EC-310 Microprocessor and Microcontroller Based Design

Chapter - 3

Call Instruction and Stack

Nazar Abbas Saqib
nazar.abbas@ceme.nust.edu.pk

Outline

- Call Instructions and Stack

CALL Instruction and Stack

- **Subroutines**
- Subroutines are used to perform the tasks which are required frequently.
- For example a task is to be performed time and again in the program, than instead of writing that code again and again, we can write a subroutine for that code and whenever required, the subroutine can be called.
- Advantages of subroutines:
  - Readability improves.
  - Less chances of errors.
  - Debugging gets easy.

CALL Instruction and Stack

- **How to write subroutines**
- ORG directive is used to place the subroutine in the memory
- Subroutine is labelled
- RETURN command must be the last command of a subroutine

```
;———— this is the delay subroutine

        ORG     300H    ;put delay at address 300H
DELAY   MOVLW   0xFF    ;WREG = 255,the counter
        MOVWF   MYREG
AGAIN   NOP             ;no op wastes clock cycles
        NOP
        DECF    MYREG, F
        BNZ     AGAIN   ;repeat until MYREG becomes 0
        RETURN          ;return to caller
```

CALL Instruction and Stack

- **Calling Subroutines**
- In PIC there are two instructions associated with subroutines:
  - CALL (long call)
  - RCALL (relative call)

- **CALL:**
- CALL instruction is used to invoke a subroutine when required.
- It is a 4-byte instruction. First 12 bits are used for opcode and next 20 bits are used for the target address.
- For call instruction, the program counter LSB is set to 0 to cater for even addresses. In this way 2M memory space is addressable.

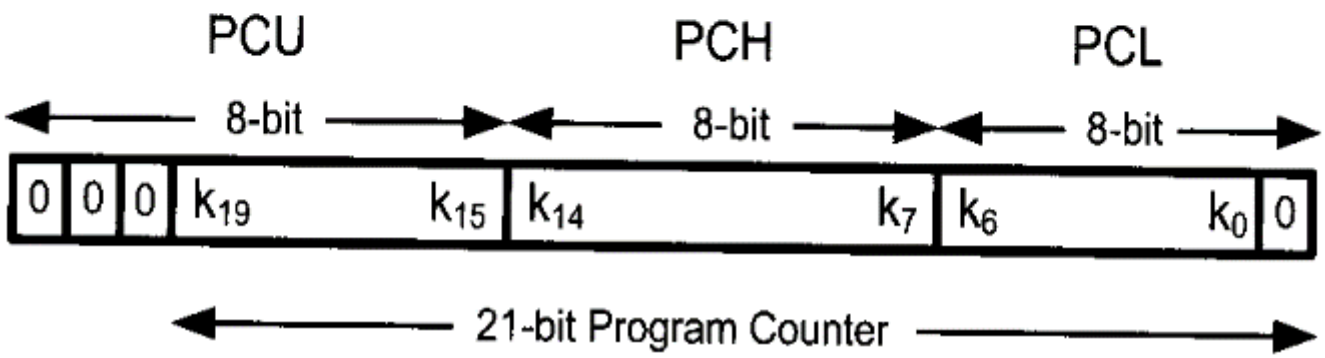
CALL Instruction and Stack

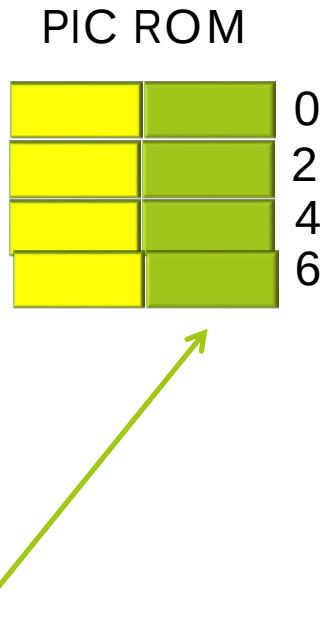
- **CALL:**

- **Call instruction can be used to call a subroutine located anywhere within the address range of 00000H and 1FFFFH.**

1110	1110	k_7kkk	$kkkk_0$
1111	$k_{19}kkk$	kkkk	$kkkk_8$

$0 \leq k \leq FFFFF$





0 at LSB ensures jumping at even addresses as PIC ROM is organized in 2 bytes

Example 3-9

Toggle all the bits of the SFR register of Port B by sending to it the values 55H and AAH continuously. Put a time delay in between each issuing of data to Port B.

Solution:

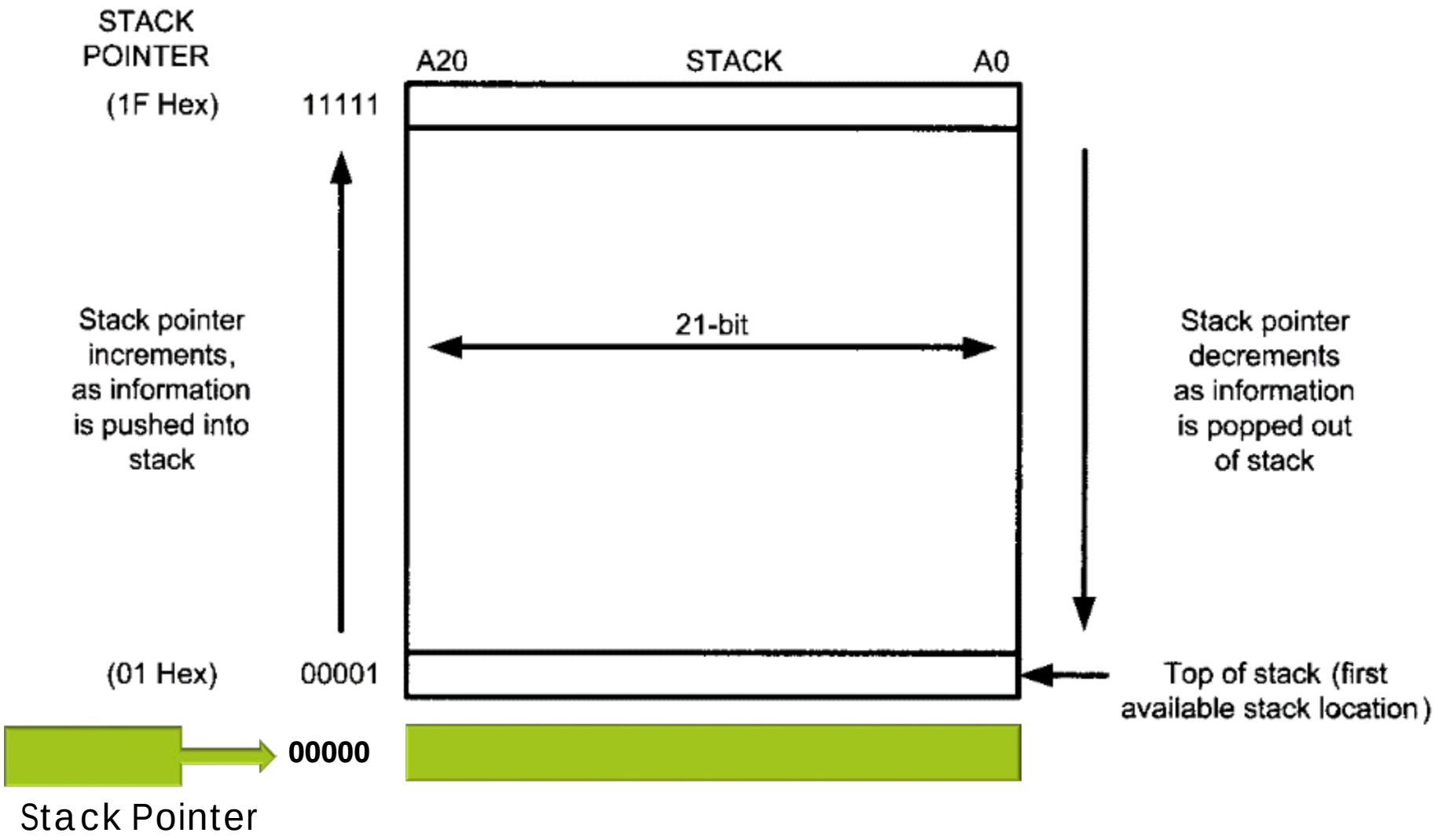
```
MYREG EQU   0x08                ;use location 08 as counter
          ORG       0
BACK      MOVLW     0x55        ;load WREG with 55H
          MOVWF     PORTB       ;send 55H to port B
          CALL      DELAY       ;time delay
          MOVLW     0xAA        ;load WREG with AA (in hex)
          MOVWF     PORTB       ;send AAH to port B
          CALL      DELAY
          GOTO      BACK        ;keep doing this indefinitely
;-------  this is the delay subroutine
          ORG       300H        ;put time delay at address 300H
DELAY     MOVLW     0xFF        ;WREG = 255,the counter
          MOVWF     MYREG
AGAIN     NOP                   ;no operation wastes clock cycles
          NOP
          DECF      MYREG, F
          BNZ       AGAIN       ;repeat until MYREG becomes 0
          RETURN                ;return to caller
          END                   ;end of asm file
```

CALL Instruction and Stack

- **Stack and stack pointer in PIC18**
- Stack is a read/write RAM memory to store very critical information temporarily.
- This storage is required by the CPU due to limited number of registers in CPU.
- The stack in PIC18 is 21-bit. This is because the PC is 21-bit.
- **Stack is used for CALL instruction to make sure that the PIC knows where to come back after execution of called subroutine.**
- Stack Pointer: The register used to access stack is known as stack pointer. Stack pointer (SP) is a 5-bit register that can take values from 00H to 1FH. This gives us a total of 32 locations.

CALL Instruction and Stack

- **Stack and stack pointer in PIC18**



CALL Instruction and Stack

- **Stack and stack pointer in PIC18**
- When PIC is powered up, the stack pointer has value 0. This means that it is pointing to location 1 of the stack. In this way PIC18 stack has 31 locations.
- Stack is a LIFO (last in first out memory)

- **How stacks are accessed in PIC18**
- Storing of information on stack is known as a PUSH.
- Loading contents of stack back into CPU register is known as POP.

CALL Instruction and Stack

- **Pushing onto the stack**
- The stack pointer (SP) is always pointing to the last used location of the stack. The last used location of the stack is known as top of the stack (TOS).
- When data is pushed into the stack, the SP is incremented. This is in contrast to x86 uP where SP is decremented.
- **Popping from the stack**
- Popping is opposite process of pushing i.e. placing the address back into the PC. When RETURN instruction is executed two processes occur:
  - Content of TOS is moved into PC.
  - SP decremented by one.

CALL Instruction and Stack

- **CALL:**
- When call instruction is executed, the control of the program shifts i.e. the program counter value is changed and the actual sequence of execution gets disturbed.
- Now the question arises that, where to return and start execution after the subroutine execution is completed.
- The solution to this problem is provided by the microcontroller itself. When CALL instruction is called, the address of instruction immediately next to CALL instruction is stored in a stack.
- After subroutine is completed, the address is popped out of the stack and stored into the PC.
- This is done by the use of RETURN instruction. Hence every subroutine must terminate at RETURN instruction.

CALL Instruction and Stack

- **CALL Instruction and the role of Stack**

- Upon executing the first "CALL Delay", the address of the instruction right below it "MOVLW 0xAA" is stored into the stack.

```
CALL    DELAY
MOVLW   0xAA
```

The address of the instruction next to call instruction is loaded into the stack.

- The control of execution is shifted to 0x300 memory location where the subroutine exists in memory.

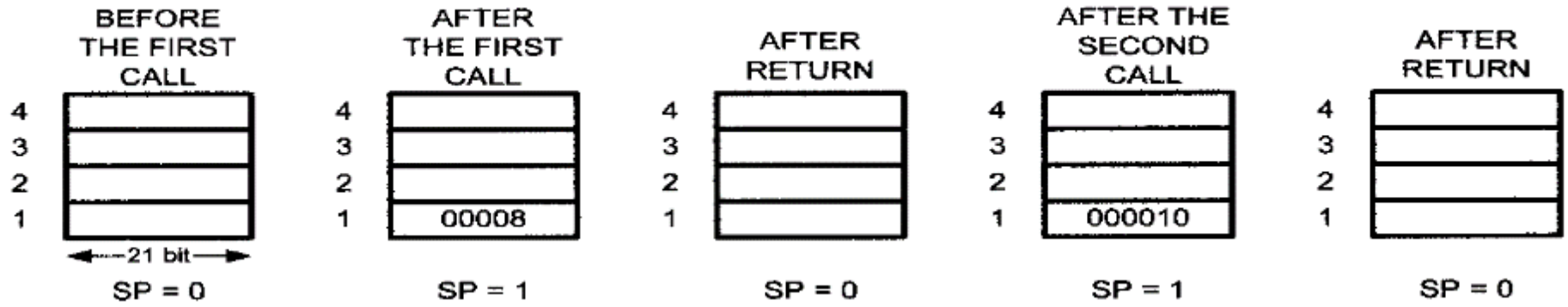
```
ORG     300H
```

- On the RETURN instruction, the address at the top of the stack is loaded into the program counter.

Example 3-11

```
LOC   OBJECT CODE      LINE SOURCE TEXT
  VALUE

                       00001 #DEFINE PORTB 0xF81
  00000008             00002 MYREG EQU 0x08        ;use location 08 as counter
                       00003
                       00004
  000000               00005         ORG    0
000000 0E55            00006 BACK    MOVLW  0x55   ;load WREG with 55H
000002 6E81            00007         MOVWF  PORTB  ;send 55H to port B
000004 EC80 F001       00008         CALL   DELAY  ;time delay
000008 0EAA            00009         MOVLW  0xAA   ;load WREG with AA (in hex)
00000A 6E81            00010         MOVWF  PORTB  ;send AAH to port B
00000C EC80 F001       00011         CALL   DELAY
000010 EF00 F000       00012         GOTO   BACK   ;keep doing this indefinitely
                       00013
                       00014 ;------- this is the delay subroutine
                       00015
  000300               00016         ORG    300H   ;put delay at address 300H
000300 0EFF            00017 DELAY   MOVLW  0xFF   ;WREG = 255,the counter
000302 6E08            00018         MOVWF  MYREG
000304 0000            00019 AGAIN   NOP            ;no op wastes clock cycles
000306 0000            00020         NOP
000308 0608            00021         DECF   MYREG, F
00030A E1FC            00022         BNZ    AGAIN  ;repeat until MYREG becomes 0
00030C 0012            00023         RETURN       ;return to caller
                       00024         END          ;end of asm file
```



Example 3-11

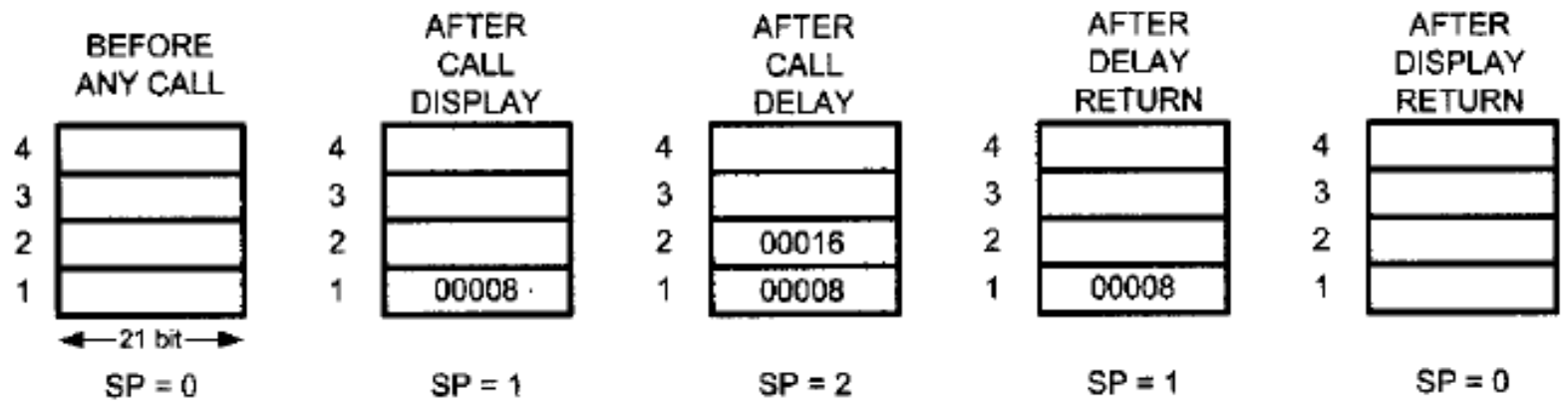
Write a program to count up from 00 to FFH and send the count to SFR of Port B. Use one CALL subroutine for sending the data to Port B and another one for time delay. Put a time delay in between each issuing of data to Port B.

Solution:

```
LOC    OBJECT CODE   LINE   SOURCE TEXT
  VALUE

                     00001         list P=PIC18F458
                     00002  #include P18F458.INC
                     00003
  00000007           00004  COUNT  EQU    0x07   ;use location 07 for count-up
  00000008           00005  MYREG  EQU    0x08   ;use location 08 for delay
                     00006
000000               00007  ORG    0
000000 0E00          00008         MOVLW  0                ;WREG = 0
000002 6E07          00009         MOVWF  COUNT            ;count = 0
000004 EC06 F000     00010  BACK   CALL   DISPLAY
000008 EF02 F000     00011         GOTO   BACK
                     00012
                     00013 ;------- increment and put it in PORTB
00000C 2A07          00014  DISPLAY INCF  COUNT,F         ;increment count
00000E C007 FF81     00015         MOVFF  COUNT,PORTB     ;send it to PORTB
000012 EC80 F001     00016         CALL   DELAY
000016 0012          00017         RETURN                 ;return to caller
```

```
                    00018
                    00019 ;------- this is the delay subroutine
000300              00020 ORG    300H   ;put time delay at address 300H
000300 0EFF         00021 DELAY  MOVLW  0xFF          ;WREG = 255, the counter
000302 6E08         00022        MOVWF  MYREG
000304 0000         00023 AGAIN  NOP    ;no operation wastes clock cycles
000306 0000         00024        NOP
000308 0000         00025        NOP
00030A 0608         00026        DECF   MYREG,F
00030C E1FB         00027        BNZ    AGAIN  ;repeat until MYREG becomes 0
00030E 0012         00028        RETURN             ;return to caller
                    00029        END                ;end of asm file
```



CALL Instruction and Stack

- **Calling many subroutines from the main program**
- Incase of multiple subroutines in a program, each subroutine is written as a separate module
- Each subroutine has its own starting address i.e. the ORG directive.
- Each subroutine has its own RETURN instruction.

CALL Instruction and Stack

```
        ORG  0
        CALL SUBR_1        ;CALL SUBROUTINE 1
        CALL SUBR_2        ;CALL SUBROUTINE 2
        CALL SUBR_3        ;CALL SUBROUTINE 3
HERE    BRA  HERE          ;STAY HERE

                           ;END OF MAIN

SUBR_1  ........
        ........
        RETURN             ;END OF SUBROUTINE 1

SUBR_2  ........
        ........
        RETURN             ;END OF SUBROUTINE 2

SUBR_3  ........
        ........
        RETURN             ;END OF SUBROUTINE 3

        END                ;END ASM FILE
```

;MAIN PROGRAM CALLING SUBROUTINES

CALL Instruction and Stack

- **RCALL (relative call)**
- RCALL is a 2-byte instruction.
- 5 bits are used for opcode and 11 bits are used for target address.
- Hence the target address must be within 2K.
- RCALL implementation mechanism is same as that of CALL. The only difference is in the size of the instruction and correspondingly the ability to access memory locations.

CALL Instruction and Stack

Example 3 - 12

Rewrite the main part of Example 3-9 as efficiently as you can.

Solution:

```
        MYREG EQU 0x08
        ORG   0
        MOVLW 0x55          ;load WREG with 55H
BACK    MOVWF PORTB                 ;issue value in PORTB SFR
        RCALL DELAY         ;time delay
        COMPF PORTB,F               ;complement Port B SFR
        BRA   BACK          ;keep doing this indefinitely
;————————this is the delay subroutine
DELAY   MOVLW 0xFF          ;WREG = 255, the counter
        MOVWF MYREG
AGAIN   NOP                 ;no operation wastes clock cycles
        NOP
        DECF  MYREG,F
        BNZ   AGAIN         ;repeat until MYREG becomes 0
        RETURN              ;return to caller (MYREG = 0)
        END                 ;end of asm file
```

CALL Instruction and Stack

- **Example 3 - 13**

A developer is using the PIC18 microcontroller chip for a product. This chip has only 4K of on-chip flash ROM. Which of the instructions, CALL or RCALL, is more useful in programming this chip?

Solution:

The RCALL instruction is more useful because it is a 2-byte instruction. It saves two bytes each time the call instruction is used. However, we must use CALL if the target address is beyond the 2K boundary.