微介實驗五

程式計數器與堆疊

日期：2024/10/29

報告者：許宸華

Outline

- 學習重點
- 實驗內容
- 材料清單
- 元件原理
- 實驗電路圖
- 軟體流程圖
- 實驗程式

Outline

- 學習重點
- 實驗內容
- 材料清單
- 元件原理
- 實驗電路圖
- 軟體流程圖
- 實驗程式

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。

Outline

- 學習重點
- 實驗內容
- 材料清單
- 元件原理
- 實驗電路圖
- 軟體流程圖
- 實驗程式

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。
- 按下按鈕時呼叫跑馬燈副程式。
- 觀察程式計數器PC與堆疊指標SP的變化。

Outline

- 學習重點
- 實驗內容
- 材料清單
- 元件原理
- 實驗電路圖
- 軟體流程圖
- 實驗程式

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	10
電容	20pF	2
	10μF	1

Outline

- 學習重點
- 實驗內容
- 材料清單
- 元件原理
- 實驗電路圖
- 軟體流程圖
- 實驗程式

元件原理

- 程式計數器 Program Counter (PC)
  - 指出程式記憶體中下一條待執行的指令位址
  - 8051的PC為16位元的暫存器
    - ➢最多指到2^{16} = 65536處
    - ➢程式記憶體最大為64K byte
  - 執行指令前會根據指令所佔的記憶體空間決定PC的累加值
  - 範例:

ADD A, #immediate

C	AC	F0	RS1	RS0	OV		P

Bytes	2	
Cycles	1	
Encoding	00100100	immediate
Operation	A = A + immediate	
Example	ADD A, #03h	

元件原理

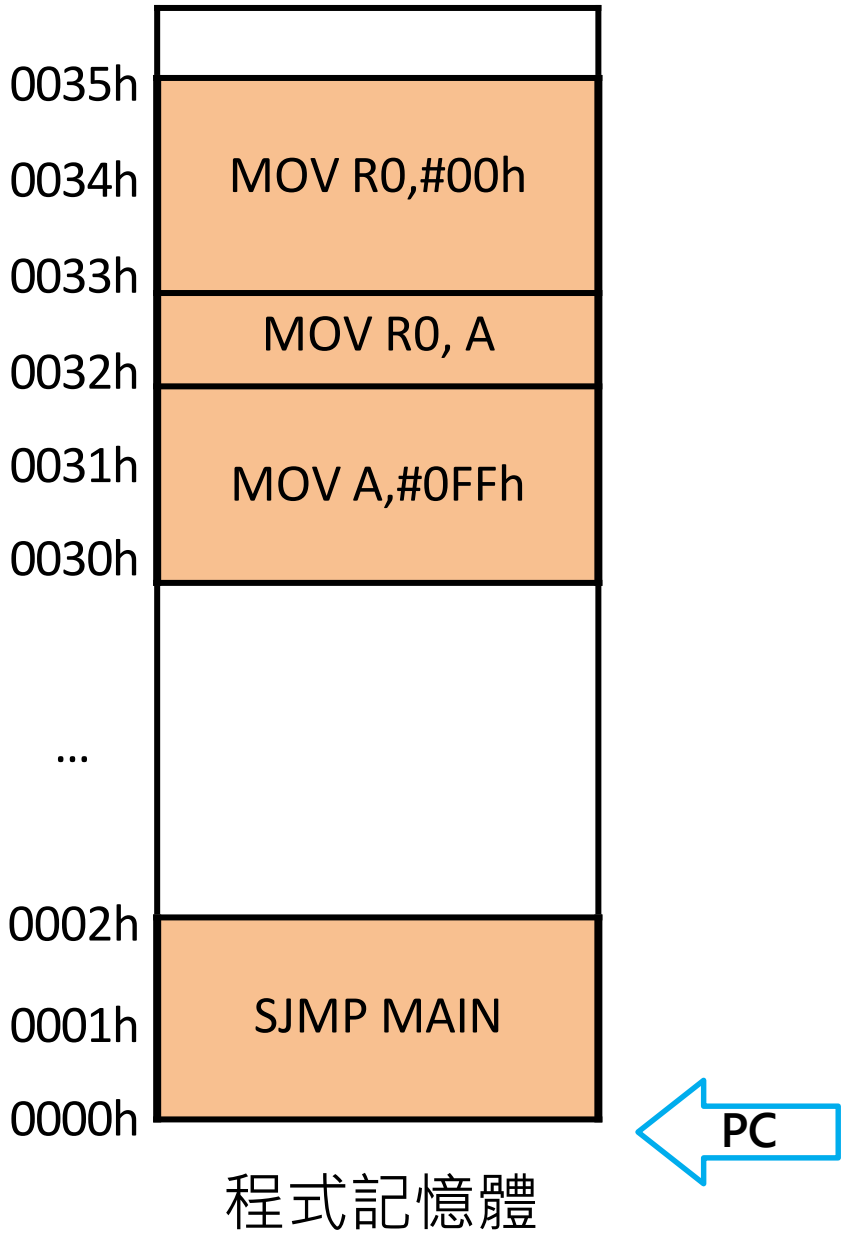
- 程式計數器 Program Counter (PC)
  - ORG
    - 虛擬指令(Pseudo Instruction)，僅用以指示組譯器
    - 用以指定接下來的指令在程式記憶體中的位址
    - 部分位址為中斷向量，因此欲使用中斷前需使用ORG跳過

元件原理

- 程式計數器 Program Counter (PC)

```
1.                ORG     0000h
2.                SJMP    MAIN
3.                ORG     0030h
4.    MAIN:       MOV     A, #0FFh
5.                MOV     R0, A
6.                MOV     R0, #00h
7.                END
```

A = 00h
R0 = 00h
PC = 00h

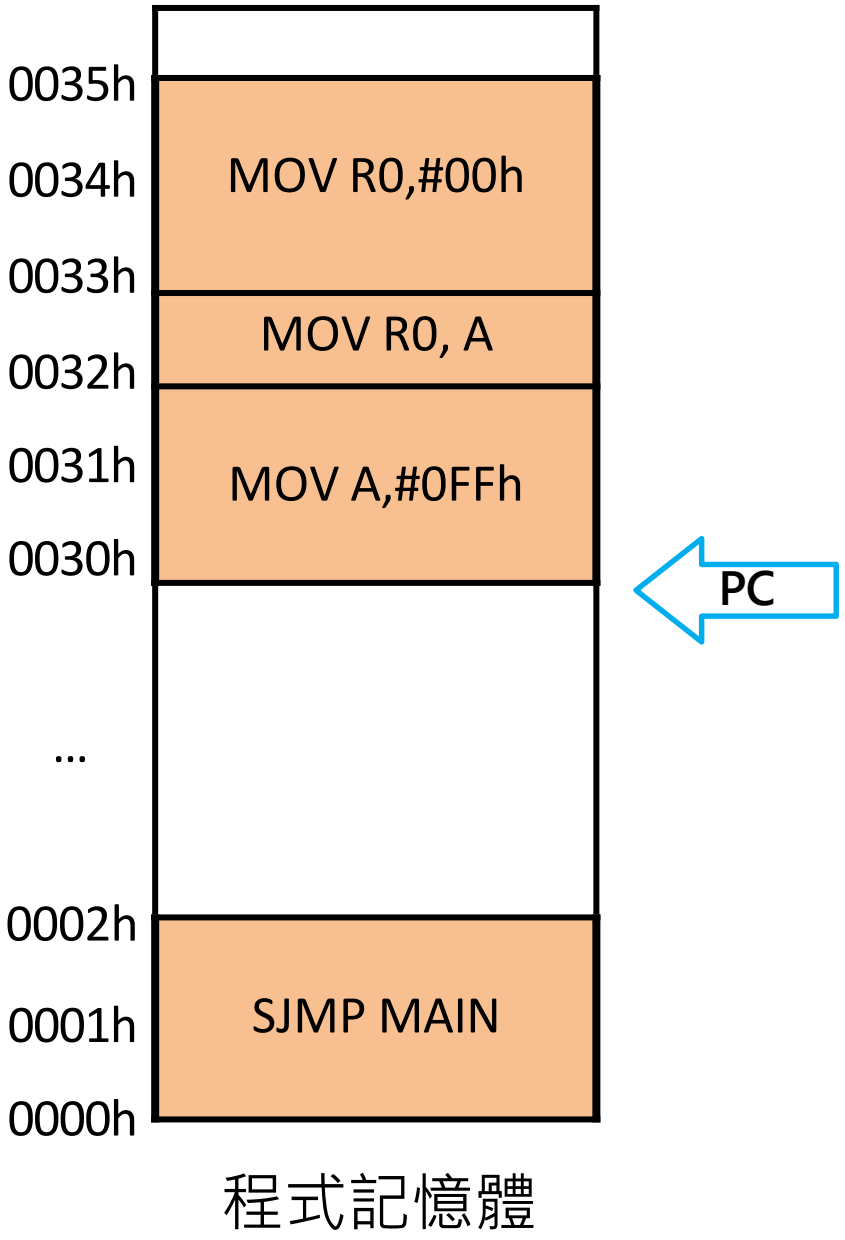


元件原理

- 程式計數器 Program Counter (PC)

```
1.                ORG     0000h
2.                SJMP    MAIN
3.                ORG     0030h
4.   MAIN:        MOV     A, #0FFh
5.                MOV     R0, A
6.                MOV     R0, #00h
7.                END
```

A = 00h
R0 = 00h
PC = 30h

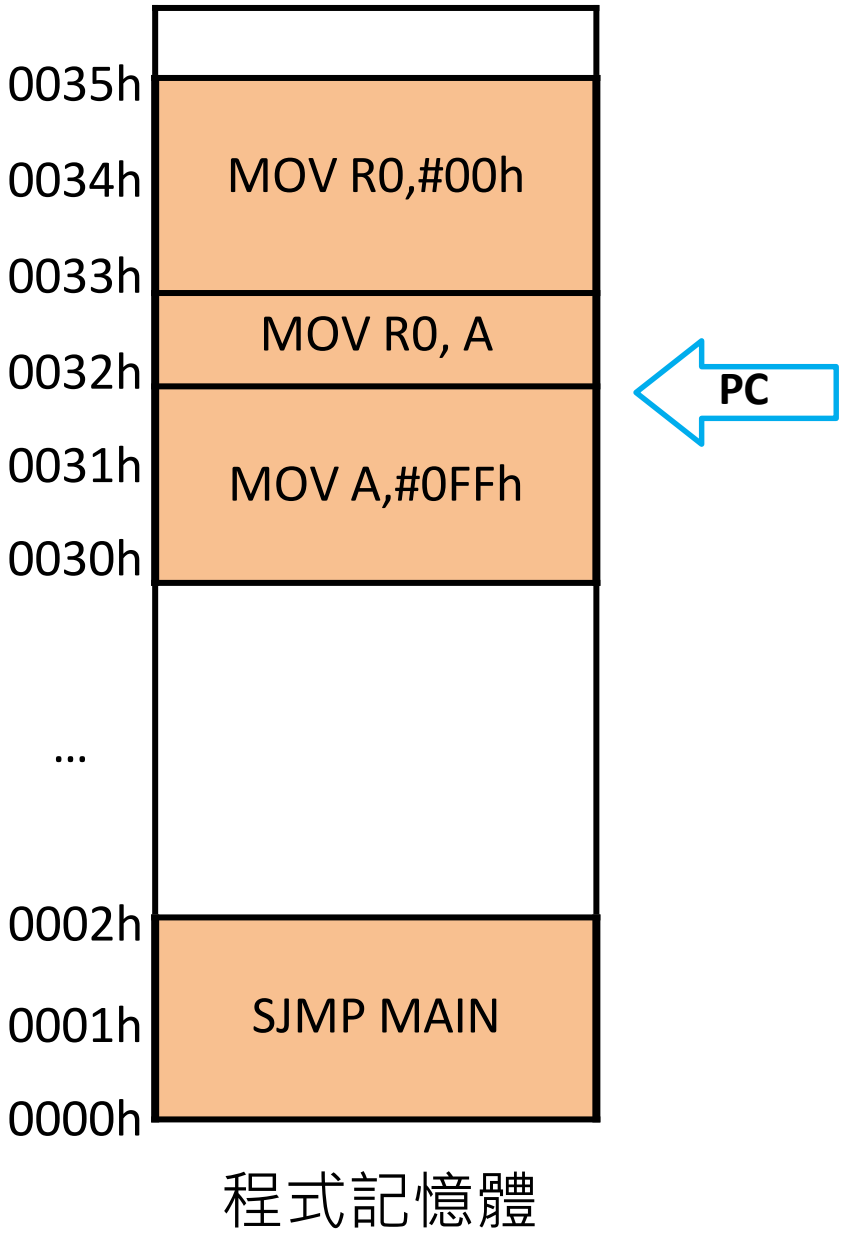


元件原理

- 程式計數器 Program Counter (PC)

```
1.                ORG     0000h
2.                SJMP    MAIN
3.                ORG     0030h
4.    MAIN:       MOV     A, #0FFh
5.                MOV     R0, A
6.                MOV     R0, #00h
7.                END
```

A = FFh
R0 = 00h
PC = 32h

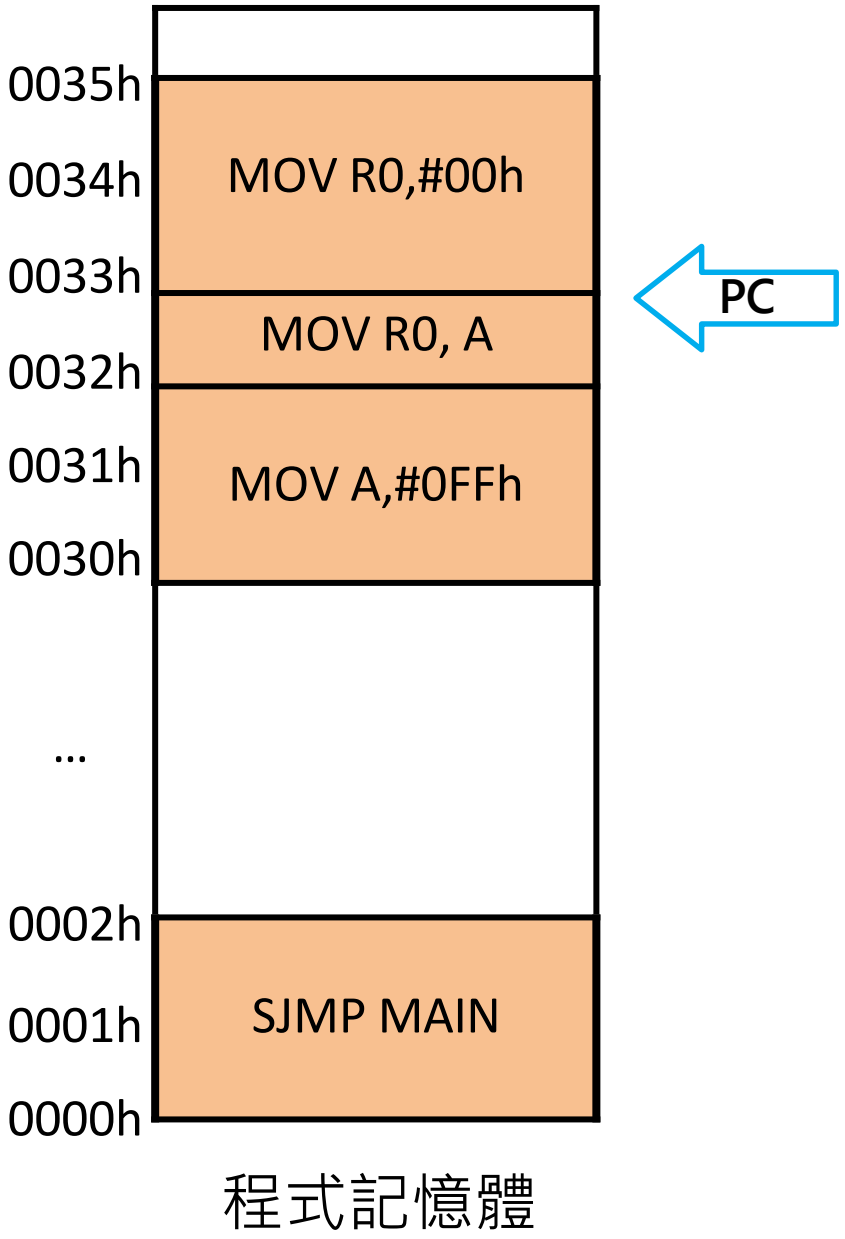


元件原理

- 程式計數器 Program Counter (PC)

```
1.                ORG     0000h
2.                SJMP    MAIN
3.                ORG     0030h
4.    MAIN:       MOV     A, #0FFh
5.                MOV     R0, A
6.                MOV     R0, #00h
7.                END
```

A = FFh
R0 = FFh
PC = 33h

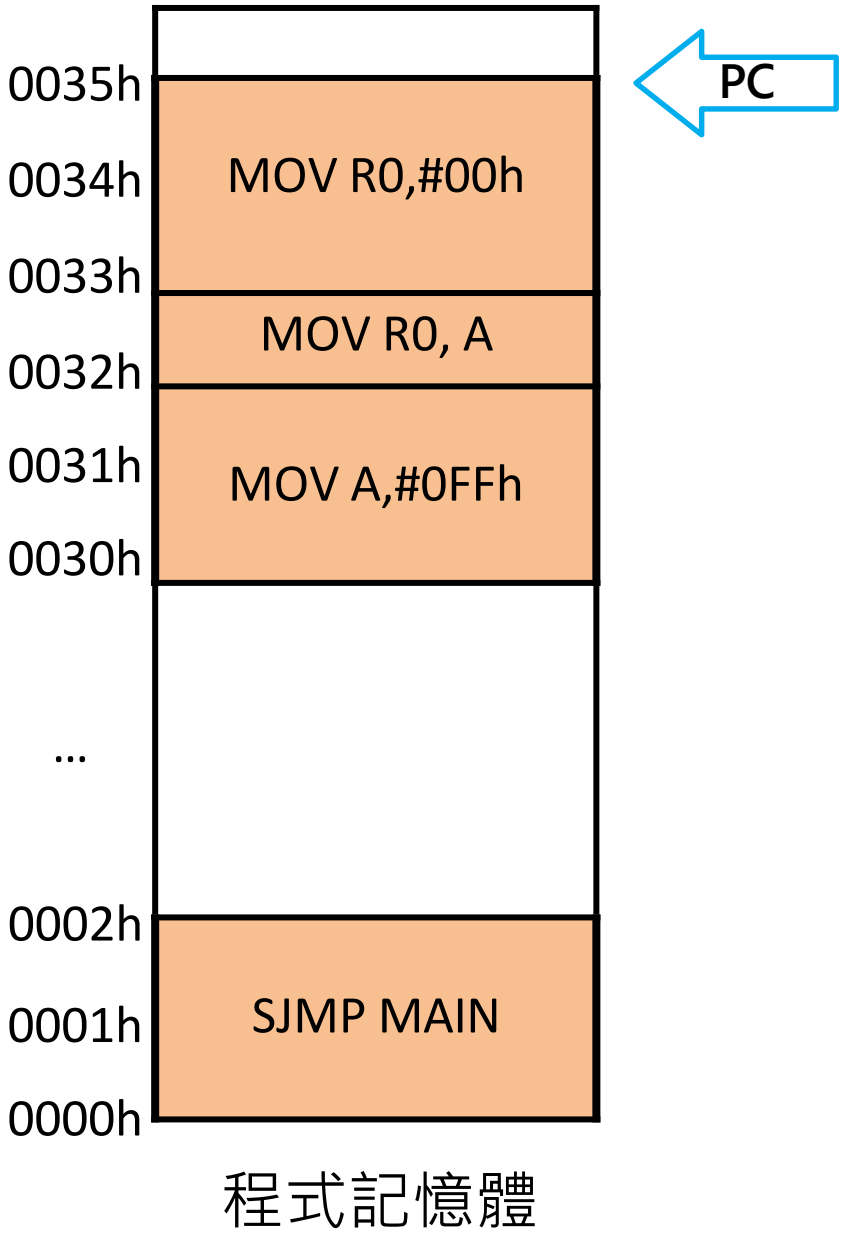


元件原理

- 程式計數器 Program Counter (PC)

```
1.                ORG     0000h
2.                SJMP    MAIN
3.                ORG     0030h
4.   MAIN:        MOV     A, #0FFh
5.                MOV     R0, A
6.                MOV     R0, #00h
7.                END
```

A = FFh
R0 = 00h
PC = 35h



元件原理

- 堆疊 Stack
  - 一種先進後出Last In First Out (LIFO)的資料結構
  - **PUSH**指令新增堆疊的資料
  - **POP**指令移出堆疊資料
  - **CALL**指令也是利用堆疊來儲存資料
  - 堆疊資料的位址由堆疊指標(SP)決定

元件原理

- 堆疊指標 Stack Pointer (SP)
  - 位於資料記憶體81H的暫存器
  - SP會指向最後一筆被加入到堆疊中的資料的記憶體位址
  - SP的預設位址為07H
    - 為了避免資料加入堆疊後可能和RB1的資料發生衝突，所以通常會將SP指到一般資料存放區(30H)之後的位址

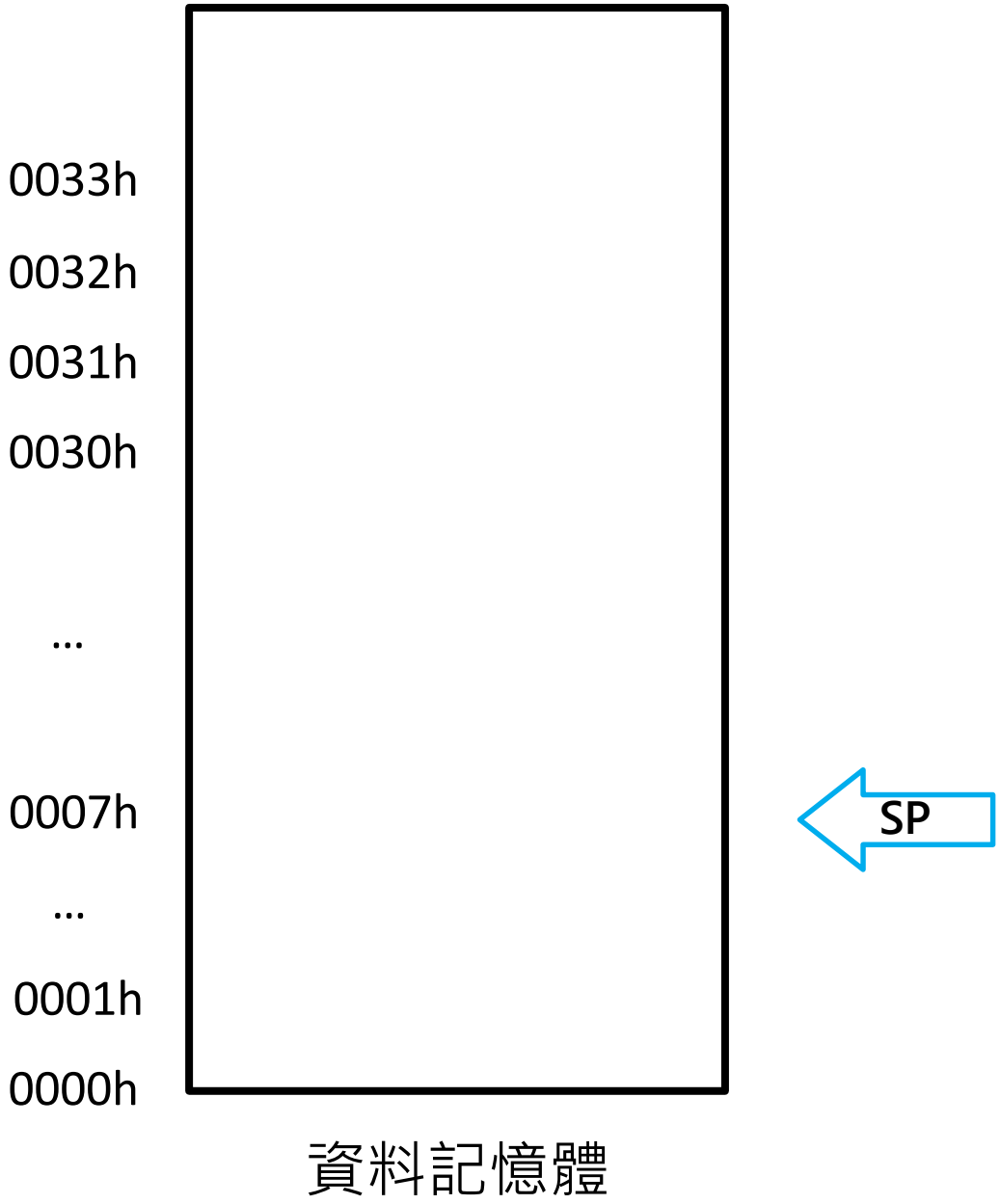
元件原理

- PUSH 與 POP
  - PUSH ：將資料加入堆疊
    - 格式：PUSH direct
    - 先將SP加1，再將direct位址的值複製到SP指向的記憶體位址
  - POP：將資料移出堆疊
    - 格式：POP direct
    - 先將SP指向的記憶體的資料複製進direct位址，再將SP減1

元件原理

- PUSH 與 POP

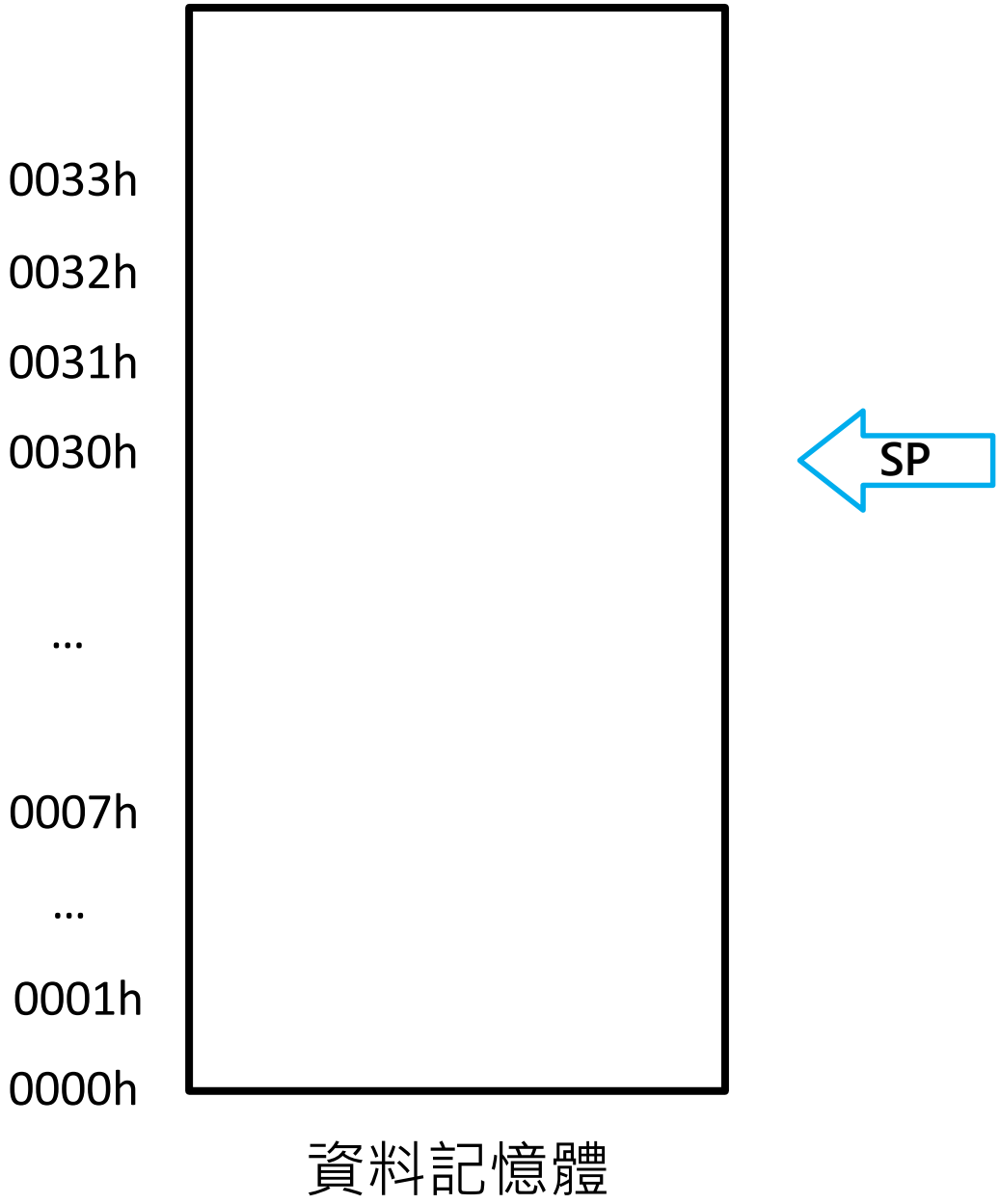
```
1.  MOV   SP,#30H
2.  MOV   R0,#01H
3.  PUSH  00H
4.  MOV   R0,#02H
5.  PUSH  00H
6.  MOV   R0,#03H
7.  PUSH  00H
8.  MOV   R0,#04H
9.  POP   00H
10. POP   01H
11. POP   00H
12. MOV   SP,#32H
13. POP   00H
```



元件原理

- PUSH 與 POP

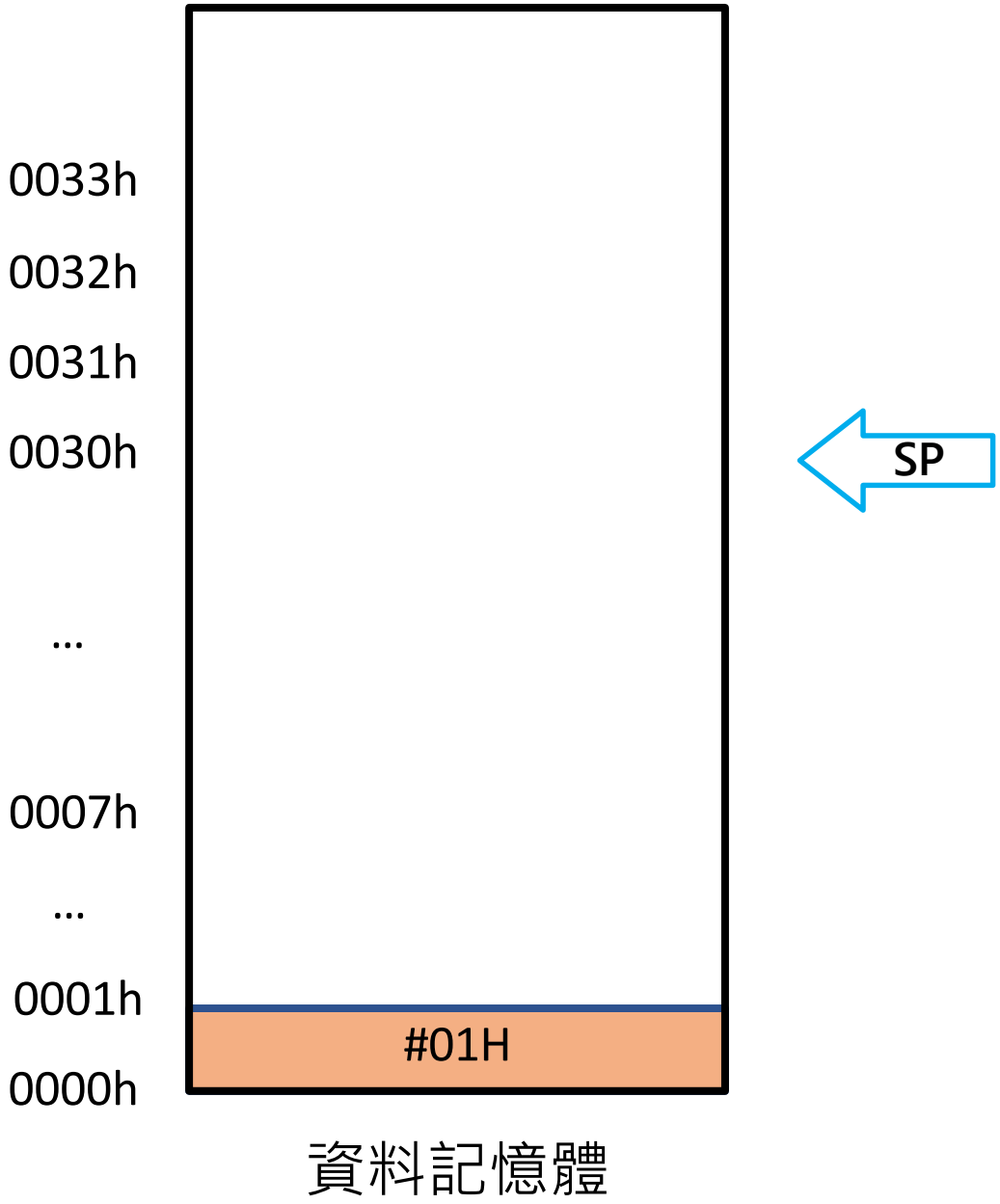
```
1.   MOV   SP,#30H
2.   MOV   R0,#01H
3.   PUSH  00H
4.   MOV   R0,#02H
5.   PUSH  00H
6.   MOV   R0,#03H
7.   PUSH  00H
8.   MOV   R0,#04H
9.   POP   00H
10.  POP   01H
11.  POP   00H
12.  MOV   SP,#32H
13.  POP   00H
```



元件原理

- PUSH 與 POP

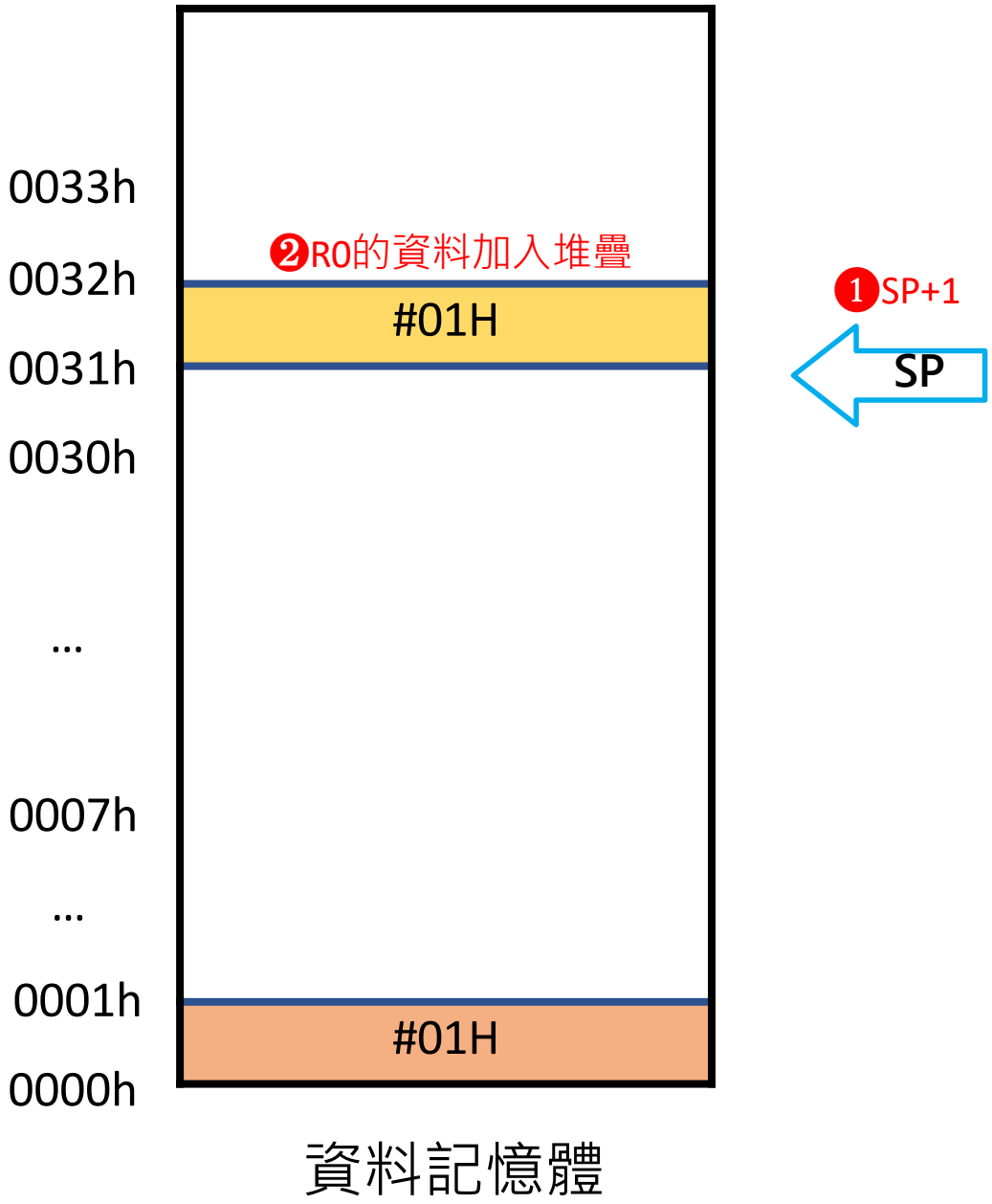
```
1.   MOV   SP,#30H
2.   MOV   R0,#01H
3.   PUSH  00H
4.   MOV   R0,#02H
5.   PUSH  00H
6.   MOV   R0,#03H
7.   PUSH  00H
8.   MOV   R0,#04H
9.   POP   00H
10.  POP   01H
11.  POP   00H
12.  MOV   SP,#32H
13.  POP   00H
```



元件原理

- PUSH 與 POP

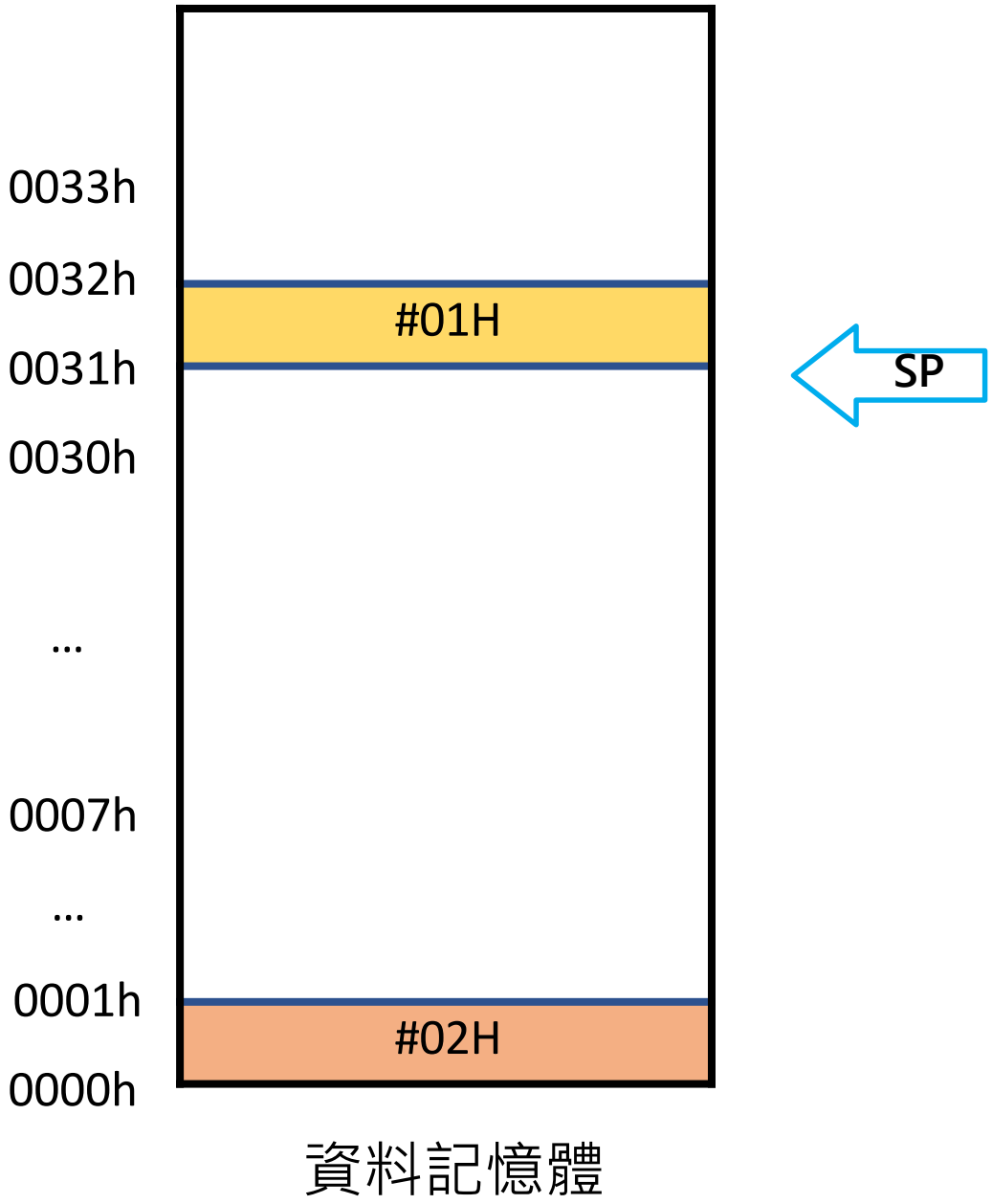
```
1.   MOV   SP,#30H
2.   MOV   R0,#01H
3.   PUSH  00H
4.   MOV   R0,#02H
5.   PUSH  00H
6.   MOV   R0,#03H
7.   PUSH  00H
8.   MOV   R0,#04H
9.   POP   00H
10.  POP   01H
11.  POP   00H
12.  MOV   SP,#32H
13.  POP   00H
```



元件原理

- PUSH 與 POP

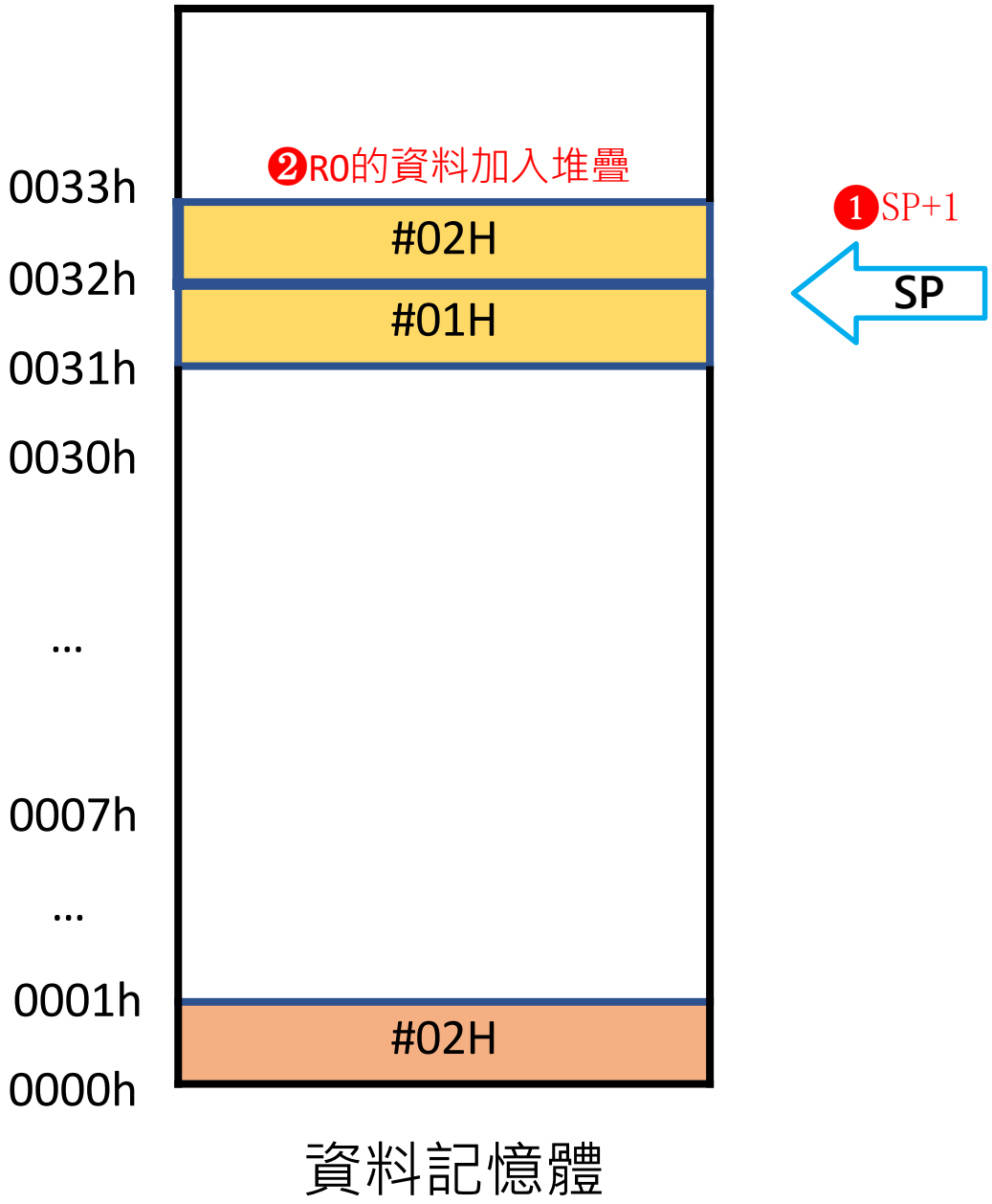
```
1.   MOV   SP,#30H
2.   MOV   R0,#01H
3.   PUSH  00H
4.   MOV   R0,#02H
5.   PUSH  00H
6.   MOV   R0,#03H
7.   PUSH  00H
8.   MOV   R0,#04H
9.   POP   00H
10.  POP   01H
11.  POP   00H
12.  MOV   SP,#32H
13.  POP   00H
```



元件原理

- PUSH 與 POP

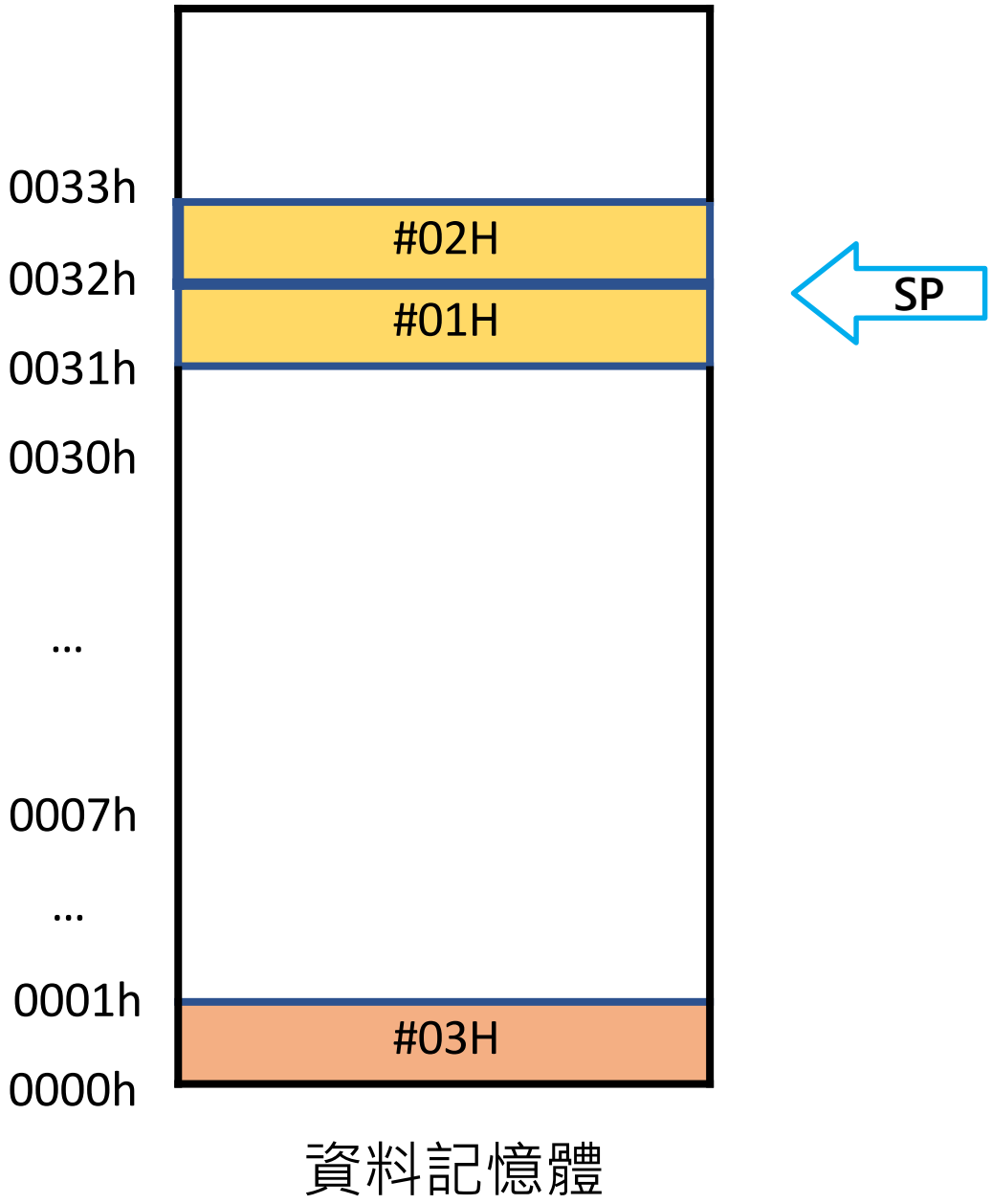
```
1.   MOV   SP,#30H
2.   MOV   R0,#01H
3.   PUSH  00H
4.   MOV   R0,#02H
5.   PUSH  00H
6.   MOV   R0,#03H
7.   PUSH  00H
8.   MOV   R0,#04H
9.   POP   00H
10.  POP   01H
11.  POP   00H
12.  MOV   SP,#32H
13.  POP   00H
```



元件原理

- PUSH 與 POP

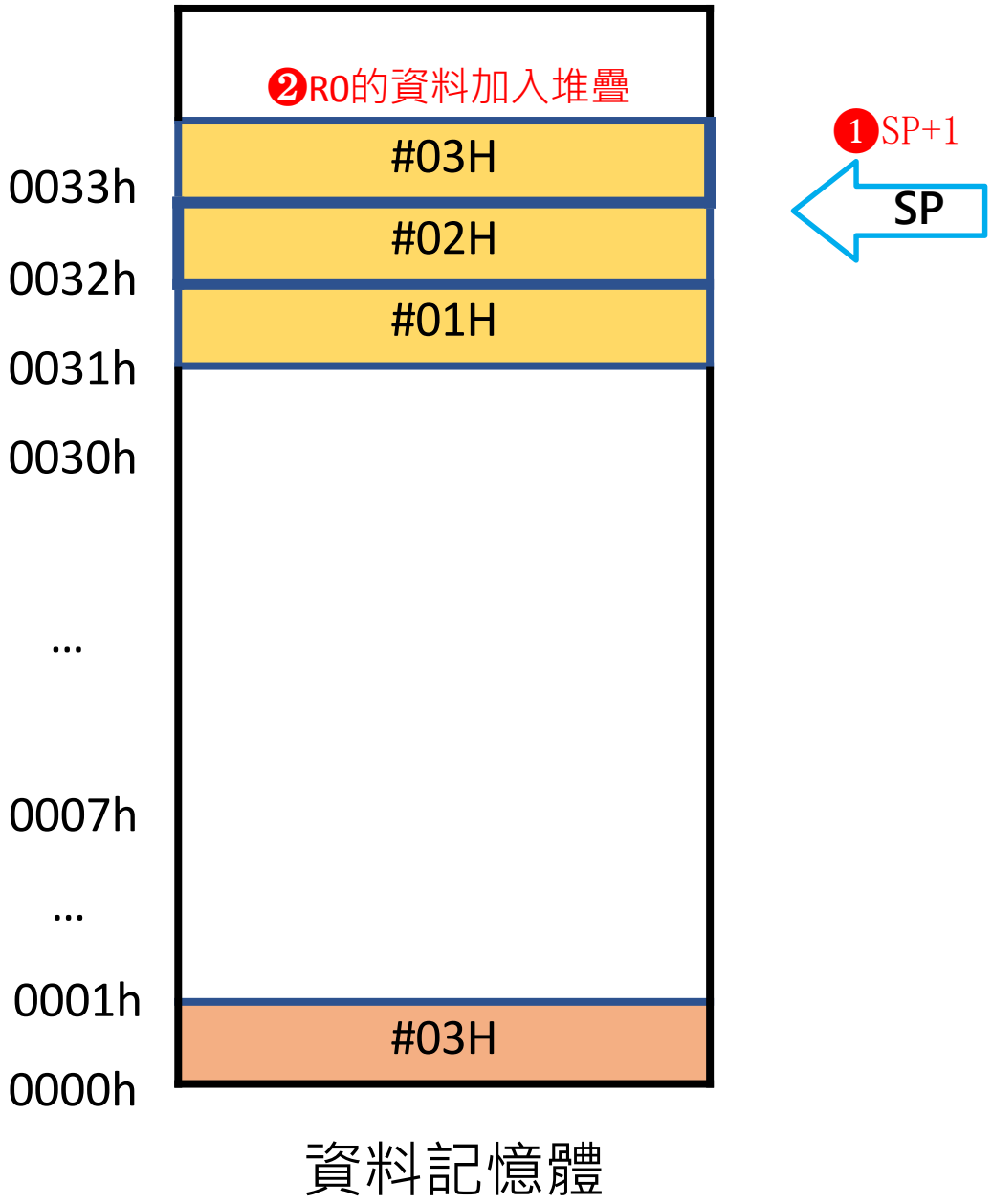
```
1.   MOV   SP,#30H
2.   MOV   R0,#01H
3.   PUSH  00H
4.   MOV   R0,#02H
5.   PUSH  00H
6.   MOV   R0,#03H
7.   PUSH  00H
8.   MOV   R0,#04H
9.   POP   00H
10.  POP   01H
11.  POP   00H
12.  MOV   SP,#32H
13.  POP   00H
```



元件原理

- PUSH 與 POP

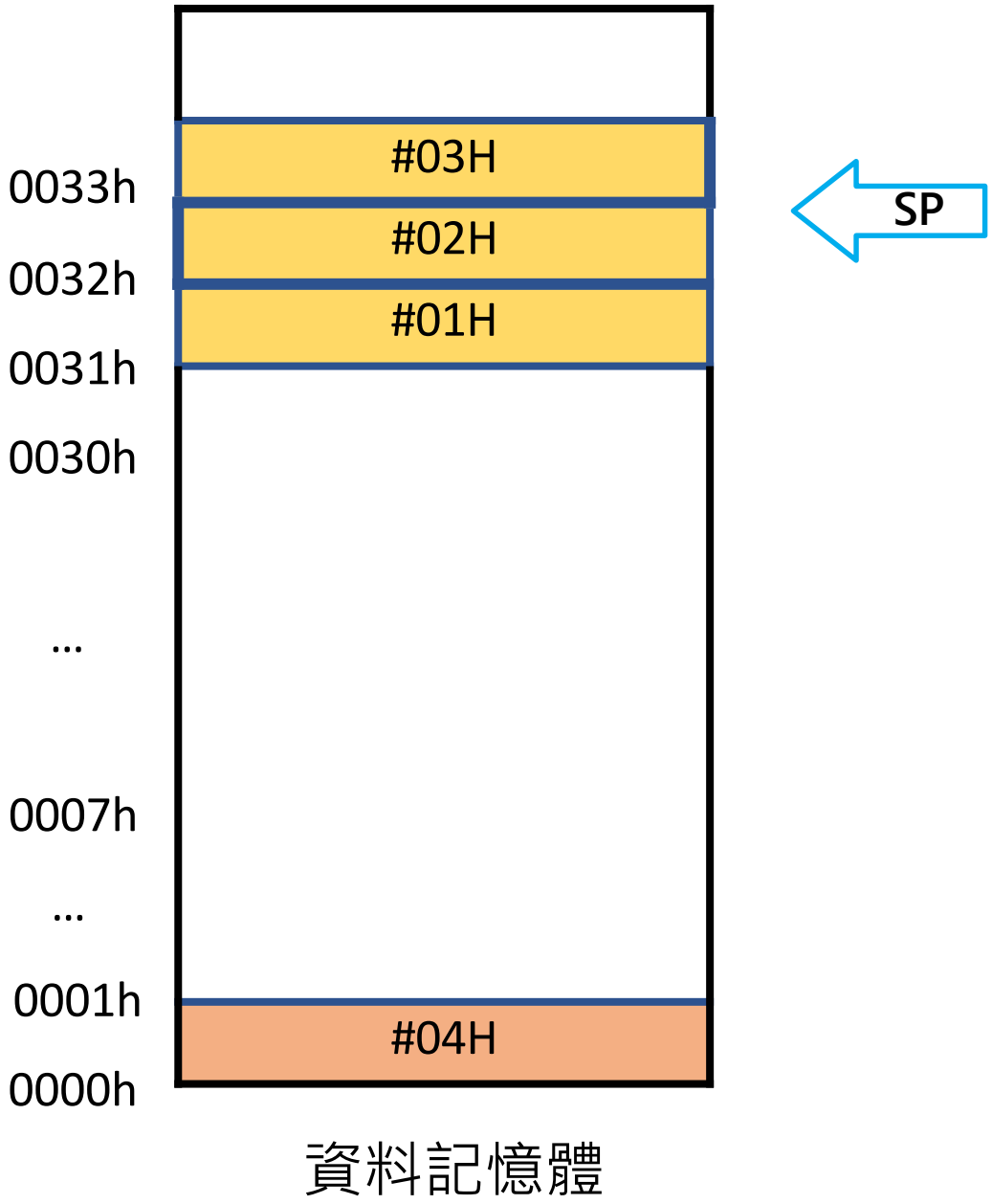
```
1.   MOV   SP,#30H
2.   MOV   R0,#01H
3.   PUSH  00H
4.   MOV   R0,#02H
5.   PUSH  00H
6.   MOV   R0,#03H
7.   PUSH  00H
8.   MOV   R0,#04H
9.   POP   00H
10.  POP   01H
11.  POP   00H
12.  MOV   SP,#32H
13.  POP   00H
```



元件原理

- PUSH 與 POP

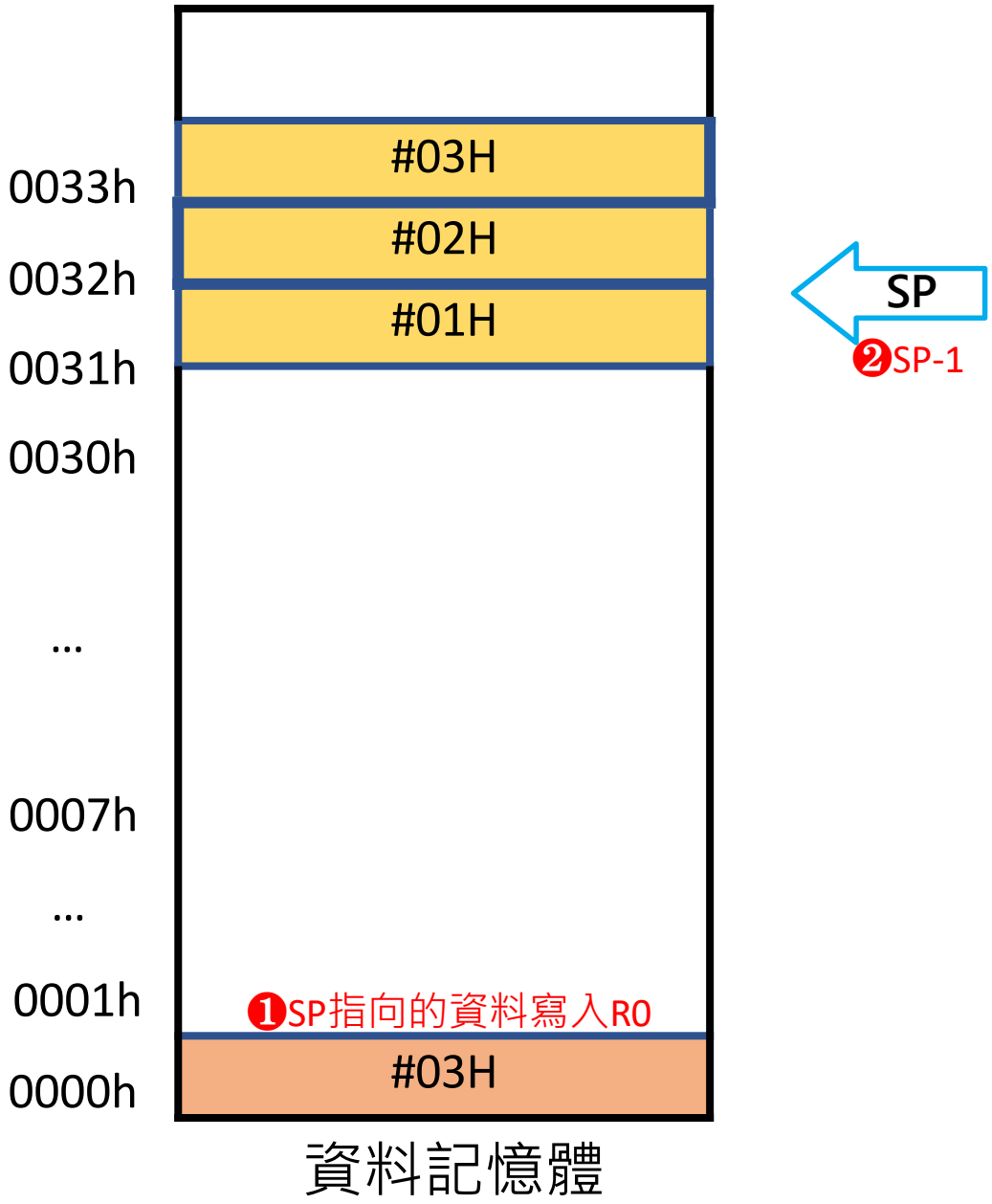
```
1.   MOV   SP,#30H
2.   MOV   R0,#01H
3.   PUSH  00H
4.   MOV   R0,#02H
5.   PUSH  00H
6.   MOV   R0,#03H
7.   PUSH  00H
8.   MOV   R0,#04H
9.   POP   00H
10.  POP   01H
11.  POP   00H
12.  MOV   SP,#32H
13.  POP   00H
```



元件原理

- PUSH 與 POP

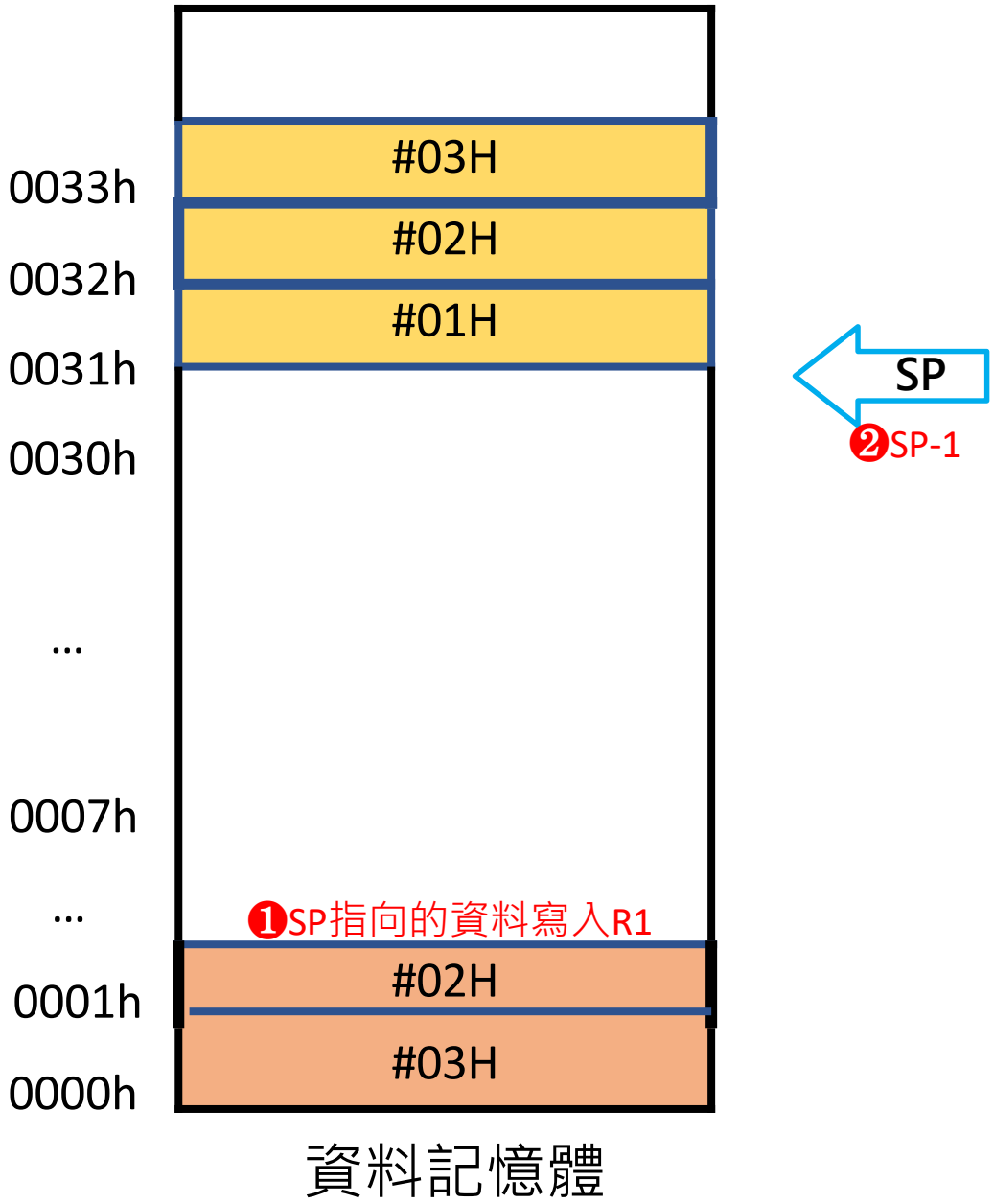
```
1.  MOV   SP,#30H
2.  MOV   R0,#01H
3.  PUSH  00H
4.  MOV   R0,#02H
5.  PUSH  00H
6.  MOV   R0,#03H
7.  PUSH  00H
8.  MOV   R0,#04H
9.  POP   00H
10. POP   01H
11. POP   00H
12. MOV   SP,#32H
13. POP   00H
```



元件原理

- PUSH 與 POP

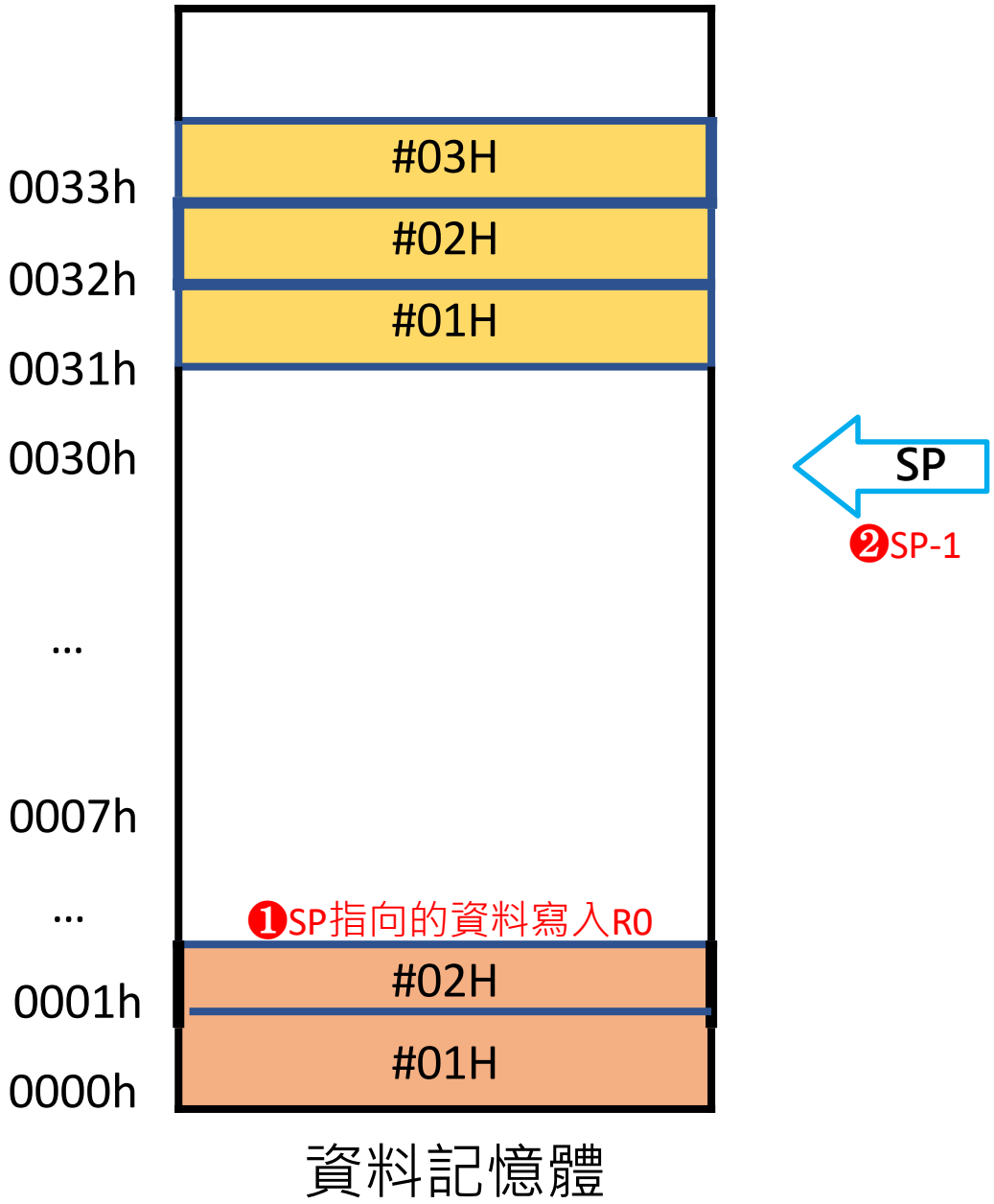
```
1.  MOV   SP,#30H
2.  MOV   R0,#01H
3.  PUSH  00H
4.  MOV   R0,#02H
5.  PUSH  00H
6.  MOV   R0,#03H
7.  PUSH  00H
8.  MOV   R0,#04H
9.  POP   00H
10. POP   01H
11. POP   00H
12. MOV   SP,#32H
13. POP   00H
```



元件原理

- PUSH 與 POP

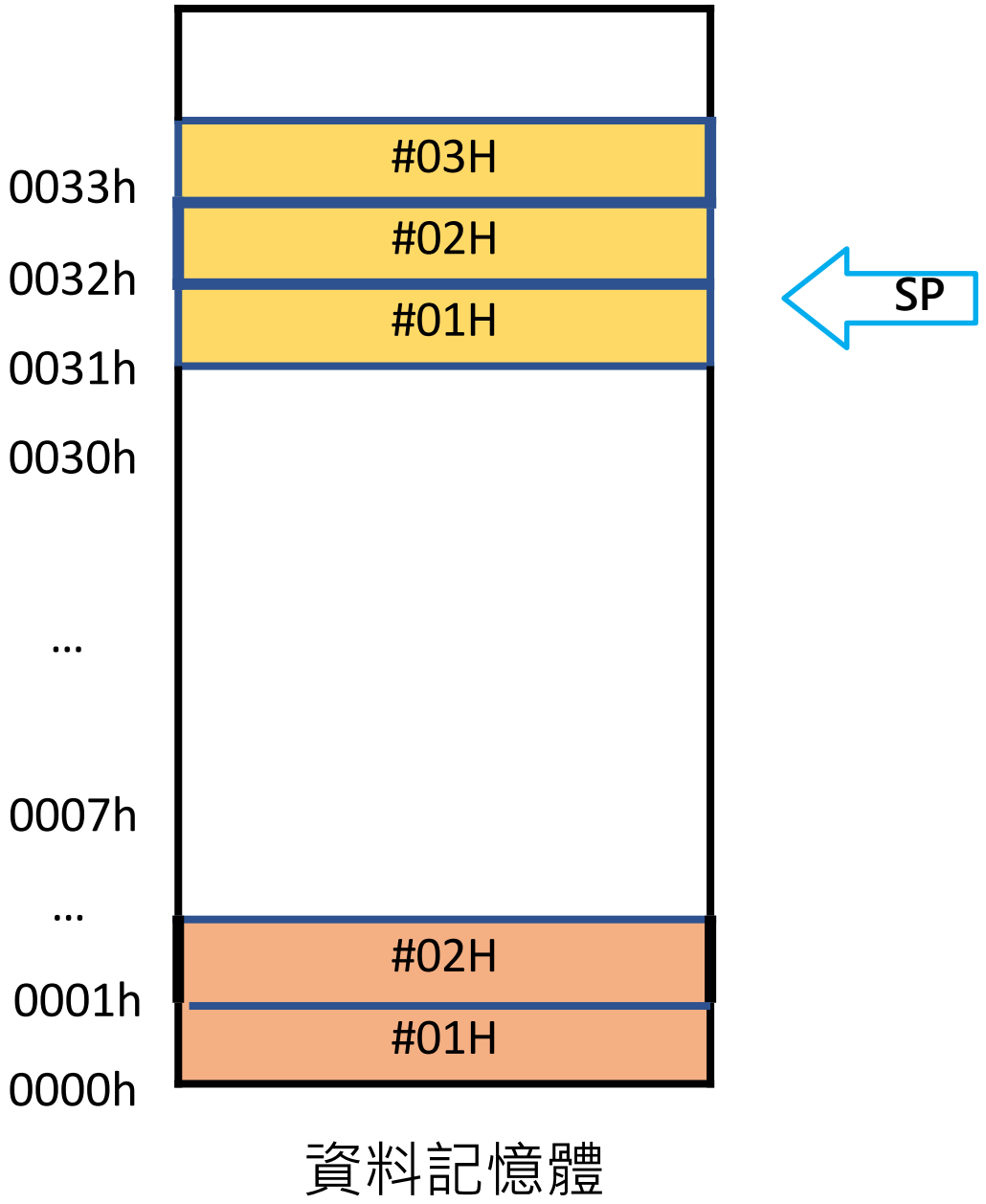
```
1.  MOV   SP,#30H
2.  MOV   R0,#01H
3.  PUSH  00H
4.  MOV   R0,#02H
5.  PUSH  00H
6.  MOV   R0,#03H
7.  PUSH  00H
8.  MOV   R0,#04H
9.  POP   00H
10. POP   01H
11. POP   00H
12. MOV   SP,#32H
13. POP   00H
```



元件原理

- PUSH 與 POP

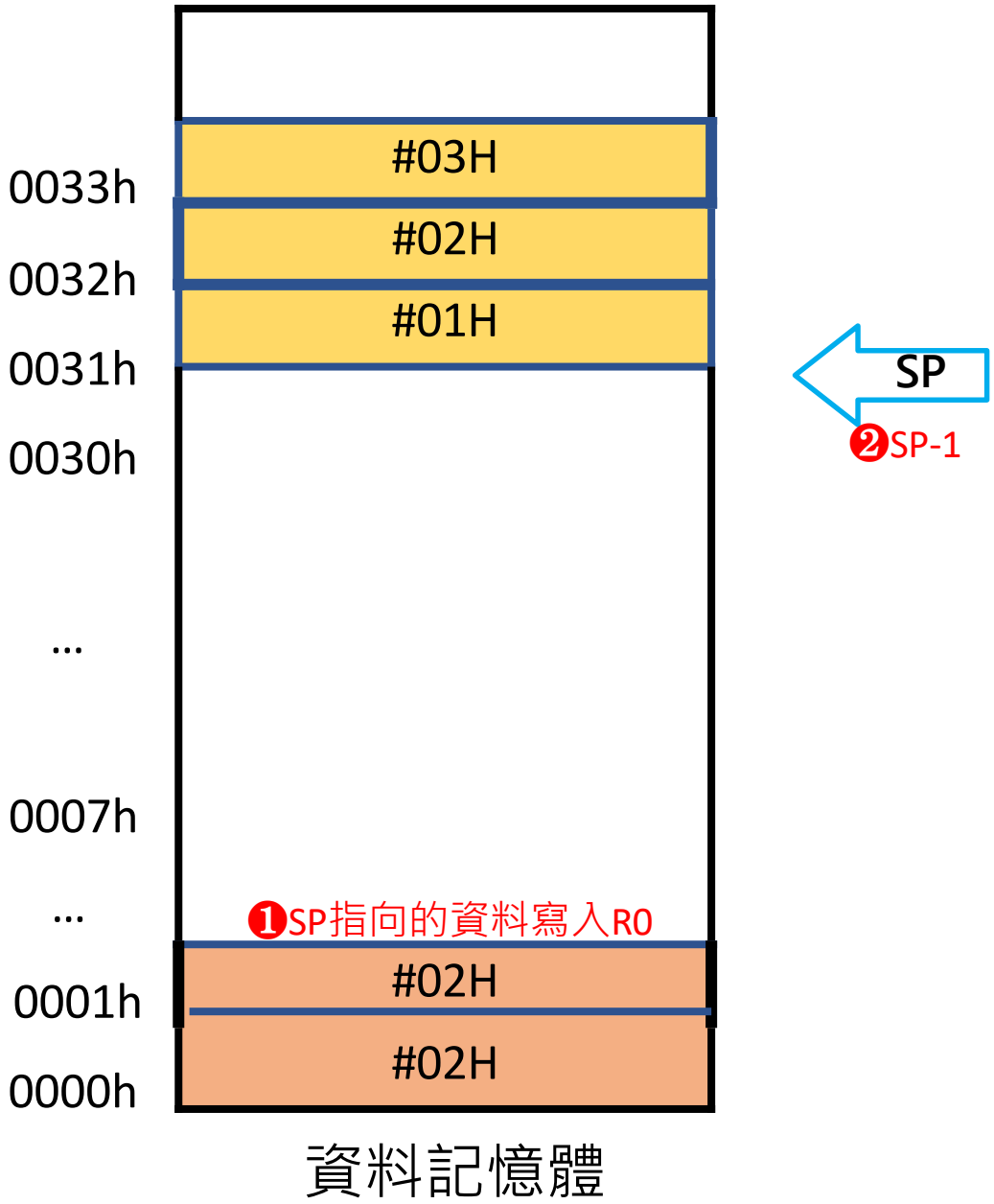
```
1.  MOV   SP,#30H
2.  MOV   R0,#01H
3.  PUSH  00H
4.  MOV   R0,#02H
5.  PUSH  00H
6.  MOV   R0,#03H
7.  PUSH  00H
8.  MOV   R0,#04H
9.  POP   00H
10. POP   01H
11. POP   00H
12. MOV   SP,#32H
13. POP   00H
```



元件原理

- PUSH 與 POP

```
1.   MOV   SP,#30H
2.   MOV   R0,#01H
3.   PUSH  00H
4.   MOV   R0,#02H
5.   PUSH  00H
6.   MOV   R0,#03H
7.   PUSH  00H
8.   MOV   R0,#04H
9.   POP   00H
10.  POP   01H
11.  POP   00H
12.  MOV   SP,#32H
13.  POP   00H
```



元件原理

- 跳躍JUMP
  - 跳躍指令為直接使PC跳到指定位址，並繼續依序執行指令
  - 條件式跳躍指令
    - DJNZ、JZ、JNB
  - 無條件跳躍
    - JMP、AJMP、LJMP、SJMP

元件原理

- LJMP、SJMP、AJMP與 JMP比較
  - LJMP
    - 指令格式：LJMP　addr16
    - 將PC直接指向程式記憶體中addr16的位址
  - SJMP
    - 指令格式：SJMP　offset
    - 將PC加二後再加上 offset 使其指向目標位址
    - offset 為二補數，因此SJMP只能向前128或向後127

元件原理

- LJMP、SJMP、AJMP與 JMP比較
  - AJMP(absolute)
    - 指令格式：AJMP addr11
    - 先將PC加二後，再將前11位元改為與addr11相同
  - JMP
    - 指令格式：JMP @A+DPTR
    - 將PC指向ACC當前值加上DPTR後的位址
  - 以上跳躍法，指令格式都可以使用指定Label的方式，例如SJMP LABEL

元件原理

- LJMP、SJMP、AJMP與 JMP比較

	LJMP	SJMP	AJMP	JMP
指令長度 (Byte)	3	2	2	1
Cycle	2	2	2	2
跳躍範圍	Anywhere	256 Byte	2KB	256 Byte

元件原理

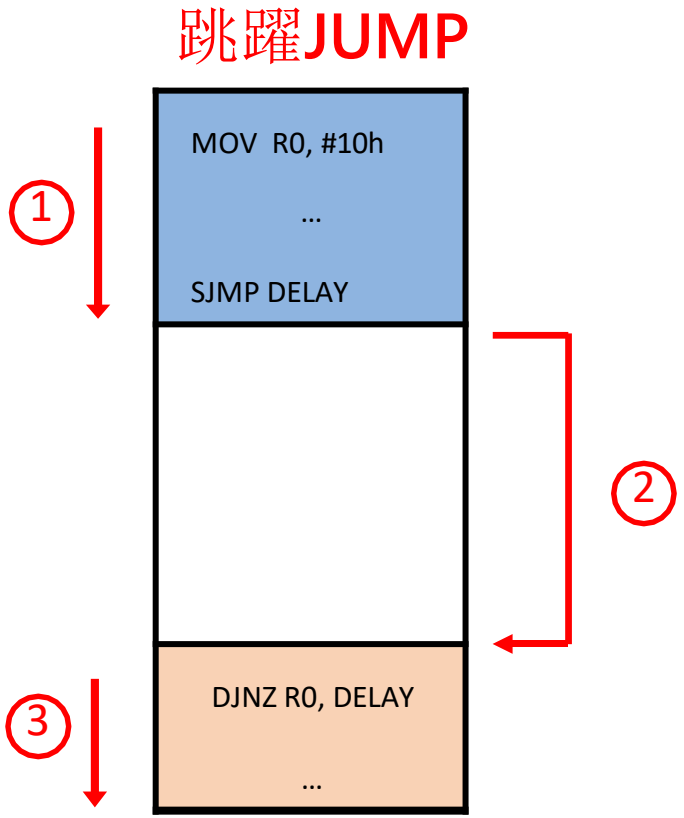
- 呼叫CALL
  - 沒有條件式呼叫
    - 僅有兩個指令格式：LCALL 和 ACALL
    - LCALL ：遠程呼叫，範圍為64K byte
    - ACALL：近程呼叫，範圍為2K byte
    - LCALL與ACALL的差別類似於LJMP與AJMP

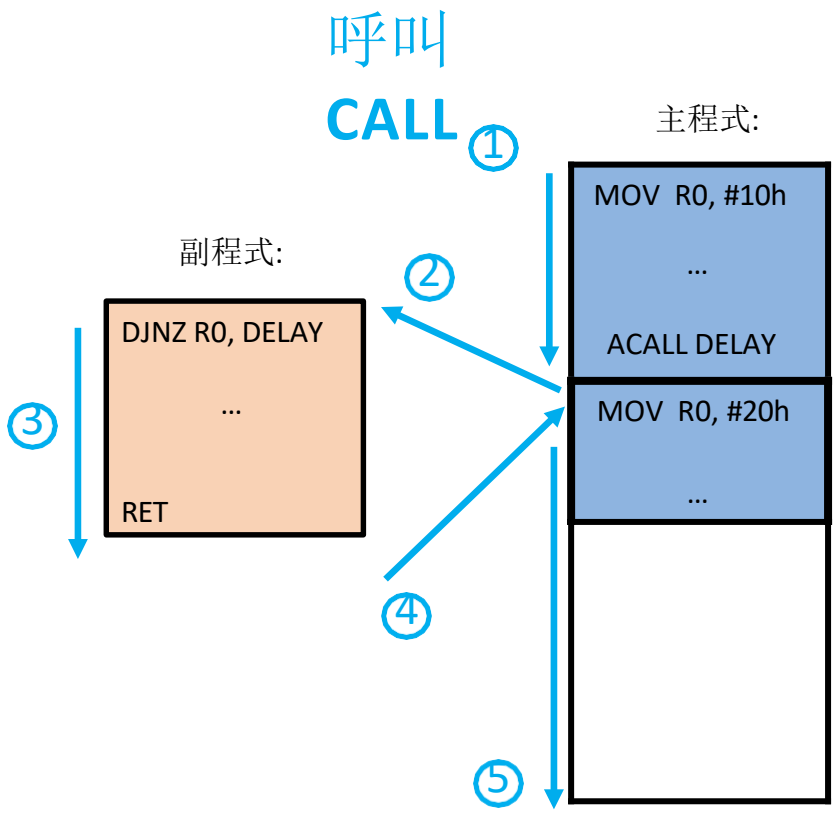
元件原理

- 呼叫CALL
  - 將PC移至指定的副程式位址，副程式執行完畢後再利用RET指令返回主程式
    - 原理：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，副程式執行完畢後再分別POP以取得PC應當返回的位址
  - 需要隨時注意堆疊的變化，以免發生錯誤

元件原理

- JMP 與 CALL比較





元件原理

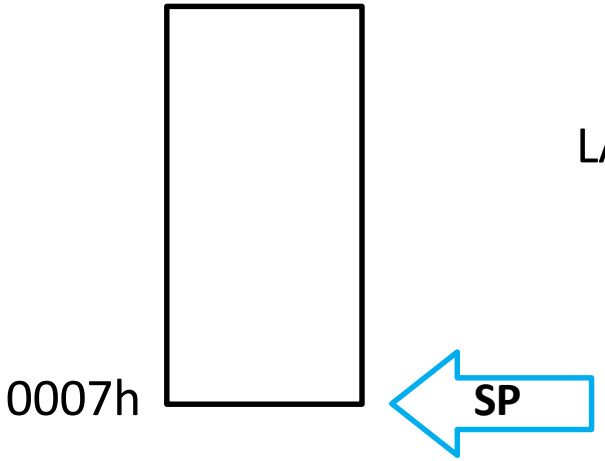
- 呼叫CALL(錯誤程式範例)

```
1.              ORG     0000h
2.              MOV     R0, #0FFh
3.              JMP     LABEL2
4.  LABEL1:     MOV     R0, #01h
5.              POP     00h
6.              RET
7.              MOV     R0, #02h
8.  LABEL2:     PUSH    00h
9.              ACALL   LABEL1
10.             JMP     LABEL2
11.             END
```

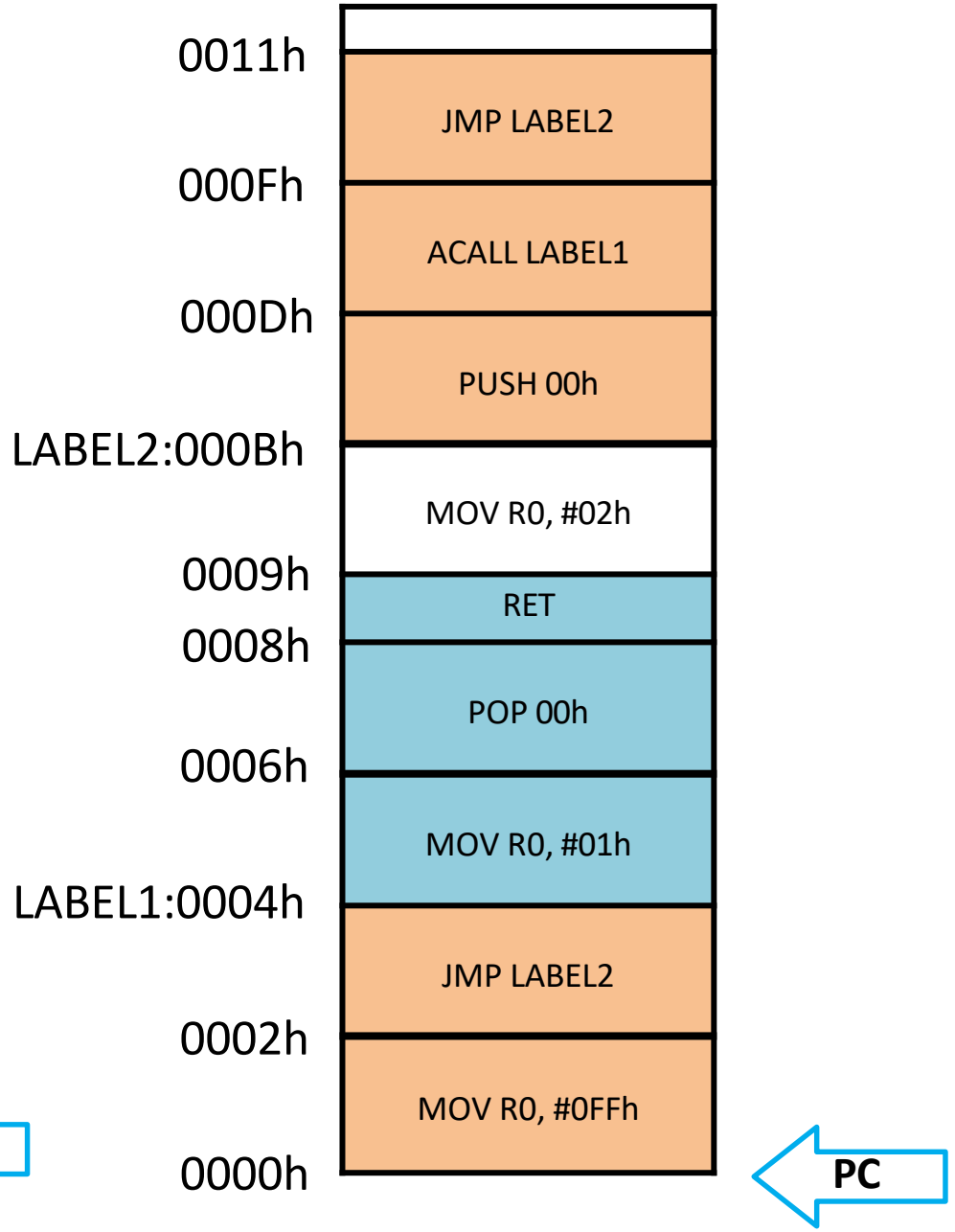
R0 = 00h

PC = 0000h

資料記憶體



程式記憶體



元件原理

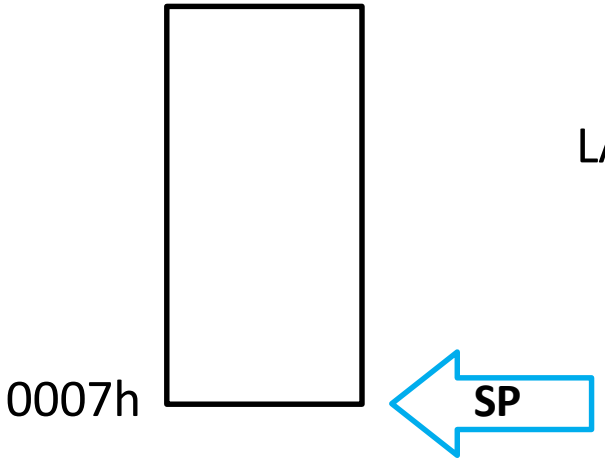
- 呼叫CALL(錯誤程式範例)

```
1.              ORG     0000h
2.              MOV     R0, #0FFh
3.              JMP     LABEL2
4.  LABEL1:     MOV     R0, #01h
5.              POP     00h
6.              RET
7.              MOV     R0, #02h
8.  LABEL2:     PUSH    00h
9.              ACALL   LABEL1
10.             JMP     LABEL2
11.             END
```

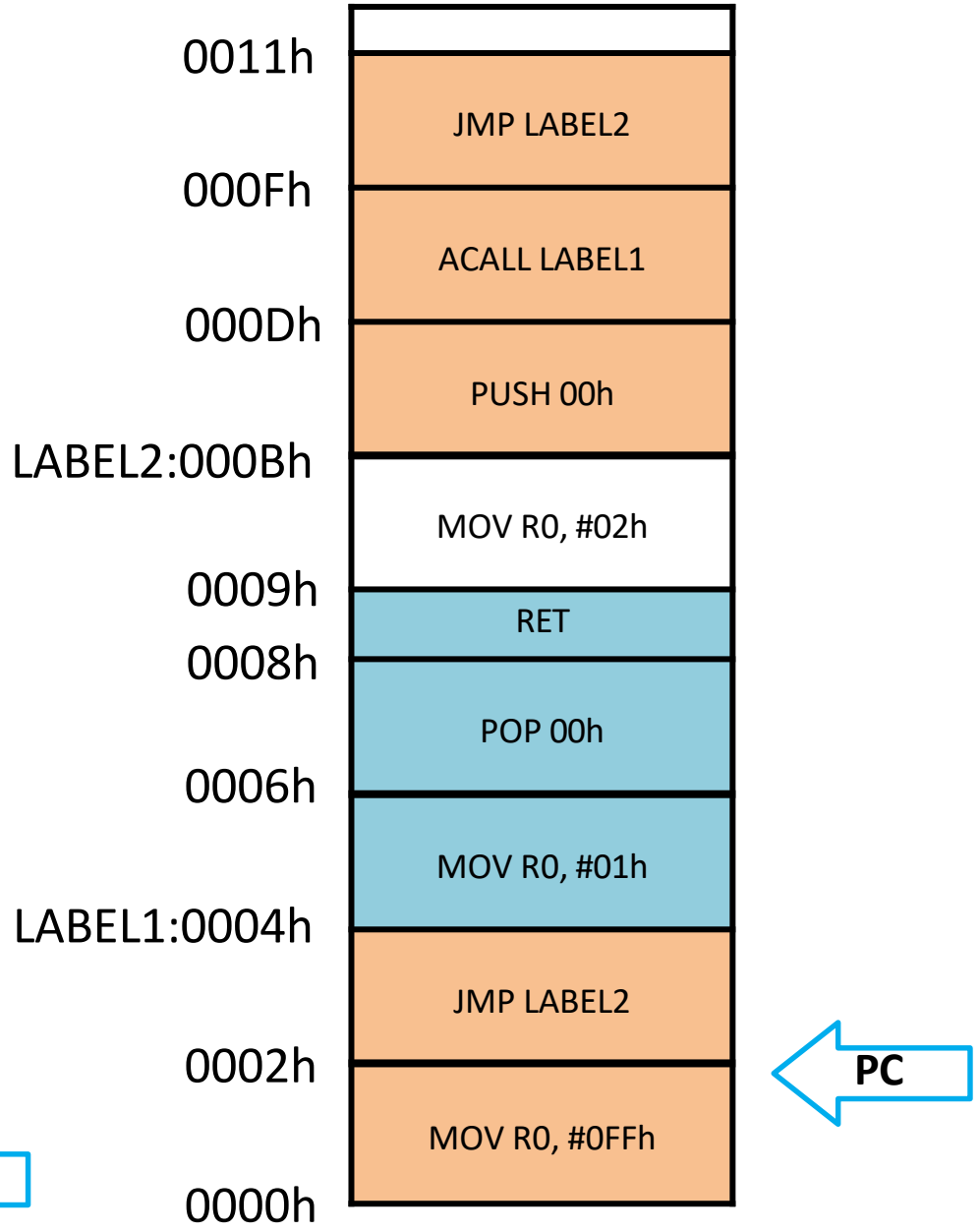
R0 = FFh

PC = 0002h

資料記憶體



程式記憶體



元件原理

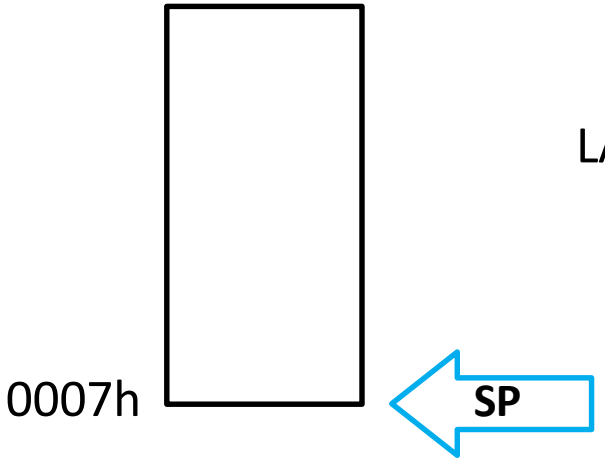
- 呼叫CALL(錯誤程式範例)

```
1.              ORG     0000h
2.              MOV     R0, #0FFh
3.              JMP     LABEL2
4.  LABEL1:     MOV     R0, #01h
5.              POP     00h
6.              RET
7.              MOV     R0, #02h
8.  LABEL2:     PUSH    00h
9.              ACALL   LABEL1
10.             JMP     LABEL2
11.             END
```

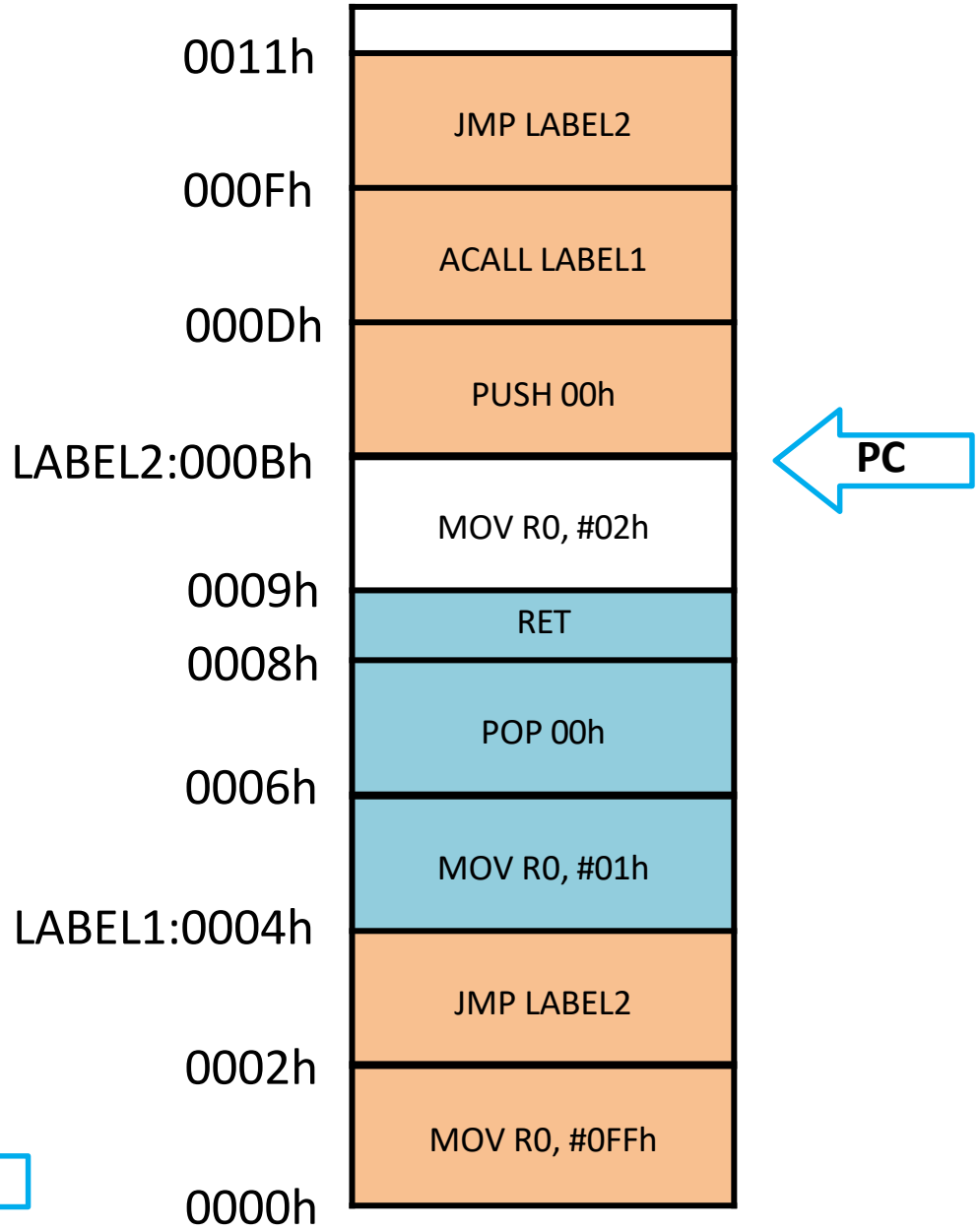
R0 = FFh

PC = 000Bh

資料記憶體



程式記憶體



元件原理

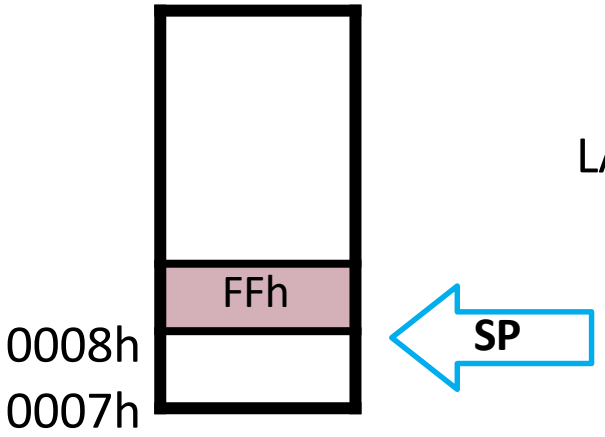
- 呼叫CALL(錯誤程式範例)

```
1.                ORG     0000h
2.                MOV     R0, #0FFh
3.                JMP     LABEL2
4.  LABEL1:       MOV     R0, #01h
5.                POP     00h
6.                RET
7.                MOV     R0, #02h
8.  LABEL2:       PUSH    00h
9.                ACALL   LABEL1
10.               JMP     LABEL2
11.               END
```

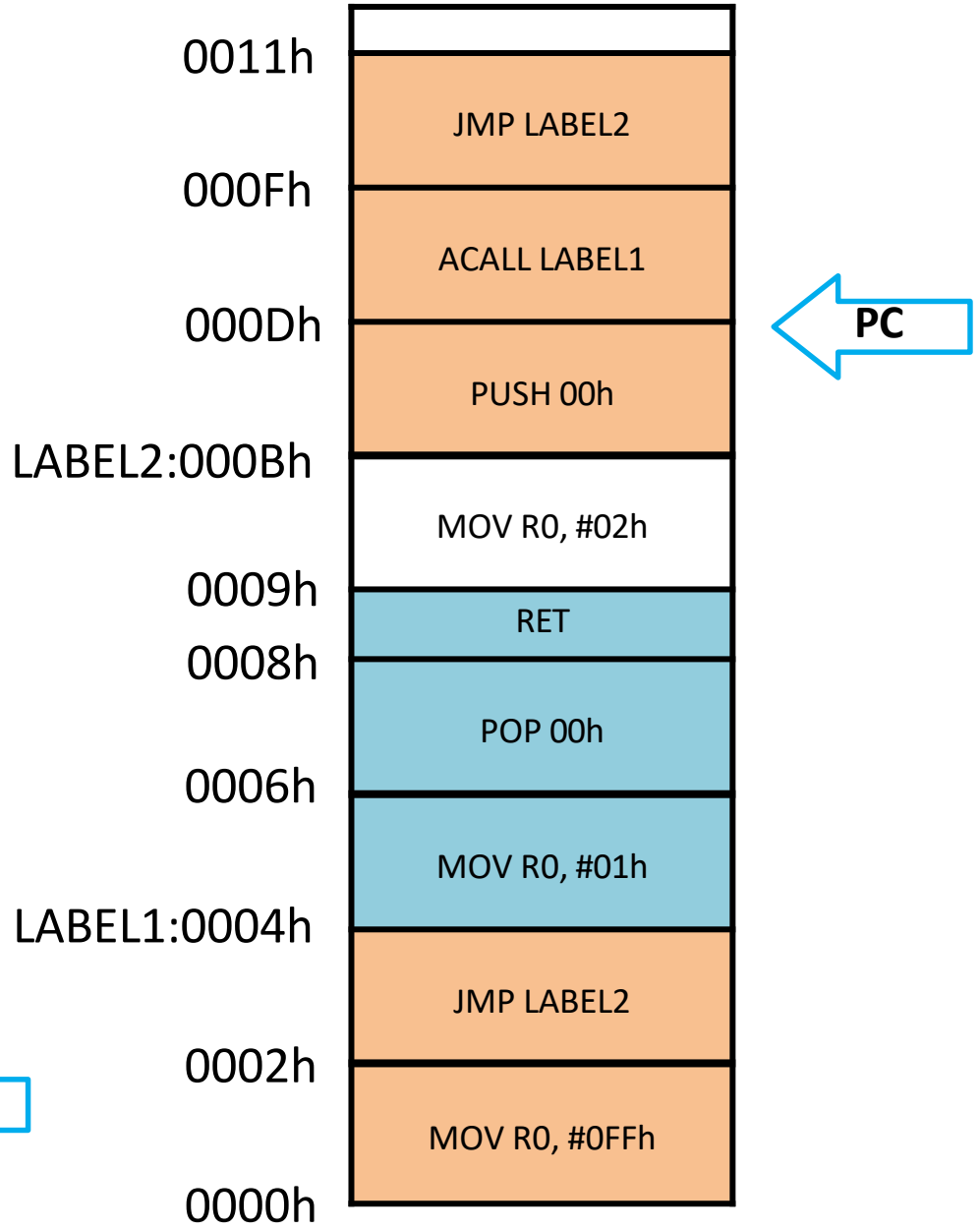
R0 = FFh

PC = 000Dh

資料記憶體



程式記憶體



元件原理

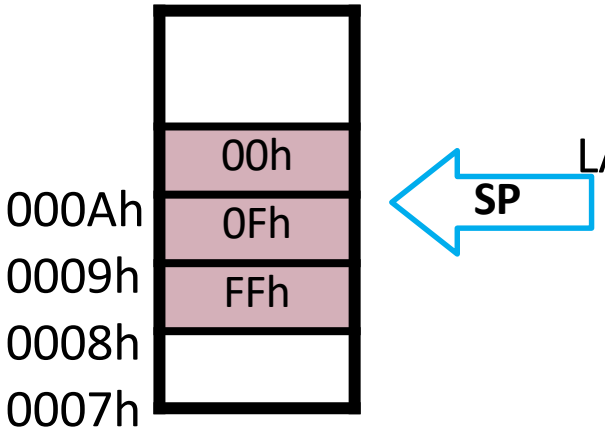
- 呼叫CALL(錯誤程式範例)

```
1.              ORG     0000h
2.              MOV     R0, #0FFh
3.              JMP     LABEL2
4.  LABEL1:     MOV     R0, #01h
5.              POP     00h
6.              RET
7.              MOV     R0, #02h
8.  LABEL2:     PUSH    00h
9.              ACALL   LABEL1
10.             JMP     LABEL2
11.             END
```

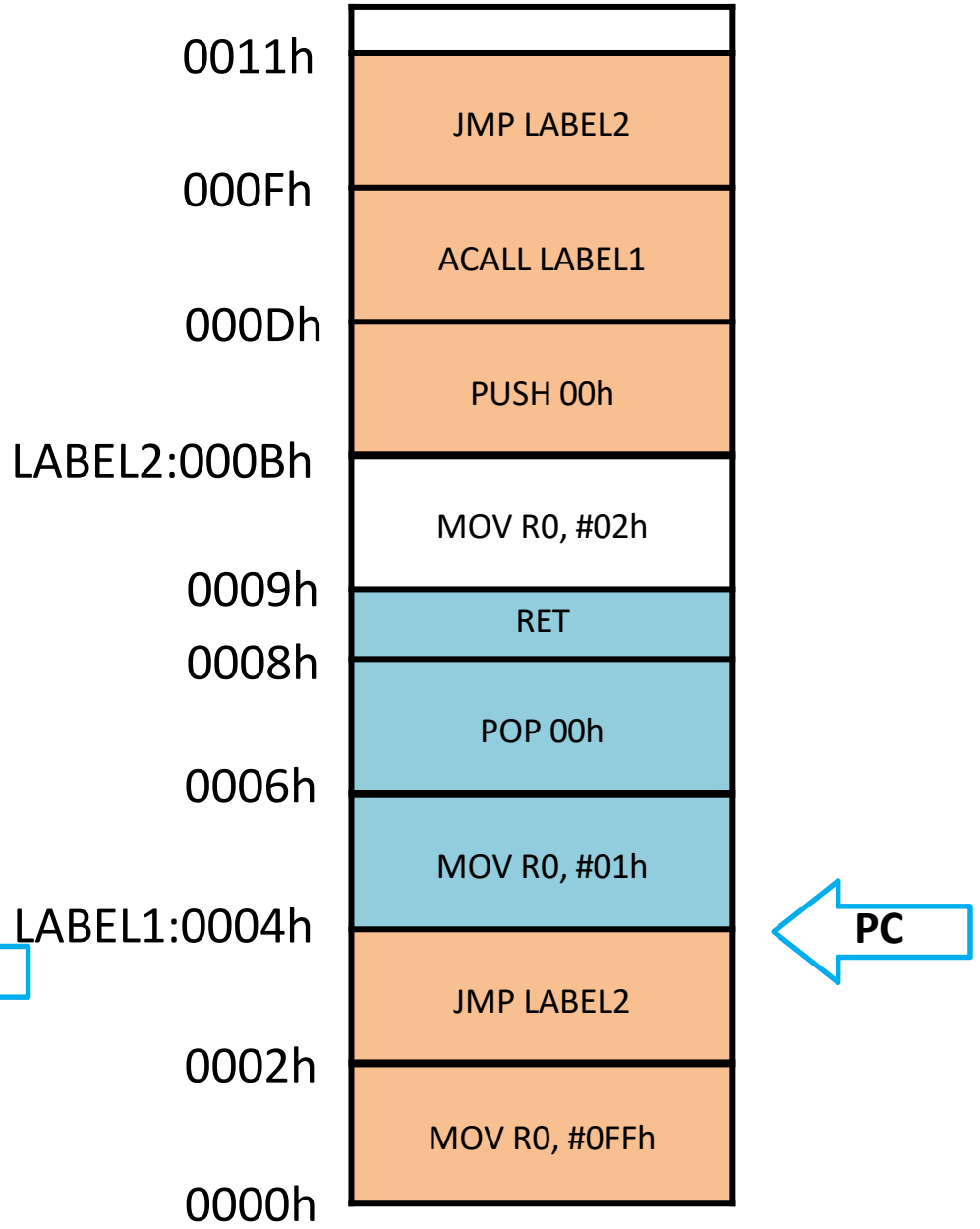
R0 = FFh

PC = 0004h

資料記憶體



程式記憶體



元件原理

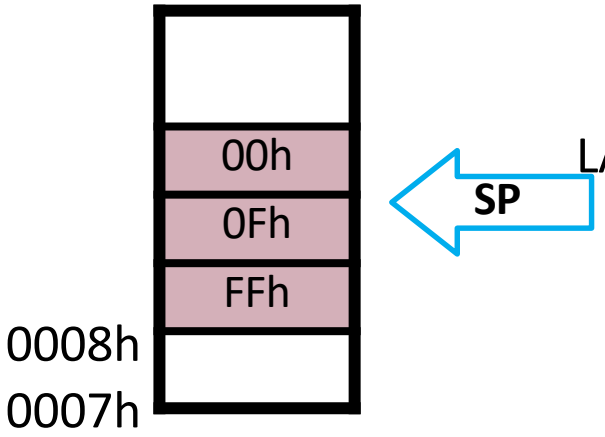
- 呼叫CALL(錯誤程式範例)

```
1.              ORG     0000h
2.              MOV     R0, #0FFh
3.              JMP     LABEL2
4.  LABEL1:     MOV     R0, #01h
5.              POP     00h
6.              RET
7.              MOV     R0, #02h
8.  LABEL2:     PUSH    00h
9.              ACALL   LABEL1
10.             JMP     LABEL2
11.             END
```

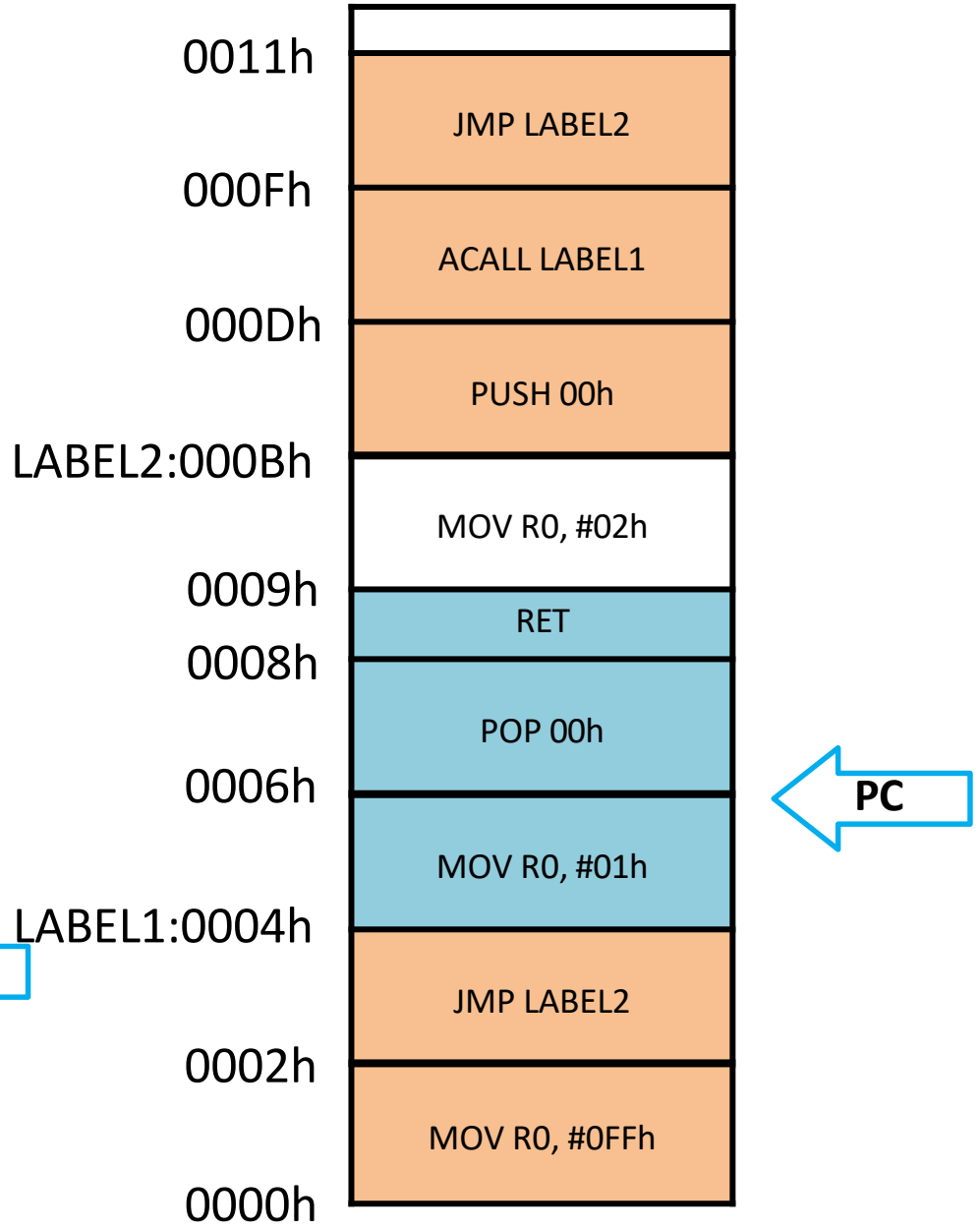
R0 = 01h

PC = 0006h

資料記憶體



程式記憶體



元件原理

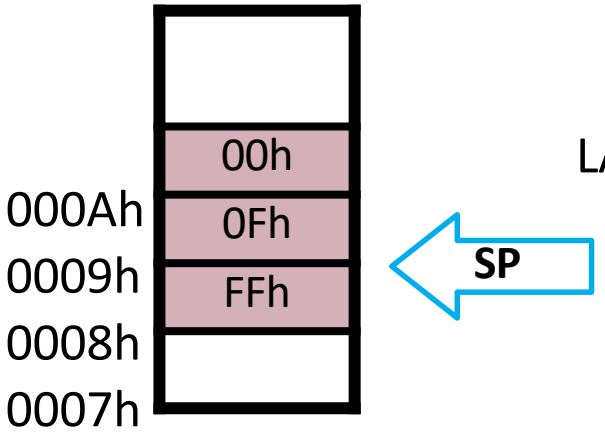
- 呼叫CALL(錯誤程式範例)

```
1.              ORG     0000h
2.              MOV     R0, #0FFh
3.              JMP     LABEL2
4.  LABEL1:     MOV     R0, #01h
5.              POP     00h
6.              RET
7.              MOV     R0, #02h
8.  LABEL2:     PUSH    00h
9.              ACALL   LABEL1
10.             JMP     LABEL2
11.             END
```

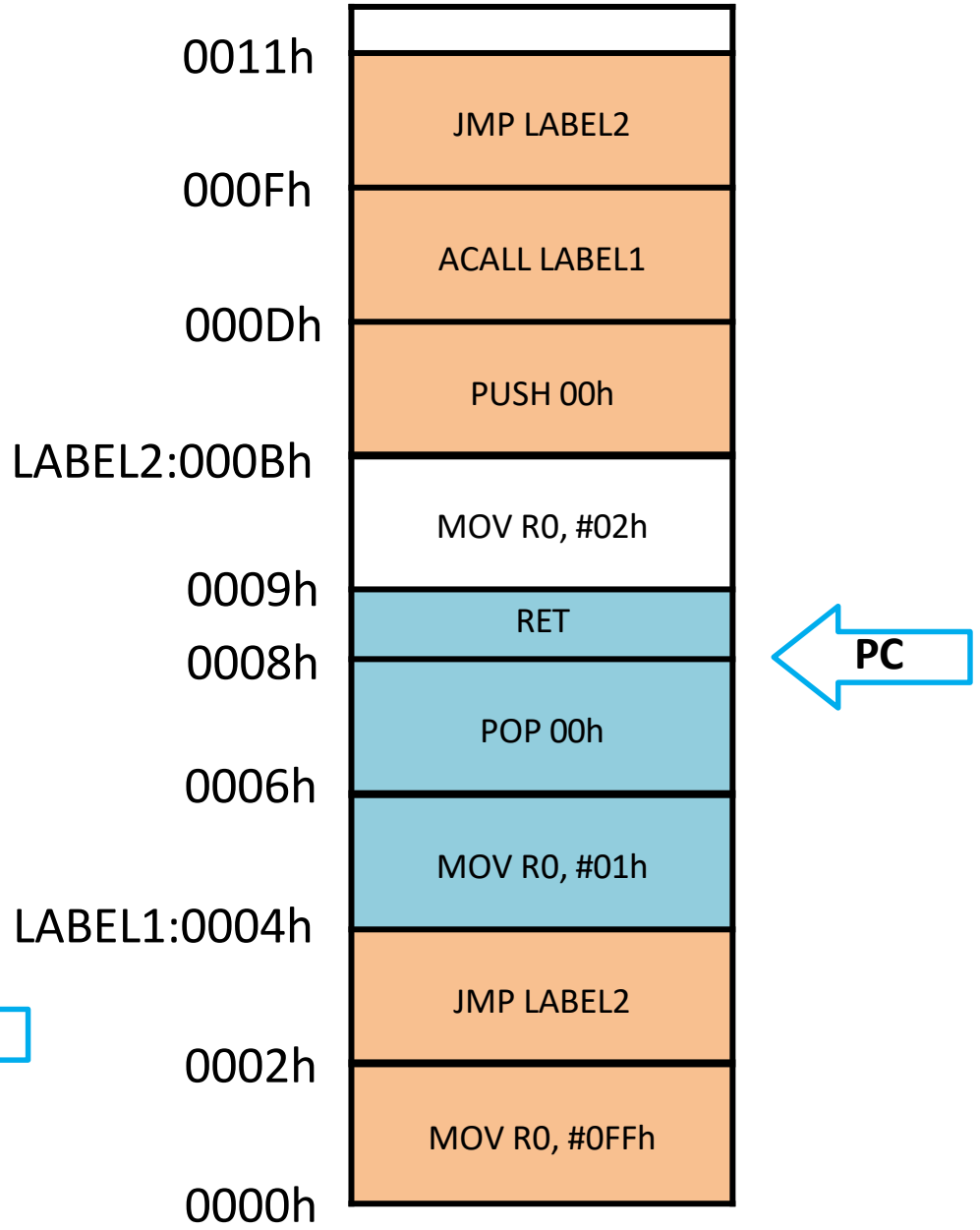
R0 = 00h

PC = 0008h

資料記憶體



程式記憶體



元件原理

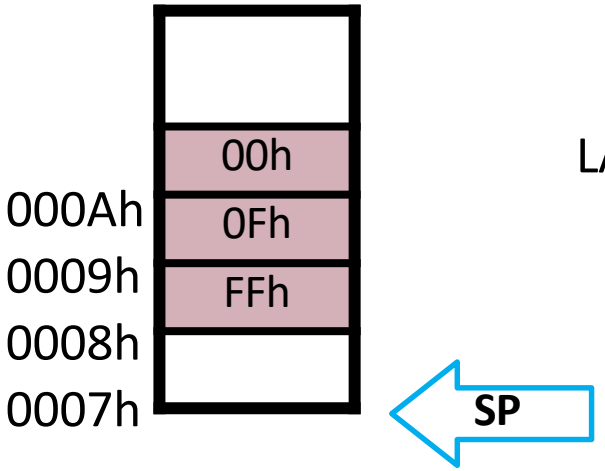
- 呼叫CALL(錯誤程式範例)

```
1.                ORG     0000h
2.                MOV     R0, #0FFh
3.                JMP     LABEL2
4.  LABEL1:       MOV     R0, #01h
5.                POP     00h
6.                RET
7.                MOV     R0, #02h
8.  LABEL2:       PUSH    00h
9.                ACALL   LABEL1
10.               JMP     LABEL2
11.               END
```

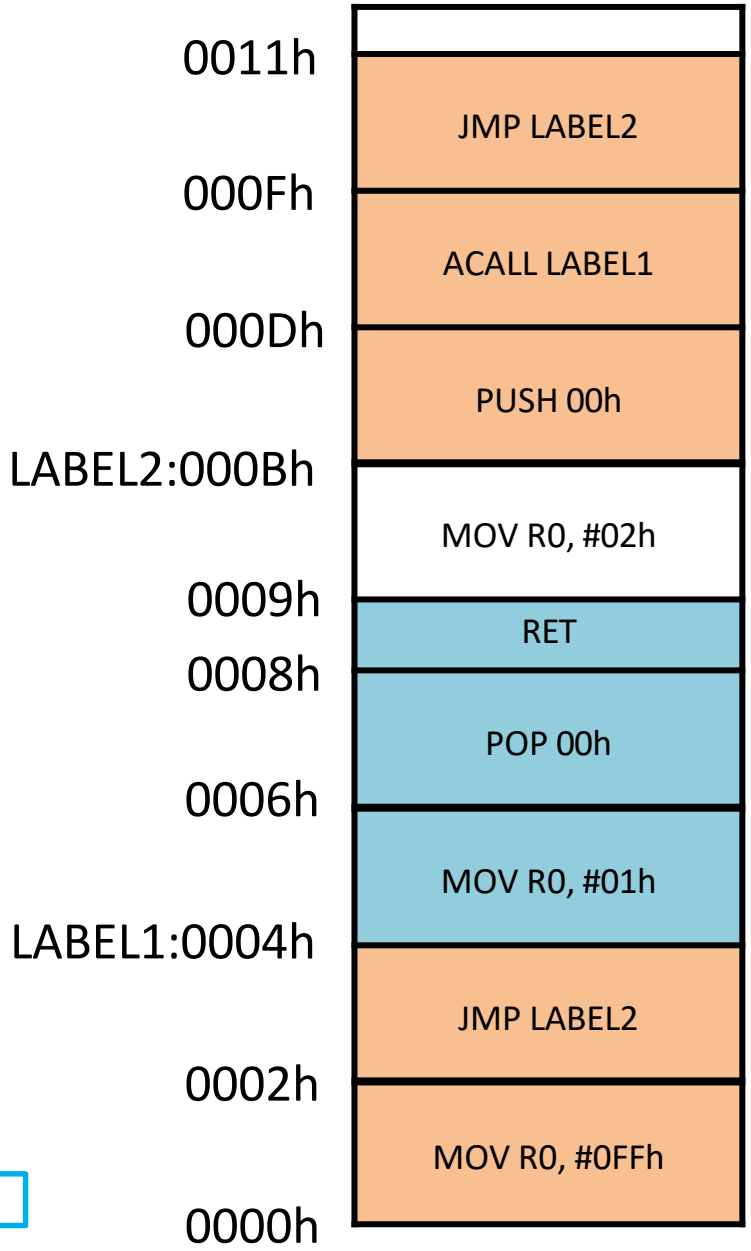
R0 = 00h

PC = 0FFFh

資料記憶體



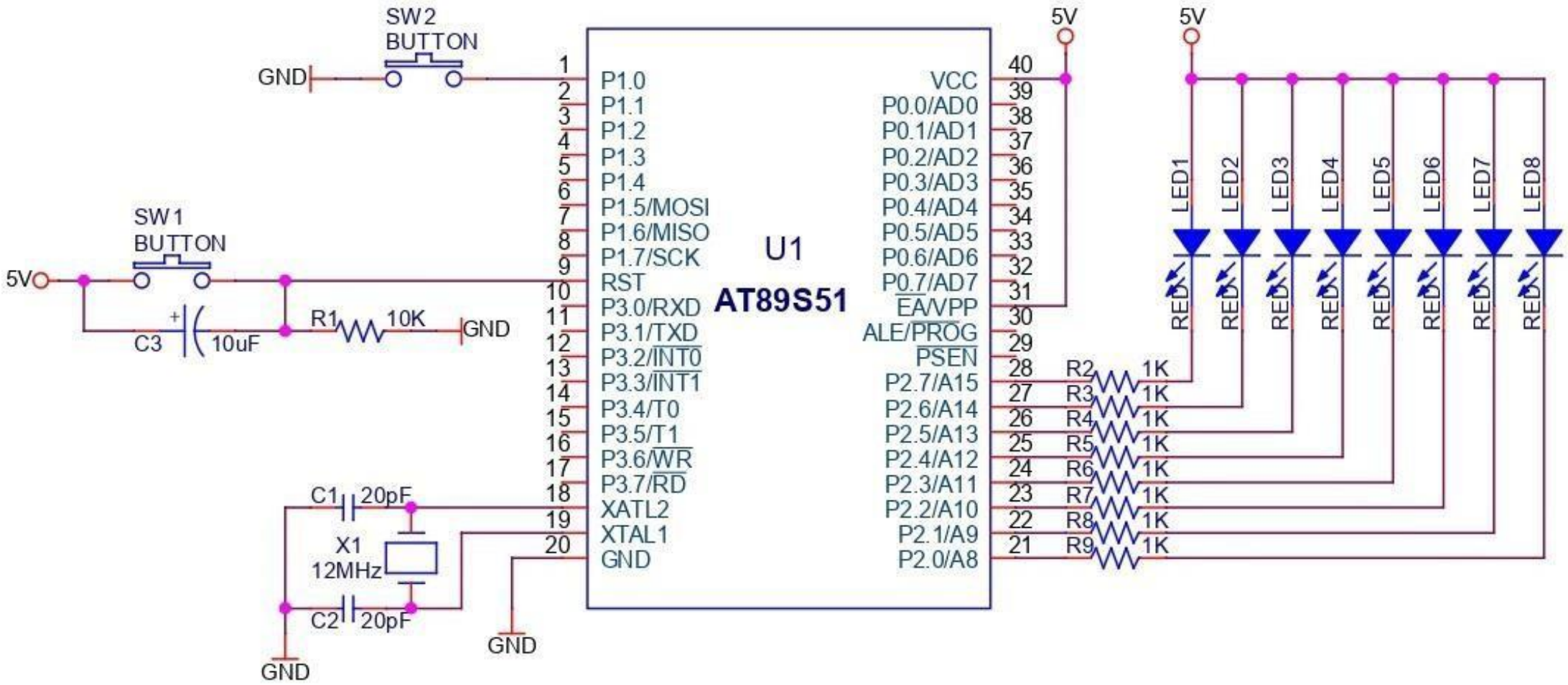
程式記憶體



Outline

- 學習重點
- 實驗內容
- 材料清單
- 元件原理
- 實驗電路圖
- 軟體流程圖
- 實驗程式

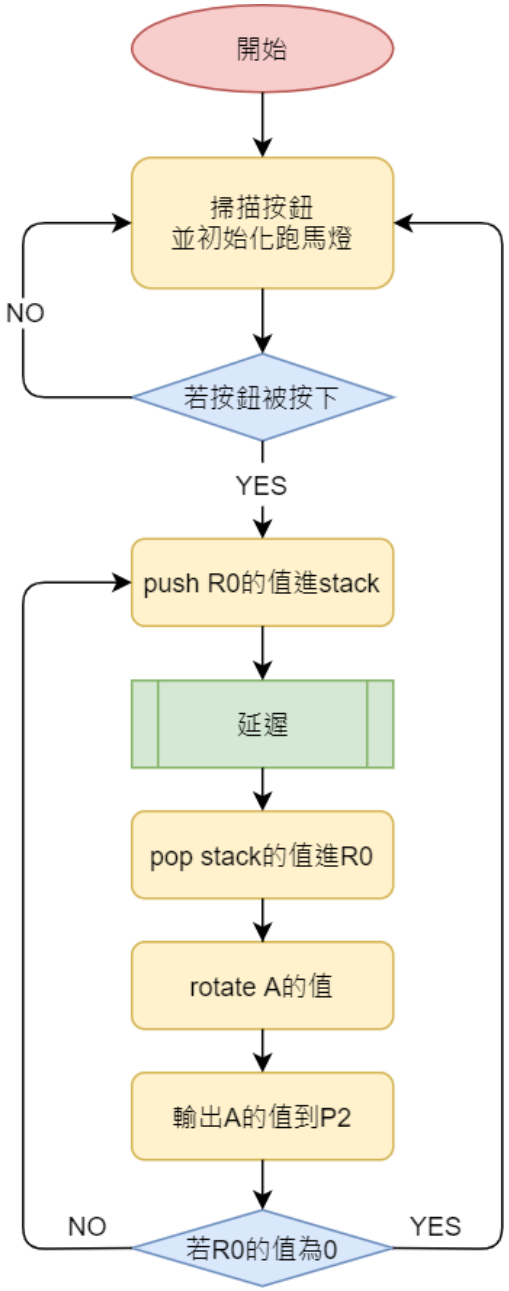
實驗電路圖


SW2
BUTTON
GND
SW1
BUTTON
5V
C3
10uF
R1
10K
GND
C1
20pF
X1
12MHz
C2
20pF
GND
GND
U1
AT89S51
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5/MOSI
P1.6/MISO
P1.7/SCK
RST
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
XATL2
XTAL1
GND
VCC
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
EA/VPP
ALE/PROG
PSEN
P2.7/A15
P2.6/A14
P2.5/A13
P2.4/A12
P2.3/A11
P2.2/A10
P2.1/A9
P2.0/A8
5V
5V
R2 1K
R3 1K
R4 1K
R5 1K
R6 1K
R7 1K
R8 1K
R9 1K
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
RED

Outline

- 學習重點
- 實驗內容
- 材料清單
- 元件原理
- 實驗電路圖
- 軟體流程圖
- 實驗程式

軟體流程圖


開始
掃描按鈕
並初始化跑馬燈
NO
若按鈕被按下
YES
push R0的值進stack
延遲
pop stack的值進R0
rotate A的值
輸出A的值到P2
NO
若R0的值為0
YES

Outline

- 學習重點
- 實驗內容
- 材料清單
- 元件原理
- 實驗電路圖
- 軟體流程圖
- 實驗程式

實驗程式

```
ORG     0000h
JMP     LOOP                ;jump into loop
ORG     0030h
LOOP:   MOV     SP, #32h    ;SP = #32H
        MOV     A, #0xFE    ;A = #0xfe
        MOV     P2, A       ;P2 = A
        SETB    P1.0        ;set p1.0 to high
        MOV     R0, #8D     ;set the execution times of marquee
        JNB     P1.0, MARQUEE   ;jump into marquee when p1.0 is low
        JMP     LOOP        ;infinite loop
```

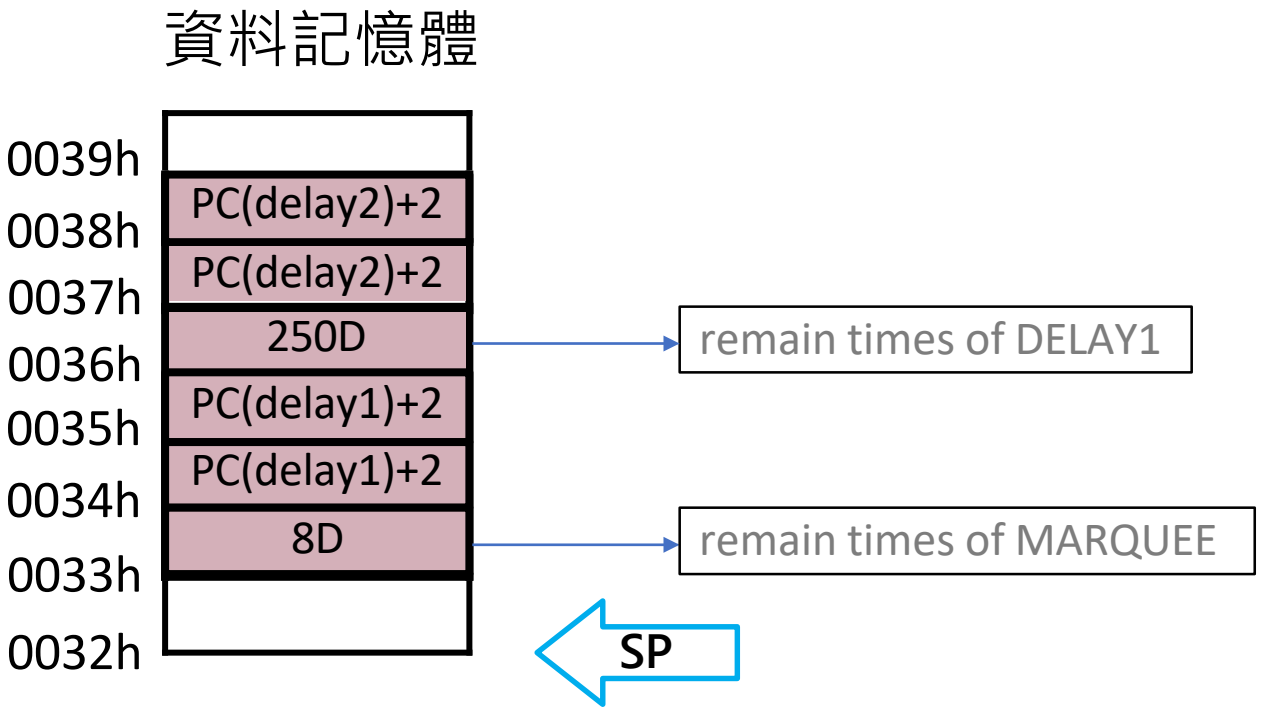
實驗程式

```
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             ;pop out the value of R0 which is pushed in
                                        line11
15.             RL      A               ;left rotate
16.             MOV     P2, A           ;set the value of A into P2
17.             DJNZ    R0, MARQUEE     ;loop back until MARQUEE execute 8 times
18.             JMP     LOOP            ;end of MARQUEE, back to LOOP
```

實驗程式

```
DELAY1:    PUSH    00h          ;push the remain times of DELAY1 into stack
           MOV     R0, #250D    ;set the execution times of DELAY2
           CALL    DELAY2       ;call DELAY2
           POP     00h          ;pop the remain times of DELAY1 back to R0
           DJNZ    R0, DELAY1   ;loop until R0 is 0
           RET                  ;return to MARQUEE
DELAY2:    DJNZ    R0, DELAY2   ;loop until R0 is 0
           RET                  ;return to DELAY1
           END
```

實驗中資料記憶體的值


資料記憶體
0039h
0038h
0037h
0036h
0035h
0034h
0033h
0032h
PC(delay2)+2
PC(delay2)+2
250D
PC(delay1)+2
PC(delay1)+2
8D
remain times of DELAY1
remain times of MARQUEE
SP

Q&A