SONiX 8-Bit MCU

ASSEMBLER

USER'S MANUAL

General Release Specification

SONiX 8-Bit Micro-Controller Development Tools

AMENDENT HISTORY

Version	Date	Description
VER 1.9	Sep. 2002	V1.9 first issue
VER 1.93	Feb. 2003	1. Modify the description of all TEXT related macro 2. Modify the description of "OUTPUT FILE" section. P31 3. Modify some examples.

SONiX 8-Bit MCU Assembly

1 SONIX ASSEMBLER

SOFTWARE INSTALLATION

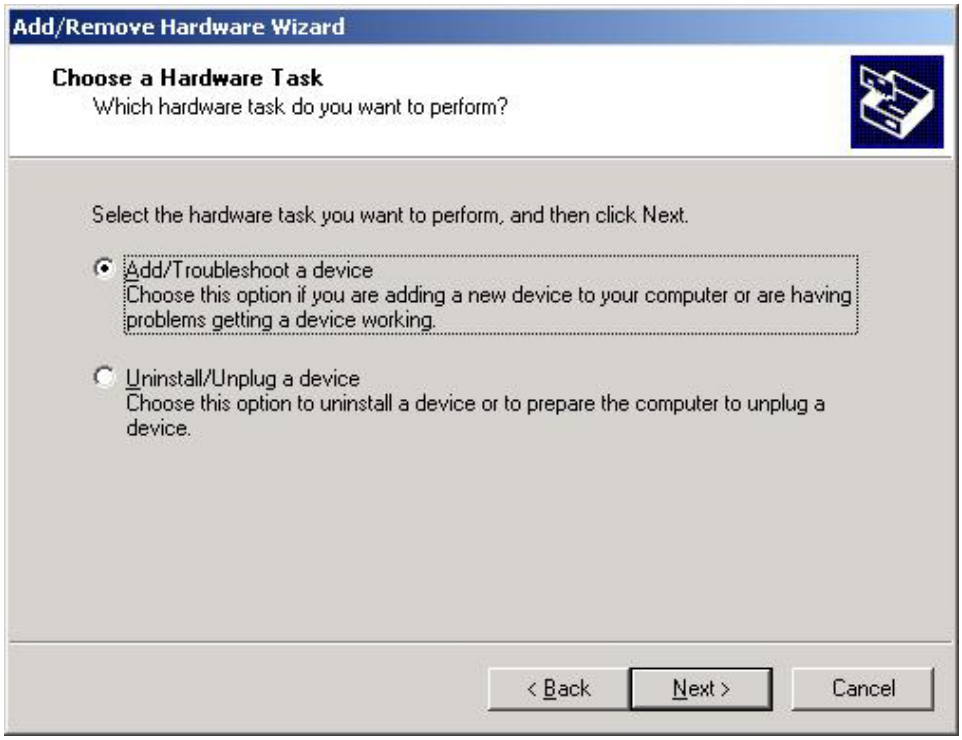
This chapter provides a start of Sonix Assembly Language from the assembler installation.

- Unzip SN8ASMxxx.ZIP to a dictionary. If the platform is Windows98/Me, just run the S8ASMxxx.exe.
- If the platform is Windows NT/2000/XP, driver needs to install before running Sonix ICE System. Below is the installation example.

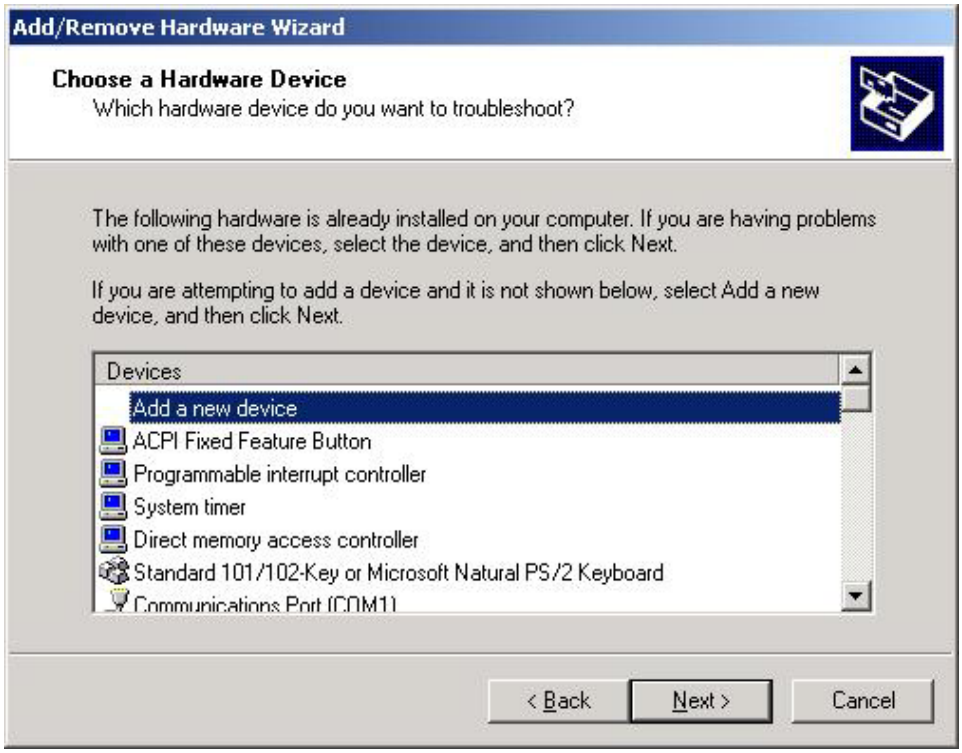
1. From the "START", choose [Setting][Control Panel], then the window below will show.

2. Select [Add/Remove Hardware]

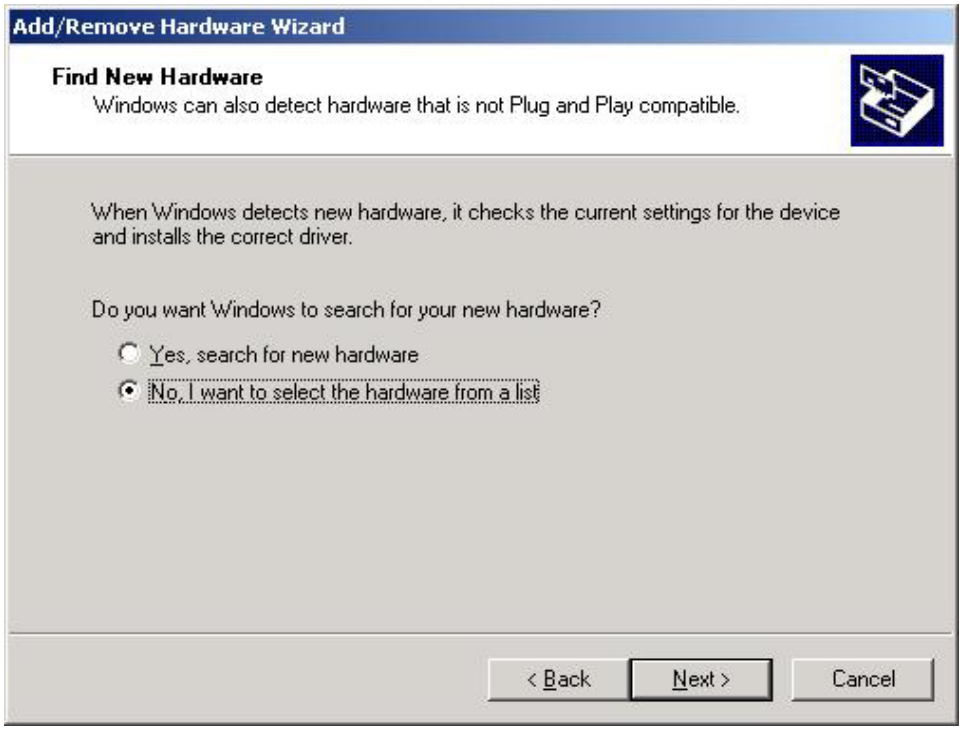
3. Click on the [Next >]



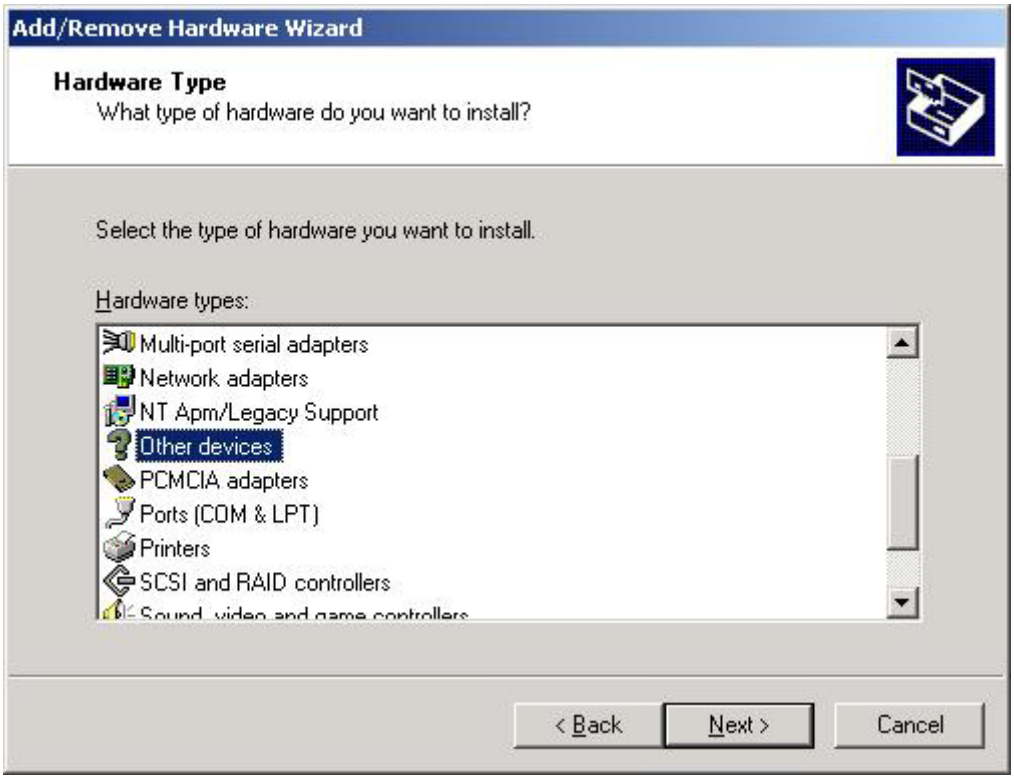
4. Select "Add a new device", and click [Next >]



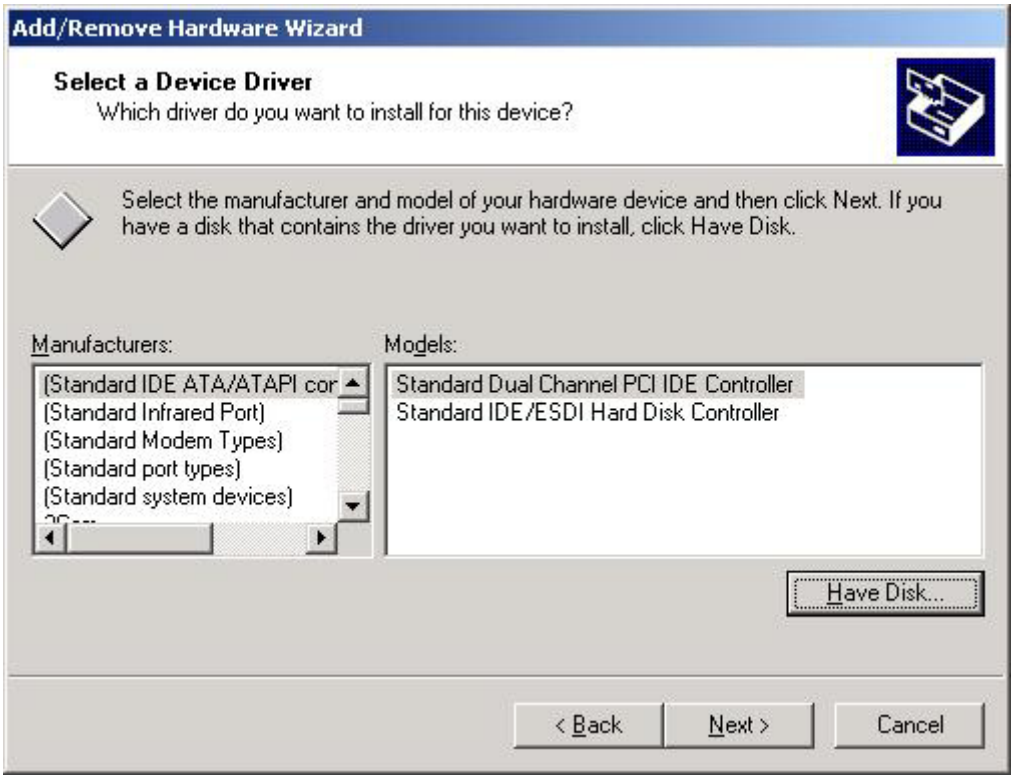
5. Select "No, I want to select the hardware from a list", and click [Next >]



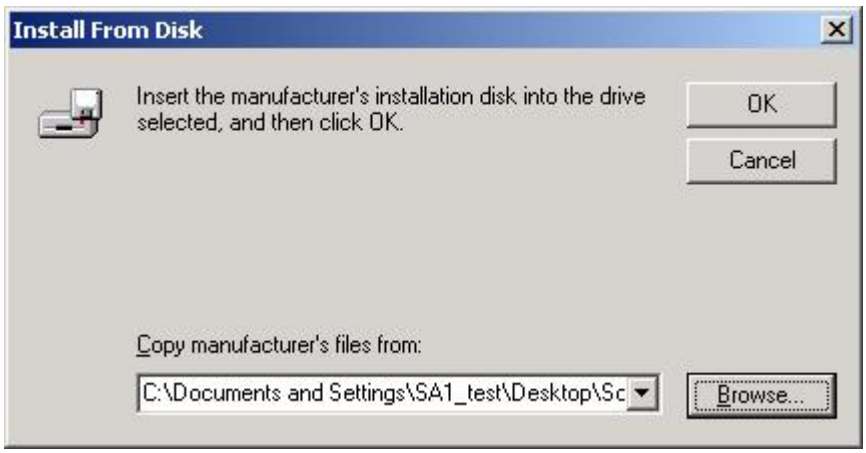
6. Select "Other devices", and click [Next >]



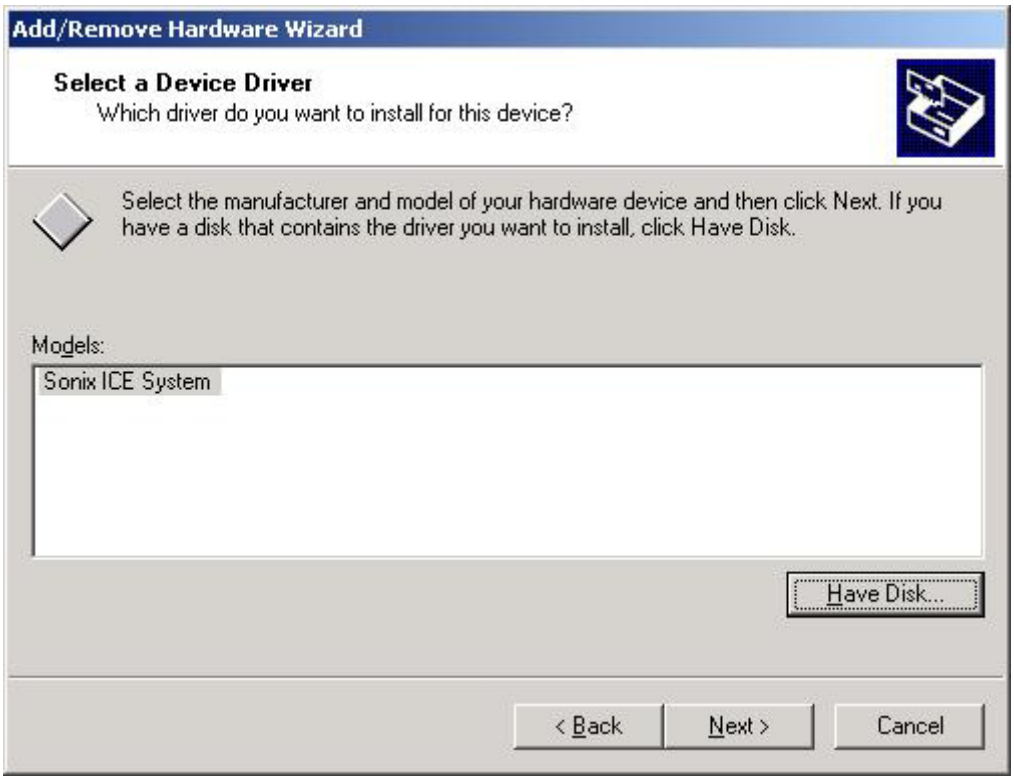
7. From the "Select a Device Driver" window, click the [Have Disk] button.



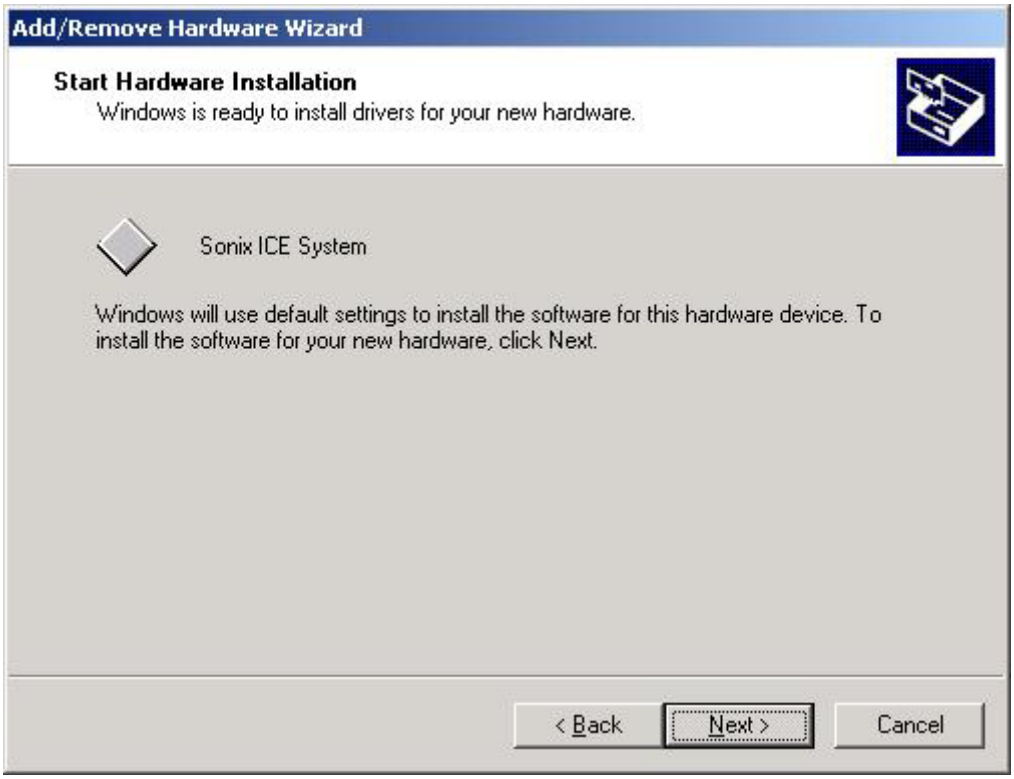
8. Find the directory that you unzip "S8ASMxxx.ZIP" by the [Browse] button. Then click [OK] button.



9. "Sonix ICE System" shows in the Models. Click [Next >].

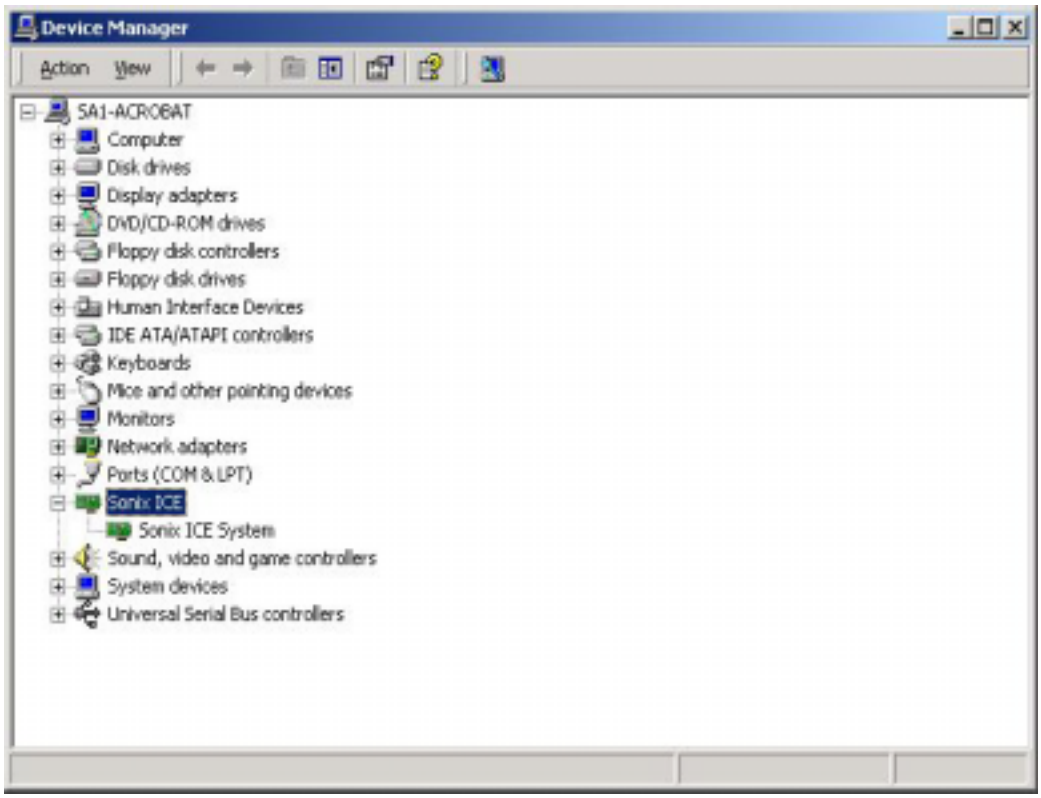


10. Click [Next >] and system will start installing the driver.



11. Click [Finish] to complete the driver installation.

12. After finishing the installation, the "System Properties/Device Manager" will add the "Sonix ICE" device to the system.



2 GENERAL FILE STRUCTURE

```
TITLE           ;User define the title. Usually omit this title.

CHIP            SN8xxxx                 ;To define chips body, Example:
                                        sn81602、sn8p1708.

INCLUDESTD      MACRO1.H                ;The system provides macro library.
                …
                instrucation or macro or include  ;Start the program.
                …

. DATA          …                       ;Ram definition.
                Accbuf DS 1             ; variable Accbuf occupies one byte of
                                        RAM
                Version EQU 0x01        ; define a constant
. CODE          …                       ;Return to programming zone.

ORG             programming location    ;Reposition the location.

DW              0, 1, …                 ;Data definition in program.
INCLUDEBIN      data in file
                …

                ENDP                    ;Can omit this command.
```

The content of all files consists of instructions and virtual instructions. The instructions can be written either by capital or letter.

INSTRUCTION

In general, the instructions in assembly language are made of four columns that can be separated by pressing "space" or "tab". The four columns are shown as follows.

LABEL INSTRUCTION OPERANDS COMMENTS

Example:

```
START:  MOV     A,#0X35     ;A = 0X35
        ADD     A,#36H      ;A = 0X6B
        DAA                 ;A = 0X71
        JMP     START
```

There are some restrictions and rules to write the commends for label, instruction, operands, and comments explained in following sections.

LABEL:

The first character of label must start with a~z, a~z, @, _ .

The rest of characters except for last character of label can be as a~z, a~z, @, _, 0~9.

The last character of label must be written as symbol‘ : ’ .

There is no limitation on how many characters for label should be but name of label can not be repeated.

In order to prevent from using so many different label names, the following instructions can be used to indicate the different label names.
First, use symbol of ‘@@ :’ as a tentative label name, and take use of this tentative label to indicate it’s previous label name and next label name. Second, use‘ @B’ to point out the previous label name located at right before the tentative label name of ‘@@ :’. The last, use ‘@F’ to point out the next label name located right after the tentative label name of ‘@@ :’.

Example:

```
        JMP     @F          ; jump to the next @@
@@:
        …       …
        JMP     @B          ; jump to the previous@@
```

INSTRUCTION:

Please refer to instruction table of datasheet to check out the currently available instructions.

OPERAND:

If there are two Operands, then should be separated by symbol of ' , ' .

Example:

```
        MOV     A, #43h
or
        MOV     A, #'C'   // block the Operand by ' '
```

If there is bit-operand, then use ' . ' to separate.

Example:

```
        B0BSET  0X86.2    // To set bit2 of 0x86 as 1.
```

or

```
        B0BSET  FC        // To default the constant value by system.
```

If operand is a memory unit, numbers can be sued to represent its address. If the number is a constant value, then the number should start with #.

Example:

```
        B0MOV   0x80, #3      // RAM[0x80] = #3
```

Besides, the symbol of $ represents currently active complier program address

Example:

```
        JMP     $             // To represent unlimited loop
        JMP     $+1           // Equivalent to two NOPs
```

More, the symbol of $ can be used to obtain the byte in high (h), middle (m), low (l)of label.

Example

```
        B0MOV   X, #DATA1$H       // X = 0X12
        B0MOV   Y, #DATA1$M       // Y = 0X34
        B0MOV   Z, #DARA1$L       // Z = 0X56
        MOVC                      // ACC = 0X90 , R = 0X78
        …
```

```
    ORG         0X123456

                Data1       DW      7890H
```

Last, the symbol of $ can also be used to define bit 14 – 17 of label as high nibble.

Example:

```
                B0MOV       PFLAG, #Far_Lab$J     ; == B0MOV  PFLAG, #30h
                JMP         Far_lab
                …
    ORG             0XC000

Far_Lab :
                …
```

COMMENTS:

In general, ‘ ; ’ and ‘//’ both represent the remarked statement. A serial wording statement written after the symbol ‘ ; ’ or ‘//’ until end of the line is the remarked statement.

Example:

```
; This is an example demo code.
// This is an example demo code.
```

Besides, the symbol of /* … */ can also be used to block a remarked statement which can be written in one or two lines.

Example:

```
/*this is an example demo code*/
```

SYSTEM OPERAND:

The system defaulted some of operands that allow users to easily use them. However, some of operands was not defaulted by the system and some of operands was depending on your selected chip. To know the defaulted operands that are existing or not, user can define the default constant value @??_existas to be 1 or 0 in the program to check if operands was provided by the system. Besides, user can also tell the operands availability by checking @p?_bits to know the number of pin.

CHIP RESERVED WORD:

Each Sonix 8-bit series MCUs has her own system registers and already been defined in the assembler as the Chip reserved word. For example, the H/L register could directly write in the program without declaration in program. And all bit-map memory space is the same. When the bit memory is used in program, a "F" prefix should be add in the name of the memory. For example, if the "GIE" bit is to be set, "b0bset FGIE" is the correct code. Please refer to datasheet for detailed system register's name and description of each chip.

NUMBER EXPRESSION:

The first digital of the number must be 0~9.

Example:

```
255          ; Decimal expression.
0xFF         ; Hexadecimal expression.
0FFh         ; Hexadecimal expression
11111111b.   ; Binary Expression
```

ARITHMETIC OPERATION:

User can use + - * / % & | ^ ~ ( ) …etc for arithmetic operations.

Example:

2 + 3 – 4 → 1

2 + 3 * 4 → 14

The followings are the arithmetic meanings of the symbol in order of priority:

<1>

( )	=	sub-expression
+	=	plus
-	=	minus
~	=	not
!	=	logical not

<2>

*	=	muliplication
/	=	division
%	=	modulo

<3>

+	=	addition
-	=	subtraction

<4>

<<	=	shl; logical shift left
>>	=	shr; logical shift right

<5>

>	=	greater than
<	=	less than
>=	=	greater than or equal to
<=	=	less than or equal to

<6>

==	=	is equal
!=	=	is not equal

<7>

&	=	and

<8>

^	=	xor

<9>

|	=	or

<10>

&&	=	logical and

<11>

||	=	logical or

ASSISTANT INSTRUCTIONS AND PSEUDO INSTRUCTIONS:

PSEUDO INSTRUCATIONS	*DESCRIPTION*
CHIP, ENDP	PROGRAM START AND THE END
EQU, =, TEXTEQU CATSTR, SUBSTR, SIZESTR, INSTR	VARIABLE EXPRESSION
.CODE, .DATA, .CONST	SECTION DEFINITION
ORG, .ALIGN	SETTING THE ADDRESS OF PROGRAM AND DATA
DS	DATA DEFINITION
DW	ROM-DATA DEFINITION WORD-SIZED
INCLUDE, INCLUDEBIN, INCLUDESTD	INCLUDE ANOTHER FILE
TITLE	USER DEFINE THE TITLE
MACRO, EXPAND, ENDM, REPEAT, FOR, FORC, EXITM	MACRO
.LIST, .NOLIST, .LISTIF, .NOLISTIF, .LISTMACRO, .NOLISTMACRO, .LISTMACROALL	CONTROL THE PROGRAMMING FILES
IF, IFE, IFB, IFNB, IFDEF, IFNDEF, IFIDN, IFDIF, IFIDNI, IFDIFI, ELSEIF, ELSEIFE, ELSEIFB, ELSEIFNB, ELSEIFDEF, ELSEIFNDEF, ELSE, ENDIF	CONTROL CONDITIONAL COMPILING
.ERR, .ERRE, .ERRNZ, .ERRB, .ERRNB, .ERRDEF, .ERRNDEF ERROR, ECHO	FUNCTION OF CONDITIONAL COMPILING OPERANDS
.EXEC	EXECUTE EXTERNAL PROGRAM
@BIT, @INT, @FIELD	FUNCTION OF BIT ARITHEMETIC

PROGRAM START AND END:

Syntax: **CHIP** **SN8XXXX**

Description:
To select IC. User can know which IC is currently available from menu [option]->[chip info]. The command should be defined prior to any assembly language and could only be defined once.

Example:

```
Chip      sn80211
```

Syntax: **ENDP**

Description：
Force to end the program and the program written right after this command would then not be compiled/assembled.

Example:

```
Endp
```

USER DEFINE THE TITLE:

Syntax: **TITLE** descritpion statment

Description:
The statement written after the title is the description statement

Example:

```
TITLE     This is a demo code.
```

VARIABLE EXPRESSION:

EQU

Syntax: VARIABLE EQU VALUE or BIT

Description:
Fixed variables can not modified.

Example:

```
TRUE    EQU   1
FALSE   EQU   0
PIN1    EQU   P0.0
NUM1    EQU   0X20+0X3
```

=

Syntax: VARIABLE = VALUE | BIT

Description:
Changeable variables can be modified.

Example:

```
TEMP    =     0
TEMP    =     TEMP + 1        // TEMP = 1
TPIN    =     TEMP.7
```

```
ENUM    MACRO   LAB:VARARG

        TEMP    =   0
        FOR     L, <LAB>
                L       EQU     TEMP
        TEMP    =       TEMP + 1
        ENDM
        ENDM
```

```
ENUM    CON0, CON1, CON2
        // CON0 EQU 0; CON1 EQU 1; CON2 EQU 2
```

TEXT MACRO TEXTEQU:

Syntax:

```
STRING    TEXTEQU   <STRING>
STRING    TEXTEQU   TEXT MACRO
STRING    TEXTEQU   % VARIABLE
STRING    TEXTEQU   % (ARITHMETIC)
```

Description:

Text macro would be changed to use for the replacement of STRING. The variable or arithmetic after % would turn to be serial words. Catstr, substr, sizestr, instr can be used to express above four different serial words.

Example 1: **<STRING>**

```
CLRA      TEXTEQU  <MOV A,#0>
```

Example 2: **TEXT MACRO**

```
CLRALU    TEXTEQU  CLRA
```

Example 3: **% VARIABLE**

```
TEMP      =        30
STR1      TEXTEQU  %TEMP        ;   TEXTEQU <0x1E>
STR1      TEXTEQU  %0d:TEMP     ;   TEXTEQU <30>
STR1      TEXTEQU  %0x:TEMP     ;   TEXTEQU <1e>
STR1      TEXTEQU  %0X:TEMP     ;   TEXTEQU <1E>
STR1      TEXTEQU  %03d:TEMP    ;   TEXTEQU <030>
STR1      TEXTEQU  %03x:TEMP    ;   TEXTEQU <01e>
STR1      TEXTEQU  %03X:TEMP    ;   TEXTEQU <01E>
```

Example 4: **% (ARITHMETIC)**

```
TEMP      =        20
STR1      TEXTEQU  %(TEMP+6)    ;   TEXTEQU <0x1A>
```

CATSTR:

Syntax:	**STRING**	**CATSTR**	**<STRING1>, <STRING2>**
	STRING	**CATSTR**	**TEXT MACRO 1, TEXT MACRO 2**
	STRING	**CATSTR**	**%VARIABLE1, %VARIABLE2**
	STRING	**CATSTR**	**%(ARITHMETIC 1), %(ARITHMETIC 2)**

Description:

Joint two serial words/text macro to create new text macro. The format of string, please refer to TEXTEQU.

Example:

```
S1      TEXTEQU   <12>
S2      CATSTR    <0x>, S1      // S2 equivalent to "0x12"
```

SUBSTR:

Syntax:	**STRING**	**SUBSTR**	**<STRING>, START, [LENGTH]**
	STRING	**SUBSTR**	**TEXT MACRO, START, [LENGTH]**
	STRING	**SUBSTR**	**% VARIABLE, START, [LENGTH]**
	STRING	**SUBSTR**	**% (ARITHMETIC), START, [LENGTH]**

Description:

Retrieve one of serial words from string. Start is the starting position of that retrieved serial words and the first character of serial words is in position 1. Length represents the length of characters. If omit the length, the serial words would be retrieved up to the end of that serial word. The format of string please refer to textequ.

Example:

```
S1      TEXTEQU   <123456>
S2      SUBSTR    S1, 4, 2      // s2 equivalent to  "45"
S3      SUBSTR    S1, 3         // s3 equivalent to  "3456"
```

SIZESTR :

Syntax:	**VALUE**	**SIZESTR**	**<STRING>**
	VALUE	**SIZESTR**	**TEXT MACRO**
	VALUE	**SIZESTR**	**% VARIABLE**
	VALUE	**SIZESTR**	**% (ARITHMETIC)**

Description:

User can tell the string's length form string. The format of strin, please refer to TEXTEQU.

Example:

```
V1          SIZESTR     <123456>      // v1 Equivalent to  6
```

INSTR:

Syntax:	**VALUE**	**INSTR**	**START, <STRING>, <SUBSTRING>**
	VALUE	**INSTR**	**START, TEXT MACRO, <SUBSTRING>**
	VALUE	**INSTR**	**START, % VARIABLE, <SUBSTRING>**
	VALUE	**INSTR**	**START, % (ARITHMETIC), <SUBSTRING>**

Description:

Find out substring from string beginning with start position in which the first character is in position 1. If substring cannot be found, value is to be 0.string. The format, please refer to TEXTEQU.

Example:

```
S1          TEXTEQU     <12,34,56>
V1          INSTR       1, S1, <,>      // v1 equivalent to 3
V2          INSTR       V1+1, S1, <,> // v2 equivalent to 6
```

SECTION DEFINITION

Syntax: **.CODE**
.DATA
.CONST

Description:
Set current address as programming section(.code), data section (.data) or constant section. (.const). The size of programming section depends on the space of rom size. The size of data section depends on the space of ram size , there is no size issue in constant section. Each section can be inter-replaceable. The system defaults the section as programming section (.code) and its starting address is 0.

Example:

```
.CODE
        MOV       A, #0
        ...

.DATA
        RAM0  DS  1          // equivalent to  ram0   equ 0
        RAM1  DS  1          // equivalent to  ram1   equ 1

.CODE
        ...

.DATA
        BUF0  DS  2            // equivalent to  buf0   equ 2
        BUF1  DS  1            // equivalent to  buf1   equ 4

.CODE
        ...
```

SET UP THE ADDRESS OF PROGRAM & DATA & CONSTANT SECTIONS:

ORG:

Syntax: **ORG** **NEW ADDRESS**

Description:
User can re-set the programming address that is generally used to interrupt the position of programming address. If there is no specific programming address be set up at the beginning of program, the system defaults the programming address is 0.

Example:

```
        MOV         A, #0FH
        B0MOV       STKP, A             // disable interrupt, set stack to bottom
        B0MOV       PFLAG, #0           // at the first 16k rom
        JMP         START

ORG     8                               // t0, t0c, t1c, p0 … interruption's starting address.
        CLR         INTRQ                   // clear all interrupt request
        RETI

START:
        …
        ORG_TMP     =     $

        ORG         ($+15) & 0X3FF0     // re-position 16 times alignment
        DW          …

        ORG         ORG_TMP + 0X100     // re-position 0x100 apart…
```

.ALIGN:

Syntax: **.ALIGN NUMBER**

Description:

Use **.**align to adjust the value of alignment for next instruction or variable. Number must be 2 multiplier such as 2、16、256，the maximum number is 65536 (0x10000).

Example:

```
// IF $ == 7
. ALIGN        16
//SAME AS ABOVE          ORG          ($+15) & 0X3FF0
// THEN $ == 16
```

DATA DEFINITION

DEFINITION OF BYTE DATA:

Syntax: **[LABEL] DB D1 [, D2 , ...]**
[LABEL] DB "STRING" [, "STRING", ...]

Description:
Define data in the program. The date must locate between 0 ~ 0xff or be a serial words blocked by the symbol of " ". Every two data become a word. Two bytes made up a word. The fist byte is low byte and the second byte is high byte. If the data can not be a word, high byte is set as 0.

Example:

```
DB      12, 34, 30, "ABCD"

equivalent to

DW      0X220C, 0X411E, 0X4342, 0X0044
```

DEFINITION OF WORD DATA:

Syntax: **[LABEL] DW D1 [, D2 , ...]**
[LABEL] DW "STRING" [, "STRING", ...]

Description:
Define data in the program. The date must locate between 0 ~ 0xffff or be a serial words blocked by the symbol of " ". Every two data become a word. Two bytes made up a word. The fist byte is low byte and the second byte is high byte. If the data can not be a word, high byte is set as 0.

Example:

```
TEMP    =   45
DW      0X1234, 5678H, TEMP+3, 2*5
DW      "ABCDEFG", 23H
HERE    DW      "HERE", "SONIX"
...
B0MOV   X, #HERE$H
B0MOV   Y, #HERE$M
B0MOV   Z, #HERE$L
MOVC    // ACC = 'H', R = 'E'
```

DEFINITION OF PROGRAMMING DATA:

Syntax: **[LABEL] DD D1 [, D2 , ...]**
[LABEL] DD "STRING" [, "STRING", ...]

Description:
Define data in the program. The date must locate between 0 ~ 0xffffffff or be a serial words blocked by the symbol of " ". Four bytes made up a dword that is often used to save label since data of label is usually over 64k.

Example:

```
TABLE DD    L1, L2, L3
    …
    L1      DW      "HELLO"
    L2      DW      "GOOD"
    L3      DW      "SONIX"
```

REGISTER DATA DEFINITION:

Syntax: **[LABEL] DS SIZE**

Description:
DS is a general term to define the ram. Size is a figure value to represent the pace in ram. The size would be a arithmetic.

Example:

```
. DATA
        MEM1        DS      1
        BUFFER1     DS      4
        XBUF        DS      0           // XBUF  EQUIVALENT TO BUFFER2
        BUFFER2     DS      8
        …
. CODE
        …
        B0MOV       H, #BUFFER1$M
        B0MOV       L, #BUFFER1$L
        MOV         A, DP0X                     // ACC = [BUFFER]
        …
```

BIT ARITHMETIC FUNCTION :

Syntax: **@BIT(parameter), @INT(parameter), @FIELD(parameter)**

Description:

Regarding bit operand, retrieve bit by ：@bit( ) , retrieve integral number by ：@int( ) , or retrieve bit column by : @fiels( ) ,

Example:

```
P_XOR   EQU     P1.4
…
B0BSET  @INT(P_XOR)-0X10.@BIT(P_XOR)  // b0bset p1m.4s set up out mode

…
MOV         A, #@FIELD(P_XOR)mailto:#@FIELD(P_XOR          // MOV   A, #10H
; (BIT4)
XOR         @INT(P_XOR), A              // XOR   P1, A     ; TOGGLE PIN
```

OUTPUT FILE

.LST	Assemble and make a listing file (.LST).
.SN8	Sonix format binary file for Sonix OTP Writer and MASK production.
.HEX	For OTP Programming in using third party writer. (ie. Hilosystem)
Other Output	Reserved.

3 INCLUDE FILES

INCLUDE:

Syntax: **INCLUDE FILE**

Description:
This instruction is followed by a programming file. If this programming file is not under the directory of currently active compiling program, user needs to indicate the path to reach the programming file. The syntax of the programming file is the same as common files whose extended file is .h. In general, the programming file would include constant value and macro and its extended file is .asm. The included file can also include another file and this another file can also include another file and so forth. The maximum lays of including is up to 255 layers.

Example:

```
INCLUDE   SN88X.H          //   include the name of self-defined variable
INCLUDE   C：\PROJECT.H    //   include self-defined macro
INCLUDE   sub\Filename.ASM
INCLUDE   ..\Parent\File2.ASM
```

INCLUDESTD:

Syntax: **INCLUDESTD FILE**

Description:
This instruction is followed by a programming file. The path of arriving this programming file is fixed under the path of sn8asm.exe. Please refer to 5.1 include section for other descriptions. **Includestd instruction is used for user easily to obtain macro function provided by the system.**

Example:

```
INCLUDESTD  MACRO1.H     //include frequently used macro in system
INCLUDESTD  MACRO2.H     //include frequently used macro in system.
```

INCLUDEBIN:

Syntax: **INCLUDEBIN FILE**

Description:

This instruction is followed by a bin file， if this bin file is not under the directory of currently active compiling program, user needs to indicate the path to reach the bin file. The date of bin file use word as a unit. If a unit of data is insufficient to make up a word, the unit is still counted a word

Example:

```
INCLUDE    SPEECH.BIN      //
INCLUDE    C：\SOUND.SND   //
```

4 MACRO

MACRO Example List

For SONIX Assembly Developer, we support below default macro for easy to design your program. After including macro1.h, macro2.h, and macro3.h all of following macros are available. Please refer marco1.h, macro2.h and macro3.h for more macro examples.

Class	Assembler mnemonic	Expansion form	NO. of word	Function	Flags			Cycle
					CF	DC	ZF	
COMMAND	CLC	B0BCLR FC	1	Clear C Flag	0	–	–	1
	STC	B0BSET FC	1	Set C Flag	1	–	–	1
	RSTWDT	B0BSET FWDRST	1	Reset Watchdog count	–	–	–	1
	EINT	B0BSET fgie	1	Global interrupt enable	–	–	–	1
	DINT	B0BCLR fgie	1	Global interrupt disable	–	–	–	1
	NOT A	XOR A, #0FFh	1	A ← ~ A	–	–	ˇ	1
	NEG A	XOR A, #0FFh ADD A, #1	2	A ← – A	ˇ	ˇ	ˇ	2
ROTATE/SHIFT	SHL memory	B0BCLR FC RLCM memory	2	memory ← memory * 2	ˇ	–	–	2
	SHR memory	B0BCLR FC RRCM memory	2	memory ← memory / 2	ˇ	–	–	2
	B2B bit1, bit2	B0BCLR bit2 B0BTS0 bit1 B0BSET bit2	3	Translate bit2 status to bit1 status	–	–	–	3
	ROL mem	RLCM mem B2B FC, mem.0	4	FC ← mem.7 mem← mem * 2 + FC	ˇ	–	–	4
	ROR mem	RRCM mem B2B FC, mem.7	4	FC ← mem.0 Mem← mem / 2 + FC * 80h	ˇ	–	–	4
	RCR mem	RRCM mem	1	Alias name	ˇ	–	–	1
	RCL mem	RLCM mem	1	Alias name	ˇ	–	–	1

Class	Assembler mnemonic	Expansion form	NO. of word	Function	Flags			Cycle
					CF	DC	ZF	
B R A N C H	JZ address	B0BTS0 FZ JMP address	2	If ZF == 1 then jump to address	–	–	–	2
	JNZ address	B0BTS1 FZ JMP address	2	If ZF == 0 then jump to address	–	–	–	2
	JC address	B0BTS0 FC JMP address	2	If CF == 1 then jump to address	–	–	–	2
	JNC address	B0BTS1 FC JMP address	2	If CF == 0 then jump to address	–	–	–	2
	JDC address	B0BTS0 FDC JMP address	2	If DCF == 1 then jump to address	–	–	–	2
	JNDC address	B0BTS1 FDC JMP address	2	If DCF == 0 then jump to address	–	–	–	2
	JB1 bit, addr	BTS0 bit JMP addr	2	If bit == 1 then jump to addr	–	–	–	2
	JB0 bit, addr	BTS1 bit JMP addr	2	If bit == 0 then jump to addr	–	–	–	2
	DJNZ mem, adr	DECMS mem JMP adr	2	mem ← mem –1 If mem != 0 then jump to adr	ˇ	–	ˇ	2-3
	IJNZ mem, adr	INCMS mem JMP adr	2	mem ← mem +1 If mem != 0 then jump to adr	ˇ	–	ˇ	2-3
	CJNE A, m, adr	CMPRS A, m JMP adr	2	If ACC != m then jump to adr	ˇ	–	ˇ	2-3
	CJE A, m, adr	CMPRS A, m JMP $+2 JMP adr	3	If ACC == m then jump to adr	ˇ	–	ˇ	3-4
	CJAE A, m, adr	CMPRS A, m B0BTS0 FC JMP adr	3	If ACC >= m (Above Equal) then jump to adr	ˇ	–	ˇ	3-4
	CJAE m, A, adr	CMPRS A, m B0BTS1 FC JMP adr	3	If m >= ACC (Above Equal) then jump to adr	ˇ	–	ˇ	3-4
	CJBE A, m, adr	CJAE m, A, adr	3	If ACC <= m (Below Equal) then jump to adr	ˇ	–	ˇ	3-4
	CJBE m, A, adr	CJAE A, m, adr	3	If m <= ACC (Below Equal) then jump to adr	ˇ	–	ˇ	3-4
	CJA A,m, adr	CMPRS A, m B0BTS1 FC JMP $+2 JMP adr	4	If ACC > m (Above) then jump to adr	ˇ	–	ˇ	4-5
	CJA m,A adr	CMPRS A, m B0BTS0 FC JMP $+2 JMP adr	4	If m > ACC (Above) then jump to adr	ˇ	–	ˇ	4-5
	CJB A,m, adr	CJA m, A, adr	4	If ACC < m (Below) then jump to adr	ˇ	–	ˇ	4-5
	CJB m,A adr	CJA A, m, adr	4	If m < ACC (Below) then jump to adr	ˇ	–	ˇ	4-5

MACRO:

Syntax: **NAME MACRO [PARA1 [, PARA2, …]]**
…
ENDM

Description:
The programming writing could be simplified by suing macro. The length of program could be shortened by suing sub-program. Taking the advantages of both macro and sub-program, the optimal program could be done. The parameters of macro is limited up to 255, the length of macro is not limited, but the macro is composed of the characters of a~z , a ~ z , 0~9 , @ #_.$. If the parameters must include special character, this special character have to be blocked by symbol of < >. Besides, macro would also include macro. There is no limitation on how many layers of macro including.
Example:

```
MOV_    MACRO       ADDRESS , VALUE
MOV         A, VALUE            //    macro content
MOV         ADDRESS, A          //
ENDM

…
MOV_        1,  #1              //    use macro
MOV_        2,  <#6 * 8 + 1>
```

EXPAND:

Syntax: same as above

Description:
When error occurs during compiling or simulation, the cursor would indicate the position of "use macro" by using macro. The cursor would indicate the position of "macro content" by using expand. This function helps to detect the errors.

REPEAT:

Syntax: **REPEAT COUNT**
... **; REPEATED COMMAND**
ENDM

Description:
Use repeat to execute a section of commands repeatedly. The times of repeating is defined by value of count. Use endm to end the section of commands. Use repeat to start the section of commands.

Example:

```
RRCM_1      MACRO       MEMORY, VALUE
REPEAT      VALUE
RLCM        MEMORY
ENDM
ENDM
...
RRCM_       0, 4        ; to make the data in ram[0] turn right four times
```

FOR:

Syntax: **FOR PARAMETER, <PARAMETER 1, PARAMETER 2 …>**
… **; the command is to be repeated**
ENDM

Description:
Use forc to execute a section of commands repeatedly. The times of repeating is defined by value of <parameter number>. Use ' comma to separate the sequence of parameter and to replace parameter value with one parameter at a time.

Example:

```
TEMP    =    0
FOR     I,  <M0, M1, M2, M3, M4, M5>
I       EQU     TEMP
TEMP    =       TEMP + 1
ENDM

; M0/1/2/3/4/5 EQU     0/1/2/3/4/5
```

FORC:

Syntax: **FORC PARAMETER, <WORD SERIAL>**
… **; the command is to be repeated.**
ENDM

Description:
Use forc to execute a section of commands repeatedly. The times of repeating is defined by value of <lengthen of word serial>. According to the sequence of characters in word serial, replace parameter value with one character at a time.

Example:

```
FORC        I, <012345>
M&I         EQU     I
ENDM
```

EXITM:

Syntax: **EXITM**

Description:

Use exitm command to end the macro in advance. This is used as macro occurs error.

Example:

```
MEM     MACRO       value
.IF   value >= 10
      ERROR         value must < 10
      EXITM
.ENDIF
      M&value       EQU       value
ENDM
```

ARITHMETIC COMMAND UNDER MACRO:

&

Syntax: **…&PARAM**

Description:
Under macro, transfer corresponding serial words into parameters.

Example:

```
FORC    I, <012345>
M&I     EQU     I       ; M0/1/2/…  EQU     0/1/2/…
ENDM
```

%

Syntax: **%PARAM**

Description:
Parameter in macro, transfer corresponding serial words into figure values.

Example:

```
ERROR_  MACRO   E1  E2  E3
ERROR   E1  E2  E3
ENDM

TEMP    =   10

ERROR_  TEMP  IS  %TEMP
```

When compiling, the error message of temp is 0xa appears.

!

Syntax: **!**

Description:
Under macro, make the next character exclude any special meaning.

Example:

```
ENUM    MACRO   CH
FORC    I, <012345>
CH!&I   EQU     I       ; CH&0/1/… EQU   0/1/…
ENDM
ENDM

ENUM    M                       ; M0/1/2/…  EQU   0/1/2/…
```

Above example is a two-layer macro as compiling, enum m would be extended to be:

```
FORC    I, <012345>
M&I     EQU     I       ; M0/1/…    EQU   0/1/…
ENDM
```

If symbol of ! is not used, then the symbol of & would be used at the first time of extension.

```
FORC    I, <012345>
MI      EQU     I       ; M0/1/…    EQU   0/1/…
ENDM
```

At the second time of extension, the symbol becomes a variable value, mi.

INCOMPLETE LOCAL ARITHMETIC:

In general, under macro, it provides label of local. If you want to execute function of local's label, please include the standard macro, macro3.h in the program.

Example:

```
    INCLUDESTD  MACRO3.H
XXX     MACRO
    LOCAL XX1, XX2
    JMP   XX1
XX1:
    JMP       XX2
XX2:
    NOP
ENDM
```

Then xxx could be called out many times, but xx1、xx2 can not be re-defined.

REQUIREMENT, PRE-SETTING AND VARIABLE PARAMETERS:

Think of following MACRO. As lacking of one parameter, there is no any error occurred.

```
ADD2    MACRO       A, B
        DW          A+B
        ENDM

ADD2    , 4                 ; DW    +4
```

If the first parameter is not omitted, rewrite the program as follows.

```
ADD2    MACRO       A : REQ,  B
        DW          A+B
        ENDM

ADD2    , 4                 ; ERROR OCCURS
```

If the second parameter is omitted, use the pre-setting value to rewrite the program as follows.

```
ADD2    MACRO       A : REQ,  B : = 0           ; OR B : = <ARITHMETIC>
        DW          A+B
        ENDM

ADD2    4                   ; DW    4+0
```

The last, if parameter is a variable parameter, rewrite the program as follows.

```
ADDN    MACRO       parmas : VARARG
        TEMP        =     0
        FOR         I, <params>
        TEMP    =   TEMP + I
        ENDM

        DW          TEMP
        ENDM

ADDN    1, 2, 3, 4              ; DW    10
```

5 CONDITIONAL ASSEMBLING CONTROL

Syntax:

IF	**expression**	; As it isn't 0, it's true
IFE	**expression**	; As it is 0, it's true
IFB	**<argue>**	; As it is blank, it's true
IFNB	**<argue>**	; As it isn't blank, it's true
IFDEF	**symbol**	; As symbol is defined, it's true
IFNDEF	**symbol**	; As symbol is not defined, it's true
IFIDN	**<str1>, <str2>**	; As str1 == str2, it's true
IFDIF	**<str1>, <str2>**	; As str1 <> str2, it's true
IFIDNI	**<str1>, <str2>**	; As str1 == str2 (either capital or little letter), it's true
IFDIFI	**<str1>, <str2>**	; As str1 <> str2 (either capital or little letter), it's true.

ELSEIF	**expression**
ELSEIFE	**expression**
ELSEIFB	**<argue>**
ELSEIFNB	**<argue>**
ELSEIFDEF	**symbol**
ELSEIFNDEF	**symbol**
ELSEIFIDN	**<str1>, <str2>**
ELSEIFIDNI	**<str1>, <str2>**
ELSEIFDIF	**<str1>, <str2>**
ELSEIFDIFI	**<str1>, <str2>**
ELSE	
ENDIF	

Description:

Use IF command to detect specific condition and to control the compiling of program. Use ENDIF to end the IF command. Also can add selectable ELSEIF / ELSE command to support nest-like structure.

Example:
In order to avoid the blank on parameters, the following syntax can be used:

```
TEMP    =    0
FOR     I, < ZERO, ONE,  ,THREE>
IFNB    <I>
I    EQU     TEMP
ENDIF
TEMP    =    TEMP + 1
ENDM
```

In order to avoid .H file be reloaded, the following syntax can be sued:

```
IFNDEF      __TESTFILE__          ; Add at the beginning of program
__TESTFILE__    EQU    1                ;
…                                 ; program
ENDIF                                 ; end program
```

Also can use following nest-like structure:

```
IF              VAR == 1
…
ELSEIF          VAR      <= 10
…
IF          VAR > 5
…
ELSE
…
ENDIF
…
ELSEIF          VAR <= 100
…
ELSE
…
ENDIF
```

ERRORS DISPLAY UNDER CONDITIONAL ASSEMBLY:

Syntax:

```
.ERR                        ; Enforce to yield error
.ERRNZ      expression      ; As it is not zero, yield error
.ERRE       expression      ; As it is zero, yield error
.ERRB       <argue>         ; As it is blank, yield error
.ERRNB      <argue>         ; As it is not blank, yield error
.ERRDEF     symbol          ; As symbol is defined, yield error
.ERRNDEF    symbol          ; As symbol is not defined, yield error
.ERRIDN     <str1>, <str2>  ; As str1 == str2, yield error
.ERRDIF     <str1>, <str2>  ; As str1 <> str2, yield error
.ERRIDNI    <str1>, <str2>  ; As str1 == str2 (regardless of capital or small letter),
                              yield error,
.ERRDIFI    <str1>, <str2>  ; As str1 <> str2 (regardless of capital or small letter),
                              yield error,
ERROR       <string>        ; Enforce to yield error and use string as an error message
ECHO        <string>        ; Yield STRING as a message
```

Description:

Above assembly instructions are able to detect the specific conditions and yield appropriate error messages that remind users to watch his program logic.

Example: ECHO Here be compiled // While assembling at this line message appears

The following example of use on MACRO parameters can be changed to original format:

```
        ADD2    MACRO       A：REQ,  B：= <0>
                DW          A+B
                ENDM
```

New format:

```
        ADD2    MACRO       A, B
                .ERRB       <A>
                IFB         <B>
                DW      A
                ELSE
                DW      A+B
                ENDIF
                ENDM
```

Apparently, this is not a good format that only control the example-like program list files.
While user select MENU [Output] -> [List] and no error occur during assembly, the system would generate a list file, LST.

. LIST:

Syntax: **.LIST**

Description:
Use **.**list to list source instructions written in the program after .list instruction in the list. This is a defaulted setting value.

Example:

```
.LIST
…                    ; will appear in the list.
.NOLIST
```

.NOLIST:

Syntax: **.NOLIST**

Description:
Use **.**nolist to close the programming list. The system will not open the programming list until another assembly instruction .list opens the programming list.

Example:

```
.NOLIST
…                    ; will not appear in the list
.LIST
```

.LISTIF:

Syntax: **.LISTIF**

Description:
Use **.**listif list conditional instructions in a section. Even though the instructions have not been assembled in that section and the instructions should be listed in the list.

Example:

```
.LISTIF
IF    0
…                    ; will appear in the list.
ENDIF
```

.NOLISTIF:

Syntax: **.NOLISTIF**

Description:
Use **.**nolistif to hide those conditional instructions which have failed in assembly and have not been assembled in the list. This is a defaulted setting value.

Example：

```
.NOLISTIF
IF    0
…                    ; will not appear in the list.
ENDIF
```

.LISTMACRO:

Syntax: **.LISTMACRO**

Description:
Use **.**listmacro to print list those instructions which are able to generate programming code or data in the macro section. This is a defaulted setting value.

Example:

```
.LISTMACRO
TEMP        =        0

REPEAT      8
DW      TEMP                ; will appear in the list
DEMP    =   TEMP + 1        ; will not appear in the list
ENDM
```

.NOLISTMACRO:

Syntax: **.NOLISTMACRO**

Description:

Use **.**listmacro to list those instructions which can not be extended in the macro sections.

Example:

```
.NOLISTMACRO
TEMP    =    0

REPEAT  8

DW      TEMP              ; will only appear machine code in the list.
DEMP    =    TEMP + 1     ; will not appear in the list.
ENDM
```

.LISTMACROALL:

Syntax **.LISTMACROALL**

Description:

Use **.**listmacroall to list all instructions in the macro section.

Example:

```
.LISTMACROALL
TEMP    =    0

REPEAT  8

DW      TEMP              ; WILL APPEAR IN THE LIST.
DEMP    =    TEMP + 1     ; WILL APPEAR IN THE LIST.
ENDM
```

Main Office:
Address: 9F, NO. 8, Hsien Cheng 5th St, Chupei City, Hsinchu, Taiwan R.O.C.
Tel: 886-3-551 0520
Fax: 886-3-551 0523

Taipei Office:
Address: 15F-2, NO. 171, Song Ted Road, Taipei, Taiwan R.O.C.
Tel: 886-2-2759 1980
Fax: 886-2-2759 8180

Hong Kong Office:
Address: Flat 3 9/F Energy Plaza 92 Granville Road, Tsimshatsui East Kowloon.
Tel: 852-2723 8086
Fax: 852-2723 9179

Technical Support by Email:
Sn8fae@sonix.com.tw