Instructor's Manual with PowerPoints
to accompany

PIC Microcontroller and Embedded Systems
Using Assembly and C for PIC18

Muhammad Ali Mazidi
Rolin D. McKinlay
Danny Causey

With contributions from
Muhammad Ali Mazidi
Javad Rasti
Rolin McKinlay
Danny Causey
Faramarz Mortezae

PEARSON
Prentice
Hall

Upper Saddle River, New Jersey
Columbus, Ohio

10 9 8 7 6 5 4 3 2 1

PEARSON
Prentice
Hall

ISBN-13: 978-0-13-119406-9
ISBN-10: 0-13-119406-2

CHAPTER 0: INTRODUCTION TO COMPUTING

SECTION 0.1: NUMBERING AND CODING SYSTEMS

1.

(a) $12_{10} = 1100_2$
(b) $123_{10} = 0111\ 1011_2$
(c) $63_{10} = 0011\ 1111_2$
(d) $128_{10} = 1000\ 0000_2$
(e) $1000_{10} = 0011\ 1110\ 1000_2$

2.

(a) $100100_2 = 36_{10}$
(b) $1000001_2 = 65_{10}$
(c) $11101_2 = 29_{10}$
(d) $1010_2 = 10_{10}$
(e) $00100010_2 = 34_{10}$

3.

(a) $100100_2 = 24_{16}$
(b) $1000001_2 = 41_{16}$
(c) $11101_2 = 1D_{16}$
(d) $1010_2 = 0A_{16}$
(e) $00100010_2 = 22_{16}$

4.

(a) $2B9_{16} = 0010\ 1011\ 1001_2$, 697_{10}
(b) $F44_{16} = 1111\ 0100\ 0100_2$, 3908_{10}
(c) $912_{16} = 1001\ 0001\ 0010_2$, 2322_{10}
(d) $2B_{16} = 0010\ 1011_2$, 43_{10}
(e) $FFFF_{16} = 1111\ 1111\ 1111\ 1111_2$, 65535_{10}

5.

(a) $12_{10} = 0C_{16}$
(b) $123_{10} = 7B_{16}$
(c) $63_{10} = 3F_{16}$
(d) $128_{10} = 80_{16}$
(e) $1000_{10} = 3E8_{16}$

6.

(a) 1001010 = 0011 0110
(b) 111001 = 0000 0111
(c) 10000010 = 0111 1110
(d) 111110001 = 0000 1111

7.

(a) 2C+3F = 6B
(b) F34+5D6 = 150A
(c) 20000+12FF = 212FF
(d) FFFF+2222 = 12221

8. (a) 24F-129 = 126_{16}
(b) FE9-5CC = $A1D_{16}$
(c) 2FFFF-FFFFF = 30000_{16}
(d) 9FF25-4DD99 = $5218C_{16}$

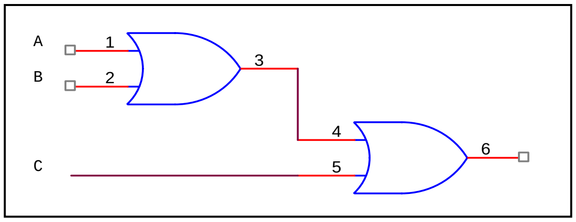
9. (a) Hex: 30, 31, 32 , 33, 34, 35, 36, 37, 38, 39
(b) Binary: 11 0000, 11 0001, 11 0010, 11 0011, 11 0100, 11 0101, 11 0110, 11 0111, 11 1000, 11 1001.

	ASCII (hex)	Binary	
0	30	011	0000
1	31	011	0001
2	32	011	0010
3	33	011	0011
4	34	011	0100
5	35	011	0101
6	36	011	0110
7	37	011	0111
8	38	011	1000
9	39	011	1001

10.
```
000000 22 55 2E 53 2E 41 2E 20 69 73 20 61 20 63 6F 75     "U.S.A. is a cou
000010 6E 74 72 79 22 0D 0A 22 69 6E 20 4E 6F 72 74 68     ntry".."in North
000020 20 41 6D 65 72 69 63 61 22 0D 0A                    America"..
```

SECTION 0.2: DIGITAL PRIMER

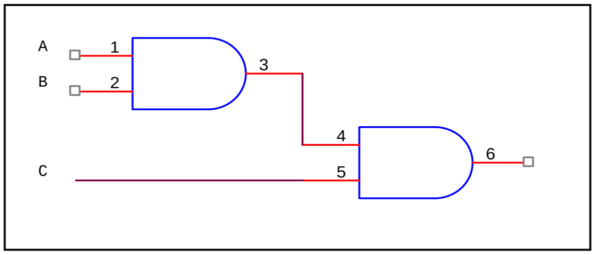
11.



12.

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

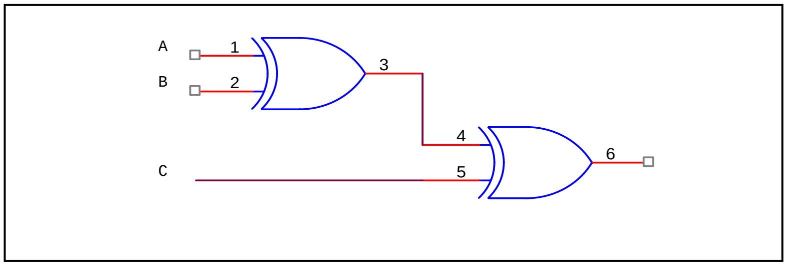
13.



14.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

15.



A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

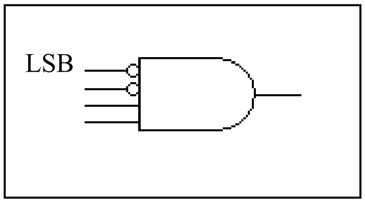
16.

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

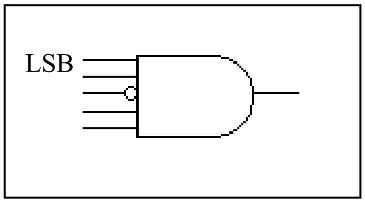
17.

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

18.



19.



20.

CLK	D	Q
No	X	NC
Yes	0	0
Yes	1	1

SECTION 0.3: INSIDE THE COMPUTER

21. (a) 4
(b) 4
(c) 4
(d) 1 048 576, 2^{20}
(e) 1024K
(f) 1 073 741 824, 2^{30}
(g) 1 048 576 K
(h) 1024M
(i) 8388608, 8192K

22. Disk storage capacity / size of a page = $(2*2^{30}) / (25*80)$ = 1 million pages

23. (a) 9FFFFh – 10000h = 8FFFFh = 589 824 bytes
(b) 576 kbytes
24. $2^{32} - 1$ = 4 294 967 295
25. (a) FFh, 255
(b) FFFFh, 65535
(c) FFFF FFFFh, 4 294 967 295
(d) FFFF FFFF FFFF FFFFh, 18 446 744 073 709 551 615
26. (a) 2^{16} = 64K
(b) 2^{24} = 16M
(c) 2^{32} = 4096 Mega, 4G
(d) 2^{48} = 256 Tera, 262144 Giga, 268435456 Mega
27. Data bus is bidirectional, address bus is unidirectional (exit CPU).
28. PC (Program Counter)
29. ALU is responsible for all arithmetic and logic calculations in the CPU.
30. Address, control and data

CHAPTER 1: THE PIC MICROCONTROLLERS: HISTORY AND FEATURES

SECTION 1.1: MICROCONTROLLERS AND EMBEDDED PROCESSORS

1. False.
2. True.
3. True.
4. True.
5. CPU, RAM, ROM, EEPROM, I/O, Timer, Serial COM port, ADC.
6. RAM and ROM.
7. Keyboard, mouse, printer.
8. Computing power and compatibility with millions and millions of PCs.
9. PIC 16x – Microchip Technology, 8051 - Intel, AVR – Atmel, Z8 – Zilog, 68HC11 – Freescale Semiconductor (Motorola).
10. 8051.
11. Power consumption.
12. The ROM area is where the executable code is stored.
13. Very, in case there is a shortage by one supplier.
14. Suppliers other than the manufacturer of the chip.
15. B is absolutely wrong, 16 bit software can not run on a 8 bit system due to special instructions and registers. But A can be true (in the case of software compatibility).

SECTION 1.2: OVERVIEW OF THE PIC18 FAMILY

16. 32 Kbytes
17. 1536 bytes
18. 4 timers
19. 1536 bytes
20. PIC18C601, PIC18C801.
21. 36 pins
22. 1 serial port
23. Flash (see the letter 'F')
24. OTP (see the letter 'C')
25. Flash (see the letter 'F')
26. Flash (see the letter 'F')
27. (a) 16K ROM, 768 Bytes RAM
    (b) 32K ROM, 1536 Bytes RAM
    (c) 128K ROM, 3936 Bytes RAM
28. The OTP version of the PIC
29. The PIC18F2420 has 16Kbytes of Flash, no EEPROM, and 768 bytes of data RAM. The PIC18F2220 has 4Kbytes of Flash, 256 bytes of EEPROM, and 512 bytes of data RAM.
30. 256 bytes.

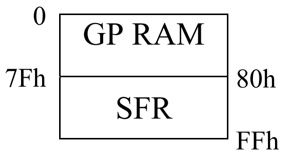
CHAPTER 2: PIC ARCHITECTURE & ASSEMBLY LANGUAGE PROGRAMMING

SECTION 2.1: THE WREG REGISTER IN PIC

1. 8
2. 8
3. 8
4. 0FFH
5. not necessary
6.
```
MOVLW    15H    ; WREG  = 15H
ADDLW    13H    ; WREG  = 15H + 13H = 28H
```
7. In (a) and (d) the operand exceeds allowed range. The operand in (f) should start with 0. The syntax of (g) is wrong.
8. (a), (c) and (d)
9.
```
MOVLW    25H    ; WREG  = 25H
ADDLW    1FH    ; WREG  = 25H + 1FH = 44H
```
10.
```
MOVLW    15H    ; WREG  = 15H
ADDLW    0EAH   ; WREG  = 15H + EAH = FFH
```
11. 255 or 0FFH
12. False. There is only one WREG in PIC.

SECTION 2.2: THE PIC FILE REGISTER

13. SRAM
14. True
15. True
16. False
17. False
18. Data RAM
19. (a) 32 bytes
(b) 80 bytes
(c) 4096 bytes
20. Each PIC has data RAM certainly, but one may not have EEPROM.
Data RAM is used to store temporary data and when power goes off, its information is lossed. But, we use EEPROM to store nonvolatile data that must remain intact even when the power is turned off.
21. Yes. like PIC18F251 or PIC18F245
22. No
23. 256
24.



25. Maximum size of the file register / Maximum size of a bank = 4096/256 = 16
26. 4096 bytes

SECTION 2.3: USING INSTRUCTIONS WITH THE DEFAULT ACCESS BANK

27. 0 – 7Fh (0 – 127)

28.

```
MOVLW 30H
MOVWF 5H
MOVLW 97H
MOVWF 6H
```

29.

```
MOVLW 55H
MOVWF 0H
MOVWF 1H
MOVWF 2H
MOVWF 3H
MOVWF 4H
MOVWF 5H
MOVWF 6H
MOVWF 7H
MOVWF 8H
```

30.

```
MOVLW 5FH
MOVWF PORTB
```

31. True.

32. True.

33. 0 or W

34. 1 or F

35. (a)

```
MOVLW 11H
MOVWF 0H
MOVWF 1H
MOVWF 2H
MOVWF 3H
MOVWF 4H
MOVWF 5H
```

(b)

```
MOVLW 0x00
ADDWF 0H, W
ADDWF 1H, W
ADDWF 2H, W
ADDWF 3H, W
ADDWF 4H, W
ADDWF 5H, W
```

36.

```
MOVLW 11H
MOVWF 0H
MOVWF 1H
MOVWF 2H
MOVWF 3H
MOVWF 4H
MOVWF 5H
MOVLW 0x00
ADDWF 0H, W
ADDWF 1H, W
ADDWF 2H, W
ADDWF 3H, W
ADDWF 4H, W
ADDWF 5H, F
```

37. (a)

```
  MOVLW 15H
  MOVWF 7H
```

(b)

```
  MOVLW 0H
  ADDWF 7H, W
  ADDWF 7H, W
  ADDWF 7H, W
  ADDWF 7H, W
  ADDWF 7H, W
```

38. (a)

```
  MOVLW 15H
  MOVWF 7H
```

(b)

```
  MOVLW 0H
  ADDWF 7H, F
  ADDWF 7H, F
  ADDWF 7H, F
  ADDWF 7H, F
  ADDWF 7H, F
```

39. MOVWF copies the contents of WREG to a location in file register. MOVF copies the contents of a location in file register to the itself (MOVF myfile, F) or WREG (MOVF myfile, W).

40. `COMF 8H, W`

41. True.

42. (a)

```
  MOVF  8H, W
  MOVWF PORTC
```

(b)

```
  MOVFF 8H, PORTC
```

SECTION 2.4: PIC STATUS REGISTER

43. 8

44. D0 – D1

45. D3 – D4

46. When there is a carry out from D7 bit.

47. When there is a carry from D3 to D4.

48. FFh + 1 = 11111111_2 + 1 = (1)00000000 => Z and C flags are both raised.

49. When the result is greater than 255 (FFh), the C flag will be raised:
(a) 54H + C4H = 118H => C = 1
(b) 00H + FFH = FFH => C = 0
(c) FFH + 5H   = 104H => C = 1

50.

```
MOVLW 0H
ADDLW 55H
ADDLW 55H
ADDLW 55H
ADDLW 55H
ADDLW 55H
```

SECTION 2.5: PIC DATA FORMAT AND DIRECTIVES

51. MYDAT_1 = 55H
MYDAT_2 = 62H
MYDAT_3 = 47H
MYDAT_4 = 50H
MYDAT_5 = C8H
MYDAT_6 = 41H
MYDAT_7 = AAH
MYDAT_8 = FFH
MYDAT_9 = 90H
MYDAT_10 = 7EH
MYDAT_11 = 0AH
MYDAT_12 = 0FH

52. DAT_1 = 22H (default radix is HEX)
DAT_2 = 56H
DAT_3 = 99H
DAT_4 = 20H
DAT_5 = F6H
DAT_6 = FBH

53.

```
  R0 EQU 0H
  R1 EQU 1H
  R2 EQU 2H
  R3 EQU 3H
  R4 EQU 4H
  R5 EQU 5H
(a)
  MOVLW 11H
  MOVWF R0
  MOVWF R1
  MOVWF R2
  MOVWF R3
  MOVWF R4
  MOVWF R5
(b)
  MOVLW 0H
  ADDWF R0, W
  ADDWF R1, W
  ADDWF R2, W
  ADDWF R3, W
  ADDWF R4, W
  ADDWF R5, W
```

SECTION 2.6: INTRODUCTION TO PIC ASSEMBLY PROGRAMMING AND SECTION 2.7: ASSEMBLING AND LINKING A PIC PROGRAM

54. low - high
55. Assembly language, because it works directly with the internal hardware of the CPU and uses them efficiently.
56. Assembler
57. True.
58. .err
59. False.
60. True. ORG is optional for MPLAB.
61. No.
62. Because they only tell the assembler what to do and do not generate any op-codes.
63. True.
64. False.
65. True.
66. hex
67. lst, hex, map, out, cod

SECTION 2.8: THE PROGRAM COUNTER AND PROGRAM ROM SPACE IN THE PIC

68. 000000H
69. When we power up the microcontroller, its Program Counter register has the default value 000000H. Therefore it fetches the content of location 000000H of the ROM and tries to execute it since it expects to see the first op-code of the program. If we place the first opcode at location 100H, the ROM burner places 0FFH into ROM locations 00H to 0FFH. Since the CPU wakes up at location 000000H, it executes the opcode 0FFFFH, which is a NOP instruction. This will delay execution of the users code until it reaches location 100H.
70. (a) – (g) 2 bytes
    (h) 4 bytes

71. For ID number = 13590 we have:

```
ORG   0
;(a)
MOVLW 1H
MOVWF 0H
MOVLW 3H
MOVWF 1H
MOVLW 5H
MOVWF 2H
MOVLW 9H
MOVWF 3H
MOVLW 0H
MOVWF 4H
;(b)
MOVLW 0H
ADDWF 0H, W
ADDWF 1H, W
ADDWF 2H, W
ADDWF 3H, W
ADDWF 4H, W
MOVWF 6H
HERE  GOTO HERE
END
```

```
LOC   OBJECT CODE LINE         SOURCE TEXT

000000            00001         ORG   0
                  00002         ;(a)
000000 0E01       00003         MOVLW 1H
000002 6E00       00004         MOVWF 0H
000004 0E03       00005         MOVLW 3H
000006 6E01       00006         MOVWF 1H
000008 0E05       00007         MOVLW 5H
00000A 6E02       00008         MOVWF 2H
00000C 0E09       00009         MOVLW 9H
00000E 6E03       00010         MOVWF 3H
000010 0E00       00011         MOVLW 0H
000012 6E04       00012         MOVWF 4H
                  00013         ;(b)
000014 0E00       00014         MOVLW 0H
000016 2400       00015         ADDWF 0H, W
000018 2401       00016         ADDWF 1H, W
00001A 2402       00017         ADDWF 2H, W
00001C 2403       00018         ADDWF 3H, W
00001E 2404       00019         ADDWF 4H, W
000020 6E06       00020         MOVWF 6H
000022 EF11 F000  00021 HERE    GOTO HERE
                  00022         END
```

72.

WORD ADDRESS	HIGH BYTE	LOW BYTE
000000h	0Eh	01h
000002h	6Eh	00h
000004h	0Eh	03h
000006h	6Eh	01h
000008h	0Eh	05h
00000Ah	6Eh	02h
00000Ch	0Eh	09h
00000Eh	6Eh	03h
000010h	0Eh	00h
000012h	6Eh	04h
000014h	0Eh	00h
000016h	24h	00h
000018h	24h	01h
00001Ah	24h	02h
00001Ch	24h	03h
00001Eh	24h	04h
000020h	6Eh	06h
000022h	EFh	11h
000024h	F0h	00h

73. (a) Size of ROM=48*1024=49152 bytes=C000h => Last location=C000h-1= BFFFh
(b) Size of ROM=96*1024=98304 bytes=18000h => Last location=18000h-1= 17FFFh
(c) Size of ROM=64*1024=65536 bytes=10000h => Last location=10000h-1= FFFFh
(d) Size of ROM=16*1024=16384 bytes=4000h => Last location=4000h-1= 3FFFh
(e) Size of ROM=128*1024=131072 bytes=20000h => Last location=20000h-1=1FFFFh

74. Since the program counter of PIC18 has 21 bits, it takes from 000000h to 1FFFFFh

75. Size of ROM = Last location address+1 = 7FFFh + 1 = 8000h = 32768 bytes = 32KB

76. Size of ROM = Last location address+1 = 3FFh + 1 = 400h = 1024 bytes = 1KB

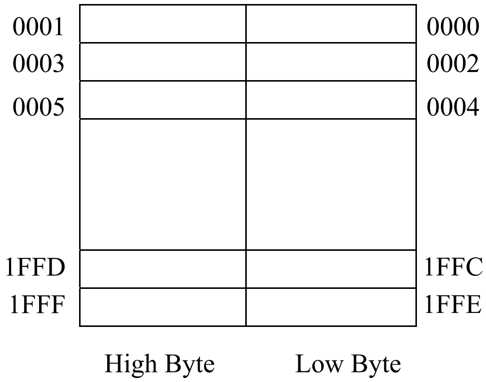
77. (a) Size of ROM=Last location address+1= 1FFFh + 1 = 2000h = 8192 bytes=8KB
(b) Size of ROM=Last location address+1=3FFFh + 1 = 4000h = 16384 bytes=16KB
(c) Size of ROM=Last location address+1=5FFFh + 1 = 6000h = 24576 bytes=24KB
(d) Size of ROM=Last location address+1=BFFFh + 1 = C000h = 49152 bytes=48KB
(e) Size of ROM=Last location address+1=FFFFh + 1 = 10000h = 65536 bytes=64KB
(f) Size of ROM=Last location address+1=1FFFFh + 1 = 20000h =131072 bytes=128KB
(g) Size of ROM=Last location address+1=2FFFh + 1 = 30000h = 196608 bytes=192KB
(h) Size of ROM=Last location address+1=3FFFh + 1 = 40000h = 262144 bytes =256KB

78. (a) Size of ROM = 4FFFFh+1 = 50000h = 327680 bytes = 320KB
(b) Size of ROM = 3FFFFh+1 = 40000h = 262144 bytes = 256KB
(c) Size of ROM = 5FFFFh+1 = 60000h = 393216 bytes = 384KB
(d) Size of ROM = 7FFFFh+1 = 80000h = 524288 bytes = 512KB
(e) Size of ROM = BFFFFh+1 = C0000h = 786432 bytes = 768KB
(f) Size of ROM = FFFFFh+1 = 100000h = 1048576 bytes = 1024KB
(g) Size of ROM = 17FFFFh+1 = 180000h = 1572864 bytes = 1536KB
(h) Size of ROM = 1FFFFFh+1 = 200000h = 2097152 bytes = 2048KB

79. 16 bits

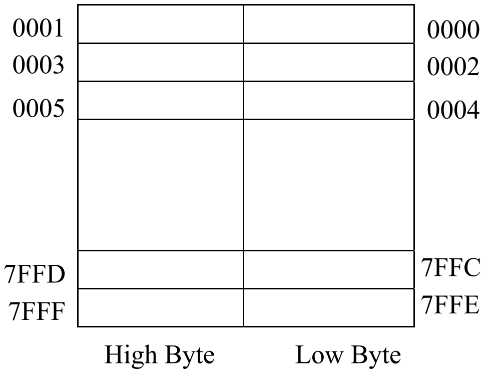
80. 16 bits

81.



82. 8KB

83.



84. 32KB

85. In Harvard architecture, there are 4 sets of buses: (1) a set of data buses for carrying data into and out of the CPU, (2) a set of address buses for accessing (addressing) the data, (3) a set of data buses for carrying code into the CPU, (4) an address bus for accessing (addressing) the code. With the use of four buses and separate accessing systems to data and code, none of them can get in the other's way and slow down the system.

86. Large number of CPU pins needed for duplicated buses – Large numbers of wire traces needed for the system wiring.

87. Because MOVLW is a 2-byte instruction. The first 8-bit is the main op-code and the second 8-bit is used for the literal value. A binary number with 8 bits can store at most 255 (11111111B – FFh).

88. Because ADDLW is a 2-byte instruction. The first 8-bit is the main op-code and the second 8-bit is used for the literal value. A binary number with 8 bits can store at most 255 (11111111B – FFh).

89. It is a 2-byte instruction. The first 8-bit is the op-code and the second 8-bit determines a location in a bank of the file register (Access bank or another bank which is determined by the LSB bit of the first 8-bit). Each bank has 256 bytes and can be covered by 8 bits for address.

90. It is a 4-byte instruction. The first 16-bit is the op-code and the source address and the second 16-bit is op-code and the destination address. In each 16-bit, 12 bits are set aside to determine the address which can cover the entire 4KB space of data RAM.

91. Because the data bus between the CPU and ROM of PIC18 microcontrollers is 16 bits wide. Whenever CPU fetches the data from ROM, it receives 2 bytes. Therefore data is always fetched starting from even addresses and because of this, program counter always holds even values and its LSB is 0.

92. Because the LSB of program counter is always 0. So it holds only even values (see problem 91).

93. Because there are only 2-byte and 4-byte instructions in PIC18 microcontrollers and all instructions are stored in ROM starting from even addresses. If program counter could get odd values, it would land at an odd address (in the middle of an instruction).

94. It is a 4-byte instruction that 20 bits out of these 32 bits are set aside to determine an address. As mentioned above, the LSB of the program counter is always zero to assure fetching op-codes from even addresses. 20 bits as well as this bit, form 21 bits that can cover 2MB of ROM.

SECTION 2.9: RISC ARCHITECTURE IN THE PIC

95. RISC stands for "Reduced Instruction Set Computer". CISC stands for "Complex (or Complete) Instruction Set Computer".
96. CISC
97. RISC
98. RISC
99. CISC
100. False

CHAPTER 3: BRANCH, CALL AND TIME DELAY LOOP

SECTION 3.1: BRANCH INSTRUCTIONS AND LOOPING

1. 255
2. the instruction following the branch instruction
3. Program Counter (the low byte)
4. Branch, 2
5. 4
6. BRA takes 2 bytes and GOTO take 4 bytes. So using BRA saves ROM space
7. False. All conditional branches are short jumps that the target can be within 128 bytes of the jump instruction.
8. False.
9. All are of the are 2 bytes except (c) which is 4 bytes.
10. It is a 2-byte instruction; The first 5 bits are the op-code and 11 bits are set aside for relative addressing of the target location. With 11 bits, the relative address is within -1024 to +1023y So the jump can be taken within a space of 2Kbytes (1023 bytes forward and 1024 bytes backward).
11. True
12.

```
R5     EQU   5
R6     EQU   6

       MOVLW D'10'
       MOVWF R5
BACK   MOVLW D'100'
       MOVWF R6
HERE   NOP
       NOP
       DECF  R6
       BNZ   HERE
       DECF  R5
       BNZ   BACK
```

13.

```
R4     EQU   4
R5     EQU   5
R6     EQU   6

       MOVLW D'10'
       MOVWF R6
BACK   MOVLW D'100'
       MOVWF R5
AGAIN  MOVLW D'100'
       MOVWF R4
HERE   NOP
       NOP
       DECF  R4
       BNZ   HERE
       DECF  R5
       BNZ   AGAIN
       DECF  R6
       BNZ   BACK
```

14. The BACK loops is performed 200 times. The HERE loops is performed 100 times and because is located in the BACK loop, is repeated 200 times. So the instructions in the HERE loop is performed 200*100 = 20,000 times.
15. Negative.
16. Positive.

SECTION 3.2: CALL INSTRUCTIONS AND STACK

17. 4
18. 2
19. False.
20. True
21. 1
22. 1
23. 01
24. When the subroutine is called, the CPU saves the address of the next instruction on to the top of the stack. At the end of the subroutine when the instruction RETURN is executed, the CPU takes the address from the top of stack and loads the PC register with it and continues to execute at the instruction after the CALL.
25. 31 locations of 21 bits each.
26. The address of the instruction after the CALL is pushed to the stack and the SP is decremented by 1.

SECTION 3.3: PIC18 TIME DELAY AND INSTRUCTION PIPELINE

27. Instruction frequency = 1/1.25µs = 800KHz; Oscillator frequency=800KHz*4=3.2MHz
28. 20MHz / 4 = 5 MHz; Instruction Cycle = 1 / 5 MHz = 200 ns
29. 10MHz / 4 = 2.5 MHz; Instruction Cycle = 1 / 2.5 MHz = 400 ns
30. 16MHz / 4 = 4 MHz; Instruction Cycle = 1 / 4 MHz = 250 ns
31. True

In the next 4 problems, we dispense with the overhead associated with the outer loops. Note that the BNZ instruction takes 2 instruction cycles when it takes jump and 1 instruction cycle when it falls through.

32. Instruction cycle = 1µs; HERE loop lasts (1+1+2)*100 – 1 = 399 instruction cycles. Overall delay = [399*200 + (1+1+1+2)*200] * 1µs = 80.8 ms
33. Instruction cycle = 250 ns; HERE loop lasts (1+1+1+2)*100 – 1 = 499 instruction cycles. Overall delay = [499*200 + (1+1+1+2)*200] * 250 ns = 25.2 ms
34. Instruction cycle = 1µs; HERE loop lasts (1+1+2)*250 – 1 = 999 instruction cycles. Overall delay = [999*200 + (1+1+1+2)*200] * 1µs = 200.8 ms
35. Instruction cycle = 400 ns; HERE loop lasts (1+2)*100 – 1 = 299 instruction cycles. Overall delay = [299*200 + (1+1+1+1+1+1+2)*200] * 400 ns = 24.56 ms. Note that the 3 NOPs are outside of the inner loop.

CHAPTER 4: PIC I/O PORT PROGRAMMING

SECTION 4.1: I/O PORT PROGRAMMING IN PIC18

1. 40
2. 4 pins. Pins 11 and 32 assigned to Vdd and pins 12 and 31 are assigned to Gnd.
3. 34
4. 7 Pins, 2-7 and 14.
5. 8 Pins, 33-40.
6. 8 Pins, 15-18 and 23-26.
7. 8 Pins, 19-22 and 27-30.
8. input
9. TRISx
10. Both are SFR registers associated to I/O ports. PORTx registers are used to access the I/O pins and TRISx registers are used to configure I/O ports as input or output.
11.

```
SETF  TRISC ;Define PORTC as input
CLRF  TRISB ;Define PORTB as output
CLRF  TRISD ;Define PORTD as output

MOVF  PORTC, W
NOP
MOVWF PORTB
MOVWF PORTD
```

12.

```
SETF  TRISD ;Define PORTD as input
CLRF  TRISB ;Define PORTB as output
CLRF  TRISC ;Define PORTC as output

MOVFF PORTD, PORTB
MOVFF PORTD, PORTC
```

13. 26 and 25 respectively
14. F80h, F81h and F82h for PORTx and F92h, F93h and F94h for TRISx.
15. (a)

```
      CLRF    TRISB
      CLRF    TRISC
BACK  MOVLW   55H
      MOVWF   PORTB
      MOVWF   PORTC
      CALL    DELAY
      MOVLW   AAH
      MOVWF   PORTB
      MOVWF   PORTC
      CALL    DELAY
      GOTO    BACK
```

(b)

```
      CLRF    TRISB
      CLRF    TRISC
      MOVLW   55H
      MOVWF   PORTB
      MOVWF   PORTC
BACK  CALL    DELAY
      COMF    PORTB
      COMF    PORTC
      BRA     BACK
```

SECTION 4.2: I/O BIT MANIPULATION PROGRAMMING

16. All SFR registers of the PIC18 (including I/O ports) are bit addressable.
17. The advantage of the bit addressing is that it allows each bit to be modified without affecting the other bits.
18. PORTB,RB2
19. Yes (since COMF is a read-modify-write instruction)
20.

```
       BCF   TRISB, 2 ;Define PB2 as output
       BCF   TRISB, 5 ;Define PB5 as output

BACK   BTG   PORTB, 2
       BTG   PORTB, 5
       CALL  DELAY
       BRA   BACK
```

21.

```
       BCF   TRISD, 3 ;Define PD3 as output
       BCF   TRISD, 7 ;Define PD7 as output
       BCF   TRISC, 5 ;Define PC5 as output

BACK   BTG   PORTD, 3
       BTG   PORTD, 7
       BTG   PORTC, 5
       CALL  DELAY
       BRA   BACK
```

22.

```
      BSF       TRISC, 3 ;Define PC3 as input
      CLRF      TRISD    ;Define PD as output
      MOVLW     0x55
BACK  BTFSS     PORTC, 3
      BRA       BACK
      MOVWF     PORTD
```

23.

```
      BSF       TRISB, 7 ;Define PB7 as input
      CLRF      TRISC    ;Define PC as output
OVER  BTFSC     PORTB, 7
      BRA       OVER
BACK  MOVLW     0x55
      MOVWF     PORTC
      CALL      DELAY
      MOVLW     0xAA
      MOVWF     PORTC
      CALL      DELAY
      BRA       BACK
```

24.

```
      BSF       TRISE, 0 ;Define PE0 as input
      CLRF      TRISB    ;Define PB as output
      CLRF      TRISC    ;Define PC as output
TEST  BTFSC     PORTE, 0
      BRA       BIT_H
      MOVLW     0x66
      MOVWF     PORTC
      BRA       TEST
BIT_H MOVLW     0x99
      MOVWF     PORTB
      BRA       TEST
```

25.

```
      BSF      TRISB, 5 ;Define PB5 as input
      BCF      TRISB, 3 ;Define PB3 as output
BACK  BTFSS    PORTB, 5
      BRA      BACK
      BCF      PORTB, 3
      BSF      PORTB, 3
      BCF      PORTB, 3
```

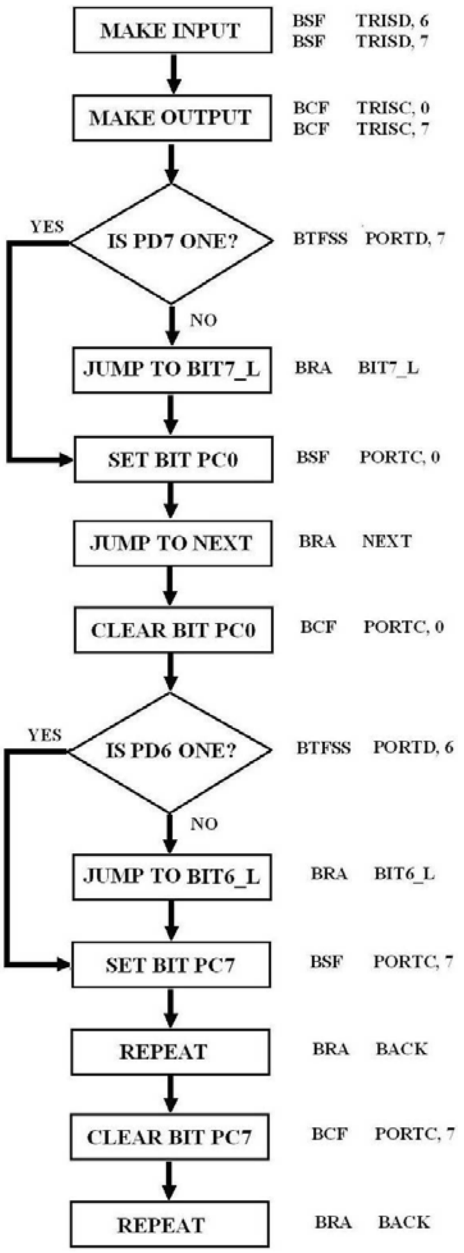
26.

```
       BSF      TRISC, 3 ;Define PC3 as input
       BCF      TRISC, 4 ;Define PC4 as output
 BACK  BTFSS    PORTB, 3
       BRA      BIT_L
       BSF      PORTB, 4
       BRA      BACK
 BIT_L BCF      PORTB, 4
       BRA      BACK
```

27. Bit number 4 (D4 of D0 – D7)

28.

```
       BSF     TRISD, 6 ;Define PD6 as input
       BSF     TRISD, 7 ;Define PD7 as input
       BCF     TRISC, 0 ;Define PC0 as output
       BCF     TRISC, 7 ;Define PC7 as output
BACK   BTFSS   PORTD, 7
       BRA     BIT7_L
       BSF     PORTC, 0
       BRA     NEXT
BIT7_L BCF     PORTC, 0
NEXT   BTFSS   PORTD, 6
       BRA     BIT6_L
       BSF     PORTC, 7
       BRA     BACK
BIT6_L BCF     PORTC, 7
       BRA       BACK
```



NOTE: All the codes above, are not complete program. To form a complete program, you should observe the standard structure of the PIC18(F458) programs which comes below. If you want to perform it on a real PIC cheap, maybe you should use **GOTO $** in some parts of programs to halt it.

```
      list P = PIC18F458
#include  P18F458.INC
      ORG   0
      ; Place your code here
      ----------

      HERE  GOTO  HERE ;Where you want to halt the program
      END
```

CHAPTER 5: ARITHMETIC, LOGIC INSTRUCTIONS ,AND PROGRAMS

SECTION 5.1: ARITHMETIC INSTRUCTIONS

1. All the calculations are done in hexadecimal system.

a)
$$\begin{array}{r} 3\ \ F \\ +4\ \ 5 \\ \hline 8\ \ 4 \end{array} \qquad C = 0, Z = 0, DC = 1$$

b)
$$\begin{array}{r} 9\ \ 9 \\ +5\ \ 8 \\ \hline F\ \ 1 \end{array} \qquad C = 0, Z = 0, DC = 1$$

c)
$$\begin{array}{r} 1 \\ +F\ \ F \\ +0\ \ 0 \\ \hline (1)\ 0\ \ 0 \end{array} \qquad C = 1, Z = 1, DC = 1$$

d)
$$\begin{array}{r} F\ \ F \\ +0\ \ 1 \\ \hline (1)\ 0\ \ 0 \end{array} \qquad C = 1, Z = 1, DC = 1$$

e)
$$\begin{array}{r} 1 \\ +F\ \ E \\ +0\ \ 0 \\ \hline F\ \ F \end{array} \qquad C = 0, Z = 0, DC = 0$$

f)
$$\begin{array}{r} 0 \\ F\ \ F \\ +0\ \ 0 \\ \hline F\ \ F \end{array} \qquad C = 0, Z = 0, DC = 0$$

2. For the ID number = 4305:

```
        MOVLW       D'4'
        ADDLW       D'3'
        DAW
        ADDLW       D'0'
        DAW
        ADDLW       D'5'
        DAW
        MOVWF       0x20
```

3.

```
MYREG       EQU   0x20
      MOVLW       0x25
      ADDLW       0x59
      ADDLW       0x65
      MOVWF       MYREG
```

4.

```
MYREG       EQU   0x20
      MOVLW       0x25
      ADDLW       0x59
      DAW
      ADDLW       0x65
      DAW
      MOVWF       MYREG
```

5. a)

```
      MOVLW       0x25
      MOVWF       0x20
      MOVWF       0x21
      MOVWF       0x22
      MOVWF       0x23
```

b)

```
      MOVF        0x20,W
      ADDWF       0x21,W
      ADDWF       0x22,W
      ADDWF       0x23,W
      MOVWF       0x60
```

6. a) 23H - 12H = 23H + 2's(12H) = 23H + EEH = (1) 11H → +11H
b) 43H - 53H = 43H + 2's(53H) = 43H + ADH = (0) F0H → -10H
c) 99H - 99H = 99H + 2's(99H) = 99H + 67H = (1) 00H → 0

7.

```
a) MOVLW    23H            b) MOVLW    43H            c) MOVLW    D'99'
   SUBLW    12H               SUBLW    53H               SUBLW    D'99'
```

8. True

9.

```
BYTE_L      EQU   0x40
BYTE_H      EQU   0x41
BYTE_U      EQU   0x42

      MOVLW       0x9A
      MOVWF       BYTE_L
      MOVLW       0x7F
      MOVWF       BYTE_H
      MOVLW       0
      MOVWF       BYTE_U

      MOVLW       0x48
      ADDWF       BYTE_L,F

      MOVLW       0xBC
      ADDWFC      BYTE_H,F

      MOVLW       0x00
      ADDWFC      BYTE_U,F
```

10.

```
BYTE_L      EQU   0x40
BYTE_H      EQU   0x41

      MOVLW       0x48
      MOVWF       BYTE_L
      MOVLW       0xBC
      MOVWF       BYTE_H

      MOVLW       0x9A
      SUBWF       BYTE_L,F

      MOVLW       0x7F
      SUBWFB      BYTE_H,F
```

11.

```
BYTE_L      EQU   0x40
BYTE_H      EQU   0x41
BYTE_U      EQU   0x42

      MOVLW       0x95
      MOVWF       BYTE_L
      MOVLW       0x77
      MOVWF       BYTE_H
      MOVLW       0
      MOVWF       BYTE_U

      MOVLW       0x48
      ADDWF       BYTE_L,W
      DAW
      MOVWF       BYTE_L

      MOVLW       0x95
      ADDWFC      BYTE_H,W
      DAW
      MOVWF       BYTE_H

      MOVLW       0x0
      ADDWFC      BYTE_U,W
      DAW
      MOVWF       BYTE_U
```

12.

```
MOVLW       D'77'
MULLW       D'34'

; The result will be in double register PRODH-PRODL
```

13.

```
NUM         EQU   0x19
MYQ         EQU   0x20
REM         EQU   0x21
DENUM       EQU   D'3'

      CLRF        MYQ
      MOVLW       D'77'
      MOVWF       NUM
      MOVLW       DENUM
AGAIN SUBWF       NUM,F
      INCF        MYQ,F
      BNN         AGAIN
      DECF        MYQ,F
      ADDWF       NUM,F
      MOVFF       NUM,REM
```

14. False

15. PRODH-PRODL

SECTION 5.2: SIGNED NUMBER CONCEPTS AND ARITHMETIC OPERATIONS

16. a) 1110 1001   b) 0000 1100   c) 1110 0100   d) 0110 1111
e) 1000 0000   f) 0111 1111

17. unsigned

18.

```
a) MOVLW   +D'15'      0000 1111 + 1111 0100 = (1)0000 0011
   ADDLW   -D'12'      OV = 0

b) MOVLW   -D'123'     1000 0101 + 1000 0001 = (1)0000 0110
   ADDLW   -D'127'     OV = 1

c) MOVLW   +25H        0010 0101 + 0011 0100 = (0)0101 1001
   ADDLW   +34H        OV = 0

d) MOVLW   -D'127'     1000 0001 + 0111 1111 = (1)0000 0000
   ADDLW   +D'127'     OV = 0
```

19. C flag is raised when there is a carry out of D7 of the result, but OV flag is raised when there is a carry from D6 to D7 and no carry out of D7 or when there is no carry from D6 to D7 and there is a carry out of D7 of the result. C flag is used to indicate overflow in unsigned arithmetic operations while OV flag is involved in signed operations.

20. When there is a carry from D6 to D7 of result, but there is no carry out of D7 (C = 0)
OR
when there is no carry from D6 to D7 of result, but there is a carry out of D7 (C=1)

21. STATUS

22. BOV and BNOV instructions - BC and BNC instructions

SECTION 5.3: LOGIC AND COMPARE INSTRUCTIONS

23. All the results are stored in WREG register.
a) 0x40   b) 0xF0   c) 0x86   d) 0x90   e) 0x60   f) 0xF0
g) 0xF0   h) 0xF9   i) 0x1E   j) 0x5A

24. a) 0x64   b) 0x7B   c) 0x3F   d) 0x58   e) 0xD7   f) 0x04
g) 0x37

25. True

26. "CPFSGT FileReg" subtracts WREG from FileReg without changing them. If the result is positive, it means FileReg > WREG and the next instructions is ignored (skipped over). Otherwise the next instruction will be performed.

27. No

28. a) 85h < 90h → No skip   b) 85h > 70h → Skip
c) Skip   d) 85h > 5Dh → No skip

29. a) 86h   b) 85h   d) 85h   e) 86h

SECTION 5.4: ROTATE INSTRUCTION AND DATA SERIALIZATION

30.

```
a) MOVLW   0x56        ; WREG = 0x56
   MOVWF   MYREG       ; WREG = 0x56 , MYREG = 0x56 = 0101 0110
   SWAPF   MYREG,F     ; WREG = 0x56 , MYREG = 0x65 = 0110 0101
   RRCF    MYREG,F     ; WREG = 0x56 , MYREG = 0x32 = 0011 0010 (C = 1)
   RRCF    MYREG,F     ; WREG = 0x56 , MYREG = 0x99 = 1001 1001 (C = 0)

b) MOVLW   0x39        ; WREG = 0x39
   BCF     STATUS, C   ; WREG = 0x39 , C = 0
   MOVWF   MYREG,F     ; WREG = 0x39 , MYREG = 0x39 = 0011 1001
   RLCF    MYREG,F     ; WREG = 0x39 , MYREG = 0x72 = 0111 0010 (C = 0)
   RLCF    MYREG,F     ; WREG = 0x39 , MYREG = 0xE4 = 1110 0100 (C = 0)

c) BCF     STATUS, C   ; C = 0
   MOVLW   0x4D        ; WREG = 0x4D
   MOVWF   MYREG       ; WREG = 0x4D , MYREG = 0x4D = 0100 1101
   SWAPF   MYREG,F     ; WREG = 0x4D , MYREG = 0xD4 = 1101 0100
   RRCF    MYREG,F     ; WREG = 0x4D , MYREG = 0x6A = 0110 1010 (C = 0)
   RRCF    MYREG,F     ; WREG = 0x4D , MYREG = 0x35  = 0011 0101 (C = 0)
   RRCF    MYREG,F     ; WREG = 0x4D , MYREG = 0x1A  = 0001 1010 (C = 1)
d) BCF     STATUS, C   ; C = 0
   MOVLW   0x7A        ; WREG = 0x7A
   MOVWF   MYREG       ; WREG = 0x7A , MYREG = 0x7A = 0111 1010
   SWAPF   MYREG,F     ; WREG = 0x7A , MYREG = 0xA7 = 1010 0111
   RLCF    MYREG,F     ; WREG = 0x7A , MYREG = 0x4E = 0100 1110 (C = 1)
   RLCF    MYREG,F     ; WREG = 0x7A , MYREG = 0x9D  = 1001 1101 (C = 0)
```

31.

```
a) RRNCF   FileReg, F          b) RLNCF   FileReg, F
   RRNCF   FileReg, F             RLNCF   FileReg, F
   RRNCF   FileReg, F             RLNCF   FileReg, F
   RRNCF   FileReg, F             RLNCF   FileReg, F
```

32.

```
MYREG       EQU   0x20 ;Contains data
COUNTER     EQU   0x21
RESULT      EQU   0x22

      CLRF        RESULT

      MOVLW       8
      MOVWF       COUNTER
AGAIN RRCF        MYREG,F
      BC          NEXT
      INCF        RESULT, F
NEXT  DECF        COUNTER,F
      BNZ         AGAIN
```

33.

```
MYREG       EQU   0x20
COUNTER     EQU   0x21
POSITION    EQU   0x22

      CLRF        POSITION

      MOVLW       8
      MOVWF       COUNTER
      MOVLW       68H
      MOVWF       MYREG

AGAIN RRCF        MYREG,F
      BC          OVER
      INCF        POSITION, F
      DECF        COUNTER,F
      BNZ         AGAIN
OVER
      ;POSITION = 3 FOR 68H (0110 1000)
```

If POSITION is 8 at the location OVER, it means no high bit is found in MYREG.

34.

```
MYREG       EQU   0x20
COUNTER     EQU   0x21
POSITION    EQU   0x22
      MOVLW       7
      MOVWF       POSITION
      MOVLW       8
      MOVWF       COUNTER
      MOVLW       68H
      MOVWF       MYREG
AGAIN RLCF        MYREG,F
      BC          OVER
      DECF        POSITION, F
      DECF        COUNTER,F
      BNZ         AGAIN
OVER
      ;POSITION = 6 FOR 68H (0110 1000)
```

If POSITION is 255 at the location OVER, it means no high bit is found in MYREG.

35. Consider value CCh (1100 1100). The sequential rotation is shown below:

right rotation: **1100 1100** → 0110 0110 → 0011 0011 → 1001 1001 → **1100 1100** → …
left rotation : **1100 1100** → 1001 1001 → 0011 0011 → 0110 0110 → **1100 1100** → …

As you see, after four rotations the value comes back to its original value. Furthermore, 4 lower bits and 4 upper bits generate stepper motor patterns independently.

```
MYREG     EQU       0x20

          MOVLW     0xCC
          MOVWF     MYREG
AGAIN     RRNCF     MYREG, F   (or   RLNCF      MYREG, F)
          BRA       AGAIN
```

SECTION 5.5: BCD AND ASCII CONVERSTION

36. In the following program we assume Little Endian storing scheme.

```
ASCII1      EQU   0x40
ASCII2      EQU   0x41
ASCII3      EQU   0x42
ASCII4      EQU   0x43

      MOVLW       MYBCD_1     ; WREG = 0x76
      ANDLW       0x0F        ; WREG = 0x06
      IORLW       0x30        ; WREG = 0x36
      MOVWF       ASCII1      ; ASCII1 = 0x36 = '6'

      MOVLW       MYBCD_1     ; WREG = 0x76
      ANDLW       0xF0        ; WREG = 0x70
      MOVWF       ASCII2      ; ASCII2 = 0x70
      SWAPF       ASCII2,F    ; ASCII2 = 0x07
      MOVLW       0x30        ; WREG = 0x30
      IORWF       ASCII2,F    ; ASCII2 = 0x37 = '7'

      MOVLW       MYBCD_2     ; WREG = 0x87
      ANDLW       0x0F        ; WREG = 0x07
      IORLW       0x30        ; WREG = 0x37
      MOVWF       ASCII3      ; ASCII3 = 0x37 = '7'

      MOVLW       MYBCD_2     ; WREG = 0x87
      ANDLW       0xF0        ; WREG = 0x80
      MOVWF       ASCII4      ; ASCII4 = 0x80
      SWAPF       ASCII4,F    ; ASCII4 = 0x08
      MOVLW       0x30        ; WREG = 0x30
      IORWF       ASCII4,F    ; ASCII4 = 0x38 = '8'
```

37.

```
BCD1        EQU   0x60
BCD2        EQU   0x61

      MOVLW       MYASC_1     ; WREG = '8' = 0x38
      ANDLW       0x0F        ; WREG = 0x08
      MOVWF       BCD1        ; BCD1 = 0x08
      SWAPF       BCD1,F      ; BCD1 = 0x80

      MOVLW       MYASC_2     ; WREG = '7' = 0x37
      ANDLW       0x0F        ; WREG = 0x07
      IORWF       BCD1,F      ; BCD1 = 0x87

      MOVLW       MYASC_3     ; WREG = '9' = 0x39
      ANDLW       0x0F        ; WREG = 0x09
      MOVWF       BCD2        ; BCD2 = 0x09
      SWAPF       BCD2,F      ; BCD2 = 0x90

      MOVLW       MYASC_4     ; WREG = '2' = 0x32
      ANDLW       0x0F        ; WREG = 0x02
      IORWF       BCD2,F      ; BCD2 = 0x92
```

CHAPTER 6: BANK SWITCHING, TABLE PROCESSING, MACROS AND MODULES

SECTION 6.1: IMMEDIATE AND DIRECT ADDRESSING MODES

1. b. MOVLW takes only one operand which is an immediate value.
2. a) Direct b) Immediate c) Direct d) Immediate e) Direct f) Direct
3. According to Figure 2-4:
   a) F81h b) FE8h c) F82h d) F83h e) FF9h f) FFAh g) FFBh h) F94h
   i) F93h j) FD8h k) FE9h

4. Bank 15 (F)
5. Direct
6. Copies the value F0h to WREG register.
7. Copies the contents of WREG register to PORTC.
8. Copies the contents of PORTC to WREG register.
9. valid
10. 00 - 7F
11. F80h - FFFh
12. a) 00 - 7F b) 80 - FF There is no gap between them.
13.

```
MOVLW 6
ADDLW 9
ADDLW 2
ADDLW 5
ADDLW 7
MOVWF 0X20
```

SECTION 6.2: REGISTER INDIRECT ADDRESSING MODE

14. FSR0 (LFSR 0, RAM ADDRESS), FSR1 (LFSR 1, RAM ADDRESS),
    FSR2 (LFSR 2, RAM ADDRESS)
15.

```
RAM_ADDR    EQU   50H
COUNT_REG   EQU   0x20
COUNT_VAL   EQU   1FH

      MOVLW COUNT_VAL
      MOVWF COUNT_REG

      LFSR  0,RAM_ADDR
      MOVLW 0xFF

AGAIN MOVWF INDF0
      INCF  FSR0L,F
      DECF  COUNT_REG,F
      BNZ   AGAIN
```

16.

```
COUNT_VAL    EQU   D'10'
COUNT_REG    EQU   0x20
SRC_ADDR     EQU   40H
DEST_ADDR    EQU   70H

      ORG   0

      MOVLW COUNT_VAL
      MOVWF COUNT_REG

      LFSR  0,SRC_ADDR
      LFSR  1,DEST_ADDR

NEXT  MOVF  INDF0,W
      MOVWF INDF1

      INCF  FSR0L,F
      INCF  FSR1L,F

      DECF  COUNT_REG,F
      BNZ   NEXT

      BRA   $
END
```

17. 12 bits

18. FSR0: FSR0L, FSR0H,   FSR1: FSR1L, FSR1H

19.

```
COUNT_VAL    EQU   7FH
COUNT_REG    EQU   0x20

      ORG   0

      MOVLW COUNT_VAL
      MOVWF COUNT_REG

      LFSR  2,0x00

AGAIN CLRF  POSTINC2,F
      DECF  COUNT_REG,F
      BNZ   AGAIN

      BRA   $
END
```

20.

```
COUNT_VAL   EQU   0FH
COUNT_REG   EQU   0x20

      ORG   0

      MOVLW COUNT_VAL
      MOVWF COUNT_REG

      LFSR  0,0x50

NEXT  COMF  POSTINC0,F
      DECF  COUNT_REG,F
      BNZ   NEXT

      BRA   $
END
```

21. INDFx keeps content of the RAM location with address FSRx.
22. 4KB

SECTION 6.3: LOOK-UP TABLE AND TABLE PROCESSING

23.

ROM Location (hex)	**200**	**201**	**202**	**203**	**204**	**205**	**206**	**207**	**208**	**209**
Contents (hex)	45	61	72	74	68	00	39	38	37	2D
ROM Location (hex)	**20A**	**20B**	**20C**	**20E**	**20F**	**210**	**211**	**212**	**208**	**209**
Contents (hex)	36	35	47	41	42	45	48	20	39	38

NOTE: 1- When you see the contents of these cells of memory in MPLAB, you will see that the contents of each two sequent bytes are displayed in reverse order (high byte is shown first). This is done to ease dealing with little endian storing scheme.
2- Every new DB directive, will align consequent bytes on the next even address (notice 00 in location 205)

24.

ROM Location (hex)	340	341	342	343	344	345
Contents (hex)	22	56	99	20	F6	FB

25. TBLPTRL, TBLPTRH and TBLPTRU. They are loaded with these instructions:

MOVLW   proper portion of a ROM address
MOVWF   TBLPTRx

which x is L (for loading bits 0-7 of pointer), H (for loading bits 8-15 of pointer) or U (for loading bits 16-20 of pointer. Bits 21-23 are set to zero).

26. TABLAT keeps content of the ROM location with address TBLPTR.
27. 21 bits. 2 MB
28. TBLPTRL, TBLPTRH and TBLPTRU

29.

```
RAM_ADDR    EQU   50H

      ORG   0000
      MOVLW 0x00
      MOVWF TBLPTRL
      MOVLW 0x6
      MOVWF TBLPTRH
      LFSR  0,RAM_ADDR
NEXT  TBLRD*+
      MOVF  TABLAT, W
      BZ    EXIT
      MOVWF POSTINC0
      BRA   NEXT
EXIT  BRA   EXIT
;----------MESSAGE
      ORG   0x600
DB    "1-800-999-9999",0
END
```

30. In the following program we assume to get "x" from PORTB and send "y" to PORTC.

```
      ORG   0000
      SETF  TRISB
      CLRF  TRISC
NEXT  MOVF  PORTB, W
      ANDLW 0x0F
      CALL  Y_TABLE
      MOVWF PORTC
      BRA   NEXT

;------- Y_TABLE SUBROUTINE
Y_TABLE
      MULLW 0x2
      MOVF  PRODL, W
      ADDWF PCL,F
      DT    D'5', D'8', D'13', D'20', D'29', D'40'
      DT    D'53', D'68', D'85', D'104'
END
```

We can replace two last lines of the Y_TABLE with:

retlw D'5'
retlw D'8'
…
retlw D'85'
retlw D'104'

31. We solve this problem by placing look-up table in data RAM. We assume that contents of RAM locations starting at address 20h are: 5-25-45-65-85-105-125-145-165-185. Input and output ports are PORTB and PORTC respectively.

```
      ORG   0000

      SETF  TRISB
      CLRF  TRISC

      LFSR  0, 0x20
NEXT  MOVF  PORTB, W
      ANDLW 0x0F
      MOVFF PLUSW0,PORTC
      BRA   NEXT
END
```

32.

```
RAM_ADDR    EQU   40H

      ORG   0000

      MOVLW 0x00
      MOVWF TBLPTRL
      MOVLW 0x7
      MOVWF TBLPTRH

      LFSR  0,RAM_ADDR
NEXT  TBLRD*+
      MOVF  TABLAT, W
      BZ    EXIT
      MOVWF POSTINC0
      BRA   NEXT
EXIT  BRA   EXIT
;----------MESSAGE
      ORG   0x700
DB    "The earth is but one country",0
END
```

33. True

34. True

35.

```
      ORG   0000

      SETF  TRISB
      CLRF  TRISC

NEXT  MOVF  PORTB, W
      ANDLW 0x0F
      CALL  ASCII_TABLE
      MOVWF PORTC
      BRA   NEXT

;------- TABLE PROCESSING
ASCII_TABLE
      MULLW 0x2
      MOVFF PRODL, WREG
      ADDWF PCL,F
      DT "0123456789ABCDEF"
END
```

SECTION 6.4: BIT-ADDRESSABILITY OF DATA RAM

36.

```
COUNT_VAL   EQU   0xFF
COUNT_REG   EQU   0x20

      ORG   0

      BCF   TRISB,5

AGAIN BSF   PORTB,5
      CALL  DELAY
      CALL  DELAY
      CALL  DELAY
      BCF   PORTB,5
      CALL  DELAY
      BRA   AGAIN

;------ DELAY SUBROUTIN
DELAY
      MOVLW COUNT_VAL
      MOVWF COUNT_REG
LOOP  NOP
      NOP
      DECF  COUNT_REG,F
      BNZ   LOOP
      RETURN
END
```

37.

```
COUNT_VAL   EQU   0xFF
COUNT_REG   EQU   0x20

      ORG   0
      BCF   TRISC,7
AGAIN BSF   PORTC,7
      CALL  DELAY
      CALL  DELAY
      CALL  DELAY
      CALL  DELAY
      BCF   PORTC,7
      CALL  DELAY
      BRA   AGAIN
;----- DELAY SUBROUTINE
DELAY
      MOVLW COUNT_VAL
      MOVWF COUNT_REG
LOOP  NOP
      NOP
      DECF  COUNT_REG,F
      BNZ   LOOP
      RETURN
END
```

38.

```
COUNT_VAL   EQU   0xFF
COUNT_REG   EQU   0x20

      ORG   0

      BSF   TRISB,4
      BCF   TRISB,7

HERE  BTFSS PORTB,4
      BRA   HERE

AGAIN BTG   PORTB,7
      CALL  DELAY
      BRA   AGAIN

;------ DELAY SUBROUTIN
DELAY
      MOVLW COUNT_VAL
      MOVWF COUNT_REG
LOOP  NOP
      NOP
      DECF  COUNT_REG,F
      BNZ   LOOP
      RETURN
END
```

39.

```
COUNT EQU 0x20
  ORG 0
  BSF TRISC,1
  BCF TRISD,0
AGAIN BTFSS PORTC,1
  BRA AGAIN
  MOVLW 0x08
  MOVWF COUNT
  MOVLW     0x55
BACK
  RRCF WREG
  BC ONE
  BCF PORTD,0
  BRA OVER
ONE BSF PORTD,0
OVER DECFSZ COUNT
  BRA BACK
HERE BRA HERE
  END
```

40. STATUS
41. Z flag is D2 of STATUS register.
42. a) valid: RB1   b) invalid   c) valid: bit number 1 of WREG
d) valid: bit number 1 of RAM location 0x30   e) valid: RD0
f) (BTG) valid: C flag of STATUS register   g) invalid   h) invalid
NOTE: instructions in parts (g) and (h) do not have syntax error, but can not manipulate bits.
43. valid
44. All of them
45. The entire data RAM is bit-addressable.
46. BCF   STATUS, 0 ; C flag is D0 of STATUS register
47. BC instruction
48. BZ instruction
49. C flag : bit 0, DC flag : bit 1, Z flag : bit 2, OV flag : bit 3
50. True
51. False
52.

```
AGAIN BTFSS STATUS, 0 ; C flag is D0 of STATUS register
      BRA   OVER
      BSF   0x10,4
      BRA   AGAIN
OVER  BCF   0x10,4
      BRA   AGAIN
```

53.

```
AGAIN BTFSS STATUS, 1 ; DC flag is D1 of STATUS register
      BRA   OVER
      BSF   0x16,2
      BRA   AGAIN
OVER  BCF   0x16,2
      BRA   AGAIN
```

54.

```
AGAIN BTFSS STATUS, 2 ; Z flag is D2 of STATUS register
      BRA   OVER
      BSF   0x12,7
      BRA   AGAIN
OVER  BCF   0x12,7
      BRA   AGAIN
```

55.

```
  BTFSC WREG,0
  BRA EXIT
  BTFSC WREG,1
  BRA EXIT
  RRNCF WREG
  RRNCF WREG
EXIT
```

56. In the following program we assume that LCD is connected to PORTB

```
      ORG   0

      CLRF  TRISB,F

      BTFSS WREG,7
      BRA   EXIT

      MOVLW 0x00
      MOVWF TBLPTRL
      MOVLW 0x2
      MOVWF TBLPTRH
NEXT  TBLRD*+
      MOVF  TABLAT,W
      BZ    EXIT
      MOVWF PORTB
      CALL DISPLAY
      BRA   NEXT
EXIT  BRA   EXIT
      ORG   200
DB    "WREG CONTAINS A NEGATIVE VALUE",0
EXIT
      END
```

57. Main part of program:

```
a)    SETF  0x20,F
b)    BSF   0x20,0
      BSF   0x20,1
      BSF   0x20,2
      BSF   0x20,3
      BSF   0x20,4
      BSF   0x20,5
      BSF   0x20,6
      BSF   0x20,7
```

58. Bits 0-2 of values divisible by 8 are zero. So we test these bits of WREG.

```
      ANDLW WREG,B'00000111'
      BZ    DIVISIBLE

; INSTRUCTIONS THAT SHOULD BE PERFORMED
; IN THE CASE THAT WREG IS NOT DIVISIBLE BY 8
-----
-----

DIVISIBLE:
; INSTRUCTIONS THAT SHOULD BE PERFORMED
; IN THE CASE THAT WREG IS DIVISIBLE BY 8
-----
-----
```

59.

```
RESULT      EQU   0x20
COUNT_VAL   EQU   0x8
COUNT_REG   EQU   0x21
MYREG       EQU   0X5

      ORG   0

      MOVLW COUNT_VAL
      MOVWF COUNT_REG

      MOVLW 0
      MOVWF RESULT

      BCF   STATUS, 0 ; CLEAR CARRY FLAG
AGAIN RRCF  MYREG,F
      BC    NEXT
      INCF  RESULT,F
NEXT  DECF  COUNT_REG,F
      BNZ   AGAIN

      BRA   $
END
```

SECTION 6.5: BANK SWITCHING IN THE PIC18

60. Direct - Register Indirect
61. Direct - Register Indirect
62. The lower 128 bytes of access bank are locations 00-7F of RAM . The upper 128 bytes of access bank are locations F80-FFF of data RAM.
63. F80-FFF
64. 16
65. "ADDWF   MYREG,F,0" adds the WREG to MYREG which is considered to be in access bank. "ADDWF   MYREG,F,1" does the same, but considers MYREG register in the bank selected by BSR register.
66. We load the BSR register via immediate addressing mode and manipulate bank registers via direct or register indirect addressing mode.
67.

```
COUNT_VAL   EQU   0x0F
COUNT_REG   EQU   0x20

      ORG   0

      MOVLW COUNT_VAL
      MOVWF COUNT_REG

      LFSR  0,0x1C0

      MOVLW 0x55
NEXT  MOVWF POSTINC0
      DECF  COUNT_REG,F
      BNZ   NEXT

      BRA   $
      END
```

68.

```
COUNT_VAL   EQU   0x0F
COUNT_REG   EQU   0x5

      ORG      0

      MOVLW    COUNT_VAL
      MOVWF    COUNT_REG

      LFSR     0,0x20
      LFSR     1,0X2D0

NEXT  MOVFF    POSTINC0,POSTINC1
      DECF     COUNT_REG,F
      BNZ      NEXT

      BRA      $
      END
```

69. "CLRF MYREG,F,0" clears all the bits of MYREG which is considered to be in access bank. "CLRF MYREG,F,1" does the same, but considers MYREG register in the bank selected by BSR register.

70. "SETF MYREG,F,0" sets all the bits of MYREG which is considered to be in access bank to high. "SETF MYREG,F,1" does the same, but considers MYREG register in the bank selected by BSR register.

71. "INCF MYREG,F,0" increases the contents of MYREG which is considered to be in access bank by 1. "INCF MYREG,F,1" does the same, but considers MYREG register in the bank selected by BSR register.

SECTION 6.6: CHECKSUM AND ASCII SUBROUTINES

72. We first add the corresponding ASCII values of characters in message "Hello" together:
s = 48h + 65h + 6Ch + 6Ch + 6Fh = 1F4h
Then we drop the carry and convert the remainder (F4h) to 2's complement form (0Ch) which is the check-sum.

73. True

74.
```
#include P18F458.inc
RAM_ADDR    EQU 40H     ;RAM space to place the bytes
COUNTREG    EQU 0x20    ;fileReg loc for counter
CNTVAL      EQU 31      ;counter value = 4 for adding 4 bytes
CNTVAL1     EQU 32      ;counter value = 5 for adding 5 bytes
                        ;including checksum byte
;------------main program
      ORG   0
      CALL  COPY_DATA
      CALL  CAL_CHKSUM
      CALL  TEST_CHKSUM
      BRA   $
;--------copying data from code ROM address 500H to data RAM loc
COPY_DATA
      MOVLW low(MYBYTE)       ;WREG = 00 LOW-byte addr
      MOVWF TBLPTRL           ;ROM data LOW-byte addr
      MOVLW hi(MYBYTE)        ;WREG = 5, HIGH-byte addr
      MOVWF TBLPTRH           ;ROM data HIGH-byte addr
      MOVLW upper(MYBYTE)     ;WREG = 00 upper-byte addr
      MOVWF TBLPRTRU          ;ROM data upper-byte addr
      LFSR  0,RAM_ADDR        ;FSR0 = RAM_ADDR, place to save
C1    TBLRD*+                 ;bring in next byte and inc TBLPTR
      MOVF  TABLAT,W          ;copy to WREG (Z = 1, if null)
      BZ    EXIT              ;is it null char? exit if yes
      MOVWF POSTINC0          ;copy WREG to RAM and inc pointer
      BRA   C1
EXIT  RETURN
;-----calculating checksum byte
CAL_CHKSUM
      MOVLW CNTVAL            ;WREG = 4
      MOVWF COUNTREG          ;load the counter, count = 4
      LFSR  0,RAM_ADDR        ;load pointer. FSR0 = 40H
      CLRF  WREG
C2    ADDWF POSTINC0,W        ;add RAM to WREG and increment FSR0
      DECF  COUNTREG,F        ;decrement counter
      BNZ   C2                ;loop until counter = zero
      XORLW 0xFF              ;1’s comp
      ADDLW 1                 ;2’compl
      MOVWF POSTINC0
      RETURN
;----------testing checksum byte
TEST_CHKSUM
      MOVLW CNTVAL1           ;WREG = 5
      MOVWF COUNTREG          ;load the counter, count = 5
      CLRF  TRISB             ;PORTB = output
      LFSR  0,RAM_ADDR        ;load pointer. FSR0 = 40H
      CLRF  WREG
C3    ADDWF POSTINC0,W        ;add RAM and increment FSR0
      DECF  COUNTREG,F        ;decrement counter
      BNZ   C3                ;loop until counter = zero
      XORLW 0x0               ;EX-OR to see if WREG = zero
      BZ    G_1               ;is result zero? then good
      MOVLW 'B'
      MOVWF PORTB             ;if not, data is bad
      RETURN
G_1   MOVLW 'G'
      MOVWF PORTB             ;data is not corrupted
      RETURN

;----------my data in program ROM
      ORG 0x500
MYBYTE DB “Hello, my fellow world citizens”
      END
```

75. ASCII

76.

```
      ORG   0000

      SETF  TRISB
      CLRF  TRISD

NEXT  MOVF  PORTB, W
      ANDLW 0x0F
      CALL  ASCII_TABLE
      MOVWF PORTD
      BRA   NEXT

;------- TABLE PROCESSING
ASCII_TABLE
      MULLW 0x2
      MOVF  PRODL, W
      ADDWF PCL,F
      DT "0123456789"
      END
```

77.

```
#include P18F458.inc
RAM_ADDR    EQU 0x20
ASC_RAM     EQU 0x40
COUNTREG    EQU 0x1F          ;fileReg loc for counter
CNTVAL      EQU D'4'          ;counter value of BCD bytes
CNTVAL1     EQU D'8'          ;counter value of ASCII bytes
;------------main program
```

```
      ORG   0
      CALL  COPY_DATA
      CALL  BCD_ASC_CONV
      CALL  DISPLAY
      BRA   $
;--------copying data from code ROM to data RAM
COPY_DATA
      MOVLW low(MYBYTE)       ;WREG = 00 LOW-byte addr
      MOVWF TBLPTRL           ;ROM data LOW-byte addr
      MOVLW hi(MYBYTE)        ;WREG = 5, HIGH-byte addr
      MOVWF TBLPTRH           ;ROM data HIGH-byte addr
      MOVLW upper(MYBYTE)     ;WREG = 00 upper-byte addr
      MOVWF TBLPRTRU          ;ROM data upper-byte addr
      LFSR  0,RAM_ADDR        ;FSR0 = RAM_ADDR, place to save
C1    TBLRD*+                 ;bring in next byte and inc TBLPTR
      MOVF  TABLAT,W          ;copy to WREG (Z = 1, if null)
      BZ    EXIT              ;is it null char? exit if yes
      MOVWF POSTINC0          ;copy WREG to RAM and inc pointer
      BRA   C1
EXIT  RETURN
;-----convert packed BCD to ASCII
BCD_ASC_CONV
      MOVLW CNTVAL            ;get the counter value
      MOVWF COUNTREG          ;load the counter
      LFSR  0,RAM_ADDR        ;FSR0 = RAM_ADR BCD byte pointer
      LFSR  1,ASC_RAM         ;FSR1 = ASC_RAM ASCII byte pointer
B2    MOVF  INDF0,W           ;copy BCD to WREG
      ANDLW 0x0F              ;mask the upper nibble (W = 09)
      IORLW 0x30              ;make it an ASCII
      MOVWF POSTINC1          ;copy to RAM and increment FSR1
      MOVF  POSTINC0,W        ;note the use of instruction
      ANDLW 0xF0              ;mask the lower nibble (W = 20H)
      SWAPF WREG
      IORLW 0x30              ;make it an ASCII
      MOVWF POSTINC1          ;copy to RAM and increment FSR1
      DECF  COUNTREG,F        ;decrement counter
      BNZ   B2                ;loop until counter = zero
      RETURN
;-----send ASCII data to port B
DISPLAY
      CLRF  TRISB             ;make PORTB output (TRSIB = FFH)
      MOVLW CNTVAL1           ;WREG = 8, send 8 bytes of data
      MOVWF COUNTREG          ;load the counter, count = 8
      LFSR  2,ASC_RAM         ;load pointer. FSR2 = 50H
B3    MOVF  POSTINC2,W        ;copy RAM to WREG and inc pointer
      MOVWF PORTB             ;copy WREG to PORTB
      DECF  COUNTREG,F        ;decrement counter
      BNZ   B3                ;loop until counter = zero
      RETURN

;----------my BCD data in program ROM
      ORG 0x700
MYBYTE DB 76H, 87H, 98H, 43H, 00H
      END
```

78.

```
#include    P18F458.INC

COUNT_VAL   EQU   4
COUNT_REG   EQU   0x20
MYBCD       EQU   0x21
RAM_ADDR    EQU   0x40
BCD_RAM     EQU   0x60

; -------- main program
      ORG   0
      CALL  COPY_DATA
      CALL  ASC_BCD_CONV
      BRA   $

;--------- copying data from code ROM to data RAM
COPY_DATA
      MOVLW 0x00
      MOVWF TBLPTRL
      MOVLW 0x03
      MOVWF TBLPTRH
      LFSR  0,RAM_ADDR
C1    TBLRD*+
      MOVF  TABLAT,W
      BZ    EXIT
      MOVWF POSTINC0
      BRA   C1
EXIT  RETURN

;---------- convert ASCII to BCD
ASC_BCD_CONV
      MOVLW COUNT_VAL
      MOVWF COUNT_REG
      LFSR  0,RAM_ADDR
      LFSR  1,BCD_RAM
B1    MOVF  POSTINC0,W
      ANDLW 0x0F
      MOVWF MYBCD
      SWAPF MYBCD,F
      MOVF  POSTINC0,W
      ANDLW 0x0F
      IORWF MYBCD,W
      MOVWF POSTINC1
      DECF  COUNT_REG,F
      BNZ   B1
      RETURN

;---------- My ASCII data in program ROM
      ORG   300H
MYDATA      DB    "87675649"

      END
```

79. If PORTD = 1000 1101 (141), the contents of RAM locations will be:

0x40 -- '1' = 0x31   0x41 -- '4' = 0x34   0x43 -- '1' = 0x39

```
#include P18F458.INC
NUME        EQU   0x00        ;RAM loc for NUME
QU          EQU   0x20        ;RAM loc for quotient
RMND_L      EQU   0x40        ;the least significant digit loc
RMND_M      EQU   0x41        ;the middle significant digit loc
RMND_H      EQU   0x42        ;the most significant digit loc
MYDEN       EQU   D'10'       ;value for divide by 10

COUNTREG    EQU 0x10          ;fileReg loc for counter
CNTVAL      EQU d'3'          ;counter value
UNPBCD_ADDR EQU 0x30
ASCII_RESULT EQU 0x40
;-----------main program
      ORG   0
      SETF  TRSID             ;make PORTD input
      CALL  BIN_DEC_CON
      CALL  DEC_ASCII_CON
      BRA   $
;-----converting BIN(HEX) TO DEC (00-FF TO 000-255)
BIN_DEC_CON
      MOVFF PORTD,WREG        ;get the binary data from PORTD
      MOVWF NUME              ;load numerator
      MOVLW MYDEN             ;WREG = 10, the denominator
      CLRF  QU                ;clear quotient
D_1   INCF  QU                ;inc quotient for every subtraction
      SUBWF NUME              ;subtract WREG from NUME value
      BC    D_1               ;if positive go back
      ADDWF NUME              ;once too many, first digit
      DECF  QU                ;once too many for quotient
      MOVFF NUME,RMND_L       ;save the first digit
      MOVFF QU,NUME           ;repeat the process one more time
      CLRF  QU                ;clear QU
D_2   INCF  QU
      SUBWF NUME              ;subtract WREG from NUME value
      BC    D_2
      ADDWF NUME              ;once too many
      DECF  QU
      MOVFF NUME,RMND_M       ;2nd digit
      MOVFF QU,RMND_H         ;3rd digit
      RETURN
;----converting unpacked BCD digits to displayable ASCII digits
DEC_ASCII_CON
      MOVLW CNTVAL            ;WREG = 10
      MOVWF COUNTREG          ;load the counter, count = 10
      LFSR  0,UNPBCD_ADDR     ;load pointer FSR0
      LFSR  1,ASCII_RESULT    ;load pointer FSR1
B3    MOVF  POSTINC0, W       ;copy RAM to WREG, increment FSR0
      ADDLW 0x30              ;make it an ASCII
      MOVWF POSTINC1          ;copy WREG and increment FSR1
      DECF  COUNTREG,F        ;decrement counter
      BNZ   B3                ;loop until counter = zero
      RETURN
      END                     ;end of the program
```

SECTION 6.7: MACROS AND MODULES

80. 1) The programs using MACROs are more readable.
    2) MACROs do not use stacks and are not at the risk of stack overflow in nested form.
81. A MACRO - Because MACROs are replaced with their corresponding instructions in assembling process.
82. 1) Each module can be written, debugged and tested individually.
    2) The failure of one module does not stop the entire project.
    3) Parallel development of a large project.
83. extern
84. global
85. We write the main program as well as BIN_DEC_CON subroutine in one file (main.asm) and the DEC_ASCII_CON subroutine in another file (dec2ascii.asm)

```
;----- MAIN.ASM-CONVERTING BIN (HEX) TO ASCII
#include    P18F458
NUME        EQU   0x00
QU          EQU   0x20
RMND_L      EQU   0x30
RMND_M      EQU   0x31
RMND_H      EQU   0x32
MYDEN       EQU   D'10'

EXTERN      DEC_ASCII_CON

PGM   CODE
;------- main program
     ORG   0000
     CALL  BIN_DEC_CON
     CALL  DEC_ASCII_CON
     BRA   $
;------- converting BIN(HEX) (00-FF) to decimal (000-255)
BIN_DEC_CON
     MOVFF       PORTB,WREG         ;get the binary data from PORTB
     MOVWF       NUME               ;load numerator
     MOVLW       MYDEN              ;WREG = 10, the denominator
     CLRF        QU                 ;clear quotient
D_1  INCF        QU                 ;inc quotient for every subtraction
     SUBWF       NUME               ;subtract WREG from NUME value
     BC          D_1                ;if positive go back
     ADDWF       NUME               ;once too many, first digit
     DECF        QU                 ;once too many for quotient
     MOVFF       NUME,RMND_L        ;save the first digit
     MOVFF       QU,NUME            ;repeat the process one more time
     CLRF        QU                 ;clear QU
D_2  INCF        QU
     SUBWF       NUME               ;subtract WREG from NUME value
     BC          D_2
     ADDWF       NUME               ;once too many
     DECF        QU
     MOVFF       NUME,RMND_M        ;2nd digit
     MOVFF       QU,RMND_H          ;3rd digit
     RETURN
     END
```

In a separate file:

```
;----- DEC2ASCII.ASM-CONVERTING UNPACKED BCD DIGITS TO ASCII
#include    P18F458
CNTVAL      EQU   4
CNTVAL1     EQU   5

      GLOBAL      DEC_ASCII_CON
PGM   CODE

DEC_ASCII_CON
      MOVLW       CNTVAL              ;WREG = 10
      MOVWF       COUNTREG            ;load the counter, count = 10
      LFSR        0,UNPBCD_ADDR       ;load pointer FSR0
      LFSR        1,ASCII_RESULT      ;load pointer FSR1
B3    MOVF        POSTINC0, W         ;copy RAM to WREG, increment FSR0
      ADDLW       0x30                ;make it an ASCII
      MOVWF       POSTINC1            ;copy WREG and increment FSR1
      DECF        COUNTREG,F          ;decrement counter
      BNZ         B3                  ;loop until counter = zero
      RETURN
      END                             ;end of the program
```

CHAPTER 7: PIC PROGRAMMING IN C

SECTION 7.1: DATA TYPES AND TIME DELAYS IN C

1. a) signed char b) unsigned char c) unsigned int
d) unsigned char e) unsigned char f) unsigned char
g) unsigned int h) unsigned char i) unsigned char

2. a) 0E H b) 18 H c) 41 H
d) 07 H e) 20 H f) 45 H
g) FF H h) 0F H

3. a) Crystal frequency
b) The way the C compiler compiles (depends on the compiler type)
Note: We assume the PIC18 family has 4 clock periods / machine cycle.

4. Crystal frequency (also choice over compiler should be taken into consideration)
5. No, that is an internal factor which is not under programmer's control. It is determined and designed by the IC designer.
6. Because in the compilation of the C code into assembly, different compilers use different methods based on their design and optimization which results in different hex file sizes.

SECTION 7.2: I/O PROGRAMMING IN C

7. The former refers to pin RB4 (pin no.37), but the latter is an internal signal which determines whether RB4 is input or output.
8. The following program toggles all bits of PORTB in 200msec intervals.

```
#include <P18F458.h>
      void MSDelay (unsigned int);
      void main (void){
            TRISB = 0;
            PORTB = 0x55;
            while (1){
                  MSDelay (200);
                  PORTB = ~PORTB;
            }
      }

      void MSDelay (unsigned int itime){
            unsigned int I; unsigned char j;

            for ( i = 0 ; i < itime ; i++ )
                  for ( j = 0 ; j < 165 ; j++ );
      }
```

9.

```
#include <P18F458.h>
      void MSDelay (unsigned int);
      void main (void){
            TRISBbits.TRISB1 = 0;
            TRISBbits.TRISB7 = 0;
            PORTBbits.RB1 = 1;
            PORTBbits.RB7 = 1;
            while (1){
                  MSDelay (200);
                  PORTB = PORTB ^ 0x82;
            }
      }

      void MSDelay (unsigned int itime){
            unsigned int i; unsigned char j;

            for ( i = 0 ; i < itime ; i++ )
                  for ( j = 0 ; j < 165 ; j++ );
      }
```

10.

```
void MyDelay (void){
      unsigned char i,j;
      for (  i = 0 ; i < 100 ; i++ )
            for ( j = 0 ; j < 165 ; j++ );
}
```

11.

```
      #include <P18F458.h>
      #define  pin   PORTBbits.RB0
      void MyDelay ( void );

      void main ( void ){
            TRISBbits.TRISB0 = 0;
            pin = 1;
            while ( 1 ){
                  pin = ~pin;
                  MyDelay ();
            }
      }
      void MyDelay ( void ){
            unsigned char i,j;
            for ( i = 0 ; i < 200 ; i++ )
                  for ( j = 0 ; j < 165 ; j++ );
      }
```

12.

```
#include <P18F458.h>
void MyDelay ( void );

void main ( void ){
      unsigned char x;
      TRISB = 0;
      while ( 1 ){
            for (x = 0; x <= 99; x++){
                  PORTB = x;
                  MyDelay();
            }
      }
}
void MyDelay ( void ){
      unsigned char i,j;
      for ( i = 0 ; i < 200 ; i++ )
            for ( j = 0 ; j < 165 ; j++ );
}
```

SECTION 7.3: LOGIC OPERATIONS IN C

13. a) PORTB = 40H b) PORTB = 50H c) PORTB = 86H
d) PORTC = 90H e) PORTC = 60H f) PORTC = F0H
g) PORTC = F0H h) PORTC = F9H i) PORTC = 1EH
j) PORTC = 5AH

14. a) PORTB = 64H b) PORTB = 7BH c) PORTC = 3FH
d) PORTC = 58H e) PORTC = D7H f) PORTD = 04H
g) PORTB = 37H

15. a) PORTB = 19H b) PORTC = E4H c) PORTB = 1AH
d) PORTB = 9CH

16.

```
#include <P18F458.h>
unsigned char swap(unsigned char);
void main (void){
      TRISB = 0;
      unsigned char myNum = 0x95;
      PORTB = swap(myNum);
      while (1);
}
unsigned char swap (unsigned char charNum){
      unsigned char n;
      n = charNum & 0x0F;
      n = n << 4;
      charNum = charNum >> 4;
      n = n | charNum;
      return n;
}
```

17. The following program reads the present byte on PORTB port continuously and shows the number of zeros on PORTC.

```
#include <P18F458.h>
unsigned char zeroCount (unsigned char);

void main (void){
      unsigned char myNum;
      TRISB = 255;
      TRISC = 0;
      while (1){
            myNum = PORTB;
            PORTC = zeroCount(myNum);
      }
}

unsigned char zeroCount (unsigned char charNum){
      unsigned char j , k = 0;
      for (j = 0 ; j < 8 ; j++ ){
            if (( charNum & 0x01 ) == 0 )
                  k++;
            charNum = charNum >> 1;
      }
      return k;
}
```

18.

```
#include <P18F458.h>

void main  void){
      TRISB = 0;
      unsigned char pattern = 0x0C;
      while ( 1 ){
            PORTB = pattern;
            pattern = pattern >> 1;
            if ( pattern == 4 )
                  pattern = 0x0C;
            if ( pattern == 1 )
                  pattern = 0x09;
      }
}
```

SECTION 7.4: DATA CONVERSION PROGRAMS IN C

19. The following program converts the given bytes into two ASCII bytes and sends these two ASCII bytes into ports PORTB and PORTC.

```
#include <P18F458.h>

unsigned char bcdNum[4] = { 0x76 , 0x87 , 0x98 , 0x43 };
void BCDtoASCII (unsigned char);

void main (void){
      unsigned char j;
      TRISB = 0;
      TRISC = 0;
      for ( j = 0 ; j < 4 ; j++ )
            BCDtoASCII (bcdNum[j]);
      while(1);
}

void BCDtoASCII (unsigned char charNum){
      unsigned char Temp;
      Temp = charNum & 0xF0;
      charNum = charNum & 0x0F;
      PORTB = charNum | 0x30;
      Temp = Temp >> 4;
      PORTC = Temp | 0x30;
}
```

20. The following program converts the string into packed BCD format and shows high-byte to PORTB and low-byte to PORTC.

```
#include <P18F458.h>

char str[] = “8767”;

void main ( void ){
      TRISC = 0;
      TRISB = 0;
      unsigned char lowByte , highByte;
      highByte = str[0] & 0x0F;
      highByte = highByte << 4;
      highByte = highByte | (str[1] & 0x0F);
      PORTB    = highByte;
      lowByte  = str[2] & 0x0F;
      lowByte  = lowByte << 4;
      lowByte  = lowByte | (str[3] & 0x0F);
      PORTC    = lowByte;
}
```

21. The following program gets an 8-bit data from PORTB and sends its ASCII equivalent to PORTD-PORTC if it is less than 100.

```
#include <P18F458.h>

void main (void){
      TRISB = 255;
      TRISC = 0;
      TRICD = 0;
      unsigned char num,D3,D2,D1;
      // num will be converted to D3-D2-D1
      while (1){
            num = PORTB;
            D1  = ( num % 10) | 0x30; // LSD
            num = num / 10;
            D2  = ( num % 10 ) | 0x30;
            D3  = num / 10; // MSD
            if ( D3 == 0 ){   // 0 <= num <= 99
                  PORTC = D1;
                  PORTD = D2;
            }
      }
}
```

If PORTB = 10001001b, it represents decimal number of 137 which cannot be a packed BCD number stored in just one byte. So, in this case the above program will not send any data bytes to PORTC and PORTD.

SECTION 7.6: PROGRAM ROM ALLOCATION IN C18

22. a) data RAM   b) code ROM   c) code ROM   d) code ROM

23. False

24. 1) Saving data RAM space (this type of data usually needs a lot of space).
2) They are usually fixed. We use ROM to save fixed data and RAM to store variable data.

25. 1) Much more storage space
2) Not occupying RAM space which may be needed for storing some essential data

26. 1) Data stored in ROM space can not be changed unless with re-programming.
2) The more code space we use for data, the less is left for instructions.

27.

```
#include <P18F458.h>
rom const unsigned char myName[] = “Javad Rasti”;
void main ( void ){
      unsigned char z = 0;
      TRISC = 0;
      while ( myName[z] != ‘\0’ )
            PORTC = myName[z++];
}
```

28. When "near" qualifier is used, the corresponding data will be stored within the first 64KB of ROM. Using "far" qualifier lets the data be stored in the whole 2MB space ROM.

29. "#pragma code" is used to place the code at a specific address location of program ROM. "#pragma romdata" is used to place the data at a specific location of program ROM.

30. .

```
#include <P18F458.h>

#pragma romdata first_name = 0x220
#pragma romdata last_name = 0x200

rom const unsigned char first_name[] = "Javad";
rom const unsigned char last_name[] = "Rasti";

void main ( void ){
      unsigned char z = 0;
      TRISC = 0;
      while ( first_name[z] != ‘\0’ )
            PORTC = first_name[z++];
      z = 0;
      while ( last_name[z] != ‘\0’ )
            PORTC = last_name[z++];

}
```

31. a) 32K b) 32K c) 128K

32. The more code space we use for data, the less is left for instructions.

SECTION 7.7: DATA RAM ALLOCATION IN C

33. a) data RAM b) data RAM c) data RAM d) data RAM
e) code ROM

34. a) 1536 bytes b) 1536 bytes c) 3936 bytes

35. All the PIC18 products have the bank size of 256 bytes. If we want to create a portable program which can run on all PIC18 products, we must limit the array size to fit into a single bank.

36. 1) Saving data RAM space (this type of data usually needs a lot of space).
2) They are usually fixed. We use ROM to save fixed data and RAM to store variable data.

37. There will be a little space left for our program variables. ROM is more appropriate to store fixed data.

38. They can be changed easily during system operation.

39. "#pragma idata" is used to declare initialized variables at a specific location of data RAM, while "#pragma udata" is used for un-initialized data.

40.

```
#include <P18F458.h>

unsigned char myName[] = “Javad Rasti”;

void main ( void ){
      unsigned char z = 0;
      TRISC = 0;
      while ( myName[z] != ‘\0’ )
            PORTC = myName[z++];
}
```

41. Using "overlay" lets the compiler use a common location to store two or more variables (if they are not active at the same time). This method saves memory.

42. True

CHAPTER 8: PIC18F HARDWARE CONNECTION AND ROM LOADERS

SECTION 8.1: PIC18F458/452 PIN CONNECTION

1. 40
2. Pins 11 and 32 for V_{dd} and pins 12 and 31 for GND.
3. 33 I/O pins
4. 13 and 14
5. 40 MHz
6. Pin number 1
7. Master Clear
8. High, Low
9. 0000
10. 00
11. 00
12. 0xFF
13. 2
14. 2
15. OSC2
16. Input
17. Input
18. 7 pins: pins 2-7 and pin 14
19. 8 pins: pins 33-40
20. 8 pins: pins 15-18 and pins 23-26
21. 8 pins: pins 19-22 and pins 27-30
22. Input
23. PORTE
24. RB4

SECTION 8.2: PIC18 CONFIGURATION REGISTERS

25. True
26. True
27. False
28. False
29. True
30. False
31. 0x300001, 0x300002, 0x300003, 0x300006
32. 8
33. CONFIG1H
34. CONFIG2L
35. CONFIG2H
36. If the V_{dd} falls below 4.2 (v), the CPU will go into the reset state and stops execution of programs without losing any data in registers. When V_{dd} rises above 4.2(v), the CPU will come out of reset and continue the program execution.

37. CONFIG OSC = HS, OSCS = OFF, BORV = 42, PWRT = ON, WDT = OFF
    CONFIG DEBUG = OFF, LVP = OFF, STVR = OFF
38. LP
39. CONFIG1H
40. (a) 12 MHz / 4 = 3 MHz and the instruction cycle time = 1/ (3 MHz) = 333 ns
    (b) 20 MHz / 4 = 5 MHz and the instruction cycle time = 1/ (5 MHz) = 200 ns
    (c) 25 MHz / 4 = 6.25 MHz and the instruction cycle time = 1/ (6.25 MHz) = 160 ns
    (d) 30 MHz / 4 = 7.5 MHz and the instruction cycle time = 1/ (7.5 MHz) = 133.33 ns

SECTION 8.3: EXPLAINING THE INTEL HEX FILE FOR PIC18

41. True
42. True
43. False
44. True
45. False- ROM size is limited to 4 gigabytes.
46. (1) The colon starts the line.
    (2) The first byte after the colon (10h) indicates that there will be 16 data bytes in this line.
    (3) The next two bytes (0000h) indicate the ROM address that the first data byte must be burned into.
    (4) The next byte (00h) shows that this is not the last line of the program.
    (5) The next sixteen bytes are the op-codes and their operand data.
    (6) The last byte is the check-sum byte of the previous bytes in the line.
47. Calculation of the check-sum byte:
    10h + 00h + 00h + 00h + 93h + 6Ah + 55h + 0Eh + 81h + 6Eh +81h + 1Eh + 07h + ECh + 00h + F0h + FCh + D7h + 02h + 0Eh = 6C4h
    Dropping the carries: C4h
    2’s complement of C4h: 3Ch (which is the last byte of line 1)
    Verification of the check-sum byte: 10h + 00h + ... + 02h + 0Eh + 3Ch = 700h
    Dropping the carries: 00h. So 3Ch is the correct check-sum byte of the line.
48. Calculation of the check-sum byte:
    0Eh + 00h + 00h + 00h + 93h + 6Ah + 55h + 0Eh + 81h + 6Eh + 81h + 1Eh + 78h + ECh +94h + F0h + FCh + D7h = 7B7h
    Dropping the carries: B7h
    2’s complement of B7h: 49h (which is the last byte of line 2)
    Verification of the check-sum byte: 0Eh + 00h + ... + FCh + D7h + 49h = 800h
    Dropping the carries: 00h. So 49h is the correct check-sum byte of the line.
49. INTHX8M is used for codes with size below 64 Kbytes. INTHX32 is used for codes with size up to 4 Gigabytes. They also differ in TT field of each line.
50. In line 4, after the colon (:), we have 10H (which is 16 in decimal) as the number of bytes in this line. The AAAA = 28F0 is the lower 16-bit address where information will be burned. Next, 00 means that this is not the last line of the record. Then the data, which is 16 bytes, is as follows: 020E076EFA0E086EFA0E096E00000000. Finally, the last byte, 56, is the checksum byte.

SECTION 8.4: PIC18 TRAINER DESIGN AND LOADING OPTIONS

51. False
52. True
53. (b)
54.

```
SETF  TRISB ;Define PORTB as input
CLRF  TRISC ;Define PORTC as output
CLRF  TRISD ;Define PORTD as output

MOVFF PORTB, PORTC
MOVFF PORTB, PORTD
```

55.

```
SETF  TRISD ;Define PORTD as input
CLRF  TRISB ;Define PORTB as output
CLRF  TRISC ;Define PORTC as output

MOVFF PORTD, PORTB
MOVFF PORTD, PORTC
```

56. Pin 40 (RB7) is PGD and pin 39 (RB6) is PGC.
57. 0000. It implies we must place our codes starting at address 0000.
58. (a)

```
      CLRF      TRISB
BACK  MOVLW     55H
      MOVWF     PORTB
      CALL      DELAY
      MOVLW     0AAH
      MOVWF     PORTB
      CALL      DELAY
      GOTO      BACK
```

(b)

```
      CLRF     TRISB
      MOVLW    55H
      MOVWF    PORTB
BACK  CALL     DELAY
      COMF     PORTB
      BRA      BACK
```

59. 7FFFh
60. 1FFFFh
61. 7FFFh

CHAPTER 9: PIC18 TIMER PROGRAMMING IN ASSEMBLY AND C

SECTION 9.1: PROGRAMMING TIMERS 0 AND 1

1. 4 timers
2. Timer0 is 16 –bit and are accessed as low and high byte
3. Timer1 is 16 –bit and are accessed as low and high byte
4. 256
5. 8
6. 8
7. Used for control of Timer0
8. True
9. 0x01
10. frequency = 10 MHZ / 4 = 2.5 MHZ period = 1 / (2.5 MHZ) = 0.4 μs
    b) frequency = 20 MHZ / 4 = 5 MHZ period = 1 / (5 MHZ) = 0.2 μs
    c) frequency = 24 MHZ / 4 = 6 MHZ period = 1 / (6 MHZ) = 0.166μs
    d) frequency = 30 MHZ / 4 = 7.5 MHZ period = 1 / (7.5MHZ) = 0.133μs

11. a) INTCON b) PIR1
12. a) FFFFH to 0000H = 65535 to 0 b) FFH to 00H = 255 to 0
13. a) When the timer reaches its maximum value of FFFFH and it rolls over to 0000
    b) When the timer reaches its maximum value of FFH and it rolls over to 00
14. True
15. True
16. 10 MHz / 4 = 2.5 MHz, 1 / 2.5 MHz = 0.4 μs,
    2 ms / 0.4 μs = 5000, 65536 – 5000 = 60536 = 0xEC78,
    TMR0H = 0xEC TMR0L = 0x78
17. 10 MHz / 4 = 2.5 MHz, 1 / 2.5 MHz = 0.4 μs,
    5 ms / 0.4 μs = 12500, 12500 / 256 = 49, 65536 – 49 = 65487 = 0xFFCF,
    TMR0H = 0xFF TMR0L = 0xCF
18. TMR1H = 0xFC TMR1L = 0xF2
19. TMR1H = 0xFE TMR1L = 0x0C
20. TMR1H = 0xFB TMR1L = 0x1E
21. 10 MHz / 4 = 2.5 MHz, 2.5 MHz / 256 = 9765.625 Hz,
    1 / 9765.625 Hz = 102.4 μs, 102.4 μs * 65536 = 6.7108864 s,
    6.7108864 s * 2 = 13.4217728 s, 1 / 13.4217728 = 0.0745058 Hz
    So the lowest frequency is equal to 0.0745058 HZ.
22. 10 MHz / 4 = 2.5 MHz, 1 / 2.5 MHz = 0.4 μs,
    Solution a) if we include 7 overhead instructions in the loop we have the following:
    0.4 μs * 7 = 2.8 μs, T = 2.8 μs * 2 = 5.6 μs, F = 1 / T = 1 / 5.6 μs 178. kHz
    Solution b) if we ignore the overhead instructions in the loop we have the following:
    Since Timer0 must have FFFFH to get the shortest period, T = 0.4 μs * 2 = 0.8 μs,
    F = 1 / T = 1 / 0.8 μs = 1.25 MHZ
23. 19.07 HZ
    208.5 kHz (same as 22 with 6 overhead instructions)
24. F1H, F2H, F3H, …, FFH, 00H = 16 states

25. Program:

```
#include <p18F458.inc>
          ORG       0x00
          GOTO START
          ORG       0x30
START     BCF       TRISB,4
          MOVLW     0x08
          MOVWF     T0CON
HERE      MOVLW     0xFB
          MOVWF     TMR0H
          MOVLW     0x1E
          MOVWF     TMR0L
          BCF       INTCON,TMR0IF
          CALL      DELAY
          BTG       PORTB,RB4
          BRA       HERE
DELAY     BSF       T0CON,TMR0ON
AGAIN     BTFSS     INTCON,TMR0IF
          BRA       AGAIN
          BCF       T0CON,TMR0ON
          RETURN
          END
```

26. Program:

```
#include <p18F458.inc>
          ORG       0x00
          GOTO      START
          ORG       0x30
START     BCF       TRISB,5
          MOVLW     0x30
          MOVWF     T1CON
HERE      MOVLW     0xFF
          MOVWF     TMR1H
          MOVLW     0xCB
          MOVWF     TMR1L
          BCF       PIR1,TMR1IF
          CALL      DELAY
          BTG       PORTB,RB5
          BRA       HERE
DELAY     BSF       T1CON,TMR1ON
AGAIN     BTFSS     PIR1,TMR1IF
          BRA       AGAIN
          BCF       PIR1,TMR1ON
          RETURN
          END
```

27. Timer0 is 8 or 16 bit timer/counter. It has an interrupt on overflow from FFH to 00H in 8 bit and FFFFH to 0000H in 16 bit. Timer1 is only 16 bit timer/counter and it has interrupt on overflow from FFFFH to 0000H. The maximum prescaler supported by Timer0 is 256. The maximum prescaler supported by Timer1 is 8. Timer1 resets from CCP module special event register.

28. a) 0xF4
b) 0xEA
c) 0xDE
d) 0xA4
e) 0x88
f) 0x98

SECTION 9.2: COUNTER PROGRAMMING

29. We must set T0CS bit of T0CON to high

30. Yes

31. RA4/T0CKI pin

32. RC0/T1CKI pin

33. Program:

```
#include <p18F458.inc>
          ORG       0x00
          GOTO      START
          ORG       0x30
START     BSF       TRISC,0
          CLRF      TRISB
          CLRF      TRISD
          MOVLW     0x02
          MOVWF     T1CON
HERE      MOVLW     0x4E
          MOVWF     TMR1H
          MOVLW     0x20
          MOVWF     TMR1L
          BCF       PIR1,TMR1IF
          BSF       T1CON,TMR1ON
AGAIN     MOVFF     TMR1H,PORTD
          MOVFF     TMR1L,PORTB
          BTFSS     PIR1,TMR1IF
          BRA       AGAIN
          BCF       T1CON,TMR1ON
          GOTO      HERE
          END
```

34. Program:

```
#include <p18F458.inc>
          ORG       0x00
          GOTO      START
          ORG       0x30
```

```
START     BSF       TRISA,RA4
          CLRF      TRISB
          MOVLW     0x68
          MOVWF     T0CON
HERE
          MOVLW     0x14
          MOVWF     TMR0L
          BCF       INTCON,TMR0IF
          BSF       T0CON,TMR0ON
AGAIN
          MOVFF     TMR0L,PORTB
          BTFSS     INTCON,TMR0IF
          BRA       AGAIN
          BCF       T0CON,TMR0ON
          GOTO      HERE
          END
```

35. 8
36. False

SECTION 9.3: PROGRAMMING TIMER 0 AND 1 IN C

37.

C Program:

```
#include <p18f458.h>
void ToDelay (void);
void main(void)
     {
     TRISBbits.TRISB4=0;
     T0CON = 0x08;
     while(1)
          {
          PORTBbits.RB4 = 1;
          ToDelay();
          PORTBbits.RB4 = 0;
          ToDelay();
          }
     }
void ToDelay()
     {
     TMR0H = 0xFB;
     TMR0L = 0x1E;
     T0CONbits.TMR0ON = 1;
     while (INTCONbits.TMR0IF == 0);
     T0CONbits.TMR0ON = 0;
     INTCONbits.TMR0IF = 0;
     }
```

38. C Program:

```
#include <p18f458.h>
void ToDelay(void);
void main(void)
     {
     TRISBbits.TRISB4=0;
     T1CON = 0x0;
     while(1)
          {
          PORTBbits.RB4 = 1;
          ToDelay();
          PORTBbits.RB4 = 0;
          ToDelay();
          }
     }

void ToDelay()
     {
     TMR1H = 0xFB;
     TMR1L = 0x1E;
     T1CONbits.TMR1ON = 1;
     while (PIR1bits.TMR1IF == 0);
     T1CONbits.TMR1ON = 0;
     PIR1bits.TMR1IF = 0;
     }
```

39. C Program:

```
#include <p18f458.h>
void ToDelay(void);
void main(void)
     {
     TRISBbits.TRISB4=0;
     T0CON = 0x08;
     while(1)
          {
          PORTBbits.RB4 = 1;
          ToDelay();
          PORTBbits.RB4 = 0;
          ToDelay();
          }
     }

void ToDelay()
     {
     TMR0H = 0xFE;
     TMR0L = 0xC7;
     T0CONbits.TMR0ON = 1;
     while (INTCONbits.TMR0IF == 0);
     T0CONbits.TMR0ON = 0;
     INTCONbits.TMR0IF = 0;
     }
```

40. C Program:

```
#include <p18f458.h>
void ToDelay(void);
void main(void)
     {
     TRISBbits.TRISB4=0;
     T1CON = 0x0;
     while(1)
          {
          PORTBbits.RB4 = 1;
          ToDelay();
          PORTBbits.RB4 = 0;
          ToDelay();
          }
     }

void ToDelay()
     {
     TMR1H = 0xFE;
     TMR1L = 0xCB;
     T1CONbits.TMR1ON = 1;
     while (PIR1bits.TMR1IF == 0);
     T1CONbits.TMR1ON = 0;
     PIR1bits.TMR1IF = 0;
     }
```

41. C Program:

```
#include <p18f458.h>

void main(void)
     {
     TRISCbits.TRISC4=1;
     TRISB = 0x0;
     TRISD = 0x00;
     T1CON = 0x02;

     while(1)
          {
          TMR1H = 0x4E;
          TMR1L = 0x20;
          T1CONbits.TMR1ON = 1;
          do
               {
               PORTD = TMR1H;
               PORTB = TMR1L;
               } while(PIR1bits.TMR1IF == 0);
          T1CONbits.TMR1ON = 0;
          PIR1bits.TMR1IF = 0;
          }
}
```

42. C Program:

```
#include <p18f458.h>

void main(void)
     {
     TRISAbits.TRISA4=1;

     TRISD = 0x00;
     T0CON = 0x68;

     while(1)
          {
          TMR1L = 0x14;
          T0CONbits.TMR0ON = 1;
          do
               {
               PORTD = TMR0L;
               } while(INTCONbits.TMR0IF == 0);
          T0CONbits.TMR0ON = 0;
          INTCONbits.TMR0IF = 0;
          }
```

SECTION 9.4: PROGRAMMING TIMERS 2 AND 3

43. a) 1:16   b) 1:8
44. FFFFH to 0000H
45. a) When TMR2 = PR2   b) When TMR3H:TMR3L rolls over from FFFFH to 0000H
46. True
47. False
48. TMR3H = 0xEC   TMR3L = 0x78
49. TMR3H = 0xF9   TMR3L = 0xE5
50. C Program:

```
#include <p18f458.h>

void main(void)
     {
     TRISCbits.TRISC0 = 1;
```

```
        TRISB = 0;
        TRISD = 0;
        T3CON = 0x02;
        TMR3H = 0x4E;
        TMR3L = 0x20;
        while (1)
             {
             T3CONbits.TMR3ON = 1;
             do
                  {
                  PORTB = TMR3L;
                  PORTD = TMR3H;
                  }while (PIR2bits.TMR3IF == 0);
             T3CONbits.TMR3ON = 0;
             PIR2bits.TMR3IF = 0;
             }
        }
```

51. Assembly Program:

```
        BCF       TRISB,3
        BCF       PORTB,3
        CLRF      T2CON
        CLRF      TMR2
        MOVLW     0xC8           ;200
        MOVWF     PR2
        BCF       PIR1,TMR2IF
        BSF       T2CON,TMR2ON
   START:
        BTFSS     PIR1,TMR2IF
        GOTO      START
        BTG       PORTB,3
        GOTO      START
```

52. C Program:

```
   #include <p18f458.h>
   void ToDelay(void);
   void main(void)
        {
        TRISBbits.TRISB4=0;
        T3CON = 0x0;
        while(1)
             {
             PORTBbits.RB4 = 1;
             ToDelay();
             PORTBbits.RB4 = 0;
             ToDelay();
             }
        }
```

```
void ToDelay()
{
     TMR3H = 0xFE;
     TMR3L = 0x5F;
     PIR2bits.TMR3IF = 0;
     T3CONbits.TMR3ON = 1;
     while (PIR2bits.TMR3IF == 0);
     T3CONbits.TMR3ON = 0;
     PIR2bits.TMR3IF = 0;
}
```

53. C Program:

```
#include <p18f458.h>
void main(void)
{
     TRISBbits.TRISB3 = 0;
     PORTBbits.RB3 = 0;

     T2CON = 0x00;
     TMR2 = 0x00;
     PR2 = 0xC8;
     PIR1bits.TMR2IF = 0;
     T2CONbits.TMR2ON = 1;
     do
          {
          while (PIR1bits.TMR2IF == 0);
          PORTBbits.RB3 = ~PORTBbits.RB3;
          }while(1);
}
```

54. C Program:

```
#include <p18f458.h>
void ToDelay(void);
void main(void)
     {
     TRISBbits.TRISB4=0;
     T3CON = 0x0;
     while(1)
          {
          PORTBbits.RB4 = 1;
          ToDelay();
          PORTBbits.RB4 = 0;
          ToDelay();
          }
     }
```

```
void ToDelay()
{
     TMR3H = 0xFB;
     TMR3L = 0x1E;
     PIR2bits.TMR3IF = 0;
     T3CONbits.TMR3ON = 1;
     while (PIR2bits.TMR3IF == 0);
     T3CONbits.TMR3ON = 0;
     PIR2bits.TMR3IF = 0;
}
```

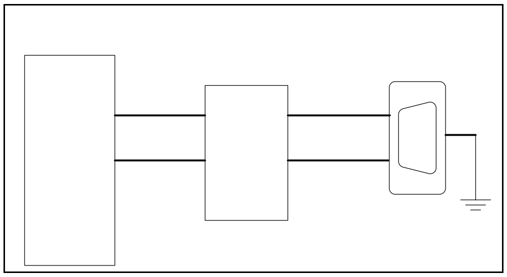
CHAPTER 10: PIC18 SERIAL PORT PROGRAMMING IN ASSEMBLY AND C

SECTION 10.1: BASICS OF SERIAL COMMUNICATION

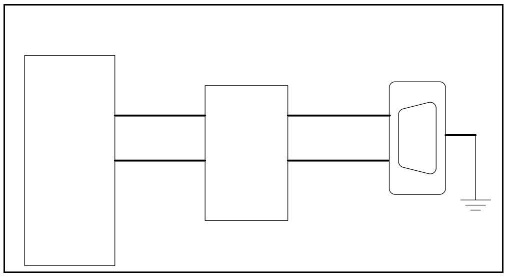
1. Parallel
2. False
3. 1010110100 ---> this start bit goes first
4. Mark
5. False. It can be 1 or 2 but we always need stop bit.
6. 7 bit data + 1 stop bit + 1 start bit = 9 bits
   2/9 x 100 = 22%
7. False
8. It makes the RS232 standard compatible with TTL logic level.
9. False
10. 9 pins, since the rest are not involved in data transfer.
11. False
12. 3: TxD RxD Gnd
13. DTE -to- DTE
14. DCD, RxD, TxD, DTR, Gnd, DSR, RTS, CTS, RI
15. 4,000,000 bits (200 x 80 x 25 x 10)
16. 416.6 seconds

SECTION 10.2: PIC18 CONNECTION TO RS232

17. 16
18. VCC = 16, GND = 15
19. 20
20. VCC = 7, GND =6
21. No
22. The MAX233 does not require an external capacitor, but it is more expensive.
23. 2
24. 2
25.



26.



27. It does not need the four capacitors that the MAX232 must have.
28. pin 25 – TxD (RC6), pin 26 – RxD (RC7)

SECTION 10.3: PIC18 SERIAL PORT PROGRAMMING IN ASSEMBLY

29. (a), (c), (e), (f)
30. None. It has its own timer
31. Bit D2 (BRGH)
32. To send a byte serially it must be placed in the TXREG register.
33. 8
34. It controls the framing of the data for serial ports.
35. 8
36. First we have 10 MHz/4 = 2.5 MHz and 2.5/16 = 156,250.
    Now each is calculated by using X = (156,250/ Desired Baud Rate) –1 formula..
    (a) 156,250/9600 - 1 = 15 = F in hex
    (b) 156,250/4800 - 1 = 32 = 20 in hex
    (c) 156,250/1200 - 1 = 129 = 81 in hex
37. 9600
38.

```
  MOVLW   B'00100000'
  MOVWF   TXSTA
  MOVLW   0x81             ;1200 bps (Fosc / (64 * Speed) - 1)
  MOVWF   SPBRG            ;write to reg
  BCF     TRISC, TX        ;make TX pin of PORTC an output pin
  BSF     RCSTA, SPEN      ;Enable Serial port of PIC18
OVER BTFSS PIR1, TXIF      ;wait until the last bit is gone
  BRA     OVER             ;stay in loop
  MOVLW   A'Z'             ;ASCII letter “Z” to be transferred
  MOVWF   TXREG            ;load the value to be transferred
  BRA     OVER             ;keep sending letter ‘G’
  END
```

39.

```
   BCF TRISC, TX      ;make TX pin of PORTC an output pin
   BSF RCSTA, SPEN    ;Enable the entire Serial port of PIC18
```

```
  MOVLW 0x20        ;transmit  at Low Baud rate
  MOVWF TXSTA       ;write to reg
  MOVLW D'2'        ;57600 bps (Fosc / (64 * Speed) - 1)
  MOVWF SPBRG       ;write to reg
OVER
  MOVLW upper(MESSAGE)
  MOVWF TBLPTRU
  MOVLW high(MESSAGE)
  MOVWF TBLPTRH
  MOVLW low(MESSAGE)
  MOVWF TBLPTRL
NEXT TBLRD*+        ;read the characters
  MOVF TABLAT,W     ;place it in WREG
  BZ  OVER          ;if end of line, start over
  CALL SENDCOM      ;send char to serial port
  BRA  NEXT         ;repeat for the next character
;-------------
SENDCOM
S1 BTFSS PIR1, TXIF ;wait unil the last bit is gone
  BRA S1            ;stay in loop
  MOVWF   TXREG     ;load the value to be transmittd
  RETURN            ;return to caller
;------------------
MESSAGE:DB     "The Earth is but One Country",0
          END
```

40. TXIF flag is raised when a byte of data is transmitted and TXREG register is empty. It is cleared when a new byte is written to TXREG.
41. RCIF flag is raised when a byte of data is received and it is sitting in RCREG register. It is cleared when the content of RCREG is copied into another register.
42. PIR1, Yes
43. SPEN bit enables or disables serial data transmission/reception
44. BCF RCSTA,SREN
45. It belongs to the TXSTA register. Upon power on rest, BRGH = 0. By making BRGH = 1, it allows us to quadruple the serial data transfer rate with the same crystal.
46. Low
47. First we have 16 MHz/4 = 4 MHz and 4 MHz/16 = 250,000.
Now each is calculated by using X = (250,000/ Desired Baud Rate) –1
(a) 250,000/9600 - 1 = 26.04 - 1 = 25 = 19 in hex
(b) 250,000/19200 - 1 = 13.02 – 1 = 12 = 0C in hex
(c) 250,000/38400 - 1 = 6.51 – 1 = 5.5 = 5
(d) 250,000/57600 - 1 = 4.34 - 1 = 3.34 = 3
48. First we have 16 MHz/4 = 4 MHz and 4 MHz/4 = 1MHz.
Now each is calculated by using X = (1MHz/ Desired Baud Rate) –1 :
(a) 1MHz/9600 - 1 = 104.16 - 1 = 103 = 67 in hex.
(b) 1MHz/19200 - 1 = 52.08 – 1 = 51 = 33 in hex.

(c) 1MHz/38400 - 1 = 26.04 – 1 = 25 = 19 in hex.

(d) 1MHz/57600 - 1 = 17.36 - 1 = 16 = 10 in hex.

49. First we have 20 MHz/4 = 5 MHz and 5 MHz/16 = 312,500 Hz.

Now each is calculated by using X = (312500 Hz/ Desired Baud Rate) –1

(a) 312500Hz/9600 - 1 = 32.55 - 1 = 31 = 1F in hex.

(b) 312500Hz/19200 - 1 = 16.27 – 1 = 15 = F in hex.

(c) 312500Hz/38400 - 1 = 8.13 – 1 = 7 .

(d) 312500Hz/57600 - 1 = 5.42 - 1 = 4 = 4

50. First we have 20 MHz/4 = 5 MHz and 5 MHz/4 = 1.25 MHz.

Now each is calculated by using X = (1.25MHz/ Desired Baud Rate) –1

(a) 1.25MHz/9600 - 1 = 130.2 - 1 = 129 = 81 in hex.

(b) 1.25MHz/19200 - 1 = 65.1 – 1 = 64 = 40 in hex

(c) 1.25MHz/38400 - 1 = 32.55 – 1 = 31 = 1F in hex.

(d) 1.25MHz/57600 - 1 = 21.7 - 1 = 22 - 1 = 21 = 15 in hex.

51. (a) 250,000/9600 -1 = 26.04 - 1 = 25.04 and error rate is (25.04-25)/26 = 0.1 %

(b) 250,000/19200-1 = 13.02 – 1=12.02 and error rate is (12.02 – 12)/13 = 0.1 %

(c) 250,000/38400 - 1 = 6.51 – 1 = 5.5 and error rate is (5.5-5)/6 = 8%

(d) 250,000/57600 - 1 = 4.34 - 1 = 3.34 and error rate is (3.34 – 3) / 4 = .8.5%

52. (a) 1MHz/9600-1=104.16-1 = 103.16 and error rate is (103.16 – 103)/104 =0.15%

(b) 1MHz/19200 - 1 = 52.08 – 1 = 51.08 and error rate is (51.08- 51)/52 = 0.15%

(c) 1MHz/38400 -1 = 26.04 – 1 = 25.04 and error rate is (25.04 – 25)/26 = 0.15%

(d) 1MHz/57600 - 1 = 17.36 - 1 = 16.36 and error rate is (16.36- 16)/17 = 2%

SECTION 10.4: PIC18 SERIAL PROGRAMMING IN C

53.

```
void main(void)
  {
    TXSTA=0x20;          //choose low buad rate,8-BIT
    SPBRG=15;            //9600 baud rate/ XTAL=10Mhz
    TXSTAbits.TXEN=1;
    RCSTAbits.SPEN=1;
    while(1)
      {
        TXREG='Z';       //place value in buffer
        while(PIR1bits.TXIF==0);   // wait until all gone
      }
  }
```

54.

```
void main(void)
  {
    unsigned char z;
    unsigned char message[]="The earth is but one country \
                             and mankind is citizens";
    TXSTA=0x20;          //choose low buad rate,8-BIT
    SPBRG=2;             //57600 baud rate/ XTAL=10Mhz
    TXSTAbits.TXEN=1;
    RCSTAbits.SPEN=1;

     do
      {
        for(z=0;z<sizeof(message);z++)  //write name
        {
          while(PIR1bits.TXIF==0);      //wait for transmit
          TXREG=message[z];        //place char in buffer
        }
      }while(1);
  }
```

CHAPTER 11: INTERRUPT PROGRAMMING IN ASSEMBLY AND C

SECTION 11.1: PIC18 INTERRUPTS

1. interrupt
2. Timer, hardware, serial
3. 0x0008 and 0x0018
4. True
5. 0x0018
6. 0x0008
7. No – All interrupts go to the same location, either low or high.
8. No – All interrupts go to the same location, either low or high.
9. INTCON
10. To bypass the interrupt vector table.
11. GIE is set to 0 so the interrupts are disabled on power up.
12. BSF INTCON, INT0IE
13. BSF INTCON, TMR0IE
14. INTCON
15. 16 bytes (8 words)
16. Unlimited- Since there is no vector entries after it. This is unlike high priority.
17. 16 bytes (8 words)
18. False
19. BCF INTCON, GIE
20. BCF INTCON, INT0IE
21. False
22. 8 bytes (4 words)

SECTION 11.2: PROGRAMMING TIMER INTERRUPTS

23. False
24. No specific address assigned. It can be assigned to low or high priority.
25. BSF INTCON, TMR0IE
26. bit 5, BSF INTCON, TMR0IE
27. When TMR0H rolls over from FFH to 00H, then the interrupt flag is raised and it is directed to the high or low priority vector location to execute the ISR.
28. True
29. When TMR1H:TMR1L rolls from FFFFH to 0000H then Timer1 flag is raised and it is directed to low or high priority vector location to execute the ISR.
30. When TMR1H rolls over from FF to 00
31.

```
     ORG  0000H
     GOTO MAIN           ;bypass interrupt vector table
;—-on default all interrupts land at address 00008
     ORG  0008H               ;interrupt vector table
     BTFSS INTCON,TMR0IF ;Timer0 interrupt?
     RETFIE              ;No. Then return to main
     GOTO  T0_ISR        ;Yes. Then go Timer0 ISR
;—-main program for initialization and keeping CPU busy
     ORG  00100H         ;after vector table space
MAIN
     BCF  TRISB,7        ;RB7 as an output
     CLRF TRISD          ;make PORTD output
     SETF TRISC          ;make PORTC input
     MOVLW 0x07          ;Timer0,16-bit,
                         ;no prescale,internal clk
     MOVWF T0CON         ;load T0CON reg
     MOVLW 0xEC          ;TMR0H = ECH, the high byte
     MOVWF TMR0H         ;load Timer0 high byte
     MOVLW 0xED          ;TMR0L = EDH, the low byte
     MOVWF TMR0L         ;load Timer0 low byte
     BCF INTCON,TMR0IF   ;clear timer interruupt flag bit
     BSF T0CON,TMR0ON    ;start Timer0
     BSF INTCON,TMR0IE   ;enable Timer 0 interrupt
     BSF INTCON,GIE      ;enable interrupts globally
;--keeping CPU busy waiting for interrupt
OVER MOVFF PORTC,PORTD   ;send data from PORTC to PORTD
     BRA OVER            ;stay in this loop forever
;--------------------------ISR for Timer 0
T0_ISR
     ORG 200H
     MOVLW 0xEC          ;TMR0H = ECH, the high byte
     MOVWF TMR0H         ;load Timer0 high byte
     MOVLW 0xED          ;TMR0L = EDH, the low byte
     MOVWF TMR0L         ;load Timer0 low byte
     BTG  PORTB,7        ;toggle RB7
     BCF INTCON,TMR0IF   ;clear timer interrupt flag bit
EXIT RETFIE ;return from interrupt (See Example 11-2)
     END
```

32.

```
     ORG  0000H
     GOTO MAIN           ;bypass interrupt vector table
;—-on default all interrupts land at address 00008
     ORG  0008H          ;interrupt vector table
     BTFSS PIR1,TMR1IF   ;Timer1 interrupt?
     RETFIE              ;No. Then return to main
     GOTO  T1_ISR        ;Yes. Then go Timer1 ISR
;—-main program for initialization and keeping CPU busy
     ORG  00100H         ;after vector table space
MAIN
     BCF  TRISB,7        ;RB7 as an output
     CLRF TRISD          ;make PORTD output
     SETF TRISC          ;make PORTC input
     CLRF T1CON          ;Timer1,16-bit,
                         ;no prescale,internal clk
     MOVLW 0xFE          ;TMR1H = FEH, the high byte
     MOVWF TMR1H         ;load Timer0 high byte
     MOVLW 0x5F          ;TMR1L = 5FH, the low byte
     MOVWF TMR1L         ;load Timer1 low byte
     BSF PIE1,TMR1IE     ;enable Timer1 interrupt
     BCF PIR1,TMR1IF     ;clear timer interruupt flag bit
     BSF T1CON,TMR1ON    ;start Timer1
     BSF INTCON,PEIE     ;enable peripheral interrupts
     BSF INTCON,GIE      ;enable interrupts globally
;--keeping CPU busy waiting for interrupt
OVER MOVFF PORTC,PORTD   ;send data from PORTC to PORTD
     BRA OVER            ;stay in this loop forever
;-------------------------ISR for Timer 0
T1_ISR
     ORG 200H
     MOVLW 0xFE          ;TMR1H = FEH, the high byte
     MOVWF TMR1H         ;load Timer1 high byte
     MOVLW 0x5F          ;TMR1L = 5FH, the low byte
     MOVWF TMR1L         ;load Timer1 low byte
     BTG  PORTB,7        ;toggle RB7
     BCF PIR1,TMR1IF     ;clear timer interrupt flag bit
EXIT RETFIE
     END
```

SECTION 11.3: PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

33. False
34. 0x0008 if high priority and 0x0018 if low priority, Pins 33, 34, and 35
35. INTCON, “BSF INTCON, INT0IE”
36. INTCON3 BSF INTCON3,INT1IE
37. BSF INTCON, INT0IE
    BSF INTCON3,INT1IE
    BSF INTCON3,INT2IE
38. When the input signal goes from high to low the processor finishes executing the current instruction and jumps to ROM location 0x008 to execute the ISR.
39. False
40. When the input signal goes from high to low the processor finishes executing the current instruction and jumps to ROM location 0x008 to execute the ISR. The CPU makes GIE = 0, blocking another interrupt from the same source or any other source. At the end of the ISR the RETFIE instruction will make GIE = 1, allowing another interrupt to come from the same source.
41. INTCON
42. INTCON3

43. True
44. In order to allow the interrupt to come in, INT0IE must be set to high. It is only after INT0IE = 1, that INT0IF being high will cause the CPU to jump to a vector location. INT0IF can become high by external pulse or by an instruction. In some applications to simulate interrupts we can use an instruction to set INT0IF to cause an interrupt.
45. In order to allow the interrupt to come in, INT1IE must be set to high. It is only after INT1IE = 1, that INT1IF being high will cause the CPU to jump to a vector location. INT1IF can become high by external pulse or by an instruction. In some applications to simulate interrupts we can use an instruction to set INT1IF to cause an interrupt.
46. When the input signal goes from high to low the processor finishes executing the current instruction and jumps to ROM location 0x008 to execute the ISR. The CPU makes GIE = 0, blocking another interrupt from the same source or any other source. At the end of the ISR the RETFIE instruction will make GIE = 1, allowing another interrupt to come from the same source.
47. False
48. True
49. Falling is the transition from high to low, Rising is the transition from low to high
50. Clear the appropriate bit in INTCON2 register
51. True
52. INTCON2 register.

SECTION 11.4: PROGRAMMING THE SERIAL COMMUNICATION INTERRUPT AND

SECTION 11.5: PORTB-CHANGE INTERRUPT

53. True
54. 0x0008, 16 bytes
55. PIR1, BSF PIE1, TXIE
56. When TXIE = 1, upon transferring the last bit of the byte out of TXREG, TXIF is raised. When TXIF is raised the code gets interrupted and jumps to the ISR to get another byte to be written to the TXREG.
57. True
58. False, TXIF is cleared automatically by the CPU upon writing to TXREG.
59. RCIF is cleared automatically by the CPU upon reading a byte from RCREG.
60. The code executing is interrupted and the CPU jumps to the ISR to write another byte to TXREG.
61. The code executing is interrupted and the CPU jumps to the ISR to read the received byte from RCREG.
62.

```
     ORG  0000H
     GOTO MAIN           ;bypass interrupt vector table
;—-on default all interrupts go to to address 00008
     ORG  0008H          ;interrupt vector table
     BTFSC PIR1,RCIF     ;Is interrupt due to transmit?
     BRA RC_ISR          ;Yes. Then go to ISR
     RETFIE              ;No. Then return
     ORG 0040H
```

```
RC_ISR                   ;service routine for RCIF
     MOVFF RCREG,PORTD   ;get new value, clear RCIF
     RETFIE              ;then return to main
;—-the main program for initialization
     ORG  00100H
MAIN
     CLRF TRISD          ;make PORTD output
     BSF TRISB,4         ;make RB4 an input
     BCF TRISC,7         ;make RC7 an output
     MOVLW 0x90          ;enable receive and choose
                         ;low baud
     MOVWF RCSTA         ;write to reg
     MOVLW D'15'         ;9600 bps
                         ;(Fosc / (64 * Speed) - 1)
     MOVWF SPBRG         ;write to reg
     BSF TRISC, RX       ;make RX pin of PORTC an
                         ;input pin
     BSF RCSTA, SPEN     ;enable the serial port
     BSF PIE1,RCIE       ;enable TX interrupt
     BSF INTCON,PEIE     ;enable peripheral interrupts
     BSF INTCON,GIE      ;enable interrupts globally
AGAIN
     BTFSS PORTB,4       ;bit test RB4
     GOTO OVER
     BSF  PORTC, 7       ;set RC7
     GOTO AGAIN
OVER
     BCF  PORTC, 7       ;clear RC7
     GOTO AGAIN          ;stay in this loop forever
          END
```

63. a. RBIF
b. INTCON
c. For PORTB-change, there is only a single flag associated with it and it is triggered on positive and negative edges.
d. Pins 37 (RB4), 38 (RB5), 39 (RB6), and 40 (RB7)

SECTION 11.6: INTERRUPT PRIORITY IN THE PIC18

64. True
65. IPEN of RCON register. Setting IPEN = 1 allows the interrupts to jump to 0x0008 or 0x0018 depending on individual xIP (priority) bits.
66. INT0 is always high priority. This is the only interrupt that does not have the option of changing the priority.
67. IPR1, BCF IPR1, TMR1IP
68. INTCON3, BCF INTCON3, INT1IP
69. It depends on the order that INT1IF and INT2IF are checked in the ISR located at 0x0018. The first one checked will be serviced first.
70. It depends on the order that TMR0IF and TMR1IF are checked in the ISR located at 0x0018. The first one checked will be serviced first.
71. It depends on the order that TMR0IF and TMR1IF are checked in the ISR located at 0x0008. The first one checked will be serviced first.
72. It depends on the order that INT1IF and INT2IF are checked in the ISR located at 0x0008. The first one checked will be serviced first.
73. The high priority ISR will be finished first, before it goes to the low priority ISR.
74. The low priority ISR will be interrupted by the high priority interrupt and the CPU will jump to the high priority ISR. Only after finishing the high priority ISR, that the CPU will finish executing the low priority ISR.
75. The GIEH will disable the high priority interrupts.
76. False
77. The GIEL will disable the low priority interrupts
78. The RETFIE at the end will enable the GIEL automatically which allows the low interrupts to come in again.
79. RETFIE 1 restores all shadow registers, WREG, BSR, and STATUS
80. Fast context allows the processor to save the three key registers WREG, BSR, and STATUS instead of saving them individually. This only works with high priority interrupts.

CHAPTER 12: LCD AND KEYBOARD INTERFACING

SECTION 12.1: LCD INTERFACING

1. 8
2. They are the control lines for the LCD. E is for latching information into the LCD, R/W is for reading from or writing information into the LCD , and RS is for making a distinction between the data and command.
3. Vee controls the contrast of the screen while Vcc provides power to the LCD.
4. command, 01h
5. 0Fh
6. RS = 0, RW = 0, E = H-to-L pulse
7. RS = 1, RW = 0, E = H-to-L pulse
8. (a)
9. True
10. Sending information to the LCD without checking the busy flag is very simple to write but the microcontroller wastes a lot of time in a delay subroutine. The advantage of monitoring the busy flag is that the next information is sent when the LCD is ready and there is no wasting of time. This is very critical, especially when the microcontroller is busy and must serve many devices. However, the subroutine for monitoring the busy flag is much harder to write and takes more code. To monitor the busy flag, we make the port D0 - D7 of the LCD an input port, read the D7 bit, and wait for low.
11. 80h is the starting location ,and 16 locations to the right is 8Fh.
12. From Table 12-3 we have C0h, which is first location of line 2.
13. 80h is the starting location, and 20 locations to the right is 13h which result in 80h + 13h = 93h.
14. This is 2 locations from C0h since C0 is the first location of line 2.
15. If 80h is the address of the first location, then adding 39 (27h) we get 80h + 27h = A7h.
16. If C0h is the address of the first location, then adding 39 (27h) we get C0h + 27h = E7h.
17. 89h since 80h + 9 = 89h. The 10th is address 9 since it starts at 0.
18. C0h + 13h = D3h. Notice that the 20th starts at 0 and goes to 13h.
19.

```
LCD_DATA  EQU PORTD      ;LCD data pins RD0-RD7
LCD_CTRL  EQU PORTC      ;LCD control pins
RS   EQU RC4             ;RS pin of LCD
RW   EQU RC5             ;R/W pin of LCD
EN   EQU RC6             ;E pin of LCD
     CLRF TRISD          ;PORTD = Output
     CLRF TRISC          ;PORTC = Output
     BCF  LCD_CTRL,EN    ;enable idle low
     CALL LDELAY         ;wait for initialization
     MOVLW 0x38          ;init. LCD 2 lines, 5x7 matrix
     CALL COMNWRT        ;call command subroutine
     CALL LDELAY         ;initialization hold
     MOVLW 0x0E          ;display on, cursor on
     CALL COMNWRT        ;call command subroutine
     CALL DELAY          ;give LCD some time
     MOVLW 0x01          ;clear LCD
     CALL COMNWRT        ;call command subroutine
     CALL DELAY          ;give LCD some time
     MOVLW 0x06          ;shift cursor right
     CALL COMNWRT        ;call command subroutine
     CALL DELAY          ;give LCD some time
     MOVLW 0x84          ;cursor at line 1, pos. 4
     CALL COMNWRT        ;call command subroutine
     CALL DELAY          ;give LCD some time
     MOVLW A'N'          ;display letter 'N'
     CALL DATAWRT        ;call display subroutine
     CALL DELAY          ;give LCD some time
     MOVLW A'O'          ;display letter 'O'
     CALL DATAWRT        ;call display subroutine
AGAIN     BTG  LCD_CTRL,0
     BRA  AGAIN          ;stay here
COMNWRT                  ;send command to LCD
     MOVWF LCD_DATA      ;copy WREG to LCD DATA pin
     BCF  LCD_CTRL,RS    ;RS = 0 for command
     BCF  LCD_CTRL,RW    ;R/W = 0 for write
     BSF  LCD_CTRL,EN    ;E = 1 for high pulse
     CALL SDELAY         ;make a wide En pulse
     BCF  LCD_CTRL,EN    ;E = 0 for H-to-L pulse
     RETURN
DATAWRT                  ;write data to LCD
     MOVWF LCD_DATA      ;copy WREG to LCD DATA pin
     BSF  LCD_CTRL,RS    ;RS = 1 for data
     BCF  LCD_CTRL,RW    ;R/W = 0 for write
     BSF  LCD_CTRL,EN    ;E = 1 for high pulse
     CALL SDELAY         ;make a wide En pulse
     BCF  LCD_CTRL,EN    ;E = 0 for H-to-L pulse
     RETURN
;look in previous chapters for delay routines
     END
```

20.

```
;PORTD = D0-D7, RC4 = RS, RC5 = R/W, RC6 = E pins
LCD_DATA  EQU PORTD      ;LCD data pins RD0-RD7
LCD_CTRL  EQU PORTC      ;LCD control pins
RS   EQU RC4             ;RS pin of LCD
RW   EQU RC5             ;R/W pin of LCD
EN   EQU RC6             ;E pin of LCD
     CLRF TRISD          ;PORTD = Output
     CLRF TRISC          ;PORTC = Output
     BCF  LCD_CTRL,EN    ;enable idle low
     CALL LDELAY         ;long delay (250 ms) for power-up
     MOVLW upper(MYCOM)
     MOVWF     TBLPTRU
     MOVLW     high(MYCOM)
     MOVWF     TBLPTRH
     MOVLW     low(MYCOM)
     MOVWF     TBLPTRL
C1   TBLRD*+
     MOVF TABLAT,W       ;give it to WREG
     IORLW 0x0           ;Is it the end of command?
     BZ   SEND_DAT       ;if yes then go to display data
     CALL COMNWRT        ;call command subroutine
     CALL DELAY          ;give LCD some time
     BRA  C1
SEND_DAT  MOVLW     upper(MYDATA)
     MOVWF     TBLPTRU
     MOVLW     high(MYDATA)
     MOVWF     TBLPTRH
     MOVLW     low(MYDATA)
     MOVWF     TBLPTRL
DT1  TBLRD*+
     MOVF TABLAT,W       ;give it to WREG
     IORLW     0x0       ;Is it the end of data string?
     BZ   OVER           ;if yes then exit
     CALL DATAWRT        ;call DATA subroutine
     CALL DELAY          ;give LCD some time
     BRA  DT1
OVER BRA  OVER           ;stay here
COMNWRT                  ;send command to LCD
     MOVWF     LCD_DATA  ;copy WREG to LCD DATA pin
     BCF  LCD_CTRL,RS    ;RS = 0 for command
     BCF  LCD_CTRL,RW    ;R/W = 0 for write
     BSF  LCD_CTRL,EN    ;E = 1 for high pulse
     CALL SDELAY         ;make a wide En pulse
     BCF  LCD_CTRL,EN    ;E = 0 for H-to-L pulse
     RETURN
DATAWRT                  ;write data to LCD
     MOVWF     LCD_DATA  ;copy WREG to LCD DATA pin
     BSF  LCD_CTRL,RS    ;RS = 1 for data
     BCF  LCD_CTRL,RW    ;R/W = 0 for write
     BSF  LCD_CTRL,EN    ;E = 1 for high pulse
     CALL      SDELAY    ;make a wide En pulse
     BCF  LCD_CTRL,EN    ;E = 0 for H-to-L pulse
     RETURN
     ORG  500H
MYCOM     DB 0x38,0x0E,0x01,0x06,0x84,0 ;commands and null
MYDATA    DB "HELLO",0   ;data and null
;look in Chapter 3 for delay routines
     END
```

SECTION 12.2: KEYBOARD INTERFACING

21. 1s
22. (a)
23. (b)
24. key 3
25. This allows the microcontroller to do other things.
26. Use a chip for the keyboard (for both key press and identification) and then pass the scan code for the pressed key to the microcontroller via a hardware interrupt.

CHAPTER 13: ADC, DAC, AND SENSOR INTERFACING

SECTION 13.1: ADC CHARACTERISTICS

1. True
2. True
3. True
4. False
5. True
6. True
7. False
8. True
9. False
10. False
11. (d)
12. (d)
13. (a) (5V) / (2^8) = 19.53 mV
    (b) (5V) / (2^{10}) = 4.88 mV
    (c) (5V) / (2^{12}) = 1.22 mV
    (d) (5V) / (2^{16}) = 76.29 µV
14. (a) since the step size is 5 mV and all inputs are high, we get 255 x 5 mV = 1.275 V.
    (b) 10011001 = 153 and 153 x 5 mV = 0.756 V.
    (c) 1101100= 108 and 108 x 5 mV = 0.540 V.
15. 1.28 V since 256 x 5 mV = 1.28V
16. (a) since the step size is 10 mV and all inputs are high, we get 255 x 10 mV = 2.56 V.
    (b) 10011001 = 153 and 153 x 10 mV = 1.53 V.
    (c) 1101100= 108 and 108 x 10 mV = 1.08 V.

SECTION 13.2: ADC PROGRAMMING IN PIC18

17. True
18. False. It is 10-bit
19. True
20. True
21. True
22. False
23. False
24. True
25. False
26. True
27. It is kept by the registers ADRESL and ADRESH. At the end of conversion the DONE flag goes low.
28. The old data gets lost.
29. (a) 1.024V / 1024 = 1 mV
    (b) 2.048 / 1024 = 2 mV
    (c) 2.56 / 1024 = 2.5 mV

30. 2 mV x 1024 = 2048mV = 2.048V

31. 3 mV x 1024 = 3.072V

32. 1 mV x 1023 = 1.023V

33. Step Size = 1.024 /1024 = 1 mV
(a) 255 x 1 mV = 0.255V
(b) 152 x 1 mV = 0.152V
(c) 208 x 1 mV = 0.208V

34. 4 mV x 1024 = 4096 mV = 4.096 V

35. Step Size = 2.56/1024 = 2.5 mV
(a) 1023 x 2.5 mV = 2.5575V
(b) 513 x 2.5mV = 1.2825V
(c) 816 x 2.5 mV = 2.04V

36. (a) 8MHz/2 = 4MHz ,Tad = 1/4MHz = 250ns, Invalid since it is faster than 1.6μs
(b) 8MHz/4 = 2MHz ,Tad = 1/2MHz = 500ns, Invalid since it is faster than 1.6μs
(c) 8MHz/8 = 1MHz ,Tad = 1/1MHz = 1μs, Invalid since it is faster than 1.6μs
(d) 8MHz/16 = 500KHz ,Tad = 1/500KHz = 2μs and conversion time is 12 x 2μs = 24μs.
(e) 8MHz/32 = 250KHz ,Tad = 1/250KHz = 4μs and conversion time is 12 x 4μs = 48μs

37. (a) 12MHz/8 = 1.5MHz ,Tad = 1/1.5 MHz = 667ns, Invalid since it is faster 1.6μs.
(d) 12MHz/16 = 750KHz ,Tad = 1/750KHz = 1.33μs, Invalid since it is faster 1.6μs.
(e) 12MHz/32 = 375KHz ,Tad = 1/375KHz = 2.67μs and conversion time is 12 x 2.67μs = 32.04μs
(e) 12MHz/64 = 187.5KHz ,Tad = 1/187.5KHz = 5.3μs and conversion time is
12 x 5.3μs = 63.6μs

38. By the instruction “BSF ADCON0,GO”

39. By monitoring the GO bit of ADCON0 register.

40. One or none at at all.

41. PORTA.

42. ADCON1

43. ADCON0

44. ADCON0

45. 01000001 = 41H

46. 11000100 = C4H

47. 00010000 = 10H

48. 00000101 = 05H

49. ADIF bit belongs to PIR1 register and ADIE belongs to PIE1 register

50. High

SECTION 13.3: DAC INTERFACING

51. True.

52. (a) 256 (b) 1024 (c) 4096

53. 2 ma * (255/256) = 1.99 ma

54. (a) 1.195 mA since 10011001B = 153 and (153 x 2 mA)/256 = 1.195 mA
(b) 1.594 mA since 11001100B = 204 and (204 x 2 mA)/256= 1.594 mA
(c) 1.859 mA since 11101110B = 238 and (238 x 2 mA)/256 = 1.859 mA

(d) 0.2656 mA
(e) 0.0703125 mA
(f) 1.063 mA

55. More
56. All high

SECTION 13.4: SENSOR INTERFACING AND SIGNAL CONDITIONING

57. The output corresponds with the input linearly.
58. 10 mV
59. Adjusting the output of a given sensor (device) to meet the needs of the ADC chip
60. To maintain constant voltage so that the variation of V_{CC} source has no effect on the V_{ref} voltage. This must be done since the source voltage for V_{ref} is V_{CC}.

CHAPTER 14: USING FLASH AND EEPROM MEMORIES FOR DATA STORAGE

SECTION 14.1: SEMICONDUCTOR MEMORY

1. The storage of the chip is measured in Megabits while the Computer memory is measured in Megabytes.
2. True, the more address lines the more memory locations.
3. True, the memory location size is fixed.
4. True, the more data lines the more memory locations
5. True
6. access time
7. True
8. electrically erasable
9. True
10. DRAM
11. SRAM
12. DRAM and SRAM
13. (c)
14. (c)
15. (a) 32Kx8, 256K
    (b) 8Kx8, 64K
    (c) 4Kx8, 32K
    (d) 8Kx8, 64K
    (e) 4Mx1, 4M
    (f) 8Kx1, 8K
    (g) 4Kx8, 32K
    (h) 2Kx8, 16K
    (i) 256Kx4, 1M
    (j) 64Kx8, 512K
16. (a) 128K 14 8
    (b) 256K 15 8
    (c) 512K 16 8
    (d) 2M 18 8
    (e) 512K 16 8
    (f) 256K 8 4
    (g) 8M 20 8
    (h) 16M 11 4
    (i) 512K 16 8

SECTION 14.2: ERASING AND WRITING TO FLASH IN THE PIC18F

17. True
18. True
19. True
20. True
21. False
22. False. It can be erased in blocks of 64 bytes, but when we write, it must be in block of 8 bytes.
23. False
24. False
25. TBLPTR and TABLAT
26. TBLPTR, TABLAT, EECON1, and EECON2
27. TBLPTR, EECON1, and EECON2
28. WREN bit is used by the programmer to enable writing to Flash and EEPROM. The WR is used by the PIC18 to indicate that write cycle is finished.
29. TBLPTR and TABLAT

30. TBLPTR, TABLAT, EECON1, and EECON2
31. In the short write the data is stored in the buffer called TABLAT, while in the long write the data is actually written(burned) into the Flash.
32. During the long write
33. 8 bytes
34. 64 bytes
35. (a) and (e)
36. (a), (c), (e), and (g)
37. 2000H, 2008H, 2010H, 2018H
38. 2000H, 2040H, 2080H, 20C0H
39.

```
     MOVLW upper(MYDATA)
     MOVWF TBLPTRU       ;load the upper address
     MOVLW high(MYDATA)
     MOVWF TBLPTRH       ;load the high byte of address
     MOVLW low(MYDATA)
     MOVWF TBLPTRL       ;load the low byte of address
     BSF EECON1,EEPGD    ;point to Flash memory
     BCF EECON1,CFGS     ;access Flash program
     BSF EECON1,WREN     ;enable write
     BSF EECON1, FREE    ;enable row erase operation
     BCF INTCON,GIE      ;disable all interrupts
     MOVLW 55H           ;wreg = 55h
     MOVWF EECON2        ;write to dummy reg
     MOVLW 0AAH          ;wreg = aah
     MOVWF EECON2        ;write to dummy reg
     BSF  EECON1,WR      ;now write it to Flash
     NOP                 ;wait
     BSF  INTCON,GIE     ;enable all interrupts
     BCF  EECON1,WREN    ;disable write to memory
     MOVLW upper(MYDATA)
     MOVWF TBLPTRU       ;load the upper address
     MOVLW high(MYDATA)
     MOVWF TBLPTRH       ;load the high byte of address
     MOVLW low(MYDATA)
     MOVWF TBLPTRL       ;load the low byte of address
;start a short write
     MOVLW A'H'          ;load the byte into WREG
     MOVWF TABLAT        ;move it to TABLATch reg
     TBLWT*+             ;perform short write and increment
     MOVLW A'E'          ;load the byte into WREG
     MOVWF TABLAT        ;move it to TABLATch reg
     TBLWT*+             ;perform short write and increment
     MOVLW A'L'          ;load the byte into WREG
     MOVWF TABLAT        ;move it to TABLATch reg
     TBLWT*+             ;perform short write
     MOVLW A'L'          ;load the byte into WREG
     MOVWF TABLAT        ;move it to TABLATch reg
     TBLWT*+             ;perform short write
     MOVLW A'O'          ;load the byte into WREG
     MOVWF TABLAT        ;move it to TABLATch reg
     TBLWT*+             ;perform short write
     BSF  EECON1,EEPGD   ;point to Flash memory
     BCF  EECON1,CFGS    ;
     BSF  EECON1,WREN    ;enable write
     BCF  INTCON,GIE     ;disable all interrupts
     MOVLW 55H           ;wreg = 55h
     MOVWF EECON2        ;write to dummy reg
     MOVLW 0AAH          ;wreg = aah
     MOVWF EECON2        ;write to dummy reg
     BSF  EECON1,WR      ;now write it to Flash
     NOP                 ;wait
     BSF  INTCON,GIE     ;enable all interrupts
     BCF  EECON1,WREN    ;disable write to memory
     GOTO $              ;stay here
     ORG 2000H
MYDATA data “ERASE ME”
     END
```

40. Add the following to the end of Problem 39.

```
     MOVLW 0x20          ;enable transmit and low baud rate
     MOVWF TXSTA         ;write to reg
     BCF  PIR1,TXIF
     MOVLW D'15'         ;9600 bps (Fosc/(64*Speed)-1)
     MOVWF SPBRG         ;write to reg
     BCF  TRISC, TX      ;make TX pin of PORTC an output
     BSF  RCSTA, SPEN    ;enable the entire serial port
     MOVLW 5             ;number of bytes in RAM
     MOVWF COUNT
     MOVLW upper (MYDATA);load TBLPTR
     MOVWF TBLPTRU
     MOVLW high (MYDATA)
     MOVWF TBLPTRH
     MOVLW low (MYDATA)
     MOVWF TBLPTRL
LN   TBLRD*+             ;read the character
     MOVF     TABLAT,W
OVER BTFSS PIR1, TXIF    ;wait to send character
     BRA OVER
     MOVWF TXREG         ;send character to serial port
     DECFSZ COUNT,F      ;loop until buffers are full
     BRA  LN             ;repeat
     END
```

SECTION 14.3: READING AND WRITING OF DATA EEPROM IN PIC18F

41. False
42. True
43. False
44. True
45. True
46. False.
47. False, there is no erase function. The write is a replace function.
48. False
49. True
50. EEADR, EEDATA, and EECON1
51. EEADR, EEDATA, EECON1, and EECON2
52. Flash can be used for both code (program) and data, while the EEPROM is used for data storage only. Writing to Flash is in block size, while writing to EEPROM is byte size
53. one byte
54. EEPGD, CFGS, and RD
55. To prevent interruption during the long write which can take up 2 msec.
56. Since reading the byte from Flash and EEPROM take only 2 machine cycle, there is no need to disable the interrupts. This is unlike the long write cycle time, which takes up to 2 msec
57.

```
     MOVLW 0x0           ;starts at location 0H of EEPROM
     MOVWF EEADR         ;load the EEPROM address
     MOVLW    A'H'       ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A'E'          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A'L'          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A'L'          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A'O'          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A' '          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A'W'          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A'O'          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A'R'          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A'L'          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
     MOVLW A'D'          ;load the byte into WREG
     MOVWF EEDATA        ;move it to EEDATA reg
     CALL EE_WRT
     INCF EEADR,F        ;point to next location
HERE BRA  HERE
EE_WRT
     BCF  EECON1,EEPGD   ;point to EEPROM memory
     BCF  EECON1,CFGS    ;
     BSF  EECON1,WREN    ;enable write
     BCF  INTCON,GIE     ;disable all interrupts
     MOVLW 0x55          ;wreg = 55h
     MOVWF EECON2        ;write to dummy reg
     MOVLW 0xAA          ;wreg = aah
     MOVWF EECON2        ;write to dummy reg
     BSF  EECON1,WR      ;now write it to Flash
     BSF  INTCON,GIE     ;enable all interrupts
EE_WAIT   BTFSS PIR2,EEIF
     BRA  EE_WAIT
     BCF  PIR2,EEIF
     RETURN
     END
```

58. Add the following to the end of Problem 57.

```
     MOVLW 0x20          ;enable transmit and low baud rate
     MOVWF TXSTA         ;write to reg
     BCF  PIR1,TXIF
     MOVLW D'15'         ;9600 bps (Fosc/(64*Speed)-1)
     MOVWF SPBRG         ;write to reg
     BCF  TRISC, TX      ;make TX pin of PORTC an output
     BSF  RCSTA, SPEN    ;enable the entire serial port
     MOVLW 0x0           ;starts at location 0H of EEPROM
     MOVWF EEADR         ;load the EEPROM address
     BCF  EECON1,EEPGD   ;point to EEPROM memory
     BCF  EECON1,CFGS    ;
     MOVLW D’11’         ;count = 11
     MOVWF COUNT
OVER BSF  EECON1,RD      ;enable read
     NOP
R1   BTFSS PIR1, TXIF
     BRA  R1
     MOVFF EEDATA,TXREG  ;read the data to serial port
     INCF EEADR,F        ;point to next location
     DECF COUNT,F        ;decrement counter
     BNZ  OVER           ;keep repeating
HERE BRA  HERE
     END
```

CHAPTER 15: CCP AND ECCP PROGRAMMING

SECTION 15.1: STANDARD AND ENHANCED CCP MODULES

1. True
2. True
3. True
4. True
5. True
6. 2
7. Pin 17

SECTION 15.2: COMPARE MODE PROGRAMMING

8. True
9. False
10. True
11. Timer1 and/or Timer3
12. When the TMR1H:TMR1L equals CCPR1H:CCPR1L.
13. PIR1
14. 0x08
15. 0x09
16. 0x02
17.

```
     MOVLW 0x02
     MOVWF CCP1CON       ;Compare mode, toggle upon match
     CLRF  T3CON         ;Timer1 for Compare
     CLRF  T1CON         ;Timer 1, internal clock, 1:1
prescaler
     BCF   TRISC,CCP1    ;CCP1 pin as output
     BSF   TRISC,T1CKI   ;T1CLK pin as input pin
     MOVLW D'10'
     MOVWF CCPR1L        ;CCPR1L = 10
     MOVLW 0x0           ;CCPR1H = 0
     MOVWF CCPR1H
OVER CLRF  TMR1H         ;clear TMR1H
     CLRF  TMR1L         ;clear TMR1L
     BCF   PIR1,CCP1IF   ;clear CCP1IF
     BSF   T1CON,TMR1ON  ;start Timer1
B1   BTFSS PIR1,CCP1IF
     BRA   B1
;---------CCP toggle CCP pin upon match
B2   BCF   T1CON,TMR1ON  ;stop Timer1
     GOTO  OVER          ;keep doing it
```

18.

```
          MOVLW 0x02
          MOVWF CCP1CON       ;Compare mode, toggle upon match
          MOVLW 0x42
          MOVWF T3CON         ;Timer3 for Compare, 1:1 prescaler
          BCF   TRISC,CCP1    ;CCP1 pin as output
          BSF   TRISC,T1CKI   ;T3CLK pin as input pin
          MOVLW 0xE8
          MOVWF CCPR1L        ;CCPR1L = E8 (3E8 = 1000)
          MOVLW 0x03          ;CCPR1H = 3
          MOVWF CCPR1H
     OVER CLRF  TMR3H         ;clear TMR3H
          CLRF  TMR3L         ;clear TMR3L
          BCF   PIR1,CCP1IF   ;clear CCP1IF
          BSF   T3CON,TMR3ON  ;start Timer3
     B1   BTFSS PIR1,CCP1IF
          BRA   B1
     ;---------CCP toggle CCP pin upon match
     B2   BCF   T3CON,TMR3ON  ;stop Timer3
          GOTO  OVER          ;keep doing it

          MOVLW 0x02
          MOVWF CCP1CON       ;Compare mode, toggle upon match
          MOVLW 0x42
          MOVWF T3CON;use Timer3 for Compare mode, 1:1 Prescaler
          BCF   TRISC,CCP1    ;CCP1 pin as output
          MOVLW 0xC3
          MOVWF CCPR1H        ;CCPR1H = 0xC3
          MOVLW 0x50
          MOVWF CCPR1L        ;CCPR1L = 0x50
     OVER CLRF  TMR3H         ;clear TMR3H
          CLRF  TMR3L         ;clear TMR3L
          BCF   PIR1,CCP1IF   ;clear CCP1IF
          BSF   T3CON,TMR3ON  ;start Timer3
     B1   BTFSS PIR1,CCP1IF
          BRA   B1
          ;CCP toggles CCP1 pin upon match
          BCF   T3CON,TMR3ON  ;stop Timer3
          GOTO  OVER          ;keep doing it
```

19.

```
          MOVLW 0x02
          MOVWF CCP1CON       ;Compare mode, toggle upon match
          MOVLW 0x0
          MOVWF T3CON         ;use Timer1 for Compare mode
          MOVLW 0x0
          MOVWF T1CON    ;Timer1, internal CLK, 1:1 prescale
          BCF   TRISC,CCP1    ;CCP1 pin as output
```

```
     MOVLW 0x30
     MOVWF CCPR1H        ;CCPR1H = 0x30
     MOVLW 0xD4
     MOVWF CCPR1L        ;CCPR1L = 0xD4
OVER CLRF  TMR1H         ;clear TMR1H
     CLRF  TMR1L         ;clear TMR1L
     BCF   PIR1,CCP1IF   ;clear CCP1IF
     BSF   T1CON,TMR1ON  ;start Timer1
B1   BTFSS  PIR1,CCP1IF
     BRA   B1
     ;CCP toggles CCP1 pin upon match
     BCF   T1CON,TMR1ON  ;stop Timer1
     GOTO  OVER          ;keep doing it
```

SECTION 15.3: CAPTURE MODE PROGRAMMING

20. True
21. False
22. False
23. Timer1 and/or Timer3
24. 0x04
25. 0x06
26. 0x00
27.

```
     MOVLW 0x05
     MOVWF CCP1CON       ;Capture mode risingedge
     MOVLW 0x42
     MOVWF T3CON         ;Timer1 for Capture,
     CLRF  TRISB         ;make PORTB output port
     CLRF  TRISD         ;make PORTD output port
     BSF   TRISC,CCP1    ;make CCP1 pin an input
     MOVLW 0x0
     MOVWF CCPR1H        ;CCPR1H = 0
     MOVWF CCPR1L        ;CCPR1L = 0
OVER CLRF  TMR3H         ;clear TMR3H
     CLRF  TMR3L         ;clear TMR3L
     BCF   PIR1,CCP1IF   ;clear CCP1IF
RE_1 BTFSS PIR1,CCP1IF
     BRA   RE_1          ;stay here for 1st rising edge
     BSF   T3CON,TMR3ON  ;start Timer3
     BCF   PIR1,CCP1IF   ;clear CCP1IF for next
RE_2 BTFSS PIR1,CCP1IF
     BRA   RE_2          ;stay here for 2nd rising edge
     BCF   T3CON,TMR3ON  ;stop Timer3
     MOVFF TMR3L,PORTB   ;put low byte on PORTB
     MOVFF TMR3H,PORTD   ;put high byte on PORTD
     GOTO OVER           ;keep doing it
```

SECTION 15.4: PWM PROGRAMMING

28. True
29. False
30. True
31. Timer2
32. 0x0C
33. CCPR1L
34. CCP1CON
35. Sets the decimal point portion of the duty cycle
36. 11
37. Fractional
38. Using the PR2 = Fosc / (4 * Fpwm * N) equation, we must set N = 8 for prescale. Therefore, we have
    PR2 = [(10 MHz / (4 * 2 kHz * 8)] – 1 = 156 – 1 = 155 and because
    155 * 25% = 38.75 we have CCPR1L = 38 and DC1B2:DC1B1 = 11 for the
    0.75 portion.
39. Using the PR2 = Fosc / (4 * Fpwm * N) equation, we must set N = 8 for prescale. Therefore, we have
    PR2 = [(10 MHz / (4 * 1.8 kHz * 8)] – 1 = 174 – 1 = 173 and because
    173 * 25% = 43.25 we have CCPR1L = 43 and DC1B2:DC1B1 = 01 for the
    0.25 portion.
40. Using the PR2 = Fosc / (4 * Fpwm * N) equation, we must set N = 8 for prescale. Therefore, we have
    PR2 = [(10 MHz / (4 * 1.5 kHz * 8)] – 1 = 208 – 1 = 207 and because
    207 * 25% = 51.75 we have CCPR1L = 51 and DC1B2:DC1B1 = 11 for the
    0.75 portion.
41. Using the PR2 = Fosc / (4 * Fpwm * N) equation, we must set N = 16 for prescale. Therefore, we have
    PR2 = [(10 MHz / (4 * 1.2 kHz * 16)] – 1 = 130 – 1 = 129 and because
    129 * 25% = 32.25 we have CCPR1L = 32 and DC1B2:DC1B1 = 01 for the
    0.25 portion.

SECTION 15.5: ECCP PROGRAMMING

42. True
43. False
44. False
45. Timer2
46. PIR2
47. 0x0A
48. Pin 27
49. Timer1 or Timer3
50. Pins 27, 28, 29, 30
51. 0x08

52. 0x4D

53. EDC1B1:EDC1B0

54. ECCP1CON

55. Sets the least significant bits, which are the decimal portion.

56. 10

57. fractional

58. Using the PR2 = Fosc / (4 * Fpwm * N) equation, we must set N = 8 for prescale. Therefore, we have
PR2 = [(10 MHz / (4 * 2 kHz * 8)] – 1 = 156 – 1 = 155 and because
155 * 25% = 38.75 we have ECCPR1L = 38 and EDC1B2:EDC1B1 = 11 for the 0.75 portion.

59. Using the PR2 = Fosc / (4 * Fpwm * N) equation, we must set N = 8 for prescale. Therefore, we have
PR2 = [(10 MHz / (4 * 1.8 kHz * 8)] – 1 = 174 – 1 = 173 and because
173 * 25% = 43.25 we have ECCPR1L = 43 and EDC1B2:EDC1B1 = 01 for the 0.25 portion.

60. Using the PR2 = Fosc / (4 * Fpwm * N) equation, we must set N = 8 for prescale. Therefore, we have
PR2 = [(10 MHz / (4 * 1.5 kHz * 8)] – 1 = 208 – 1 = 207 and because
207 * 25% = 51.75 we have ECCPR1L = 51 and EDC1B2:EDC1B1 = 11 for the 0.75 portion.

61. Using the PR2 = Fosc / (4 * Fpwm * N) equation, we must set N = 16 for prescale. Therefore, we have
PR2 = [(10 MHz / (4 * 1.2 kHz * 16)] – 1 = 130 – 1 = 129 and because
129 * 25% = 32.25 we have ECCPR1L = 32 and EDC1B2:EDC1B1 = 01 for the 0.25 portion.

CHAPTER 16: SPI PROTOCOL AND DS1306 RTC INTERFACING

SECTION 16.1: SPI BUS PROTOCOL

1. True
2. False
3. True
4. False
5. True
6. True
7. True
8. False
9. SDI and SDO are tied together
10. D7 is high for a write, D7 is low for a read

SECTION 16.2: DS1306 RTC INTERFACING AND PROGRAMMING

11. 16
12. Pin 16
13. 2
14. True
15. True
16. 2099
17. SDI is used to bring data into the SPI device. SDO is used to send data out of the SPI device. SCL is used to synchronize data transfer between the SPI device and microcontroller.
18. input
19. low, high
20. Vbat, if it is connected to an external source, otherwise it is lost.
21. Voltage Battery backup, input
22. Gnd
23. Input, Vcc
24. Input, +5
25. Output
26. Output
27. Output
28. Output
29. 128, Read 00-7F, Write 80-FF
30. Undefined if all power is lost. If secondary source is available, it will retain the data.
31. It will retain the last value, since only the general purpose RAM is nonvolatile.
32. When Vcc1 drops below Vbat
33. 0x00-0x02
34. 0x03-0x06
35. 0x82, bit 5
36. 0x82, bit 6

37. 0x06
38. 0x7F
39. True

```
40.       BSF  PORTC,RC2      ;enable the RTC
          CALL SDELAY
          MOVLW 0x06          ;years register address for read
          CALL SPI            ;send address to DS1306
          CALL SPI            ;get the year
          MOVWF PORTB         ;save the year
          BCF  PORTC,RC2      ;disable RTC
          MOVLW 0x20
          MOVWF PORTD

41.       BSF  PORTC,RC2      ;enable the RTC
          CALL SDELAY
          MOVLW 0x01     ;minutess register address for read
          CALL SPI            ;send address to DS1306
          CALL SPI            ;get the minutes
          MOVWF PORTB         ;save the minutes
          CALL SPI            ;get the hour
          MOVWF PORTD         ;save the hour
          BCF  PORTC,RC2      ;disable RTC

42.       BSF PORTC,RC2       ;enable the RTC
          MOVLW 0x80     ;seconds register address for write
          CALL SPI            ;send address
          MOVLW 0x05          ;05 seconds
          CALL SPI            ;send seconds
          MOVLW 0x15          ;15 minutes
          CALL SPI            ;send minutes
          MOVLW 0x69          ;12-hour clock at 9 pm hour
          CALL SPI            ;send hour
          BCF  PORTC,RC2      ;disable RTC

43.       BSF PORTC,RC2       ;enable the RTC
          MOVLW 0x80     ;seconds register address for write
          CALL SPI            ;send address
          MOVLW 0x19          ;19 seconds
          CALL SPI            ;send seconds
          MOVLW 0x47          ;47 minutes
          CALL SPI            ;send minutes
          MOVLW 0x22          ;24-hour clock at 22 hours
          CALL SPI            ;send hour
          BCF PORTC,RC2       ;disable RTC

44.       BSF PORTC,RC2       ;enable the RTC
          MOVLW 0x84          ;date register address for write
          CALL SPI            ;send address
          MOVLW 0x14          ;14th of the month
          CALL SPI            ;send day of the month
          MOVLW 0x05          ;May
          CALL SPI            ;send month
          MOVLW 0x09          ;2009
          CALL SPI            ;send year
          BCF  PORTC,RC2      ;disable RTC
```

45. Vbat is a non-rechargeable source. Vcc2 provides a trickle charge for a rechargeable source.

SECTION 16.3: DS1306 RTC PROGRAMMING IN C

```
46.       PORTCbits.RC2 = 1;  //begin multi byte read
          SDELAY(1);
          SPI(0x00);          //seconds register address
          for(i=0;i<3;i++)
               {
                    data[i] = SPI(0x00);     //get time and save
               }
          PORTCbits.RC2 = 0;            //end of multi byte read
          //-- convert time and display HH:MM:SS  AM/PM
          BCDtoASCIIandSEND(data[2]);   //the hour
          BCDtoASCIIandSEND(data[1]);   //the minute
          BCDtoASCIIandSEND(data[0]);   //the second

47.       PORTCbits.RC2 = 1;  //begin multi byte read
          SDELAY(1);
          SPI(0x06);          //years register address
          PORTB = SPI(0x00);  //get year and send to PORTB
               PORTCbits.RC2 = 0;  //end of multi byte read
          //-- convert time and display HH:MM:SS  AM/PM
          PORTD = 0x20;       //the century

48.       PORTCbits.RC2 = 1;  //begin multi byte read
          SDELAY(1);
          SPI(0x01);          //minutes register address
          for(i=0;i<2;i++)
               {
                    data[i] = SPI(0x00);     //get time and save
               }
          PORTCbits.RC2 = 0;  //end of multi byte read
          //-- convert time and display HH:MM:SS  AM/PM
          PORTD = data[1];    //the hour
          PORTB = data[0];    //the minute
```

```
49.       PORTCbits.RC2 = 1;  //enable the RTC
          SDELAY(1);
          SPI(0x8F);          //control register address
          SPI(0x00);          //clear WP bit for write
          PORTCbits.RC2 = 0;  //end of single-byte write
          SDELAY(1);
          PORTCbits.RC2 = 1;  //begin multi byte write
          SPI(0x80);          //seconds register address
          SPI(0x05);          //05 seconds
          SPI(0x15);          //15 minutes
          SPI(0x69);          //12-hour clock at 9 pm hour
          PORTCbits.RC2 = 0;  //end multi byte write

50.       PORTCbits.RC2 = 1;  //enable the RTC
          SDELAY(1);
          SPI(0x8F);          //control register address
          SPI(0x00);          //clear WP bit for write
          PORTCbits.RC2 = 0;  //end of single-byte write
          SDELAY(1);
          PORTCbits.RC2 = 1;  //begin multi byte write
          SPI(0x80);          //seconds register address
          SPI(0x19);          //19 seconds
          SPI(0x47);          //47 minutes
          SPI(0x22);          //24-hour clock at 22 hours
          PORTCbits.RC2 = 0;  //end multi byte write

51.       PORTCbits.RC2 = 1;  //enable the RTC
          SDELAY(1);
          SPI(0x8F);          //control register address
          SPI(0x00);          //clear WP bit for write
          PORTCbits.RC2 = 0;  //end of single-byte write
          SDELAY(1);
          PORTCbits.RC2 = 1;  //begin multi byte write
          SPI(0x84);          //date register address
          SPI(0x14);          //14th of the month
          SPI(0x05);          //May
          SPI(0x09);          //2009
          PORTCbits.RC2 = 0;  //end multi byte write
```

52. The RTC does not keep track of the century on its own, since it goes from 00 to 99 for the year.

SECTION 16.4: ALARM AND INTERRUPT FEATURES OF THE DS1306

53. Output, low
54. Output

55. bits 0 and 1, Clear the WP bit and then set the AIE1 and AIE0 bits.
56. bit 2, Clear WP bit and then set the 1-Hz bit.
57. bit 0
58. bit 1
59. True
60. The options associated with Alarm1.
61. To make an external signal for hardware interrupt.
62. IRQF0 is an interrupt flag and AIE0 enables the flag to go out to INT0 pin
63. IRQF1 is an interrupt flag and AIE1 enables the flag to go out to INT1 pin
64. Set bit 2 of Control register at address 0x8F.
65. 0x8B
66. The IRQ1F goes high when the seconds match and the minutes and hours are set to 0x80 of Alarm1.

CHAPTER 17: MOTOR CONTROL: RELAY, PWM, DC, AND STEPPER MOTORS

SECTION 17.1: RELAYS AND OPTOISOLATORS

1. False
2. True
3. There is no coil, spring or mechanical contact
4. False
5. 4.167 mA
6. Motor control and telecommunications
7. size and speed
8. EM relay
9. True
10. False

SECTION 17.2: STEPPER MOTOR INTERFACING

11. 4 degrees
12. 48
13. 1001, 1100, 0110
14. 0110, 0011, 1001
15. 0011, 0110, 1100
16. 1100, 1001, 0011
17. The ULN2003 provides addition current for driving the motor, No
18. b
19. Slows the motor
20. Decrease the time delay between steps

SECTION 17.3: DC MOTOR INTERFACING AND PWM

21. Stepper motor
22. The higher the load, the higher the current draw
23. False
24. no-load
25. Speed control
26. Positioning
27. True
28. Pulse width modulation. It varies the time on which affects the speed of the motor
29. Vary the PWM
30. An optoisolator protects the microcontroller pin