1. 特色

符合ANSI C標準規範
IEEE相容浮點運算標準
內含97個常見函式庫
支援65C02 CPU
可直接指定暫存器及I/O變數(by SFR)
可指定程式碼或速度最佳化
支援BANK記憶體分段模式
中斷程式可直接使用C語言

2. 程式執行流程

CPU reset後，會先進入CSTARTUP.S14執行，程序如下： [C-78]

1. **CLD**
2. **Initialize Processor stack**
3. **Call __low_level_init副程式（含在lowinit.c）及視需要Initialize data segments**
4. **Call main()主程式。 (結束後仍可回到CSTARTUP.S14)**

2.1 修改CSTARTUP

1. 修改cstartup.s14 檔案
2. 執行a6502 cstartup.s14（如此將產生cstartup.r14 之object檔案）
3. 將library更新：執行xlib.exe，並輸入下列操作

```
def-cpu 6502
rep-mod CSTARTUP library名稱
exit
```

變更過後的library可以覆蓋過原先的檔案，從此將會使用新的CSTARTUP程序；
若是不想將原來的library覆蓋，可以將新產生的library放在project路徑，然後
在.xcl檔案裡面加上下列資料也可以只用指定的library:

```
library名稱
```

3. C library定義 [C-93]

IAR-C的標準library包含三個部分：

1. **標準C library定義。**
2. **CSTARTUP定義的程序。**
3. **內定供6502使用的low-level功能。**

使用方式：
首先必須先指定合適的記憶體模式與CPU的library，並在程式中include需用到的Head files。

詳細Library定義參考[C-94]~[C-170]

Library種類命名分類如下：

LIBRARY	COMMAND LINE	MEMORY	BANK支援	CPU種類
CLT	-mt	Tiny	No bank	6502/65C02
CLL	-ml	Large	No bank	6502/65C02
CLTB	-mb	Tiny	Bank	6502/65C02
CLLB	-mB	Large	Bank	6502/65C02
CLTP	-mtp	Tiny	6502	Protel 6502
CLLP	-mlp	Large	6502	Protel 6502
CLTBP	-mbp	Tiny	Bank	Protel 6502
CLLBP	-mBp	Large	Bank	Protel 6502

PS：

1. Tiny: 所有變數default放在Zero page裡面.
2. Large: 所有變數default放在Zero page外面.

4. XCL硬體定義檔分析：

-c6502
（定義CPU）

-Z(ZPAGE)ZPAGE,C_ARGZ,Z_UDATA,Z_IDATA=A0-EF
（指定 ZPAGE,C_ARGZ,Z_UDATA,Z_IDATA 使用 "ZPAGE" segments的A0H-EFH區域）
-Z(ZPAGE)EXPR_STACK+20=80
（保留給 "EXPR_STACK" 20H 個bytes，從80H開始；它是C運算需要的stack空間）
-Z(ZPAGE)INT_EXPR_STACK+10
（保留給 INT_EXPR_STACK 10H 個bytes，從"ZPAGE" segments剩下的位址起算；如果有在C程式用到 ”中斷”，則需要保留空間）
-Z(NPAGE)CSTACK+80=100
（保留 CSTACK 80H 個bytes，從100H開始；這是給65c02保留的硬體stack空間）
-Z(NPAGE)C_ARGN=180-3FF
（指定 C_ARGN 使用 "NPAGE" segments的180H-3FFH區域）
-Z(NPAGE)NPAGE,N_UDATA,N_IDATA,ECSTR=400-FFF
（指定 NPAGE,N_UDATA,N_IDATA,ECSTR 使用 "NPAGE" segments的400H-FFFH區域）
-Z(NPAGE)RF_STACK+0
（指定 RF_STACK 使用 "NPAGE" segments的空間，若設 ”0” 則內訂爲100H個bytes；它是提供給 recursive function，用來保存自動變數與參數）
-ZLCDm+0=1000
（自訂 “LCDm” segment空間，從1000H開始）
-Z(CODE)CODE,RCODE,NONBANKED,Z_CDATA,N_CDATA,C_ICALL,C_RECFN,CSTR,CCSTR,CONST=4000-7FDF
（指定CODE, RCODE, NONBANKED, Z_CDATA, N_CDATA, C_ICALL, C_RECFN, CSTR, CCSTR, CONST 使用 “non-banked ROM” segments的4000H-7FDFH區域）
-Z(CODE)INTVEC=7FE0-7FFF
（指定 "INTVEC" interrupt segment 在7FE0-7FFF區域）

-b(CODE)BANK1:BANK2:DAVID3:ANY4:TEST5=018000,8000,10000,64
（指定BANK結構：有BANK1…….TEST5等5個bank，第一個bank位址由bank號碼01的8000H開始；每個bank大小是8000H；每個bank的間隔是bank加1、位址加0000，最多有64H個banks）

-M004000-007FFF=04000-07FFF
-M018000-01FFFF=08000-0FFFF
-M028000-02FFFF=10000-17FFF
-M038000-03FFFF=18000-1FFFF
-M048000-04FFFF=20000-27FFF
（邏輯位址與實體位址對應表，左邊是邏輯位址，右邊是實體位址；在SSI之下必須mark起來不使用）

-e_small_write=_formatted_write
（指定printf/sprintf 使用精簡版的程序，功能較標準的ANSI縮水，但空間只佔ANSI 20%） [C-76]
-e_medium_read=_formatted_read
（指定scanf/sscanf 使用精簡版的程序，功能較標準的ANSI縮水，但佔空間較少） [C-77]

cllb /* -mB -v1 */
（指定使用的library爲cllb）

-D?BANK_REGISTER=32
（指定Bank switch register位址） [C-80]
-D?BANK_REGISTER_MASK=FF
（指定Bank switch register的mask；可分別指定可設定的bit）
-D?BANK_LOW=8000
（指定BANK啓始的邏輯位址）
-D?BANK_HIGH=FFE0
（指定BANK結束的邏輯位址）

Example:

```
SEGMENT              SPACE   START ADDRESS     END ADDRESS       SIZE   TYPE   ALIGN
=======              =====   =============     ===========       ====   ====   =====
EXPR_STACK                     0080 - 009F         20    rel     0
ZPAGE                          00A0 - 00A5          6    rel     0
C_ARGZ                         00A6 - 00A7          2    rel     0
Z_UDATA                           00A8              0    rel     0
Z_IDATA                           00A8              0    rel     0
INT_EXPR_STACK                 00A8 - 00B7         10    rel     0
CSTACK                         0100 - 017F         80    rel     0
C_ARGN                         0180 - 01BD         3E    rel     0
N_UDATA                           0400              0    rel     0
NPAGE                             0400              0    dse     0
N_IDATA                        0400 - 0403          4    rel     0
ECSTR                             0404              0    rel     0
RF_STACK                          0404              0    dse     0
LCDm                           1000 - 12FF        300    rel     0
CODE                           4000 - 455F        560    rel     0
RCODE                          4560 - 4CD7        778    rel     0
NONBANKED                      4CD8 - 4DFB        124    rel     0
Z_CDATA                           4DFC              0    rel     0
N_CDATA                        4DFC - 4DFF          4    rel     0
C_ICALL                           4E00              0    dse     0
C_RECFN                           4E00              0    dse     0
CCSTR                             4E00              0    rel     0
CSTR                              4E00              0    dse     0
CONST                          4E00 - 4E00          1    rel     0
INTVEC                         7FEE - 7FFD         10    com     0
BANK1                           0001:8000            0    bnk     0
TEST2                           0002:8000            0    bnk     0
DAVID3
```

Stack: [C-73]

Stack設的太小，程式執行上可能超過而造成潛藏的錯誤；設的太大將會浪費記憶空間；實務上可以先將欲使用的區域填上固定值，

然後再執行程式，最後檢視實際有變更的區域即可大致估算需要的空間。

Expression stack: 程式執行中需要臨時暫存結果的空間（X暫存器指示最上面的stack位置）。
Interrupt expression stack: 進入interrupt時期，需要臨時暫存結果的空間（X暫存器指示最上面的stack位置）。
Processor stack: CPU的硬體stack，儲存function呼叫的return地址，以及儲存CPU的狀態與暫存器資料。
Recursion stack: 執行recursion function時，儲存local變數及參數的空間。

Default stack size:

- Expression stack: 32(0x20)
- Interrupt expression stack: 4(0x04)
- Processor stack: 256(0x100)
- Recursion stack: 256(0x100)

OTHER:

1. putchar.c 及 getchar.c是IAR-C最基本的I/O呼叫功能，要使用其他C的I/O指令需先將這兩個function改寫成適合目前的MCU。[C-64]

2. 當有使用malloc or calloc，內定heap size是2000 bytes，要變更heap size請修改heap.c [C-78]

5. Compiler參數

CPU選擇 [C-70]
6502: (-v0)
65c02: (-v1)
65c02 protel: (-v2)

Memory模式選擇 [C-71]
Tiny bank: (-mb)
Large bank: (-mB)

SPEED OPTIMIZATION機制： [C-49]
可產生較佳執行效能的方式，內定值是3（-s）。

數 值	效 能	Debug操作情況
0	沒有最佳化	正常
1~3	普通效能	正常
4~6	較高效能	有些結構無法debug
7~9	最高效能	有些結構無法debug

PS: speed或size最佳化只能二選一，不能同時。

SIZE OPTIMIZATION機制： [C-50]
可產生較小空間的程式碼，內定值是3（-z）。

數 值	效 能	Debug操作情況
0	沒有最佳化	正常
1~3	普通壓縮	正常
4~6	較高壓縮	有些結構無法debug
7~9	最高壓縮	有些結構無法debug

PS: speed或size最佳化只能二選一，不能同時。

允許C語言擴充特定關鍵字 [C-42]
當需要使用到IAR-C的擴充字集，例如sfr, zpage……等，須要下此參數（-e）

允許 “//” 的用法 [C-43]
這是C++的用法，用來表示註解的方式，可以下此參數使它合法在IAR使用（-K）

關閉warnings的警告顯示 [C-43]
關閉Compiler時的警告訊息顯示（-w）

產生debug information [C-51]

要產生debug information必須指定此參數（-r）
在有需要Debug的場合，例如要使用SSI開發工具時，必須要指定此參數編譯，否則無法產生debugger需要的資訊。

6. 資料精度表示 [C-89]

Data type	Bytes	Range	Notes
Sfr			special-function registers 可直接宣告MCU的control register，在C裡面當變數直接用
char (default)	1	0 ~ 255	內定無號數
char (-c option)	1	-128 ~ 127	指定有號數
signed char	1	-128 ~ 127	有號數
unsigned char	1	0 ~ 255	無號數
short, int	2	-32768 ~ 32767	有號數
unsigned short unsigned int	2	0 ~ 65535	無號數
long	4	-2147483648 ~ 2147483648	有號數
unsigned long	4	0 ~ 4294967295	無號數
pointer	2		指標
float, double, long double	4	(+-)1.18E-38 ~ (+-)3.39E+38	有號浮點數

在6502 IAR-C中較有效率的資料表示：

1. 儘量使用8-bit資料型態。
2. 儘量使用unsigned資料型態。

7. IAR-C特有定義 / 擴充字集

7.1 擴充字集

zpage, npage	可指定變數在 Zero page或是Normal page
no_init	可指定變數在 non_volatile RAM
sfr	可直接定義變數在control register或I/O
Interrupt, monitor	指定中斷程序及非中斷程序
non_banked	宣告同bank函式使用

7.1.1 Interrup使用

“interrupt”是用在宣告中斷副程式，INT vector起始位置是定義在.xcl硬體描述檔上的”INTVEC”，使用格式如下：

Interrupt [vector] function-name()

1. “vector” 內容表示此函數的offset vector，也就是真實向量在 “INTVEC+vector”。
2. “vector” 可以省略，如果省略則向量將必須從CSTARTUP指定。
3. 中斷函式不可直接傳進或傳出參數。

7.1.2 Monitor使用

“monitor”是用在宣告禁止中斷副程式，也就是優先權凌駕中斷的程序可以放在裡面，執行這樣的程式將可受不被中斷的保護，在編譯上它將在函式內將中斷disable，使用格式如下：

monitor function-name()

7.1.3 Non_banked使用

“non_banked”是用來宣告這個函式將放在Non-bank下呼叫，使用格式如下：

non_banked function-name()

7.2 ＃PRAGMA定義

#pragma bitfields=default #pragma bitfields=reversed	Bitfield Orienation
#pragma codeseg(SEG_NAME)	Code Segment 決定程式碼的放置位址
#pragma language=default #pragma language=extended	Extension Control 讓Compiler可接受擴充指令
#pragma function=default #pragma function=interrupt #pragma function=monitor #pragma function=non_banked	Function Attribute
#pragma memory=constseg(SEG_NAME) #pragma memory=dataseg(SEG_NAME) #pragma memory=default #pragma memory=no_init #pragma memory=npage #pragma memory=zpage	Memory Usage
#pragma warnings=default #pragma warnings=off #pragma warnings=on	Warning Message Control

7.2.1 bitfields

“bitfields”宣告是用來指定bit資料結構的定義方向，爲了可以將C放在不同的平台上達到相容度，說明如下：

```
struct
{
        short a:3;      /* a is 3 bits */
        short :5;       /* this reserves a hole of 5 bits */
        short b:4;      /* b is 4 bits */
} bits;                 /* bits is 16 bits */
```

15 12	11 8	7 3	2 0
Hole(4)	b:4	Hole(5)	a:3

對照組如下：

```
#pragma bitfields=reversed
struct
{
        short a:3;      /* a is 3 bits */
        short :5;       /* this reserves a hole of 5 bits */
        short b:4;      /* b is 4 bits */
} bits;                 /* bits is 16 bits */
```

15 13 12 8 7 4 3 0

a:3	Hole(5)	b:4	Hole(4)

7.3 特定使用FUNCTION

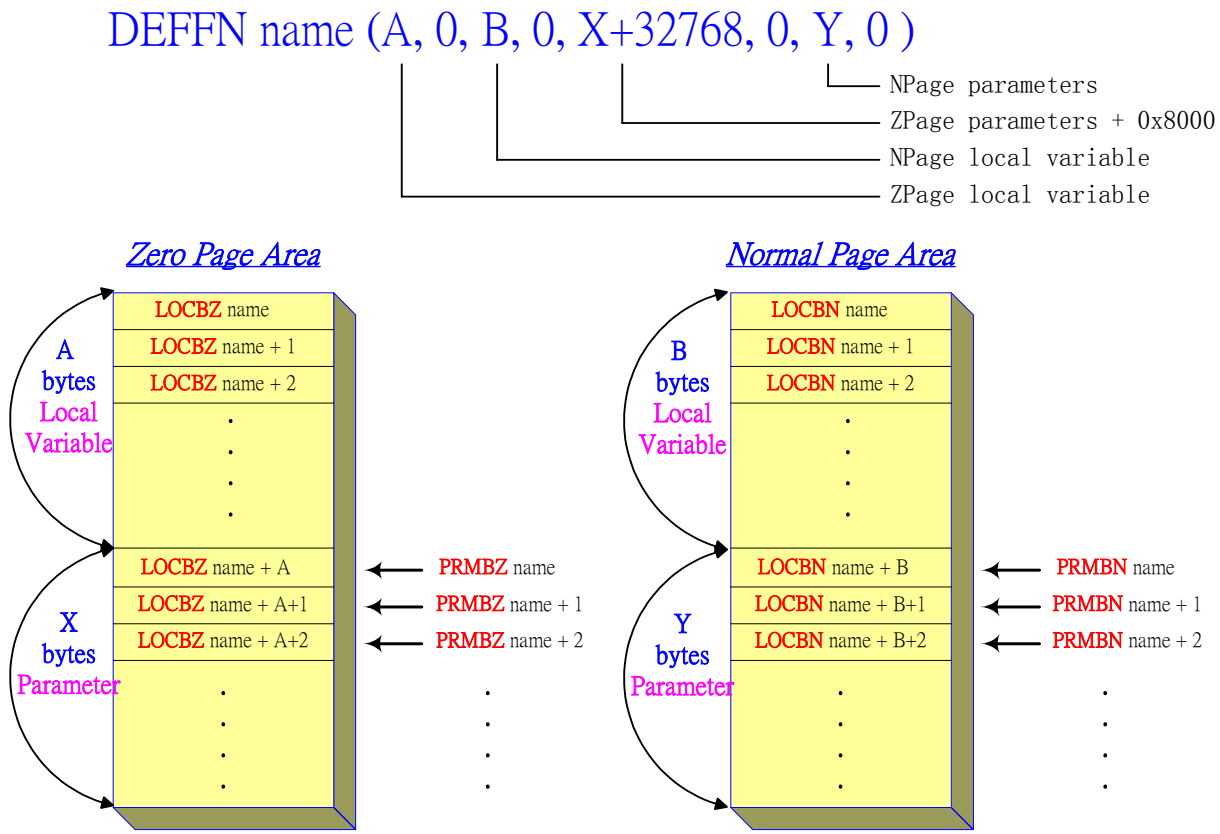
_args$	傳回函式中所有參數的型別（by array）
_argt$	傳回參數的型別
address_24_of	傳回函式或資料的bank與位址值
break_instruction	相當於6502 “BRK”
cld_instruction	相當於6502 “CLD”
disable_interrupt	相當於6502 “SEI”
enable_interrupt	相當於6502 “CLI”
nop_instruction	相當於6502 “NOP”

8. 使用ASSEMBLY介面

8.1 Keywords list

名 稱	說 明
DEFFN name	宣告指定的函數
IFREF name	指定函數的位址指標 (pointer)，可得到函數位址
LOCBN name	指定函數的Normal page Local變數起始位址
LOCBZ name	指定函數的Zero page Local變數起始位址
PRMBN name	傳入指定函數的Normal page參數起始位址
PRMBZ name	傳入指定函數的Zero page參數起始位址
REFFN name	代表指定函數進入點位址 (JSR呼叫用)

8.2 Function memory mapping



8.3 範例一

```
;-------------------------------------------------------------------------------
;       void TestFN (char X , char Y, char W, char H)
        public TestFN
        deffn  TestFN(0,0,1,0,32768,0,4,0)
;                     |   |     |     +---- NPage parameters
;                     |   |     +---------- ZPage parameters plus 0x8000
;                     |   +---------------- autos NPage
;                     +-------------------- autos ZPage
;-------------------------------------------------------------------------------
TestFN:
        STA   np:LOCBN TestFN+1     ; 儲存X參數.

IDB1:   LDA   np:LOCBN TestFN+3     ; W
        STA   np:LOCBN TestFN       ; 儲存到Local變數

        LDY   np:LOCBN TestFN+2     ; Y
        LDA   np:LOCBN TestFN+1     ; X
        LSR   A
        STA   IND

        LDA   np:LOCBN TestFN+1     ; X
        AND    #00000111b
        TAY

        INC   np:LOCBN TestFN+2     ; Y ++
        DEC   np:LOCBN TestFN+4     ; H --
        BNE   IDB1
        RTS
```

8.4 範例二

C source：

```
char assem(char pc1, char pc2, zpage int pi1, int pi2)
{
        int     my_a;
        int     my_b;
        my_a= pc1;
        my_b= pc2;
        return ( pi1+pi2 );
}

void main(void)
{
        int  main_x= 255;
        assem('x', 'y', main_x, 2);
}
```

Assembler：

```
        NAME        shell(18)
        RSEG        CODE(0)
        PUBLIC      assem
        DEFFN       assem(0,0, 4,0, 32770,0, 4,0)
        PUBLIC      main
        DEFFN       main(0,0, 2,0, 32768,0, 0,0),assem
        EXTERN      ?CL6502MDL_1_00_L00
RSEG CODE
; 1. char assem (char pc1, char pc2, zpage int pi1, int pi2)
; 2. {
assem:
        STA   np:LOCBN assem+4      ; pc1 （第一個byte參數放在A必須自己存）
; 3. int my_a;
; 4. int my_b;
; 5. my_a= pc1;
        LDA   np:LOCBN assem+4      ; pc1
        STA   np:LOCBN assem        ; my_a low
        LDA   #0
        STA   np:LOCBN assem+1      ; my_a high
; 6. my_b= pc2;
        LDA   np:LOCBN assem+5      ; pc2
        STA   np:LOCBN assem+2      ; my_b low
        LDA   #0
        STA   np:LOCBN assem+3      ; my_b high
; 7. return (pi1+pi2);
        LDA   np:LOCBN assem+6      ; pi2
        CLC
```

```
        ADC   zp:LOCBZ assem          ; pi1
; 8. }
        RTS                           ; 藉由A暫存器傳回運算值，並且結束函數
; 9.
; 10. void main (void)
; 11. {
main:
; 12. int main_x= 255;
        LDY   #255
        STY   np:LOCBN main           ; main_x  low
        LDY   #0
        STY   np:LOCBN main+1         ; main_x  high

; 13. assem('x', 'y', main_x, 2);
        LDA   #2
        STA   np:PRMBN assem+2        ; 02
        LDA   #0
        STA   np:PRMBN assem+3        ; 00
        LDA   np:LOCBN main           ; main_x  low
        STA   zp:PRMBZ assem
        LDA   np:LOCBN main+1         ; main_x  high
        STA   zp:PRMBZ assem+1
        LDA   #121
        STA   np:PRMBN assem+1        ; 'y'
        LDA   #120
        JSR   np:REFFN assem          ; 'x'
; 14. }
        RTS
        END
```

8.5 重點摘要

1. 傳入函數的參數，第一個byte會放在A暫存器，不會放在PRMBZ（或PRMBN）中。
2. 在呼叫一個函數前，須將參數的第一個byte放在A暫存器傳遞。
3. 呼叫函式後，“A”、“Y”暫存器以及Z、N、C flags 可能被修改，如有需要必須在事前作備份。
4. “X”暫存器通常用來做Expression stack指標。
5. 函數傳回值若只有一個Byte將透過“A”暫存器傳回；**若超過一個byte，則將改由推入Expression stack指標傳回。**
6. LOCBZ及PRMBZ會傳回8-bit address；LOCBN及PRMBN會傳回16-bit address。
7. 當DEFFN定義的函式不需宣告Local variable時，可以簡化成只擺兩組parameter參數（parameter參數不能省略）。
8. 要設計可以讓C使用的組合語言函數，需宣告成“PUBLIC”。

9. 應用

1. 將data放在bank上，由nonbank呼叫的方法。 [C-82]