實驗五 程式計數器與堆疊

1. 學習重點

- 了解程式計數器 PC 的性質，並藉由模擬來觀察程式執行過程中 PC 的變化。
- 了解 8051 的堆疊性質、堆疊指標 SP 以及 PUSH、POP 等堆疊相關指令。
- 了解呼叫指令，並與跳躍指令比較兩者的差異。

2. 材料清單

表 5-1、材料清單

器材名稱		數量
AT89S51		1
12MHz 石英震盪器		1
LED 二極體		8
按壓開關		2
電阻	1kΩ	8
	10kΩ	1
電容	20pF	2
	10μF	1

3. 元件原理

程式計數器 Program Counter

程式計數器是一個 16 位元的暫存器，用來指向下一條要被執行的指令在程式記憶體中的位址。8051 的每個指令都有各自在程式記憶體中所需的空間，且看似相同的指令還會根據目標不同而有著不同的長度，例如最常見的 MOV 指令，若是將指定的值移入指定的暫存器，其所需的程式記憶體空間為 2 Byte，而若是將累加器 A 的值移入指定暫存器，其所需的記憶體空間為 1 Byte。當指令執行完後，會依據剛才執行的指令在程式記憶體的空間決定 PC 跳躍的距離。PC 下一個指向的位址除了指令執行完畢後自動累加外，也可以透過 JMP 指令來指定接下來 PC 指向的位址。由於程式記憶體中有部分的位址是有特殊的用途，

例如中斷向量等，因此我們通常會利用假指令 ORG 來使指令寫入程式記憶體時避開那些位址，此部分會於未來的中斷實驗解釋。

堆疊 Stack 與堆疊指標 Stack Pointer

堆疊 Stack 是一種資料結構，其特徵為先進後出（First In Last Out, FILO），與之相對的資料結構是佇列 Queue，佇列則是先進先出（First In First Out, FIFO）。在 8051 中的程式中時常需要使用到堆疊的方式來儲存資料，但在資料記憶體中並沒有特定規劃一個部份是專屬於堆疊使用的，堆疊的資料儲存於何處是由堆疊指標 Stack Pointer（SP）決定，每當有資料需要儲存進堆疊時，便會將資料儲存進 SP 指向的位址。SP 的預設值為 07H，意即指向資料記憶體中 07H 的位址，而將資料儲存進堆疊的指令都是先使 SP 加一後再將資料儲存於該位址，也就是說在不更改 SP 預設指向的位址時，首筆被存入堆疊的資料將儲存於資料記憶體中 08H 的位址，該位址恰好也是暫存器庫 1 的 R0 位址，為了避免可能的錯誤，一般我們會使用 MOV 指令使 SP 指向 30H 後的位址，因資料記憶體 30H 到 7FH 的位址並無特定用途，僅作為一般資料的存放區。

與堆疊直接相關的指令有兩個，分別為 PUSH 與 POP，正如其字面意思，PUSH 指令是將資料「壓」入堆疊中，而 POP 則是使資料從堆疊中「彈」出；然而並不是只有這兩個指令與堆疊相關，也有其他指令會將資料存入堆疊或是存堆疊中取出，並改變 SP 的值，例如接下來即將介紹的呼叫 CALL 指令，因此在使用堆疊相關指令時需注意 SP 的變化，以免發生錯誤。

PUSH 指令使用方法

- 格式：PUSH direct
- 作用：將堆疊指標加一後，將資料記憶體中 direct 位址所存的資料複製進堆疊指標所指向的資料記憶體位址中。
- 範例：*MOV SP, 30H*
  PUSH 01H

資料記憶體 01H 的位址為暫存器庫 0 的 R1，因此這段指令將使 SP 指向資料記憶體 30H 的位址，再將 SP 加一後指向 31H 的位址，最後將暫存器庫 0 的 R1 資料複製進 31H。

POP 指令使用方法

- 格式：POP direct

- 用途：將 SP 目前所指向位址儲存的資料複製進 direct 位址，之後將 SP 減一。
- 範例：*MOV SP, 30H*
  PUSH 01H
  POP 02H

承 PUSH 指令的範例，POP 指令便會將方才存入堆疊的資料複製進暫存器 R2，也就是利用堆疊將資料從暫存器 R1 複製進 R2。

在使用 PUSH 與 POP 指令時必須注意這兩個指令都是將值複製進堆疊或是複製出堆疊，換言之，我們依舊可以藉由調整 SP 的值來取得已經被 POP 過的堆疊資料。

呼叫 CALL 與跳躍 JMP 的比較

CALL 和 JMP 都是組合語言中常用來切換程式的指令，在之前的實驗中我們都是使用 JMP 來跳躍至 DELAY 等副程式，當副程式執行完畢後便需要再次使用 JMP 回到主程式，然而 CALL 卻不需要刻意指定返回的位址就能回到主程式，以下便來進行兩者的比較：

跳躍 JMP

除了無條件式的跳躍，例如 LJMP 或是 SJMP，還有其他需要滿足條件方能進行跳躍的條件式跳躍，例如 DJNZ、JNB 等，因此跳躍的特色之一為能進行分支跳躍，就像是 C 語言的 if else 判斷式。

不論是有條件還是無條件的跳躍，指令最後都需要指定要跳往的位址，其原理便是將程式計數器的值改成目標位址，來達成程式不依照指令位於程式記憶體的順序來執行；也因此使用跳躍指令後程式計數器便「有去無回」，如果有回到跳躍前位址的需求便需要我們再次使用跳躍指令回到跳躍前的位址。

呼叫 CALL

僅有兩個指令：LCALL 和 ACALL，其差別在於能前往的程式記憶體距離遠近，並不像是跳躍有著多種指令，呼叫僅有無條件式的呼叫，因此無法達成分支呼叫。

呼叫與跳躍類似，都是藉由更改程式計數器來前往目標位址，然而和跳躍不同的是，在執行呼叫指令時，會分別將 PC 加上呼叫指令長度的 0-7 位元和 8-15 位元 PUSH 進堆疊，而當呼叫的程式執行完畢後分別 POP 以取得 PC 應當返

回的位址，讓程式可以接續剛剛呼叫前的指令繼續執行。

正如前面所述，呼叫的指令是有使用到堆疊的，因此若是副程式中也有使用到堆疊就必須注意是否有更改到存入的程式計數器之值，或是使用 RET 時堆疊指標指向了錯誤的位址導致無法返回正確的位址。

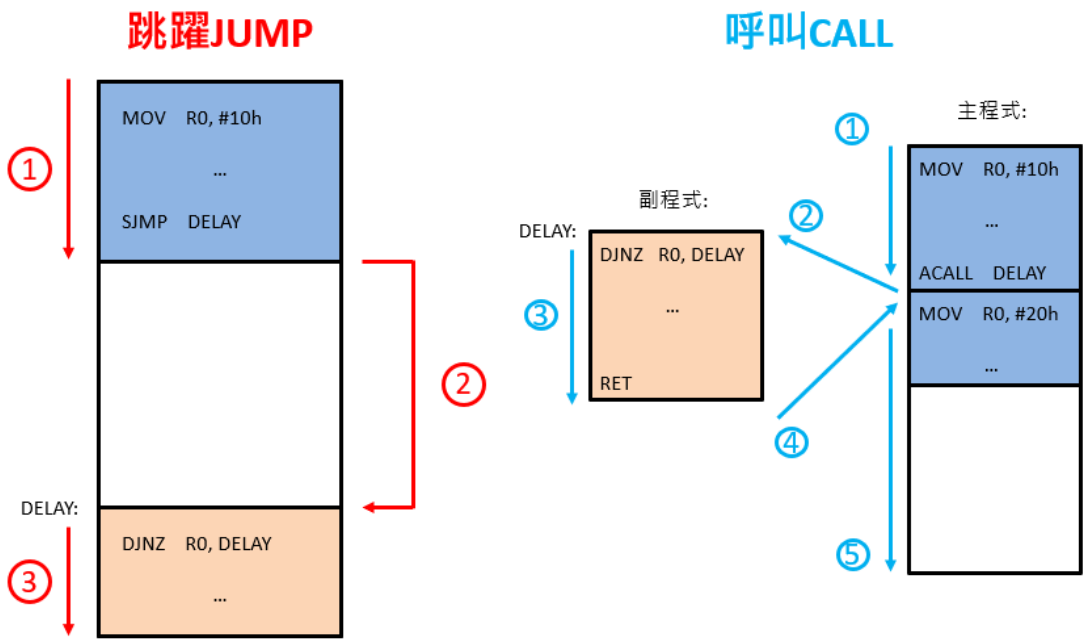

圖 5-1、CALL 與 JMP 的比較

4. 實驗內容

利用掃描的方式檢查連接於 8051 P1.0 的按鈕是否被按下，當按下按鈕時呼叫跑馬燈的副程式，並藉此實驗觀察程式計數器 PC 與堆疊指標 SP 的變化，了解 8051 中堆疊的性質以及使用方法。

5. 實驗電路圖

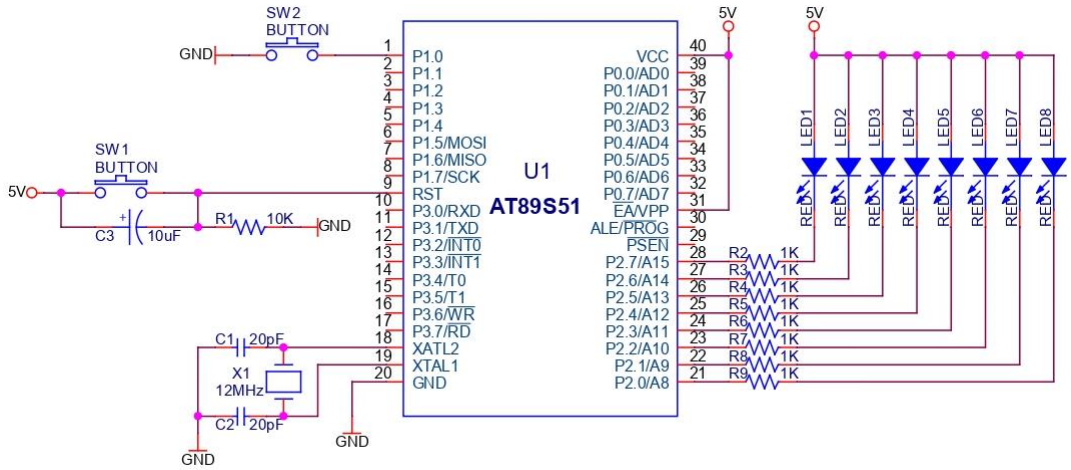

圖 5-2、實驗五基礎題參考電路圖

6. 軟體流程圖

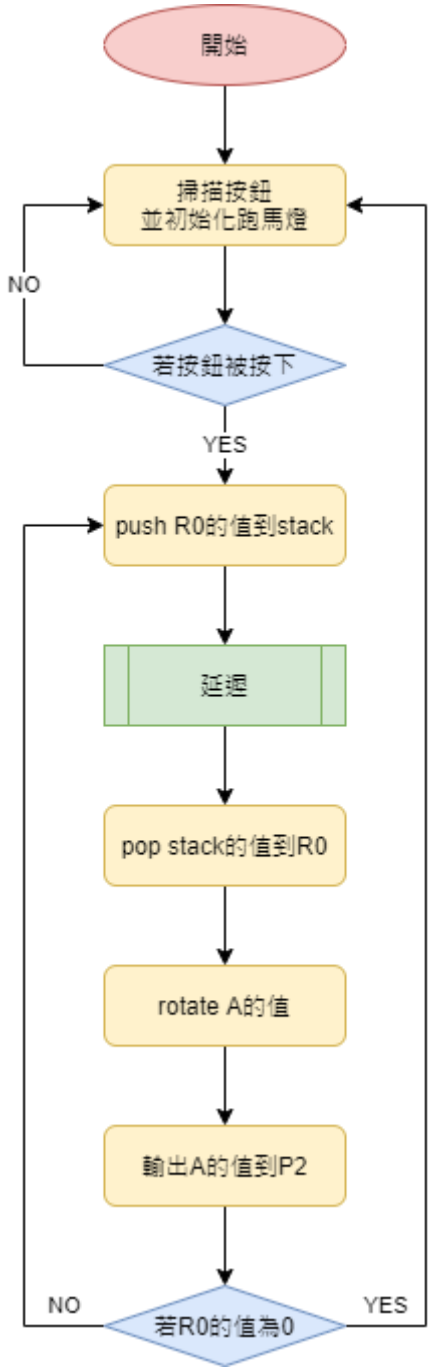

圖 5-3、實驗五基礎題參考軟體流程圖

7. 範例程式碼

```
1                   ORG 0000h
2                   JMP LOOP            ;jump into loop
3                   ORG 0030h
4   LOOP:           MOV SP, #32H        ;SP = #32H
5                   MOV A, #0xfe        ;A = #0xfe
6                   MOV P2, A           ;P2 = A
7                   SETB  P1.0          ;set p1.0 to high
8                   MOV R0, #8d         ;set the execution times of marquee
9                   JNB P1.0, MARQUEE   ;jump into marquee when p1.0 is low
10                  JMP LOOP            ;infinite loop
11  MARQUEE:        PUSH 00h            ;push the value of R0 into stack
12                  MOV R0, #250d       ;set the execution times of DELAY1
13                  CALL DELAY1         ;call DELAY1
14                  POP 00h
15                  ;pop out the value of R0 which is pushed in line 9
16                  RL A                ;left rotate A(left shift)
17                  MOV P2, A           ;set the value of A into P2
18                  DJNZ R0, MARQUEE
19                  ;loop back until MARQUEE execute 8 times
20                  JMP LOOP            ;end of MARQUEE, back to LOOP
21  DELAY1:         PUSH 00h
22                  ;push the remain times of DELAY1 into stack
23                  MOV R0, #250d       ;set the execution times of DELAY2
24                  CALL DELAY2         ;call DELAY2
25                  POP 00h
26                  ;pop the remain times of DELAY1 back to R0
27                  DJNZ R0, DELAY1     ;loop until R0 is 0
28                  RET                 ;return to MARQUEE
29  DELAY2:         DJNZ R0, DELAY2     ;loop untie R0 is 0
30                  RET                 ;return to DELAY1
31                  END
```

實驗中資料記憶體的值