

# GNU g++ inline assembly (intel x86)

---

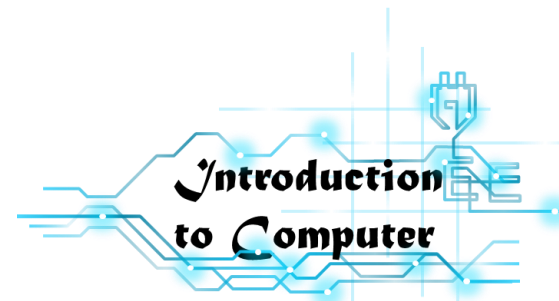
TA: 電子一 謝明倫

yans@media.ee.ntu.edu.tw

2014/03/16

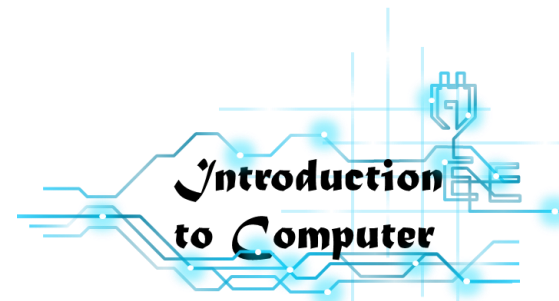


臺灣大學

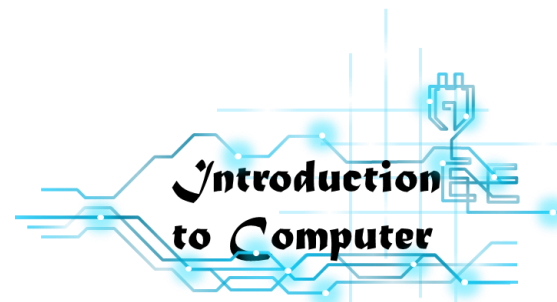


# Why Intel x86 assembly?

- 作業目的: 讓同學可以實際操作、編寫真實的組合語言
  - Intel x86應該是各位電腦上最常見的架構了
  - 對於未來的實用性: 不高
    - 但可以增加大家對於計算機結構的認識
    - 可能會用到組語的人: 處理器、嵌入式、韌體...等領域
- Assembly is highly machine dependent
  - 如果你的電腦不是Intel架構的，就無法執行
  - =>請洽校計中或系計中電腦



# 環境設定/説明



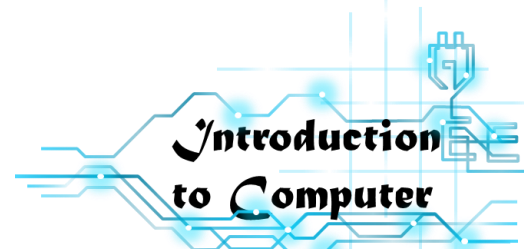
# Intel x86 assembly 兩大風格

- Intel x86 assembly有兩種寫法(意義完全相同): Intel vs AT&T

| 差異    | Intel風格                                                                                                                                                                       | AT&T風格                                                                                                                                                                        |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 參數寫法  | 原樣<br>Reg範例 <code>eax</code><br>Const範例 <code>901</code>                                                                                                                      | Reg前需加%, Const前須加\$<br>Reg範例 <code>%eax</code> (涵義似C++的*取值)<br>Const範例 <code>\$901</code>                                                                                     |
| 參數順序  | Destination first<br>範例 <code>mov eax, ebx</code><br>意義 <code>eax ← ebx</code> ( <code>eax=ebx</code> in C++)                                                                 | Destination last<br>範例 <code>movl %ebx, %eax</code><br>意義 <code>ebx → eax</code> ( <code>eax=ebx</code> in C++)                                                               |
| 資料長度  | 從register名字自動辨認<br>32bits範例 <code>mov eax, ebx</code><br>16bits範例 <code>mov ax, bx</code>                                                                                     | 運算符後面帶資訊(q:8,l:4,w:2,b:1)<br>32bits範例 <code>movl eax, ebx</code><br>16bits範例 <code>movw ax, bx</code>                                                                         |
| 記憶體位置 | 以中括弧表示: <code>size [Reg或Const]</code><br>範例 <code>mov eax, dword [ ebx + 4 ]</code><br>意義 <code>eax ← Mem.at(ebx+4)</code><br>C++: <code>eax = * ( uint32_t*)(ebx+4) )</code> | 以小括弧: <code>shift (base, index, scale)</code><br>範例 <code>movl 4(%ebx), %eax</code><br>意義 <code>Mem.at(ebx+4) → eax</code><br>C++: <code>eax = * ( uint32_t*)(ebx+4) )</code> |
|       | 上面的 <code>dword</code> 相當於AT&T的 <code>mov"l"</code><br><code>byte:1,word:2,dword:4</code>                                                                                     |                                                                                                                                                                               |

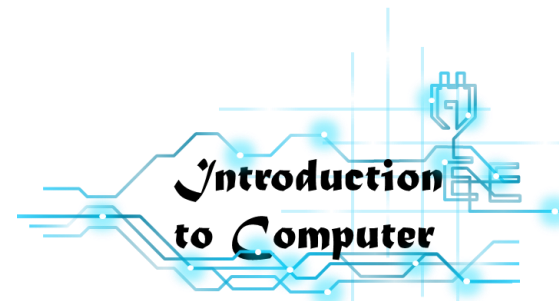
助教OS: destination放前面的寫法在組語裡比較常見

如果只想寫好作業, 不想多作了解, 可以跳過這一頁



# Intel x86 assembly 兩大風格

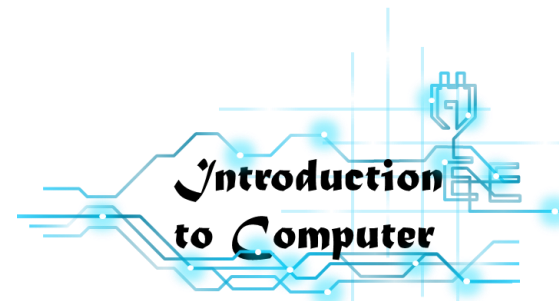
- 與compiler的關係
  - GNU g++ 預設是AT&T style, 但也可以改用Intel style
  - Visual C++只能用Intel style
- 所以本次作業統一採用Intel style
  - Compiler則限用老師上課講過的 GNU g++
- 結論: 作業限制:
  - Intel style Intel x86 inline assembly by GNU g++
    - inline的意義後續會解說



# 整份程式編譯成組語

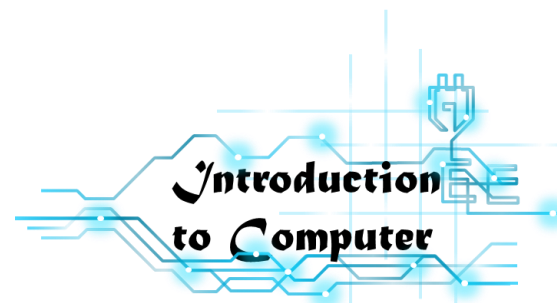
- GNU g++整份程式編譯成組語的方式 (本次作業不需要)
  - AT&T style: `g++ -S x.cpp -o x.asm`
  - Intel style: `g++ -S -masm=intel x.cpp -o x.asm`
  - -S表示輸出assembly
  - -masm=intel表示用intel style
  - -o後面接的是輸出檔檔名(可自己取)

如果只想寫好作業, 不想多作了解, 可以跳過這一頁



# MinGW設定

- MinGW (Minimalist GNU for Windows)
  - 讓windows也能有GNU維護的程式
- 可直接從MinGW官網下載
  - 可以參考這篇介紹
    - <http://shunyuan-chou.blogspot.tw/2010/02/2-mingw-windows-gnu.html>
  - 或者code::blocks有附
    - 位於C:\Program Files (x86)\CodeBlocks\MinGW\bin



# 命令提示字元(cmd)的使用

- 在資料夾裡按shift+右鍵=>開啟命令提示字元
  - 請注意cmd目前工作位置!!
    - 一開始是在開啟的資料夾裡，可以透過cd 某位置來跳換
      - 絕對路徑Ex. cd D:\homework\hw1
      - 相對路徑Ex cd ../hw2 (..)表示往上一層)
    - 可以透過dir指令，顯示目前工作位置裡所有的檔案
    - 可以直接在此輸入 某程式.exe 以執行該程式
- 但g++.exe跟你的程式hw2.cpp可能在不同資料夾裡
  - 需要讓cmd知道分別在哪兒
    - 可以都用絕對位置
    - 也可以設定環境變數=>讓cmd記得以後g++都代表 C:\mingw\bin\g++.exe
    - 或利用set path=c:\MinGW\bin;%path%多一些工作位置
  - 詳細說明可參考
    - <http://shunyuan-chou.blogspot.tw/2010/02/3-mingw-msys.html>

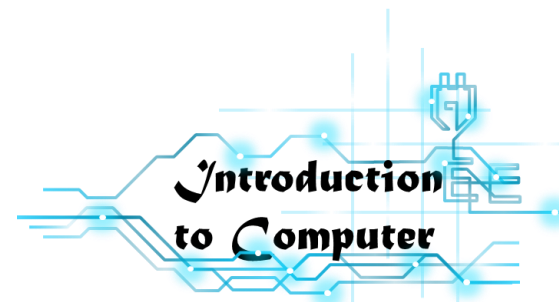
如果只想寫好作業, 不想多作了解, 可以跳過這一頁

助教OS:想要把整份程式編譯成組語才會用到這裡  
作業可以直接用code::blocks或dev C++就好。

Introduction  
to Computer

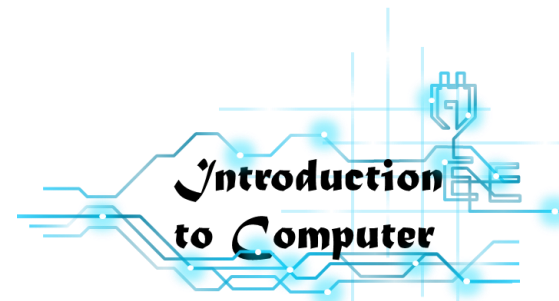


# GNU INLINE ASSEMBLY



# Inline assembly

- 在C++裡面嵌入一部分組合語言
  - 有些地方是C++, 有些地方是組合語言
- Inline assembly不是C++的標準
  - 但大多數compiler都有自己支援的寫法
  - 但不同compiler間彼此不通用



# GNU g++ inline assembly

asm為g++ inline assembly語法

volatile為C++關鍵字=>不要優化改變語法

```
int result, num;  
asm volatile(  
    ".intel_syntax;\n"
```

```
// ===== add your codes here =====  
" mov    eax,    %1    ;\n "  
" mov    %0,    eax    ;\n "  
// =====
```

告知g++改用Intel style (否則預設為AT&T)

```
".att_syntax;\n"
```

回復AT&T style (因為後面要用傳統方式傳遞變數)

```
: "=r" (result)
```

```
: "r"  (num)
```

```
: "%eax", "%ebx", "%ecx", "%edx"
```

```
);
```

主要部份  
一雙引號句為  
一行組合語言,  
內部以;\n結尾

參數傳遞部分

涉及的register

用冒號:把內部區隔成四個部分

```
asm ( assembler template  
    : output operands  
    : input operands  
    : clobbered registers list  
    );
```

# GNU g++ inline assembly

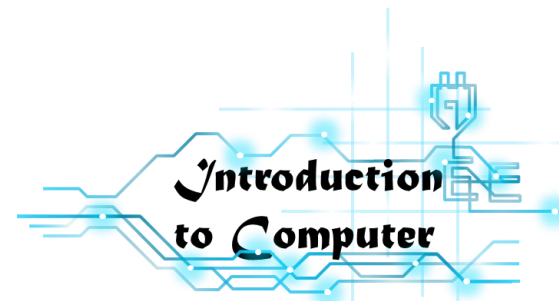
- 參數傳遞部分

- output(會被改變資料者) 要用“=r”開頭
- input要用“r”開頭以做為區別
  - r的意義是: 希望透過register做傳遞
    - 另外還有g,N,a,b,c,d...的不同方式, 有興趣者可以另行研究
- 如果有多個的時候, 用“,”逗號區隔

: “=r”(result) : “r”(num1), “r”(num2)

- 在組語裡面依表列順序以“%0”,“%1”,“%2”...代表之

```
int result, num;
asm volatile(
    ".intel_syntax;\n"
    // ===== add your codes here =====
    " mov    eax,    %1    ;\n "
    " mov    %0,    eax    ;\n "
    // =====
    ".att_syntax;\n"
    : "=r"(result)
    : "r" (num)
    : "%eax", "%ebx", "%ecx", "%edx"
);
```



# GNU g++ inline assembly

- 其他補充

- assembly部分每行以;\n結尾

- 若不然，迴圈會用到的(jmp所需的)label會出問題

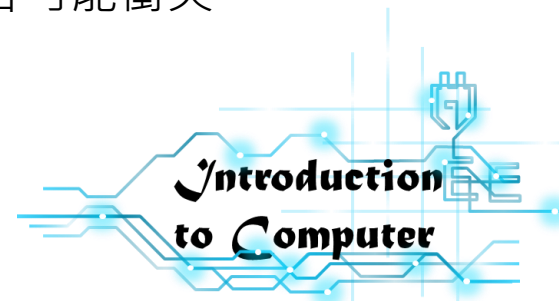
- label語法: “<label>:”

- label名字可以自己取
    - 但要避免使用L0,L1...等等，因為這些是compiler習慣使用的
    - 例如出現xxx已經被宣告過的錯誤

```
"CHECK: cmp ebx, 0      ;\n"  je      ISOK          ;\n"xor  edx,edx           ;\n" idiv  ebx              ;\n"mov  eax,ebx           ;\n"mov  ebx,edx           ;\n"jmp  CHECK             ;\n"ISOK: mov  %0,eax       ;\n"
```

- 涉及register的表列請據實填寫

- 如果沒寫這一個, 內部跟外部的register所記憶的內容可能衝突
    - =>造成segmentation fault之類的 (尤其在-O1,-O2時)



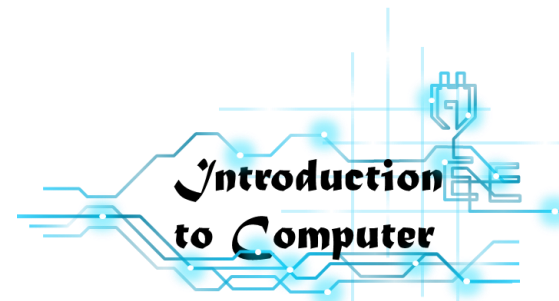
# Visual C++ inline assembly

- 語法是\_\_asm{ }
  - 在大括弧裡面寫assembly
  - 不用雙引號包, 不用;\n換行
  - 變數傳遞直接寫C++內的變數名字
- 只能用Intel style

```
__asm {  
    push ebp  
    mov ebp, esp  
    sub esp, __LOCAL_SIZE  
}
```

- 本頁純粹補充...本次作業請勿使用此寫法

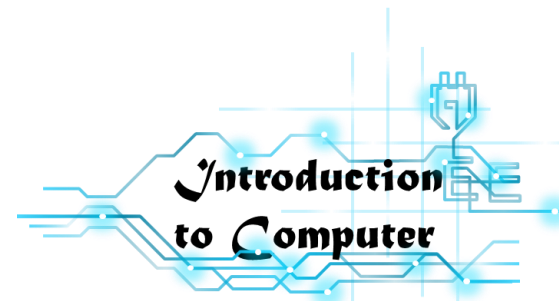
如果只想寫好作業, 不想多  
作了解, 可以跳過這一頁



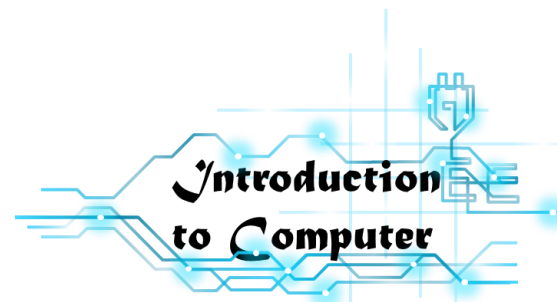
# 如果想寫可以到處適用的code該怎麼辦?

- 每個compiler都會利用#define來訂專於自己的東西
  - 可以利用#ifdef來區隔
- GNU g++:
  - #ifdef \_\_GNUC\_\_
- Visual C++:
  - #ifdef \_MSC\_VER

如果只想寫好作業, 不想多作了解, 可以跳過這一頁



# 組語內容





# Register 名稱

強烈建議只使用這些

General-purpose:  
一般用途的

General-purpose Registers

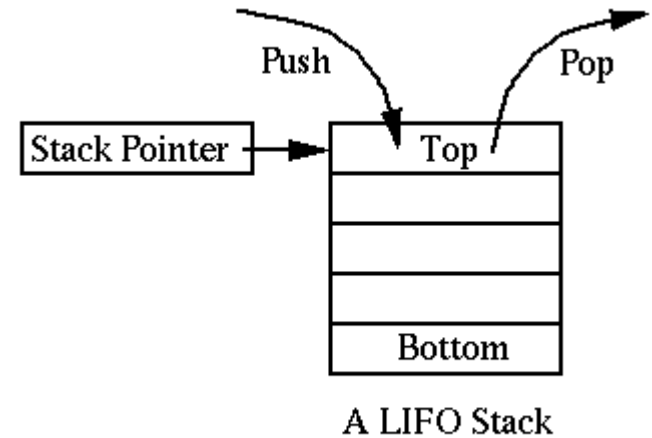
|                               |           |         |        |
|-------------------------------|-----------|---------|--------|
|                               |           | 16 bits |        |
|                               |           | 8 bits  | 8 bits |
| <b>EAX</b>                    | <b>AX</b> | AH      | AL     |
| <b>EBX</b>                    | <b>BX</b> | BH      | BL     |
| <b>ECX</b>                    | <b>CX</b> | CH      | CL     |
| <b>EDX</b>                    | <b>DX</b> | DH      | DL     |
| <b>ESI</b>                    |           |         |        |
| <b>EDI</b>                    |           |         |        |
| <b>ESP</b><br>(stack pointer) |           |         |        |
| <b>EBP</b><br>(base pointer)  |           |         |        |
|                               |           | 32 bits |        |

# 常用語法偶舉 – 運算用

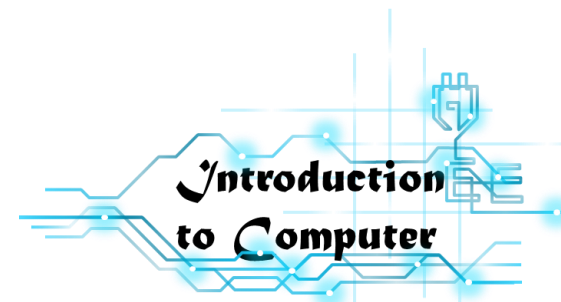
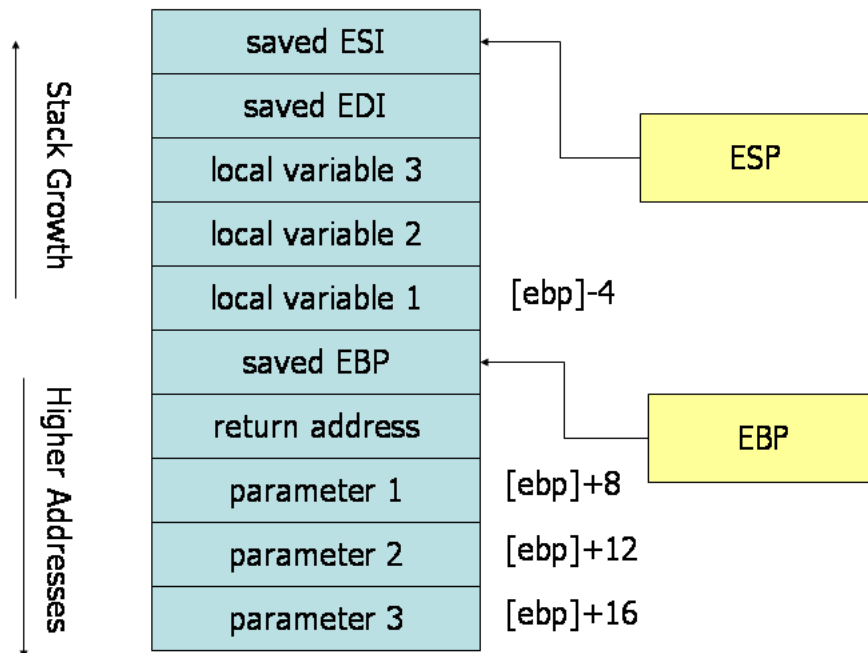
| 語法                           | 功能                                                  | 說明                                                |
|------------------------------|-----------------------------------------------------|---------------------------------------------------|
| <b>mov</b> <t1>, <t2>        | <t1> = <t2>                                         |                                                   |
| <b>add</b> <t1>, <t2>        | <t1> += <t2>                                        | sub同理                                             |
| <b>inc</b> <t1>              | ++<t1>                                              | dec同理 (--<t1>)                                    |
| <b>imul</b> <t1>, <t2>       | <t1> *= <t2>                                        |                                                   |
| <b>imul</b> <t1>, <t2>, <t3> | <t1> = <t2> * <t3>                                  | <t3>只能是constant                                   |
| <b>idiv</b> <t1>             | num = [edx eax]<br>eax = num/<t1><br>edx = num%<t1> | 把edx, eax組合成一64bits的數字，對<t1>做整數除法<br>商放eax, 餘放edx |
| <b>shl</b> <t1>, <t2>        | <t1> <<= <t2>                                       | shr同理                                             |
| <b>and</b> <t1>, <t2>        | <t1> &= <t2>                                        | or, xor同理                                         |

# 常用語法偶舉 – stack

| 語法               | 功能                                        |
|------------------|-------------------------------------------|
| <b>push</b> <t1> | $esp -= 4$<br>$*esp = \langle t1 \rangle$ |
| <b>pop</b> <t1>  | $\langle t1 \rangle = *esp$<br>$esp += 4$ |



x86 stack往address小的方向長



# 常用語法偶舉 – control flow

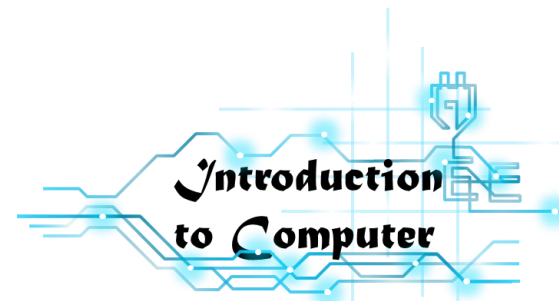
| 語法                    | 功能                       | 說明            |
|-----------------------|--------------------------|---------------|
| <b>jmp</b> <label>    | goto <label>             |               |
| <b>cmp</b> <t1>, <t2> | 對<t1> <t2>做比較            | 以下的語法皆須與cmp搭配 |
| <b>jle</b> <label>    | if (<t1>==<t2>) goto...  |               |
| <b>jne</b> <label>    | if (<t1>!=<t2>) goto...  |               |
| <b>jg</b> <label>     | if (<t1> > <t2>) goto... |               |
| <b>jge</b> <label>    | if (<t1> >= <t2>) goto.. |               |
| <b>jl</b> <label>     | if (<t1> < <t2>) goto... |               |
| <b>jle</b> <label>    | if (<t1> <= <t2>) goto.. |               |

## *Example*

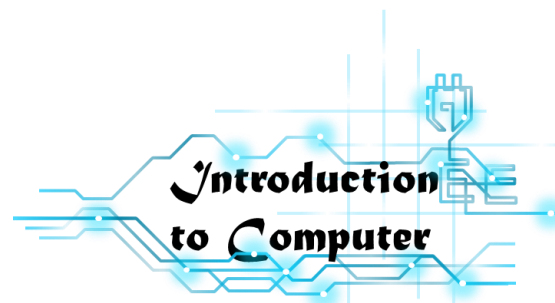
```
cmp eax, ebx
```

```
jle done
```

```
=> if (eax<=ebx) goto done
```



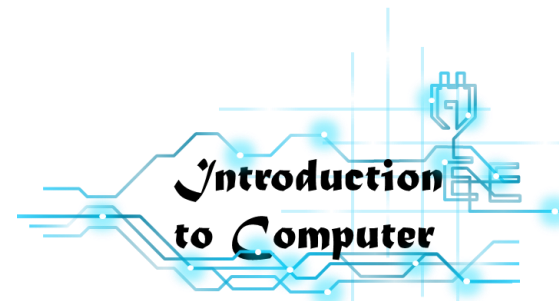
其他



## Hint: how to beat -O2

- `idiv`速度慢
  - 除以10相當於 乘以(某二進位數)再>>一定量
    - 不過會有小小的誤差
- 利用我們只會給0~9999999

如果只想寫好作業, 不想多作了解, 可以跳過這一頁



# 參考資料

- 清楚的簡易指令教學(intel syntax)
  - <http://www.cs.virginia.edu/~evans/cs216/guides/x86.html>
- 可以看asm( ::: ) 放的東西+GCC上intel-AT&T syntax互換
  - [http://wiki.osdev.org/Inline\\_Assembly](http://wiki.osdev.org/Inline_Assembly)
- 補充上面，最後要加一行".att\_syntax;"
  - <http://stackoverflow.com/questions/5397677/gcc-intel-syntax-inline-assembly>
- 有一個msvc inline assembly的範例
  - <http://msdn.microsoft.com/en-us/library/5f7adz6y.aspx>
- 一些cross platform的#ifdef用法
  - <http://stackoverflow.com/questions/8082590/how-do-you-use-the-preprocessor-for-making-a-cross-platform-library>

