# MECE336 Microprocessors I
# Subroutines

Dr. Kurtuluş Erinç Akdoğan

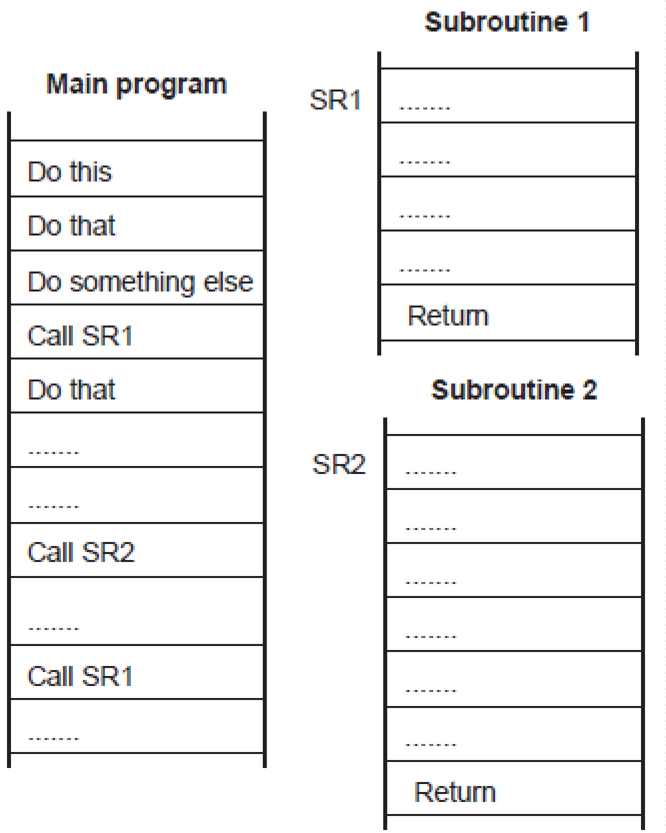
kurtuluserinc@cankaya.edu.tr

Course Webpage: http://MECE336.cankaya.edu.tr

ÇANKAYA ÜNİVERSİTESİ
MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ

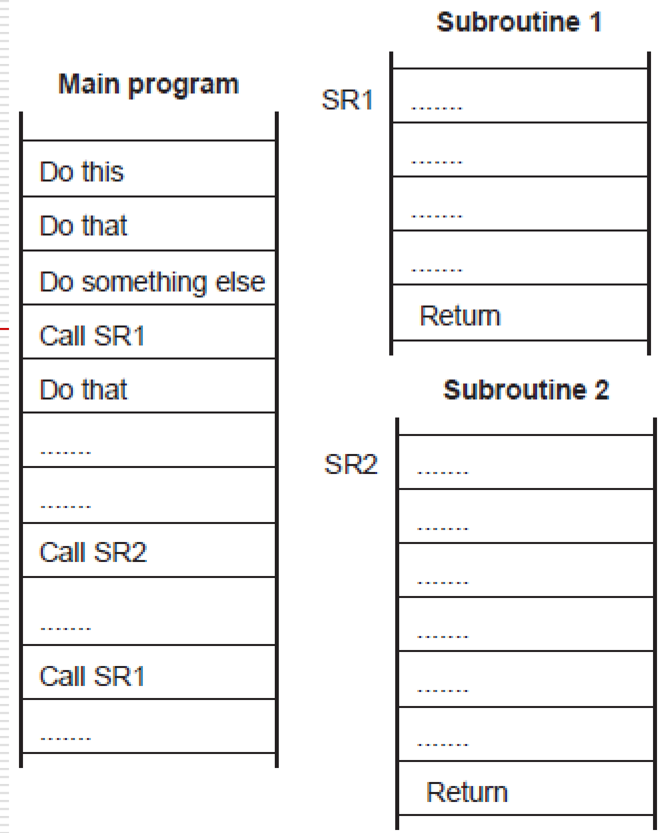
# Subroutines

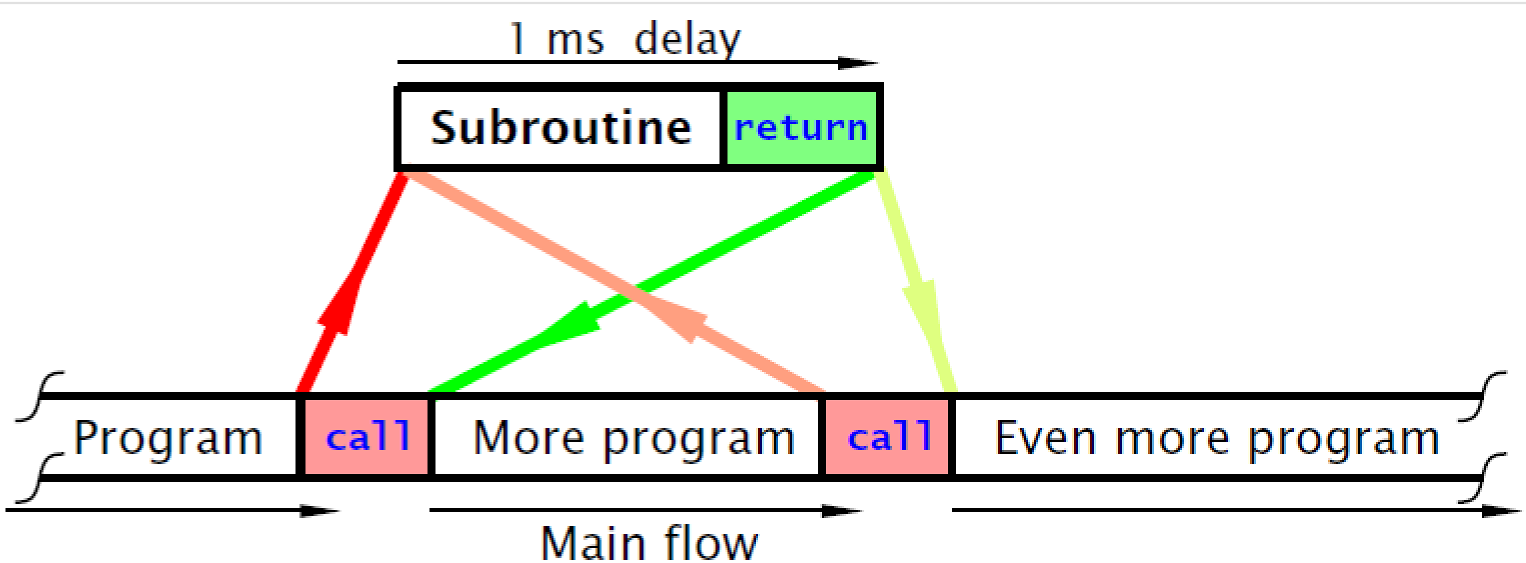
- As we develop bigger programs, we quickly find that there are program sections that are so useful that we would like to use them in different places.
- Yet it is tedious, and space- and memory consuming, to write out the program section whenever it is needed.
- Enter the subroutine.
- The subroutine is a program section structured in such a way that it can be called from anywhere in the program.
- Once it has been executed the program continues to execute from wherever it was before.



# Subroutines

- At some point in the main program there is an instruction 'Call SR1'.
- Program execution then switches to Subroutine 1, identified by its label.
- The subroutine must end with a 'Return from Subroutine' instruction.
- Program execution then continues from the instruction after the Call instruction.
- A little later in the program another subroutine is called, followed a little later by another call to the first routine.



# Instructions

- **CALL k** Send program flow directly to a program line or label. The position of the CALL instruction is pushed into the stack. A RETURN instruction will send the program flow back to the position where the CALL was made.

- **RETURN** This instruction will send the program flow to the last position pushed into the stack. Usually, this is done by a previous CALL instruction

| Mnemonic, operands | | Description | Cycles | 14 Bit opcode | | | |
|---|---|---|---|---|---|---|---|
| | | | | MSb | | | LSb |
| CALL | k | Call subroutine | 2 | 10 | 0kkk | kkk | kkk |
| RETURN | | Return from Subroutine | 2 | 00 | 0000 | 0000 | 1000 |

# Program Memory And The Stack

-Program Counter, the Stack and the actual program memory, work together

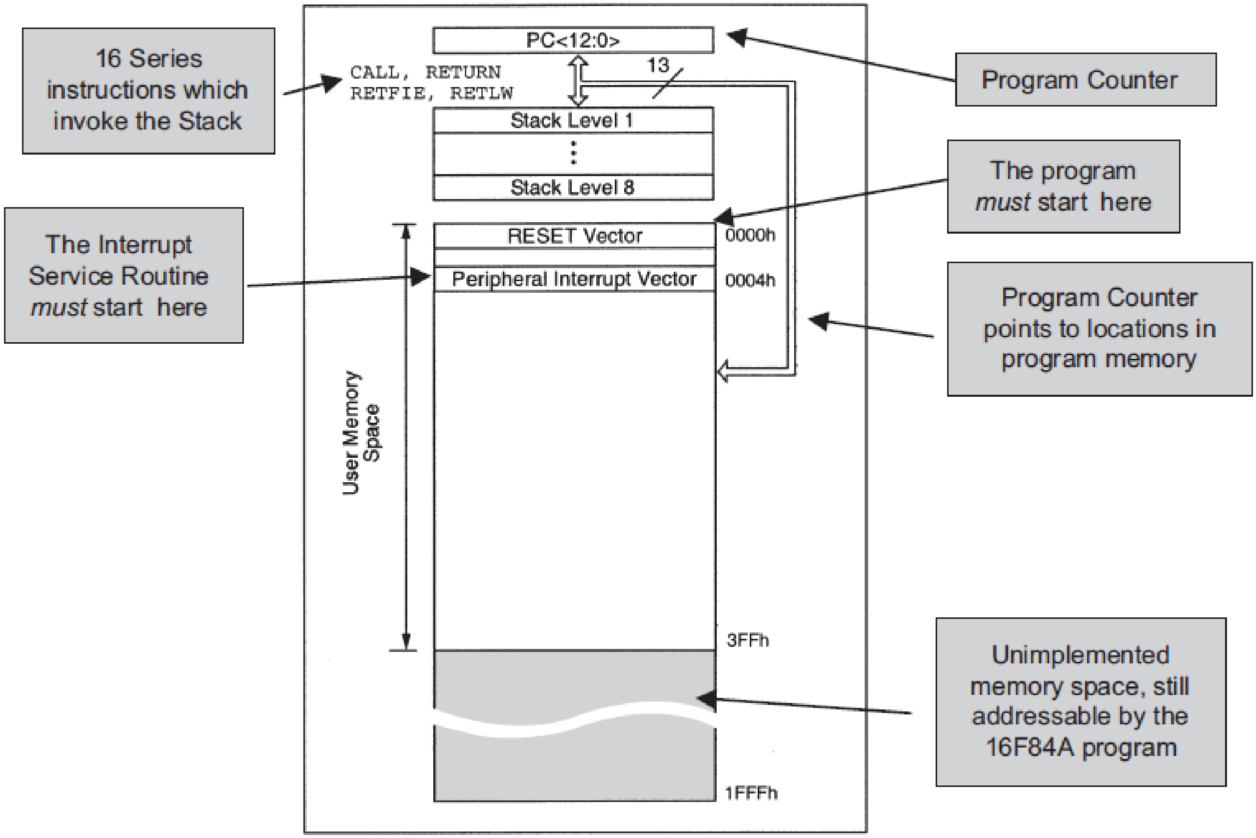
-The program memory is loaded with the program

code that the microcontroller executes.

-The program is in the form of a list of instructions.

-Program Counter acts as a pointer and holds the address of the next instruction that is to be executed by the microcontroller.

-Address range of the program memory is from 0000 to 03FFH. With its 13-bit Program Counter, the microcontroller can theoretically address a range from 0000 to 1FFFH.



# Program Memory And The Stack

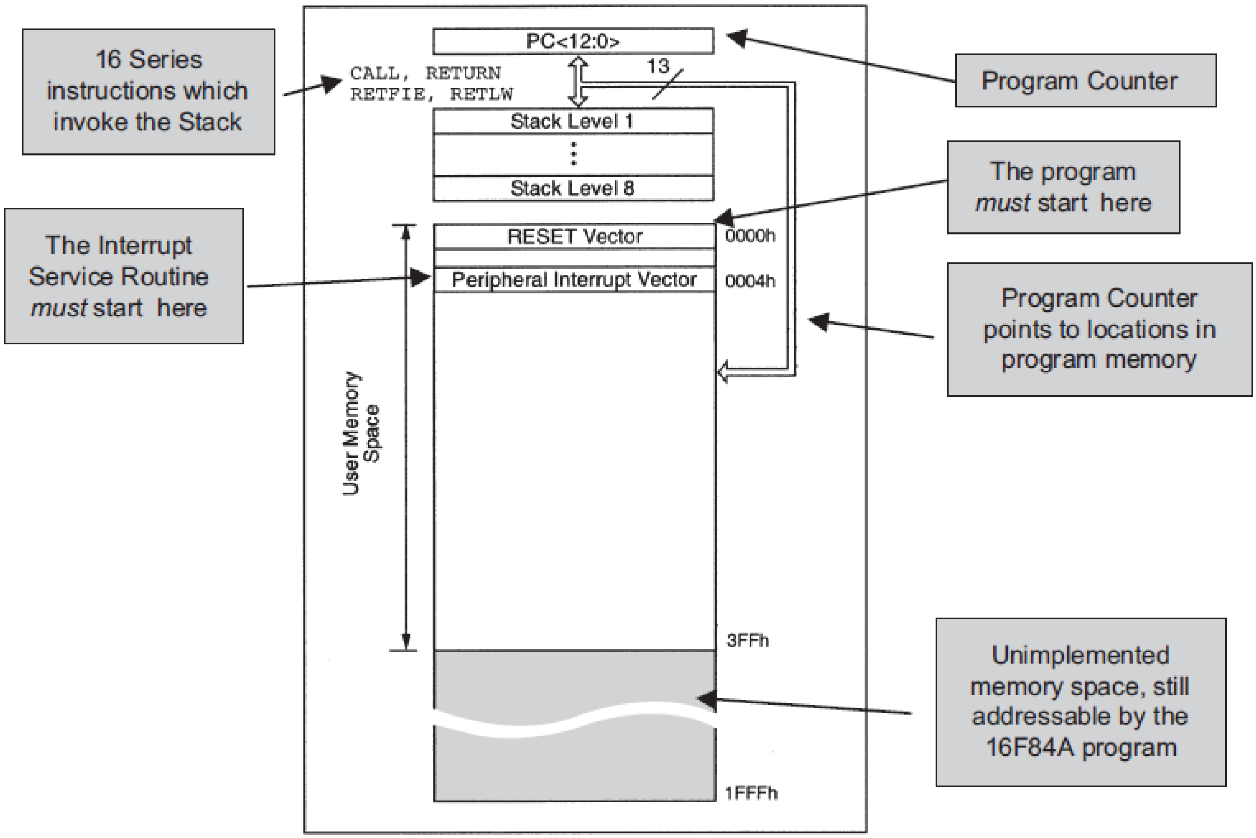
-Stack is a temporary memory that stores values of the program counter in case of special instructions (CALL, RETURN)

-Stack is structured as LIFO memory – last in, first out

-'reset vector' is first location in the program memory.

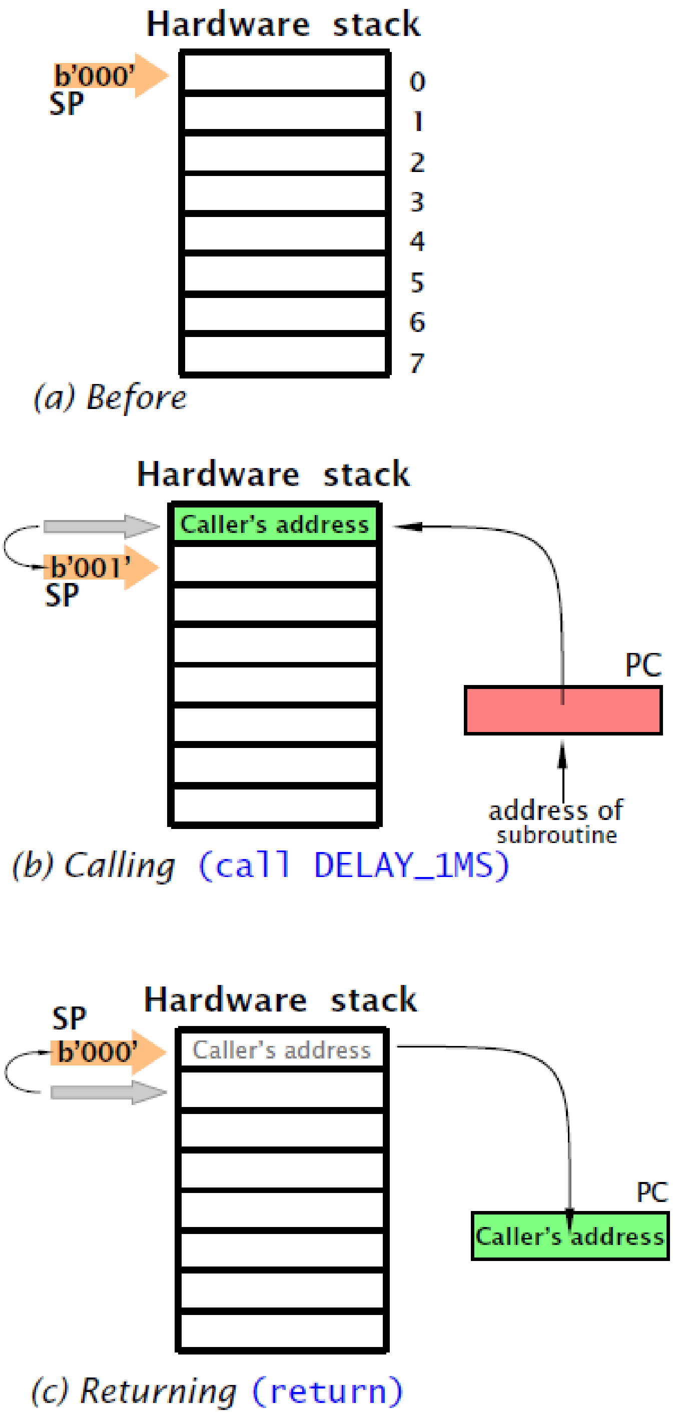
-When the program starts running for the first time, for example on power-up, the Program Counter is set to 0000.

-The programmer must therefore place his/her first instruction at this location.



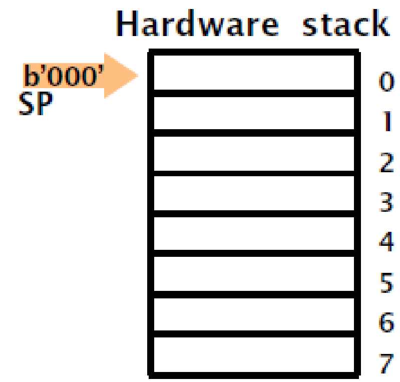
# CALL, RETURN: Procedure

- The action of the Call instruction is two-fold.
- It saves the contents of the Program Counter onto the Stack so that the CPU will know where to come back to after it has finished the subroutine.
- It then loads the subroutine start address into the Program Counter.
- Program execution thus continues at the subroutine.
- The return instruction complements the action of the Call.
- It loads the Program Counter with the data held at the top of the Stack, which will be the address of the instruction following the Call instruction.
- Program execution then continues at this address.
- Subroutine Call and Return instructions must always work in pairs.

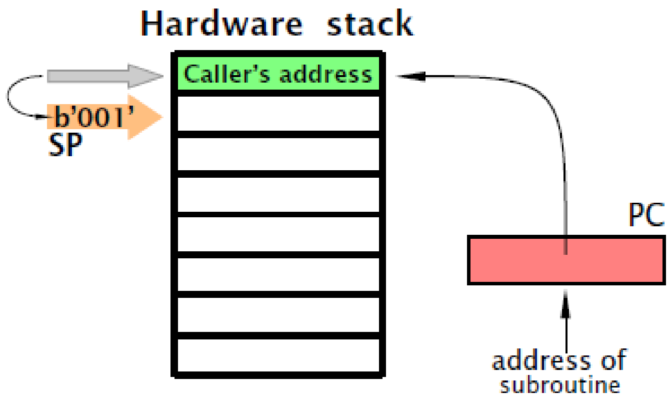


# CALL, RETURN: Procedure

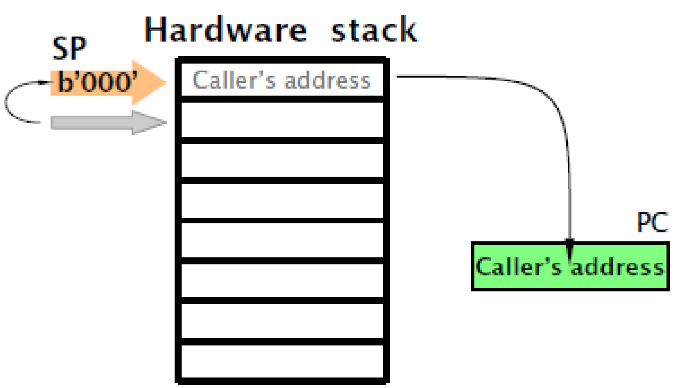
- In Figure the situation is shown after a call to a subroutine labelled DELAY_1MS. The execution sequence of this call DELAY_1MS is:
- 1.Copy the 13-bit contents of the PC into the stack register pointed to by the Stack Pointer. This will be the address of the instruction following the call instruction.
- 2. The Stack Pointer is incremented.
- 3. The destination address (which we assume is labelled DELAY_1MS), that is the location in the Program store of the entry point instruction of the subroutine, overwrites the original state of the Program counter. Effectively this causes the program execution to transfer to the subroutine.
- Apart from the pushing of the return address into the stack in steps 1 and 2, call acts exactly like a plain goto. Thus call requires two instruction cycles for execution, as the pipeline needs to be flushed to remove the next caller instruction which is already in situ.



(a) Before



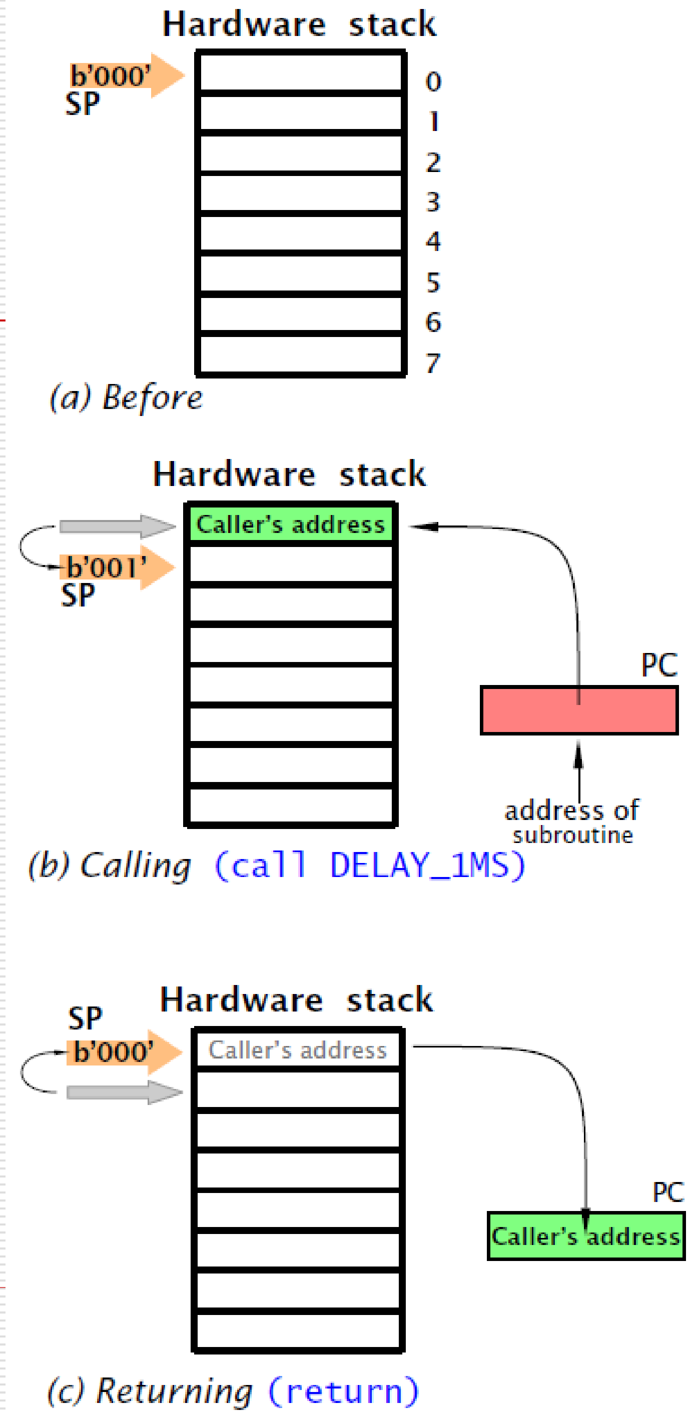
(b) Calling (call DELAY_1MS)



(c) Returning (return)

# CALL, RETURN: Procedure

- The exit point from the subroutine should be a return instruction. This reverses the push action of call and pulls the return address back out from the stack into the PC – as shown in Fig. This also requires a flush of the Pipeline, and takes two cycles. The execution sequence of return is:
- 1.Decrement the Stack Pointer.
- 2. Copy the address in the stack register pointed to by the Stack Pointer into the Program Counter.



# Fibonnacci Program Extended Version

-A counter has been included to show how many numbers in the series have been calculated.
-The program tests for range overflow by checking the Carry bit after each addition.
-When the 8-bit range is exceeded, it reverses the series by subtracting.
-You will notice that **c** and **z** are defined as labels in the opening equates section.
-The program starts as before by preloading the first three numbers in the series into the memory store.
-It starts moving up the series from the label **forward**.
-The two most recent numbers are added and the **Carry** bit then checked.
-If it is set, the 8-bit range has been exceeded and the program will need to reverse.
-Assuming **Carry** was not set, the program then increments the **counter** and shuffles the numbers in the memory store, discarding the oldest.
-The program then loops up to **forward**.
-If, however, the **Carry** had been set, the program branches to reverse. Now it works down the series by **subtraction**.
-It tests the **counter** number to determine when it should return to **forward**.

```
status equ 03
c        equ 0
z        equ 2
;these memory locations hold the three highest values of the Fibonacci series
fib0     equ  10      ;lowest number (oldest when going up,
                      ;newest when reversing down)
fib1     equ  11      ;middle number
fib2     equ  12      ;highest number
fibtemp equ  13       ;temporary location for newest number
counter equ  14       ;indicates value reached, opening value is 3

   org 00
;preload initial values
         movlw 0
         movwf fib0
         movlw 1
         movwf fib1
         movwf fib2
         movlw 3
         movwf counter ;we have preloaded the first three numbers,
                       ;so start count at 3
;
forward movf  fib1,0
         addwf fib2,0
         btfsc status,c      ;test if we have overflowed 8-bit range
         goto  reverse       ;here if we have overflowed, hence reverse down
         movwf fibtemp       ;latest number now placed in fibtemp
         incf  counter,1
;now shuffle numbers held, discarding the oldest
         movf  fib1,0        ;first move middle number, to overwrite oldest
         movwf fib0
         movf  fib2,0
         movwf fib1
         movf  fibtemp,0
         movwf fib2
         goto  forward
;when reversing down, subtract fib0 from fib1 to form new fib0
reverse movf  fib0,0
         subwf fib1,0
         movwf fibtemp       ;latest number now placed in fibtemp
         decf  counter,1
;now shuffle numbers held, discarding the oldest
         movf fib1,0         ;first move middle number, to overwrite oldest
         movwf fib2
         movf fib0,0
         movwf fib1
         movf fibtemp,0
         movwf fib0
;test if counter has reached 3, in which case return to forward
         movf  counter,0
         sublw 3
         btfsc status,z
         goto  forward
         goto  reverse
;
         end
```

**The rule: $x_n = x_{n-1} + x_{n-2}$**

**: Fib2 = Fib1+ Fib0**

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

**The reverse rule: $x_{n-3} = x_{n-1} - x_{n-2}$**

**: Fib0 = Fib2- Fib1**

..., 34, 21, 13, 8, 5, 3, 2, 1, 1, 0,

# Fibonnacci Program With Subroutines

- Fibonacci program is rewritten by replacing two code sections with subroutines.
- Each subroutine has been created simply by taking out a block of code from the main body of the program, labelling the first subroutine line, and terminating the block with a return instruction.
- The label effectively becomes the name of the subroutine.
- The subroutines have been grouped together and placed after the end of the main program.
- Each subroutine is called at the appropriate place in the program, using the call instruction and invoking the subroutine name.

```
status equ 03
c       equ 0
z       equ 2
;these memory locations hold the three highest values of the Fibonacci series
fib0    equ  10      ;lowest number (oldest when going up,
                     ;newest when reversing down)
fib1    equ  11      ;middle number
fib2    equ  12      ;highest number
fibtemp equ  13      ;temporary location for newest number
counter equ  14      ;indicates value reached, opening value is 3

  org 00
;preload initial values
       movlw 0
       movwf fib0
       movlw 1
       movwf fib1
       movwf fib2
       movlw 3
       movwf counter;we have preloaded the first three numbers,
                    ;so start count at 3
forward movf  fib1,0
             addwf fib2,0
             btfsc status,c      ;test if we have overflowed 8-bit range
             goto  reverse       ;here if we have overflowed,
                                 ;hence reverse down the series
             movwf fibtemp       ;latest number now placed in fibtemp
             incf  counter,1
;now shuffle numbers held, discarding the oldest
           call shuffle_up
             goto  forward
;when reversing down, we will subtract fib0 from fib1 to form new fib0
reverse movf  fib0,0
             subwf fib1,0
             movwf fibtemp       ;latest number now placed in fibtemp
             decf  counter,1
;now shuffle numbers held, discarding the oldest
             call shuffle_down
;test if counter has reached 3, in which case return to forward
             movf  counter,0
             sublw 3
             btfsc status,z
             goto  forward
             goto  reverse
;***********************************
;Subroutines
;***********************************
;Shuffles numbers in series, moving fib1 to fib0, fib2 to fib1, fibtemp to fib2
shuffle_up movf  fib1,0    ;first move middle number, to overwrite oldest
             movwf fib0
             movf  fib2,0
             movwf fib1
             movf  fibtemp,0
             movwf fib2
             return
;Shuffles numbers in series, moving fib1 to fib2, fib0 to fib1, fibtemp to fib0
shuffle_down movf fib1,0   ;first move middle number, to overwrite oldest
             movwf fib2
             movf fib0,0
             movwf fib1
             movf fibtemp,0
             movwf fib0
             return
             end
```

# Delay: General Formulation of a Single Delay Loop

**General Delay Loop** | **Notes**

```
counter   equ     counterAddress
nIt       equ     N
          movlw   nIt
          movwf   counter
loop      nop
            ⋮
          nop
          decfsz  counter,1
          goto    loop
          nop
          end
```

**Result**

- Assume loop contains $k$ nop instructions and oscillator frequency $f$
  $\Rightarrow (k+3)\cdot(N-1)+k+2+1=(k+3)\cdot N$ instruction cycles
  $\Rightarrow$ Delay of $(k+3)\cdot N\cdot 4/f$ between loop and end

# Subroutines: Examples

- Write a program that turns ON and OFF a LED connected to RB0 pin of PORTB with a 1 ms delay using a subroutine for the delay. The oscillator frequency is 4 MHz.

```
list p=16f84a
include "p16f84a.inc"
__config _CP_OFF&_WDT_OFF&_XT_OSC

org 0
main
Counter equ 0x0C ; free RAM location 12
N equ D'200' ; decimal constant 200
clrf PORTB;
bsf STATUS, RP0;
clrf TRISB;
bcf STATUS, RP0;
movlw N;
movwf Counter;
bsf PORTB,0;
LOOP
nop;
nop;
decfsz Counter, 1;
goto LOOP;
nop;
bcf PORTB,0;
end;
```

# Subroutines: Examples

*With Subroutine*

```
list p=16f84a
include "p16f84a.inc"
__config _CP_OFF&_WDT_OFF&_XT_OSC

org 0
main
Counter equ 0x0C ; free RAM location 12
N equ D'200' ; decimal constant 200
clrf PORTB;
bsf STATUS, RP0;
clrf TRISB;
bcf STATUS, RP0;
movlw N;
movwf Counter;
bsf PORTB,0;
call    delay;
bcf PORTB,0;

delay;  delay subroutine for N
LOOP
nop;
nop;
decfsz Counter, 1;
goto LOOP;
nop;
return; return to main program after N iterations

end;
```

*Without Subroutine*

```
list p=16f84a
include "p16f84a.inc"
__config _CP_OFF&_WDT_OFF&_XT_OSC

org 0
main
Counter equ 0x0C ; free RAM location 12
N equ D'200' ; decimal constant 200
clrf PORTB;
bsf STATUS, RP0;
clrf TRISB;
bcf STATUS, RP0;
movlw N;
movwf Counter;
bsf PORTB,0;
LOOP
nop;
nop;
decfsz Counter, 1;
goto LOOP;
nop;
bcf PORTB,0;
end;
```

# Cascaded Delay Loops

- Assume there are $k_1$ `nop` instructions in loop 1
- Assume there are $k_2$ `nop` instructions in loop 2
- Assume the oscillator frequency is $f$
- Number of instruction cycles of inner loop as before using $N_2$ and $k_2$
  $C_2 = (k_2 + 3) \cdot N_2$ instruction cycles in loop 2
- Number of instruction cycles of outer loop
  $C_1 = (k_1 + C_2 + 5) \cdot N_1$ instruction cycles in loop 1
- Overall delay:

$$C_1 \cdot \frac{4}{f} = (k_1 + C_2 + 5) \cdot N_1 \cdot \frac{4}{f} = (k_1 + (k_2 + 3) \cdot N_2 + 5) \cdot N_1 \cdot \frac{4}{f}$$

```
list       p=16f84a
include    "p16f84a.inc"

org        0
counter1   equ       counterAddress1
counter2   equ       counterAddress2
nIt1       equ       N1
nIt2       equ       N2
movlw      nIt1
movwf      counter1
loop1      nop
           :
           nop
           movlw     nIt2
           movwf     counter2
loop2      nop
           :
           nop
           decfsz    counter2,1
           goto      loop2
           nop
           decfsz    counter1,1
           goto      loop1
           nop
           end
```

# Subroutines: Examples Long Delay Subroutine

- Write a program that turns ON and OFF a LED connected to PORTB with a 0.6 s delay using a subroutine for the delay. The oscillator frequency is 4 MHz.

```
 list p=16f84a
    include "p16f84a.inc"
    __config _CP_OFF&_WDT_OFF&_XT_OSC

    org 0
    main
    counter1 equ 0x0C ; free RAM location 12
    counter2 equ 0x0D;
    N1              equ .250;
    N2              equ .239;
    clrf    PORTB;
    bsf             STATUS, RP0;
    clrf    TRISB;
    bcf             STATUS, RP0;
    movlw   N1;
    movwf   counter1;
    bsf             PORTB,0;
loop1
    nop;
    nop;
    nop;
    nop;
    nop;
    movlw   N2;
    movwf   counter2;
loop2
    nop;
    nop;
    nop;
    nop;
    nop;
    nop;
    nop;
    decfsz  counter2,1;
    goto    loop2;
    nop;
    decfsz  counter1,1;
    goto    loop1;
    bcf PORTB,0;
    clrw;
    end;
```

## With Subroutine

```
 list p=16f84a
   include "p16f84a.inc"
   __config _CP_OFF&_WDT_OFF&_XT_OSC

   org 0
   main
   counter1 equ 0x0C ; free RAM location 12
   counter2 equ 0x0D;
   N1          equ .250; decimal constant 10
   N2          equ .239;
   clrf   PORTB;
   bsf         STATUS, RP0;
   clrf   TRISB;
   bcf         STATUS, RP0;
   movlw   N1;
   movwf   counter1;
   bsf         PORTB,0;

   call delay;
   bcf PORTB,0;
   clrw;

delay;
loop1
   nop;
   nop;
   nop;
   nop;
   nop;
   movlw   N2;
   movwf   counter2;
loop2
   nop;
   nop;
   nop;
   nop;
   nop;
   nop;
   nop;
   decfsz  counter2,1;
   goto    loop2;
   nop;
   decfsz  counter1,1;
   goto    loop1;
   return;

   end;
```

## Without Subroutine

```
 list p=16f84a
   include "p16f84a.inc"
   __config _CP_OFF&_WDT_OFF&_XT_OSC

   org 0
   main
   counter1 equ 0x0C ; free RAM location 12
   counter2 equ 0x0D;
   N1          equ .250; decimal constant 10
   N2          equ .239;
   clrf   PORTB;
   bsf         STATUS, RP0;
   clrf   TRISB;
   bcf         STATUS, RP0;
   movlw   N1;
   movwf   counter1;
   bsf         PORTB,0;
loop1
   nop;
   nop;
   nop;
   nop;
   nop;
   movlw   N2;
   movwf   counter2;
loop2
   nop;
   nop;
   nop;
   nop;
   nop;
   nop;
   nop;
   decfsz  counter2,1;
   goto    loop2;
   nop;
   decfsz  counter1,1;
   goto    loop1;
   bcf PORTB,0;
   clrw;
   end;
```

# Subroutines: Examples General Delay Subroutine

- Write a delay subroutine delay Nms with a delay N · 100 ms. The oscillator frequency is 4 MHz. The value of N is passed in the working register W. Use the subroutine in a blinking LED application

```
N   equu        .10;
movlw   N;move N to working register for delay subroutine


delay_Nms;  delay subroutine for N*100ms delay
   movwf   0x0E;
loopN;
   call    delay_100ms;    call 100ms delay subroutine
   decfsz  0x0E,1;
   goto    loopN;
   return; return to main program after N iterations

delay_100ms; delay subroutine for 100ms delay
   movlw   .250;
   movwf   0x0C; counter for outer loop
loop1;      outer loop with N1=250 iterations
   nop;    k1 = 3
   nop;
   nop;
   movlw   .98;
   movwf   0x0D; counter for inner loop;
loop2; inner loop with N2 = 98 iterations
   nop;    k2 = 1
   decfsz  0x0D,1; decrement counter2 (inner loop)
   goto    loop2;
   nop;
   decfsz  0x0C,1; decrement couter1 (outer loop)
   goto    loop1;
   return; return to delay_Nms subroutine
```

# Subroutines: Examples Moving LEDs

- Write a program such that LEDs connected to the pins of PORTB are turned on one after another with a delay of 1 s. Start from RB0.

```
 list p=16f84a;
   include "p16f84a.inc"

N   equu            .10;

   org 0;
main;
   bsf              STATUS,5;
   clrf    TRISB;  all PORTB pins are output
   bcf              STATUS,5;
   movlw   b'00000001';
   movwf   PORTB;  turn on led at RB0
   bcf              STATUS,0;

loop;
   movlw   N;    move N to working register for delay subroutine
   call    delay_Nms;
   btfsc   PORTB,7;
   rlf              PORTB,1; rotate left PORTB twice if RB7 is 1 (otherwise LEDs will be off)
   rlf              PORTB,1; rotate left PORTB over carry one time after each delay
   goto    loop;

delay_Nms;  delay subroutine for N*100ms delay
   movwf   0x0E;
loopN;
   call    delay_100ms;    call 100ms delay subroutine
   decfsz  0x0E,1;
   goto    loopN;
   return; return to main program after N iterations

delay_100ms; delay subroutine for 100ms delay
   movlw   .250;
   movwf   0x0C; counter for outer loop
loop1;           outer loop with N1=250 iterations
   nop;    k1 = 3
   nop;
   nop;
   movlw   .98;
   movwf   0x0D; counter for inner loop;
loop2; inner loop with N2 = 98 iterations
   nop;    k2 = 1
   decfsz  0x0D,1; decrement counter2 (inner loop)
   goto    loop2;
   nop;
   decfsz  0x0C,1; decrement couter1 (outer loop)
   goto    loop1;
   return; return to delay_Nms subroutine

   end;
```