金門大學 資訊工程系 102 學年度下學期 系統程式 期末考 出題者：陳鍾誠

考生姓名：　　學號：　　得分：

所有題目都是奇數號同學做奇數題，偶數號同學做偶數題，做錯題目將不予計分！

名詞解釋：每題 4 分 (4*7=28 分)

1. 巨集處理器 (Macro Processor)
   用來將「高階語言或組合語言」中巨集展開的程式，展開後將不再有巨集，可以被編譯器或組譯器處理。
2. 堆疊機 (Stack Machine)
   堆疊機則是利用將參數推入堆疊之後，再從堆疊中取出適當數量的元數進行運算，因此運算指令通常沒有參數。 例如、PUSH x, PUSH y, ADD 這樣的模式，就是一個簡單的堆疊機程式片斷。
3. 暫存器機 (Register Machine)
   暫存器機將暫存器代號編入指令當中，以指定進行運算的暫存器。 舉例而言，像是 ADD R1, R2, R3 就是一種暫存器機的指令，運算過程通常透過暫存器才能完成。
4. 剖析器 (Parser)
   用來將原始程式轉換爲語法樹或者進行語法比對的程式，剖析後才能進行程式碼產生的動作。
5. 遞迴下降法 (Recursive Descent Parsing)
   遞迴下降法是一種剖析方法，其方法是將 BNF 或 EBNF 語法轉換成對應的遞迴程式，然後從上而下進行剖析的一種方法。
6. Context Switch (內文切換)
   當排班系統確定將行程切換成另一個時，會進行暫存器與環境保存，然後再準備環境 (載入暫存器與環境) 給下一個行程的動作，這個動作稱爲內文切換。內文切換由於經常被呼叫，速度過慢的話會嚴重影響系統效能，因此大多會用組合語言撰寫。
7. 虛擬機 (Virtual Machine)
   在真實機器上透過軟體模擬處理器指令集的方式，讓程式與作業系統可以在該環境下執行的一種軟體執行環境，稱爲虛擬機。
8. Thread (線程、執行緒)
   Thread 是一種執行函數的方法，這些函數可以在作業系統的安排下同時或交錯執行，而且相互之間可以共用變數，有點像是行程 (Process) 的概念，只是較爲輕量級，因此也稱爲輕量級行程。
9. 驅動程式 (drivers)
   將周邊裝置的存取包裝成一套函數呼叫的方法，這種函數稱爲驅動程式。
10. 基底-界限暫存器
    基底界限暫存器是一種記憶體管理單元 (MMU)，透過基底暫存器達到重定位功能，讓程式不需要在載入時進行重定位，而界限暫存器則可以用來限制記憶體存取的範圍，避免程式存取超過範圍的記憶空間，造成系統記憶體被破壞，以防止駭客程式的非法存取行爲。
11. 排程問題 (Scheduling)
    在行程管理系統中，決定下一個要被執行的「行程」是哪一個的問題，稱爲排程問題。常見的排程方法有「先到先做排程、循環分時排程、優先權排程」等等。
12. 競爭情況 (Race Condition)
    當兩個 Thread 共用某個變數，而且同時寫入該變數時，可能會造成執行結果不一致的情況，稱爲競爭情況
13. 死結 (Deadlock)
    爲了避免競爭情況，因此必須引入鎖定機制，但是鎖定機制可能會造成「死結」的情況，所謂的死結就是我鎖住你要的資源，而你也鎖住我要的資源，我不讓給你、你也不讓給我，因此就卡死在那裏，兩個 Thread 或 Process 都不能繼續執行了。
14. 中斷向量 (Interrupt Vector)
    當處理器要支援中斷機制時，通常會在保留一小塊固定位址的記憶體，這塊記憶體幾乎都是跳耀指令 (JMP)，當某個中斷發生時，硬體就會自動跳到特定的位址執行，以便處理此中斷，這一小塊包含數個跳躍址令的記憶體，就稱爲中斷向量。

請說明下列指令或關鍵字之用途：每題 3 分 (3*3=9)

1. #ifdef
   C 語言當中用來判斷某符號是否被定義的巨集判斷指令，例如 #ifdef DEBUG 可用用來判斷是否有定義 DEBUG 這個符號。
2. volatile (C 語言關鍵字)
   volatile 是揮發性的意思，在 C 語言中可以用來加在變數上，告訴編譯器該變數可能會被外部的輸出入修改，

避免編譯器認爲這些變數的值沒有改變而導致錯誤的最佳化問題。

3. **gcc -E macroDebug.c -o MacroDebug_E.c**
   gcc 中的 -E 參數要求 gcc 只展開巨集，但是不要轉換成組合語言或機器馬。
4. **gcc -S optimize.c -o optimize_O0.s -O0**
   gcc 中的 -S 參數要求 gcc 將程式轉換爲組合語言，但是不要轉換成機器碼。
5. **objdump -x hello.o**
   objdump 可將目的檔傾印以便閱讀，而 -x 參數則是要求印出表頭。
6. **ar -r libabc.a a.o b.o c.o**
   ar 是用來建立函式庫的指令，上述指令將 a.o b.o c.o 等三個檔案壓入到 libabc.a 函式庫中。

簡答題： (每題 5 分, 5*3=15)

1. 請舉例說明何謂記憶體映射輸出入？
將某些記憶體位址對映到輸出入裝置暫存器，以進行輸出入的一種方法，舉例而言，假如 0xFF00 代表七段顯示器的位址，那麼 STB R1, 0xFF00 則可以將 R1 的內容輸出到七段顯示器上， 讓七段顯示器顯示某些字樣。

2. 請舉例說明何謂專用輸出入指令？
若某 CPU 指令即擁有專用輸出入指令，像是 TD, IN, OUT 等，那麼就可以用這些指令對「輸出入埠」進行存取，達成輸出入的行爲。舉例而言，假如 0x37 代表七段顯示器，那麼 OUT R1, 0x37 則可以將 R1 的內容輸出到七段顯示器上， 讓七段顯示器顯示某些字樣。

3. 請說明單工 (Single Tasking) 與多工 (Multi Tasking) 的差異？
單工的電腦只能執行一個行程，而多工的電腦可以讓很多行程同時存在系統當中，讓作業系統挑選並進行排程。

4. 請問輪詢 (Polling) 是指甚麼技術？
輪詢是一種輸出入時的程式技巧，代表輪流的詢問每一個裝置，如果該裝置有輸入或可以輸出時，就進行輸出入動作，這種輪流詢問的技巧稱爲輪詢。

考生姓名：　　學號：　　得分：

5. 請問中斷 (Interrupt) 技術如何用在作業系統的行程切換上？
當作業系統要執行某個程式，或將控制權交給某行程之前，可以先設定一個「時間中斷」，強制在一小段時間之後引發該中斷，這樣作業系統就能強制取回 CPU 的控制權，避免某些行程當機或霸佔處理器不放。

6. 請問何謂 Round-Robin 排程方法？
Round-Robin 是大輪迴式的排程方式，方法是對每個行程設定一段時間切片，當時間耗盡時就收回控制權並進行行程切換，讓下一個行程可以執行的排程方法。

簡答題：語法部分 (每題 10 分)

1. 請根據下列語法，畫出 3*7-5/4 的任意兩顆不同剖析樹，然後說明何謂「有歧義的語法」！ (10%)

E = N | E [+-*/] E
N = [0-9]+

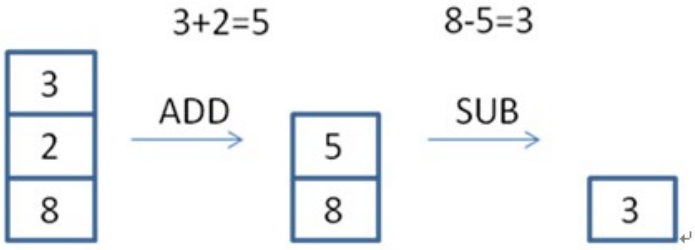
2. 請說明下列堆疊機指令的執行過程與結果 (10%)

PUSH 8
PUSH 2
PUSH 3
ADD
SUB

1. E = E*E = E * (E - E) = E * (E – (E/E)) = 3*(7-(5/4))
E = E-E = (E*E) – (E/E) = (3*7)-(5/4)

兩者語法樹不同，而且執行結果的語義也不同，這種根據同一個語法卻有兩種不同剖析樹的結果，稱爲有歧義的語法，是在設計語法時就應該要避免的。

2.



程式註解題：請爲下列程式加上說明 (函數請說明整個的用途, 對應硬體請參考後面的圖, 每題 3 分, 3*6=18)。

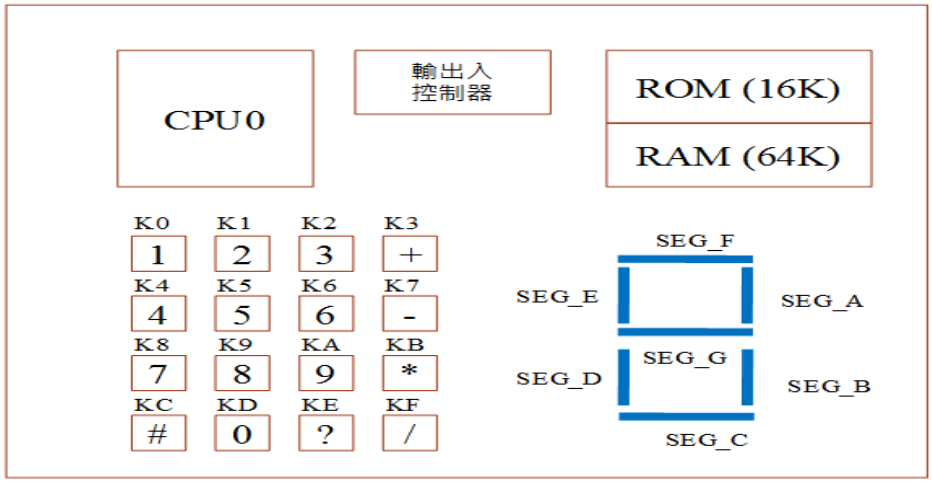
1. #define BYTE unsigned char // 定義 8 位元無號整數 BYTE 爲 unsigned char

```
2. #define UINT16 unsigned short // 定義 8 位元無號整數 UINT16 爲 unsigned short
3. #define BOOL unsigned char // 定義布林值 BOOL 爲 unsigned char
4. #define SEG7_REG (*(volatile BYTE*)0xFFFFFF00) // 定義七段顯示器的記憶體映射輸出入。
5. #define KEY_REG1 (*(volatile BYTE*) 0xFFFFFF01) // 定義鍵盤的記憶體映射輸出入, K8~KF (前八位元)。
6. #define KEY_REG2 (*(volatile BYTE*) 0xFFFFFF02) // 定義鍵盤的記憶體映射輸出入, K0~K7 (後八位元)。
7. #define KEY (KEY_REG2 << 8 | KEY_REG1) // 定義鍵盤的記憶體映射輸出入 K0~KF (共 16 個位元)
8. #define BYTE seg7map[]={ /*0*/ 0x3F, /*1*/ 0x18, /*2*/ 0x6D, /*3*/ 0x67, /*4*/ 0x53, /*5*/ 0x76, /*6*/ 0x7E, /*7*/ 0x23, /*8*/ 0x7F, /*9*/ 0x77 }; // 定義 0~9 所對應的七段顯示器顯示樣式。

9. #define char keymap[]={ '1', '2', '3', '+', '4', '5', '6', '-', '7', '8', '9', '*','#', '0', '?', '/'}; // 定義鍵盤代號與字元的對應方式。

10. void seg7_show(char c) { // 用來顯示字元 c 到七段顯示器上的驅動程式。
   SEG7_REG = seg7map[c-'0'];
}
11. char keyboard_getkey() { // 用來讀取哪個按鍵被按下的驅動程式。
  UINT16 key = KEY;
  for (int i=0; i<16; i++) {
  UINT16 mask = 0x0001 << i;
   if (key & mask !=0)
     return keymap[i];
 }
 return 0;
}
12. BOOL keyboard_ishit() { // 用來判斷是否有任何按鍵被按下的驅動程式。
 return (KEY != 0)
}
```

考生姓名：　　學號：　　得分：



表格 11.1 簡易電腦 M0 的硬體對映手冊 (Data Sheet)

Reg bit	7	6	5	4	3	2	1	0
IO_REG0 0xFFFFFF00		SEG_G	SEG_F	SEG_E	SEG_D	SEG_C	SEG_B	SEG_A
IO_REG1 0xFFFFFF01	KF	KE	KD	KC	KB	KA	K9	K8
IO_REG2 0xFFFFFF02	K7	K6	K5	K4	K3	K2	K1	K0

程式題：每題 10 分，10*2 = 20

1. 請寫一個 **CPU0** 的組合語言程式可以在上述架構中，在七段顯示器顯示一個 **5** 字。 **2.** 請寫一個 **CPU0** 的組合語言程式可以在上述架構中，在七段顯示器顯示一個 **2** 字。	**3.** 請寫一個 **CPU0** 的組合語言程式可以在上述架構中，判斷鍵盤數字 **7** 是否被按下。 **4.** 請寫一個 **CPU0** 的組合語言程式可以在上述架構中，判斷鍵盤數字 **3** 是否被按下。
1. **LD R8, IO_BASE** **LDI R1, 0x76** **STB R1, [R8+0x00]** **RET** **IO_BASE: WORD 0xFFFFFF00** 2. **LD R8, IO_BASE** **LDI R1, 0x6D** **STB R1, [R8+0x00]** **RET** **IO_BASE: WORD 0xFFFFFF00**	**3.** **LD R8, IO_BASE** **LD R3, [R8+1]** **LDI R7, 0x01** **AND R1, R3, R7 // R1 爲 0 的話，代表 7 沒被按下，若爲 1 則代表被按下。** **RET** **IO_BASE: WORD 0xFFFFFF00** **4.** **LD R8, IO_BASE** **LD R3, [R8+2]** **LDI R7, 0x04** **AND R1, R3, R7 // R1 爲 0 的話，代表 3 沒被按下，若爲 1 則代表被按下。** **RET** **IO_BASE: WORD 0xFFFFFF00**