歡迎來到數字的世界！為了能夠開始我們的教學，必須得先來聊一聊數字的基本概念，廢話不多說，從最簡單的部分來開始吧！

2.1 數字的起源與簡介

數字這玩意兒，打從人類自原始時代的數羊開始，到現在成為股市上下起伏的指標，數字這玩意兒就一直在咱們的日常生活中出現，並且，我們還一直不斷地使用它，不論你的目的是什麼。

在不知道什麼時代的時代，也不知道是哪一個民族，他們使用一種叫做十進位的方式來運用數字，十進位的觀念我相信大家都懂，就字面上的意思來說，就是逢十就進位。而這種數字的用法就跟你在日常生活裡頭與菜市場的阿嬤買東西殺價時所使用的數字 1、2、3、4、5 是一樣的，使用的方法雖然簡單，但我們還是先來弄清楚一下十進位好了。

十進位數字：0、1、2、3、4、5、6、7、8、9，就在我們從數字 9 打算要數到數字 10 的時候，個位數的 9 就要進位，並且在十位數的地方添上 1，此時 9 則是重新回到 0 的地方去，於是進位後的數字就成為了 10。

當然啦！如果你要繼續從數字 10 的地方來開始加碼也是可以的，接下來的數字就會是 11、12、13、14、15、16、17、18、19，而就在我們從數字 19 算到數字 20 的時候，19 當中的個位數 9 就要進位，並且在十進位數字 1 的地方把數字改寫為 2，此時 9 則又是重新回到數字 0 的地方去，於是進位後的數字就成為了 20。

我們對數字的算法，就是在這樣子的情況之下，透過不斷地進位以及不斷地循環，從而構成了我們日常生活裡頭的數字系統，而這數字系統一直被我們所沿用，直到現在。

十進位的做法一直很受我們人類的歡迎，但十進位的定義和做法終究只是人類自己所定義出來的一套系統。既然是定義出來的，那我也可以說我也要自己設計一套屬於自己的進位模式，然後讓全世界的人類也能夠使用！

碁峯 www.gotop.com.tw

之後我們再來把十六進位數字稍微地延伸下去，此時我們會得到什麼結果呢？

十六進位數字：0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F、10、11…

哈！這又是什麼鬼，為什麼英文字母 F 的後面竟然會出現 10 ？那意思是說，當我們在對十六進位數字從 0 開始計數，一直算到 F（或 15）之時，這時候 F 後面竟然接的是「十進位」數字 10 ？（請注意我把十進位這三個字括號了起來）

當然不是！為了澄清這個誤會，就讓咱們繼續來玩個比較法，當比較完之後，你就會知道 F 後面所接的數字 10 到底是什麼意思了。

十進位數字：0、1、2、3、4、5、6、7、8、9、10、11、12、13、14、15、16、17

十六進位數字：0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F、10、11

我們又比較了十進位與十六進位這兩排數字，從中我們除了上面的結論 10=A、11=B、12=C、13=D、14=E、15=F 之外，我們更得出了以下的結論：

十進位數字 16 等同於 十六進位數字 10

十進位數字 17 等同於 十六進位數字 11

也就是說，當我們在計算十六進位數字 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F、10、11 之時，這之中的數字 10 並不是十進位數字當中的數字 10，而是十進位數字當中的 16，所以這時候 F 後面所接的數字 10 是「十六進位」的數字 10，而不是「十進位數字」當中的數字 10（請注意我把十六進位這三個字括號了起來）

講白一點就是：

（十進位數字）16=（十六進位數字）10

（十進位數字）17=（十六進位數字）11

碁峰 www.gotop.com.tw

本章將告訴各位逐步執行的基本原理與技巧，為了簡便起見，我們謹以 main 函數為例來做説明。

4.1 事前準備

Step 1 請打開 Visual Studio，並輸入如下的程式之後，在 main 函數的地方設置一個斷點，情況如下所示：

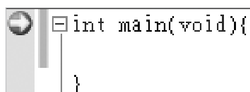


圖 4.1.1

Step 2 緊接著，進入反組譯的部份：

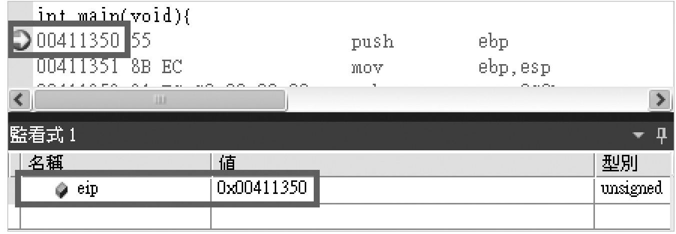
```
int main(void){
00411350 55                   push       ebp
00411351 8B EC                mov        ebp,esp
00411353 81 EC C0 00 00 00    sub        esp,0C0h
00411359 53                   push       ebx
0041135A 56                   push       esi
0041135B 57                   push       edi
0041135C 8D BD 40 FF FF FF    lea        edi,[ebp-0C0h]
00411362 B9 30 00 00 00       mov        ecx,30h
00411367 B8 CC CC CC CC       mov        eax,0CCCCCCCCh
0041136C F3 AB                rep stos   dword ptr es:[edi]

}
0041136E 33 C0                xor        eax,eax
00411370 5F                   pop        edi
00411371 5E                   pop        esi
00411372 5B                   pop        ebx
00411373 8B E5                mov        esp,ebp
00411375 5D                   pop        ebp
00411376 C3                   ret
```

圖 4.1.2

以前在介紹組合語言之時就曾經説過：「只要 ip（或 32 位元的 eip）的數值指到哪，程式就從那來開始執行」，由於我們把斷點斷在 main 函數上，因此，此時程式的運作起點當然也就是 eip 所指向的 main 函數的記憶體位址。

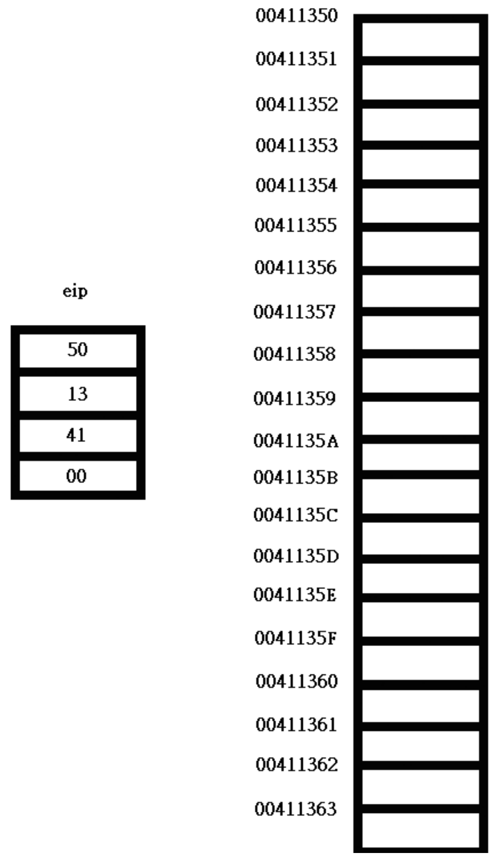
碁峯 www.gotop.com.tw

注意，這時候的 eip 其實就是黃色箭頭所對應到的記憶體位址，請比較下面框起來的部份：



⬆ 圖 4.1.5

因此數值 0x00411350 不但是 main 函數的入口位址，同時也是 eip 目前所指向的記憶體位址，因此，我們可以把 eip 填上去，圖示如下所示：



⬆ 圖 4.1.6

碁峰 www.gotop.com.tw

4.2 逐步執行

好，目前已經萬事具備，現在就讓我們直接地來看看程式經過一段一段的「逐步執行」之後，eip 與程式碼是怎樣地來做變化，首先，按下鍵盤上的 F11（按下 F10 也可以，F10 的話則是會在遇到函數的時候直接執行函數的運行結果，而不進入函數裡頭去）。

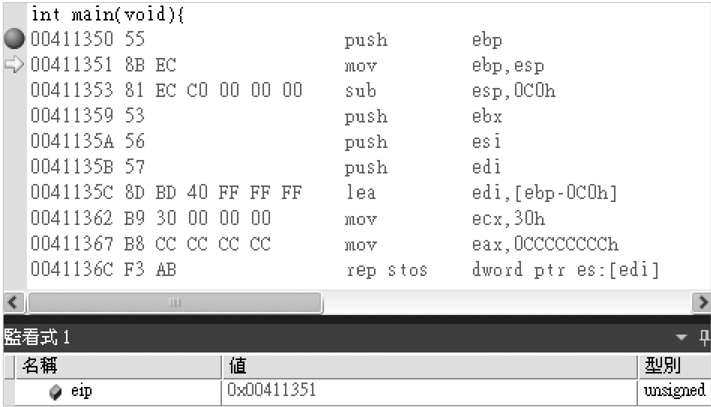


圖 4.2.1

這時候原本指向記憶體位址 0x00411350 的黃色箭頭已經往下一行移動，而且可以看到下面框框裡頭的 eip 暫存器已經變成了記憶體位址 0x00411351，Visual Studio 為了方便我們來辨識，還特意把變化後的數值用紅色來做表示，因此，圖 4.1.8 的變化如下：

碁峰 www.gotop.com.tw

5.3 使用 IDA Pro 分析 HelloWorld

首先，讓我們來看 HelloWorld，HelloWorld 的 C 語言程式碼如下所示：

```
#include<stdio.h>
#include<stdlib.h>

int main(void){

    printf("HelloWorld\n");

    system("pause");
    return 0;

}
```

⬆ 圖 5.3.1

當我們在 IDA Pro 裡頭把 HelloWorld 當中的堆疊去掉之後，剩下就只會看到這兩行程式碼：

```
.text:004113B0                push    offset aHelloworld ; "HelloWorld\n"
.text:004113B5                call    ds:__imp__printf
```

在位址 004113B0 裡頭，字符串 Hello World 的記憶體位址被當作參數，然後被 push 進堆疊當中，之後在位址 004113B5 的地方則是使用 call 指令來調用 C 語言當中的 printf 函數，最後顯示出 Hello World 這一串文字。

以上，就是使用 IDA Pro 來觀察 HelloWorld 在底層裡頭的運作過程，接下來，讓我們來看看加法運算。

碁峯 www.gotop.com.tw

5.4 使用 IDA Pro 分析加法運算

加法運算的 C 語言程式碼如右圖所示：

```
#include<stdio.h>
#include<stdlib.h>

int main(void){

    int number1=1;
    int number2=2;

    printf("The sum is %d\n",number1+number2);

    system("pause");
    return 0;

}
```

⬆ 圖 5.4.1

在 IDA Pro 當中去掉堆疊之後的反組譯程式碼如下所示：

```
.text:00411390 var_14          = dword ptr -14h
.text:00411390 var_8           = dword ptr -8

.text:004113AE                mov     [ebp+var_8], 1
.text:004113B5                mov     [ebp+var_14], 2
.text:004113BC                mov     eax, [ebp+var_8]
.text:004113BF                add     eax, [ebp+var_14]
.text:004113C2                mov     esi, esp
.text:004113C4                push    eax
.text:004113C5                push    offset aTheSumIsD ; "The sum is %d\n"
.text:004113CA                call    ds:__imp__printf
```

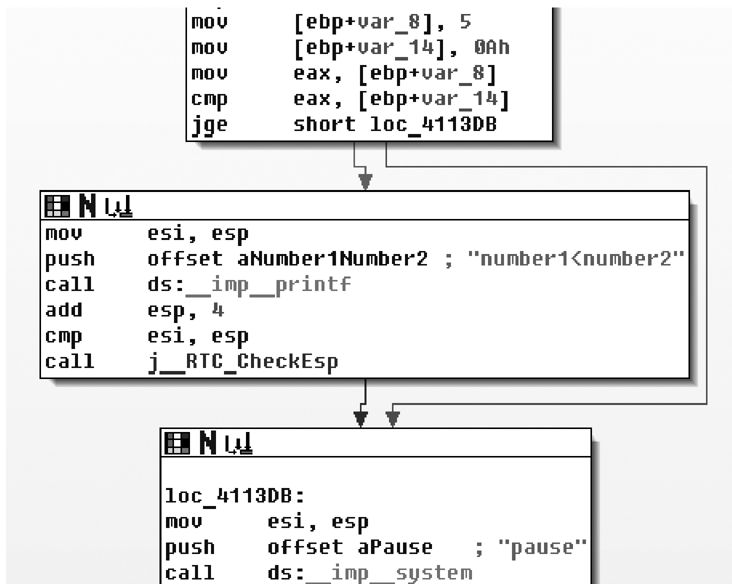
記憶體位址 00411390~004113B5 的部份在之前的教學當中已經都講過了，現在就讓我們從位址 004113BC 的地方來開始看起，位址 004113BC 的意思是說把位址 ebp+var_8 裡頭的十進位數字 1（十六進位數字 1）放進暫存器 eax 當中，然後位址 004113BF 的地方告訴我們，把位址 ebp+var_14 當中的十進位數字 2（十六進位數字 2）與暫存器 eax 當中的十進位數字 1 一起來做相加，最後把相加後的結果放進暫存器 eax 當中，剩下的 push 部分我就不說了，讀者可自行理解。

往後為了我們的教學，一律以 IDA Pro 為主，Visual Studio 為輔。

碁峰 www.gotop.com.tw

在上面的那兩張圖當中，不知道各位讀者們發現了些什麼？我想，各位聰明的讀者們應該都已經發現到了流程圖的出現對吧？沒錯，就是流程圖，流程圖這種東西我想大家應該都懂，不過要是各位讀者不懂或者是對流程圖沒什麼概念的話也沒關係，讓我們一起來分析分析下面這張圖之後，就算沒有流程圖的預備知識也一定會懂得流程圖，以及流程圖在這道程式當中扮演著什麼樣的關鍵角色。

下圖是整個程式的核心關鍵：



⬆ 圖 6.1.4

我們來研究研究上圖傳達給我們的意義，可以看到關鍵反組譯程式碼如下所示：

反組譯程式碼	對應程式碼	解說
`mov [ebp+var_8], 5`	`int number1=5;`	把十進位數字 5 放進位址 ebp+var_8（或 number1）中
`mov [ebp+var_14], 0Ah`	`int number2=10;`	把十進位數字 10 放進位址 ebp+var_14（或 number2）中

碁峯 www.gotop.com.tw

執行結果如下所示：

⬆ 圖 6.2.1

在 Ida Pro 裡頭，這道程式的內容如下所示（我們取關鍵的地方來研究即可）：

⬆ 圖 6.2.2 反組譯程式碼流程圖

反組譯程式碼	對應程式碼	解說
jge short loc_4113DD	if(number1<number2)	意思是比較 15 和 10，如果 15 大於等於 10 的話則跳到位址 loc_4113DD 的地方去，也就是走右邊的綠色線去執行 "number1>number2"，否則，就走左邊的紅色線去執行 number1<number2。 PS：jge 標記－有號數比較時，其結果數值大於等於時跳到標記。

碁峰 www.gotop.com.tw

也就是：

```
call    j_function
```

注意！這裡調用了組合語言裡頭的指令 call 指令來調用函數 function，但現在問題來了，已知函數 function 出現在流程圖當中，但卻看不到函數 function 的基本結構，這到底是怎麼回事？理由很簡單，因為我們得從記憶體配置程式的地方來觀察到底函數 function 是出現在哪裡以及其內部結構。

首先進入記憶體配置程式的地方，在流程圖的空白處隨便按下滑鼠右鍵，如下圖所示：

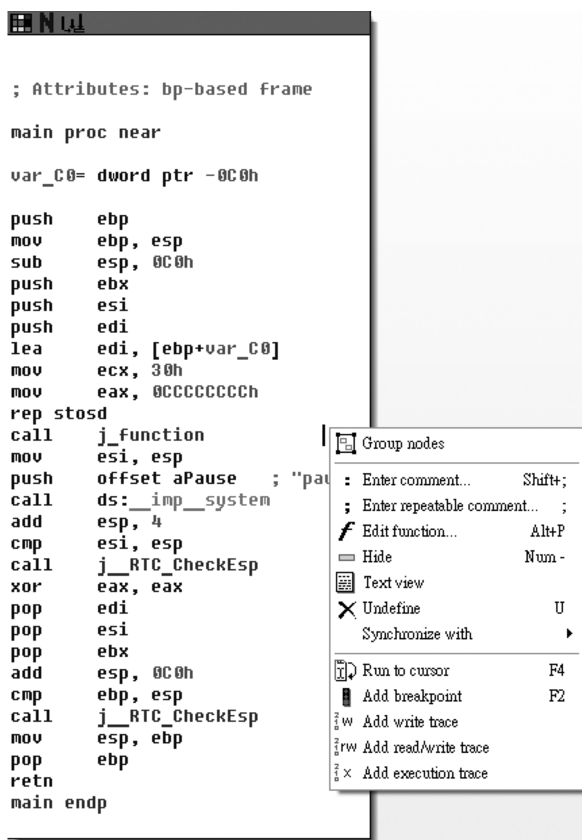


圖 8.1.4 按下滑鼠右鍵

碁峰 www.gotop.com.tw

關鍵的地方就在於下面的程式碼：

反組譯程式碼	對應程式碼	解說
mov [ebp+var_1C], 0	int Array[]={0,1,2,3,4};	把數字 0 放進陣列中
mov [ebp+var_18], 1		把數字 1 放進陣列中
mov [ebp+var_14], 2		把數字 2 放進陣列中
mov [ebp+var_10], 3		把數字 3 放進陣列中
mov [ebp+var_C], 4		把數字 4 放進陣列中
mov [ebp+var_28], 0	for(**int i=0;**i<=4;i++) 注意粗體字的部份	把數字 0 放進變數 i 中
jmp short loc_4113ED		無條件跳躍到位址 loc_4113ED 的地方去

無條件跳躍到位址 loc_4113ED 的地方去：

⬆ 圖 9.2.4 反組譯程式碼流程圖

碁峰 www.gotop.com.tw

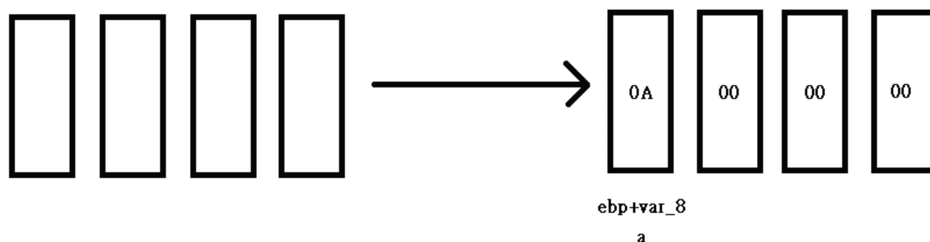
```
push    offset aTheValueOfBp_1 ; "The Value of ***&bPtr is %d\n"
call    ds:__imp__printf
add     esp, 8
cmp     esi, esp
call    j__RTC_CheckEsp
mov     eax, [ebp+var_2C]
mov     ecx, [eax]
mov     edx, [ecx]
mov     esi, esp
mov     eax, [edx]
push    eax
push    offset aTheValueOfCptr ; "The Value of ***cPtr is %d\n\n"
call    ds:__imp__printf
```

⬆ 圖 10.2.5 反組譯流程圖 4

讓我們一步一步地來拆解這道程式：

反組譯程式碼	對應程式碼	解說
mov [ebp+var_8], 0Ah	int a=10;	把數字 10 放進變數 a 的位址 ebp+var_8 中

如下圖：



⬆ 圖 10.2.6 把數字 10 放進變數 a 的位址 ebp+var_8 中

反組譯程式碼	對應程式碼	解說
lea eax, [ebp+var_8]	aPtr=&a;	把變數 a 的位址 ebp+var_8 放進暫存器 eax 中

碁峯 www.gotop.com.tw