金門大學 資訊工程系 101 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. 高階語言 (High Level Programming Language)
   像是 C, Java, JavaScript, Python, Ruby 等語言稱為高階語言，指不是組合語言或機器語言的程式語言。
3. 掃描器 (Lexer, Scanner)
   用來將原始程式切割成一個一個基本詞彙單元的程式，切割後的結果會送往剖析器去進行語法剖析。
4. 剖析器 (Parser)
   用來將原始程式轉換為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. Linux
   Linux 是一種類似 UNIX 的開放原始碼的作業系統，創建者是 Linus Torvalds ，現在廣泛用於伺服端與嵌入式系統中， Google 的 Android 系統也是建構在 Linux 之上的。
7. JVM (Java Virtual Machine)
   JVM 是 Java 虛擬機，此虛擬機可以用來解譯 Java 編譯後的 ByteCode，並採用 JIT (Just in time compiling) 的方式加快速度。
8. 行程管理系統
   執行中的程式稱為行程 (Process)，在作業系統當中進行行程管理操作的模組，稱為行程管理系統 (Process Management System)，其中的「排程」與「行程切換」是關鍵的基本單元。
9. 記憶體管理系統
   作業系統中用來管理記憶體的模組，稱為記憶體管理系統，包含「動態記憶體管理」(像是 C 語言的 malloc 函數要求分配記憶體時所要做的動作) 與「虛擬記憶體」管理。
10. MMU (Memory Management Unit)
    記憶體管理系統經常需要搭配硬體的 MMU (Memory Management Unit) 才能有效的發揮效成，並確保系統的安全性。所謂的 MMU 做法，最簡單的有「基底-界限」暫存器，複雜一點的有「分頁、分段、分段式分頁」等做法。
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，稱為排程問題。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，稱為競爭情況。
13. 死結 (Deadlock)
    為了避免競爭情況，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，就稱為中斷向量。

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. #define
   C 語言中用 #define 來定義符號或巨集，與 #ifdef 搭配可以用來進行條件式的巨集展開。

3. **gcc -E macroDebug.c -o MacroDebug_E.c**
   用 gcc 來將有巨集的 macroDebug.c 程式，展開成無巨集的 MacroDebug_E.c 程式。
4. **gcc -E -D_DEBUG_ macroDebug.c -o MacroDebug_DEBUG_E.c**
   用 gcc 來將有巨集的 macroDebug.c 程式，展開成無巨集的 MacroDebug_DEBUG_E .c 程式。，但是有用 -D_DEBUG_ 定義 _DEBUG_ 這個符號，因此 程式中若有 #ifdef _DEBUG_ 時將會展開其內容。
5. **gcc -S optimize.c -o optimize_O0.s -O0**
   用 gcc 將 optimize.c 程式轉換為組合語言 optimize_O0.s (因為 -S 參數代表輸出組合語言)，而且 -O0 代表不進行最佳化。
6. **gcc -S optimize.c -o optimize_O3.s -O3**
   用 gcc 將 optimize.c 程式轉換為組合語言 optimize_O0.s (因為 -S 參數代表輸出組合語言)，而且 -O3 代表要進行最高級的最佳化。

考生姓名： 學號： 得分：

簡答題：語法部分 (每題 10 分)

1. 請根據下列語法，畫出 **3*5+8** 的兩顆不同剖析樹，然後說明何謂「有歧義的語法」！ **(10%)**

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

E = E*E = E * (E + E) = 3*(5+8) = 39
E = E+E = (E * E) + E = (3*5)+8 = 23

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，稱為有歧義的語法，是在設計語法時就應該要避免的。

2. 請根據上列語法，畫出 **2-4*9** 的兩顆不同剖析樹，然後說明何謂「有歧義的語法」！**(10%)**

E = E*E = E * (E + E) = 3*(5+8) = 39
E = E+E = (E * E) + E = (3*5)+8 = 23

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，稱為有歧義的語法，是在設計語法時就應該要避免的。

簡答題：關於編譯器的六大階段，請回答下列問題 (每題 5 分, 5*3=15)
3. 請問掃描階段的輸入與輸出為何，功能是甚麼呢？

掃描階段：輸入為原始程式碼、輸出為一個一個的詞彙 (token) (也有可能附加上詞彙的形態、行號位置等資訊)。
功能為將原始程式碼切分成一個一個的詞彙。

4. 請問剖析階段的輸入與輸出為何，功能是甚麼呢？

剖析階段：輸入為原始程式碼 (或者一個一個的詞彙 token) ，輸出為語法樹 (也有可能不建語法樹而直接在剖析完一個節點後就進行後續處理)。
功能為根據語法辨認原始程式碼，並建立樹狀節點的程式。

5. 請問語義分析階段的輸入與輸出為何，功能是甚麼呢？

語義分析階段：輸入為語法樹或樹狀節點 ，輸出為具有變數型態語意標示的語法樹或節點。
功能為確認變數型態的相容性，並且標示變數型態。

6. 請問中間碼產生階段的輸入與輸出為何，功能是甚麼呢？

中間碼產生階段：輸入為語法樹或樹狀節點 (有型態標示) ，輸出為該程式所對應的中間碼。
功能為將語法樹轉換為中間碼型態，以便後續處理。

7. 請問最佳化階段的輸入與輸出為何，功能是甚麼呢？

在編譯器中最佳化可能不只被作一次，在產生中間碼時可能就會有一次最佳化，其輸入為語法樹，輸出為中間碼，而在產生組合語言或目的碼時又可能會再做一次最佳化，其輸入為中間碼，輸出為組合語言或目的碼。

8. 請問組合語言 (目的碼) 產生階段的輸入與輸出為何，功能是甚麼呢？

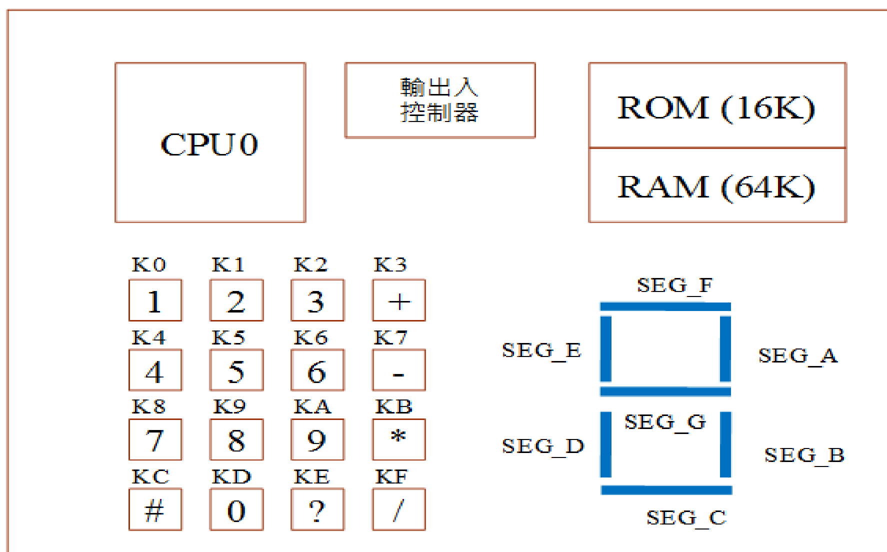
組合語言 (目的碼) 產生階段的輸入為中間碼，輸出為組合語言或目的碼，功能示將中間碼轉換成需要考慮機器架構與暫存器配置的組合語言或目的碼。

程式註解題：請為下列程式加上說明 (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 為 unsigned char
2. #define UINT16 unsigned short // 定義 8 位元無號整數 UINT16 為 unsigned short
3. #define BOOL unsigned char // 定義布林值 BOOL 為 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 keyboa
 return (KEY != 0
}
```



表格 11.1 簡易電腦 MO 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 **0** 字。

```
LD R8, IO_BASE
LDB R1, [0x3F]
STB R1, [R8+0x00]
RET
IO_BASE WORD 0xFFFFFF00
```

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 **3** 字。

```
LD R8, IO_BASE
LDB R1, [0x67]
STB R1, [R8+0x00]
RET
IO_BASE WORD 0xFFFFFF00
```

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 **2** 是否被按下。

```
LD R8,  IO_BASE
LD R3, [R8+2]
LDI R7, 0x02
AND R4, R3, R7  // R4 為 0 的話，代表 2 沒被按下，若為 1 則代表被按下。
RET
IO_BASE WORD 0xFFFFFF00
```

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 **6** 是否被按下。

```
LD R8,  IO_BASE
LD R3, [R8+2]
LDI R7, 0x40
AND R4, R3, R7  // R4 為 0 的話，代表 6 沒被按下，若為 1 則代表被按下。
RET
IO_BASE WORD 0xFFFFFF00
```