

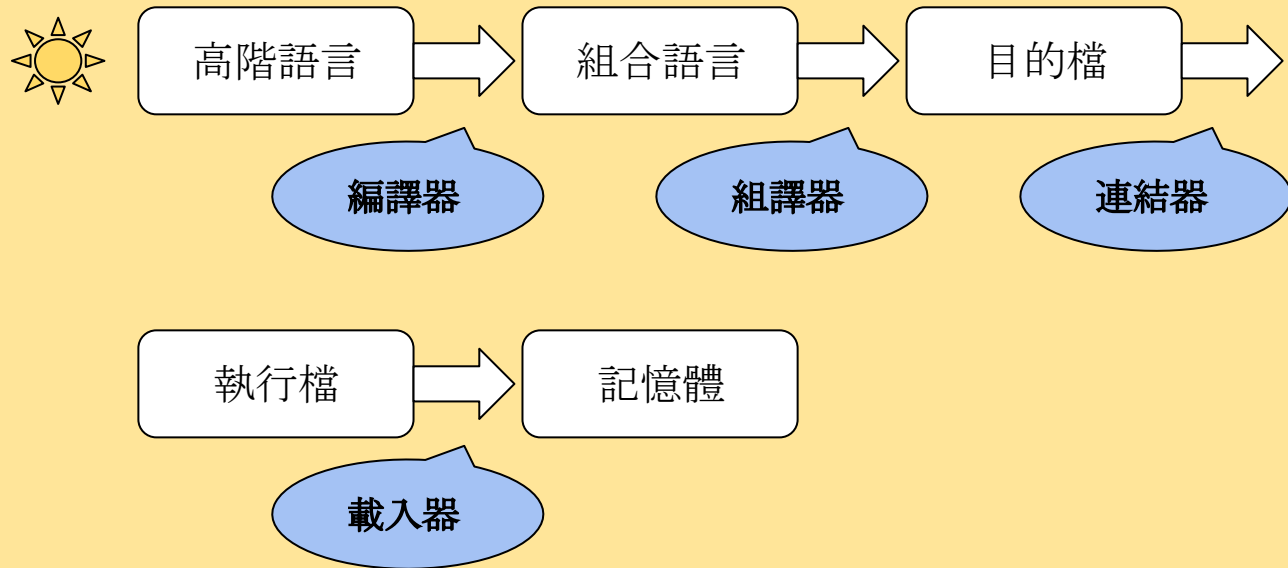
SYSTEM
PROGRAM

Windows 系統程式研究

資工二 楊紓萍

附註: 其實是看老師的課本再多一點自己的想法跟理解而已, 所以算是 閱讀心得吧?

怎麼執行



前置作業



如何在 visual studio code 環境中寫 c 語言？

首先，下載MinGW

附註：老師上課教的是在codeblock裡面找，但是我有現成的MinGW檔案所以就直接用了XD (因為我忘記我的codeblock 存在哪裡)



然後點開裡面的bin 檔案，複製其路徑

名稱	修改日期	類型
bin	2018/9/26 上午 0...	檔案資料夾
include	2018/9/26 上午 0...	檔案資料夾
lib	2018/9/26 上午 0...	檔案資料夾
libexec	2018/9/26 上午 0...	檔案資料夾
mingw32	2018/9/26 上午 0...	檔案資料夾
msys	2018/9/26 上午 0...	檔案資料夾
share	2018/9/26 上午 0...	檔案資料夾
var	2018/9/26 上午 0...	檔案資料夾

前置作業



找到控制台

調整電腦設定



系統及安全性

檢閱您的電腦狀態

使用檔案歷程記錄來儲存檔案的備份副本
備份與還原 (Windows 7)



網路和網際網路

檢視網路狀態及工作



硬體和音效

檢視裝置和印表機

新增裝置

調整常用的行動設定



程式集

解除安裝程式



使用者帳戶

變更帳戶類型



外觀及個人化



時鐘和區域

變更日期、時間或數字格式



輕鬆存取

讓 Windows 建議設定

最佳化視覺顯示

前置作業



系統及安全性>系統>進階系統設定

調整電腦設定



系統及安全性

檢閱您的電腦狀態
使用檔案歷程記錄來儲存檔案的備份副本
備份與還原 (Windows 7)



網路和網際網路

檢視網路狀態及工作



硬體和音效

檢視裝置和印表機
新增裝置
調整常用的行動設定



程式集

解除安裝程式



使用者帳戶

變更帳戶類型



外觀及個人化



時鐘和區域

變更日期、時間或數字格式



輕鬆存取

讓 Windows 建議設定
最佳化視覺顯示

附註:像我的就長這樣



進階系統設定

前置作業



環境變數>在系統變數的格子裡選Path>新增並且把剛剛複製的路徑貼進去

然後重啟 visual studio code 就完成了

打開隨便一個 c 語言程式的 terminal,

輸入gcc (檔案名).c -o (執行檔名稱)就可以執行了

像是: `gcc hello.c -o hello`

hello.c是 c 語言檔案的名稱, hello是執行檔的名稱

然後再輸入./(執行檔名稱)就可以執行了, 像是: `./hello`

CPU0的概念與電腦的基本結構



老師的 CPU0 的概念：

CPU0 = ALU + 暫存器 + 控制單元



根據 Von Neumann Model, 電腦的基本結構分為四種：

記憶體、算術邏輯單元、控制單元、輸入 / 輸出

下一頁分別介紹這四個子系統

電腦的基本結構

記憶體 (memory): 程式執行時儲存資料及程式的地方。

算術邏輯單元 (arithmetic logic unit): 執行算術及邏輯運算的地方。

控制單元 (control unit): 控制著記憶體、算術邏輯單元及輸入 / 輸出子系統的運作。

輸入 / 輸出 (input / output): 輸入子系統接收資料及來自電腦外部的程式, 輸出子系統則是將運算處理的結果傳送至電腦外部。

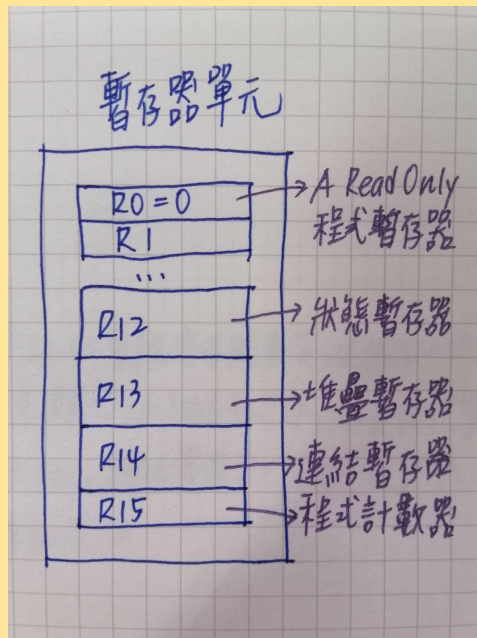
CPU0的概念



老師的 CPU0 的暫存器：

在 CPU0 當中，暫存器 R1 ~ R15 的作用是儲存運算資料，而 CPU0 的暫存器裡面所儲存的二進位資料通常只能被當成整數來運算(整數以二補數的方式來表示)。

附註：直接截圖雖然比較好看，但是就不會像是自己做的，所以裡面的圖片幾乎都是手繪



組合語言

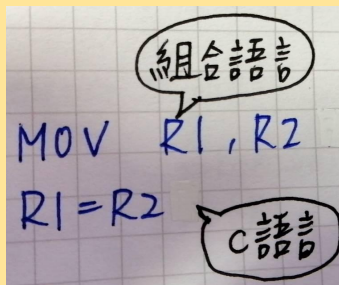


資料移動

如下圖, 就是把 x 指定給 y

```
int main(void){  
    int x = 1;  
    int y;  
    y = x ;  
}
```

在CPU0中的MOV指令可以做到類似的事情, 但是兩個參數都只能是暫存器, 換句話說就是把資料從一個暫存器搬到另一個去, 如果要移動記憶體中的資料, 用 LD 指令跟 ST 指令可以做到相同效果



=



組合語言




模擬條件判斷

就是 c 語言中的 if 判斷句

```
int main(void){  
    if (A > B) C = A;  
    else C = B;  
}
```

組合語言中雖沒有 if 指令, 但可以利用 CMP 與條件跳躍指令模擬 if 的功能這裡使用 CMP 與 JGT 兩個指令, 模擬出 if (A > B) 的功能



if (R1 > R2)
C = A;

else
C = B;

```
LD R1, A // R1=A  
LD R2, B // R2=B  
CMP R1, R2 // if(R1>R2)  
JGT IF // 跳去IF  
ST R2, C // C=R2  
JMP EXIT  
IF: ST R1, C // C=R1  
EXIT: RET
```

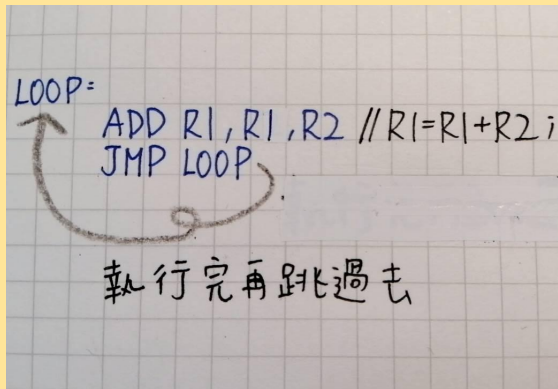
組合語言



模擬迴圈

模擬 c 語言中的 while 判斷句

可以用JMP來達成



=

```
int main(void){  
    while(1){  
        R1 = R1 + R2;  
    }  
}
```

組合語言



模擬迴圈

若不是無窮迴圈，就必須同時使用比較指令與條件式跳躍，讓程式有機會跳出迴圈

```
LD R1, sum //R1=sum
LD R2, i
LDI R3, 10
LDI R4, 1
FOR: CMP R2, R3 //if(i>10)
      JGT EXIT //跳到EXIT
      ADD R1, R2, R1 //sum=sum+i
      ADD R2, R4, R2 //i++
      JMP FOR
EXIT: RET
i: WORD 1
sum: WORD 0
```

=

```
int i, sum=0;

int main(void){
    for(i=1; i<10; i++){
        sum+=1;
    }
}
```

組譯器



將組合語言轉譯成目的碼



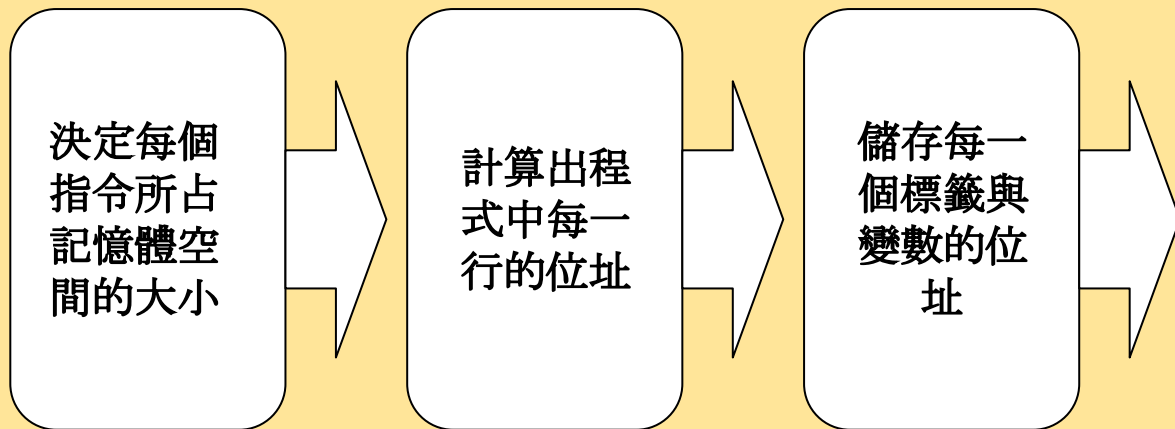
二階段的組譯方式

- 運算元轉換:將指令名稱轉換為機器語言
- 參數轉換:將暫存器轉為代號, 符號轉換成機器位址
- 資料轉換:將原始程式當中的資料常數轉換為內部的機器碼
- 目的碼產生:根據指令格式轉換成目的碼, 輸出到目的檔中

組譯器



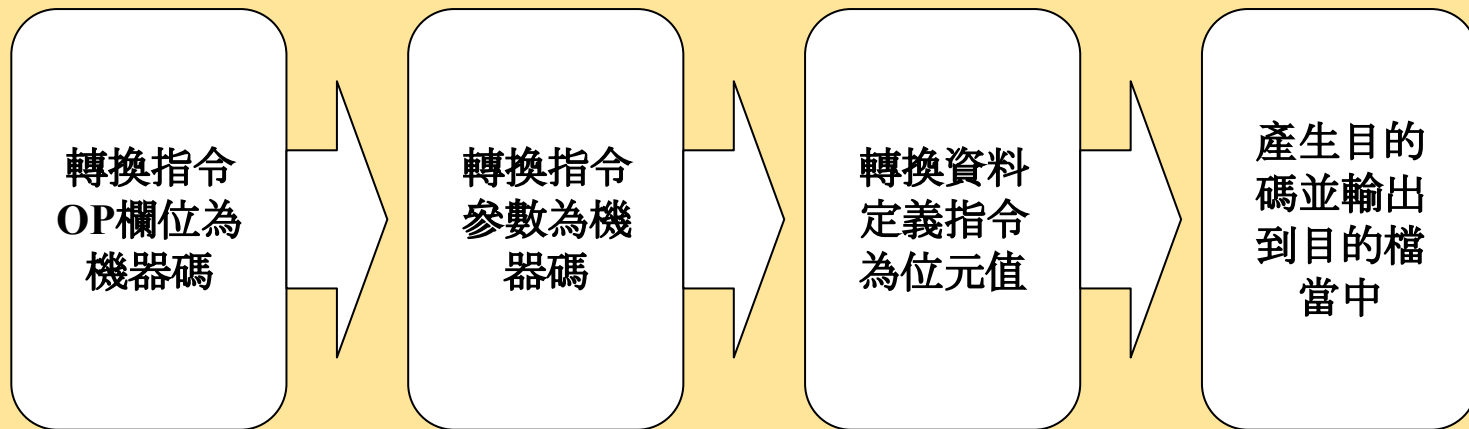
組譯器的第一階段



組譯器



組譯器的第一階段



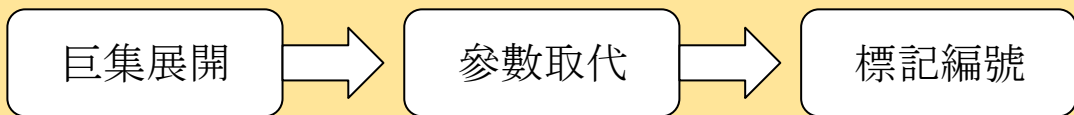
巨集處理器



方便程式撰寫, 避免重複撰寫程式的工具



在程式被編譯前巨集處理器會先將程式當中的巨集展開, 再交給編譯器或組譯器



巨集處理



1.定義巨集 (儲存到記憶體)



2.展開巨集(展開巨集、參數取代、標記編號等)

參考資料



<https://zh.wikipedia.org/wiki/%E7%B3%BB%E7%B5%B1%E7%A8%8B%E5%BC%8F%E8%A8%AD%E8%A8%88>



老師的課本