

微處理機與介面設計序章

日期：2024/09/24

報告者：許宸華

Outline

- MPU vs MCU vs SoC
- 8051介紹
- C、Assembly、Machine code介紹
- 應用開發環境架設
- 軟體流程圖與範例程式碼
- Keil C51 : Debug mode介紹

Outline

- MPU vs MCU vs SoC
- 8051介紹
- C、Assembly、Machine code介紹
- 應用開發環境架設
- 軟體流程圖與範例程式碼
- Keil C51 : Debug mode介紹

MPU vs MCU vs SoC

- MPU (Micro Processor Unit) 微處理器單元
 - 將中央處理器CPU (Central Processing Unit) 封裝於一個晶片中。
 - 晶片內沒有其他周邊裝置，需要搭配其他ROM、RAM等裝置使用。
 - 例如：Zilog Z80、8086。
- MCU (Micro Controller Unit) 微控制器單元
 - 除了中央處理器，還包含其他周邊裝置封裝在同一晶片，常見的周邊裝置包含ROM、RAM、UART、Timer、I/O等。
 - 例如：8051、PIC、ATmega328、STM32系列等晶片。

MPU vs MCU vs SoC

- SoC (System on Chip) 系統晶片
 - 將各種具有運算核心功能的晶片連同RAM、ROM、振盪器等封裝在同一張晶片中。
 - 常見的SoC主要的架構包含了：
 - 負責運算的單元
 - 記憶體控制單元
 - 類比數位的轉換器
 - 管理供電來源的穩壓器
 - 根據功能需求，還可能加入GPU、DSP、網路晶片等
 - 例如：Raspberry Pi的主晶片BCM2837B0、BeagleBone的主晶片AM3358/9。

Outline

- MPU vs MCU vs SoC
- 8051介紹
- C、Assembly、Machine code介紹
- 應用開發環境架設
- 軟體流程圖與範例程式碼
- Keil C51 : Debug mode介紹

8051內部元件

- 運算核心為8位元的CPU。
- 內部有4KB的內部程式記憶體(ROM)和128Bytes的內部資料記憶體(RAM)。
 - 兩者最大皆可擴充至外部64KB。
 - 現今的8051多以Flash memory取代ROM。
- 具有內部震盪電路。
 - 利用石英震盪晶體搭配兩個電容即可對8051提供時脈來源。
- 具有4組可位元定址的GPIO (General-purpose input/output)
 - 分別為P0、P1、P2、P3。
 - 其中P0腳位設計上主要做為位址或資料傳輸用，若要拿來提供訊號需要外接上拉電阻。

8051內部元件

- 具有2組16位元的timer。
- 具有6個中斷來源。
 - 分別為INT0 (外部中斷0)、INT1 (外部中斷1)、T0 (計時器0)、T1 (計時器1)、RXD & TXD (UART)。
 - 另外還有一個reset中斷。
- 具有1組全雙工序列埠UART (Universal Asynchronous Receiver Transmitter)。

可位元定址:

一byte由8bits組成，當我們可直接控制1byte中任意1個bit的值為0或1，
這個byte即可位元定址的

8051腳位

- 晶片在出廠時，通常會做一些記號以便判讀腳位。
- 長條形的晶片經常會在晶片第1腳位那端做上圓點或圓圈的記號因此可以從8051上面的圓圈判斷晶片各個腳位的位置。

1	P1.0	VCC	40
2	P1.1	P0.0_AD0	39
3	P1.2	P0.1_AD1	38
4	P1.3	P0.2_AD2	37
5	P1.4	P0.3_AD3	36
6	P1.5_MOSI	P0.4_AD4	35
7	P1.6_MISO	P0.5_AD5	34
8	P1.7_SCK	P0.6_AD6	33
9	RST	P0.7_AD7	32
10	P3.0/RXD	EA/VPP	31
11	P3.1/TXD	ALE/PROG	30
12	P3.2/INT0	PSEN	29
13	P3.3/INT1	P2.7_A15	28
14	P3.4/T0	P2.6_A14	27
15	P3.5/T1	P2.5_A13	26
16	P3.6/WR	P2.4_A12	25
17	P3.7/RD	P2.3_A11	24
18	XTAL2	P2.2_A10	23
19	XTAL1	P2.1_A9	22
20	GND	P2.0_A8	21

AT89S51

8051內部暫存器

- R0、R1、R2、R3、R4、R5、R6、R7暫存器
 - 此8個暫存器主要被使用來存放資料，搭配各式各樣的指令以達成不同邏輯的實現。
- A 暫存器 (ACC，累加器)
 - A暫存器主要被用於各種數值的運算，包含了加、減、乘、除、左移、右移、AND、OR、XOR等。
- B暫存器
 - B暫存器通常與A搭配，被用在計算乘法以及除法上。
- P0、P1、P2、P3暫存器
 - 此4個暫存器分別對應到8051 GPIO的4組總共32支腳位，8051將會根據收到的指令將數值利用這些暫存器接收進來或是傳送出去。

除此之外，還有其他許多會影響8051表現的暫存器，將於後續實驗陸續介紹。

Outline

- MPU vs MCU vs SoC
- 8051 介紹
- C、Assembly、Machine code 介紹
- 應用開發環境架設
- 軟體流程圖與範例程式碼
- Keil C51 : Debug mode 介紹

8051程式碼編寫方式

- 現在編寫8051的程式碼主要有2種編寫方式
 - 使用C語言。
 - 使用Assembly語言。

8051開發語言	優點	缺點
C語言	<ul style="list-style-type: none">• 開發功能快速	<ul style="list-style-type: none">• 程式碼較大• 效能較差• 記憶體消耗較多• 無法或難以編寫部分功能
Assembly語言	<ul style="list-style-type: none">• 可以理解底層架構• 能夠掌控許多設定	<ul style="list-style-type: none">• 較為複雜

Preprocessor(預處理器)

- Preprocessor會將收到的C語言程式碼進行前置處理。
- 去除掉不必要的空白。
- 將有用「#」符號告知要預先處理的程式碼處理完畢。
 - 例如：將#include的header file加進程式碼開頭、將#define 的文字替換掉...

Compiler(編譯器)

- Compiler會根據自己的邏輯將preprocessor處理完的檔案轉換為assembly code。
- 將一些它認為不需要或者可以合併的程式碼進行最佳化。

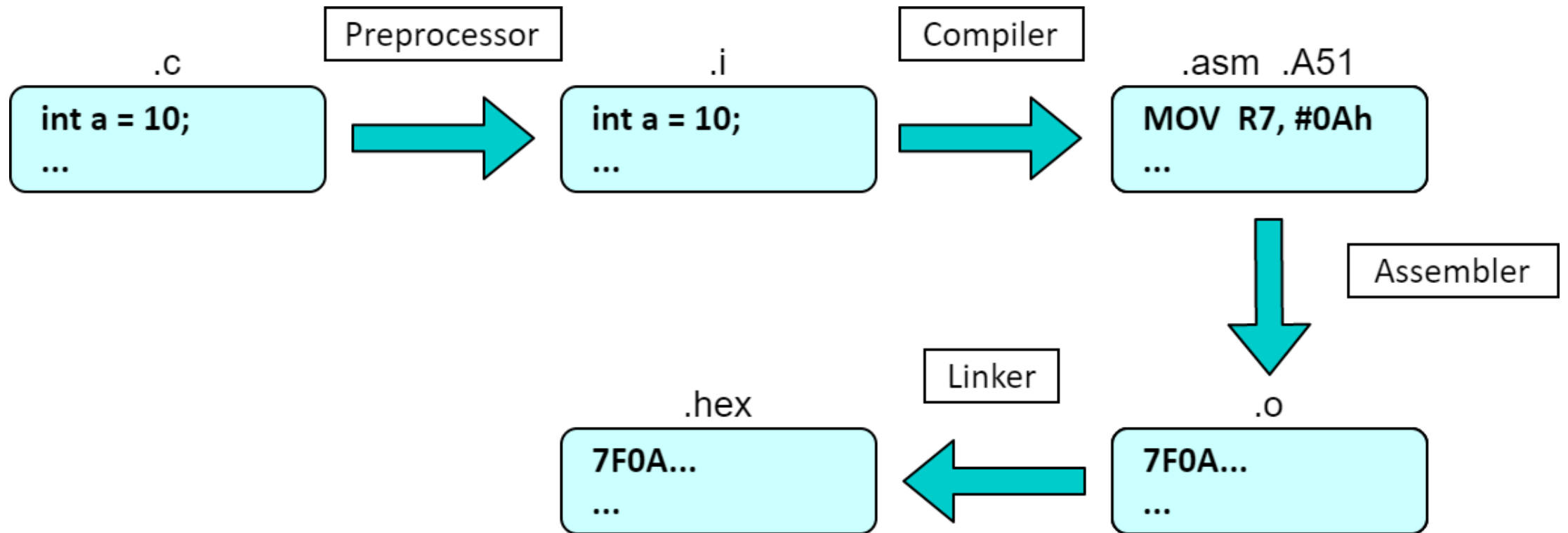
Assembler(組譯器)

- Assembler主要的任務是將assembly code轉換成給8051讀取的machine code。

Linker(鏈結器)

- 一個專案可能會有很多程式碼以及許多library(函式庫)。
- compiler和assembler在工作時會在個別產出的檔案中還無法標上實際位址的程式碼留下標記。
- Linker的工作便是將這些檔案結合起來，最終產出一支完整可以燒錄進8051執行的hex file (16進制的machine code)。

8051 C語言程式碼轉換為machine code流程



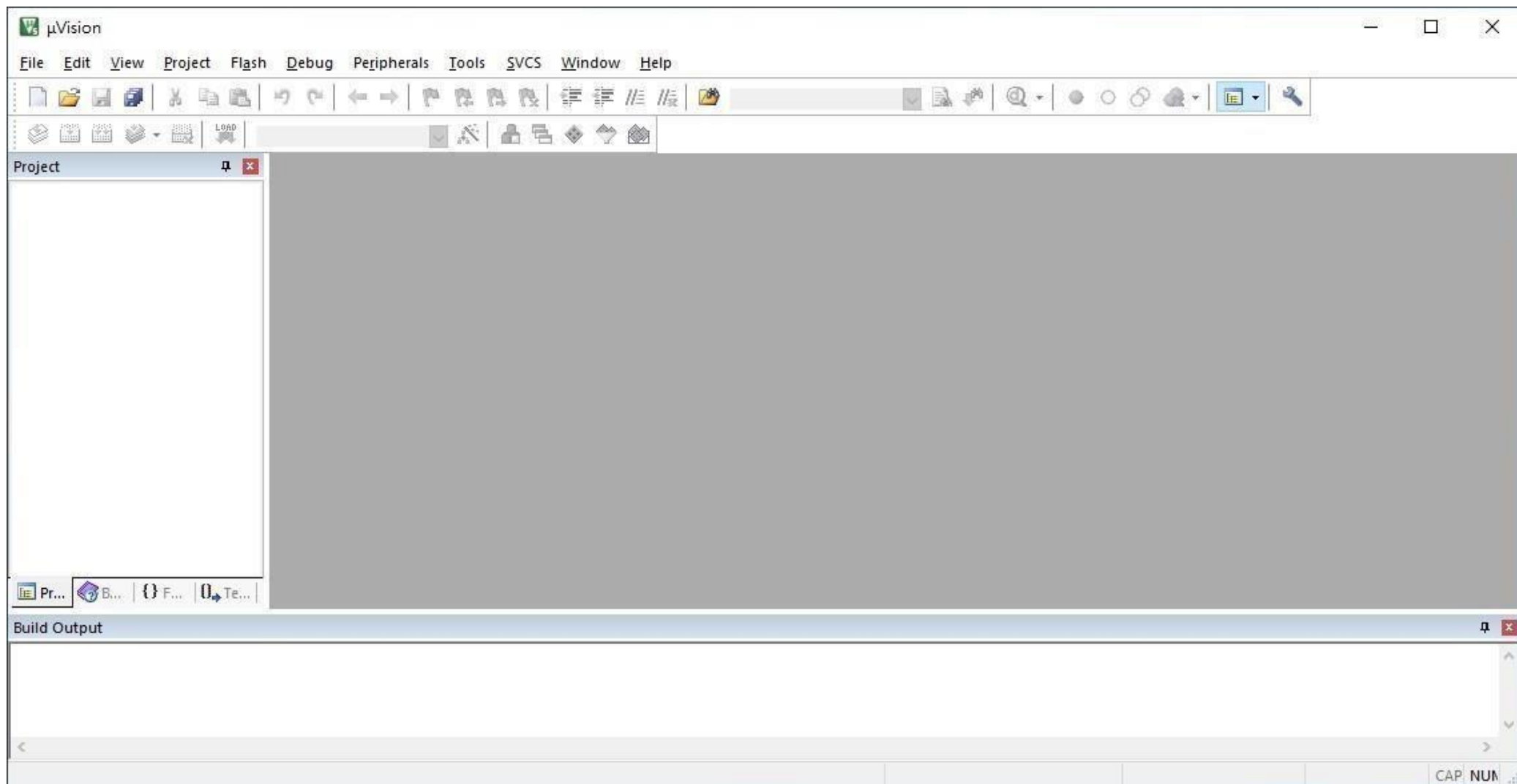
Outline

- MPU vs MCU vs SoC
- 8051介紹
- C、Assembly、Machine code介紹
- 應用開發環境架設
- 軟體流程圖與範例程式碼
- Keil C51 : Debug mode介紹

下載Keil C51

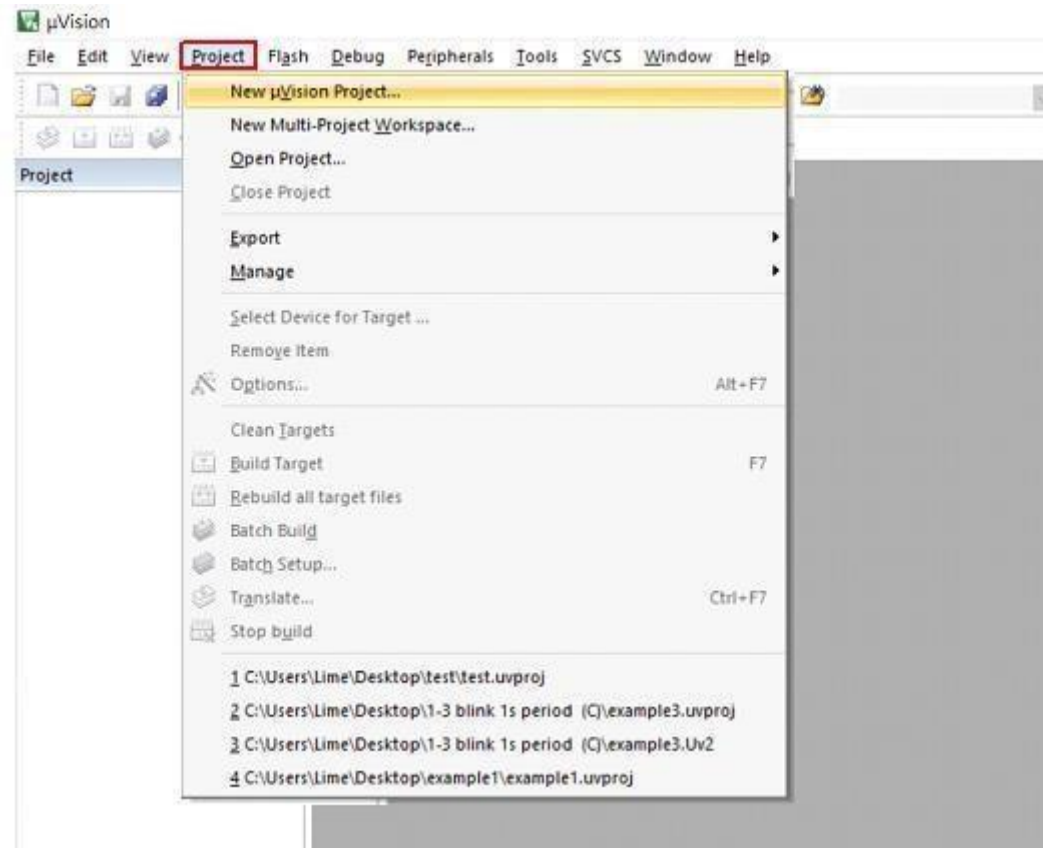
- 到[Keil C51下載網址](#)填寫資料後下載exe檔並安裝。
- 安裝時一直下一步即可，安裝路徑自由選擇。
- 該檔案為試用版，程式最多只能寫到2Kbytes，且不支援使用C語言與assembly語言混用，但本課程不需要用到這些功能，試用版已相當足夠。

安裝完Keil C51之後，打開會看見以下畫面



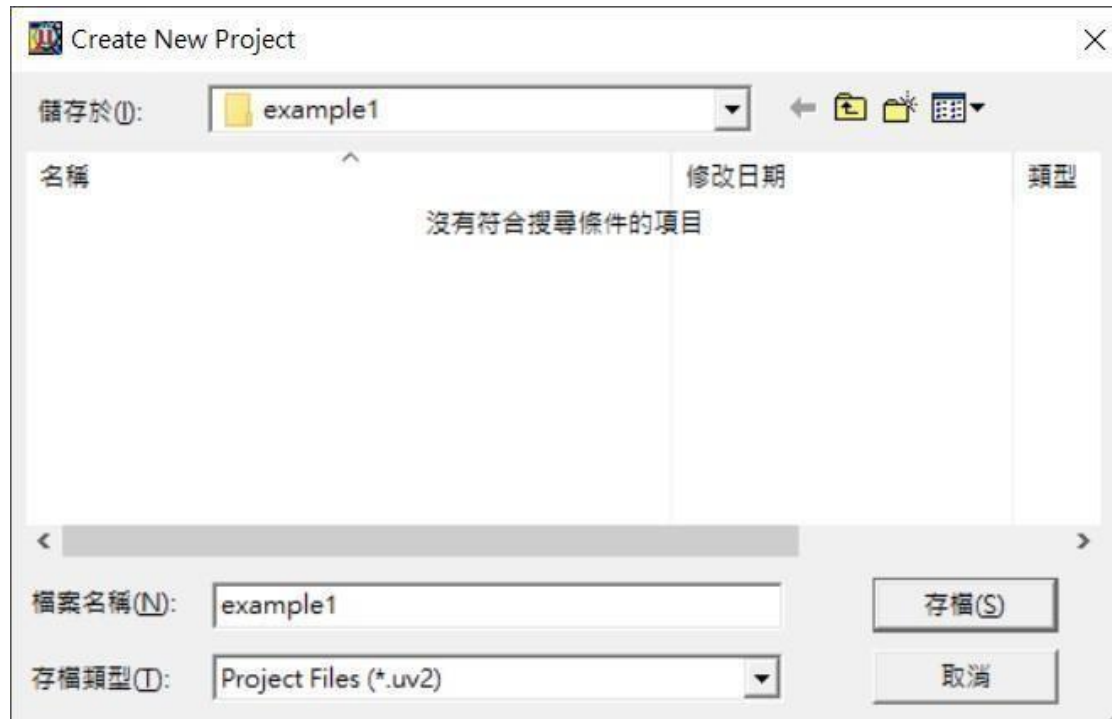
創立專案

- 在下拉式選單Project裡面選取New μ Vision Project。



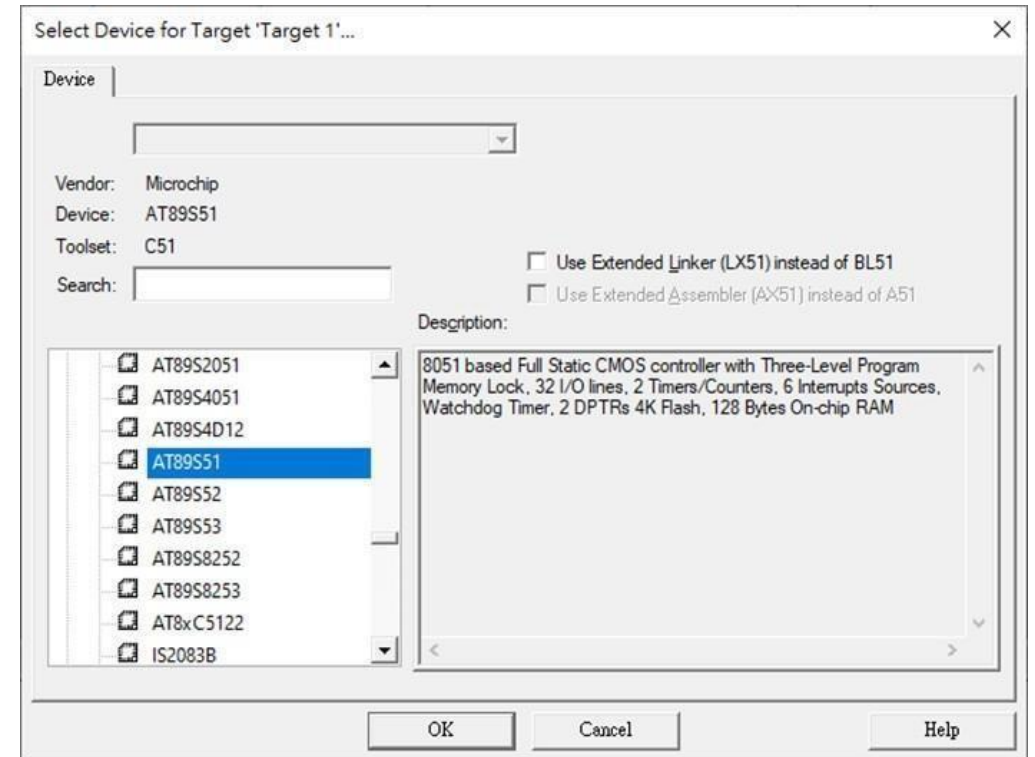
創立專案

- 選一個喜歡的資料夾創立專案，這邊在example1資料夾內創名為example1的專案。



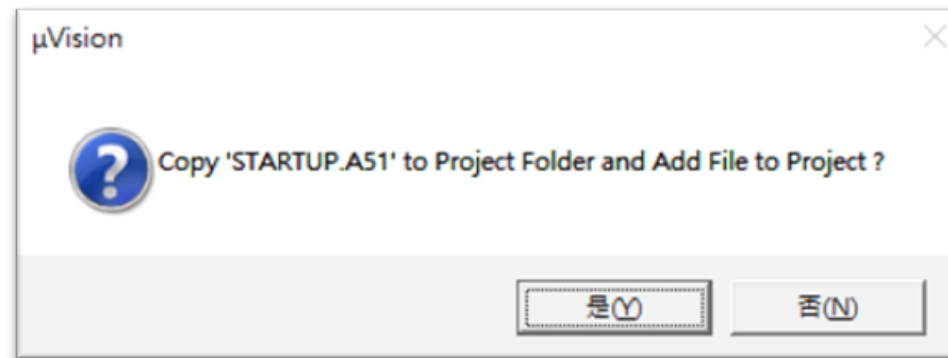
選擇晶片型號

- 我們使用的晶片大多是Atmel公司的AT89S51。
- 根據你的晶片上面的資訊做選擇有可能是AT89S52、AT89C52...
- 由於Atmel公司已經2016年被Microchip收購，因此此處左下角一開始選擇的廠商請找Microchip。



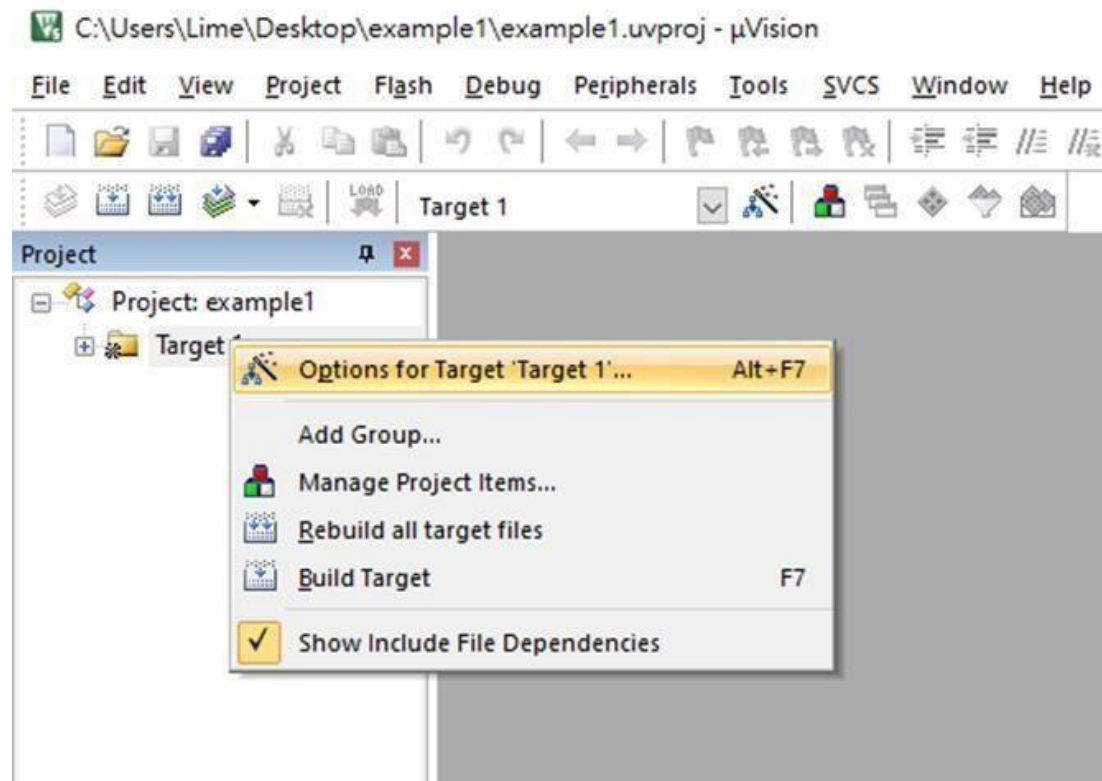
Startup Code

- 接著會問你要不要複製8051的startup code進專案。
- 它的用途是將一些暫存器初始化，好奇在做甚麼的可以搜尋 startup.A51: 用戶上電初始化程序
- 選是



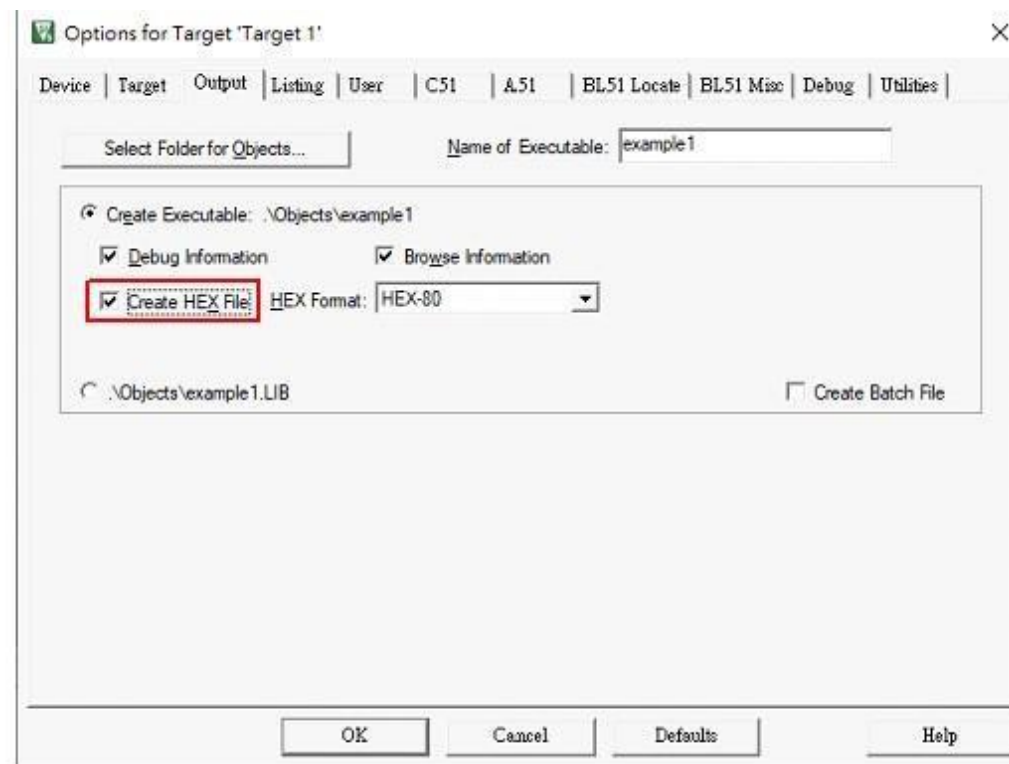
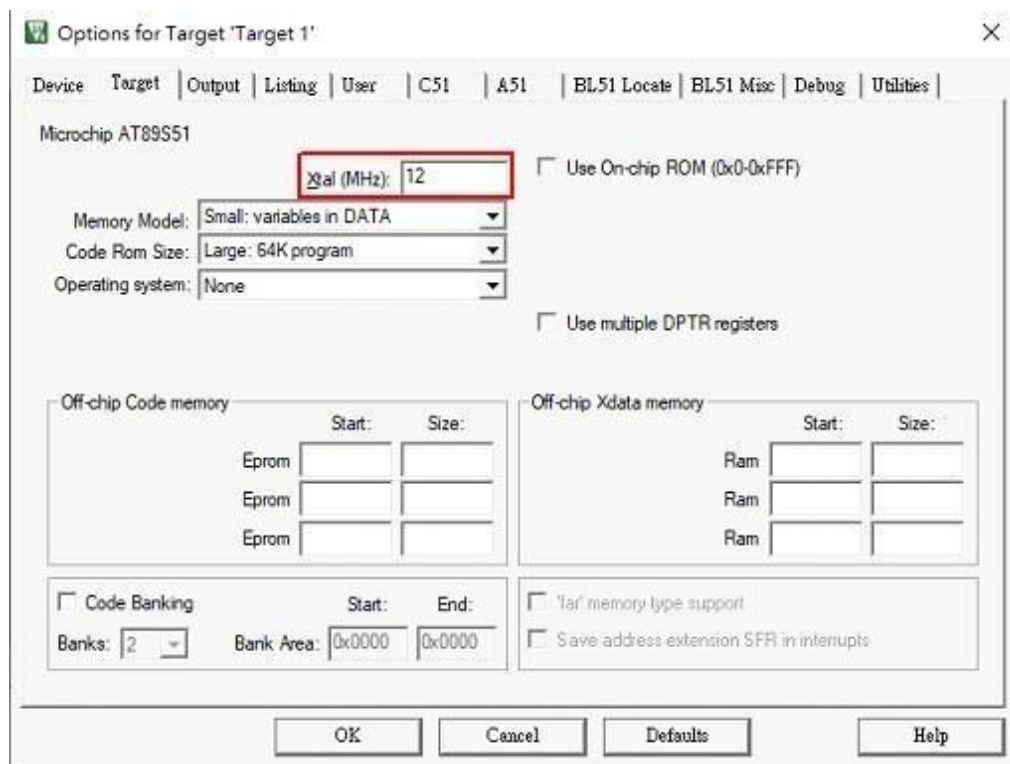
進行專案設定

- 對著Target 1 按滑鼠右鍵，選擇Options for Target 'Target 1'。



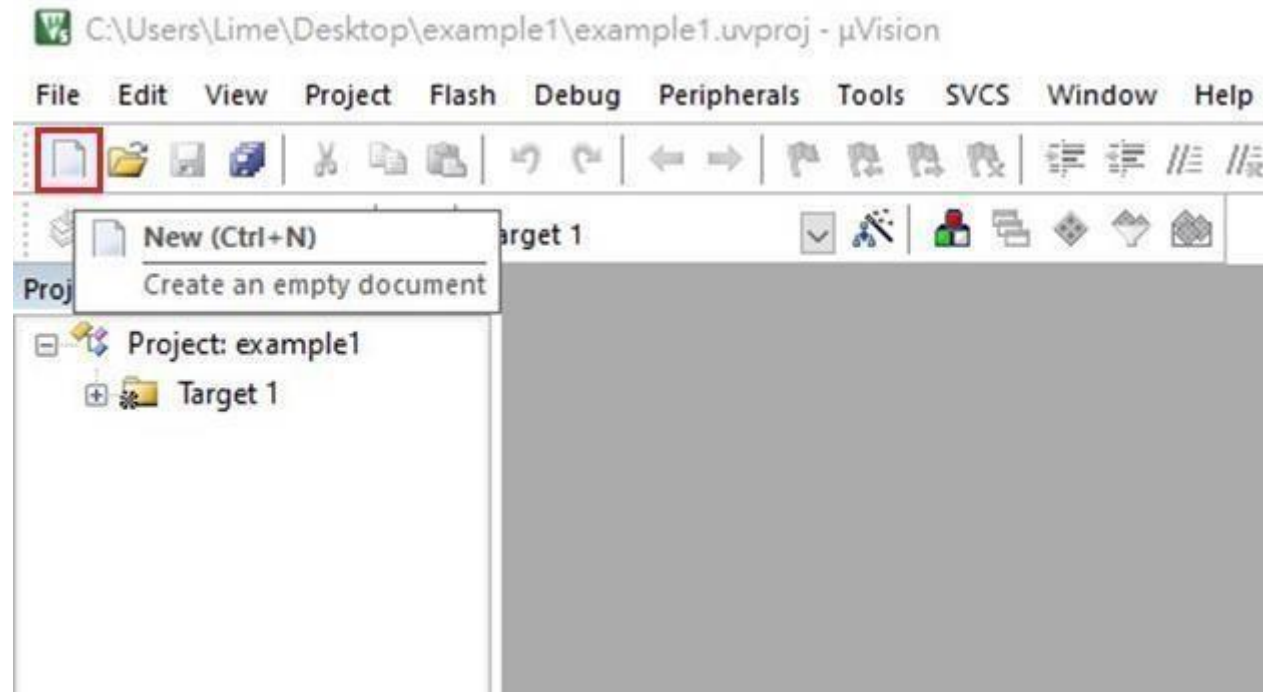
專案設定

- Target目錄，Xtal調整為12MHz(實驗所使用的石英震盪晶體頻率)。
- Output目錄，勾選 Create HEX File，該hex檔將會被燒錄進8051晶片中。



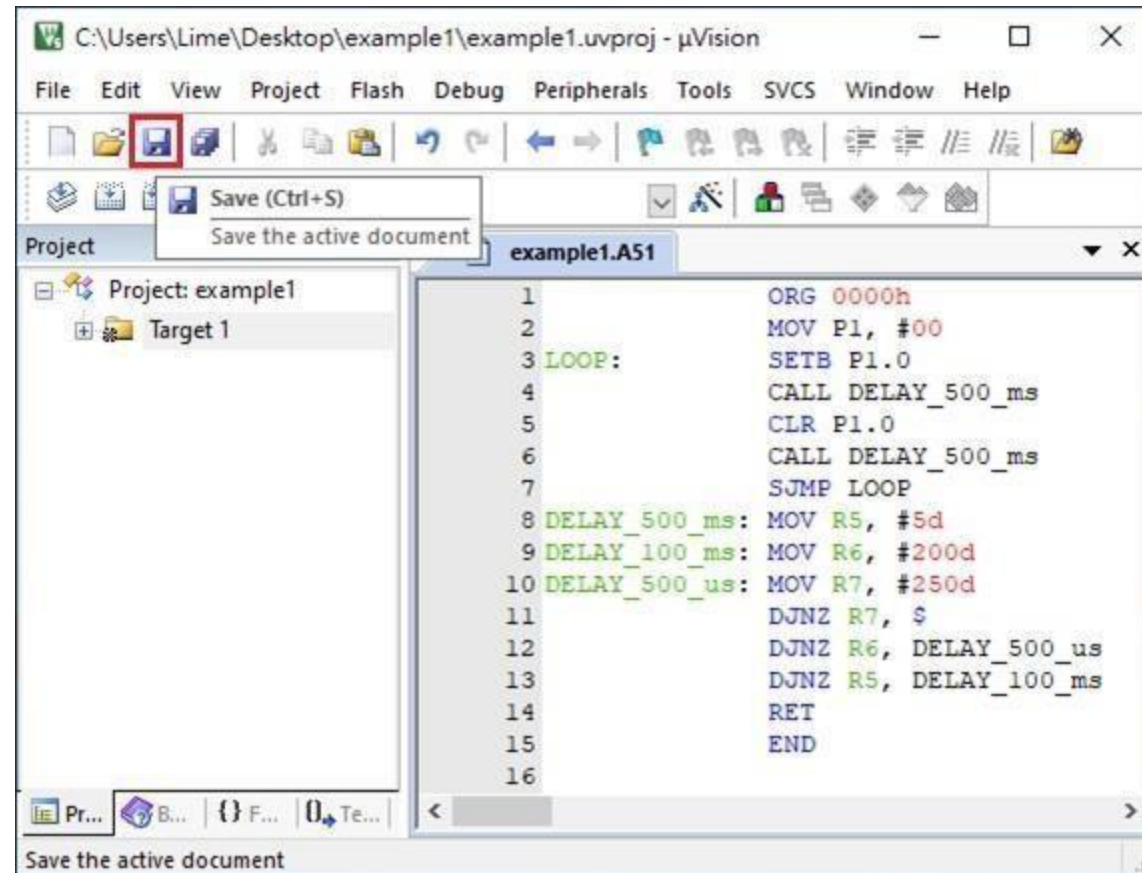
開始編寫8051 assembly language

- 在左上角的選單選擇New，新增檔案後開始寫程式。



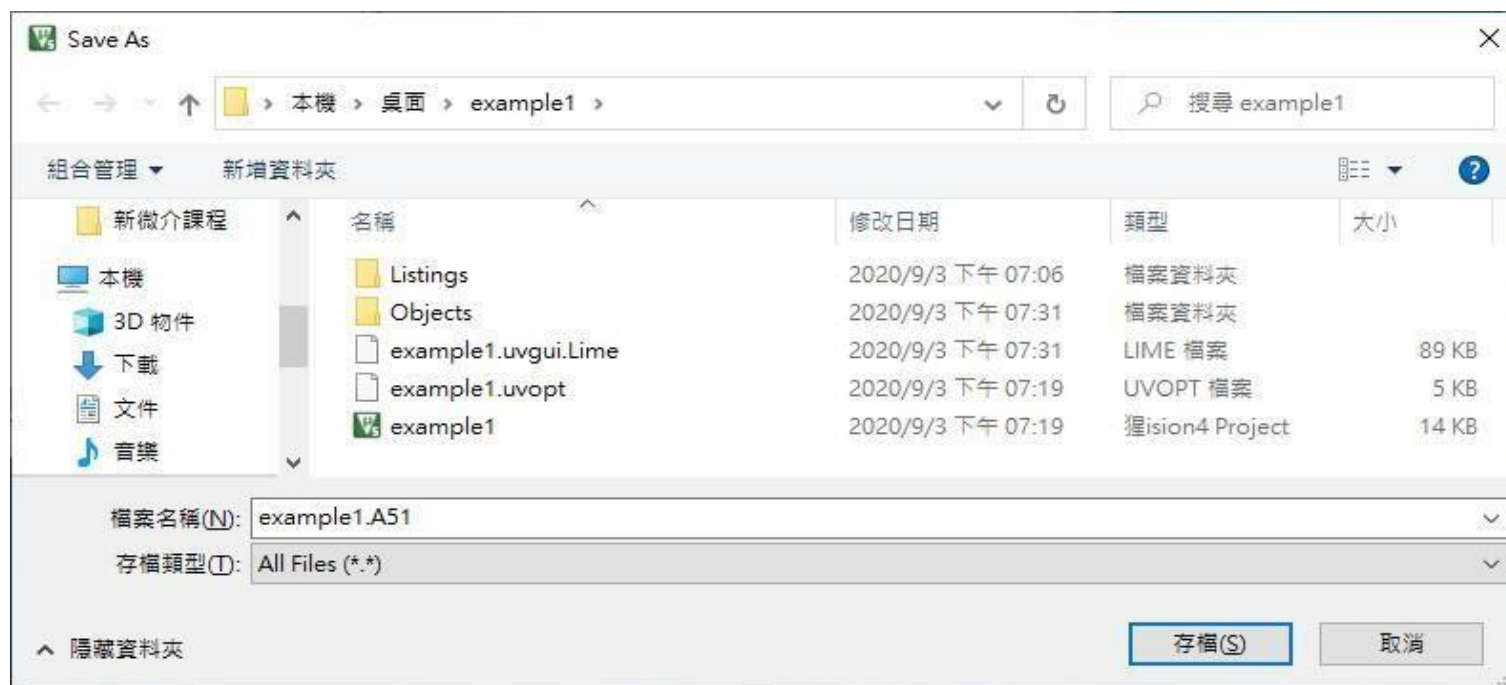
儲存程式碼

- 寫好程式碼後，按下save。



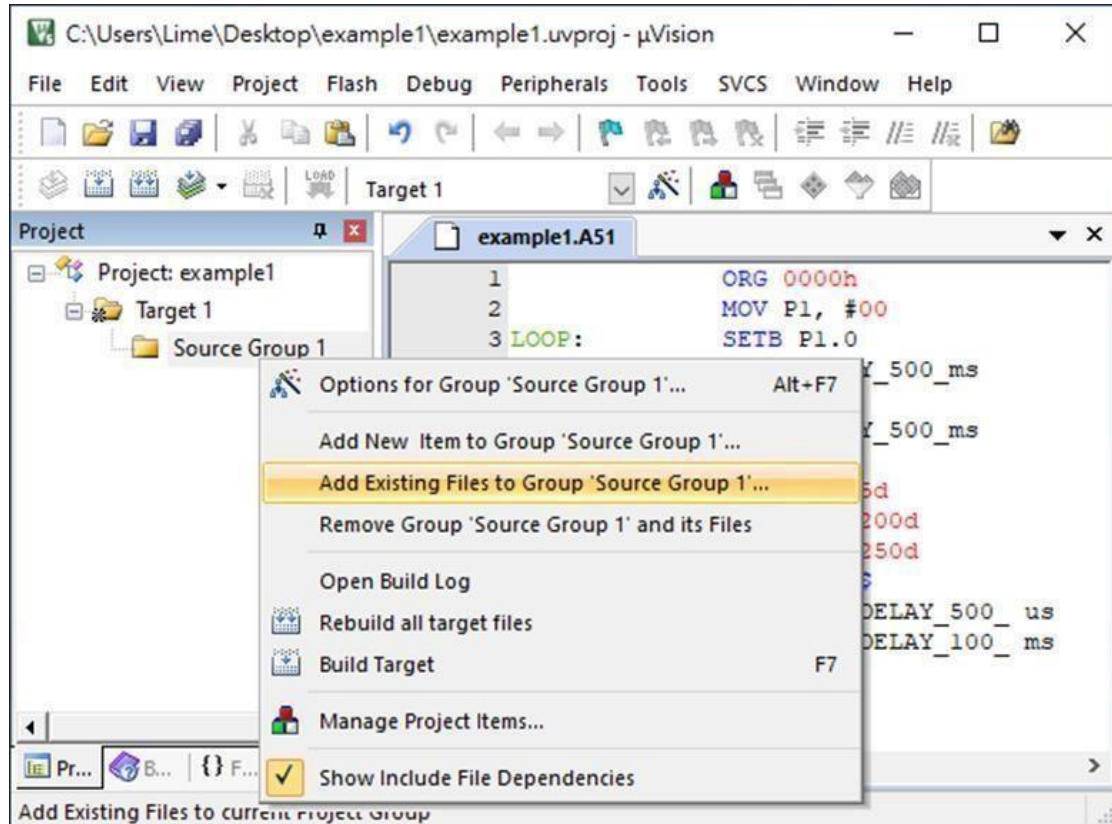
儲存程式碼

- 若為assembly 語言，將檔案儲存為自己訂的檔名.A51 。
- 若為c語言，則儲存為自己訂的檔名.c 。



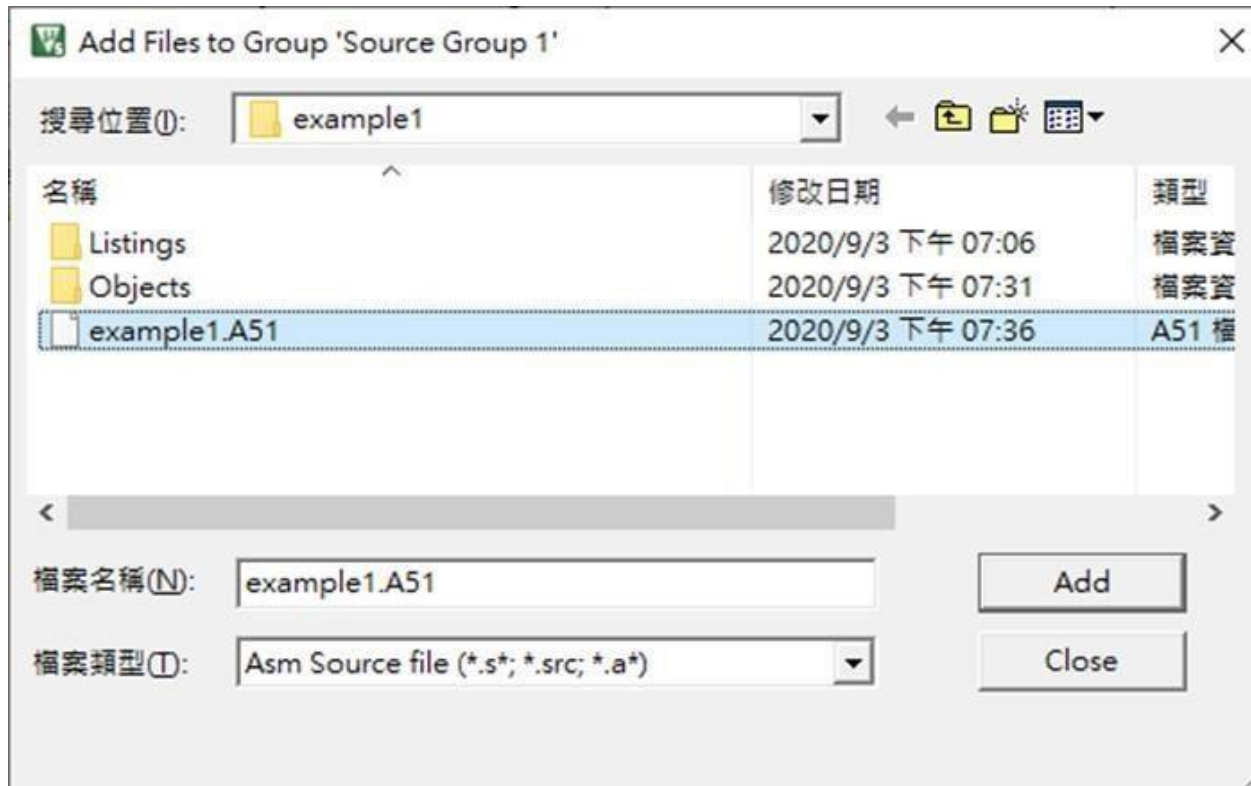
將檔案加入專案

- 在Source Group 1 按下滑鼠右鍵，選擇Add Existing Files to Group的選項將檔案加入專案。



將檔案加入專案

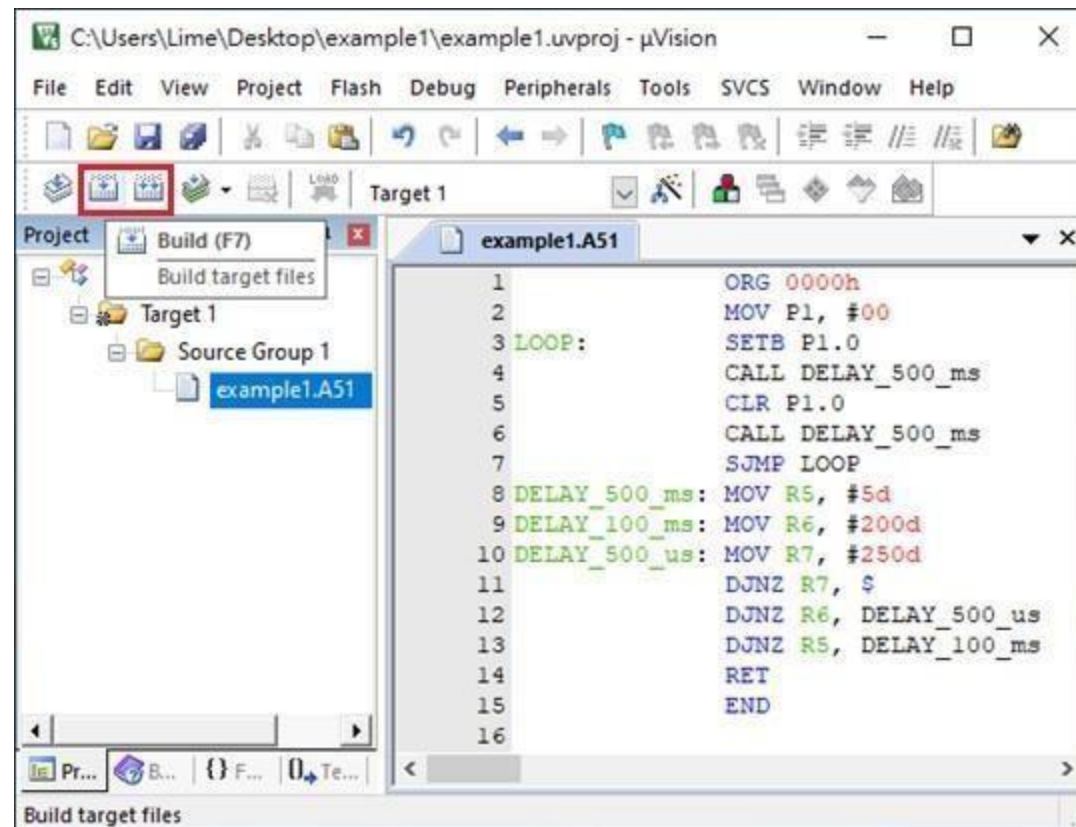
- 將檔案類型改成Asm Source file，選擇剛剛寫的程式碼檔案，按下Add後再按下Close。



編譯、產生hex檔

- 按下左上角的Build target (單箭頭向下) 或Rebuild all target files (雙箭頭向下) 後進行編譯，成功即會產生hex檔，失敗則要進行debug

。



安裝燒錄相關軟體

- 當同學們寫好程式並編譯出hex檔之後，接下來需要下載以下軟體搭配燒錄器進行燒錄。
 - 下載燒錄器SP200SE驅動程式，並根據你所使用的作業系統安裝驅動程式。
 - 下載並安裝WLPRO_V220_SETUP.exe燒錄軟體。

附註：此兩個檔案將會放在moodle上。

- 安裝好這兩項軟體後，接下來便可開始進行燒錄動作。

8051放置在燒錄器上的位置

- 請將燒錄器接到電腦後再把8051放上去。
- 讓8051有半圓圈的地方與燒錄器拉桿的位置離最遠，別放錯方向

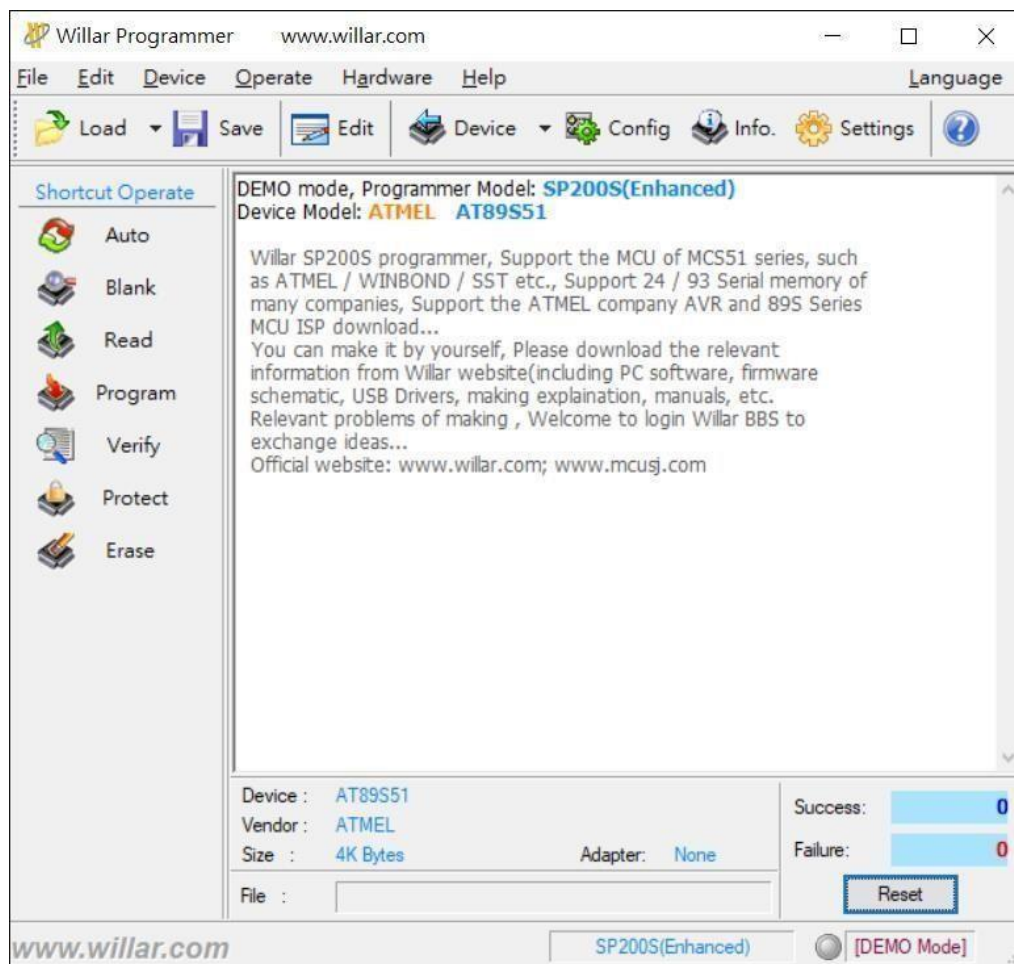
1	P1.0	VCC	40
2	P1.1	P0.0_AD0	39
3	P1.2	P0.1_AD1	38
4	P1.3	P0.2_AD2	37
5	P1.4	P0.3_AD3	36
6	P1.5_MOSI	P0.4_AD4	35
7	P1.6_MISO	P0.5_AD5	34
8	P1.7_SCK	P0.6_AD6	33
9	RST	P0.7_AD7	32
10	P3.0/RXD	EA/VPP	31
11	P3.1/TXD	ALE/PROG	30
12	P3.2/INT0	PSEN	29
13	P3.3/INT1	P2.7_A15	28
14	P3.4/T0	P2.6_A14	27
15	P3.5/T1	P2.5_A13	26
16	P3.6/WR	P2.4_A12	25
17	P3.7/RD	P2.3_A11	24
18	XTAL2	P2.2_A10	23
19	XTAL1	P2.1_A9	22
20	GND	P2.0_A8	21

AT89S51



將hex檔燒錄進8051

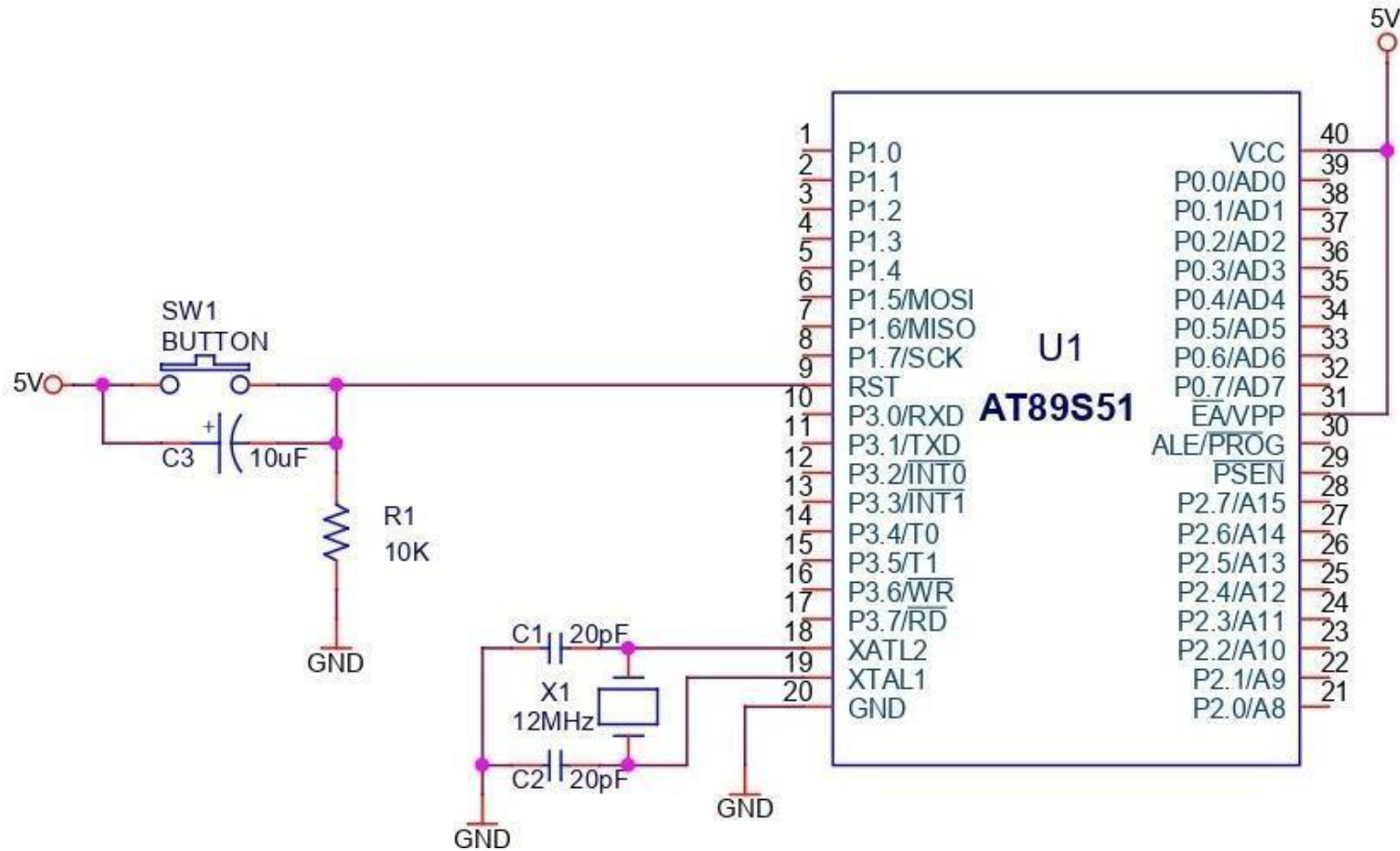
- 將燒錄器連接到電腦後啟動WLPRO燒錄軟體，會看到以下介面。



將hex檔燒錄進8051

- 若沒連接好會顯示DEMO mode選項。
- 在Device的地方點選手上的晶片型號。
 - 可能為AT89S51、AT89S52、AT89C52...
- 接著依以下步驟進行燒錄。
 - Erase(清空8051內部FLASH的資料)。
 - Load(選擇剛才編譯好的hex檔)。
 - Program(將hex檔燒錄進8051)。
 - Verify(驗證)。
附註：這4個步驟若有問題通常是硬體發生故障。
- 完成步驟便成功將程式碼燒錄進8051的FLASH memory。

8051基本電路圖



8051基礎電路圖注意事項

- 8051所使用的Vcc為5V。
- 20腳位接地。
- 40腳位接Vcc。
- 31腳位接Vcc代表使用8051內部的程式記憶體，接地代表使用外部程式記憶體，本課程實驗不會用到外接記憶體，因此接Vcc
- 18、19號腳位中間接有石英震盪晶體，本課程所使用的時脈為12 MHz，提供8051時脈來源，另外接兩個陶瓷電容讓震盪訊號更穩定。
- 9號腳位為reset，將系統上電後通常要將此腳位通Vcc觸發reset 8051才會正常運作。
- 由於本課程進行的實驗較為簡單可以先不考慮debounce，將9號腳位用一條杜邦線接到Vcc進行reset。

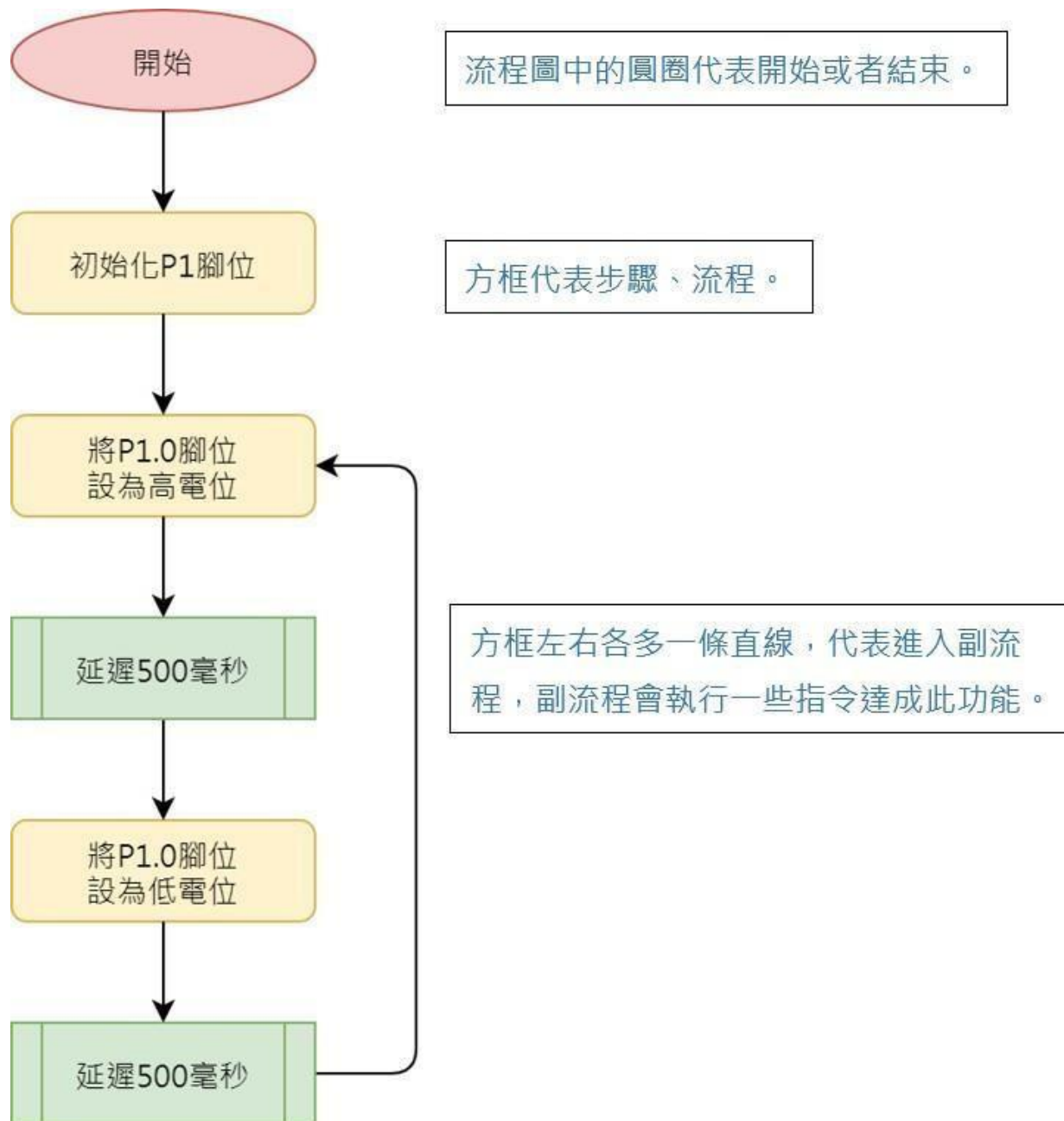
Outline

- MPU vs MCU vs SoC
- 8051介紹
- C、Assembly、Machine code介紹
- 應用開發環境架設
- 軟體流程圖與範例程式碼
- Keil C51 : Debug mode介紹

軟體流程

- 以下將會以assembly和c語言分別撰寫兩支功能相近的程式碼。
- 利用三個迴圈讓8051的P1.0腳位以約略1秒為週期輸出5V、0V。
- 此部分一開始若看了無法理解部分內容沒關係，掌握一些技巧和對內容有些印象即可。

軟體流程圖



Assembly語言範例程式碼

```
1.          ORG 0                                ;code start from 0
2.          MOV P1, #00H                          ;set P1 pins low
3.  LOOP:    SETBP1.0                             ;set P1.0 pin high
4.          CALL DELAY_500_ms                     ;call delay function
5.          CLR P1.0                              ;set P1.0 pin low
6.          CALL DELAY_500_ms                     ;call delay function
7.          SJMP LOOP                             ;jump back to loop symbol
8.  DELAY_500_ms: MOV R5, #5                       ;move decimal 5 into R5 register
9.  DELAY_100_ms: MOV R6, #200                    ;move decimal 200 into R6 register
10. DELAY_500_us: MOV R7, #250                   ;move decimal 250 into R7 register
11.          DJNZ R7, $                            ;jump to itself, it cost 2microseconds for R7(250) times
12.          DJNZ R6, DELAY_500_us                ;jump to address DELAY_500_us for R6(200) times
13.          DJNZ R5, DELAY_100_ms                ;jump to address DELAY_100_ms for R5(5) times
14.          RET                                  ;return to call function address
15.          END                                  ;end the code
```

附註：assembly程式碼中的「;」功能同C語言的「//」，用來做註解用。

Assembly code 程式碼補充

- DELAY_500_us函數執行了1次第9行的MOV指令 (耗時1微秒) 以及第11行的DJNZ指令250次(耗時500微秒)，因此總耗時其實為501微秒
- DELAY_100_ms和DELAY_500_ms以此類推，最終總共會累計達 500毫秒多一些。
- 這邊同學們可以先體會一下assembly語言的邏輯即可。

補充

- 程式碼通常不會從0000h的位置開始寫(*ORG 0000h*)，詳情留待實驗第六章「中斷」進行介紹，但目前初學先不用考慮。
- 當使用12MHz的石英震盪晶體驅動8051時，每12次的震盪被8051定義為1個 machinecycle，耗時1微秒，而一個指令消耗的時間便是以不同個machine cycle為單位。
- 若不懂assembly code的指令功能，可以到[arm KEIL 8051指令集](#)查詢
- 若想要知道每個指令耗時多久，可以看moodle上附的8051指令集。

C語言範例程式碼

```
1.  #include <regx51.h>
2. void delay (unsigned int);
3.
4. void main()
5. {
6.     P1 = 0x00;    //set P1 pins low
7.     while(1)      //infinite loop
8.     {
9.         P1 = 0x01; //set P1.0 pin high
10.        delay(5);   //delay 5*100 milliseconds
11.        P1 = 0x00;  //set P1.0 pin low
12.        delay(5);   //delay 5*100 milliseconds
13.    }
14. }
15.
16. //delay about a*100 milliseconds
17. void delay(unsigned int a)
18. {
19.     unsigned char i, j;
20.     for( ; a>0; a--)
21.         for(i = 0; i < 131; i++)
22.             for(j = 0; j < 250; j++);
23. }
```

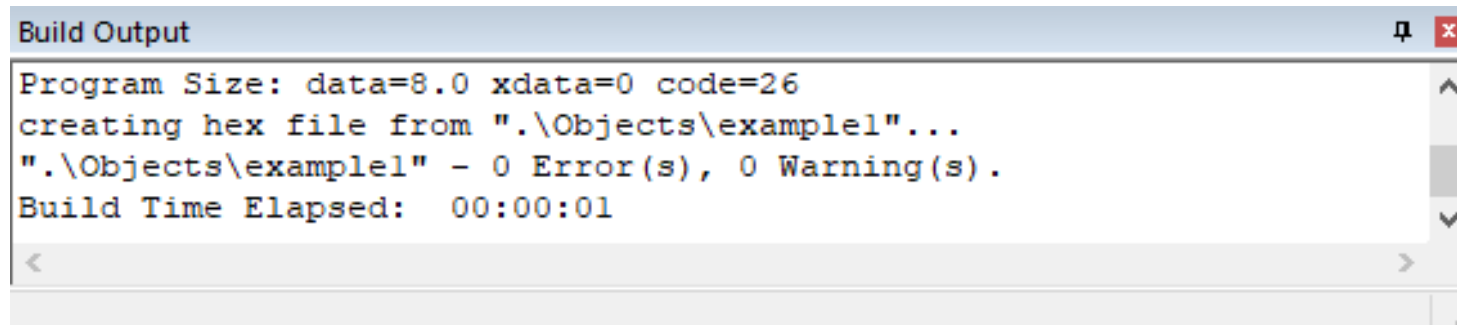
C code 程式碼補充

- 此處C語言的程式碼同樣使用了三個迴圈的方式來進行延遲，雖然每個迴圈延遲的時間不相同，但最後也可以做出接近延遲0.5秒的效果。
- `regx51.h` 內定義了許多暫存器、旗標等對應到的位址。
- `delay(5)`將會執行5次延遲約為100毫秒的空迴圈，達到延遲約為0.5秒的效果。
- 實際上要精確得知延遲了多久需要將hex檔反組譯(disassembly)，將總共消耗的machine cycle計算出來。
- 此處同學可以體會到程式碼雖然較簡單，但無法掌控的因素變多了

補充

- 用C語言(高階語言) 進行程式碼編寫，通常會降低效能、 增加程式碼體積與記憶體消耗
- 以上述的兩個範例程式碼為例：
 - assembly code編譯出的hex檔大小佔8051 FLASH memory的空間為26 Bytes。
 - C code編譯出的hex檔大小佔8051 FLASH memory的空間為66 Bytes。

附註：要觀察編譯出的machine code佔8051 FLASH memory的空間為多少時，Keil C51的Build視窗在編譯程式後會顯示相關資訊。



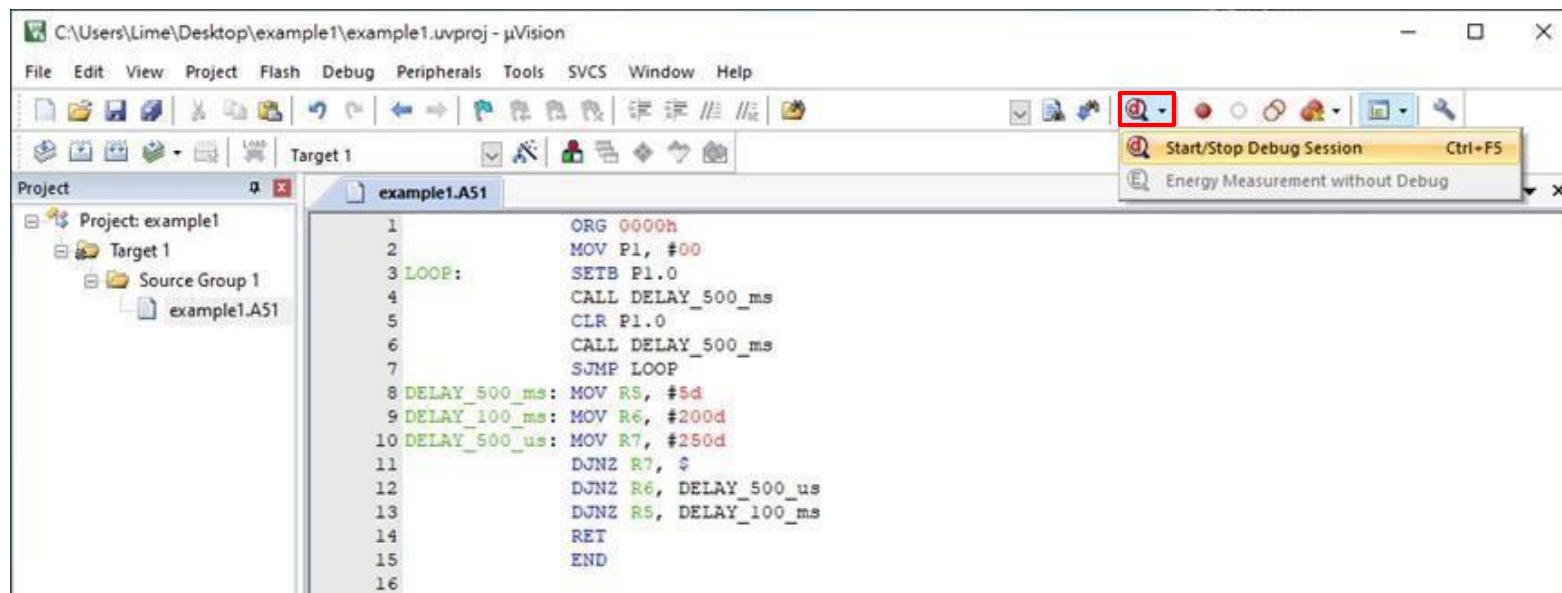
```
Build Output
Program Size: data=8.0 xdata=0 code=26
creating hex file from ".\Objects\example1"...
".\Objects\example1" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:01
```

Outline

- MPU vs MCU vs SoC
- 8051介紹
- C、Assembly、Machine code介紹
- 應用開發環境架設
- 軟體流程圖與範例程式碼
- Keil C51 : Debug mode介紹

Keil C51 進入 debug mode

- 選擇下圖中的放大鏡可以進入debug模式。
- 需將程式碼編譯過後再啟動。
- 該模式下可以輕易的檢查8051執行過程中的狀態，對學習與開發有相當大的幫助。



模擬程式碼執行

- Debug模式下，會發現編譯等選項改變為模擬程式運行的功能。



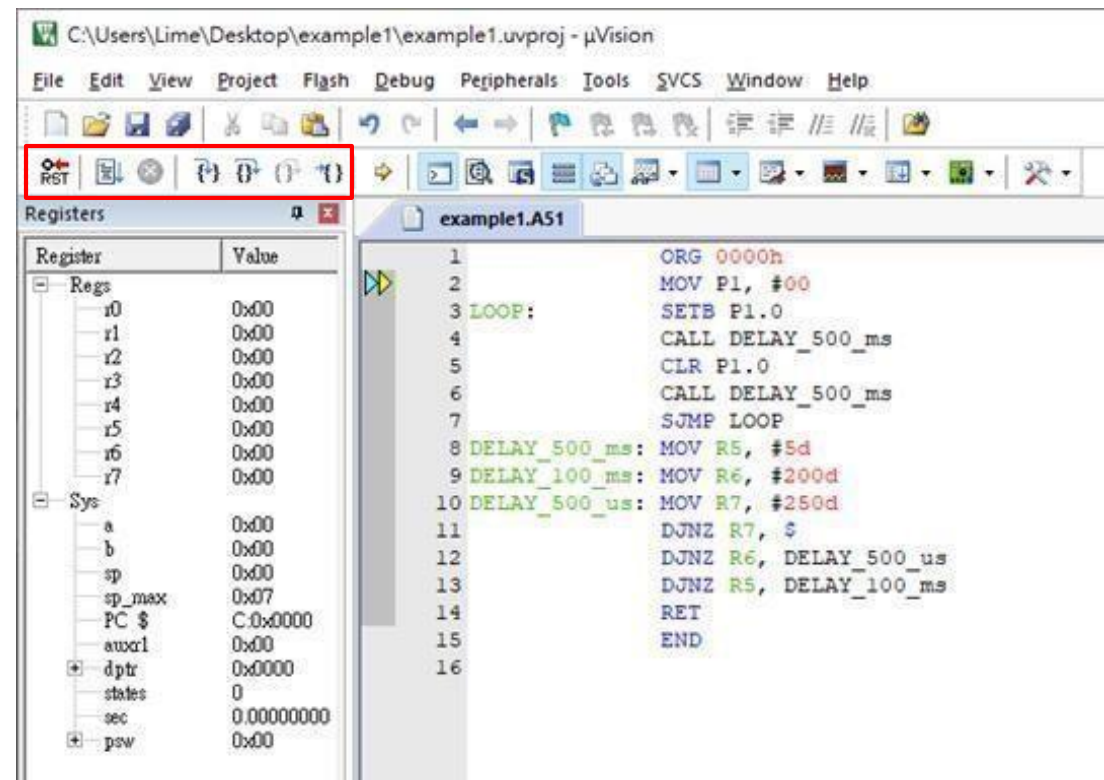
Reset CPU: 模擬8051觸發reset腳位後的情況。



Run: 模擬程式碼執行。

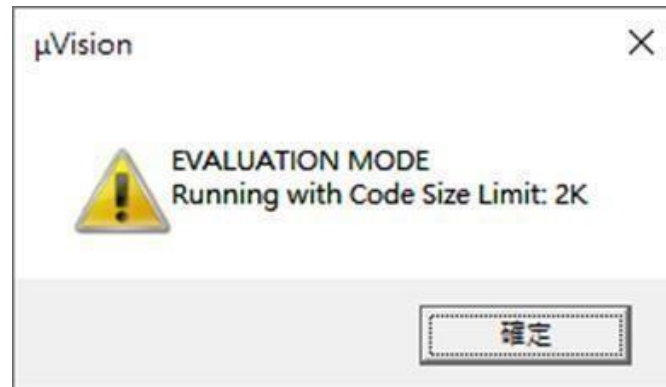


Halt: 暫停程式碼執行。







Code Size 2K bytes警告

- 由於使用的是試用版的Keil C51，因此會顯示程式碼大小有2kbytes限制的警告。
- 本課程實驗的程式碼通常不會寫到這麼大，直接按確定。

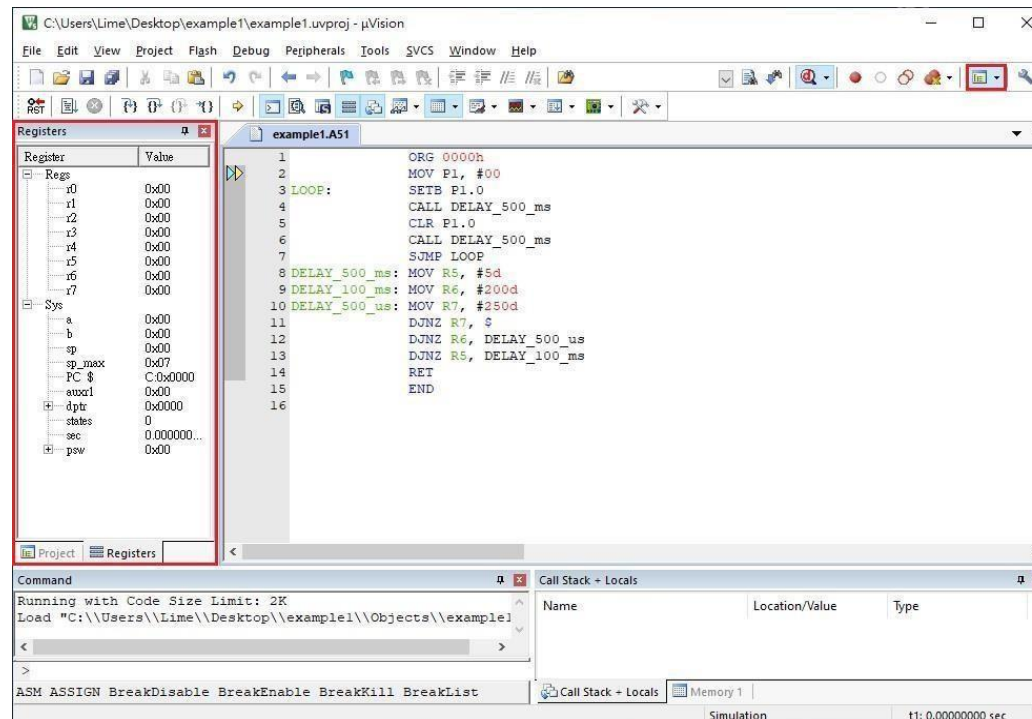


模擬程式碼執行

-  Step into: 模擬程式碼一行一行執行(單步執行)，按一次執行一個指令，可以根據函式進行跳躍。
-  Step over: 平常時功能同單步執行，如果那一行有呼叫函式時使用的話會直接執行完函式內容並回傳。
-  Step out: 執行完目前所在的函式並回傳。
-  Run to Cursor line: 將指令執行到滑鼠游標標記的那一個指令。

Project Windows

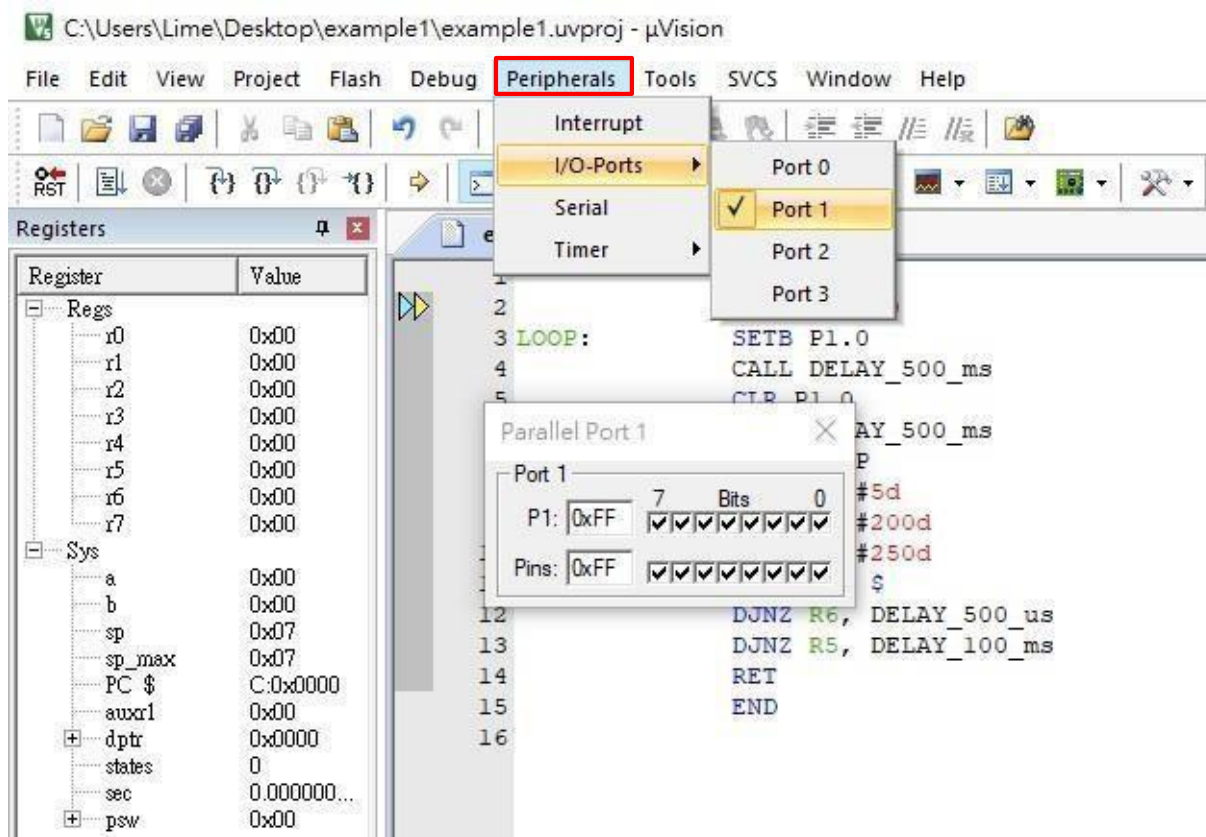
- 打開這個視窗，平常時會顯示你的專案檔案內容。
- Debug模式時，它會自動切換為顯示8051暫存器的狀態。
- 可以搭配單步執行等功能，觀察8051暫存器的變化。



周邊設備(Peripherals)

- 可以搭配單步執行等功能模擬8051 GPIO腳位上週邊設備的狀態

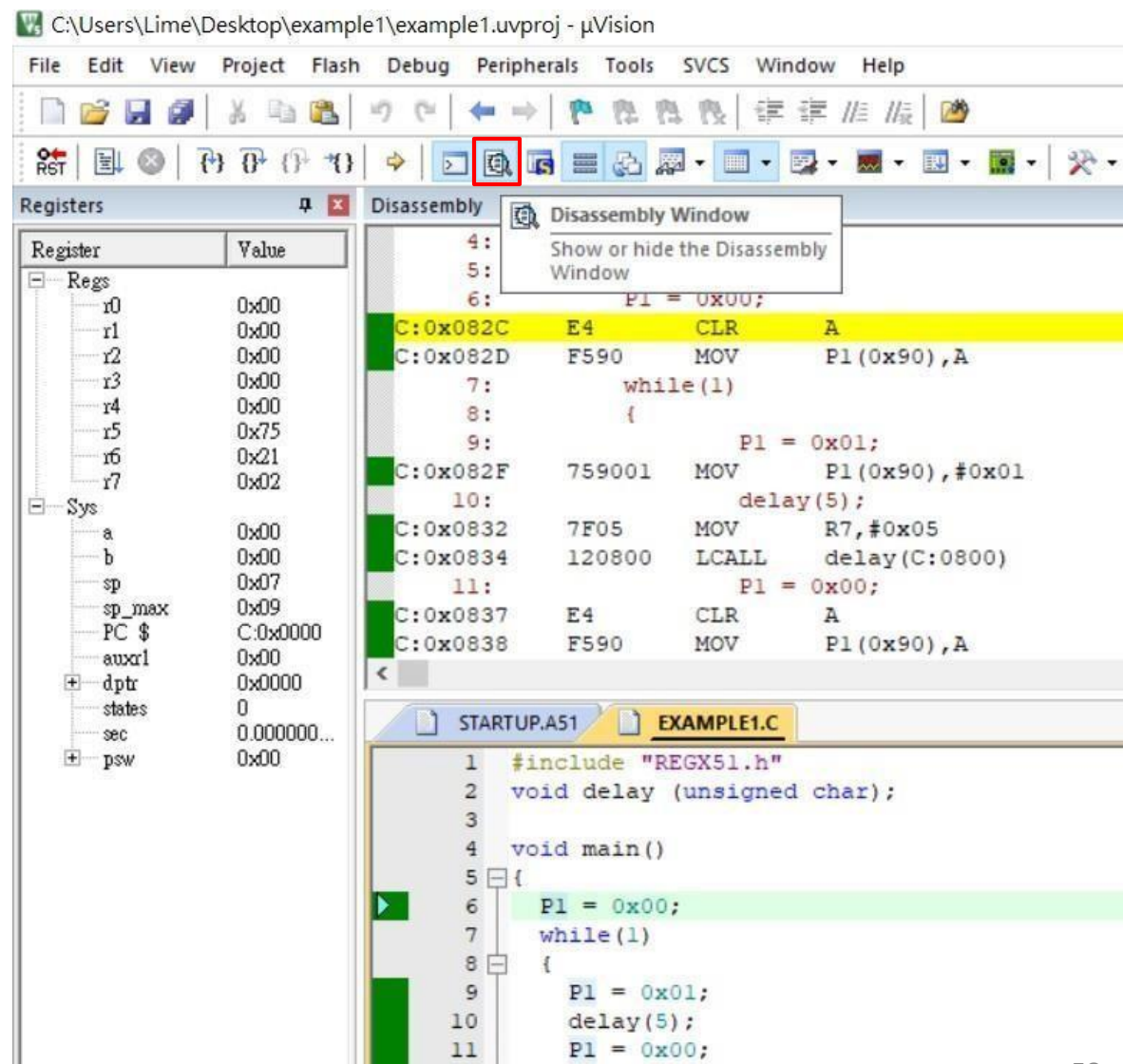
◦



Disassembly Window

- 模擬將編譯過後的機器碼反組譯成組合語言
- 可以看出每個指令在 8051ROM 所在的位址。
- 可以利用此功能搭配單步執行，觀察編譯後的邏輯是否有bug或是compiler 進行程式碼最佳化。

附註：程式碼若沒編譯過此處指令將會全部都是NOP。



Q&A