第 1 章習題

1.1　請說明何謂系統軟體？

參考解答：

系統軟體是相對於應用軟體而言的，凡是專門設計給程式設計師使用的軟體，就被稱為系統軟體，而設計給一般大眾使用的軟體，則稱為應用軟體

1.2　請列出你所知道的系統軟體。

參考解答：

組譯器、載入器、連結器、巨集處理器、編譯器、直譯器、虛擬機

gcc javac, java, linux, …

1.3　請說明系統軟體與系統程式兩者有何區別。

參考解答：

系統程式：是與作業系統或硬體相關的程式設計技術，指的是程式設計的技術。

系統軟體：是程式設計師開發程式時所使用的軟體，指的是程式設計時使用的軟體。

1.4　請說明組合語言在系統軟體學習上的角色。

參考解答：

學習組合語言有助於理解電腦硬體的設計原理，並且對系統軟體的理解有相當大的幫助，學會組合語言後才能理解組譯器、編譯器、虛擬機、作業系統等系統軟體的設計原理。

1.5　請說明 C 語言在系統程式上的用途。

參考解答：

C 語言在設計上考慮了許多系統程式的因素，包含與組合語言的銜接、硬體的控制、與執行速度等等。UNIX 的成功也帶動了 C 語言的進一步發展。在今天, C 語言是系統程式、嵌入式系統與作業系統等領域的首選語言。學習 C 語言有助於理解系統程式的概念，像是記憶體映射輸出入，就是一種 C 語言所擅長的系統程式技術。

1.6　請列出您所經常使用的程式語言，並說明其相關的系統軟體之用法。

參考解答：

我常用的語言有 C, Java, C#，相關的系統軟體如下。

C: Dev C++ 整合開發環境, GNU gcc 編譯器

Java : javac 編譯器, java 虛擬機 JVM

C#: Visual Studio 整合開發環境，其中的編譯器是 csc。

1.7 請從網路下載 Dev C++ 軟體，並參照附錄 D 的說明，安裝並使用 Dev C++ 撰寫 C 語言程式，並學習該軟體的用法。

參考解答：請直接參考附錄 D 進行操作。

1.8 請找出 Dev C++ 當中的 GNU 工具，並在設定好 PATH 環境變數後，試用 gcc 指令編譯任意一個 C 語言程式 (設定方法請參考附錄 D)。

參考解答：請直接參考附錄 D 進行操作。

第 2 章習題

2.1　請畫出馮紐曼電腦的基本架構圖。
參考解答：

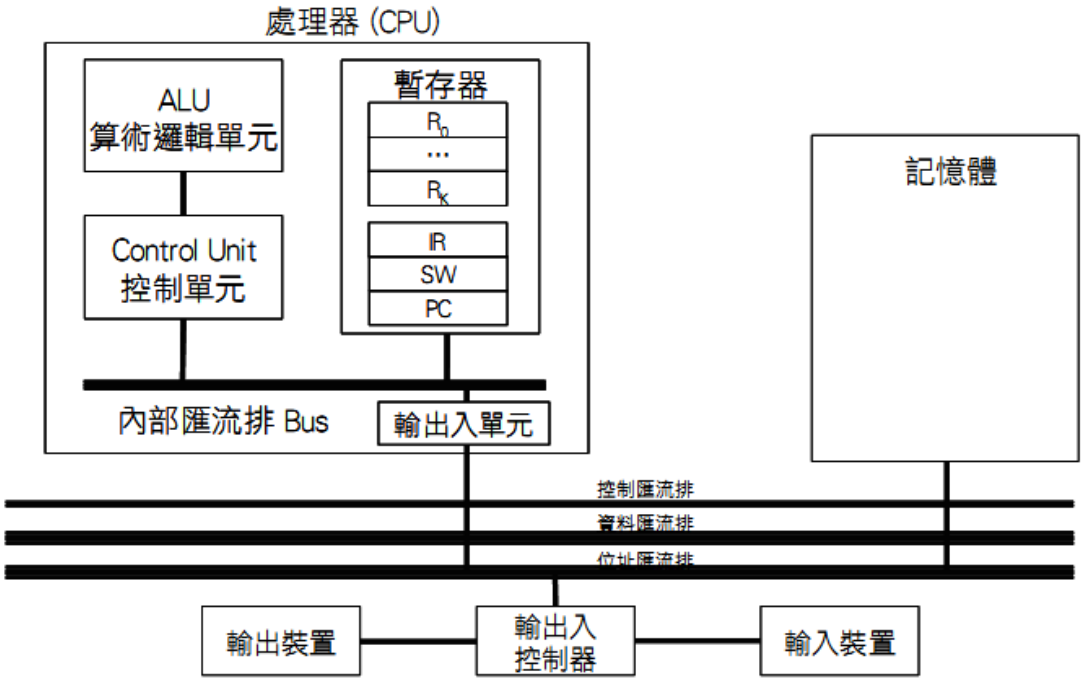


圖 2.2　馮紐曼電腦的結構

2.2　請說明暫存器在電腦中的用途？
參考解答：
暫存器通常位於電腦的 CPU 當中，CPU 會根據指令存取暫存器，並且將計算結果儲存在暫存器中，由於 CPU 存取暫存器的速度比記憶體快，因此通常會盡可能將目前最常用的記憶體變數存入暫存器，直到暫存器不足或者計算動作完成後在存回記憶體中。

2.3　請說明控制單元在電腦中的用途？
參考解答：
控制單元 (Control Unit) 是 CPU 的指揮中心，通常包含許多多工器 (Multiplixer)，這些多工器會決定資料的流向。舉例而言，在指令提取階段中，控制單元會將程式計數器 PC 的值，上傳道位址匯流排中，然後設定控制線為讀取，讓記憶體將指令傳上資料匯流排，當控制單元偵測到資料回傳訊息後，就會將回傳的指令碼放入到指令暫存器中，以作為解碼階段的解碼基礎。

2.4 請說明 ALU 在電腦中的用途？

參考解答：

ALU 是 CPU 中的算術單元 (Arithmetic Logic Unit)，其功能是進行加減乘除等運算，ALU 的硬體電路通常是一組加法器，有時也會包含乘法器。

控制單元會控制 ALU 的運算功能，以便再需要進行計算時，讓兩個輸入資料進入 ALU 以取得運算結果，這些運算結果通常會在控制單元的導引下回流到暫存器當中。

2.5 請問 CPU0 有哪些暫存器？並說明這些暫存器的功能。

參考解答：

CPU0 包含 15 個可存取暫存器 R1-R15, 以及一個唯讀的常數暫存器 R0。R0 的值永遠都是常數零, 由於常數零在電腦上被使用的頻率極高, 因此 CPU0 特別為常數零分配一個暫存器。

在可存取的暫存器 R1-R15 當中, R12、R13、R14 與 R15 具有特殊用途。R12 是狀態暫存器 (Status Word, SW), R13 則是堆疊暫存器 (Stack Pointer Register, SP), R14 代表連結暫存器 (Link Register, LR), 而 R15 則是程式計數器 (Program Counter, PC)。

2.6 請問 CPU0 的指令可分為哪幾類？並且以範例說明每一類指令的功能？

參考解答：

在表格 2.1 中的 CPU0 指令類型中, 包含了『載入儲存』、『運算指令』、『跳躍指令』、『堆疊指令』等四大類型的指令，其功能如下。

載入儲存：存取記憶體

運算指令：進行加減乘除等運算。

跳躍指令：改變程式計數器 (PC)，讓程式跳到目標位址繼續執行。

堆疊指令：存取堆疊，像是 PUSH 或 POP 等。

2.7 請問 CPU0 當中有哪些定址方式, 並以範例加以說明？

參考解答：

CPU0 的定址模式可分為三種, 包含『立即載入』、『相對定址』與『索引定址』等。在 CPU0 當中, 採用立即載入的指令只有 LDI 一個, 相對定址的指令有 LD、ST、LDB、STB 等, 索引定址的指令有 LDR、STR、LBR、SBR 等。

範例：

```
LDI R1, 100;  將常數 100 載入到 R1 暫存器中。
```

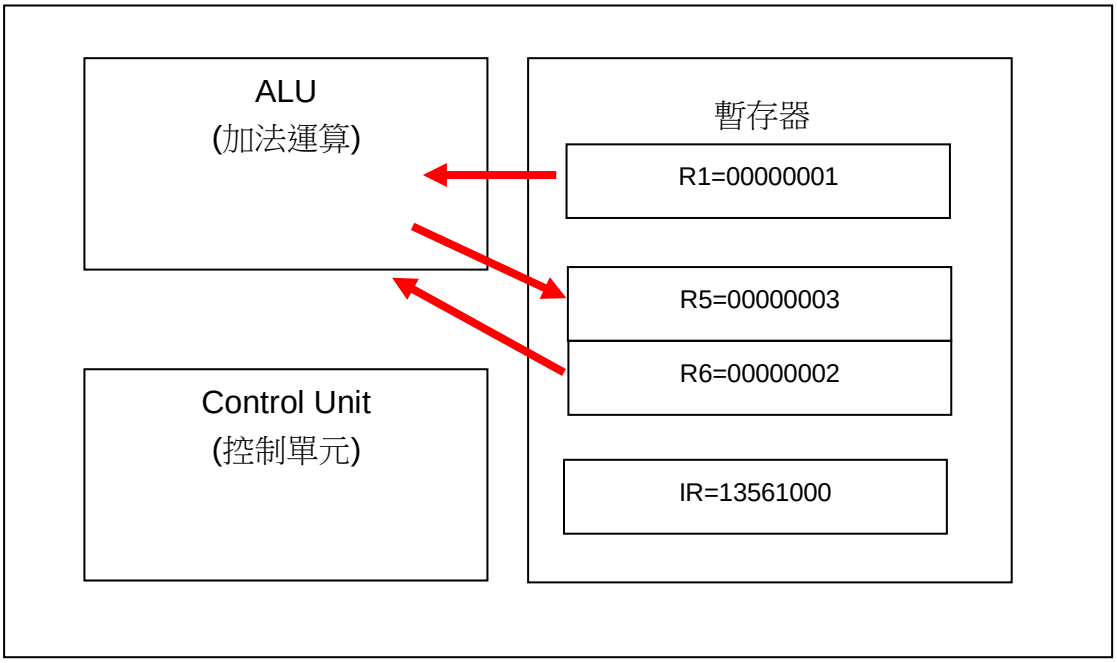
LD R1, [100]; 將記憶體位址 100 中的內容載入到 R1 暫存器中。
LDR R1, [R2+R3]; 將記憶體位址 R2+R3 中的內容載入到 R1 暫存器中。

2.8 請說明 CPU0 程式的執行原理，並說明指令暫存器與程式計數器在程式執行時的作用？
參考解答：
CPU0 的每個指令大小都是 4 bytes，每當一個指令擷取後程式計數器 PC 就會被加上 4 ，因此指令會按照順序執行。在指令擷取階段，CPU0 會將 PC 的內容放到位址匯流排上，然後設定控制線為記憶體讀取狀態，當記憶體完成讀取後，會將指令傳到匯流排上，此時 CPU0 就會取得指令碼放入指令暫存器 IR 中。

接著會進入執行階段，CPU0 根據 IR 的內容決定 ALU 的輸出與輸入，並且設定 ALU 的運算模式，於是資料開始流入 ALU 進行計算，計算結果會流出到指定的暫存器中，如此就執行完一個指令了。

2.9 請畫出 ADD R5, R6, R1 指令的資料流向圖，並說明該指令的運作方法。
參考解答：

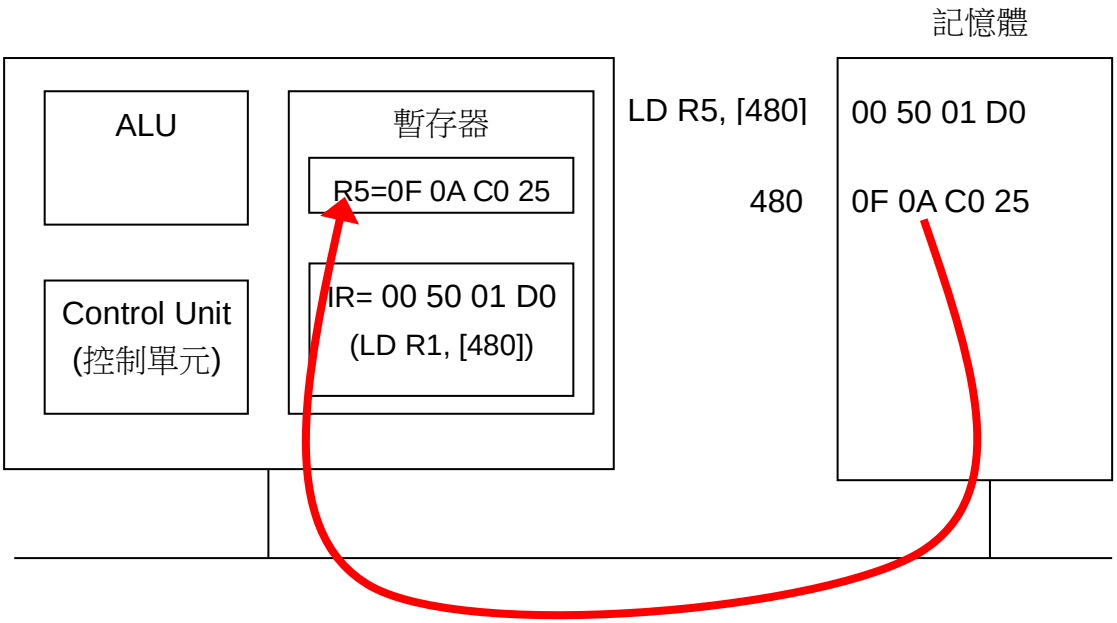
控制單元根據指令暫存器 IR 的指示，打開 R6 與 R1 輸入到內部匯流排的開關，於是 R6 與 R1 輸入到 ALU 當中，接著在 ALU 計算出結果之後，控制單元設定 ALU 的輸出流回到 R5 暫存器中，其資料流向圖如下所示。



2.10 請畫出 LD R5, [480] 指令的資料流向圖，並說明該指令的運作方法。

參考解答：

控制單元根據指令暫存器 IR 的指示，解碼後發現指令為 LD，於是將位址 480 (即 16 進位的 1D0) 輸出到位址匯流排中，然後設定控制匯流排為記憶體讀取模式，當記憶體偵測到讀取訊息後，會將位於 480 中的資料 (例如圖中的 0F 0A C0 25) 傳送到資料匯流排上，接著控制單元會將這個資料存入到暫存器 R5 當中，如下圖所示。



2.11　請簡要說明 IA32 處理器的特性？

參考解答：

IA32 是 Intel 公司所設計的處理器, 屬於 x86 系列處理器的成員。x86 處理器採用傳統的複雜指令集架構，包含四個通用暫存器　EAX, EBX, ECX, EDX。這些通用暫存器常用來載入變數與儲存運算結果。暫存器 EBP (Extended Base Pointer) 可是框架指標 (Frame Pointer, FP)。ESP(Extended Stack Pointer) 則是堆疊指標, ESI (Extended Source Index) 與 EDI (Extended Destination Index) 則是索引暫存器, ESI 通常指向來源位址, EDI 則指向目標位址。

IA32 內含浮點運算功能，可以進行整數、浮點數的加減乘除法。由於是複雜指令集，IA32 的每個指令通常具有許多不同的格式，這些格式提供組合語言較大的彈性，但相對的也較不容易記憶，因此撰寫 IA32 的組合語言是一件困難的工程。

第 3 章習題

3.1　請寫出一個 CPU0 的組合語言程式, 可以計算 a=b*3+c-d 的算式。

參考解答：

```
LD R1, b
LDI R2, 3
LD R3, c
LD R4, d
MUL R1, R1, R2
ADD R1, R1, R3
SUB R1, R1, R4
ST R1, a
```

3.2　請寫出一個 CPU0 的組合語言副程式 swap, 可以將暫存器 R1 與 R2 的內容交換。

參考解答：

```
Swap:
   MOV R3, R1
   MOV R1, R2
   MOV R2, R3
   RET
```

3.3　請寫出一個 CPU0 的組合語言副程式 isPrime, 可以判斷暫存器 R2 當中的值是否為質數, 如果是就將 R1 設為 1 傳回, 否則就將 R1 設為 0。

參考解答：

	組合語言	C 語言 (對照版)	C 語言 (簡化版)
1	LDI R1, 1	void main() {	BOOL isPrime(int R2) {
2	LDI R3, 0	R1 = sum;	R1 = TRUE;
3	FOR: CMP R3, R2	R2 = i;	for (R3=0; R3<R2; R3++) {
4	JGE EXIT	R3 = ten;	R4 = R2/R3; // 幾倍
5	DIV R2, R2, R3	R4 = one;	R5 = R3*R4; // 最接近的
6	MUL R5, R3, R4	if (R2 > R3)//(i > 10)	倍數
7	SUB R6, R2, R5	goto EXIT;	R6 = R2-R5; // 餘數
8	CMP R6, R0	R1 = R2 + R1;	if (R6 == 0) // 若餘數為 0
9	JEQ NO	R2 = R2 + R4;	R1 = FALSE;
10	JMP FOR	goto FOR;	return;

11	NO: LDI R1, 0	EXIT:return;	}
12	EXIT: RET	int i;	return;
13	i RESW 1	int sum = 0;	}
14	sum WORD 0	int ten = 10;	
15	ten WORD 10	int one = 1;	
16	one WORD 1	}	
	END		

3.4 請寫出一個 CPU0 的組合語言程式, 可以計算出 2*2+4*4… +100*100 的結果, 並將結果儲存在變數 sum 當中

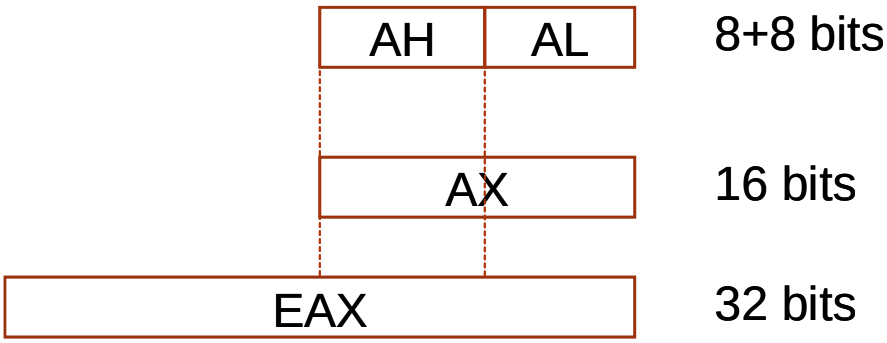
參考解答：

	組合語言	C 語言 (對照版)	C 語言 (簡化版)
1	LD R1, sum	void main() {	void main() {
2	LD R2, i	R1 = sum;	int sum=0;
3	LDI R3, 100	R2 = i;	int i;
4	LDI R4, 1	R3 = 100;	for (i=0; i<=100; i++)
5	FOR:CMP R2, R3	R4 = 1;	sum += i*i;
6	JGT EXIT	if (R2 > R3)//(i > 10)	}
7	MUL R3, R2, R2	goto EXIT;	
8	ADD R1, R3, R1	R3 = R2*R2	
9	ADD R2, R4, R2	R1 = R1 + R3;	
10	JMP FOR	R2 = R2 + R4;	
11	EXIT: RET	goto FOR;	
12	i: RESW 1	EXIT:return;	
13	sum:WORD 0	int i;	
14		int sum = 0;	
15		}	
16			

3.5 請以圖解的方式, 說明在 IA32 處理器的 eax 暫存器中, 為何會有 eax, ax, ah, al 等不同名稱, 這些名稱代表的是哪個部分？

參考解答：

因為 IA32 的 x86 CPU 有悠久的歷史，在 80286 的時代使用 16 位元的 AX，在 80386 之後擴充為 32 位元的 EAX，而 AX 又被分為兩個位元組 AH 與 AL，如下圖所示。



3.6　請寫出一個 IA32 的組合語言副程式 swap，可以將暫存器 EAX 與 EBX 的內容交換。

參考解答：

```
Swap:
   MOV R3, R1
   MOV R1, R2
   MOV R2, R3
   RET
```

3.7　請寫出一個 IA32 的組合語言副程式 isPrime，可以判斷暫存器 EBX 當中的值是否為質數，如果是就將 EAX 設為 1 傳回，否則就將 EAX 設為 0。
參考解答：

	GNU 組合語言	C 語言
1	.text	int isPrime(int n) {
2	.globl _isPrime	int i;
3	.def _isPrime; .scl 2; .type 32;	for (i=1; i<n; i++) {
4	.endef	if (n % i == 0)
5	_isPrime:	return 0;
6	pushl %ebp	}
7	movl %esp, %ebp	return 1;
8	subl $12, %esp	}
9	movl $1, i	
10	L2:	
11	movl i, %eax	

12	cmpln, %eax	
13	jge L3	
14	movl n, %edx	
15	leal i, %eax	
16	movl %eax, t	// t=n%i
	movl %edx, %eax	
	movl t, %ecx	
	cltd	
	idivl (%ecx)	
	testl %edx, %edx	
	jne L4	
	movl $0, n	
	jmp L1	
	L4:	
	leal i, %eax	
	incl (%eax)	
	jmp L2	
	L3:	
	movl $1, n	
	L1:	
	movl n, %eax	
	leave	
	ret	
	.data	
	n .long 1	
	i .long 1	
	t .long 1	

3.8 請撰寫一個 IA32 的組合語言程式, 可以計算 2*2+4*4…+100*100 的結果後傳回, 然後仿照 3.5 節的作法, 使用 GNU 的 gcc 編譯連結該程式, 並且執行看看結果是否正確。

參考解答：

	組合語言	C 語言 (簡化版)
1	.file "sum2.c"	int sum2() {
2	.text	int i, sum=0;
3	.p2align 4,,15	for (i=1; i<100; i++)

4	.globl _sum2	sum += i*i;
5	_sum2:	return sum;
6	pushl %ebp	}
7	xorl %ecx, %ecx	
8	movl %esp, %ebp	
9	movl $1, %edx	
10	.p2align 4,,15	
11	L5:	
12	movl %edx, %eax	
13	imull %edx, %eax	
14	incl %edx	
15	addl %eax, %ecx	
16	cmpl$99, %edx	
	jle L5	
	popl %ebp	
	movl %ecx, %eax	
	ret	

第 **4** 章習題

4.1　請說明組譯器的輸入、輸出與功能為何？

參考解答：
組譯器的輸入是組合語言程式、輸出是目的檔或執行檔，而其功能為將組合語言程式組譯後，輸出指令機器碼到目的檔中。

4.2　請說明組譯器第一階段 (PASS1) 的功能為何？
參考解答：
組譯器第一階段 (PASS1) 的功能有兩個，一是計算各個指令與資料的位址，二是建立符號表，並將符號的位址填入符號表中。

4.3　請說明組譯器第二階段 (PASS2) 的功能為何？
參考解答：
請說明組譯器第二階段 (PASS2) 的功能主要是將組合語言的指令與資料轉換為機器碼，然後將機器碼記錄在目的檔中。

4.4　請說明組譯器當中的符號表之用途為何？
參考解答：
組譯器當中的符號表是用來記錄各個符號的位址，符號的類型可分為程式標記與資料標記，組譯器會在 PASS1 建立符號表，然後在 PASS2 取出符號位址以供組譯器編定機器碼。

4.5　請說明組譯器當中的指令表之用途為何？
參考解答：
組譯器當中的指令表是用來記錄指令的助憶名稱、指令代碼、編碼格式與指令大小等訊息的一個表格，通常使用雜湊表實作，以供組譯器 PASS1 查出指令大小，並且在 PASS2 將指令轉換為機器碼。

4.6　請仿照範例 4.4, 使用本書第 12 章所實作的 as0 組譯器, 組譯 Ex4_1.asm0 組合語言檔, 並仔細觀察其輸出結果。
參考解答：
(略) 請逕行參考範例 4.4 的操作方式。

4.7　請閱讀本書第 12 章所附的 Assembler.c 與 Assembler.h 等 C 語言程式，並且對照本章的演算法，以學習 CPU0 組譯器的實作方式。

參考解答：

(略) 請逕行參考書籍光碟或本書網站中的程式。

4.8　請按照 4.4 節的方法，操作 GNU 工具對組合語言進行組譯動作，並檢視組譯報表，找出各個符號的位址。

參考解答：

(略) 請逕行參考範例 4.4 的操作方式。

4.9　請於 http://kipirvine.com/asm/ 網站下載 Kip Irvine 書籍組合語言程式範例，並以 Visual Studio 進行組譯與執行。

參考解答：

請參考 Irvine 網站中的 Getting started with MASM 一文，其網址如下：

http://kipirvine.com/asm/gettingStartedVS2010/index.htm

第 5 章習題

5.1　請說明連結器的輸入、輸出與功能為何？
參考解答：
連結器的輸入為一群目的檔，輸出為單一的可執行檔，與功能為將目的檔連接後，重新安排符號表中的符號位址，並且利用修正記錄修改指令中的符號位址，在所有的符號位址都確定之後，才輸出可執行檔。

5.2　請說明載入器的功能為何？
參考解答：
載入器的功能是將執行檔 (或目的檔) 載入到記憶體中，並且根據載入的記憶體位址修正記憶體中的指令與資料，然後將程式計數器設定為該執行檔的起始位址，交由 CPU 開始執行該程式。

5.3　請說明 CPU0 組合語言當中的 .text, .data 與 .bss 等假指令的用途為何？
參考解答：
.text 是程式段，其內容通常為組合語言指令，像是 LD R1, 100 等指令。
.data 是資料段，內容通常為資料定義，像是 sum: WORD 0 這樣的宣告指令。
.bss 是 Block Started by Symbol 的簡稱，內容通常是未初始化的變數定義，像是 array: RESW 100 這樣的宣告指令。

5.4　請說明 CPU0 組合語言當中的 .global 與 .extern 等假指令的用途為何？
參考解答：
.global 用來宣告一個變數為全域變數，可以被其他程式所存取，像是 .global stack 就是將 stack 變數宣告為全域變數。

.extern 用來宣告一個變數為外部變數，其位址暫時無法決定，必須等到連結時才能確定，像是 .extern stack 就是將 stack 變數宣告為外部變數。

5.5　請說明 CPU0 目的檔中的 T, D, B, S, M 等記錄各有何用途？
參考解答：
T 紀錄：程式段 (.text) 紀錄，通常用來存放機器指令。
D 紀錄：資料段 (.data) 紀錄，通常用來存放資料的機器碼。
B 紀錄：BSS 段 (.bss) 紀錄，通常用來紀錄未初始化變數的長度。
S 紀錄：符號段 (symbol table) 紀錄，通常用來存放符號表。
M 紀錄：修改段 (modify) 紀錄，通常用來存放修正指令，可於連結或載入時進

行程式與資料的內容修改。

5.6　請說明連結器是如何處理外部引用問題的？
參考解答：
組譯器在組譯到外部變數時，由於無法決定記憶體位址，因此會暫時填入 0 給該符號，並為指令加上修正紀錄。接著，當連結器連結外部目的檔時，一但發現該外部變數的位址被定義時，就會填入該外部變數的位址到符號表中，並根據修正紀錄修改目的檔中的指令或資料，以便讓指令中的外部變數具有正確的位址。

5.7　請說明目的檔中符號表的用途？
參考解答：
目的檔中符號表記錄了所有的標記符號的位址，這讓連結器可以在重新定位時修正符號的位址，並且可以根據修改紀錄與符號表進行修正動作，以便正確的修正目的檔，如此在連結時才能對目的檔進行修正，以輸出執行檔。

5.8　請使用 gcc 加上-S -c 參數, 分別編譯範例 5.2 中的三個程式,以分別產生組合語言檔。
參考解答：
(略) 請逕行參考範例 5.8 進行操作。

5.9　繼續前一題, 請使用 gcc 分別組譯前一題所產生的三個組合語言檔, 產生目的檔。
參考解答：
(略) 請逕行參考範例 5.9 進行操作。

5.10　繼續前一題, 請使用 gcc 連結前一題所產生的三個目的檔, 輸出執行檔。
參考解答：
(略) 請逕行參考範例 5.8 進行操作。

5.11　繼續前一題, 請使用 nm 指令分別觀看這三個目的檔與輸出的執行檔。
參考解答：
(略) 請逕行參考範例 5.11 進行操作。

5.12　繼續前一題, 請使用 objdump 指令分別觀看這三個目的檔與輸出的執行檔。
參考解答：
(略) 請逕行參考範例 5.14 進行操作。

5.13　繼續前一題, 請找出其中的符號表部分。

參考解答：

```
SYMBOL TABLE:
[  0](sec -2)  (fl 0x00) (ty   0)  (scl 103)  (nx 1) 0x00000000 StackFunc.c
File
[  2](sec  1) (fl 0x00)         (ty  20)  (scl   2)       (nx 1) 0x00000000 _push
AUX tagndx 0 ttlsiz 0x0 lnnos 0 next 0
[  4](sec  1) (fl 0x00)         (ty  20)  (scl   2)       (nx 0) 0x0000001c _pop
[  5](sec  1) (fl 0x00)         (ty   0)  (scl   3)       (nx 1) 0x00000000 .text
AUX scnlen 0x33 nreloc 6 nlnno 0
[  7](sec  2) (fl 0x00)         (ty   0)  (scl   3)       (nx 1) 0x00000000 .data
AUX scnlen 0x0 nreloc 0 nlnno 0
[  9](sec  3) (fl 0x00)         (ty   0)  (scl   3) (nx 1) 0x00000000 .bss
AUX scnlen 0x0 nreloc 0 nlnno 0
[ 11](sec  0) (fl 0x00)         (ty   0)  (scl   2)       (nx 0) 0x00000000 _top
[ 12](sec  0) (fl 0x00)         (ty   0)  (scl   2)       (nx 0) 0x00000000 _stack
```

第 6 章習題

6.1　請說明巨集處理器的輸入、輸出與功能為何？
參考解答：
巨集處理器的輸入為一個程式，其中可能會有巨集定義與巨集呼叫。
巨集處理器的輸出為一個已經將巨集展開後的程式，其中不再有巨集定義與巨集呼叫。
巨集處理器與功能就是將輸入程式的巨集展開，當發現巨集定義時，就將巨集內容記錄到符號表中，而在遇到巨集呼叫時，就將該巨集定義從符號表中取出，然後根據呼叫參數進行巨集展開的動作，並將展開後的內容寫入輸出檔中。

6.2　請說明巨集處理器會如何處理巨集參數？
參考解答：
巨集處理器遇到巨集呼叫時，就將該巨集定義從符號表中取出，然後根據呼叫參數進行巨集展開的動作。展開時通常會根據參數的順序，將內容中的參數取代成呼叫參數，如此即完成巨集展開的動作了。

6.3　請說明巨集處理器在展開標記時會產生甚麼問題，應如何解決？
參考解答：
當有兩次以上的巨集呼叫，呼叫同一個巨集時，如果直接展開標記就會造成標記重複定義的問題。因此巨集處理器在展開巨集內容時，會為標記加上編號，以避免標記重複的問題。

6.4　請使用 gcc 工具將範例 6.2 展開，觀察展開後的檔案，並說明展開前後的對應關係。
參考解答：

指令：gcc -E macro.c -o macro_E.c
內容：(略) 請逕行參考範例 6.2。

6.5　請使用 gcc 工具將範例 6.3 展開，觀察展開後的檔案，並說明展開前後的對應關係。
參考解答：
指令：gcc -E macroDebug.c -o MacroDebug_E.c
內容：(略) 請逕行參考範例 6.3。

第 **7** 章習題

7.1　請說明何謂 BNF 語法？何謂 EBNF 語法？並比較兩者的異同。
參考解答：
BNF 是一種用來描述程式語言語法的規則集合，這些規則的語法就稱為 BNF 語法。我們可以根據 BNF 撰寫語法剖析程式，但是當我們用遞迴下降式的方式撰寫剖析器時， BNF 語法會造成左遞迴問題，我們通常可以用重複語句改寫 BNF 為 EBNF 語法，以消除語法中的左遞迴規則，這種具有重複語句的 BNF 語法就稱為 EBNF 語法。

7.2　請將 BNF 語法 A = B | A '.' B 轉換為 EBNF 語法。
參考解答：
利用重複語句消除左遞迴現象，將語句改為 A = B ('.' B)*

7.3　請寫出 C 語言當中 for 迴圈的 BNF 語法。
參考解答：
FOR = 'for' '(' EXP ';' EXP ';' EXP ')' STMT

7.4　請說明何謂直譯器？
參考解答：
直譯器是一種可以直接解譯程式，而不需要經過編譯動作的軟體。直譯器會在語法剖析動作完成之後，直接對語法樹進行解譯的動作。解譯時對語法樹中的每個節點，進行解譯的動作，這種動作通常採用遞迴下降的解譯方式，由上而下依序解譯每個節點，模擬對應的動作，並設定對應的變數。

7.5　請說明何謂編譯器？
參考解答：
編譯器是用來將程式編譯為組合語言的工具，編譯器會在語法剖析動作完成之後，再度將語法樹轉換成組合語言輸出，有時編譯器會順便呼叫組譯器將編譯完成的組合語言，組譯為目的檔或執行檔直接輸出，以方便使用者使用。

7.6　請比較直譯器與編譯器兩者的異同。
參考解答：
直譯器與編譯器的相同點是都會進行語法剖析動作，因此通常都包含了一個剖析器，這個剖析器可以剖析程式後建立語法樹。另外，編譯器與直譯器的目的都是為了讓程式可以執行，因此都是程式設計師所使用的重要工具。

直譯器與編譯器的相異點是，直譯器在剖析完成後就直接解譯語法樹，而不需要轉換成其他格式輸出。但是編譯器就必須將語法樹轉換成組合語言後再行輸出，然後經過組譯連結動作後輸出執行檔，之後才能交由載入器將程式載入後執行。

由於執行檔被載入到記憶體後，是由 CPU 直接執行的，因此編譯後的程式執行速度非常快，通常是直譯方式的數十倍甚至上百倍，因此強調速度的程式語言最好採用編譯的方式，而比較注重彈性的程式語言則會採用解譯動作。

7.7 請說明何謂語法理論？

參考解答：

語法理論可以用來描述語言的語法，Chromsky 所提出的生成語法就可以被用來描述自然語言 (像是中英文) 的語法，或者是程式語言 (像是 C, C#, Java) 的語法。一但語法能正確的描述，我們才能撰寫出剖析器、編譯器與直譯器等程式。

7.8 請說明何謂語意理論？

參考解答：

語法理論只描述語法結構，但是並不會說明程式應該如何執行，而語意理論則用來描述程式的執行方式，通常針對不同的語法會有不同的語意結構，像是循序結構的指令是按照順序執行的，這就是其語意，而迴圈結構的區塊會被執行很多次，直到離開迴圈的條件成立為止，這就是迴圈結構的語意。

7.9 請說明何謂框架 ？

參考解答：

在高階語言執行時，由於相當依賴堆疊以存取變數、參數等記憶區塊，因此用框架暫存器記住區塊的開頭，以便作為變數定址的基準位址，這個方式就稱為框架。

7.10 請舉例說明 C 語言如何利用框架暫存器存取參數與區域變數？

參考解答：

(略) 請逕行參考範例 7.1 與圖 7.18。

第 8 章習題

8.1　請為 C0 語言加上 if 語句的 EBNF 語法, 加入到圖 8.2 中。

參考解答：

```
BASE    = FOR | IF | STMT ';'
...
IF      = 'if' '(' COND ')' BASE ('else' BASE)?
…
```

8.2　接續上題, 請在圖 8.9 當中加入剖析 if 語句的演算法。

參考解答：

```
function parseIf()
   pushNode("IF")
   next("if")
   next("(")
   parseCond()
   next(")")
   parseBase()
   if (isNext("else"))
      next("else")
      parseBase()
   end if
   popNode("IF")
end
```

8.3　接續上題, 請在圖 8.13 當中加入將 if 語句轉為中間碼的演算法。

參考解答：

```
if (node.tag=IF)
   cond = node.childs[2]
   base1 = node.childs[4]
   base2 = node.childs[6]
   condOp = generate(cond);
   negateOp(condOp, negOp);
   labelEndIf = "ENDIF"+tempIfCount;
   labelBase1 = "BASE1_"+tempIfCount;
```

```
    labelBase2 = "BASE2_"+tempIfCount;
    pcode("", "J", condOp, "", base1);
    pcode("", "J", negOp, "", base2);
    pcode(labelBase1, "", "", "", "");
    generate(base1);
    pcode("", "J", labelEndif, "", "");
    pcode(labelBase2, "", "", "", "");
    generate(base2);
    pcode(labelBase2, "", "", "", "");
    pcode(labelEndif, "", "", "", "");
    return NULL;
else
    ...
```

8.4 請為範例 8.5 (b) 的無最佳化組合語言，提出一個簡單的最佳化機制，並寫出您的最佳化方法實施後，所產生的組合語言程式碼。

參考解答：

利用符號表紀錄暫存器與變數的對應關係，當符號還在暫存器當中時，就不需要進行載入儲存動作，以加快執行速度，減少指令數量。

其最佳化結果就如範例 8.5 (c) 所示。

8.5 請使用 gcc 的 -O0 與 -O3 參數，分別已無最佳化與高級最佳化的方式編譯任意一個 C 語言程式為組合語言，並觀察其編譯後的組合語言，指出最佳化後哪些指令被省略了。

參考解答：

(略) 請逕行參考範例 8.9。

第 9 章習題

9.1　請說明何謂虛擬機器？

參考解答：

虛擬機器是用軟體模擬硬體指令集的一種程式，像是 QEMU、Bochs、Virtual PC、VMWare、Virtual Box 等都是虛擬機器軟體。我們可以將程式直接放在虛擬機器上面執行，感覺就好像真正在硬體機器上執行一樣。

9.2　請說明何謂記憶體機？

參考解答：

記憶體機的任何指令可以存取所有的記憶體，其運算元均為記憶體，該機器中沒有暫存器的概念。像是 ADD X, Y, Z 就是一個記憶體機的加法運算指令，其中 X,Y,Z 均為記憶體位址。

9.3　請說明何謂堆疊機？

參考解答：

堆疊機當中用堆疊以取代暫存器，因此所有的運算都直接在堆疊上進行，運算結果也直接被推回到堆疊當中，因此除了 PUSH、POP 指令會存取記憶體之外，其他指令均只包含運算碼，沒有參數的部份。像是 ADD 指令就會將堆疊最上層的兩個元素取出後，再將加法的運算結果推回堆疊當中。

9.4　請說明何謂暫存器機？

參考解答：

暫存器機通常利用暫存器作為運算的來源與目標，像是 ADD R1, R2, R3 就會將 R2 與 R3 相加之後，存回到暫存器 R1 當中。

9.5　請閱讀本書的第 12 章, 並取得 ch12/CPU0.c 這個程式, 看看這個虛擬機如何設計的。

參考解答：

(略) 請逕行參考第 12 章與範例光碟中的程式碼。

9.6　請寫出一個 Java 程式, 並且使用 javac 編譯該程式, 然後使用 java 指令執行該程式。

參考解答：

(略) 請逕行參考範例 9.3 的程式與操作方式。

9.7　接續上一題, 請使用 javap 將上一題產生的 bytecode 反組譯, 並分析反組譯後的程式碼？

參考解答：
(略) 請逕行參考範例 9.5 的程式與操作方式。

9.8　請安裝 Virtual PC, 然後在其中安裝 DOS 作業系統。

參考解答：
(略) 請逕行參考 9.5 節的 Virtual PC 操作方式。

第 **10** 章習題

10.1　請說明何謂行程？何謂執行緒？行程與執行緒有何不同？

參考解答：

行程是執行中的程式，作業系統必須為每個行程儲存堆疊、暫存器、記憶空間、開啟檔案串列等資訊，然後才能進行行程切換的動作。執行緒是行程的一部份，許多個執行緒可以共同組成一個行程，這些執行緒之間會共用記憶體、開啟檔案串列等資訊，因此在作業系統進行執行緒切換動作時，只需要保存堆疊指標與暫存器等資訊，因此執行緒的切換動作通常較快，而且執行緒之間可以共用變數。

10.2　請說明排程系統中的行程切換機制是如何進行的？當行程切換時作業系統需要保存哪些資料？

參考解答：

當行程進行系統呼叫，或者被中斷機制強制中斷時，作業系統會決定是否要進行切換動作。

切換時會先將原行程的暫存器、堆疊指標、記憶體空間映射資訊、開啟檔案列表等資訊儲存到作業系統核心的記憶體中，然後在從記憶體中這些新行程的資訊，最後再將新行程的程式計數器 PC 載入後，新行程就會開始執行，這樣就完成了行程切換的動作。

10.3　請說明何謂 MMU 單元, 具有 MMU 單元處理器的定址方式與沒有 MMU 者有何不同？

參考解答：

MMU 是記憶體管理單元 (Memory Management Unit) 的簡稱，目前在作業系統中所使用的 MMU 技術頗為多樣，像是「重定位暫存器」、「基底-界線暫存器」、「分頁機制」、「分段機制」、「分段式分頁機制」等等，MMU 讓作業系統得以為行程實現較完善的記憶體管理保護機制，「重定位暫存器」讓載入器不需要在載入時進行指令位址部份的修正動作，只要適當的設定 MMU 即可。而「基底-界線暫存器」除了具有上述好處之外，還可以防止程式存取其他程式的記憶體空間，以保護作業系統與其它使用者程式。「分段」與「分頁」的組合機制則除了具有這兩個好處之外，還可以實作出虛擬記憶體機制，讓作業系統可以在記憶體不足時，將記憶體內容儲存在硬碟當中，以便載入更多的行程，強化多工機制。

10.4　請說明何謂驅動程式？驅動程式與輸出入系統有何關係？

參考解答：

驅動程式是一種控制底層硬體的程式，像是鍵盤驅動程式、記憶卡驅動程式等。這些程式必須被作業系統或上層的程式呼叫，才能夠在適當的時機控制這些硬體

裝置。

作業系統的輸出入子系統，主要的功能即是用來呼叫驅動程式，以便順利控制輸出入硬體裝置。一個常見的作法是由作業系統規定一套驅動程式的標準介面函數，所有的驅動程式必須實作這些介面函數，然後透過「註冊-反向呼叫」的方式，讓作業系統得以在需要的時候呼叫這些驅動程式，以便完成輸出入動作。並且利用這些驅動程式做為檔案系統的實作基礎。

10.5　請說明何謂檔案系統？為何在 Linux 當中可以將裝置當作檔案進行讀寫呢？

參考解答：

檔案系統是作業系統的一個子系統，會呼叫輸出入子系統的驅動程式，以便在需要時存取輸出入裝置。但是，檔案系統會將輸出入子系統提供的介面函數進一步封裝成「檔案、資料夾與磁碟分割」等單元，以便有效的統整輸出入裝置的記憶空間，提供給使用者使用。

在 Linux 當中，徹底的將輸出入系統與檔案系統緊密的結合起來，讓所有的裝置都納入檔案系統的管理，並且直接以檔案系統的「檔案、資料夾、inode 與磁碟分割」作為輸出入子系統必需實作的標準介面，因此讓所有的裝置都納入檔案系統的管轄當中。這種方式讓整個作業系統有了相當統一的存取結構，是 Linux 從 UNIX 系統中所繼承下來的優良遺產。

10.6　請說明 Linux 中的行程管理系統採用哪些策略？

參考解答：

Linux 行程管理採用的是具有優先權的 Round-Robin 排程機制，優先權可分為靜態與動態兩種。每個行程都會分配到一個基本的時間配額，並且設定優先權。行程還可以根據需求，設定為普通行程或即時行程兩類。即時行程與普通行程有著不同的排程策略，以便適應即時的要求。

Linux 透過行程描述器，儲存行程的相關資訊，以便實作行程的建立、排程、切換與毀滅動作。使用者可以透過 fork() 這個系統呼叫從舊有的行程中分叉出新行程，然後再透過 execvp() 這樣的函數載入新的程式，以建立新的行程內容。

10.7　請說明 Linux 中的記憶體管理系統採用哪些策略？

參考解答：

X86 版本 Linux 利用 GDT 指向核心的分頁, 然後用 LDT 指向使用者行程的分頁。LDT 中所記載的是各個行程的分段表, 以及行程的狀態段 (Task State Segment : TSS)。而 GDT 中則會記載這些分段表的

起始點, TSS 起始點, 以及核心的各分段起點。圖 10.16 顯示了 x86 版的 Linux 的分段記憶體管理機制。

透過圖 10.16 的分段機制, Linux 可為每一個行程分配一塊大小不等的區段。其中每一個區段又可能佔據許多個分頁, x86 版本的 Linux 採用 IA32 的延伸分頁模式, 利用 IA32『分段+雙層分頁』的延伸記憶體管理模式, 因此每個頁框的大小為 4KB。

當需要進行分段配置 (例如載入行程) 時, Linux 會使用對偶式記憶體管理演算法 (Buddy System Algorithm) 配置分頁, 在 Buddy 系統中一個頁框代表一段連續的分頁, 該演算法將頁框區分為十種區塊大小, 分別包含了 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 個連續的頁框, 其中每個區塊的第一個頁框位置一定是區塊大小的倍數。舉例而言, 一個包含 32 個頁框的區塊之起始位址一定是 32 * 4KB 的倍數。

當 Linux 需要配置的是小量的記憶體 (像是 malloc 函數所需的記憶體) 時, 採用的是一種稱為 Slab Allocator 的配置器, 其中被配置的資料稱為物件 (Object)。 Slab 中的物件會被儲存在 Buddy 系統所分配的頁框中, 假如要分配一個大小為 30 bytes 的物件時, Slab 會先向 Buddy 系統要求取得一個最小的分頁 (大小為 4KB), 然後分配個 Slab 配置器。然後 Slab 配置器會保留一些位元以記錄配置資訊, 然後將剩下的空間均分為大小 30 的物件。於是當未來再有類似的配置請求時, 就可以直接將這些空的物件配置出去。

10.8 請說明 Linux 中的輸出入管理系統採用哪些策略？
參考解答：
Linux 將硬體裝置分為『區塊、字元、網路』等三種類型, 這三類型的驅動程式都必須支援檔案存取的介面, 因為在 Linux 當中裝置以檔案的方式呈現的。像是 /dev/hda1, /dev/sda1, /dev/tty1 等, 程可以透過開檔 open()、讀檔 read()、寫檔 write() 的方式存取裝置像存取一個檔案一樣。

因此, 所有的驅動程式必須支援檔案 (file) 的操作 (file_operations)便將裝置偽裝成檔案, 供作業系統與應用程式進行呼叫。這種將裝置裝成檔案的方式, 是從 UNIX 所承襲下來的一種相當成功的模式。

10.9 請說明 Linux 中的檔案管理系統採用哪些策略？
參考解答：
Linux 的檔案系統包含了『檔案』、『目錄』、『路徑』、『屬性』等高層邏輯概念, ,

然而底層的裝置所面對的是「位元、磁區、磁軌、控制暫存器」等概念。為了連接這兩個不同層次的概念，Linux 採用了 inode 這個組織結構，將磁區組合成檔案，並且將檔案透過 inode 連結到目錄結構中。這便是 Linux 檔案管理系統的設計方式，採用以 inode 進行連結管理的策略。

10.10 請安裝 Ubuntu Linux 的最新的版本, 然後使用看看。
參考解答：
(請實際操作) 您可以從 Ubuntu 官方網站 http://www.ubuntu.com/ 中下載 *.iso 的映像檔，燒錄到光碟或隨身碟中安裝。或者您也可以安裝 VMWare 等虛擬機器後在將 *.iso 檔載入虛擬機器中安裝成一台虛擬機。另外，您也可以用 Wubi 安裝程式，將 Ubuntu 安裝到您的作業系統中，而不需要做磁碟分割的動作 https://wiki.ubuntu.com/WubiGuide。另外，您還可以

10.11 請於 Cygwin 環境下編譯 ch10/fork.c 檔案, 並執行看看。
參考解答：
(略) 請參考範例 10.3 的操作方式。

10.12 請於 Cygwin 環境下編譯 ch10/thread.c 檔案, 並執行看看。
參考解答：
(略) 請參考範例 10.4 的操作方式。

10.13 請於 http://www.kernel.org/ 網站中下載 Linux 核心原始碼的最新版本, 然後看看其中的 include/linux/sched.h、arch/x86/ include/asm/processor.h、arch/x86/include/asm/system.h 等原始碼, 試著理解其運作邏輯。
參考解答：
(略) 請實際下載閱讀。

第 **11** 章習題

11.1　請說明何謂專用輸出入指令？

參考解答：

專用的輸出入指令，像是 in, out 等 CPU 所提供的指令，是專門為了輸出入而設計的處理器指令，通常會透過 IO 控制器將輸出入裝置連接到一個稱為輸出入埠 (IO port) 的暫存器上，然後讓 CPU 透過 in, out 等指令存取這些埠 (暫存器)，以便進行輸出入的動作。

11.2　請說明何謂記憶體映射輸出入？

參考解答：

有接 CPU 並不提供專用的輸出入指令，而是直接採用 LD、ST 等記憶體存取指令，存取並控制輸出入裝置。為了讓 CPU 能順利的透過 LD、ST 等記憶體存取指令控制輸出入裝置，裝置控制器會偵測匯流排上的訊息，並且將裝置上的暫存器映射到某一個記憶體位址上，讓 CPU 可以順利透過記憶體存取指令存取並控制這些裝置。

11.3　請說明何謂中斷機制？中斷發生時程是會跳到哪裡呢？

參考解答：

中斷機制是由輸出入裝置，利用中斷訊號，主動回報輸出入裝置情況給 CPU 的一種技術。這種技術必須依靠硬體的配合，當輸出入裝置想要回報訊息時，可透過匯流排，傳遞中斷訊號給 CPU。此時, CPU 會暫停目前正在執行的程式，跳到對應的中斷向量上，該中斷向量內會包含一個跳向中斷函數的指令，讓 CPU 開始執行該中斷函數。

11.4　請說明啟動程式應該做些甚麼事呢？

參考解答：

電腦開機後，第一個被執行的記憶體位址，稱為啟動位址，啟動程式的第一個指令，必須被正確的燒錄到該位址中，才能正確的啟動。然而，程式設計師對啟動程式通常會感到迷惑。因為啟動程式必須在電腦一開機時就存在記憶體中，因此，只能將啟動程式放到 ROM 或 FLASH 等永久性儲存體當中。

啟動程式必須建立程式的執行環境，讓其他程式得以順利執行，其主要任務包含下列四項：

1. 設定中斷向量，啟動中斷機制。

2. 設定 CPU 與主機板的各項參數，讓 CPU 與主機板得以進入正確的狀態。
3. 將存放在永久儲存體中的程式與資料搬到記憶體中。
4. 設定高階語言的執行環境，包含設定堆疊 (Stack)、堆積 (Heap) 等區域。

11.5 請寫出一個完整的 C 語言程式，讓 M0 實驗板在數字按鍵被按下時，於七段顯示器當中顯示對應的數字。

參考解答：

```
#define BYTE unsigned char
#define UINT16 unsigned short
#define BOOL unsigned char
#define SEG7_REG(*(volatile BYTE*)0xFFFFFF00)
#define KEY_REG1 (*(volatile BYTE*) 0xFFFFFF01)
#define KEY_REG2 (*(volatile BYTE*) 0xFFFFFF02)
#define KEY (KEY_REG2 << 8 | KEY_REG1)

#define BYTE seg7map[]={ /*0*/ 0x3F,
   /*1*/ 0x18, /*2*/ 0x6D, /*3*/ 0x67,
   /*4*/ 0x53, /*5*/ 0x76, /*6*/ 0x7E,
   /*7*/ 0x23, /*8*/ 0x7F, /*9*/ 0x77 };
#define char keymap[]={
   '1', '2', '3', '+',
   '4', '5', '6', '-',
   '7', '8', '9', '*',
   '#', '0', '?', '/'};

// 七段顯示器驅動程式
void seg7_show(char c) {
   SEG7_REG = map7seg[c-'0'];
}
// 鍵盤驅動程式
char keyboard_getkey() {
   UNIT16 key = KEY;
   for (int i=0; i<16; i++) {
      UNIT 16 mask = 0x0001 << i;
      if (key & mask !=0)
         return keymap[i];
   }
```

```
    return 0;
}
BOOL keyboard_ishit() {
    return (KEY != 0)
}

int main() {
    while (1) {
        while (!keyboard_ishit()) {}
        char key = keyboard_getkey();
        if (key >='0' && key <='9')
            seg7_show(key);
    }
}
```

11.6　同上題, 但是請以 CPU0 的組合語言撰寫。

參考解答：

(略) 由於本題用組合語言寫會顯得相當得長,因此建議老師不要出這一提作為習題，除非老師刻意要出難題時，就可以請同學針對這題作答。

第 **12** 章習題

12.1　請撰寫一個 C0 語言的程式 fib.c0, 可以利用 for 迴圈的方式算出費氏序列中的 f(10) 的值, 費氏序列的規則為 f(n) = f(n-1)+f(n-2), 而且 f(0) = 1, f(1)=1。
參考解答：

請注意，由於 c0 語言不支援陣列，因此不能使用陣列實作本題。而且 C0 語言的變數名稱目前不支援底線，因此不能使用像 fn_1 這樣的名稱。所以在本題中，我們使用 fn 代表 f(n)，fnd 代表 f(n-1)，fni 代表 f(n+1)。

C0 語言程式	說明

fn = 1;
fnd = 1;
for (i=1; i<10; i++) {
　fni = fn + fnd;
　fnd = fn;
　fn = fni;
}
return fn;
	n=1 時 fn=1, 即 f(1) = 1
	fnd 代表 f(n-1)，f(0) = 1
	for (i=0; i<10; i++)
	　fni = f(n+1) = f(n)+f(n-1) = fn + fnd
	　n 向前推移，n=n+1 時 fnd=fn
	　n 向前推移，n=n+1 時 fn=fni
	
	傳回 fn

12.2　請利用 c0c 編譯器, 將 fib.c0 編譯為組合語言 fib.asm0。
參考解答：

```
D:₩ch12>c0c fib.c0 fib.asm0

compile file:fib.c0

======= tokenize =======

token=fn

token==

token=1

token=;
```

```
token=fnd

token==

token=1

token=;

token=for

token=(

token=i

token==

token=1

token=;

token=i

token=<

token=10

token=;

token=i

token=++

token=)

token={

token=fni
```

```
token==

token=fn

token=+

token=fnd

token=;

token=fnd

token==

token=fn

token=;

token=fn

token==

token=fni

token=;

token=}

token=return

token=fn

token=;

tokens->count = 40

token=fn , type=id
```

token== , type==

token=1 , type=number

token=; , type=;

token=fnd , type=id

token== , type==

token=1 , type=number

token=; , type=;

token=for , type=for

token=(, type=(

token=i , type=id

token== , type==

token=1 , type=number

token=; , type=;

token=i , type=id

token=< , type=<

token=10 , type=number

token=; , type=;

token=i , type=id

token=++ , type=++

token=) , type=)

token={ , type={

token=fni , type=id

token== , type==

token=fn , type=id

token=+ , type=+

token=fnd , type=id

token=; , type=;

token=fnd , type=id

token== , type==

token=fn , type=id

token=; , type=;

token=fn , type=id

token== , type==

token=fni , type=id

token=; , type=;

token=} , type=}

token=return , type=return

token=fn , type=id

```
token=; , type=;

======= parsing ========

+PROG

 +BaseList

  +BASE

   +STMT

     idx=0, token=fn, type=id

     idx=1, token==, type==

    +EXP

      idx=2, token=1, type=number

    -EXP

   -STMT

    idx=3, token=;, type=;

  -BASE

  +BASE

   +STMT

     idx=4, token=fnd, type=id

     idx=5, token==, type==

    +EXP
```

```
      idx=6, token=1, type=number
   -EXP
  -STMT
   idx=7, token=;, type=;
-BASE
+BASE
 +FOR
    idx=8, token=for, type=for
    idx=9, token=(, type=(
   +STMT
      idx=10, token=i, type=id
      idx=11, token==, type==
    +EXP
       idx=12, token=1, type=number
    -EXP
   -STMT
    idx=13, token=;, type=;
   +COND
    +EXP
```

```
    idx=14, token=i, type=id
  -EXP
   idx=15, token=<, type=<
  +EXP
    idx=16, token=10, type=number
  -EXP
-COND
  idx=17, token=;, type=;
+STMT
   idx=18, token=i, type=id
   idx=19, token=++, type=++
-STMT
  idx=20, token=), type=)
+BLOCK
   idx=21, token={, type={
  +BaseList
   +BASE
    +STMT
      idx=22, token=fni, type=id
```

```
    idx=23, token==, type==
   +EXP
     idx=24, token=fn, type=id
     idx=25, token=+, type=+
     idx=26, token=fnd, type=id
   -EXP
  -STMT
   idx=27, token=;, type=;
-BASE
+BASE
  +STMT
    idx=28, token=fnd, type=id
    idx=29, token==, type==
   +EXP
     idx=30, token=fn, type=id
   -EXP
  -STMT
   idx=31, token=;, type=;
-BASE
```

```
      +BASE
       +STMT
          idx=32, token=fn, type=id
          idx=33, token==, type==
         +EXP
            idx=34, token=fni, type=id
         -EXP
        -STMT
          idx=35, token=;, type=;
      -BASE
     -BaseList
      idx=36, token=}, type=}
   -BLOCK
  -FOR
-BASE
+BASE
 +STMT
    idx=37, token=return, type=return
    idx=38, token=fn, type=id
```

```
    -STMT
      idx=39, token=;, type=;
   -BASE
  -BaseList
-PROG
=====PCODE=====
            =       1               fn
            =       1               fnd
            =       1               i
FOR0:
            CMP    i       10
            J       >=              _FOR0
            +       fn      fnd     T0
            =       T0              fni
            =       fn              fnd
            =       fni             fn
            +       i       1       i
            J                       FOR0
_FOR0:
```

```
            RET                 fn

=====AsmFile:fib.asm0======

            LDI   R1    1

            ST    R1    fn

            LDI   R1    1

            ST    R1    fnd

            LDI   R1    1

            ST    R1    i

FOR0:

            LD    R1    i

            LDI   R2    10

            CMP   R1    R2

            JGE   _FOR0

            LD    R1    fn

            LD    R2    fnd

            ADD   R3    R1    R2

            ST    R3    T0

            LD    R1    T0

            ST    R1    fni
```

```
            LD      R1      fn
            ST      R1      fnd
            LD      R1      fni
            ST      R1      fn
            LD      R1      i
            LDI     R2      1
            ADD     R3      R1      R2
            ST      R3      i
            JMP     FOR0
_FOR0:
            LD      R1      fn
            RET
fnd:        RESW 1
fni:        RESW 1
fn:         RESW 1
i:          RESW 1
T0:         RESW 1
```

12.3　請利用 as0 組譯器, 將 fib.asm0 組譯為目的檔 fib.obj0。

參考解答：

```
D:₩ch12>as0 fib.asm0 fib.obj0

Assembler:asmFile=fib.asm0 objFile=fib.obj0

===============Assemble=============

            LDI   R1    1

            ST    R1    fn

            LDI   R1    1

            ST    R1    fnd

            LDI   R1    1

            ST    R1    i

FOR0:

            LD    R1    i

            LDI   R2    10

            CMP   R1    R2

            JGE   _FOR0

            LD    R1    fn

            LD    R2    fnd

            ADD   R3    R1    R2

            ST    R3    T0

            LD    R1    T0
```

```
            ST      R1      fni

            LD      R1      fn

            ST      R1      fnd

            LD      R1      fni

            ST      R1      fn

            LD      R1      i

            LDI     R2      1

            ADD     R3      R1      R2

            ST      R3      i

            JMP     FOR0

_FOR0:

            LD      R1      fn

            RET

fnd:        RESW 1

fni:        RESW 1

fn:         RESW 1

i:          RESW 1

T0:         RESW 1

==================PASS1================
```

```
0000            LDI   R1    1           L   8 (NULL)
0004            ST    R1    FN          L   1 (NULL)
0008            LDI   R1    1           L   8 (NULL)
000C            ST    R1    FND         L   1 (NULL)
0010            LDI   R1    1           L   8 (NULL)
0014            ST    R1    I           L   1 (NULL)
0018 FOR0:                              FF (NULL)
0018            LD    R1    I           L   0 (NULL)
001C            LDI   R2    10          L   8 (NULL)
0020            CMP   R1    R2          A 10 (NULL)
0024            JGE   _FOR0             J 25 (NULL)
0028            LD    R1    FN          L   0 (NULL)
002C            LD    R2    FND         L   0 (NULL)
0030            ADD   R3    R1    R2    A 13 (NULL)
0034            ST    R3    T0          L   1 (NULL)
0038            LD    R1    T0          L   0 (NULL)
003C            ST    R1    FNI         L   1 (NULL)
0040            LD    R1    FN          L   0 (NULL)
0044            ST    R1    FND         L   1 (NULL)
```

```
0048              LD     R1     FNI          L    0 (NULL)

004C              ST     R1     FN           L    1 (NULL)

0050              LD     R1     I            L    0 (NULL)

0054              LDI    R2     1            L    8 (NULL)

0058              ADD    R3     R1     R2    A 13 (NULL)

005C              ST     R3     I            L    1 (NULL)

0060              JMP    FOR0                J 26 (NULL)

0064 _FOR0:                                  FF (NULL)

0064              LD     R1     FN           L    0 (NULL)

0068              RET                        J 2C (NULL)

006C FND:         RESW 1                     D F0 (NULL)

0070 FNI:         RESW 1                     D F0 (NULL)

0074 FN:          RESW 1                     D F0 (NULL)

0078 I:           RESW 1                     D F0 (NULL)

007C T0:          RESW 1                     D F0 (NULL)

===============SYMBOL TABLE==========

007C T0:          RESW 1                     D F0 (NULL)

0018 FOR0:                                   FF (NULL)

006C FND:         RESW 1                     D F0 (NULL)
```

```
0070 FNI:         RESW 1                    D F0 (NULL)

0074 FN:          RESW 1                    D F0 (NULL)

0078 I:           RESW 1                    D F0 (NULL)

0064 _FOR0:                                 FF (NULL)

==============PASS2==============

0000              LDI    R1     1           L    8 08100001

0004              ST     R1     FN          L    1 011F006C

0008              LDI    R1     1           L    8 08100001

000C              ST     R1     FND         L    1 011F005C

0010              LDI    R1     1           L    8 08100001

0014              ST     R1     I           L    1 011F0060

0018 FOR0:                                  FF

0018              LD     R1     I           L    0 001F005C

001C              LDI    R2     10          L    8 0820000A

0020              CMP    R1     R2          A 10 10120000

0024              JGE    _FOR0              J 25 2500003C

0028              LD     R1     FN          L    0 001F0048

002C              LD     R2     FND         L    0 002F003C

0030              ADD    R3     R1     R2   A 13 13312000
```

```
0034              ST     R3     T0            L    1 013F0044

0038              LD     R1     T0            L    0 001F0040

003C              ST     R1     FNI           L    1 011F0030

0040              LD     R1     FN            L    0 001F0030

0044              ST     R1     FND           L    1 011F0024

0048              LD     R1     FNI           L    0 001F0024

004C              ST     R1     FN            L    1 011F0024

0050              LD     R1     I             L    0 001F0024

0054              LDI    R2     1             L    8 08200001

0058              ADD    R3     R1     R2     A 13 13312000

005C              ST     R3     I             L    1 013F0018

0060              JMP    FOR0                 J 26 26FFFFB4

0064 _FOR0:                                   FF

0064              LD     R1     FN            L    0 001F000C

0068              RET                         J 2C 2C000000

006C FND:         RESW 1                      D F0 00000000

0070 FNI:         RESW 1                      D F0 00000000

0074 FN:          RESW 1                      D F0 00000000

0078 I:           RESW 1                      D F0 00000000
```

```
007C T0:          RESW 1                    D F0 00000000

==========Save to ObjFile:fib.obj0==========

08100001011F006C08100001011F005C08100001011F0060001F005C0820000A1012

00002500003C

001F0048002F003C13312000013F0044001F0040011F0030001F0030011F0024001

F0024011F0024

001F00240820000113312000013F001826FFFFB4001F000C2C0000000000000000

00000000000000

0000000000000000
```

12.4 請利用 vm0 虛擬機, 執行 fib.obj0, 並檢查看看 f(10) 的結果是否正確。

參考解答：

```
D:₩ch12>vm0 fib.obj0

===VM0:run fib.obj0 on CPU0===

PC=00000004 IR=08100001 SW=00000000 R[01]=0X00000001=1

PC=00000008 IR=011F006C SW=00000000 R[01]=0X00000001=1

PC=0000000C IR=08100001 SW=00000000 R[01]=0X00000001=1

PC=00000010 IR=011F005C SW=00000000 R[01]=0X00000001=1

PC=00000014 IR=08100001 SW=00000000 R[01]=0X00000001=1

PC=00000018 IR=011F0060 SW=00000000 R[01]=0X00000001=1
```

```
PC=0000001C IR=001F005C SW=00000000 R[01]=0X00000001=1

PC=00000020 IR=0820000A SW=00000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648

PC=00000028 IR=2500003C SW=80000000 R[00]=0X00000000=0

PC=0000002C IR=001F0048 SW=80000000 R[01]=0X00000001=1

PC=00000030 IR=002F003C SW=80000000 R[02]=0X00000001=1

PC=00000034 IR=13312000 SW=80000000 R[03]=0X00000002=2

PC=00000038 IR=013F0044 SW=80000000 R[03]=0X00000002=2

PC=0000003C IR=001F0040 SW=80000000 R[01]=0X00000002=2

PC=00000040 IR=011F0030 SW=80000000 R[01]=0X00000002=2

PC=00000044 IR=001F0030 SW=80000000 R[01]=0X00000001=1

PC=00000048 IR=011F0024 SW=80000000 R[01]=0X00000001=1

PC=0000004C IR=001F0024 SW=80000000 R[01]=0X00000002=2

PC=00000050 IR=011F0024 SW=80000000 R[01]=0X00000002=2

PC=00000054 IR=001F0024 SW=80000000 R[01]=0X00000001=1

PC=00000058 IR=08200001 SW=80000000 R[02]=0X00000001=1

PC=0000005C IR=13312000 SW=80000000 R[03]=0X00000002=2

PC=00000060 IR=013F0018 SW=80000000 R[03]=0X00000002=2

PC=00000018 IR=26FFFFB4 SW=80000000 R[15]=0X00000018=24
```

```
PC=0000001C IR=001F005C SW=80000000 R[01]=0X00000002=2

PC=00000020 IR=0820000A SW=80000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648

PC=00000028 IR=2500003C SW=80000000 R[00]=0X00000000=0

PC=0000002C IR=001F0048 SW=80000000 R[01]=0X00000002=2

PC=00000030 IR=002F003C SW=80000000 R[02]=0X00000001=1

PC=00000034 IR=13312000 SW=80000000 R[03]=0X00000003=3

PC=00000038 IR=013F0044 SW=80000000 R[03]=0X00000003=3

PC=0000003C IR=001F0040 SW=80000000 R[01]=0X00000003=3

PC=00000040 IR=011F0030 SW=80000000 R[01]=0X00000003=3

PC=00000044 IR=001F0030 SW=80000000 R[01]=0X00000002=2

PC=00000048 IR=011F0024 SW=80000000 R[01]=0X00000002=2

PC=0000004C IR=001F0024 SW=80000000 R[01]=0X00000003=3

PC=00000050 IR=011F0024 SW=80000000 R[01]=0X00000003=3

PC=00000054 IR=001F0024 SW=80000000 R[01]=0X00000002=2

PC=00000058 IR=08200001 SW=80000000 R[02]=0X00000001=1

PC=0000005C IR=13312000 SW=80000000 R[03]=0X00000003=3

PC=00000060 IR=013F0018 SW=80000000 R[03]=0X00000003=3

PC=00000018 IR=26FFFFB4 SW=80000000 R[15]=0X00000018=24
```

```
PC=0000001C IR=001F005C SW=80000000 R[01]=0X00000003=3

PC=00000020 IR=0820000A SW=80000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648

PC=00000028 IR=2500003C SW=80000000 R[00]=0X00000000=0

PC=0000002C IR=001F0048 SW=80000000 R[01]=0X00000003=3

PC=00000030 IR=002F003C SW=80000000 R[02]=0X00000002=2

PC=00000034 IR=13312000 SW=80000000 R[03]=0X00000005=5

PC=00000038 IR=013F0044 SW=80000000 R[03]=0X00000005=5

PC=0000003C IR=001F0040 SW=80000000 R[01]=0X00000005=5

PC=00000040 IR=011F0030 SW=80000000 R[01]=0X00000005=5

PC=00000044 IR=001F0030 SW=80000000 R[01]=0X00000003=3

PC=00000048 IR=011F0024 SW=80000000 R[01]=0X00000003=3

PC=0000004C IR=001F0024 SW=80000000 R[01]=0X00000005=5

PC=00000050 IR=011F0024 SW=80000000 R[01]=0X00000005=5

PC=00000054 IR=001F0024 SW=80000000 R[01]=0X00000003=3

PC=00000058 IR=08200001 SW=80000000 R[02]=0X00000001=1

PC=0000005C IR=13312000 SW=80000000 R[03]=0X00000004=4

PC=00000060 IR=013F0018 SW=80000000 R[03]=0X00000004=4

PC=00000018 IR=26FFFFB4 SW=80000000 R[15]=0X00000018=24
```

```
PC=0000001C IR=001F005C SW=80000000 R[01]=0X00000004=4

PC=00000020 IR=0820000A SW=80000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648

PC=00000028 IR=2500003C SW=80000000 R[00]=0X00000000=0

PC=0000002C IR=001F0048 SW=80000000 R[01]=0X00000005=5

PC=00000030 IR=002F003C SW=80000000 R[02]=0X00000003=3

PC=00000034 IR=13312000 SW=80000000 R[03]=0X00000008=8

PC=00000038 IR=013F0044 SW=80000000 R[03]=0X00000008=8

PC=0000003C IR=001F0040 SW=80000000 R[01]=0X00000008=8

PC=00000040 IR=011F0030 SW=80000000 R[01]=0X00000008=8

PC=00000044 IR=001F0030 SW=80000000 R[01]=0X00000005=5

PC=00000048 IR=011F0024 SW=80000000 R[01]=0X00000005=5

PC=0000004C IR=001F0024 SW=80000000 R[01]=0X00000008=8

PC=00000050 IR=011F0024 SW=80000000 R[01]=0X00000008=8

PC=00000054 IR=001F0024 SW=80000000 R[01]=0X00000004=4

PC=00000058 IR=08200001 SW=80000000 R[02]=0X00000001=1

PC=0000005C IR=13312000 SW=80000000 R[03]=0X00000005=5

PC=00000060 IR=013F0018 SW=80000000 R[03]=0X00000005=5

PC=00000018 IR=26FFFFB4 SW=80000000 R[15]=0X00000018=24
```

PC=0000001C IR=001F005C SW=80000000 R[01]=0X00000005=5

PC=00000020 IR=0820000A SW=80000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648

PC=00000028 IR=2500003C SW=80000000 R[00]=0X00000000=0

PC=0000002C IR=001F0048 SW=80000000 R[01]=0X00000008=8

PC=00000030 IR=002F003C SW=80000000 R[02]=0X00000005=5

PC=00000034 IR=13312000 SW=80000000 R[03]=0X0000000D=13

PC=00000038 IR=013F0044 SW=80000000 R[03]=0X0000000D=13

PC=0000003C IR=001F0040 SW=80000000 R[01]=0X0000000D=13

PC=00000040 IR=011F0030 SW=80000000 R[01]=0X0000000D=13

PC=00000044 IR=001F0030 SW=80000000 R[01]=0X00000008=8

PC=00000048 IR=011F0024 SW=80000000 R[01]=0X00000008=8

PC=0000004C IR=001F0024 SW=80000000 R[01]=0X0000000D=13

PC=00000050 IR=011F0024 SW=80000000 R[01]=0X0000000D=13

PC=00000054 IR=001F0024 SW=80000000 R[01]=0X00000005=5

PC=00000058 IR=08200001 SW=80000000 R[02]=0X00000001=1

PC=0000005C IR=13312000 SW=80000000 R[03]=0X00000006=6

PC=00000060 IR=013F0018 SW=80000000 R[03]=0X00000006=6

PC=00000018 IR=26FFFFB4 SW=80000000 R[15]=0X00000018=24

```
PC=0000001C IR=001F005C SW=80000000 R[01]=0X00000006=6

PC=00000020 IR=0820000A SW=80000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648

PC=00000028 IR=2500003C SW=80000000 R[00]=0X00000000=0

PC=0000002C IR=001F0048 SW=80000000 R[01]=0X0000000D=13

PC=00000030 IR=002F003C SW=80000000 R[02]=0X00000008=8

PC=00000034 IR=13312000 SW=80000000 R[03]=0X00000015=21

PC=00000038 IR=013F0044 SW=80000000 R[03]=0X00000015=21

PC=0000003C IR=001F0040 SW=80000000 R[01]=0X00000015=21

PC=00000040 IR=011F0030 SW=80000000 R[01]=0X00000015=21

PC=00000044 IR=001F0030 SW=80000000 R[01]=0X0000000D=13

PC=00000048 IR=011F0024 SW=80000000 R[01]=0X0000000D=13

PC=0000004C IR=001F0024 SW=80000000 R[01]=0X00000015=21

PC=00000050 IR=011F0024 SW=80000000 R[01]=0X00000015=21

PC=00000054 IR=001F0024 SW=80000000 R[01]=0X00000006=6

PC=00000058 IR=08200001 SW=80000000 R[02]=0X00000001=1

PC=0000005C IR=13312000 SW=80000000 R[03]=0X00000007=7

PC=00000060 IR=013F0018 SW=80000000 R[03]=0X00000007=7

PC=00000018 IR=26FFFFB4 SW=80000000 R[15]=0X00000018=24
```

```
PC=0000001C IR=001F005C SW=80000000 R[01]=0X00000007=7

PC=00000020 IR=0820000A SW=80000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648

PC=00000028 IR=2500003C SW=80000000 R[00]=0X00000000=0

PC=0000002C IR=001F0048 SW=80000000 R[01]=0X00000015=21

PC=00000030 IR=002F003C SW=80000000 R[02]=0X0000000D=13

PC=00000034 IR=13312000 SW=80000000 R[03]=0X00000022=34

PC=00000038 IR=013F0044 SW=80000000 R[03]=0X00000022=34

PC=0000003C IR=001F0040 SW=80000000 R[01]=0X00000022=34

PC=00000040 IR=011F0030 SW=80000000 R[01]=0X00000022=34

PC=00000044 IR=001F0030 SW=80000000 R[01]=0X00000015=21

PC=00000048 IR=011F0024 SW=80000000 R[01]=0X00000015=21

PC=0000004C IR=001F0024 SW=80000000 R[01]=0X00000022=34

PC=00000050 IR=011F0024 SW=80000000 R[01]=0X00000022=34

PC=00000054 IR=001F0024 SW=80000000 R[01]=0X00000007=7

PC=00000058 IR=08200001 SW=80000000 R[02]=0X00000001=1

PC=0000005C IR=13312000 SW=80000000 R[03]=0X00000008=8

PC=00000060 IR=013F0018 SW=80000000 R[03]=0X00000008=8

PC=00000018 IR=26FFFFB4 SW=80000000 R[15]=0X00000018=24
```

```
PC=0000001C IR=001F005C SW=80000000 R[01]=0X00000008=8

PC=00000020 IR=0820000A SW=80000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648

PC=00000028 IR=2500003C SW=80000000 R[00]=0X00000000=0

PC=0000002C IR=001F0048 SW=80000000 R[01]=0X00000022=34

PC=00000030 IR=002F003C SW=80000000 R[02]=0X00000015=21

PC=00000034 IR=13312000 SW=80000000 R[03]=0X00000037=55

PC=00000038 IR=013F0044 SW=80000000 R[03]=0X00000037=55

PC=0000003C IR=001F0040 SW=80000000 R[01]=0X00000037=55

PC=00000040 IR=011F0030 SW=80000000 R[01]=0X00000037=55

PC=00000044 IR=001F0030 SW=80000000 R[01]=0X00000022=34

PC=00000048 IR=011F0024 SW=80000000 R[01]=0X00000022=34

PC=0000004C IR=001F0024 SW=80000000 R[01]=0X00000037=55

PC=00000050 IR=011F0024 SW=80000000 R[01]=0X00000037=55

PC=00000054 IR=001F0024 SW=80000000 R[01]=0X00000008=8

PC=00000058 IR=08200001 SW=80000000 R[02]=0X00000001=1

PC=0000005C IR=13312000 SW=80000000 R[03]=0X00000009=9

PC=00000060 IR=013F0018 SW=80000000 R[03]=0X00000009=9

PC=00000018 IR=26FFFFB4 SW=80000000 R[15]=0X00000018=24
```

```
PC=0000001C IR=001F005C SW=80000000 R[01]=0X00000009=9

PC=00000020 IR=0820000A SW=80000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648

PC=00000028 IR=2500003C SW=80000000 R[00]=0X00000000=0

PC=0000002C IR=001F0048 SW=80000000 R[01]=0X00000037=55

PC=00000030 IR=002F003C SW=80000000 R[02]=0X00000022=34

PC=00000034 IR=13312000 SW=80000000 R[03]=0X00000059=89

PC=00000038 IR=013F0044 SW=80000000 R[03]=0X00000059=89

PC=0000003C IR=001F0040 SW=80000000 R[01]=0X00000059=89

PC=00000040 IR=011F0030 SW=80000000 R[01]=0X00000059=89

PC=00000044 IR=001F0030 SW=80000000 R[01]=0X00000037=55

PC=00000048 IR=011F0024 SW=80000000 R[01]=0X00000037=55

PC=0000004C IR=001F0024 SW=80000000 R[01]=0X00000059=89

PC=00000050 IR=011F0024 SW=80000000 R[01]=0X00000059=89

PC=00000054 IR=001F0024 SW=80000000 R[01]=0X00000009=9

PC=00000058 IR=08200001 SW=80000000 R[02]=0X00000001=1

PC=0000005C IR=13312000 SW=80000000 R[03]=0X0000000A=10

PC=00000060 IR=013F0018 SW=80000000 R[03]=0X0000000A=10

PC=00000018 IR=26FFFFB4 SW=80000000 R[15]=0X00000018=24
```

```
PC=0000001C IR=001F005C SW=80000000 R[01]=0X0000000A=10

PC=00000020 IR=0820000A SW=80000000 R[02]=0X0000000A=10

PC=00000024 IR=10120000 SW=40000000 R[12]=0X40000000=1073741824

PC=00000064 IR=2500003C SW=40000000 R[00]=0X00000000=0

PC=00000068 IR=001F000C SW=40000000 R[01]=0X00000059=89

PC=0000006C IR=2C000000 SW=40000000 R[00]=0X00000000=0



===CPU0 dump registers===

IR =0x2c000000=738197504

R[00]=0x00000000=0

R[01]=0x00000059=89

R[02]=0x0000000a=10

R[03]=0x0000000a=10

R[04]=0x00000000=0

R[05]=0x00000000=0

R[06]=0x00000000=0

R[07]=0x00000000=0

R[08]=0x00000000=0

R[09]=0x00000000=0
```

```
R[10]=0x00000000=0

R[11]=0x00000000=0

R[12]=0x40000000=1073741824

R[13]=0x00000000=0

R[14]=0xffffffff=-1

R[15]=0x0000006c=108
```

12.5　請為 C0 語言加上 if 條件的規則為 IF = 'if' '(' COND ')' BASE '(' 'elseif' '(' COND ')' BASE)* ('else' BASE)?，然後修改本章的剖析器程式，加入可以處理該規則的程式。

參考解答：從這題開始的題目是連貫的，可以作為期末作業出給學生實作。

首先請修改 Scanner.c，將 elseif 與 else 等兩個詞彙加入到關鍵字變數 KEYWORDS 當中，如下所示。

```
char KEYWORDS[] = "|if|elseif|else|for|while|return|";
```

接著請修改 Parser.c，加入 parseIf 函數，並修改 parseBase()以驅動 parseIf()

```
// 剖析 IF = 'if' '(' COND ')' BASE ('elseif' '(' COND ')' BASE)* ('else' BASE)?
void parseIf(Parser *p) {
   push(p, "IF");                          // 建立 IF 的樹根
   next(p, "if");                          // 取得 if
   next(p, "(");                           // 取得 (
   parseCond(p);                           // 剖析 COND
   next(p, ")");                           // 取得 )
   parseBase(p);                           // 剖析 BASE
   while (isNext(p, "elseif")) {           // 連續處理 elseif
      next(p, "elseif");                   // 取得 elseif
      next(p, "(");                        // 取得 (
      parseCond(p);                        // 剖析 COND
```

```
        next(p, ")");                                   // 取得 )
        parseBase(p);                                   // 剖析 BASE
    }
    if (isNext(p, "else")) {                            // 如果有 else
        next(p, "else");                                // 取得 else
        parseBase(p);                                   // 剖析 BASE
    }
    pop(p, "IF");                                       // 取出 IF 的剖析樹
}

// 另外還必須修改 parseBase()，以加入 IF 規則。BASE = FOR | IF | STMT ';'
void parseBase(Parser *p) {                             // 剖析 BASE = FOR|IF|STMT 規則
    push(p, "BASE");
    if (isNext(p, "for"))                               // 建立 BASE 的樹根
        parseFor(p);                                    // 如果下一個詞彙是 for
    else if (isNext(p, "if"))                           // 建立 BASE 的樹根
        parseIf(p);                                     // 如果下一個詞彙是 for
    else {                                              //   根據 FOR 規則進行剖析
        parseStmt(p);                                   // 否則
        next(p, ";");                                   //   根據 STMT 規則進行剖析
    }
    pop(p, "BASE");                                     // 取出 BASE 的剖析樹
}
```

12.6 繼續前一題，請修改本章的程式碼產生器程式，以產生上述的 if 規則之程式

參考解答：

請為 Generator.h 中的 Generator 結構加上 ifCount，以計算 IF 節點的數量

```
typedef struct
{
        // 程式碼產生器物件
    HashTable *symTable;
//      符號表
    Tree *tree;
//      剖析樹
    FILE *asmFile;
```

```
//     輸出的 CPU0 組合語言檔
   int forCount, varCount, ifCount;
//     For 迴圈與臨時變數的數量
} Generator;
```

然後再修改 Generator.c 中的

```
Tree* GenCode(Generator *g, Tree *node, char *rzVar) {
…
   } else if (strEqual(node->type, "IF")) {                    // 處理 IF 節點
      // IF = 'if' '(' COND ')' BASE ('elseif' '(' COND ')' BASE)* ('else' BASE)?
      g->ifCount ++;
      char caseLabel[100], exitLabel[100], condOp[100], negOp[100];
      int i, caseCount=0;
      sprintf(exitLabel, "IF%dEXIT", g->ifCount);              // 離開標記
      sprintf(caseLabel, "IF%dCASE%d", g->ifCount, caseCount); // 第一個 if 的 case 標記
      for (i=0; i<node->childs->count-4; i+=5) {               // 連續處理 if 或 elseif 語句
         Tree *ifNode = node->childs->item[i];                 // 取得 if 節點
         if (strPartOf(node->type, "|if|elseif|")) break;      // 如果不是 if 或 elseif，跳開
         Tree *cond = node->childs->item[i+2];                 // 取得 COND 節點
         GenPcode(g, caseLabel, "", "", "", "");               // 中間碼：產生這一個 case 標記
         GenCode(g, cond, condOp);                             // 遞迴產生 COND
         negateOp(condOp, negOp);                              // 互補運算 negOp
         sprintf(caseLabel, "IF%dCASE%d", g->ifCount, ++caseCount); // 下一個 if 的 case 標記
         GenPcode(g, "", "J", negOp, "", caseLabel);           // 中間碼：例如 J < IF1CASE1
         Tree *base = node->childs->item[i+4];                 // 取得 BASE 部分
         GenCode(g, base, nullVar);                            // 遞迴產生 BASE
         GenPcode(g, "", "J", "", "", exitLabel);              // 中間碼：例如 J IF1EXIT
      }
      if (i < node->childs->count) {                           // 如果最後有 else
         Tree *elseNode    = node->childs->item[i];            // 取得 else 節點
         assert(strEqual(elseNode->type, "else"));             // 確認該節點是 else
         Tree *base    = node->childs->item[i+1];              // 取得 base 節點
         GenPcode(g, caseLabel, "", "", "", "");               // 中間碼：例如 J < IF1CASE4
         GenCode(g, base, nullVar);                            // 展開 base
      }
      GenPcode(g, exitLabel, "", "", "", "");                  // 離開標記：例如 IF1EXIT
      return NULL;
```

當您完成上述程式碼之後，可以撰寫測試程式，以對這個修改進行測試，以下是筆者所使用的一個測試程式 – ifTest.c0

```
score = 85;
if (score >= 90)
   degree = 1;
elseif (score >= 80)
   degree = 2;
elseif (score >= 70)
   degree = 3;
elseif (score >= 60)
   degree = 4;
else
   degree = 5;
return degree;
```

然後，您可以使用下列指令，對 ifTest.c0 進行編譯、組譯、執行等動作，其結果如下所示。

```
D:₩ch12>c0c ifTest.c0 ifTest.asm0

compile file:ifTest.c0

======= tokenize =======

token=score

token==

token=85

token=;

token=if

token=(
```

```
token=score
token=>=
token=90
token=)
token=degree
token==
token=1
token=;
token=elseif
token=(
token=score
token=>=
token=80
token=)
token=degree
token==
token=2
token=;
token=elseif
```

```
token=(

token=score

token=>=

token=70

token=)

token=degree

token==

token=3

token=;

token=elseif

token=(

token=score

token=>=

token=60

token=)

token=degree

token==

token=4

token=;
```

```
token=else

token=degree

token==

token=5

token=;

token=return

token=degree

token=;

tokens->count = 52

token=score , type=id

token== , type==

token=85 , type=number

token=; , type=;

token=if , type=if

token=( , type=(

token=score , type=id

token=>= , type=>=

token=90 , type=number

token=) , type=)
```

token=degree , type=id

token== , type==

token=1 , type=number

token=; , type=;

token=elseif , type=elseif

token=(, type=(

token=score , type=id

token=>= , type=>=

token=80 , type=number

token=) , type=)

token=degree , type=id

token== , type==

token=2 , type=number

token=; , type=;

token=elseif , type=elseif

token=(, type=(

token=score , type=id

token=>= , type=>=

token=70 , type=number

token=) , type=)

token=degree , type=id

token== , type==

token=3 , type=number

token=; , type=;

token=elseif , type=elseif

token=(, type=(

token=score , type=id

token=>= , type=>=

token=60 , type=number

token=) , type=)

token=degree , type=id

token== , type==

token=4 , type=number

token=; , type=;

token=else , type=else

token=degree , type=id

token== , type==

token=5 , type=number

```
token=; , type=;

token=return , type=return

token=degree , type=id

token=; , type=;

======= parsing ========

+PROG

 +BaseList

  +BASE

   +STMT

     idx=0, token=score, type=id

     idx=1, token==, type==

    +EXP

      idx=2, token=85, type=number

    -EXP

   -STMT

    idx=3, token=;, type=;

  -BASE

  +BASE

   +IF
```

```
 idx=4, token=if, type=if
 idx=5, token=(, type=(
+COND
 +EXP
    idx=6, token=score, type=id
 -EXP
  idx=7, token=>=, type=>=
 +EXP
    idx=8, token=90, type=number
 -EXP
-COND
 idx=9, token=), type=)
+BASE
 +STMT
    idx=10, token=degree, type=id
    idx=11, token==, type==
   +EXP
      idx=12, token=1, type=number
   -EXP
```

```
  -STMT
    idx=13, token=;, type=;
-BASE
  idx=14, token=elseif, type=elseif
  idx=15, token=(, type=(
+COND
  +EXP
     idx=16, token=score, type=id
  -EXP
    idx=17, token=>=, type=>=
  +EXP
     idx=18, token=80, type=number
  -EXP
-COND
  idx=19, token=), type=)
+BASE
  +STMT
     idx=20, token=degree, type=id
     idx=21, token==, type==
```

```
    +EXP
      idx=22, token=2, type=number
    -EXP
  -STMT
    idx=23, token=;, type=;
-BASE
  idx=24, token=elseif, type=elseif
  idx=25, token=(, type=(
+COND
  +EXP
    idx=26, token=score, type=id
  -EXP
    idx=27, token=>=, type=>=
  +EXP
    idx=28, token=70, type=number
  -EXP
-COND
  idx=29, token=), type=)
+BASE
```

```
 +STMT
    idx=30, token=degree, type=id
    idx=31, token==, type==
   +EXP
     idx=32, token=3, type=number
   -EXP
 -STMT
  idx=33, token=;, type=;
-BASE
 idx=34, token=elseif, type=elseif
 idx=35, token=(, type=(
+COND
 +EXP
    idx=36, token=score, type=id
 -EXP
  idx=37, token=>=, type=>=
 +EXP
    idx=38, token=60, type=number
 -EXP
```

```
-COND
 idx=39, token=), type=)
+BASE
 +STMT
    idx=40, token=degree, type=id
    idx=41, token==, type==
   +EXP
      idx=42, token=4, type=number
   -EXP
 -STMT
   idx=43, token=;, type=;
-BASE
 idx=44, token=else, type=else
+BASE
 +STMT
    idx=45, token=degree, type=id
    idx=46, token==, type==
   +EXP
      idx=47, token=5, type=number
```

```
        -EXP
       -STMT
        idx=48, token=;, type=;
     -BASE
    -IF
   -BASE
   +BASE
    +STMT
      idx=49, token=return, type=return
      idx=50, token=degree, type=id
    -STMT
     idx=51, token=;, type=;
   -BASE
  -BaseList
-PROG
=====PCODE=====
            =       85              score
IF1CASE0:
            CMP    score 90
```

```
        J       <       IF1CASE1
        =       1       degree
        J               IF1EXIT
IF1CASE1:
        CMP     score 80
        J       <       IF1CASE2
        =       2       degree
        J               IF1EXIT
IF1CASE2:
        CMP     score 70
        J       <       IF1CASE3
        =       3       degree
        J               IF1EXIT
IF1CASE3:
        CMP     score 60
        J       <       IF1CASE4
        =       4       degree
        J               IF1EXIT
IF1CASE4:
```

```
            =       5               degree

IF1EXIT:

            RET                     degree

=====AsmFile:ifTest.asm0======

            LDI    R1     85

            ST     R1     score

IF1CASE0:

            LD     R1     score

            LDI    R2     90

            CMP    R1     R2

            JLT    IF1CASE1

            LDI    R1     1

            ST     R1     degree

            JMP    IF1EXIT

IF1CASE1:

            LD     R1     score

            LDI    R2     80

            CMP    R1     R2

            JLT    IF1CASE2
```

```
        LDI   R1    2
        ST    R1    degree
        JMP   IF1EXIT
IF1CASE2:
        LD    R1    score
        LDI   R2    70
        CMP   R1    R2
        JLT   IF1CASE3
        LDI   R1    3
        ST    R1    degree
        JMP   IF1EXIT
IF1CASE3:
        LD    R1    score
        LDI   R2    60
        CMP   R1    R2
        JLT   IF1CASE4
        LDI   R1    4
        ST    R1    degree
        JMP   IF1EXIT
```

```
IF1CASE4:
            LDI   R1    5
            ST    R1    degree
IF1EXIT:
            LD    R1    degree
            RET
score:    RESW 1
degree:   RESW 1
D:₩ch12>as0 ifTest.asm0 ifTest.obj0
Assembler:asmFile=ifTest.asm0 objFile=ifTest.obj0
===============Assemble============
            LDI   R1    85
            ST    R1    score
IF1CASE0:
            LD    R1    score
            LDI   R2    90
            CMP   R1    R2
            JLT   IF1CASE1
            LDI   R1    1
```

```
            ST      R1      degree
            JMP     IF1EXIT
IF1CASE1:
            LD      R1      score
            LDI     R2      80
            CMP     R1      R2
            JLT     IF1CASE2
            LDI     R1      2
            ST      R1      degree
            JMP     IF1EXIT
IF1CASE2:
            LD      R1      score
            LDI     R2      70
            CMP     R1      R2
            JLT     IF1CASE3
            LDI     R1      3
            ST      R1      degree
            JMP     IF1EXIT
IF1CASE3:
```

```
            LD      R1      score

            LDI     R2      60

            CMP     R1      R2

            JLT     IF1CASE4

            LDI     R1      4

            ST      R1      degree

            JMP     IF1EXIT

IF1CASE4:

            LDI     R1      5

            ST      R1      degree

IF1EXIT:

            LD      R1      degree

            RET

score:      RESW 1

degree:     RESW 1

==================PASS1================

0000                LDI     R1      85              L    8 (NULL)

0004                ST      R1      SCORE           L    1 (NULL)

0008 IF1CASE0:                                      FF (NULL)
```

```
0008              LD     R1     SCORE        L    0 (NULL)

000C              LDI    R2     90           L    8 (NULL)

0010              CMP    R1     R2           A 10 (NULL)

0014              JLT    IF1CASE1            J 22 (NULL)

0018              LDI    R1     1            L    8 (NULL)

001C              ST     R1     DEGREE       L    1 (NULL)

0020              JMP    IF1EXIT             J 26 (NULL)

0024 IF1CASE1:                               FF (NULL)

0024              LD     R1     SCORE        L    0 (NULL)

0028              LDI    R2     80           L    8 (NULL)

002C              CMP    R1     R2           A 10 (NULL)

0030              JLT    IF1CASE2            J 22 (NULL)

0034              LDI    R1     2            L    8 (NULL)

0038              ST     R1     DEGREE       L    1 (NULL)

003C              JMP    IF1EXIT             J 26 (NULL)

0040 IF1CASE2:                               FF (NULL)

0040              LD     R1     SCORE        L    0 (NULL)

0044              LDI    R2     70           L    8 (NULL)

0048              CMP    R1     R2           A 10 (NULL)
```

```
004C              JLT    IF1CASE3          J 22 (NULL)

0050              LDI    R1     3          L    8 (NULL)

0054              ST     R1     DEGREE     L    1 (NULL)

0058              JMP    IF1EXIT           J 26 (NULL)

005C IF1CASE3:                             FF (NULL)

005C              LD     R1     SCORE      L    0 (NULL)

0060              LDI    R2     60         L    8 (NULL)

0064              CMP    R1     R2         A 10 (NULL)

0068              JLT    IF1CASE4          J 22 (NULL)

006C              LDI    R1     4          L    8 (NULL)

0070              ST     R1     DEGREE     L    1 (NULL)

0074              JMP    IF1EXIT           J 26 (NULL)

0078 IF1CASE4:                             FF (NULL)

0078              LDI    R1     5          L    8 (NULL)

007C              ST     R1     DEGREE     L    1 (NULL)

0080 IF1EXIT:                              FF (NULL)

0080              LD     R1     DEGREE     L    0 (NULL)

0084              RET                       J 2C (NULL)

0088 SCORE:       RESW 1                    D F0 (NULL)
```

```
008C DEGREE:    RESW 1                    D F0 (NULL)

===============SYMBOL TABLE=========

0008 IF1CASE0:                            FF (NULL)

0024 IF1CASE1:                            FF (NULL)

0040 IF1CASE2:                            FF (NULL)

005C IF1CASE3:                            FF (NULL)

0078 IF1CASE4:                            FF (NULL)

0088 SCORE:     RESW 1                    D F0 (NULL)

008C DEGREE:    RESW 1                    D F0 (NULL)

0080 IF1EXIT:                             FF (NULL)

==============PASS2==============

0000            LDI    R1     85          L    8 08100055

0004            ST     R1     SCORE       L    1 011F0080

0008 IF1CASE0:                            FF

0008            LD     R1     SCORE       L    0 001F007C

000C            LDI    R2     90          L    8 0820005A

0010            CMP    R1     R2          A 10 10120000

0014            JLT    IF1CASE1          J 22 22 00000C

0018            LDI    R1     1           L    8 08100001
```

```
001C              ST     R1     DEGREE       L    1 011F006C

0020              JMP    IF1EXIT            J 26 2600005C

0024 IF1CASE1:                                FF

0024              LD     R1     SCORE        L    0 001F0060

0028              LDI    R2     80           L    8 08200050

002C              CMP    R1     R2           A 10 10120000

0030              JLT    IF1CASE2           J 22 2200000C

0034              LDI    R1     2            L    8 08100002

0038              ST     R1     DEGREE       L    1 011F0050

003C              JMP    IF1EXIT            J 26 26000040

0040 IF1CASE2:                                FF

0040              LD     R1     SCORE        L    0 001F0044

0044              LDI    R2     70           L    8 08200046

0048              CMP    R1     R2           A 10 10120000

004C              JLT    IF1CASE3           J 22 2200000C

0050              LDI    R1     3            L    8 08100003

0054              ST     R1     DEGREE       L    1 011F0034

0058              JMP    IF1EXIT            J 26 26000024

005C IF1CASE3:                                FF
```

```
005C              LD      R1      SCORE         L    0 001F0028

0060              LDI     R2      60            L    8 0820003C

0064              CMP     R1      R2            A 10 10120000

0068              JLT     IF1CASE4              J 22 2200000C

006C              LDI     R1      4             L    8 08100004

0070              ST      R1      DEGREE        L    1 011F0018

0074              JMP     IF1EXIT               J 26 26000008

0078 IF1CASE4:                                    FF

0078              LDI     R1      5             L    8 08100005

007C              ST      R1      DEGREE        L    1 011F000C

0080 IF1EXIT:                                     FF

0080              LD      R1      DEGREE        L    0 001F0008

0084              RET                           J 2C 2C000000

0088 SCORE:       RESW 1                        D F0 00000000

008C DEGREE:      RESW 1                        D F0 00000000

==========Save to ObjFile:ifTest.obj0==========

08100055011F0080001F007C0820005A101200002200000C08100001011F006C26

00005C001F006008200050101200002200000C08100002011F005026000040001F

00440820004610120000220000C08100003011F003426000024001F002808200
```

```
03C101200002200000C08100004011F001826000008081000005011F000C001F0008
2C00000000000000000000
```

```
D:₩ch12>vm0 ifTest.obj0
===VM0:run ifTest.obj0 on CPU0===
PC=00000004 IR=08100055 SW=00000000 R[01]=0X00000055=85
PC=00000008 IR=011F0080 SW=00000000 R[01]=0X00000055=85
PC=0000000C IR=001F007C SW=00000000 R[01]=0X00000055=85
PC=00000010 IR=0820005A SW=00000000 R[02]=0X0000005A=90
PC=00000014 IR=10120000 SW=80000000 R[12]=0X80000000=-2147483648
PC=00000024 IR=2200000C SW=80000000 R[00]=0X00000000=0
PC=00000028 IR=001F0060 SW=80000000 R[01]=0X00000055=85
PC=0000002C IR=08200050 SW=80000000 R[02]=0X00000050=80
PC=00000030 IR=10120000 SW=00000000 R[12]=0X00000000=0
PC=00000034 IR=2200000C SW=00000000 R[00]=0X00000000=0
PC=00000038 IR=08100002 SW=00000000 R[01]=0X00000002=2
PC=0000003C IR=011F0050 SW=00000000 R[01]=0X00000002=2
PC=00000080 IR=26000040 SW=00000000 R[00]=0X00000000=0
PC=00000084 IR=001F0008 SW=00000000 R[01]=0X00000002=2
```

```
PC=00000088 IR=2C000000 SW=00000000 R[00]=0X00000000=0



===CPU0 dump registers===

IR =0x2c000000=738197504

R[00]=0x00000000=0

R[01]=0x00000002=2

R[02]=0x00000050=80

R[03]=0x00000000=0

R[04]=0x00000000=0

R[05]=0x00000000=0

R[06]=0x00000000=0

R[07]=0x00000000=0

R[08]=0x00000000=0

R[09]=0x00000000=0

R[10]=0x00000000=0

R[11]=0x00000000=0

R[12]=0x00000000=0

R[13]=0x00000000=0

R[14]=0xffffffff=-1
```

```
R[15]=0x00000088=136
```

您可以看到最後 R[01]=0x00000002=2，代表該程式的返回值為 R1=2，也就是 degree = 2，這是正確的結果。