	NY6B075A	184K x 10	184K x 10	0x001E—0x03FF	16 I/O
108	NY6B085A	208K x 10	208K x 10	0x001E—0x03FF	16 I/O
109	NY6C112A	272K x 10	272K x 10	0x001E—0x03FF	24 I/O
110	NY6C132A	320K x 10	320K x 10	0x001E—0x03FF	24 I/O
111	NY6C158A	384K x 10	384K x 10	0x001E—0x03FF	24 I/O
112	NY6C185A	448K x 10	448K x 10	0x001E—0x03FF	24 I/O
113	NY6C225A	544K x 10	544K x 10	0x001E—0x03FF	24 I/O
114	NY6C265A	640K x 10	640K x 10	0x001E—0x03FF	24 I/O
115	NY6C305A	736K x 10	736K x 10	0x001E—0x03FF	24 I/O
116	NY6C345A	832K x 10	832K x 10	0x001E—0x03FF	24 I/O
117	NY7A004A	16K x 12	16K x 12	0x0010 – 0x03FF	8 I/O
118	NY7A007A	24K x 12	24K x 12	0x0010 – 0x03FF	8 I/O
119	NY7A010A	32K x 12	32K x 12	0x0010 – 0x03FF	8 I/O
120	NY7A016A	48K x 12	48K x 12	0x0010 – 0x03FF	8 I/O
121	NY7A021A	64K x 12	64K x 12	0x0010 – 0x03FF	8 I/O
122	NY7A032A	96K x 12	96K x 12	0x0010 – 0x03FF	8 I/O
123	NY7A043A	128K x 12	128K x 12	0x0010 – 0x03FF	8 I/O
124	NY7A054A	160K x 12	160K x 12	0x0010 – 0x03FF	8 I/O
125	NY7A065A	192K x 12	192K x 12	0x0010 – 0x03FF	8 I/O
126	NY7B007A	24K x 12	24K x 12	0x0010 – 0x03FF	16 I/O
127	NY7B010A	32K x 12	32K x 12	0x0010 – 0x03FF	16 I/O
128	NY7B016A	48K x 12	48K x 12	0x0010 – 0x03FF	16 I/O
129	NY7B021A	64K x 12	64K x 12	0x0010 – 0x03FF	16 I/O
130	NY7B032A	96K x 12	96K x 12	0x0010 – 0x03FF	16 I/O
131	NY7B043A	128K x 12	128K x 12	0x0010 – 0x03FF	16 I/O
132	NY7B054A	160K x 12	160K x 12	0x0010 – 0x03FF	16 I/O
133	NY7B065A	192K x 12	192K x 12	0x0010 – 0x03FF	16 I/O
134	NY7B076A	224K x 12	224K x 12	0x0010 – 0x03FF	16 I/O
135	NY7B087A	256K x 12	256K x 12	0x0010 – 0x03FF	16 I/O
136	NY7C010A	32K x 12	32K x 12	0x0010 – 0x03FF	24 I/O
137	NY7C016A	48K x 12	48K x 12	0x0010 – 0x03FF	24 I/O
138	NY7C021A	64K x 12	64K x 12	0x0010 – 0x03FF	24 I/O
139	NY7C032A	96K x 12	96K x 12	0x0010 – 0x03FF	24 I/O

No.	IC type	PROG ROM size	DATA ROM size	Reserved Memory	I/O Pin Count
140	NY7C043A	128K x 12	128K x 12	0x0010 – 0x03FF	24 I/O
141	NY7C054A	160K x 12	160K x 12	0x0010 – 0x03FF	24 I/O
142	NY7C065A	192K x 12	192K x 12	0x0010 – 0x03FF	24 I/O
143	NY7C076A	224K x 12	224K x 12	0x0010 – 0x03FF	24 I/O
144	NY7C087A	256K x 12	256K x 12	0x0010 – 0x03FF	24 I/O
145	NY7C110A	328K x 12	328K x 12	0x0010 – 0x03FF	24 I/O
146	NY7C130A	384K x 12	384K x 12	0x0010 – 0x03FF	24 I/O
147	NY7C150A	448K x 12	448K x 12	0x0010 – 0x03FF	24 I/O
148	NY7C170A	512K x 12	512K x 12	0x0010 – 0x03FF	24 I/O
149	NY7C220A	656K x 12	656K x 12	0x0010 – 0x03FF	24 I/O
150	NY7C260A	768K x 12	768K x 12	0x0010 – 0x03FF	24 I/O
151	NY7C305A	896K x 12	896K x 12	0x0010 – 0x03FF	24 I/O
152	NY7C345A	1024K x 12	1024K x 12	0x0010 – 0x03FF	24 I/O
153	NY8A050D	512 x 14	512 x 14	-	6 I/O
154	NY8A050E	512 x 14	512 x 14	-	6 I/O
155	NY8A051B	1K x 14	1K x 14	-	6 I/O
156	NY8A051D	1K x 14	1K x 14	-	6 I/O
157	NY8A051F	1K x 14	1K x 14	-	6 I/O
158	NY8A051G	1K x 14	1K x 14	-	6 I/O
159	NY8A051H	1K x 14	1K x 14	-	6 I/O
160	NY8A051H1	1K x 14	1K x 14	-	6 I/O
161	NY8A051J	1K x 14	1K x 14	-	6 I/O
162	NY8A051K	1K x 14	1K x 14	-	6 I/O
163	NY8A051L	1K x 14	1K x 14	-	6 I/O
164	NY8A052E	1.5K x 14	1.5K x 14	-	14 I/O
165	NY8A053B	1K x 14	1K x 14	-	12 I/O
166	NY8A053D	1K x 14	1K x 14	-	12 I/O
167	NY8A053E	1K x 14	1K x 14	-	12 I/O
168	NY8A054A	2K x 14	2K x 14	-	14 I/O
169	NY8A054D	2K x 14	2K x 14	-	14 I/O
170	NY8A054E	2K x 14	2K x 14	-	14 I/O
171	NY8A054E1	2K x 14	2K x 14	-	14 I/O
172	NY8A056A	1K x 14	1K x 14	-	16 I/O
173	NY8AE51F	1K x 14	1K x 14	-	6 I/O
174	NY8B060D	1K x 14	1K x 14	-	6 I/O

No.	IC type	PROG ROM size	DATA ROM size	Reserved Memory	I/O Pin Count
175	NY8B061E	1.25K x 14	1.25K x 14	-	14 I/O
176	NY8B062A	2K x 14	2K x 14	-	14 I/O
177	NY8B062B	2K x 14	2K x 14	-	14 I/O
178	NY8B062D	2K x 14	2K x 14	-	14 I/O
179	NY8B062E	2K x 14	2K x 14	-	14 I/O
180	NY8B062F	2K x 14	2K x 14	-	14 I/O
181	NY8B062F1	2K x 14	2K x 14	-	14 I/O
182	NY8B072A	2K x 14	2K x 14	-	18 I/O
183	NY8BE62D	2K x 14	2K x 14	-	14 I/O
184	NY8BM61D	2K x 14	2K x 14	-	14 I/O
185	NY8BM62D	2K x 14	2K x 14	-	14 I/O
186	NY8BM72A	2K x 14	2K x 14	-	18 I/O
187	NY8BM84A	4K x 16	4K x 16	-	22 I/O
188	NY8F2481	4K x 16	4K x 16	-	22 I/O
189	NY8TE64A	4K x 14	4K x 14	-	18 I/O
190	NY8TM52D	2K x 14	2K x 14	-	6 I/O
191	NY8LP05A	5K x 8	5K x 8	0x0000~0x07FF	16 I/O
192	NY8LP08A	8K x 8	8K x 8	0x0000~0x07FF	24 I/O
193	NY8LP10A	17K x 8	17K x 8	0x0000~0x07FF	16 I/O
194	NY8LP11A	17K x 8	17K x 8	0x0000~0x07FF	16 I/O
195	NY9T001A	4K x 10	4K x 10	0x001F – 0x01FF	4 I/O
196	NY9T004A	8K x 10	8K x 10	0x001F – 0x01FF	8 I/O
197	NY9T008A	12K x 10	12K x 10	0x001F – 0x01FF	16 I/O
198	NY9T016A	16K x 10	16K x 10	0x001F – 0x01FF	24 I/O
199	NY9UP01A	768 x 10	768 x 10	0x0040 – 0x004F	13 I/O
200	NY9UP02A	1280 x 10	1280 x 10	0x0020 – 0x004F	13 I/O
201	NY9UP08A	8K x 10	8K x 10	0x0020 – 0x004F	13 I/O
202	NY9U032B	32K x 10	32K x 10	0x001F – 0x03FF	16 I/O
203	NY9U064B	64K x 10	64K x 10	0x001F – 0x03FF	16 I/O

Appendix B - Glossary

To provide a common frame of reference, this glossary defines the terms that are used in this document. This glossary contains definitions for the terms used in the *NYASM*.

B.1 Terms

Nyquest MCU

Nyquest MCU refers to the NY4/NY5/NY7 micro-controller family.

Application

A set of software and hardware developed by the user, usually designed to be a product controlled by a Nyquest micro-controller.

Assemble

The act of executing the *NYASM* macro assembler to translate source code to machine code.

Binary File

An *NYASM* single executable output.

Build

A function that recompiles all the source files for an application.

Control Directives

Control directives permit sections of conditionally assembled code.

Data Directives

Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, that is, by meaningful names.

Data RAM

General purpose file registers from RAM on the MCU device being emulated.

Directives

Directives provide control of the assembler's operation by telling *NYASM* how to treat mnemonics, define data, and format the listing file. Directives make coding easier and provide custom output according to specific needs.

Expressions

Expressions are used in the operand field of the source line and may contain constants, symbols, or any combination of constants and symbols separated by arithmetic operators.

Forward reference

Forward reference means to apply variable or function before defining data. *NYASM* doesn't allow the forward reference command, e.g., the undefined Macro can not be applied, the undefined constant can not be applied.

Identifier

A function or variable name.

Initialized Data

Data which is defined with an initial value. In C, int myVar=5; defines a variable which will reside in an initialized data section.

Listing Directives

Listing Directives are those directives that control the *NYASM* listing file format. They allow the specification of base-numbering system, reserved memory access and other listing control.

Listing File

A listing file is an ASCII text file that shows the machine code generated for each assembly instruction, *NYASM* directive, or macro encountered in a source file.

Local Label

A local label is one that is defined with the LOCAL directive. These labels are particular to a given instance of the macro's instantiation. In other words, the symbols and labels that are declared as local are purged from the symbol table when the ENDM macro is encountered.

Macro

A macro is a collection of assembler instructions that are included in the assembly code when the macro name is encountered in the source code. Macros must be defined before they are used; forward references to macros are not allowed. All statements following the MACRO directive are part of the macro definition. Labels used within the macro must be local to the macro so the macro can be called repetitively.

Macro Directives

These directives control the execution and data allocation within macro body definitions.

Mnemonics

These are instructions that are translated directly into machine code. These are used to perform arithmetic and logical operations on data residing in program or data memory of a micro-controller. They also have the ability to move data in and out of registers and memory as well as change the flow of program execution. Also referred to as **Opcodes**.

NYASM

Nyquest Technology Corporation Limited's MCU assembler.

Nesting Depth

Macros can be nested to 16 levels deep (default). Maximum depth is 256.

Operators

Operators are arithmetic symbols, like the plus sign '+' and the minus sign '-', that are used when forming well-defined expressions. Each operator has an assigned precedence.

PC

Any IBM PC compatible Personal Computer.

PC Host

The computer running Windows XP/7/8.

Precedence

Precedence is the concept that some elements of an expression get evaluated before others. Operators of the same precedence are evaluated from left to right.

Program Memory

Memory in the emulator or simulator containing the downloaded target application firmware.

Project

A set of source files and instructions to build the binary code for an application.

Radix

Radix is the base-numbering system that the assembler uses when evaluating expressions. The default radix is decimal (base 10). You can change the default radix and override the default radix with certain radix override operators.

RAM

Random Access Memory (Data Memory).

Raw Data

The binary representation of code or data.

Recursion

This is the concept that a macro, having been defined, can call itself. Great care should be taken when writing recursive macros; it is easy to get caught in an infinite loop where there will be no exit from the recursion.

ROM

Read-only Memory.

Source Code

Source code consists of Nyquest MCU instructions and *NYASM* directives and macros that will be translated into machine code. This code is suitable for use by an Nyquest development system product like NYIDE.

Source File

The ASCII text file of Nyquest MCU instructions and *NYASM* directives and macros (source code) that will be translated into machine code. It is an ASCII file that can be created using any ASCII text editor.

Stack

An area in data memory where function arguments, return values, local variables, and return addresses are stored.

Symbol

A symbol is a general purpose mechanism for describing the various pieces which comprise a program. These pieces include function names, variable names, file names, macro names, etc.

Un-initialized Data

Data which is defined without an initial value. In C, int myVar.