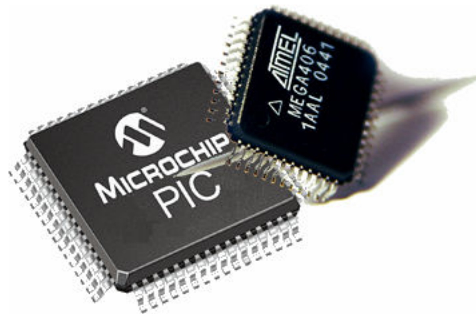
Subroutines & Software Delay Routines



Subroutines & Software Delay Routines

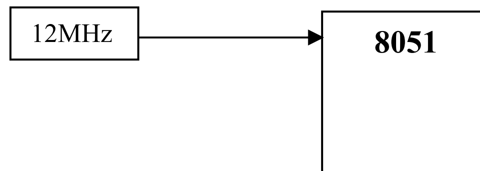
This chapter introduces software based timing delay routines. The examples introduce the useful programming concept of sub-routines.

For an 8051 microcomputer a single instruction cycle is executed for every 12 clock cycles of the processor clock. Thus, for an 8051 clocked at 12MHz. the instruction cycle time is one microsecond, as follows:

$$\textbf{Instruction cycle time} = \frac{\textbf{12 clock cycles}}{\textbf{12 x } 10^{6} \textbf{ cycles/sec.}} = 10^{-6} \textbf{ seconds, or 1 } \mu\textbf{sec.}$$

The shortest instructions will execute in one instruction cycle, i.e. 1 µsec. Other instructions may take two or more instruction cycle times to execute.

A given instruction will take one or more instruction cycles to execute (e.g. 1, 2 or 3 µsecs.)



3.1 SOME EXAMPLE ROUTINES

Sample routine to delay 1 millisecond: **ONE_MILLI_SUB**
Consider a software routine called 'ONE_MILLI_SUB',written as a subroutine program, which takes a known 1000 instructions cycles (approx.) to execute. Thus it takes 1000 µsecs, or 1 millisecond, to execute. The program is written as a subroutine since it may be called on frequently as part of some longer timing delay routines. The flow chart for the routine is shown in figure 3.1 and the source code for the subroutine is shown in listing 3.1. Note, register R7 is used as a loop counter. It is good practice in writing subroutines to save to the stack (PUSH) any registers used in the subroutine and to restore (POP) such registers when finished. See how R7 is saved and retrieved in the program. We say that R7 is 'preserved'. This is important as the program which called the subroutine may be using R7 for another purpose and a subroutine should not be allowed to 'accidentally' change the value of a register used elsewhere.

When calling a subroutine the Program Counter (PC) is automatically pushed onto the stack, so the SP (Stack Pointer) is incremented by 2 when a subroutine is entered. The PC is automatically retrieved when returning from the subroutine, decrementing the SP by 2.

In the ONE_MILLI_SUB subroutine a tight loop is executed 250 times. The number of instruction cycles per instruction is known, as published by the 8051 manufacturer. The tight loop is as follows:

NOP	*takes 1 instruction cycle to execute*	
NOP	*takes 1 instruction cycle to execute*	
DJNZ R7, LOOP_1_MILLI	*takes 2 instruction cycle to execute*	
Total instruction cycles =	*4*	

So, it takes 4 instruction cycles, or 4 µsecs, to execute the loop. Thus, if we execute the loop 250 times it will take a 1000 µsecs (250 x 4), i.e. 1 millisecond, to complete the loops.

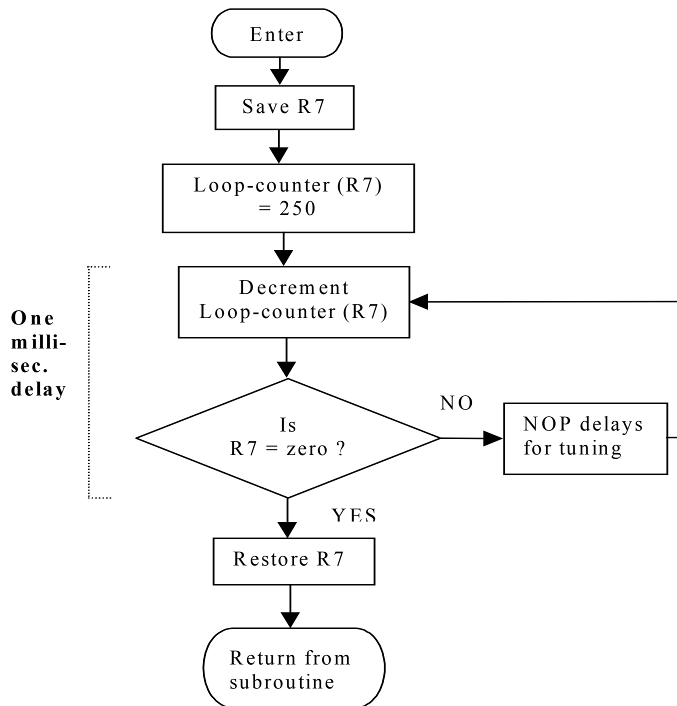


Figure 3.1 ONE_MILLI_SUB flow chart

```
;==============================================================
; ONE_MILLI_SUB:
; Subroutine to delay ONE millisecond
; Uses register R7 but preserves this register
;==============================================================
ONE_MILLI_SUB:

        PUSH 07h                    ; save R7 to stack
        MOV R7, #250d               ; 250 decimal to R7 to count 250 loops

LOOP_1_MILLI:                       ; loops  250 times
        NOP                         ; inserted NOPs to cause delay
        NOP                         ;
        DJNZ R7, LOOP_1_MILLI       ; decrement R7, if not zero loop back

        POP 07h                     ; restore R7 to original value

        RET                         ; return from subroutine
```

Listing 3.1 Source code for: ONE_MILLI_SUB

Sample routine to delay 1 second: ONE_SEC_SUB

The ONE_SEC_SUB subroutine, when called, causes a delay of ONE second. This subroutine calls the ONE_MILLI_SUB subroutine and is structured so that the ONE_MILLI_SUB subroutine is called exactly 1000 times, thus causing a total delay of 1000 milli. seconds, i.e. ONE second. (There are some small inaccuracies, which will be ignored for now). Note, R7 is used again as the loop counter (we could have used another register). Since R7 is preserved in the ONE_SEC_SUB subroutine, its value is not corrupted within the ONE_SEC_SUB subroutine. This example shows how one subroutine can call another subroutine, demonstrating the concept of *subroutine nesting*. It is interesting to track the value of the Stack Pointer (SP) during program operation. The flow chart for the ONE_SEC_SUB routine is shown in figure 3.2 and the source code is shown in listing 3.2.

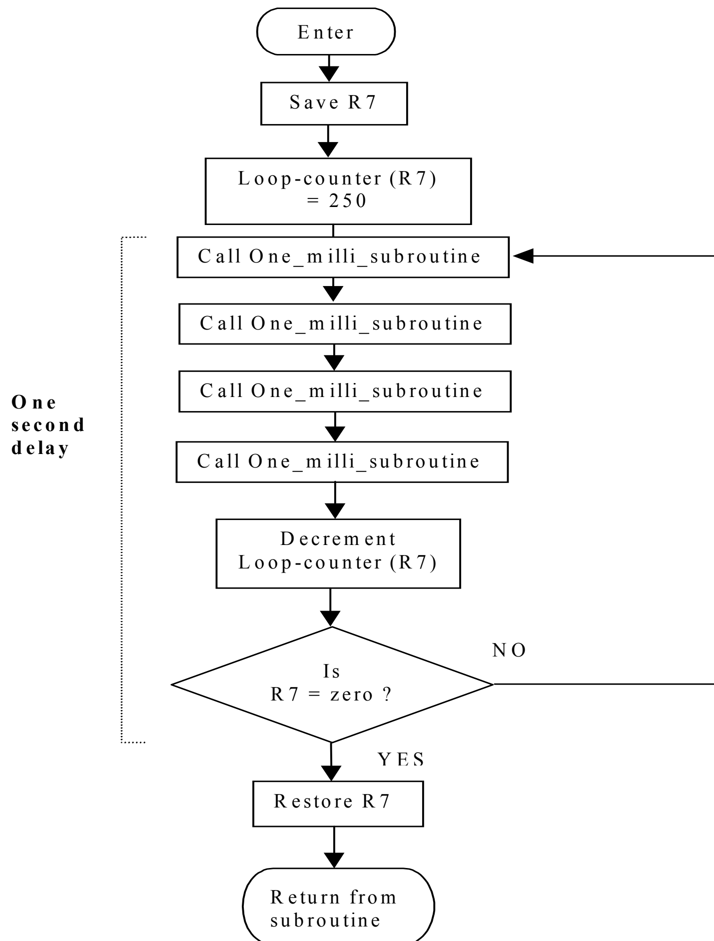


Figure 3.2 ONE_SEC_SUB flow chart

```
;=============================================================
; ONE_SEC_SUB
; Subroutine to delay ONE second
; Uses register R7 but preserves this register
;=============================================================

ONE_SEC_SUB:

        PUSH 07h                    ; save R7 to stack

        MOV R7, #250d               ; 250 decimal to R7 to count 250 loops

LOOP_SEC:                           ; Calls 4 one millisec. delays, 250 times

        LCALL ONE_MILLI_SUB         ; call subroutine to delay 1 millisecond
        LCALL ONE_MILLI_SUB         ; call subroutine to delay 1 millisecond
        LCALL ONE_MILLI_SUB         ; call subroutine to delay 1 millisecond
        LCALL ONE_MILLI_SUB         ; call subroutine to delay 1 millisecond

        DJNZ R7, LOOP_SEC           ; decrement R7, if not zero loop back

        POP 07h                     ; restore R7 to original value

        RET                         ; return from subroutine
```

Listing 3.2 Source code for: ONE_SEC_SUB

Sample routine to delay N seconds: PROG_DELAY_SUB

PROG_DELAY_SUB is a subroutine, which will cause a delay for a specified number of seconds. The subroutine is called with the required number, N, of delay seconds specified in the accumulator. The subroutine calls the ONE_SEC_SUB subroutine, which in turn calls the ONE_MILLI_SUB subroutine. Here is a further example of nesting subroutines. The PROG_DELAY_SUB subroutine preserves the accumulator value. The subroutine also checks to see if it has been called with a zero value in the accumulator. If this is the case the subroutine returns immediately without causing further delay. A maximum delay of 255 seconds can be specified, i.e. accumulator can have a maximum value of 0FFh (255 decimal). The program provides a simple example of passing a parameter to a subroutine where the accumulator is used to pass a number N into the subroutine. The flow chart for the PROG_DELAY_SUB routine is given in figure 3.3 and the assembly language source code is given in listing 3.3.

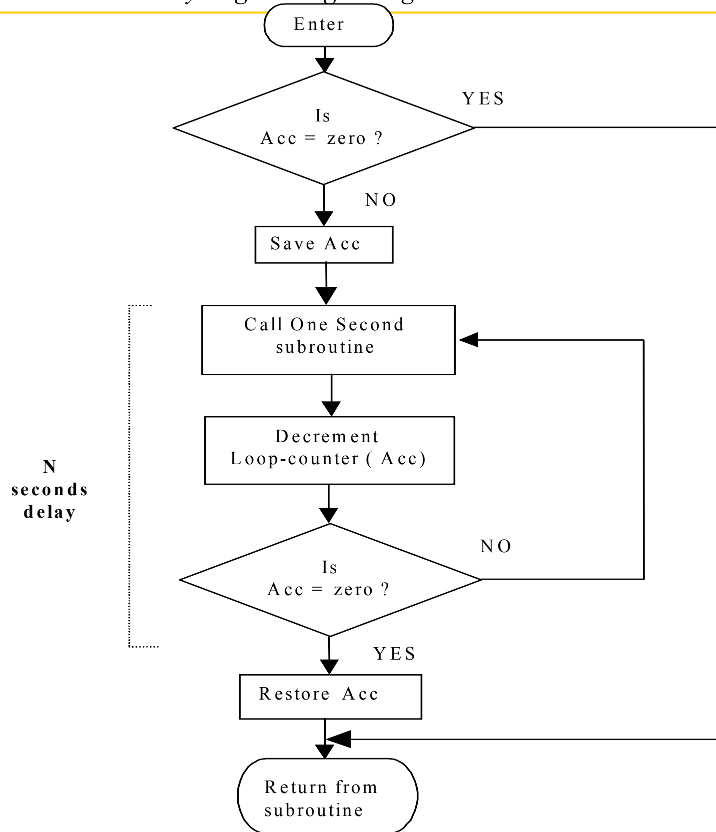


Figure 3.3 PROG_DELAY_SUB flow chart

```
;=========================================================================
; PROG_DELAY_SUB        Programmable Delay Subroutine
; Subroutine to delay N number of seconds. N is defined in A (accumulator)
; and passed to the subroutine. A is preserved.
; If N=0 the subroutine returns immediately. N max. value is FFh (255d)
;=========================================================================

PROG_DELAY_SUB:

        CJNE A, #00h, OK                ; if A=0 then exit
        LJMP DONE                       ; exit

OK:     PUSH Acc                        ; save A to stack

LOOP_N:                                 ; calls one second delay, no. of times in A

        LCALL  ONE_SEC_SUB              ; call subroutine to delay 1 second

         DJNZ Acc, LOOP_N               ; decrement A, if not zero loop back

        POP Acc                         ; restore Acc to original value
DONE:
        RET                             ; return from subroutine
```

Listing 3.3 Source code for: PROG_DELAY_SUB

Example application using a time delay

In this example an 8051 microcomputer, clocked at 12MHz., will be connected to a loudspeaker and a program will be written to sound the loudspeaker at a frequency of 500Hz. Figure 3.4 shows the hardware interface where the loudspeaker is connected to Port 1 at pin P1.0. A simple transistor is used as an amplifier as the 8051 output port does not have enough current drive capability to drive the loudspeaker directly. Figure 3.4 also shows a simple timing diagram to explain how the 500Hz. square wave is generated by the software. The ONE_MILLI_SUB subroutine is used to provide the basic time delay for each half cycle. Listing 3.4 shows the source code for the program.

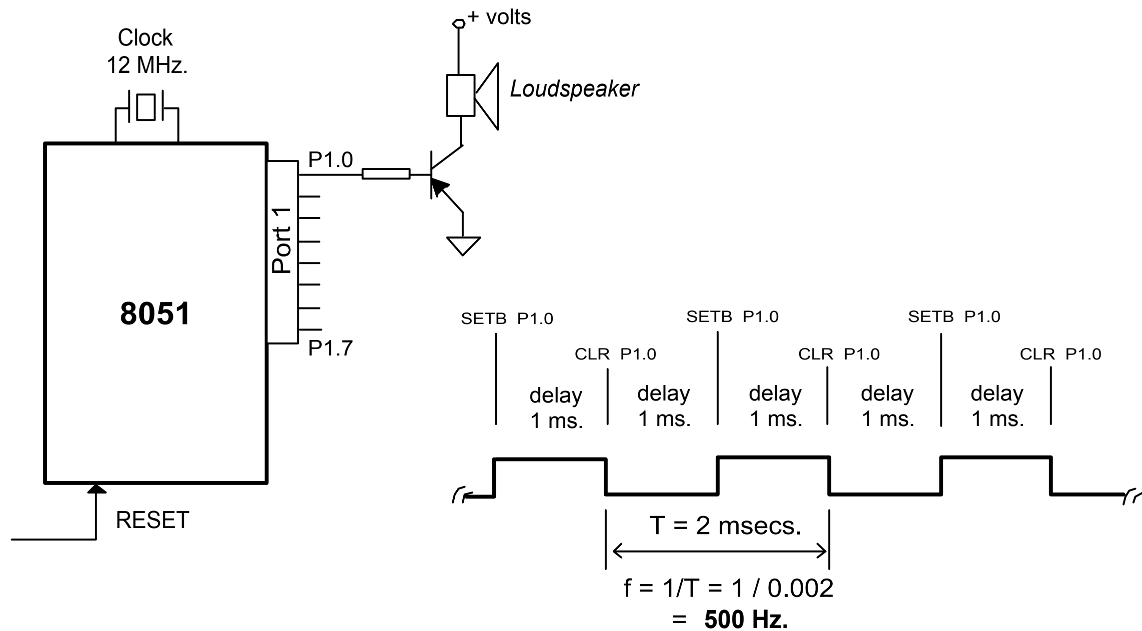


Figure 3.4 Hardware circuit with timing diagram

```
;=========================================
; SOUND.A51
; This program sounds a 500Hz. tone at Port 1, pin 0
;;=========================================

        ORG 0000h                   ; start address is 0000h

        MOV P1, #00                 ; clear all bits on P1

LOOP:
        SETB  P1.0                  ; set P1.0 high
        LCALL  ONE_MILLI_SUB        ; delay one millisecond
        CLR  P1.0                   ; set P1.0 low
        LCALL  ONE_MILLI_SUB        ; delay one millisecond

        LJMP  LOOP                  ; loop around!

;================================================
; ONE_MILLI_SUB:
; Subroutine to delay ONE millisecond
; Uses register R7 but preserves this register
;=================================================
ONE_MILLI_SUB:

        PUSH 07h                    ; save R7 to stack
        MOV R7, #250d                ; 250 decimal to R7 to count 250 loops

LOOP_1_MILLI:                       ; loops  250 times
        NOP                         ; inserted NOPs to cause delay
        NOP                         ;
        DJNZ R7, LOOP_1_MILLI       ; decrement R7, if not zero loop back

        POP 07h                     ; restore R7 to original value

        RET                         ; return from subroutine

END                                 ; end of program
```

Listing 3.4 Source code for example program to sound 500Hz. note

3.2 A NOTE ON THE OPERATION OF THE STACK POINTER

When a subroutine is called the current content of the Program Counter (PC) is save to the stack, the low byte of the PC is save first, followed by the high byte. Thus the Stack Pointer (SP) in incremented by 2. When a RET (return from subroutine) instruction is executed the stored PC value on the stack is restored to the PC, thus decrementing the SP by 2.

When a byte is PUSHed to the stack, the SP in incremented by one so as to point to the next available stack location. Conversely, when a byte is POP'ed from the stack the SP is decremented by one.

Figure 3.5 shows the organisation of the stack area within the I-RAM memory space.

The stack values during the operation of the nested subroutine example are shown in figure 3.6. Here it is assumed that the SP is initialised to 07h. This is possible where the alternative register banks are not used in a program. The stack then has a ceiling value of 20h, if we want to preserve the 'bit addressable' RAM area. It is probably more common to initialise the SP higher up in the internal RAM at location 2Fh. The diagram shows how data is saved to the stack.

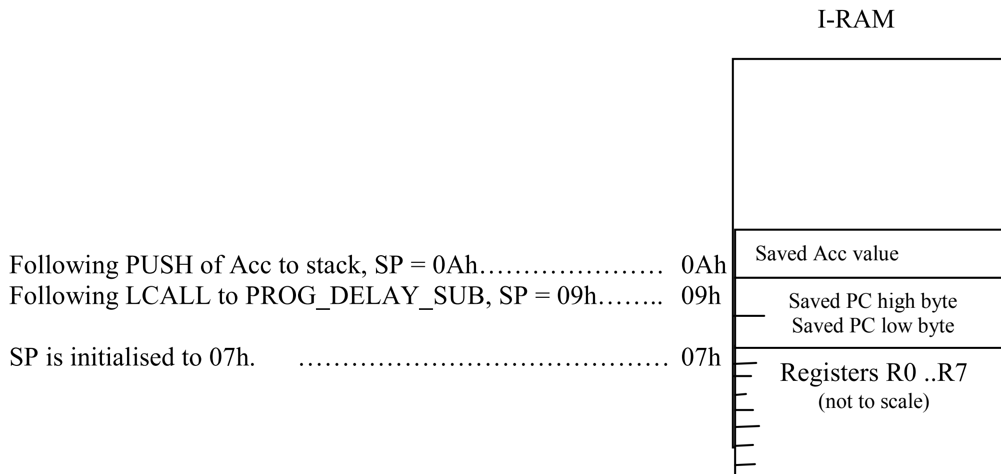
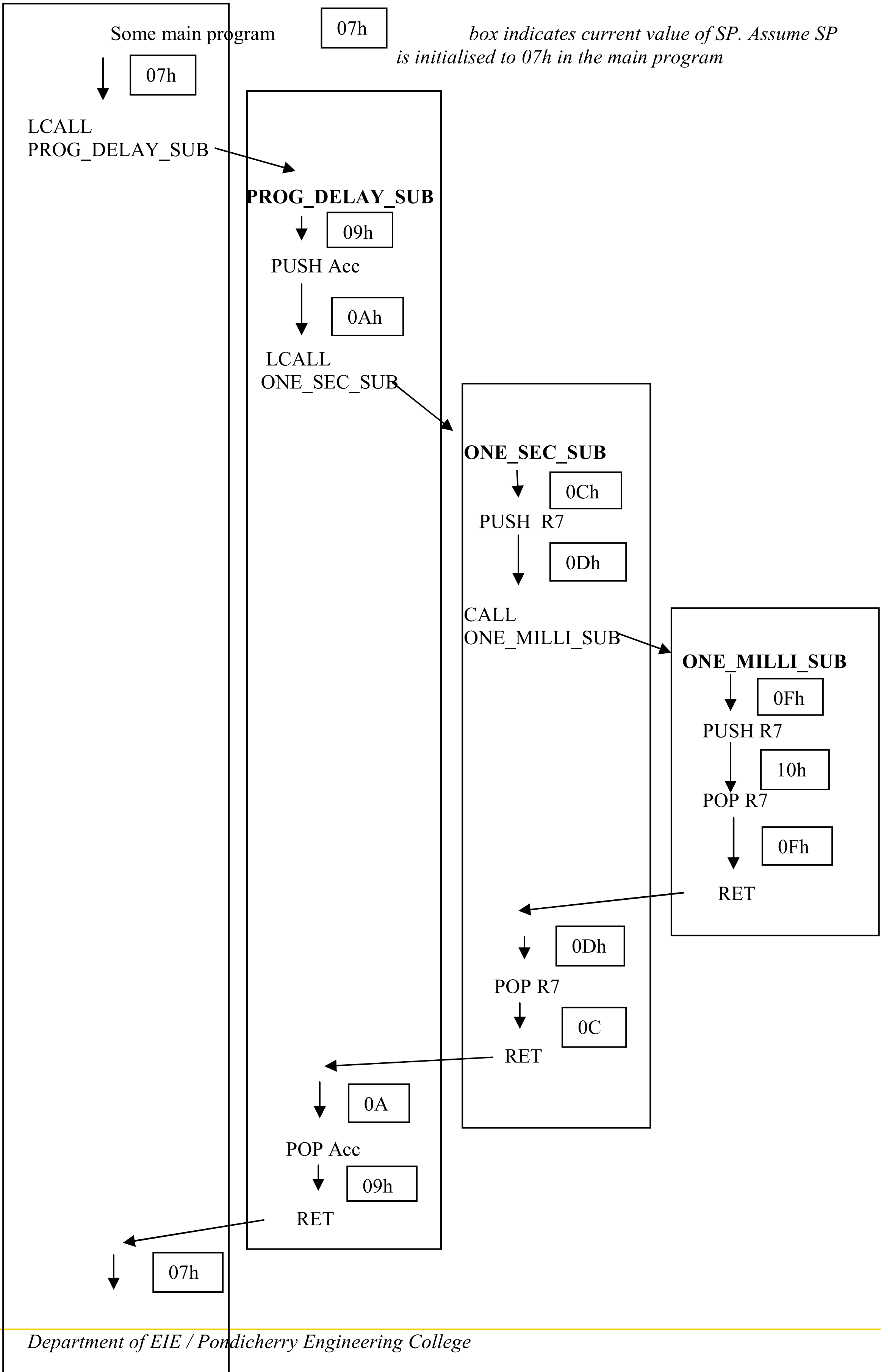


Figure 3.5 The stack operation


Some main program
07h
box indicates current value of SP. Assume SP is initialised to 07h in the main program
07h
LCALL
PROG_DELAY_SUB
PROG_DELAY_SUB
09h
PUSH Acc
0Ah
LCALL
ONE_SEC_SUB
ONE_SEC_SUB
0Ch
PUSH R7
0Dh
CALL
ONE_MILLI_SUB
ONE_MILLI_SUB
0Fh
PUSH R7
10h
POP R7
0Fh
RET
0Dh
POP R7
0C
RET
0A
POP Acc
09h
RET
07h

Main program

Figure 3.6 Example showing values of the Stack Pointer during nested subroutine operation.