SH66xx 匯編語言用戶手冊 V1.0

SH66xx Assembly Language User's Manual V1.0

目錄

一. 簡介

SH66xx 匯編器爲交叉匯編器，目標機(target machine)爲基于 6610 核的系列通用 MCU，宿主機(host machine)爲兼容的 IBM PC 機，匯編器工作在 MS-DOS 及 Windows 環境， 接受符號形式的 SH66xx 匯編語言作爲輸入，其輸出爲 SH66xx 的機器指令文件(obj 文件,或其它格式的指令文件如 Intel HEX 格式)及一些其他相關輸出文件.

二. 匯編器的使用

匯編器的使用支持兩種方式的調用，分別爲基于 MS-DOS 的命令行方式和基于窗口的執行方式

2.1. 匯編器在 DOS 命令行下以如下方式調用：

ASM [options] filename[.asm] [Enter]

其中 filename 爲要進行匯編的源文件，源文件爲由任意文本編輯器生成的 66xx 的匯編文件.對[option]命令行選項的說明具體如下：

[option]	說明
/F [BIN16|INHX16|INHX8M|INHX8S]	指定輸出編碼文件格式
/P [66P10|66P10C|66P20|66P20A|66P22|6622|66P22A]	指定目標處理器的型號(未全)
/R [HEX|OCT|DEC]	指定程序默認的基數
/W [1|0]	顯示或隱藏警告消息
/I	忽略程序中符號的大小寫
/DE	顯示詳細的錯誤報告
/L	不輸出 list 文件
/V	顯示版本號及版權
/H	顯示幫助信息

三. 匯編程序的結構

匯編的源程序由一或多條語句(statements)或注釋(comments)組成.

每條語句分行書寫並由指令助記符(mnemonics)，匯編指示符(directives)，宏(macros)，符號(symbols)，表達式(expressions)，常數(constants)組成.

3.1 匯編語句行的格式

一般，源程序中的一條語句行以如下形式書寫：

[< 標號 >] < 命令 > [< 操作數 >] [< 注釋 >] [CR]

1. 標號 (label)

標號爲一條語句行的第一部分，爲可選，其書寫規則爲以字母開始後面由字母,數字,下劃線組成的字符串，通常標號以冒號結束，但若標號頂行書寫，標號可不以冒號結束，其後面第一個空白符(空格 space，制表符 tab，回車符 end-of-line)爲其結束符. (標號的長度不作限制，在保證每行的長度在 200 字符以內即可)

標號一般爲程序中參考到的地址，標號的值爲該位置處的程序地址. 標號可以單獨占一行.

2. 命令 (command)

命令部分與其它部分以空白符分隔，命令部分可由指令助記符，匯編指示符和宏組成. 對於 SH66xx 而言 ，指令助記符 43 條， 匯編指示符 13 種.

3. 操作數 (Operand)

操作數部分緊跟在命令部分之後，與命令部分以空白符分隔. 操作數部分依命令部分的不同可能含有零個，一個或多個部分，它們之間以逗號(“,”)分隔.

4. 注釋 (Comment)

注釋部分在一語句行的最後，以分號”;”開頭到行末都認爲是注釋，同時也可以支持 C++風格的行注釋(以”//”開頭)，注釋部分是可選的，並可單獨占一行.

5. [CR]

彙編語句以回車符結束。

3.2 常量 (Constants)

常量爲字符,字符串或數字，匯編器將其解釋爲一固定的數值. 匯編器支持不同基數(radix)的數，如十六進制，十進制，八進制和二進制. 十進制是匯編器默認的基.

1．字符或字符串常量

字符串常量以單引號或雙引號開始，並以一匹配的引號結束，匯編器將會將引號之間的字符轉化爲其對應的 ASCII 的值作爲常量值. 如 ‘A’ , “THIS IS AN EXAMPLE”

2. 數字常量

數字常量由數字和字母組成，它們代表的值由其選擇的基數決定，匯編其中支持 4 種基的指定，分別爲十六進制，十進制，八進制，二進制. 基的指定通過附加在數字之後的後綴字符表示.

如下：

後綴字符	基數	組成字符	例子
B	二進制	0 和 1	1010B = $10_{(10)}$
O	八進制	0 ~ 7	037O = $31_{(10)}$
D	十進制	0 ~ 9	279D = $279_{(10)}$
H	十六進制	0 ~ 9 a~f及A~F	7ffH = $2047_{(10)}$

如果沒有指定基，則匯編器默認的基爲是十進制，

此外對於十六進制數，基的指定也可以通過數字前加表示基的前綴"0x"或"$"表示，如 0x7FF = $7FF = 7FFH. 十六進制的 A~F 使用大小寫字符都可以.

對於十進制數，可以通過數字前加前綴"."指定， 即一個十進制數可以表示爲 "123" (默認爲十進制)，或 "123D" (指定爲十進制數) 或 ".123" (指定爲十進制數).

數字前都可以加單目運算符"+"或"-"表示正或負值

3.3 符號 (Symbol)

符號在程序中表示一個值，它可以是符號常量，可以是變量，可以是地址標號. 符號可以作爲命令的操作數出現.

1. 符號名

符號名爲用戶定義的字母,數字,特殊符號組合在一起，在程序中作爲某種標識出現. 符號名的定義規則和程序標號類似，以字母開頭，後面跟字母，數字，下劃線或特殊字符"@"，匯編器的默認值是區分符號的大小寫的，通過調用匯編器時的 /I 參數忽略符號的大小寫

2. 符號類型

符號類型提供符號的屬性信息，它由符號的定義方式決定. 符號類型分三種，分別爲 DATA 型, VAR 型, ADDR 型. 具體如下：

DATA 型：用戶使用 EQU 匯編指示定義的符號， 符號的值爲定值

VAR 型：用戶使用 SET 匯編指示定義的符號，符號的值可變，可以在後面的程序中用 SET 更改

ADDR 型：用戶定義的標號地址或程序地址

DATA 和 VAR 型的符號在定義時所參考到的符號必須是定義點之前定義過的符號，符號在程序中可以作爲操作數或標號，但參考的符號必須在程序的其它地方有定義，否則匯編器會報錯.

匯編器中一個特殊的符號爲 '$'，它表示該處的程序地址，它可以應用到操作數部分或表達式中.

3.4 表達式 (Expression)

表達式可以使用在源語句的操作數中，操作數由符號，常量或由運算符(Operation)連接而成的符號,常量的組合組成.

1. 運算符

表達式中使用的運算符分爲算述運算符，邏輯運算符，關係運算符，位運算符等四類. 具體爲

運算符	描述	例子	
+	加/正	1+2 +8	算述運算符
-	減/負	3-2 -9	

*	乘	4 * 8	
/	除	9 / 5	
%	求余	13 % 6	
**	乘方	3 ** 2	
<<	左移	5 << 2	
>>	右移	3 >> 7	
==	相等	x == y	關係運算符
!=	不等	x != y	
<=	小於等於	3 <= 5	
>=	大於等於	5 >= 3	
>	大於	7 > 4	
<	小於	6 < 9	
&&	邏輯與	x && y	邏輯運算符
||	邏輯或	x || y	
!	邏輯非	! x	
&	位與	x & y	位運算符
|	位或	x | y	
^	異或	x ^ y	
~	位取反	~x	
( )	括號		

“ + ” 和 “ – “ 可作爲單目運算符的正負，也可作爲雙目運算符的加減.

2. 優先級(priority)和結合性(associativity)

運算符的優先級和結合性遵從 C 語言對其的約定. 除單目運算符 “+”, “-“, “!”, “~”的結合性爲右結合外，其餘都是左結合，優先級和結合性的關係如下所示：

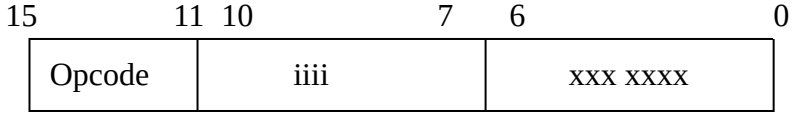
運算符	結合性	優先級由高至低
( )	由左至右	最高
! ~ + - (單目運算符)	由右至左	
**	由左至右	
* / %	由左至右	
+ - (雙目運算符)	由左至右	
<< >>	由左至右	

> >= < <=	由左至右	
== !=	由左至右	
&	由左至右	
^	由左至右	
|	由左至右	
&&	由左至右	
||	由左至右	最低

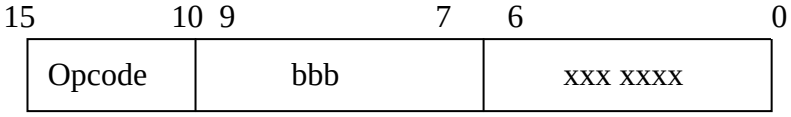
四. SH66xx 指令助記符(mnemonics)

SH66xx 匯編器中的指令助記符對應 SH66xx 系列芯片對應的指令，共 43 條， 伴隨適當的操作數出現在語句行的命令部分. 按指令編碼的不同可以分爲三類

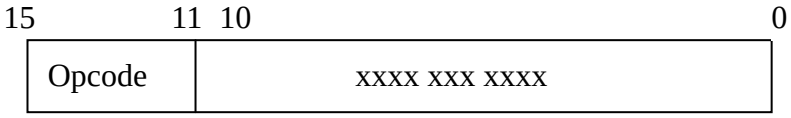
立即數類指令



存儲器類指令



跳轉分支類指令



此外還有一些特殊的指令和固定編碼的指令,

詳細的指令編碼表爲：

			15…11	10 … 7	6…0
立即數類指令	LDI	Mx, I	01111	iiii	x
	ORIM	Mx, I	01101	iiii	x
	EORIM	Mx, I	01100	iiii	x
	ANDIM	Mx, I	01110	iiii	x
	SBI	Mx, I	01010	iiii	x
	SBIM	Mx, I	01011	iiii	x
	ADI	Mx, I	01000	iiii	x
	ADIM	Mx, I	01001	iiii	x
跳轉分支類指令	JMP	x	1110p	x	
	BAZ	x	10010	x	
	BNZ	x	10000	x	
	BA0	x	10100	x	
	BA1	x	10101	x	

	BA2	x	10110	x	
	BA3	x	10111	x	
	BC	x	10011	x	
	BNC	x	10001	x	
	TJMP		11110	1111 111 1111	
	CALL	x	11000	x	
	RTNW	H, L	11010	000h hhh llll	
	RTNI		11010	1000 000 0000	
特殊指令	HALT		11011	0000 000 0000	
	STOP		11011	1000 000 0000	
	NOP		11111	1111 111 1111	
	SHR		11110	0000 000 0000	
			15…10	9 … 7	6…0
存儲器類指令	LDA	Mx, bbb	001110	bbb	x
	STA	Mx, bbb	001111	bbb	x
	ADD	Mx, bbb	000010	bbb	x
	ADDM	Mx, bbb	000011	bbb	x
	ADC	Mx, bbb	000000	bbb	x
	ADCM	Mx, bbb	000001	bbb	x
	SUB	Mx, bbb	000110	bbb	x
	SUBM	Mx, bbb	000111	bbb	x
	SBC	Mx, bbb	000100	bbb	x
	SBCM	Mx, bbb	000101	bbb	x
	DAA	Mx	110010	110	x
	DAS	Mx	110011	010	x
	AND	Mx, bbb	001100	bbb	x
	ANDM	Mx, bbb	001101	bbb	x
	OR	Mx, bbb	001010	bbb	x
	ORM	Mx, bbb	001011	bbb	x
	EOR	Mx, bbb	001000	bbb	x
	EORM	Mx, bbb	001001	bbb	x

五. SH66xx 匯編指示 (Assembler Directives)

匯編指示同指令助記符類似，可以伴隨一些相應的操作數出現在語句行的命令部分，但匯編指示不會生成指令碼，它只是幫助匯編器進行程序控制，數據分配，或格式化輸出等. SH66xx 匯編指示共 13 種，列表如下：

匯編指示	作用
DW/DATA	在 memory 中初始化數據
END	標明程序結束
EQU	定義 DATA 類型的符號並付給一個值
IF..ELSE..ENDIF	條件匯編

INCLUDE	包含外部匯編文件
LIST	提供 LIST 選項, 如 memory size , processor type
ORG	設定程序初始地址
PAGE	在 LIST 文件中插入換頁符
RES	設定一段受保護的 Memory
SET	定義或設定一個變量類型的符號
SPAC	在 LIST 文件中插入空行
TITLE	指定 LIST 文件頭部信息
ROMSIZE	定義最大的程序空間

1. DW/DATA --- 在 memory 中初始化數據

語法格式： [< label >] DW < operand >
[< label >] DATA < operand >
描述：對程序存儲空間進行數據的初始化, 操作數部分爲一組常量
例子：DW 10h, 20h, 30h
DATA "This is a test"

2. END --- 標明程序結束

語法格式： [< label >] END [< comment >]
描述：標明程序結束
例子： END ; terminate the program

3. EQU --- 定義 DATA 類型的符號並付給一個值

語法格式：< symbol > EQU < expression > [< comment >]
描述：定義 DATA 類型的符號常量, 定義 symbol 的值就是表達式的值, 表達式只能參考定義過的符號.
例子： SYNC EQU 19H

4. IF..ELSE..ENDIF 條件匯編

語法格式：
```
IF <expression>
    <source lines>
ELSE
    <source lines>
ENDIF
```
描述：根據表達式的真假進行有條件的匯編, IF 與 ENDIF 必須匹配
例子：
```
IF  1
    JMP 1FFh
```

```
        ELSE
            JMP 0FFh
        ENDIF
```

6. INCLUDE --- 包含外部匯編文件

語法格式： [<label>] INCLUDE "<filename>" [<comment>]
描述： 包含其它的匯編文件，文件名可以包含完整路徑
例子： INCLUDE "REG.H"

6. LIST --- 提供 LIST 選項，如 memory size , processor type

語法格式： LIST P = < processor type >
(66P20|66P22)
LIST R = < radix >
(DEC*|HEX|OCT)
LIST F = < output format >
(BIN16*|INHX16|INHX8M|INHX8S)
* 默認值
描述： 用來指示匯編器改變默認值
例子： LIST P=66P22, R=HEX, F=INHX16

7. ORG --- 設定程序初始地址

語法格式：[<label>] ORG <expression> [<comment>]
描述：設定程序初始地址
例子： ORG 0h
ORG 100h

8. PAGE --- 在 LIST 文件中插入換頁符

語法格式：[<label>] PAGE [<comment>]
描述：插入換頁符
例子： PAGE

9. RES --- 設定一段受保護的 Memory

語法格式：[<label>] RES <expression> [<comment>]
描述：程序計數器以按表達式的值增加
例子：RES 10

10. SET --- 定義或設定一個變量類型的符號

語法格式：[< symbol >] SET <expression> [<comment>]
描述：定義或設定一個變量類型的符號，並可以在後續的程序中通過 SET 改變符號的值
例子：FIVE SET 5

11. SPAC --- 在 LIST 文件中插入空行

語法格式：[<label>] SPAC <expression> [<comment>]
描述：在 LIST 文件中插入表達式值數量的空行
例子：SPAC 3

12. TITLE --- 指定 LIST 文件頭部信息

語法格式：[<label>]　TITLE "<string>"　[<comment>]
描述：　在 LIST 文件第一行插入指定的字符串
例子：　TITLE　"TEST.ASM"

13. ROMSIZE --- 定義最大的程序空間

語法格式：ROMSIZE = expression [<comment>]
描述：設定最大的程序空間
例子：ROMSIZE = 0x400

六. 宏的使用(MACRO)

宏的使用包括宏的定義和宏的調用, 宏的定義由一系列指令或匯編指示組成, 在宏調用的地方將會插入指令或匯編指示. 宏在調用點之前必須有定義過.

6.1 宏的定義 (macro definition)

宏的定義可以分爲三個部分：宏頭(macro heading), 宏体(macro body), 宏結束(macro terminator)

1. 宏頭(macro head)

宏的頭部格式爲：

< symbol >　MACRO　[<parameter>, ... <parameter>]　[<comment>]

其中 symbol 爲宏的名字, parameter 來自宏調用的參數

宏名字的規則遵從之前對符號的規則約定, 如果宏的名字和其它符號相同, 將會報錯.

2. 宏体(macro body)

宏体緊跟在宏頭定義之後, 宏体可以由一系列指令和匯編指示組成, 其中可能包含對參數的操作, 宏在調用的時候將宏体的部分完全的替換到調用的地方.

3. 宏結束(macro end)

宏以"ENDM"指示結束, 格式爲：

[<label>] ENDM [<comment>]

在 ENDM 和宏名字定義 MACRO 之間不能有另外的 MACRO, 也就是說宏不可以嵌套.

6.2 宏的調用(macro call)

在宏定義了之後，就可以進行宏的調用. 格式爲：

[<label>] <name> [<arg> [,<arg>] ...] [<comment>]

其中 label 爲程序地址, name 爲定義的宏的名字 arg 列表爲傳遞給宏的參數，參數列表以最後的空白符或分號結束.

宏調用本身不佔用程序空間，但替換了宏体之後，宏的實例從但前程序地址開始，逗號可以用來保留參數的位置，這種情況下的該參數位空，參數列表以空白符或者分號結束

6.3 參數

參數以字符形式傳遞到宏的實例中，因而符號是通過名字傳遞的而非符號的值，也就是說直到宏展開之後參數的值才被計算出來，因而在宏中使用 SET 匯編指示可以改變傳遞到 macro 中的參數的值

6.4 局部符號

局部符號爲定義在宏中的符號，這些符號只在宏中有意義，實際上匯編器處理局部符號的方法是在每次宏調用的時候將其更名爲內部名字，形式爲??0001, ??0002 等. 內部名字的定義格式爲:

LOCAL <label> [,<label>]

局部符號的定義必須緊跟在宏頭之後第一行.

七. 局部標號

局部標號以字符'?'開始。用戶可以像一般的標號一樣使用它們，唯一不同的是局部標號存在一定的生存週期，而前面介紹的一般的標號都是全局的。局部標號的生存週期限制在兩個全局標號之間，在這個區間以外，這些局部標號是不可見或無效的，用戶不可以使用它們就好像它們從沒有被定義過一樣。例如：

```
START:
        SBI FLG0,05H
        BC ?1
        LDI FLG0,05H
?1:
        NOP
        SBI FLG1,05H
        BC ?2
        LDI FLG1,05H
?2:
        ADIM FLG2,01H
        NOP
        NOP
        JMP ?1
END:
        ……
```

局部標號"?1"和'?2"在兩個全局標號"START"和"END"之間是有效的，用戶可以在他們的生存區間內像一般標號一樣使用它們，但是在"START"之前和"END"之後，局部標號變失去了意義。如果用戶仍然在外部使用它們，則會發生符號未定義的錯誤信息，這就是因爲在外部它們就好像從沒定義過一樣。

八. 匯編器的錯誤檢查

匯編器在匯編過程中會進行錯誤檢查, 並指明錯誤類型, 報告錯誤消息(Error message), 指明出現錯誤語句. 匯編器可以提供的錯誤檢查及提供的一些主要錯誤或警告消息有：

1. Bad instruction statement
2. Symbol not defined
3. Symbol Redefinition
4. Miss operand(s)
5. Too many operands
6. IF..ELSE..ENDIF directive error
7. LIST directive error
8. Value is out of range
9. Unknown character
10. DataList directive error
11. Unknown output parameter
12. Unknown processor type
13. Argument error
14. Immediate data truncated to 4 bits
15. Bank operand truncated to 3 bits
16. Memory operand truncated to 7 bits
17. Target operand truncated to 11 bits
18. Unknown Assembly Directive Usage
19. Unmatched IF..ELSE..ENDIF
20. missing "," between operands
21. missing target operand
22. missing immediate operand

23. missing ram bank operand
24. redundant operands
25. PC exceed Romsize
26. DATA type symbol redefinition
27. Bad TITLE operands
28. macro name redefinition
29. symbol is not MACRO
30. Processor type redefinition
31. Default radix redefinition
32. Output format redefinition
33. Bad expression
34. too many arguments
35. Crossing ROM Bank boundary
36. Bad numeric string format
37. Unmatched quotation
38. Bad character
39. Unknown character

九. 附錄

A. 編碼文件格式：

SH66xx Assembler 匯編器可以輸出的格式分爲二進制格式, 16 位 INTEL 格式, 8 位 INTEL 格式, 和 8 位混合的 INTEL 格式, 他們都是普遍採用的編碼文件格式. 具體說明如下：

a. Binary file Format

A pure 16-bit binary file, which is generated by a compiler
or cross assembler. The data is only readable by the processor.
A screen dump of SAMPLE.OBJ will be as follows:

```
18FD:0100   01 02 43 07 00 08 64 0C-21 00 A5 02 04 00 00 08    ..C...d.!.....
18FD:0110   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
18FD:0120   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

```
18FD:0130   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0140   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0150   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0160   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0170   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0180   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0190   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:01A0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:01B0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:01C0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:01D0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:01E0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:01F0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0200   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0210   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0220   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0230   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0240   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0250   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0260   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0270   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0280   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0290   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:02A0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:02B0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:02C0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:02D0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:02E0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:02F0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
18FD:0300   00 0C 05 00 25 00 26 00-06 00 03 0C 02 00 64 0C    ....%.&.......d.
18FD:0310   21 00 A6 02 0A 0C 30 00-00 09 F0 02 0C 0B 09 0B    !.....0.........
18FD:0320   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

b. INHX16: Intel Hex file formats

This is the most commonly used format for file interchange with EPROM programmers. A complete Intel Hex file contains one or more hexadecimal records. The file ends with an end of file record.

Each data record begins with a nine character prefix and ends with

a two character checksum. Each letter corresponds to one hexadecimal digit in ASCII representation.

Example
:BBAAAATTHHHH...HHHCC

Definitions
: Record start character
BB Byte count. The hexadecimal number of data bytes in the record.
AAAA Load address in hexadecimal of first data byte in this record.
TT Record type. The record type is 00 for data records and 01 for the end record.
HH One hexadecimal data byte.
CC Record checksum. This is the 2's complement of the summation of all the bytes in the record from the byte count through the last byte. While the summation is calculated, it is always truncated to a one byte result.

c. INHX8S: Split 8-bit Intellec Hex file format

This format will be output if the INHX8S option is used with the LIST F directive or with the '/f' option on the command line.

This format produces two 8-bit Hex files, one containing the address/data pairs for the high order 8 bits and the other will contain the low-order 8 bits. File extensions for the object code will be '.HXL' and 'HXH' for low and high order files respectively.

Example:

SAMPLE.HXL:	SAMPLE.HXH:
:0103FE0000FE	:0103FE000BF3
:080000000143006421A5040086	:080000000207080C00020008D1
:080200000005252606030026437	:080200000C000000000C000CD2
:0802100021A60A3000F00C09E0	:0802100000020C0009020B0BB7
:00000001FF	:00000001FF

d. INHX8M: Merged 8-bit Intellex Hex Format

This format will be output if the INHX8M option is used with the

LIST F directive or with the '/f' option on the command line.

This format produces one 8-bit Hex file with a low-byte/high-byte combination. Since each address can only contain 8 bits in this format, all addresses will be doubled. File extensions for the object code will be '.OBJ'.

Example:

```
:0203FE00000BF2
:10000000010243070008640C2100A5020400000857
:10020000000C0500250026000600030C0200640C0B
:100210002100A6020A0C30000009F0020C0B090BA9
:00000001FF
```

B. LIST 文件格式 ：

List 文件主要由四列組成，分別是行號，程序地址，匯編編碼，源程序碼. 程序地址和匯編編碼都以寬度爲 4 的十六進制數組成

List 文件提供各項信息的對應，包括程序地址，匯編碼，源碼等，其中源碼包括其在相應源文件中的行號.

下面爲一個 List 文件的例子：

```
1                   1: ORG 100H
2                   2:      data1:
3     0x0100  0x0010        3: DW     10h,20h,30h
4     0x0101  0x0020
5     0x0102  0x0030
6     0x0103  0x7d20        4: LDI      LFT, 1010B
7     0x0104  0x0054        5: DATA     "This is a test"
8     0x0105  0x0068
9     0x0106  0x0069
10    0x0107  0x0073
11    0x0108  0x0020
12    0x0109  0x0069
13    0x010a  0x0073
14    0x010b  0x0020
```

```
15   0x010c   0x0061
16   0x010d   0x0020
17   0x010e   0x0074
18   0x010f   0x0065
19   0x0110   0x0073
20   0x0111   0x0074
21                    6: LFT EQU 20H
22                    7: N      EQU LFT
23
24                    9: ORG 0
25   0x0000                10: include "test3.asm"
26              +     1: ORG 900H
27   0x0900   0xe900    +     2: JMP 900H
28
29
30                    12: IF      ( N>10H )&& (N == LFT )
31   0x0901   0x7d20          13:     LDI         LFT, 1010B
32                    14: ELSE
33                    15:     LDI N,01H
34                    16: prog1: LDI LFT, 1011B
35                    17: ENDIF
36   0x0902                18: include "test4.asm"
37   0x0902   0x0c20    +     1: label:     ADDM    20H,00
38
39                    20: end
```

C. 符號文件格式

符號文件格式較簡單，它提供程序中定義的所有符號，並指明它們的類型，值，和所在的行，並提供程序中符號的一些統計信息

下面爲一個符號文件的例子：

```
Cross Reference
9 symbols

Symbol          Type      Value    Line
aa              ADDR      0001     0003
```

aalab	ADDR	0109	0090
gg	DATA	0020	0001
kk	ADDR	0000	0002
lab	ADDR	0004	0006
label	ADDR	002F	0049
label2	ADDR	0104	0063
label4	ADDR	002D	0047
prog	ADDR	0100	0056

D. MAP 文件格式

MAP 文件用於提供程序的調試信息, 其格式較爲複雜, 這裡只做簡單的介紹, 不建議用戶使用.

二進制的 MAP 文件如下組成

- a 16-integer header block
- a 2048-byte source file table, and
- a source line table of line number in the source file to program memory address of each executable source line.

Map File Header**:** Starts at offset 0 in MAP file and occupies 16 integers (32-bytes).

Int	Value	Description
0	0xAAFF	Map ID
1	0x0120	Version
2	0x1654	Device type
3	0x0200	Program memory size of the processor
4	0x01FF	Reset vector address
5	0x003E	Program counter address of main( ) function
6-15	0x0000	Reserved

Source File Table**:** Starts at offset 0x20 in MAP file. The table consists of a source file list, up to 30 files. Each file name occupies 64 bytes.

BYTE	Description
0	length of filename
1-63	filename with path

Source Line Information: Starts at offset 0x0800 in MAP file. The table consists of a list of source line number that corresponds to each program memory address. Each source line number occupies 4 integers (8 bytes).

Int	Value	Description
0	0x0000	Program counter address
1	0x003B	Source line number
2	0x0000	List line number (optional)
3	0x8000	lower byte: source file number according to source file table upper byte: flag

E. 錯誤信息文件格式

錯誤信息文件提供匯編過程中發現的錯誤信息, 每條報錯的信息格式爲：
<File> (line number):(error/warning) <message>
沒有錯誤的源文件匯編之後錯誤信息文件爲空
一個錯誤信息文件的例子如下：
lshift.asm(23): Symbol is not defined
lshift.asm(26): Symbol is not defined
lshift.asm(29): Symbol is not defined
lshift.asm(40): Symbol is not defined
lshift.asm(41): Symbol is not defined