MICROCONTROLLERS AND EMBEDDED SYSTEMS COURSE

CLEVELAND INSTITUTE OF ELECTRONICS

[Enroll Today www.ciebookstore.com]

Lessons 8102-8120 (F680)

This is course was developed by the Instruction Department at CIE.

For use with *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)* by Ramesh S. Gaonkar (ISBN: 978-1-4018-7914-3)

Published by DELMAR/Cengage Learning (2007)

Table of Contents

Chat with Your Instructor

Welcome to the exciting world of microcontrollers. In this course we will delve into the PIC18 microcontroller family and learn about the fundamentals of microcontrollers and their application in embedded systems. This course contains ten lesson assignments covering material from the textbook *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)* by Ramesh S. Gaonkar.

The textbook does contain a few errors and the author did provide an errata sheet indicating the corrections, which have been included in this study guide. You will find them as a *Note* *** so if you have any questions regarding the errors please let us know. If you encounter other errors not mentioned in the study guide please let us know and we will contact the author.

A CD does accompany the textbook and it includes a simulator for the PIC18, source codes and data sheets. In Chapter 4 the PIC18 Simulator is discussed and you are welcome to experiment with the simulator but another study has been prepared by the Instructors at CIE that will use the simulator to write programs. If you are enrolled in a diploma program with CIE you will receive the study guide after completing this study guide automatically. If you are taking this course through CIE's Bookstore then the laboratory study guide is also available from the Bookstore.

If you have any questions please contact the Instruction Department at 1-800-243-6446 (216-781-9400) or email at faculty@cie-wc.edu for assistance. We hope you enjoy this course and we know you will come away with a better understanding of how embedded systems and more specifically microcontrollers influence our daily lives.

Lesson 8102-Microprocessors and Microcontrollers
Microcontrollers and Embedded Systems

Assignment Checklist

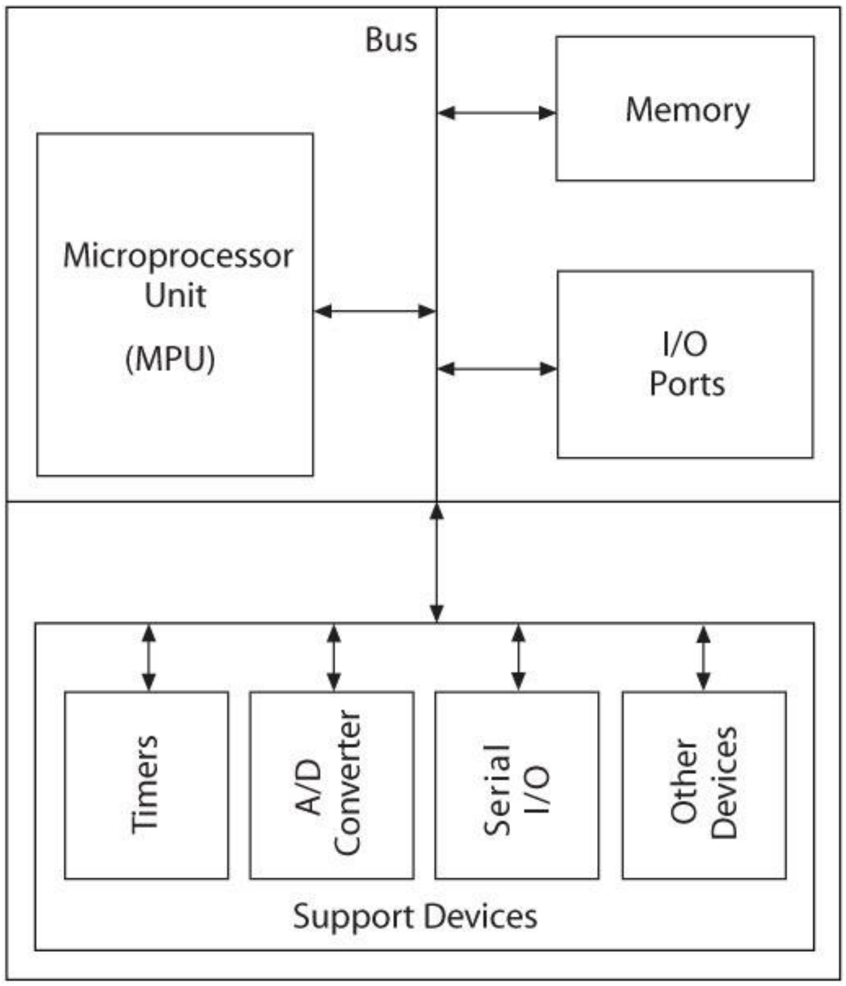
- ✓ Read Chapter 1 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Lesson 8102 in this Study Guide
- ✓ Take the Examination for this lesson

Chapter 1 Lecture/Discussion

The microcontroller is an integrated electronic computing and logic device that includes three major components on a single chip

- Microprocessor
- Memory
- I/O ports

The microcontroller—**Block Diagram**



A microcontroller unit (MCU) also includes support devices.

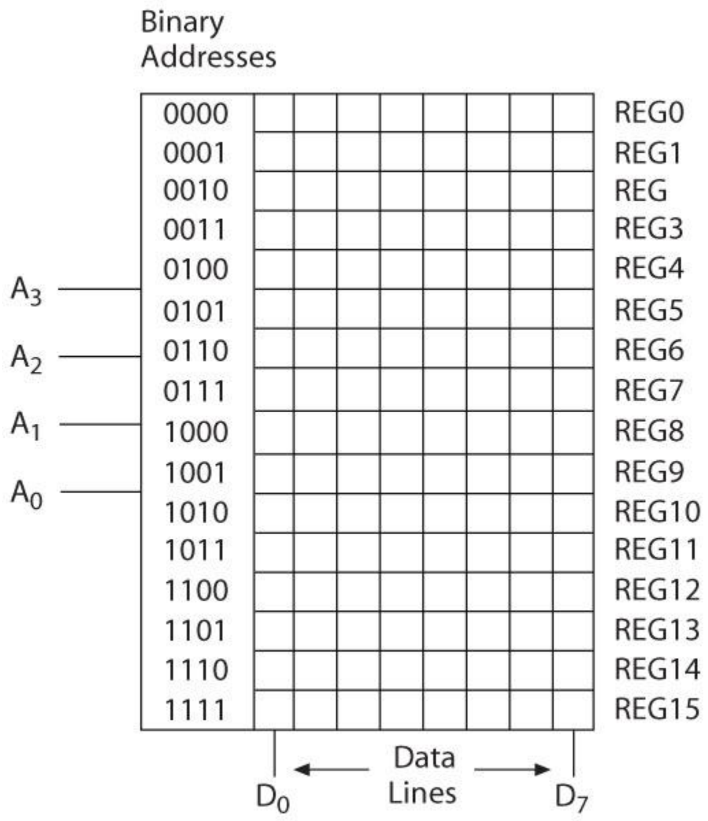
- Timers
- A/D converter
- Serial I/O
- Parallel Slave Port

Microprocessor (MPU) - A group of electronic circuits fabricated on a semiconductor chip that can read binary instructions written in memory and process binary data according to those instructions.

The four terms of CPU, MPU, processor and microprocessor are synonymous so the textbook will use them interchangeably.

Memory-A semiconductor storage device consisting of registers that store binary bits
Two major categories

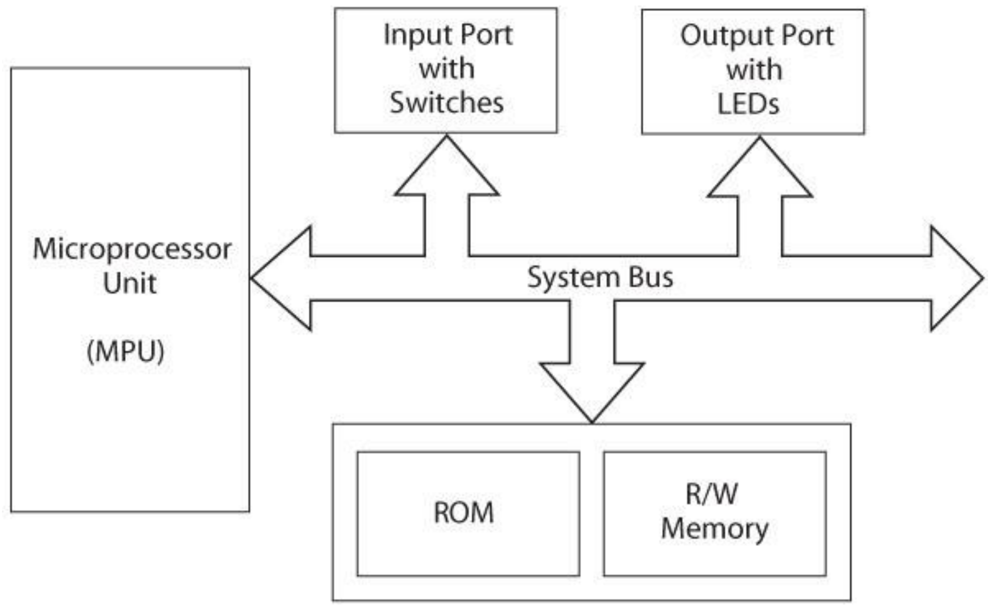
- Read/Write Memory (R/WM)
- Read-only-Memory (ROM)



Input/Output (I/O)
Input devices such as switches and keyboards provide binary information to the microprocessor (MPU) Output devices such as LEDs, video screens, and printers receive information from the microprocessor (MPU).

Bus
The three components – MPU, memory, and I/O – are connected by a group of wires called the **bus**. It is a common communication path between these components.

– Wires function as a group
– Same wires connected to multiple devices



Microprocessor Architecture
The MPU communicates with memory and I/O using the system bus consisting of:

– Address bus: unidirectional and carries memory and I/O addresses
– Data bus: bidirectional; transfers binary data and instructions between MPU and memory and I/O
– Control lines: Read and Write timing signals asserted by MPU

The microprocessor (MPU) is a computing and logic device that executes binary instructions in a sequence stored in memory.

– General purpose
– Binary
– Register-based
– Clock-driven
– Programmable

I/O (Input/Output)
Peripherals

– Input devices
    • switches, keyboard, mouse, scanner, and digital camera
– Output devices
    • LEDs, LCD, video screen, and printer
– Interfacing devices
    • buffer and latch

Software: From Machine to High-Level Languages

Machine Language: binary instructions (0 and 1)

- Difficult to decipher and write
- Prone to cause many errors in writing

All programs converted into the machine language of a processor for execution

Assembly Language: machine instructions represented in mnemonics

- Has one-to-one correspondence with machine instructions
- Efficient in execution and use of memory; machine-specific and not easy to troubleshoot

High-Level Languages (such as BASIC, C, and C++)

- Written in statements of spoken languages (such as English)
  - machine independent
  - easy to write and troubleshoot
  - requires large memory and less efficient in execution

Data Format (8-bit)

- Unsigned Integers: All eight bits (Bit 0 to Bit 7) represent the magnitude of a number
  - Range 0 to FF in Hex and 0 to 255 in decimal

- Signed Integers: Seven bits (Bit 0 to Bit 6) represent the magnitude of a number
  - The eighth bit (Bit 7) represents the sign of a number. The number is positive when Bit7 is zero and negative when Bit7 is one.
  - Positive numbers: 0 to 7F (0 to 127)
  - Negative numbers: 80 to FF (-1 to -128)
  - All negative numbers are represented in 2's complement

- Binary Coded Decimal Numbers (BCD)
  - 8 bits of a number divided into groups of four, and each group represents a decimal digit from 0 to 9
  - Four-bit combinations from A through F in Hex are invalid in BCD numbers
    - Example: 0010 0101 represents the binary coding of the decimal number 25 which is different in value from 25H. (H represents Hexadecimal base)

- American Standard Code for Information Interchange (ASCII)
  - Seven-bit alphanumeric code with 128 combinations (00 to 7F)
  - Represents English alphabet, decimal digits from 0 to 9, symbols, and commands

MPU-Based Systems

System hardware

- Includes discrete components such as a microprocessor, memory, and I/O
- Components connected by buses
  - address, data, and control

System software

- A group of programs that monitors the functions of the entire system

MCU-Based Systems

- Includes microprocessor, memory, I/O ports, and support devices (such as timers) on a single semiconductor chip
- Buses are generally not available to a system designer
- I/O ports are generally multiplexed and can be programmed to perform different functions

Historical Perspective and Look Ahead

- Technological advances
  - Transistors and Integrated Circuits from SSI to ULSI
- Birth of a microprocessor and its revolutionary impact
- von Neumann (Princeton) versus Harvard Architecture
- CISC versus RISC processors
- Microprocessors to microcontrollers

Lesson 8102 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. _____________ is not one of the components in a microprocessor-based system.
   (1) Microprocessor (3) I/O
   (2) Memory (4) Mouse

2. Which one describes a processor on a single semiconductor surface?
   (1) MPU (3) CPU
   (2) Microprocessor (4) All of the above

3. _____________ type of memory is volatile.
   (1) ROM (3) Both are volatile
   (2) RAM (4) Neither are volatile

4. A _____________ is comprised of eight bits.
   (1) Bit (3) Byte
   (2) Nibble (4) Word

5. Output devices provide binary information to the processor.
   (1) True (2) False

6. _____________ is not an input device.
   (1) Scanner (3) Mouse
   (2) Keyboard (4) Monitor

7. ____________ is used to transfer binary data and instructions.
   (1) Control lines (3) Address bus
   (2) Data bus (4) None of the above

8. How many bits can be stored in 1 K-byte (KB) of memory?
   (1) 1024 (3) 1000
   (2) 8192 (4) 255

9. In 8 K-byte (KB) of memory there are ____________ registers with the hexadecimal address of the last register being ____________.

(1) 8192, 8191H	(3) 8000, 1FFFH
(2) 8192, 2000H	(4) 8192, 1FFFH

10. How many registers can a 12 bit processor address?

(1) 12	(3) 2048
(2) 4096	(4) 1024

11. The last address of a memory chip is 07FFH. What is the size of the chip?

(1) 1 KB	(3) 4KB
(2) 2 KB	(4) None of the above

12. A range of memory extends from 00000H to 1FFFFH. How big is this range?

(1) 2 KB	(3) 6 KB
(2) 4 KB	(4) 8 KB

13. A microcontroller has a R/W memory range that starts at 2000H and ends at 21FFH. How many memory locations would this be?

(1) 512	(3) 511
(2) 1024	(4) 1023

14. Which bit indicates a negative number in the 8 bit MPU?

(1) Bit 0	(3) Bit 5
(2) Bit 7	(4) Bit 2

15. If 78_H was unsigned, what value would it have in decimal?

(1) 78	(3) 120
(2) 128	(4) 127

16. What would the hex value of -12 in decimal be?

(1) FF_H	(3) $F1_H$
(2) $0C_H$	(4) $F4_H$

17. The decimal number 138 would be __________ in hex and ___________ in binary.

(1) 76_H, 10001010_2	(3) $8A_H$, 10101010_2
(2) $8A_H$, 10001010_2	(4) 76_H, 10101010_2

18. ASCII is limited to 128 characters as it only covers 7 bits.

(1) True	(2) False

19. If the ASCII code is 5AH, what letter or character does it represent?

(1) a (lower case)	(3) Z (upper case)
(2) @	(4) %

20. Which bit differentiates an upper case ASCII letter from a lower case letter?
    (1) Bit 1 (3) Bit 3
    (2) Bit 7 (4) Bit 5

21. Assembly language is efficient in execution because ____________________?
    (1) It has separate memory storage for instructions
    (2) The translated code from assembly to machine does not have any overhead.
    (3) It uses masking to set upper and lower case letters
    (4) It really is not that efficient

22. The end result of an operation is 98H. What would the decimal value be if it were signed?
    (1) 152 (3) -98
    (2) -102 (4) -66

23. How many registers can a 21 bit processor address?
    (1) 2214896 (3) 2097152
    (2) 1487942 (4) None are correct

24. In 4 MB of memory there are ___________ registers, with the hex address of the last register being ___________.
    (1) 4194304, 3FFFFF (3) 4194304, 400000
    (2) 4194304, 4194303 (4) 4000000, 3FFFFF

25. An 8 bit processor would be able to represent the hex value -138 in decimal.
    (1) True (2) False

End of Examination

Lesson 8104-PIC18F Architecture and Instruction Set
Microcontrollers and Embedded Systems

Assignment Checklist

- ✓ Read Chapter 2 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Chapter 3 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Review Appendix A *PIC18FXXX/XXXX Instruction Set*
- ✓ Read Lesson 8104 in this Study Guide
- ✓ Take the Examination for this lesson

Chapter 2 Lecture/Discussion

PIC18F Microcontroller Families

- PIC microcontrollers are designed using the Harvard Architecture which includes:
  - Microprocessor unit (MPU)
  - Program memory for instructions
  - Data memory for data
  - I/O ports
  - Support devices such as timers

Microprocessor Unit

- Includes Arithmetic Logic Unit (ALU), Registers, and Control Unit
  - Arithmetic Logic Unit (ALU)
    - WREG – working register
    - Status register that stores flags
    - Instruction decoder
- Registers
  - Bank Select Register (BSR)
    - 4-bit register used in direct addressing the data memory
  - File Select Registers (FSRs)
    - 16-bit registers used as memory pointers in indirect addressing data memory
  - Program Counter (PC)
    - 21-bit register that holds the program memory address while executing programs
- Control unit
  - Provides timing and control signals to various Read and Write operations

PIC18F - Address Buses

- Address bus
  - 21-bit address bus for program memory addressing capacity: 2 MB of memory
  - 12-bit address bus for data memory addressing capacity: 4 KB of memory

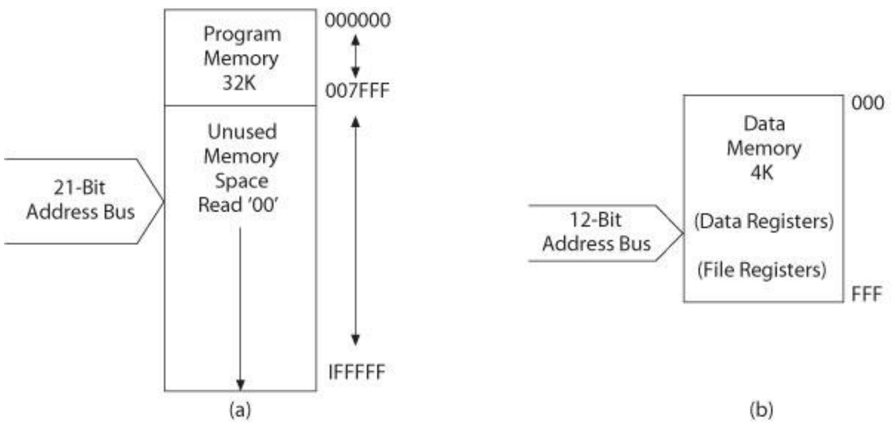
Data Bus and Control Signals

- Data bus
  - 16-bit instruction/data bus for program memory
  - 8-bit data bus for data memory
- Control signals
  - Read and Write

PIC18F452/4520 Memory

- Program Memory: 32 K
  - Address range: 000000 to 007FFFH
- Data Memory: 4 K
  - Address range: 000 to FFFH
- Data EEPROM
  - Not part of the data memory space
  - Addressed through special function registers

Note: The textbook has an error with the caption of Figure 2-3 listing it as PIC252



Program memory with addresses — Data memory with addresses

*Figure 2-3 PIC18F452/4520 Memory

PIC18F452 I/O Ports

- Five I/O ports
  - PORT A through PORT E
  - Most I/O pins are multiplexed
  - Generally have eight I/O pins with a few exceptions
  - Addresses already assigned to these ports in the design stage
  - Each port is identified by its assigned SFR

Processes of Data Transfer

- Parallel data transfer
- Serial data transfer
    - Status check
    - Interrupts
        - Requests from external devices
        - Requests from internal peripherals
        - Reset

MCU Support Devices

- Timers
- Master Synchronous Serial Port (MSSP)
- Addressable USART
- A/D converter
- Parallel Slave Port (PSP)
- Capture, Compare and PWM (CCP Module)

PIC18F Special Features

- Sleep mode
- Watchdog timer (WDT)
- Code protection
- In-circuit serial programming
- In-circuit debugger

PIC18F Instructions and Assembly Language

- Has 77 instructions
    - Earlier PIC family of microcontrollers have either 33 or 35 instructions
- In PIC18F instruction set, all instructions are 16-bit word length except four instructions that are 32-bit length

***Note: On page 53 the second line in the information box contains an error in the *Comments* column. In the textbook it reads “Set up PORTB as output” but it should have been “Set up PORTC as output” ***

Chapter 3 Discussion

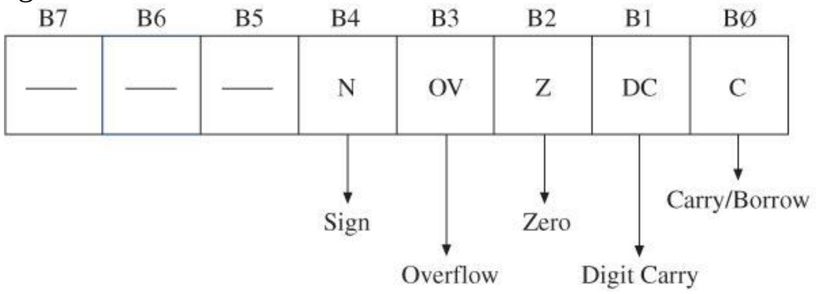
PIC18F Programming Model

- The representation of the internal architecture of a microprocessor, necessary to write assembly language programs
- Divided into two groups
  - ALU Arithmetic Logic Unit (ALU)
  - Special Function Registers (SFRs) from data memory

Registers

- WREG
  - 8-bit Working Register (equivalent to an accumulator)
- BSR: Bank Select Register (0 to F)
  - 8-bit Register
    - Only low-order four bits are used to provide MSB four bits of a12-bit address of data memory.
- STATUS: Flag Register

Flags in Status Register



- C (Carry/Borrow Flag): set when an addition generates a carry and a subtraction generates a borrow
- DC (Digit Carry Flag): also called Half Carry flag; set when carry generated from Bit3 to Bit4 in an arithmetic operation
- Z (Zero Flag): set when result of an operation is zero
- OV (Overflow Flag): set when result of an operation of signed numbers goes beyond seven bits
- N (Negative Flag): set when bit B7 is one of the result of an arithmetic /logic operation

File Select Registers (FSR)

- File Select Registers
  - FSR0, FSR1, and FSR2
  - FSR: composed of two 8-bit registers (FSRH and FSRL)
  - Used as pointers for data registers and associated with index (INDF) registers
  - Holds 12-bit address of data register
- Program Counter (PC)
  - 21-bit register functions as a pointer to program memory during program execution
- Table Pointer
  - 21-bit register used as a memory pointer to copy bytes between program memory and data registers

- Stack Pointer (SP)
  - Register used to point to the stack
- Stack
  - 31 word-sized registers used for temporary storage of memory addresses during execution of a program
- Special Function Registers (SFRs):
  - Data registers associated with I/O ports, support devices, and processes of data transfer

Introduction to PIC18 Instruction Set
****Note: On page 69 in Section 3.2 *Introduction to PIC18 Instruction Set* in line 4 the sentence begins with "instructions of the extended set…" and it contains an error according to the author. It should read, "instructions of the extended set are not discussed in the text."

- Includes 77 instructions; 73 one word (16-bit) long and remaining four two words (32-bit) long
- Divided into seven groups
  - Move (Data Copy) and Load
  - Arithmetic
  - Logic
  - Program Redirection (Branch/Jump)
  - Bit Manipulation
  - Table Read/Write
  - Machine Control

Instruction Format
- The PIC18F instruction format divided into four groups
  - Byte-Oriented operations
  - Bit-Oriented operations
  - Literal operations
  - Branch operations

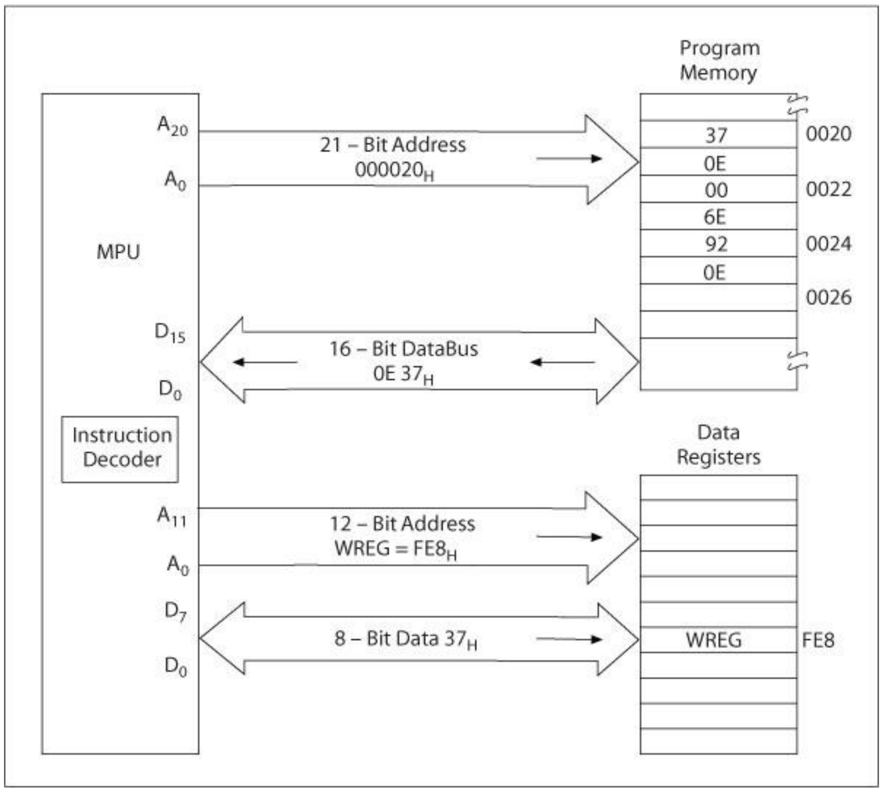
The execution of an instruction is important to understand so it is illustrated below as well as in the textbook.

Execution of an Instruction
Instruction: MOVLW 0x37 ; Load 37H in W

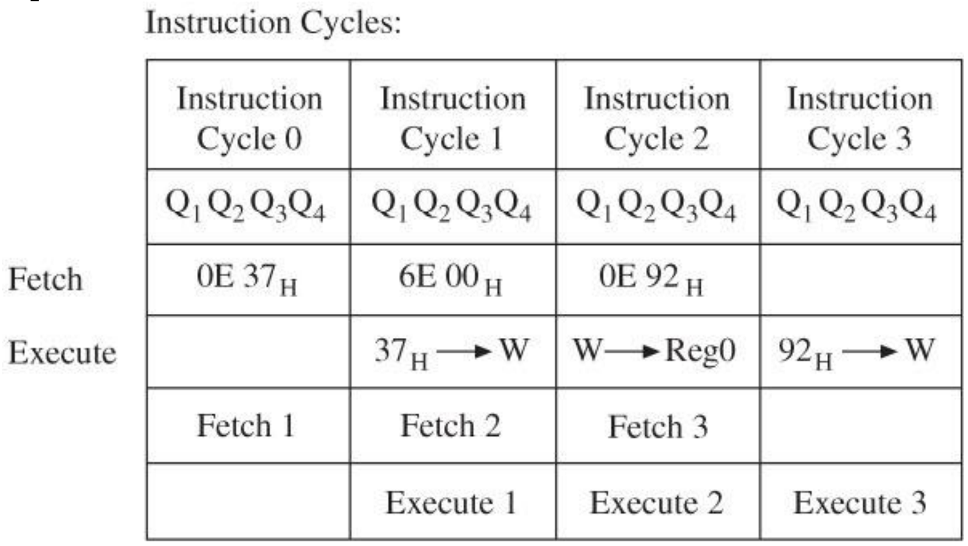
Memory Address	Hex Code	Mnemonics
000020	37	MOVLW 0x37
000021	0E	

Bus Contents during the Execution of Instruction

- Execution of MOVLW 0x37 instruction



Pipeline Fetch and Execution



Instruction Cycles:

	Instruction Cycle 0	Instruction Cycle 1	Instruction Cycle 2	Instruction Cycle 3
	$Q_1Q_2Q_3Q_4$	$Q_1Q_2Q_3Q_4$	$Q_1Q_2Q_3Q_4$	$Q_1Q_2Q_3Q_4$
Fetch	$0E\ 37_H$	$6E\ 00_H$	$0E\ 92_H$	
Execute		$37_H \rightarrow W$	$W \rightarrow Reg0$	$92_H \rightarrow W$
	Fetch 1	Fetch 2	Fetch 3	
		Execute 1	Execute 2	Execute 3

Lesson 8104 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. The _________________ is not a component of the MPU.
   (1) Microprocessor (3) Memory
   (2) EEPROM (4) I/O ports

2. The address bus is ____________________.
   (1) Unidirectional (3) Both, it depends on the instruction
   (2) Bidirectional (4) None of the above

3. What is the addressing capacity of an MPU with 19 address lines?
   (1) 5.12 KB (3) 512 KB
   (2) 51.2 KB (4) 5120 KB

4. How many address lines would be required to address 128 KB of memory?
   (1) 11 (3) 13
   (2) 15 (4) 17

5. A 14 bit processor has _____________ bits of data memory.
   (1) 524,288 (3) 131,072
   (2) 16,384 (4) 8,192

6. The MCU consists solely of the MPU and the I/O ports.
   (1) True (2) False

7. Harvard architecture in in the MPU means there is separate memory storage for instructions and data.
   (1) True (2) False

8. The W register in the PIC18 MCU is the same as the ___________ in other microcontrollers.
   (1) Program counter (3) Accumulator
   (2) Register (4) Control unit

9. How wide is the BSR?
   (1) Four bits (3) Sixteen bits
   (2) Eight bits (4) Twenty-four bits

10. The FSR registers are 21 bits wide in order to hold the 12 bit address of the data register and the 4 bit address of the memory bank.
    (1) True (2) False

11. How many registers can the FSR access?
    (1) 1024 (3) 4096
    (2) 2048 (4) 8192

12. What is the address range (beginning to ending addresses) of the special function registers in the PIC18F MCU?
    (1) 00_H to FF_H (3) 80_H to FF_H
    (2) 00_H to 80_H (4) None of the above

13. The mode a fax machine transfer data is __________________?
    (1) serial (3) both serial and parallel
    (2) parallel (4) none of the above

14. The interrupt process is initialized by the _______________.
    (1) CPU (3) RAM
    (2) peripheral device (4) instruction

15. The interrupt process is _________________ initiated.
    (1) internally (3) serial
    (2) externally (4) parallel

16. Which of these will not generate an interrupt for the PIC18F MCU?
    (1) Peripheral devices (3) A/D Converters
    (2) Timers (4) RAM

17. Which is not a typical support device included on the MCU chip?
    (1) Timers (3) A/D converter module
    (2) Flash memory (4) Serial module

The following code is used to answer Questions 18 to 21

```
MOVLW     0xFA

ADDLW     0x38
```

18. After the **MOVLW 0xFA** is executed what is the contents of the W register?
    (1) 00_H (3) 38_H
    (2) 32_H (4) FA_H

19. After the **ADDLW 0x38** is executed what is the contents of the W register?
    (1) 00_H (3) 38_H
    (2) 32_H (4) FA_H

20. If the numbers were unsigned, what would be the result of the addition?

(1) 138_H	(3) 38_H
(2) FA_H	(4) 32_H

21. If the numbers are signed, what would be the result?

(1) -32_H	(3) -38_H
(2) FA_H	(4) $+32_H$

22. A one-word branch instruction requires __________ cycles for execution.

(1) 1	(3) 3
(2) 2	(4) 4

23. The interrupt process cannot be disabled.

(1) True	(2) False

24. Multiplexing means I/O ports are bidirectional.

(1) True	(2) False

25. Peripheral devices can process data faster than the MPU can send it.

(1) True	(2) False

End of Examination

Lesson 8106-Programming and Problem Solving
Microcontrollers and Embedded Systems

Assignment Checklist

- ✓ Read Chapter 4 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Lesson 8106 in this Study Guide
- ✓ Take the Examination for this lesson

Chapter 4 Lecture/Discussion

Flowcharting

- Flowchart
  - – A graphical representation of processes (tasks) to be performed and the sequence to be followed in solving computational problem
- Symbols shown in Figure 4.1 are used commonly to draw a flowchart

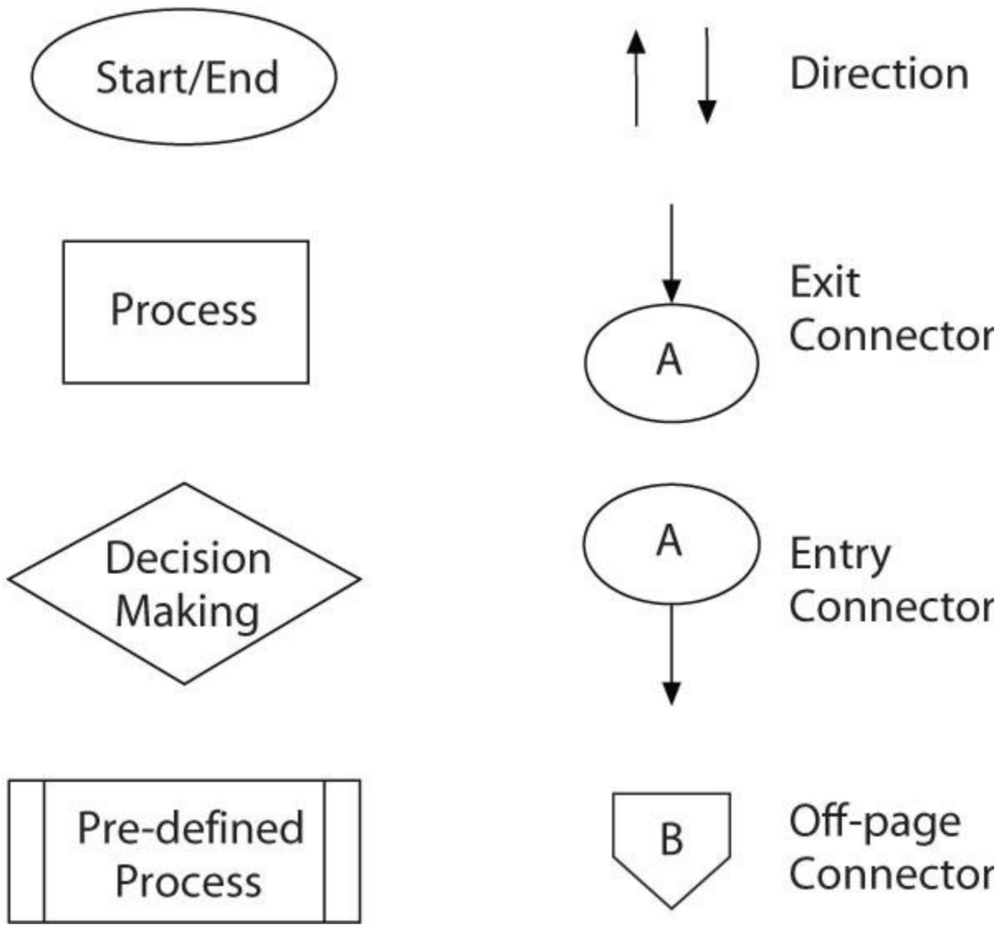
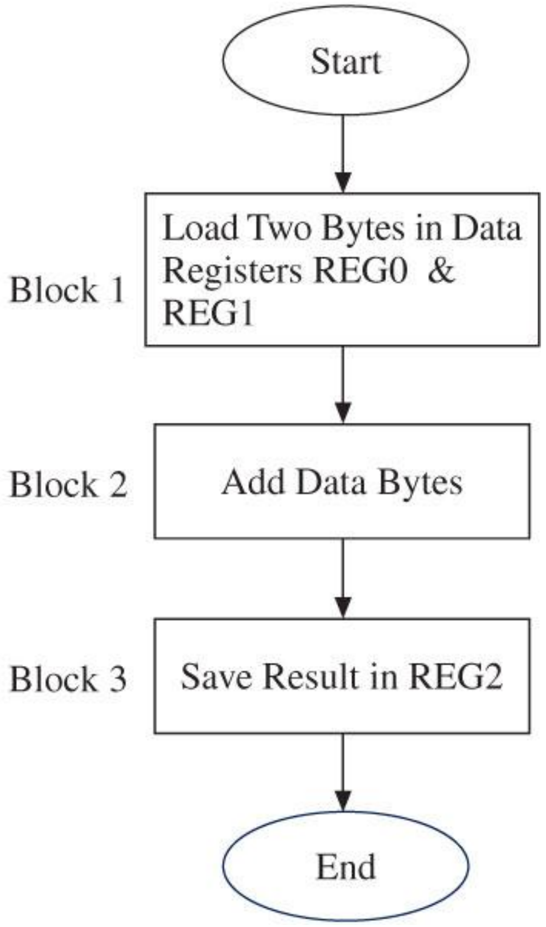


Figure 4-1 Flowchart Symbols

Example 4.1

- Write instructions to load two bytes (37H and 92H) in data registers REG0 and REG1. Add the bytes and store the sum in REG2.
- Steps
  - Load the two bytes in data registers REG0 and REG1.
  - Add the bytes.
  - Save the sum in data register REG2.

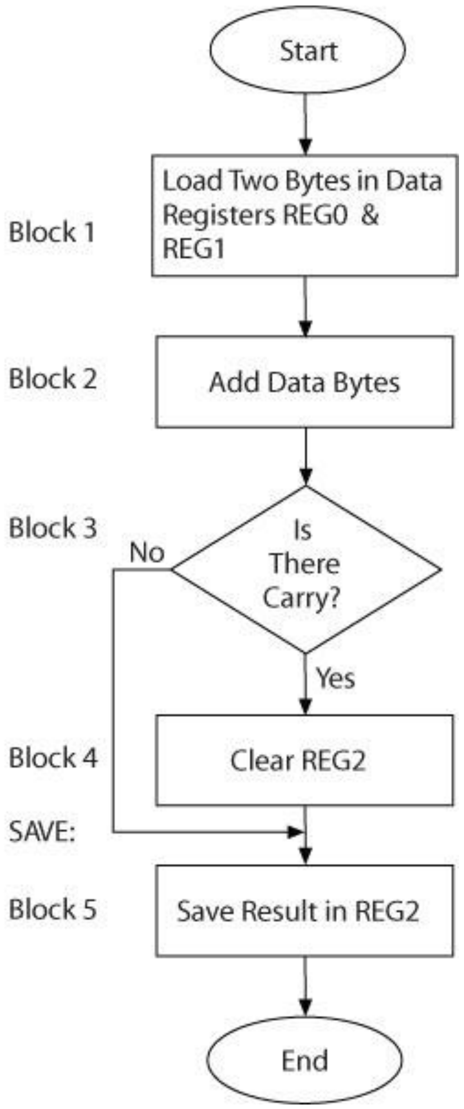


Steps in Writing and Executing Assembly Language Program

- Analyze the problem.
- Draw a flowchart.
- Convert the flowchart in mnemonics.
- Look up Hex code and assign memory addresses.
- Enter the Hex code into memory of a lab training board.
- Execute the program.
- Debug the program if necessary.

Illustrative Program: Addition With Carry Check

- Write instructions to load two bytes, Byte1 (F2H) and Byte2 (32H), in data registers REG0 and REG1 respectively and add the bytes.
- If the sum generates a carry, clear the data register REG2; otherwise, save the sum in REG2.



Integrated Development Environment (IDE)

- Steps in using IDE
  - Editing
  - Assembling
  - Linking
  - Downloading
  - Executing
  - Simulation
  - Debugging

Writing a Program Using an Assembler

- The assembly language program includes:
  - Program in mnemonics
  - Assembler directives
  - Comments

Assembly Language Format

- Typical statement of an assembly language source code has four fields:
  - Label
  - Opcode (operation code)
  - Operand (data, register, or memory address to be operated on)
  - Comment
- Format example

```
Label       Opcode      Operand     Comment
START:      MOVLW       0xF2        ;Load F2H in W
        ↑            ↑        ↑     ↑
    Space         Space    Space  Semicolon
    Or Colon
```

Assembler Directives

• ORG	Origin
• END	End of assembly
• EQU	Equate
• SET	Defines an assembler variable
• #INCLUDE	Include resources from available library
• RADIX	Number format
• DB	Define byte
• DW	Define word
• CBLOCK	Define a block of constants
• ENDC	End of block of constants
• RES	Reserve memory

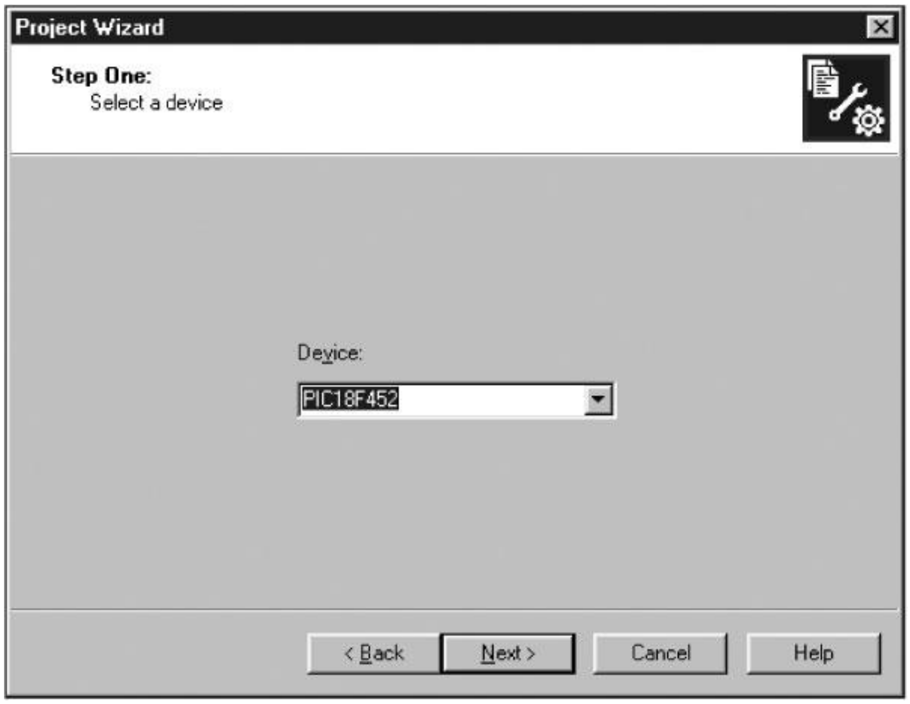
Using MPLAB IDE to Write, Assemble, and Build Project

- Write source code using MPLAB editor.
- Create a new project.
- Select language tool suite.
- Name your project.
- Add files to assemble.
- Build the project.

A Laboratory Study Guide will cover the following procedure along with several simulation exercises

Using MPLAB IDE

- To create a new project
  - Step 1: Open MPLAB IDE→ Select Project → Project Wizard → Select Device→ PIC18F452 →Next

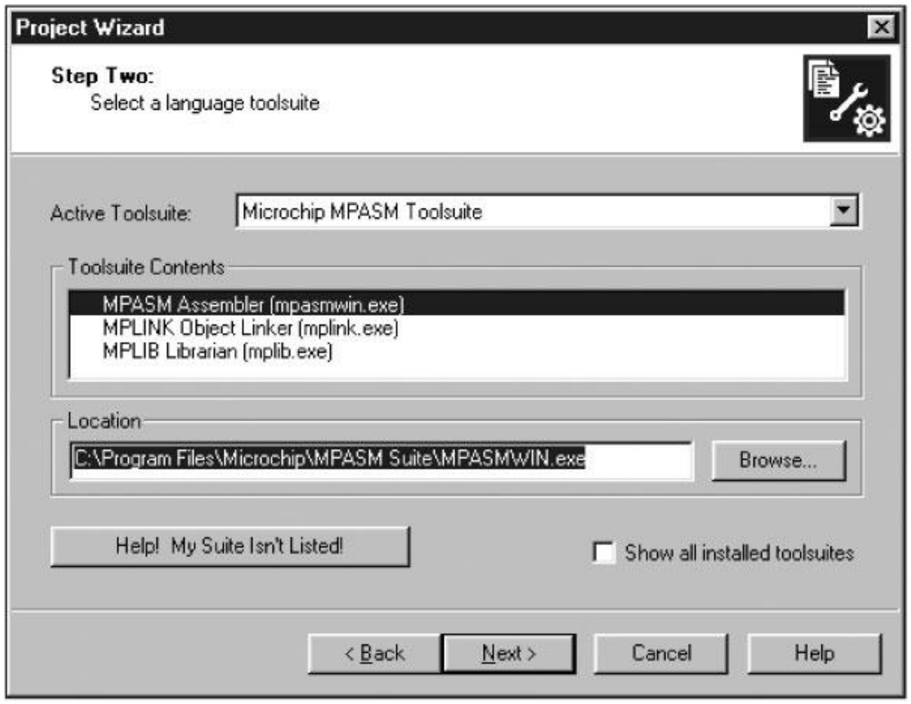


Project Wizard

Step One:
Select a device

Device: PIC18F452

  - Step 2: Select a Language Toolsuite: Microchip MPASM Toolsuite →Next



Project Wizard

Step Two:
Select a language toolsuite

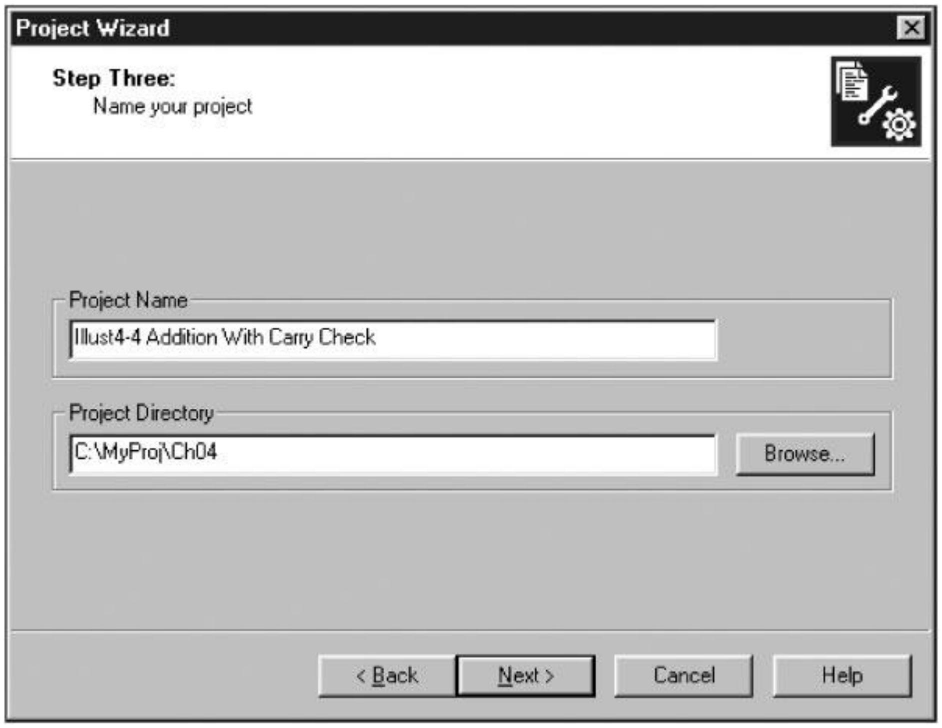
Active Toolsuite: Microchip MPASM Toolsuite

Toolsuite Contents
MPASM Assembler (mpasmwin.exe)
MPLINK Object Linker (mplink.exe)
MPLIB Librarian (mplib.exe)

Location
C:\Program Files\Microchip\MPASM Suite\MPASMWIN.exe

☐ Show all installed toolsuites

– Step 3. Name Your Project: Illust4-4 Addition with Carry Check
Browse → MyProj\Ch04 → Next

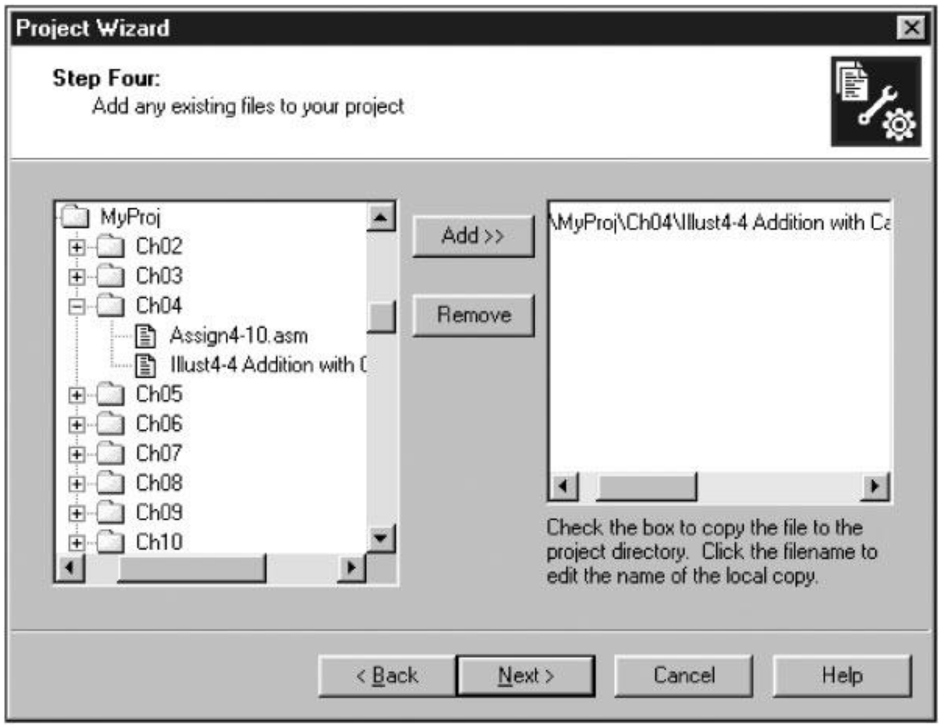


Step Three:
Name your project

Project Name: Illust4-4 Addition With Carry Check

Project Directory: C:\MyProj\Ch04

– Step 4: Add → Add Source Files → Next

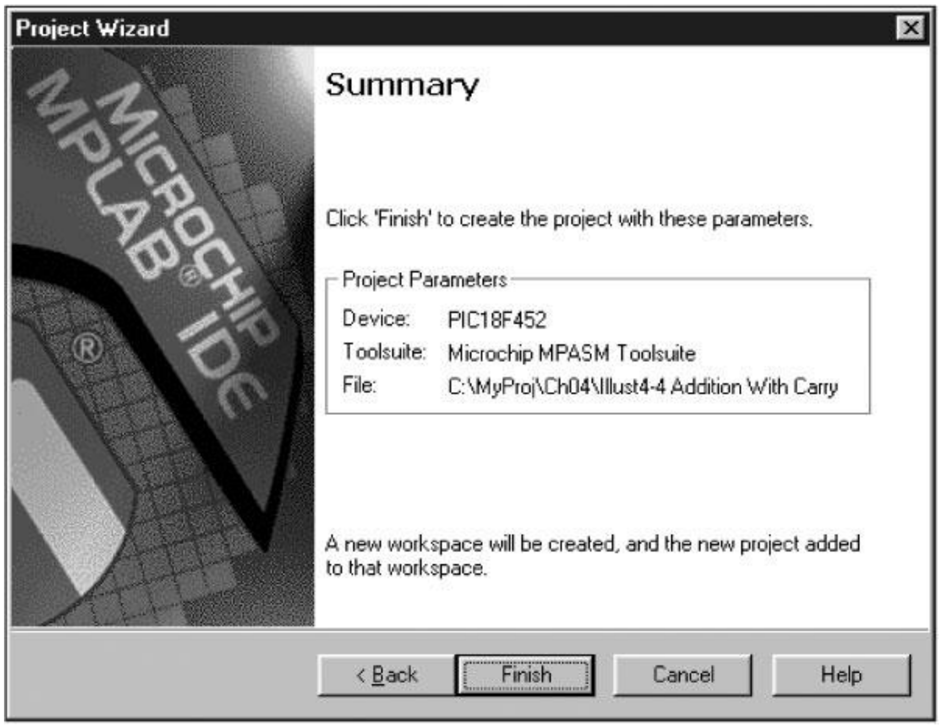


Step Four:
Add any existing files to your project

\MyProj\Ch04\Illust4-4 Addition with Ca

Check the box to copy the file to the project directory. Click the filename to edit the name of the local copy.

– Summary → Finish



Project Wizard

Summary

Click 'Finish' to create the project with these parameters.

Project Parameters

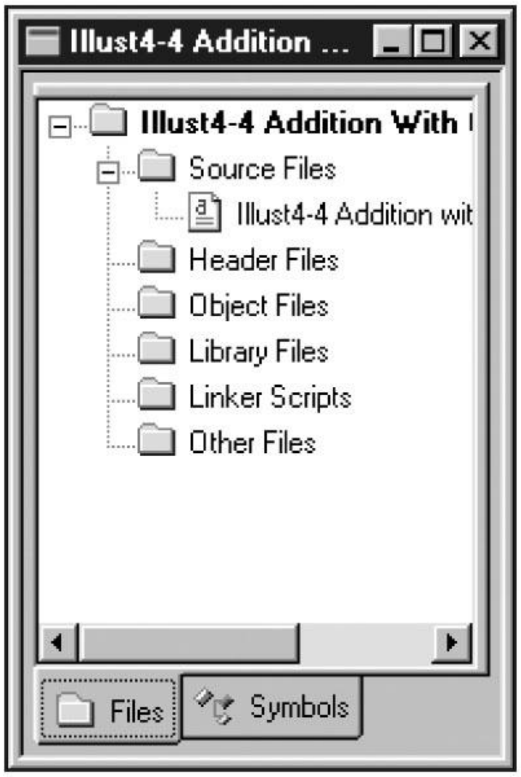
Device: PIC18F452

Toolsuite: Microchip MPASM Toolsuite

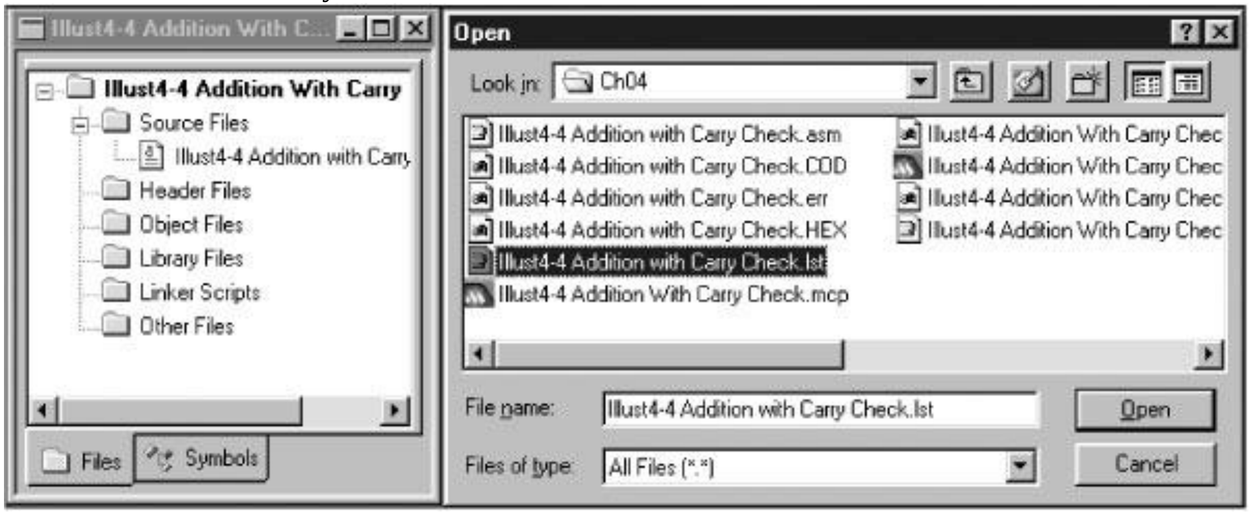
File: C:\MyProj\Ch04\Illust4-4 Addition With Carry

A new workspace will be created, and the new project added to that workspace.

Project Window



List of Files Generated by MPLAB Assembler


Illust4-4 Addition With C...
Illust4-4 Addition With Carry
Source Files
Illust4-4 Addition with Carry
Header Files
Object Files
Library Files
Linker Scripts
Other Files
Files
Symbols
Open
Look in: Ch04
Illust4-4 Addition with Carry Check.asm
Illust4-4 Addition with Carry Check.COD
Illust4-4 Addition with Carry Check.err
Illust4-4 Addition with Carry Check.HEX
Illust4-4 Addition with Carry Check.lst
Illust4-4 Addition With Carry Check.mcp
Illust4-4 Addition With Carry Chec
Illust4-4 Addition With Carry Chec
Illust4-4 Addition With Carry Chec
Illust4-4 Addition With Carry Chec
File name: Illust4-4 Addition with Carry Check.lst
Files of type: All Files (*.*)
Open
Cancel

Understanding the List File

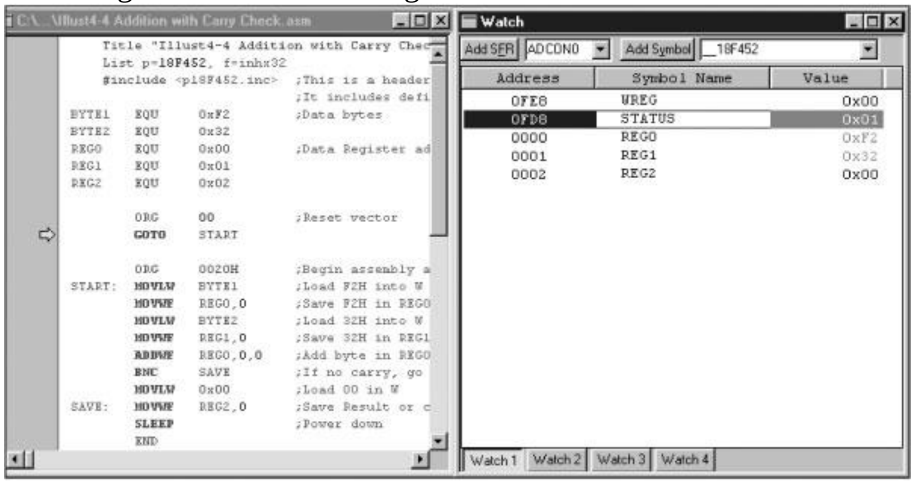
- List file generated primarily for documentation
- Includes seven columns
  - Memory addresses where binary code is stored
  - Hex code
  - Line numbers
  - Contents of source file
    - Labels
    - Opcode
    - Operands
    - Comments

On Page 112 there are a couple of errors in the program at the top of the page. On Program Line 000024 the Text portion has *;Load 32H into W* and the missing information is *;Load 32H into W* ***register so register is missing. And the next line below this has *register26 6E01* and the word register should not have been there but the corrected information is *000026 6E01*.

Executing a Program Using Simulator

- Steps in setting up MPLAB simulator
  - Select Debugger → Select tool→ MPLABSIM
  - Select Debugger → Settings → Change frequency if necessary
  - Select View → Watch → Add registers to observe

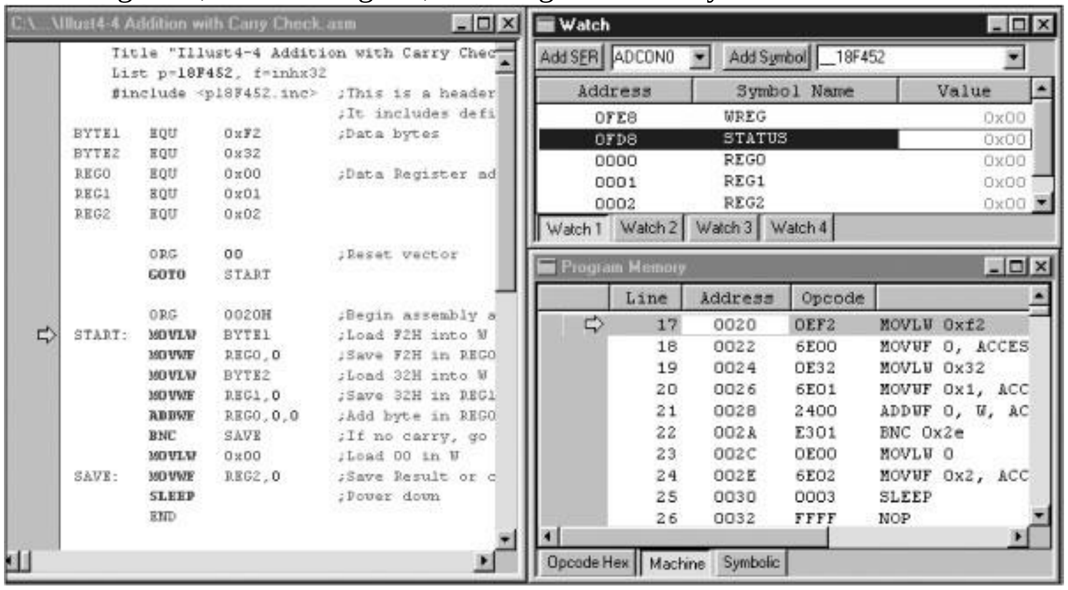
View Registers and Source Program in MPLAB Simulator



```
    Title "Illust4-4 Addition with Carry Chec
    List p=18F452, f=inhx32
    #include <p18F452.inc>  ;This is a header
                            ;It includes defi
BYTE1   EQU     0xF2        ;Data bytes
BYTE2   EQU     0x32
REG0    EQU     0x00        ;Data Register ad
REG1    EQU     0x01
REG2    EQU     0x02

        ORG     00          ;Reset vector
        GOTO    START

        ORG     0020H       ;Begin assembly a
START:  MOVLW   BYTE1       ;Load F2H into W
        MOVWF   REG0,0      ;Save F2H in REG0
        MOVLW   BYTE2       ;Load 32H into W
        MOVWF   REG1,0      ;Save 32H in REG1
        ADDWF   REG0,0,0    ;Add byte in REG0
        BNC     SAVE        ;If no carry, go
        MOVLW   0x00        ;Load 00 in W
SAVE:   MOVWF   REG2,0      ;Save Result or c
        SLEEP               ;Power down
        END
```

Address	Symbol Name	Value
0FE8	WREG	0x00
0FD8	STATUS	0x01
0000	REG0	0xF2
0001	REG1	0x32
0002	REG2	0x00

View Registers, Source Program, and Program Memory in MPLAB Simulator



```
    Title "Illust4-4 Addition with Carry Chec
    List p=18F452, f=inhx32
    #include <p18F452.inc>  ;This is a header
                            ;It includes defi
BYTE1   EQU     0xF2        ;Data bytes
BYTE2   EQU     0x32
REG0    EQU     0x00        ;Data Register ad
REG1    EQU     0x01
REG2    EQU     0x02

        ORG     00          ;Reset vector
        GOTO    START

        ORG     0020H       ;Begin assembly a
START:  MOVLW   BYTE1       ;Load F2H into W
        MOVWF   REG0,0      ;Save F2H in REG0
        MOVLW   BYTE2       ;Load 32H into W
        MOVWF   REG1,0      ;Save 32H in REG1
        ADDWF   REG0,0,0    ;Add byte in REG0
        BNC     SAVE        ;If no carry, go
        MOVLW   0x00        ;Load 00 in W
SAVE:   MOVWF   REG2,0      ;Save Result or c
        SLEEP               ;Power down
        END
```

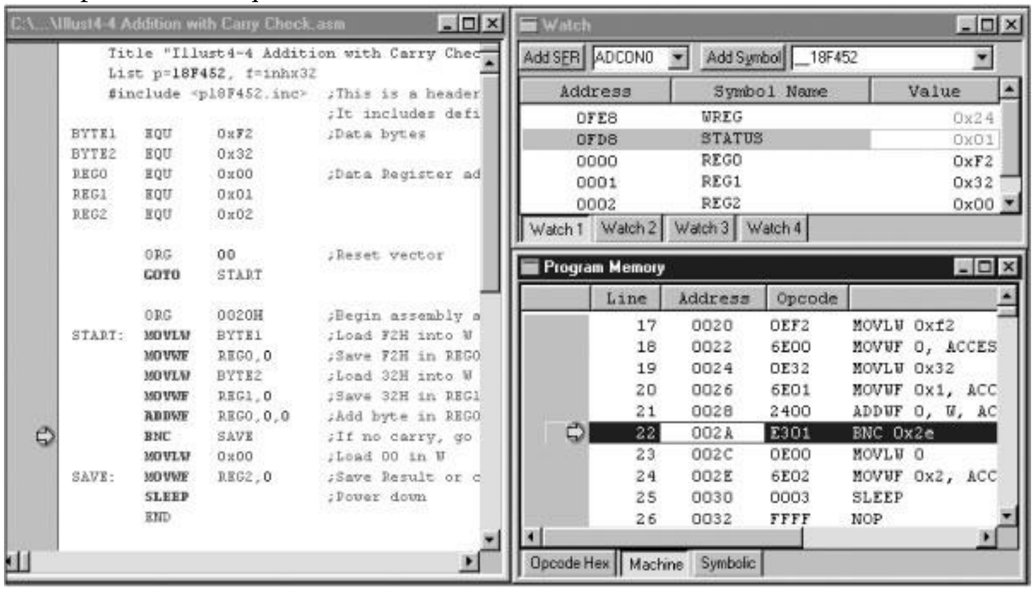
Address	Symbol Name	Value
0FE8	WREG	0x00
0FD8	STATUS	0x00
0000	REG0	0x00
0001	REG1	0x00
0002	REG2	0x00

Line	Address	Opcode	
17	0020	0EF2	MOVLW 0xf2
18	0022	6E00	MOVWF 0, ACCES
19	0024	0E32	MOVLW 0x32
20	0026	6E01	MOVWF 0x1, ACC
21	0028	2400	ADDWF 0, W, AC
22	002A	E301	BNC 0x2e
23	002C	0E00	MOVLW 0
24	002E	6E02	MOVWF 0x2, ACC
25	0030	0003	SLEEP
26	0032	FFFF	NOP

Debugging a Program

- Single-step technique
  - Enables user to execute one instruction at a time and observe registers for expected results
- Breakpoint technique
  - Enables user to execute a group of instructions at a time and observe registers for expected results
- Tracing code
  - MPLAB can track execution of each instruction and display data which can be examined for errors

Breakpoint Technique

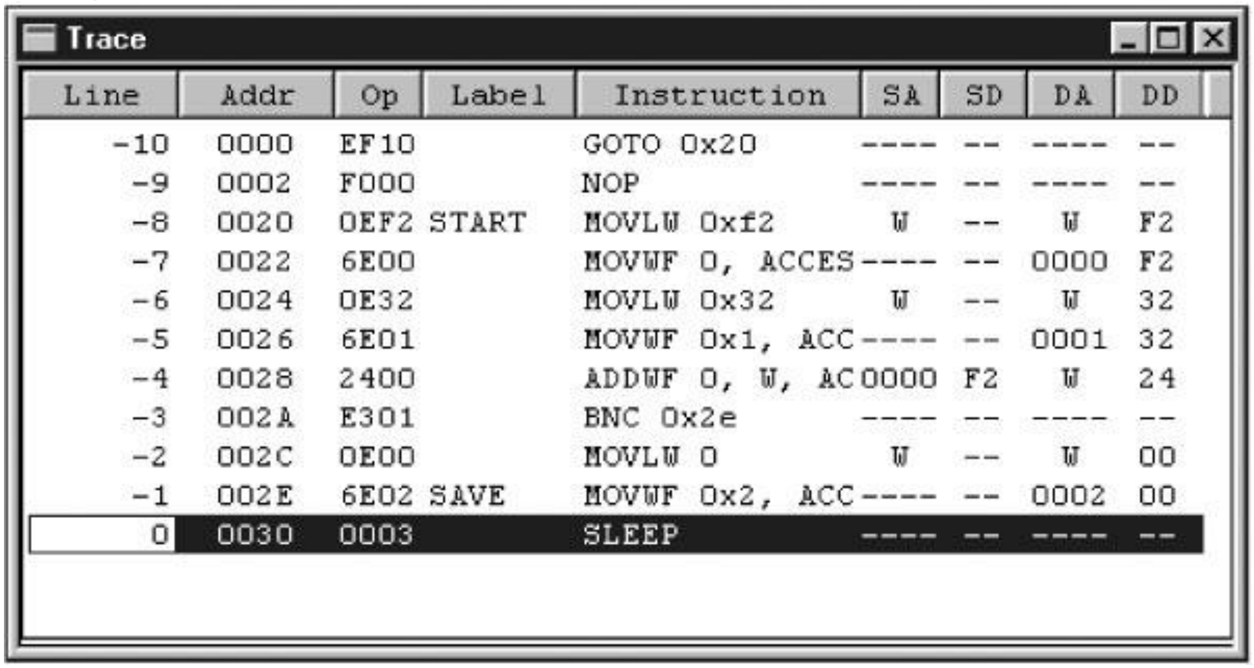


```
    Title "Illust4-4 Addition with Carry Chec
    List p=18F452, f=inhx32
    #include <p18F452.inc>  ;This is a header
                            ;It includes defi
BYTE1   EQU     0xF2        ;Data bytes
BYTE2   EQU     0x32
REG0    EQU     0x00        ;Data Register ad
REG1    EQU     0x01
REG2    EQU     0x02

        ORG     00          ;Reset vector
        GOTO    START

        ORG     0020H       ;Begin assembly a
START:  MOVLW   BYTE1       ;Load F2H into W
        MOVWF   REG0,0      ;Save F2H in REG0
        MOVLW   BYTE2       ;Load 32H into W
        MOVWF   REG1,0      ;Save 32H in REG1
        ADDWF   REG0,0,0    ;Add byte in REG0
        BNC     SAVE        ;If no carry, go
        MOVLW   0x00        ;Load 00 in W
SAVE:   MOVWF   REG2,0      ;Save Result or c
        SLEEP               ;Power down
        END
```

Address	Symbol Name	Value
0FE8	WREG	0x24
0FD8	STATUS	0x01
0000	REG0	0xF2
0001	REG1	0x32
0002	REG2	0x00

Line	Address	Opcode	
17	0020	0EF2	MOVLW 0xf2
18	0022	6E00	MOVWF 0, ACCES
19	0024	0E32	MOVLW 0x32
20	0026	6E01	MOVWF 0x1, ACC
21	0028	2400	ADDWF 0, W, AC
22	002A	E301	BNC 0x2e
23	002C	0E00	MOVLW 0
24	002E	6E02	MOVWF 0x2, ACC
25	0030	0003	SLEEP
26	0032	FFFF	NOP

Tracing Code



Line	Addr	Op	Label	Instruction	SA	SD	DA	DD
-10	0000	EF10		GOTO 0x20	----	--	----	--
-9	0002	F000		NOP	----	--	----	--
-8	0020	0EF2	START	MOVLW 0xf2	W	--	W	F2
-7	0022	6E00		MOVWF 0, ACCES	----	--	0000	F2
-6	0024	0E32		MOVLW 0x32	W	--	W	32
-5	0026	6E01		MOVWF 0x1, ACC	----	--	0001	32
-4	0028	2400		ADDWF 0, W, AC	0000	F2	W	24
-3	002A	E301		BNC 0x2e	----	--	----	--
-2	002C	0E00		MOVLW 0	W	--	W	00
-1	002E	6E02	SAVE	MOVWF 0x2, ACC	----	--	0002	00
0	0030	0003		SLEEP	----	--	----	--

Lesson 8106 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. A(n) ____________ is a graphical representation of processes to be performed and the sequence to be followed in solving computational problem.
   (1) program (3) flowchart
   (2) flow diagram (4) assembler

2. A(n) ___________ is a program that converts the mnemonics of a source program into binary machine language, equivalent Hex codes, and assigns memory address to these codes.
   (1) program (3) flowchart
   (2) flow diagram (4) assembler

3. Which is not part of an assembly language statement?
   (1) Operation (3) Comments
   (2) Operand (4) Pseudocode

Questions 4, 5 and 6 are based on the following program.

```
START:      MOVLW       0x67
            ADDLW       0x33
            SLEEP
```

4. What would be the contents of the W register after the program has been run?
   (1) 100_H (3) $9A_H$
   (2) AA_H (4) 34_H

5. After the addition has taken place, what flags would be set?
   (1) N and OV (3) N, OV, Z and C
   (2) N, OV, and Z (4) No flags were set

6. The STATUS register would be expressed as which byte?
   (1) 18_H (3) $1D_H$
   (2) $1C_H$ (4) 00_H

Questions 7, 8 and 9 are based on the following program.

```
START:      MOVLW      0x42
            SUBLW      0x33
            SLEEP
```

7. What would be the contents of the W register after the program has been run?
   (1) 09_H (3) $F1_H$
   (2) $0F_H$ (4) FF_H

8. What flags would be set after the subtraction has taken place?
   (1) N and OV (3) N and DC
   (2) N and Z (4) All of the flags would be set

9. The STATUS register would be expressed as which byte?
   (1) 18_H (3) 12_H
   (2) 14_H (4) $1F_H$

10. Which term is synonymous with pseudocode?
   (1) Instructions (3) Label
   (2) Pseudo-ops (4) Operand

11. An assembly language program can contain more than one ORG statement?
   (1) True (2) False

12. A List file of an assembled program includes ______________________________?
   (1) the source code only
   (2) source code with comments
   (3) source code with comments, Hex code, and memory addresses
   (4) none of the above are in a List file

13. The flowchart symbol for the beginning of a program is the ________________?
   (1) Rectangle (3) Oval
   (2) Diamond (4) Circle with Arrow

14. The flowchart symbol for the decision point of a program is the ________________?
   (1) Rectangle (3) Oval
   (2) Diamond (4) Circle with Arrow

15. The off-page connector is shown with a letter "A"?
   (1) True (2) False

End of Examination

Lesson 8108-Branch Instructions, Logic, and Bit Manipulation
Microcontrollers and Embedded Systems

Assignment Checklist

- ✓ Read Chapter 5 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Chapter 6 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Lesson 8108 in this Study Guide
- ✓ Take the Examination for this lesson

<u>Chapter 5 Lecture/Discussion</u>

Data Copy (Move) and Set/Clear Operations
The data copy operations are classified as:

- Loading 8-bit data directly in WREG
- Copying data between WREG and data (file) register including I/O ports
- Copying data from one data (file) register to another data (file) register
- Clearing or setting all data bits in data (file) register
- Exchanging low-order four bits (nibble) with high-order four bits in data (file) register

Addressing Modes

- A way of specifying of an operand
  - Direct addressing
    - The operand is a part of the instruction
  - Indirect addressing
    - An address is specified in a register and the MPU looks up the address in that register

*****Understanding the various instruction sets is very important to understanding coding. You do not need to memorize the instruction sets but simply have a general idea of the operation as it helps in programming. Appendix A is used for reference so that way you don't have to memorize every instruction set.*

Using File Select Registers (FSRs) as Pointers to Data Registers

- Three registers: FSR0, FSR1, and FSR2
- Each can be used in five different formats:
  - INDF0: Use FSR0 as pointer
  - POSTINC0: Use FSR0 as pointer and increment FSR0
  - POSTDEC0: Use FSR0 as pointer and decrement SR0
  - PREINC0: Increment FSR0 first and use as pointer
  - PLUSW0: Add W to FSR0 and use as pointer

Using Table Pointers to Copy Data from Program Memory into Table Latch

- TBLRD*
  - Copy from Program Memory into Table Latch Using Table Pointer
- TBLRD*+
  - Copy from Program Memory into Table Latch and Increment Table Pointer

- TBLRD*-
  - Copy from Program Memory into Table Latch and Decrement Table Pointer
- TBLRD+*
  - Increment Table Pointer first and then copy from Program Memory into Table Latch

Using Table Pointers to Copy Data from Table Latch into Program Memory
- TBLWT*
  - Copy from Table Latch into Program Memory Using Table Pointer
- TBL WT*+
  - Copy from Table Latch into Program Memory and Increment Table Pointer
- TBL WT*-
  - Copy from Table Latch into Program Memory and Decrement Table Pointer
- TBLWT+*
  - Increment Table Pointer first and then copy from Table Latch into Program Memory

Arithmetic Operations
- The PIC18F MPU performs the following arithmetic operations:
  - Add
  - Subtract
  - Multiply
  - Complement (1s and 2s)
  - Increment
  - Decrement

***<u>Example 5.4 on Page 136</u> contains an error in the **Results:** section. It should read as follows:

```
WREG        0 1 0 0 1 1 1 1  (4FH)
+
REG20       0 1 0 0 1 0 0 0  (48H)
Carry       1     1
            -----------------------------------
            1 0 0 1  0 1 1 1  (97H)
```

The two zeros in bold were missing.

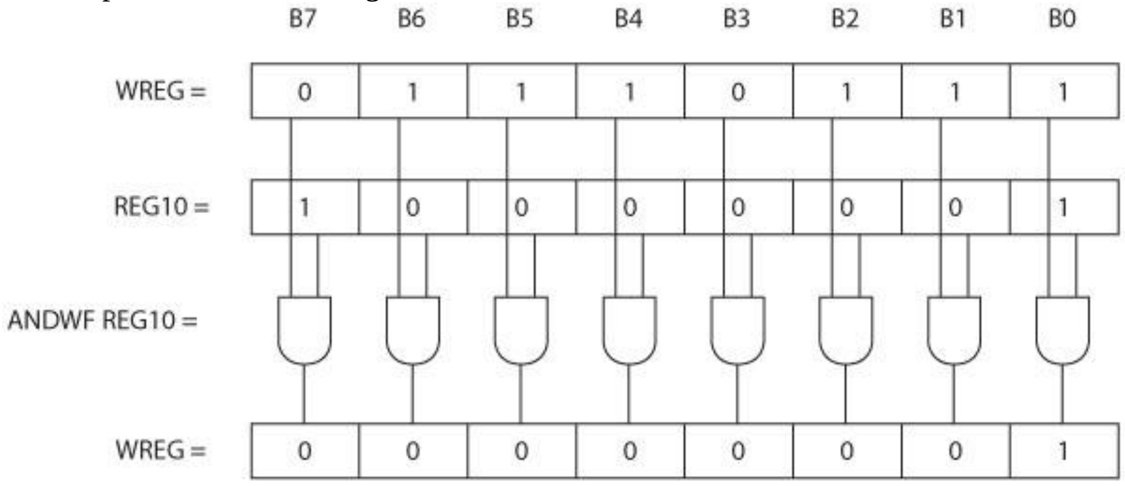
****On Page 156 the 5.6.3 Program contains a few errors. The second line of the program has:
COUNTER EQU 0x010 when it should be COUNTER EQU 0x01
Line 4 has GOTO START0 when it should be GOTO START
Line 5 has ORG 0x200 when it should be ORG 0x20

****On Page 164 the 5.7.3 Program contains an error. Toward the end under SKIP: line 3 has:
MOVWF ORTC when it should be MOVWF PORTC

Chapter 6 Lecture/Discussion

Logic Instructions: AND, IOR (Inclusive OR), and XOR (Exclusive OR)

AND Operations of Two Registers



In Example 6.1 we are performing an AND instruction.

Instruction ANDLW 0x38

WREG = 1 0 0 0 0 1 1 1 = 87_H

AND

8-bit Literal = 0 0 1 1 1 0 0 0 = 38_H

WREG = 0 0 0 0 0 0 0 0 = 0 0 Flag Status: N = 0, Z = 1

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Example 6.3 is an IOR and XOR

WREG = 1 0 0 0 0 0 1 0 = 82_H

IOR

Setting Byte = 0 0 0 0 0 0 1 1 = 03_H

WREG = 1 0 0 0 0 0 1 1 = 83_H

XOR

Toggling Byte = 1 1 0 0 0 0 0 0 = $C0_H$

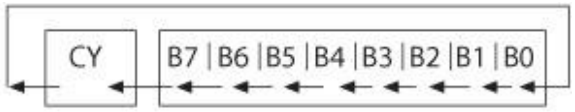
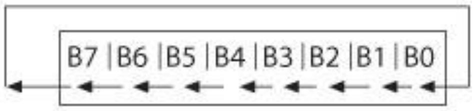
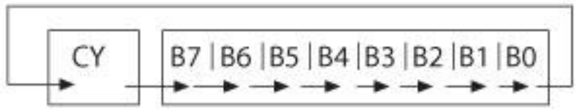
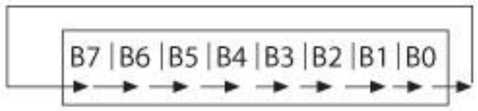
WREG = 0 1 0 0 0 0 1 1 = 43_H

Bit Rotation

- The instruction set includes four instructions that can shift a bit to the adjacent position—either left or right.
- The instructions are further classified as 8-bit rotation and 9-bit rotation.
  - In 9-bit rotation, carry flag becomes the ninth bit.

Another description of Bit Rotation is a Shift Register.

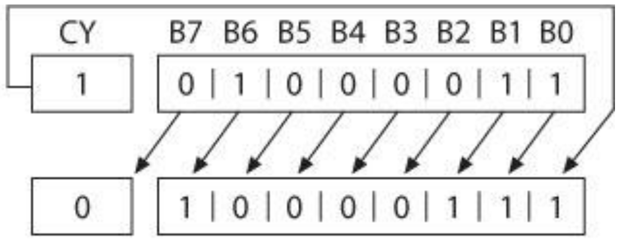
Rotate Instructions are:

RLCF F, d, a	Rotate Left through Carry If d = 1, save result in F, and if d = 0, save in W	
RLNCF F, d, a	Rotate Left with No Carry If d = 1, save result in F, and if d = 0, save in W	
RRCF F, d, a	Rotate Right through Carry If d = 1, save result in F, and if d = 0, save in W	
RRNCF F, d, a	Rotate Right with No Carry If d = 1, save result in F, and if d = 0, save in W	

Example 6.3—Rotate Left and Register Contents

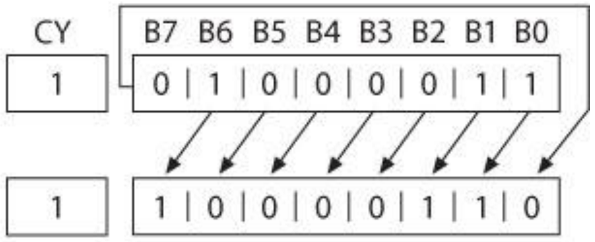
REG1: Initial Contents — REG1 = 43H

RLCF REG1, 1, 0
Rotate Left through Carry
Save in REG1 because d = 1 — REG1 = 87H

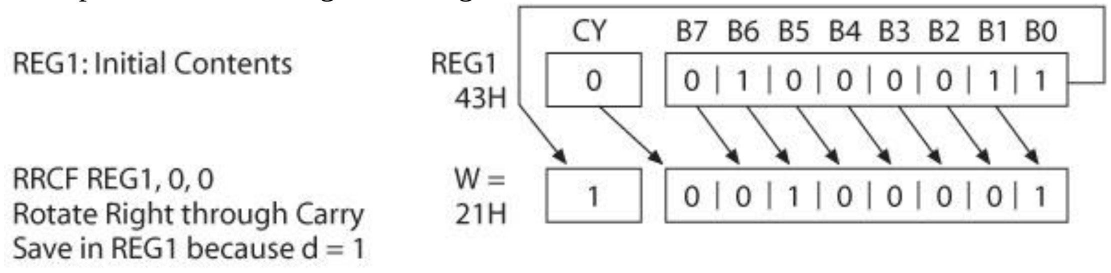


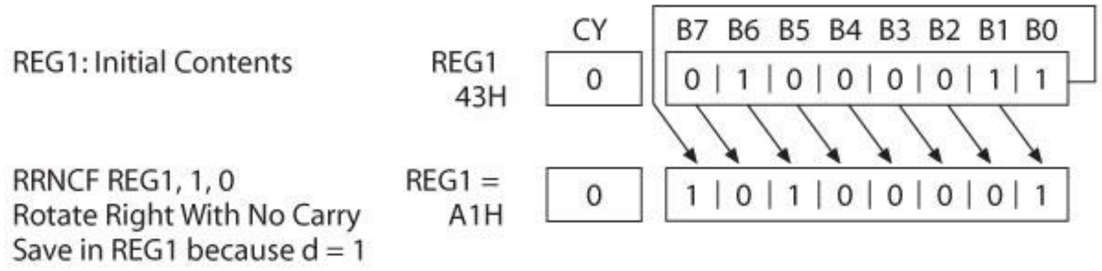
REG1: Initial Contents — REG1 = 43H

RLNCF REG1, 0, 0
Rotate Left With No Carry
Save in W because d = 0 — W = 86H



Example 6.7—Rotate Right and Register Contents





****On the top of Page 193, under section 6.3-Multiply and Divide Operations, the last line of the instructions has an extra L in MULLWF F, a and its example. It should be:

MULWF F, a: MULWF REG1,0

Division

- The PIC18F instruction set does not include any instruction that divides two numbers.
- Commonly used algorithms to divide numbers
  - Using the instruction Rotate Right
  - Repeated Subtraction

****The 6.5.3 Program begins on Page 199 and continues to Page 200. On Page 200 the program contains an error. The 5th line up from END has a wrap around from the comments line. The HI_SUM goes with the comment line being part of *decrement HI_SUM*.

Lesson 8108 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. If two unsigned numbers $3F_H$ and 47_H are added together what is their result in the W register?
   (1) 86_H (3) 134_H
   (2) 206_H (4) None of these are correct

2. Is the result from Question 1 Positive or Negative?
   (1) Positive (2) Negative

3. The STATUS register would have the Hex value of ____________ after the addition of the two unsigned numbers in Question 1.
   (1) 18_H (3) 00_H
   (2) 08_H (4) Cannot be determined

4. If the numbers in Question 1 were *signed* numbers, the result would be the same.
   (1) True (2) False

5. If the instructions were writing for subtracting the byte $7F_H$ from the byte 25_H what would be the final result after the program was executed?
   (1) $5A_H$ (3) $A6_H$
   (2) $A5_H$ (4) None of these are correct

6. Which flag or flags would be set after the subtraction in Question 5 was completed?
   (1) N (3) OV, DC and C
   (2) N and OV (4) All of the flags would be reset

7. If two bytes 78_H and $F2_H$ were added together, the W register would be ___________?
   (1) $22A_H$ (3) $6A_H$
   (2) $16A_H$ (4) Cannot be determined

8. How many times is LOOP1 executed in the following instructions?

```
          MOVLW           0x32
          MOVEWF          REG2, 0
LOOP1: DECF               REG2, 1
          BNZ             LOOP1
```

(1) 1 (3) 50
(2) 32 (4) 82

9. In the program of Question 8, if the instruction BNZ was changed to BZ how many times is LOOP1 executed?
(1) 1 (3) 50
(2) 32 (4) 82

10. Calculate the time delay of LOOP1 in the program of Question 8 if the clock frequency is 10-MHz. Ignore the difference in execution of the last cycle of the BNZ instruction.
(1) 0.6-μS (3) 60-μS
(2) 6-μS (4) 600-μS

11. If the bytes 97_H and 68_H were ANDed what would be the result?
(1) FF_H (3) 07_H
(2) $D1_H$ (4) 00_H

12. If the bytes $F8_H$ and 31_H were ANDed what would be the result?
(1) 129_H (3) 39_H
(2) $A7_H$ (4) 30_H

13. After the ANDing has taken place in Question 12, which flag or flags are affected?
(1) None, all are reset (3) N is set
(2) N and Z are reset (4) N and Z are set

14. If the bytes $F8_H$ and 37_H are ORed what would be the result?
(1) FF_H (3) $12F_H$
(2) $C1_H$ (4) $1CD_H$

15. After the ORing has taken place in Question 14, which flag or flags are affected?
(1) None, all are set (3) N is reset
(2) N is set (4) N and C are reset

16. If the bytes 97_H and 67_H were ORed what would be the result?
(1) FF_H (3) $0F_H$
(2) $F7_H$ (4) 00_H

17. What is the instruction to reset the OV flag in the STATUS register?
    (1) BCF STATUS, 3
    (2) BCF STATUS, 2
    (3) BCF STATUS, 5
    (4) BCF STATUS, 7

18. The instruction to load a byte in WREG is?
    (1) MOVWF BYTE
    (2) MOVFW BYTE
    (3) MOVLW BYTE
    (4) MOVWL BYTE

19. If the bytes 31_H and $C8_H$ were exclusive ORed, the results would be?
    (1) Cannot exclusively OR bytes in PIC MPU.
    (2) Exclusive OR these two bytes would produce an erroneous result.
    (3) The result would be $F9_H$.
    (4) The result would be 00_H.

20. Which flag or flags would be set after 31_H and $C8_H$ were exclusive ORed?
    (1) All of the flags.
    (2) None of the flags.
    (3) N would be reset.
    (4) N would be set.

End of Examination

Lesson 8110-Stack and Subroutines
Microcontrollers and Embedded Systems

Assignment Checklist

- ✓ Read Chapter 7 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Lesson 8110 in this Study Guide
- ✓ Take the Examination for this lesson

<u>Chapter 7 Lecture/Discussion</u>

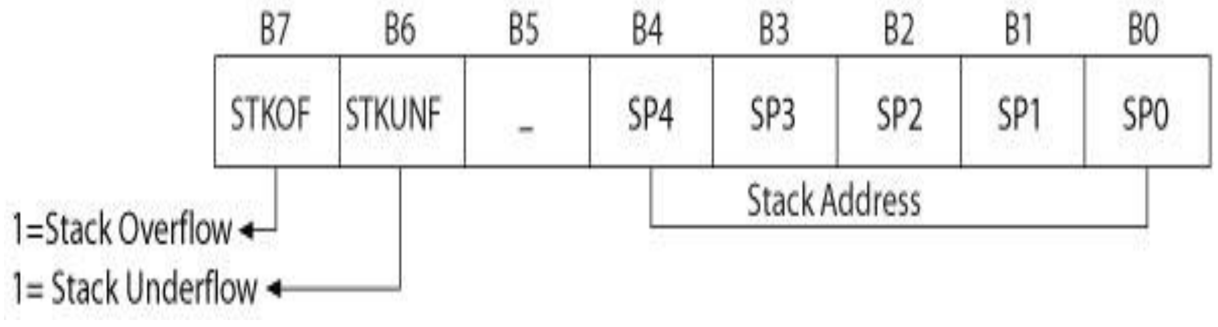
Stack

- Temporary memory storage space used during the execution of a program
- Can be part of R/W memory or specially designed group of registers
- Stack Pointer (SP)
    - The MPU uses a register called the stack pointer, similar to the program counter (PC), to keep track of available stack locations.

PIC18 Microcontroller Stack

- Consists of 31 registers-21-bit wide, called the hardware stack
    - Stack is neither a part of program memory or data registers.
    - To identify these 31 registers, 5-bit address is needed
    - PIC18 uses one of the special function registers called STKPTR (Stack Pointer) to keep track of the available stack locations (registers).

STKPTR (Stack Pointer) Register



- SP4-SP0: Stack Address
- STKOF: Stack overflow
    - When the user attempts to use more than 31 registers to store information (data bytes) on the stack, BIT7 in the STKPTR register is set to indicate an overflow.
- STKUNF: Stack underflow
    - When the user attempts to retrieve more information than what is stored previously on the stack, BIT6 in the STKPTR register is set to indicate an underflow.

Instructions to Store and Retrieve Information from the Stack

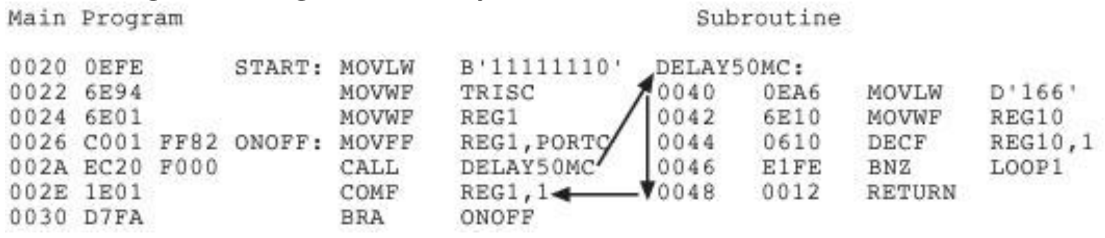
- PUSH ;Increment the memory address in the stack ;pointer (by one) and store the contents of the ;program counter (PC+2) on the top of the ;stack
- POP ;Discard the address of the top of the stack ;and decrement the stack pointer by one
- The contents of the stack (21-bit address), pointed by the stack pointer, are copied into three special function registers
  - TOSU (Top-of-Stack Upper), TOSH (High), and TOSL (Low)

Subroutine

- A group of instructions that performs a specified task
- Written independent of a main program
- Can be called multiple times to perform task by main program or by another subroutine
- Call and Return instructions used to call a subroutine and return from the subroutine

Example 7.2

- Program Listing with Memory Addresses



```
Main Program                                         Subroutine

0020 0EFE      START: MOVLW   B'11111110'   DELAY50MC:
0022 6E94             MOVWF   TRISC         0040   0EA6   MOVLW    D'166'
0024 6E01             MOVWF   REG1          0042   6E10   MOVWF    REG10
0026 C001 FF82 ONOFF: MOVFF   REG1,PORTC    0044   0610   DECF     REG10,1
002A EC20 F000        CALL    DELAY50MC     0046   E1FE   BNZ      LOOP1
002E 1E01             COMF    REG1,1        0048   0012   RETURN
0030 D7FA             BRA     ONOFF
```

Subroutine Documentation and Parameter Passing

- Parameter passing
  - Information exchanged between a calling program and a subroutine
- Subroutine documentation should include:
  - Function of a subroutine: Brief description of what it does
  - Input parameters: Information that should be provided by calling program to subroutine
  - Output parameters: Information or results provided by subroutine to calling program
  - Registers modified: List of registers changed by the subroutine
  - List of subroutines called: List of other subroutines called by the called subroutine

Macros and Software Stack

- Macro
  - Group of assembly language instructions that can be labeled with name
  - Short cut provided by assembler
  - Format includes three parts
    - Header: includes a name (label) of the macro in the label field, MACRO as pseudo code in the opcode field, and a list of arguments in the operand field

    Example: DELAY MACRO VAR1, VAR2, VAR3

- Body of instructions: a set of assembler instructions and pseudo-codes necessary to accomplish the task of a macro
- Termination: Endm (end of macro); pseudo code used to indicate the end of a macro

Subroutine versus Macro

- Subroutine
  - Requires instructions such as Call and Return, and the stack
  - Memory space required by a subroutine does not depend on how many times it is called
  - Is less efficient in execution than that of a macro because it includes overhead instructions such as Call and Return
- Macro
  - Based on assembler
  - Shortcut in writing assembly code
  - Memory space required is dependent on how many times it is called
  - In execution, more efficient because it does not have overhead instructions

****On Page 225 the last box of Program 7.5.3 contains an error. The box begins with:

```
        MOVWF       LO_SUM
        MOVLW       0x03                ;
Shifting right three times – set up counter for 3
```

What it should be is:

```
        MOVWF       LO_SUM
        MOVLW       0x03                ;Shifting right three times – set up counter for 3
```

Lesson 8110 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. A _________ is a temporary storage space used to store MPU register information.
   (1) Program counter (3) Stack
   (2) Stack pointer (4) Subroutine

2. The __________ is a register in a MPU that is used to hold an address of the memory location that is available to store the information.
   (1) Program counter (3) Stack
   (2) Stack pointer (4) Subroutine

3. The ___________ holds the address of the memory location of the next code to be executed.
   (1) Program counter (3) Stack
   (2) Stack pointer (4) Subroutine

4. A _____________ is a group of instructions that performs a specific task.
   (1) Program counter (3) Stack
   (2) Stack pointer (4) Subroutine

5. A ___________ is a group of assembly language instructions that can be labeled with a name.
   (1) Subroutine (3) Micro
   (2) Shell (4) Macro

6. The stack pointer register is the same size as the program counter.
   (1) True (2) False

7. The stack in the PIC18 microcontroller consists of how many registers?
   (1) 8 (3) 21
   (2) 31 (4) 5

8. Each register in the stack is _______ bits wide.
   (1) 8 (3) 21
   (2) 31 (4) 5

9. The stack pointer is a(n) ____ bit register.
   (1) 8 (3) 21
   (2) 31 (4) 5

10. The term *output parameter* to a subroutine is explained as the information needed to the subroutine by the calling program.
    (1) True (2) False

11. The advantage of a subroutine is that it is a more efficient execution than a macro.
    (1) True (2) False

12. A disadvantage with a macro is that its code is included in the program every time it is invoked.
    (1) True (2) False

13. The stack pointer that holds the address of the TOS register is ______ bits wide.
    (1) 8 (3) 21
    (2) 31 (4) 5

14. When the user tries to use more registers than the PIC18 microcontroller has for the stack, then this is called _____________?
    (1) overflow (3) push
    (2) underflow (4) pull

15. The bit number in the STKPTR Register that is associated with the answer to Question 14 is _____________?
    (1) B4 (3) B6
    (2) B5 (4) B7

End of Examination

Lesson 8112-Application Programs and Software Design
Microcontrollers and Embedded Systems

Assignment Checklist

- ✓ Read Chapter 8 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Lesson 8112 in this Study Guide
- ✓ Take the Examination for this lesson

Chapter 8 Lecture/Discussion

Application Programs

- BCD to Binary Conversion
- Binary to BCD Conversion
- ASCII Code to Binary Conversion
- Binary to ASCII Code Conversion
- Multiplication of 16-bit Numbers
- Division of Two Unsigned Numbers

BCD to Binary Conversion

- In many microcontroller-based products, data are entered in decimal format.
- The numbers A through F are invalid in the BCD number system.
- It is inefficient to process the data in BCD numbers.
  - Therefore, it is necessary to convert the BCD data into binary numbers.
- Problem statement
  - Given a packed BCD number in WREG:
    - Write a subroutine to unpack the number.
    - Convert it into its equivalent binary number.
    - Return the number in WREG.
- Problem analysis
  - The BCD numbers include only ten digits from 0 to 9.
  - The value of the digit is based on its position in a given number, called positional weighting.
  - Example, in decimal number 97, the value of 9 is 90.
  - To find the binary value of 97BCD, the steps are:
    - Packed 97BCD = 1001 0111
    - Step 1: Unpack the number
    - 0 9 (00001001) and 07 (00000111)
    - Step 2: Multiply high-order digit by 10 and add low-order digit = (00001001) x 10 + (0000111)

Binary to BCD Conversion

- Data processing inside microcontrollers done in binary
- Displays generally in BCD numbers
- Examples of BCD displays
  - Readouts on the dashboard of a car, microwave ovens, measuring instruments, and electronic clocks

- Necessary that binary output of MCU be converted into BCD digits
  - Accomplished by dividing binary number by powers of ten
- Problem statement
  - Given an 8-bit temperature reading in WREG
    - Write a subroutine to convert the reading into equivalent BCD numbers.
    - Store the numbers as unpacked digits in buffer registers BUFF0, BUFF1, and BUFF2.
- Problem analysis
  - A binary reading is converted into BCD numbers by dividing the reading by powers of ten until a remainder is found.
  - The largest 8-bit number is FFH, equivalent to 255BCD.
  - Three registers (BCD2, BCD1, and BCD0) are needed to save BCD digits for an 8-bit number larger than 100 and two divisors of powers of ten (100 and 10).
  - Subtract 100 from the number and continue to subtract until the number becomes less than divisor (100).
- Problem analysis (cont'd)
  - Once the number becomes negative, the last subtraction must be cancelled by adding the divisor (100) to the result.
  - The number of subtractions represents the most significant digit (BCD2 among three digits) in the quotient.
  - Continue the same process with the divisor 10 and find BCD1.
  - The remainder after dividing by ten represents the least significant BCD0.

ASCII Code to Binary Conversion

- The microcontrollers operate in binary language, but human beings use alphanumeric symbols (alphabets and numbers) to communicate.
- To facilitate communication between human beings and a microcontroller, we need to translate between alphanumeric symbols and binary language.
  - The commonly used code for such translation is ASCII (American Standard Code for Information Exchange).
    - 7-bit code that represents 128 characters, and its range is from 00 to 7FH; eighth bit always zero
- Problem Statement
  - Write a subroutine to convert an ASCII code for a Hex digit from 0 to FH, provided in WREG, into its binary equivalent.
  - The subroutine should also check for any invalid codes below 30H.
- Problem Analysis
  - The ASCII code for digits 0 to 9 is from 30H to 39H, in sequential order.
  - Converting these codes into their binary equivalent can be accomplished by subtracting 30H from the code.
  - The code for digits A though F ranges from 41H to 46H; there is a gap of seven digits (3AH to 40H) from the code of the last digit 9 (39H).
    - Therefore, if we subtract 7 again (after subtracting 30H) from the code, we can find the binary equivalent digits A through F.

Binary to ASCII Code Conversion

- The MPU performs data processing in binary language.
- To display results at output devices such as printers or video screen, binary readings must be converted into ASCII characters.

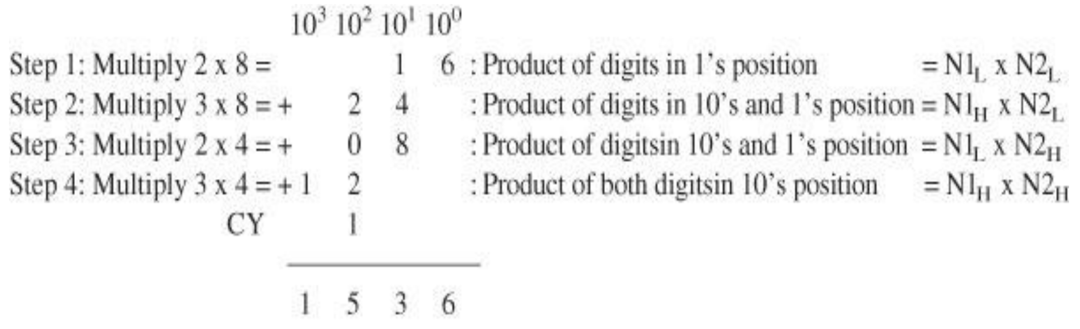
****On Page 238 the textbook has an error in Section 8.4.1 as the Problem Statement should read as stated below:

- Problem statement
  - **Write a subroutine** to convert an 8-bit binary number given in WREG into its ASCII characters.

- Problem analysis
  - An 8-bit number must be first unpacked to get two digits.
  - The process of finding ASCII code for the unpacked digits is just the opposite to that of ASCII to binary conversion.
  - If the digit is between 0 and 9, add 30H.
  - If the digit is larger than 9 (from A to F), add additional 07H.

Multiplication of 16-bit Numbers

- The PIC18 microcontroller includes multiply instructions that multiply 8-bit numbers and store the result in 16-bit register PRODH and PRODL.
- To multiply 16-bit numbers, numbers should be split in groups of 8 bits and use the multiply instruction to multiply 8 bits at a time and add them as in the manual multiplication.
- Problem statement
  - Write subroutines to multiply two 16-bit numbers stored in registers NUM1H, NUM1L, NUM2H, and NUM2L and store the product in four data registers PROD0 (LSB) through PROD3 (MSB).
- We are dealing with number systems with positional weightings meaning the position of the digit in a given number determines its value.
- When we multiply numbers in decimal system, we multiply two digits at a time and we place the product in an appropriate position.
  - An example of multiplying two-digit decimal numbers is shown next.
- Example of multiplying two-digit decimal numbers:
  - $N1_H . N1_L$ x $N2_H . N2_L$ = 32_{10} x 48_{10} = 1536_{10}. The steps are as follows:



	10^3	10^2	10^1	10^0		
Step 1: Multiply 2 x 8 =			1	6	: Product of digits in 1's position	$= N1_L \times N2_L$
Step 2: Multiply 3 x 8 = +		2	4		: Product of digits in 10's and 1's position	$= N1_H \times N2_L$
Step 3: Multiply 2 x 4 = +		0	8		: Product of digitsin 10's and 1's position	$= N1_L \times N2_H$
Step 4: Multiply 3 x 4 = +	1	2			: Product of both digitsin 10's position	$= N1_H \times N2_H$
CY		1				
	1	5	3	6		

- In the above multiplication, two digits are multiplied at a time with all four combinations.

- The addition is done by shifting the multiplication results to the left position based on the positions of the digits.

****The 8.5.3 Program has a few errors in the code. The first four lines should be stated as:

```
Line 1      NUM1LO    EQU   0x02   ;16-bit number1-0102H
Line 2      NUM1HI    EQU   0x01
Line 3      NUM2LO    EQU   0x05   ;16-bit number1-0205H
Line 4      NUM1HI    EQU   0x02
```

Software Design

- Project statement
  - Design a software program to meet the following specifications:
    - Given a string of temperature readings that terminates in a null character 00, find the average temperature.
    - Convert the average temperature reading in BCD digits into ASCII characters to be displayed by some other subroutine.
    - A new data set of temperature readings is recorded every 100 ms and it is indicated by setting Bit0 (carry flag) in STATUS register.
- Project analysis
  - By examining the project statements carefully, many clues can be found to divide the project in small segments.
  - These small segments can be written as independent modules or subroutines.
  - The project is divided as follows:
    - In this project, the number of bytes is variable and determined by the last null character.
    - A separate subroutine can be written to check for the null character and count the number of bytes in the string.
    - To find the average reading, these readings must be added first that may generate carries, and the sum will require more than one 8-bit register.
  - To find the average temperature reading, the sum must be divided by the number of readings.
  - The number of bytes in the string counted in the previous subroutine must be passed on to this division process.
  - The repeated subtraction method to divide a 16-bit sum by an 8-bit divisor can be used.
  - The average temperature is in binary (Hex).
    - The reading must be converted in BCD numbers.
    - The subroutine in Section 8.2 converts an 8-bit Hex number in three BCD numbers.
  - To display these digits at ASCII peripherals, these digits must be converted into ASCII codes by using the subroutine BINASC discussed in Section 8.4.
  - New data string is available every 100 ms, and the availability of new data set is indicated by setting the carry flag in the STATUS register.
  - This process must be repeated every 100 ms.
  - This program should check for the carry flag at the beginning before it begins the execution and stays in a continuous loop until the carry flag is set.

Lesson 8112 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. Using subroutines to convert the numbers generated by the microprocessor program to BCD for a display means that we divide the number by multiples of ______.
   (1) 10 in hexadecimal
   (2) 10 in binary
   (3) 10 in decimal
   (4) 10 in ASCII

2. The label to denote where the subroutine program resides is important to the operation of the program.
   (1) True
   (2) False

3. In the following subroutine program called UNPACK, which line is responsible for masking the high-order nibble?

```
UNPACK:     ANDLW 0x0F
            MOVWF BUFFER1
            MOVF REG1, W
            SWAPF WREG, 0
            ANDLW 0xOF
            MOVWF BUFFER2
            RETURN
            END
```

   (1) ANDLW 0x0F (the first usage)
   (2) MOVWF BUFFER1
   (3) SWAPF WREG, 0
   (4) MOVWF BUFFER2

4. Which line is responsible for saving the high-order nibble?
   (1) ANDLW 0x0F
   (2) MOVWF BUFFER1
   (3) SWAPF WREG, 0
   (4) MOVWF BUFFER2

5. In the following subroutine program called UNPACK, which line is responsible for masking the low-order nibble?

```
UNPACK:     ANDLW 0x0F
            MOVWF BUFFER1
            RRNCF REG1, 1
            RRNCF REG1, 1
            RRNCF REG1, 1
            RRNCF REG1, 1
            ANDLW 0x0F
            MOVWF BUFFER2
            RETURN
            END
```

(1) ANDLW 0x0F (the first usage
(2) MOVWF BUFFER1
(3) ANDLW 0x0F (the second usage
(4) RRNCF REG1, 1

6. The difference between the subroutine in problem 3 and the one in problem 5 is that we used three RRNCF instructions to replace the SWAPF instruction
(1) True
(2) False

7. If we analyze the program fragment from Section 8.4 in the text, how many subroutines will the microprocessor encounter?
(1) One
(2) Two
(3) Three
(4) There are no subroutines in the referenced program fragment

8. In Section 8.5 of the text, the program used multiplies two 16-bit numbers together. Which section of the program is responsible for multiplying NUM1LO and NUM2HI?
(1) SECT1
(2) SECT3
(3) SECT2
(4) SECT4

9. If we analyze the program fragment from Section 8.7 in the text, how many subroutines will the microprocessor encounter?
(1) Two
(2) Three
(3) Four
(4) Five

10. The program shown in Section 8.6 of the text has to clear the carry flag and the register REMAINDR before executing any of the division process.
(1) True
(2) False

11. Ensuring that the labels used for subroutines are not duplicated or undefined is one thing we as programmers really do not have to worry about happening.
(1) True
(2) False

12. In Section 8.5 of the text, the program used multiplies two 16-bit numbers together. Which section of the program is responsible for multiplying NUM1LO and NUM2LO?

(5) SECT1	(7) SECT2
(6) SECT3	(8) SECT4

13. What is the code used to communicate between video screens or printers and the microcontroller?

(1) Binary	(3) ASCII
(2) Hexadecimal	(4) BCD

14. If we were to modify the addition subroutine from Section 8.7 like this, what would the result be?

```
ADDITION:   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;Function:      Adds the data bytes in the data string                            ;
            ;Input:         Address in FSR0 to point to data string and                       ;
            ;               Count of data bytes in COUNTER register                           ;
            ;Output:        16-bit sum-High byte in CYREG and low byte in W                   ;
            ;Registers Changed: COUNTER and FSR0                                              ;
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CYREG       EQU         0x03
            CLRRF       CYREG               ;Clear carry register
            LFSR        FSR0,BUFFER         ;Set up FSR0 as pointer for data registers
            MOVLW       0x00                ;Clear W register to save sum
NEXT:       BTFSC       INDF0,7
            BRA         SKIP
            ADDWF       INDF0,W             ;Add byte and save in W
            BNC         SKIP                ;Check for carry-if no carry jump to SKIP
            INCF        CYREG               ;If there is carry, increment CYREG
SKIP:       MOVF        POSTINC0,1          ;Increment the pointer
            DECF        COUNTER,1,0         ; Next count and save count in register
            BNZ         NEXT                ;If COUNT ≠ 0, go back to add next byte
            RETURN
```

(1) There would be no change in the operation
(2) It would add only positive readings
(3) It would add only negative readings
(4) The program would not function at all

15. In the breakout sections of each major section is an area entitled Program Execution and Troubleshooting. How does the author recommend you check each program?
(1) By setting breakpoints and then running the program
(2) By single stepping through the program
(3) By just running the entire program
(4) The author uses each one of these but applies them strategically

End of Examination

Lesson 8114-Interrupts and Timers
Microcontrollers and Embedded Systems

Assignment Checklist

- ✓ Read Chapter 10 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Chapter 11 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Lesson 8114 in this Study Guide
- ✓ Take the Examination for this lesson

<u>Chapter 10 Lecture/Discussion</u>

Basic Concepts in Interrupts

- An interrupt is a communication process set up in a microprocessor or microcontroller in which:
  - An internal or external device requests the MPU to stop the processing
  - The MPU acknowledges the request
  - Attends to the request
  - Goes back to processing where it was interrupted

Types of Interrupts

- Hardware interrupts (Figure 10-1)
  - Maskable: can be masked or disabled
  - Two groups: external and internal interrupts
  - Non-maskable: cannot be disabled
  - Software interrupts: generally used when the situation requires stop processing and start all over
  - Examples: divide by zero or stack overflow
  - Generally, microcontrollers do not include software interrupts

MPU Response to Interrupts

- When the interrupt process is enabled, the MPU, during execution, checks the interrupt request flag just before the end of each instruction.
- If the interrupt request is present, the MPU:
  - Completes the execution of the instruction
  - Resets the interrupt flag
  - Saves the address of the program counter on the stack
- Some interrupt processes also save contents of MPU registers on the stack.
  - Stops the execution
- To restart the execution, the MPU needs to be redirected to the memory location where the interrupt request can be met.
  - Accomplished by interrupt vectors
- The set of instructions written to meet the request (or to accomplish the task) is called an interrupt service routine (ISR).
- Once the request is accomplished, the MPU should find its way back to the instruction, next memory location where it was interrupted.
  - Accomplished by a specific return instruction

Interrupt Vectors

- Direct the MPU to the location where the interrupt request is accomplished.
- They are:
  - Defined memory location where a specific memory location/s is assigned to the interrupt request
  - Defined vector location where specific memory locations assigned to store the vector addresses of the ISRs
  - Specified by external hardware: The interrupt vector address (or a part of it) is provided through external hardware using an interrupt acknowledge signal.

Interrupt Service Routine (ISR)

- A group of instructions that accomplishes the task requested by the interrupting source
- Similar to a subroutine except that the ISR must be terminated in a Return instruction specially designed for interrupts
  - The Return instruction, when executed, finds the return address on the stack and redirects the program execution where the program was interrupted.
  - Some Return instructions are designed to retrieve the contents of MPU registers if saved as a part of the interrupts.

Interrupt Priorities

- Rationale for priorities
  - Multiple interrupt sources exist in a system, and more than one interrupt requests can arrive simultaneously.
  - When one request is being served (meaning when the MPU is executing an ISR), another request can arrive.
  - Therefore, the interrupt requests must be prioritized. Most MCUs (and MPUs) include an interrupt priority scheme. Some are based on hardware and some use software.

Reset as a Special Purpose Interrupt

- Reset is an external signal that enables the processor to begin execution or interrupts the processor if the processor is executing instructions.
- There are at least two types of resets in microcontroller-based systems.
  - Power-on reset and manual reset
- When the reset signal is activated, it establishes or reestablishes the initial conditions of the processor and directs the processor to a specific starting memory location.

PIC18 Interrupts

- PIC18 Microcontroller family
  - Has multiple sources that can send interrupt requests
  - Does not have any non-maskable or software interrupts; all interrupts are maskable
  - Has a priority scheme divided into two groups
    - High priority and low priority
  - Uses many Special Function Registers (SFRs) to implement the interrupt process

PIC18 Interrupt Sources

- Divided into two groups
  - External sources and internal peripheral sources on the MCU chip
    - External sources
      - Three pins of PORTB -RB0/INTO, RB1/INT1,and RB2/INT2
      - These pins can be used to connect external interrupting sources such as keyboards or switches
  - Change in logic levels of pins RB4-RB7 of PORTB can be recognized as interrupts
- Internal peripheral sources
  - Examples: Timers, A/D Converter, Serial I/O, and Low-Voltage Detection Module
- SFRs
  - Used to setup the interrupt process:

• RCON	Register Control
• INTCON	Interrupt Control
• IINTCON2	Interrupt Control2
• INTCON3	Interrupt Control3
• PIR1 and PIR2	Peripheral Interrupt Register1 & 2
• PIE1 and PIE2	Peripheral Interrupt Enable 1 & 2
• IPR1 and IPR2	Interrupt Priority Register 1 & 2

- RCON register sets up global priority.
- INTERCON registers deal primarily with external interrupt sources.
- PIR, PIE, and IPR handle internal peripheral interrupts.
- To recognize the occurrence of an interrupt request, the MPU needs to check the following three bits:
  - The flag bit to indicate that an interrupt request is present
  - The enable bit to redirect the program execution to the interrupt vector address
  - The priority bit (if set) to select priority

Interrupt Priorities and RCON Register

- Any interrupt can be set up as high-priority or low-priority.
  - All high-priority interrupts are directed to the interrupt vector location 000008H.
  - All low-priority interrupts are directed to the interrupt vector location 000018H.
  - A high-priority interrupt can interrupt a low-priority interrupt in progress.
- The interrupt priority feature is enabled by Bit7 (IPEN) in RCON register.

External Interrupts and INTCON Registers

- Figure 10-4 (a), (b), and (c) shows three registers with interrupt bit specifications primarily for external interrupt sources.
- Some bits are used for internal peripherals such as Timer0.

****Note in Figure 10-4 (a) on Page 319 at the top of the page the information in B7 and the first line of the register description is incorrect. It should be GIE/GIEH and not GIE/GEIH as shown.

Interrupt Service Routine (ISR)

- Similar to a subroutine
- Attends to the request of an interrupting source
- Must be terminated with the instruction RETFIE (Return from Interrupt)
- Should save register contents that may be affected by the code in the ISR
- When an interrupt occurs, the MPU:
    - Completes the instruction being executed
    - Disables global interrupt enable
    - Places the address from the program counter on the stack (the address where the MPU should return after the completion of the ISR)
- The RETFIE instruction
    - Gets the address from the top of the stack
    - Places it in the program counter
    - Enables the global interrupt enable bit (INTCON <7>)
    - Retrieves the contents of registers W, BSR, and STATUS in high-priority interrupts
- RETFIE instruction format
    - RETFIE [s] ;Return from interrupt: s = 0 or 1
        - If s =1, the MPU also retrieves the contents of W, BSR, and STATUS register (previously saved) before enabling the global interrupt bit.
    - Format: RETFIE FAST
        - The same as RETFIE 1 except the formats are different
- In high-priority interrupts:
    - The contents of W, STATUS, and BSR registers are automatically saved into respective registers, called shadow registers.
    - RETFIE 1 (or RETFIE FAST) retrieves the contents of these registers.
- In low-priority interrupts
    - These registers must be saved as a part of an ISR if these registers are affected by the code in the ISR.

****Note on Page 321 in Example 10.1 the section INT1_ISR: has an error in the first line. It should be BCF INTCON3, INT1IF

Internal Interrupts and Related Registers

- The PIC18 MCU includes many internal devices such as the timers and the A/D converter that can interrupt the MPU.
- Each interrupt is associated with three bits
    - Priority, interrupt request flag, and enable (similar to the that of external interrupts)
- Registers associated with the internal interrupts
    - IPR: Interrupt Priority Register
    - PIR: Peripheral Interrupt Request (Flag)
    - PIE: Peripheral Interrupt Enable

Handling Multiple Interrupt Sources

- In PIC18 MCU, all interrupt requests are directed to one of two memory locations:
  - 000008H (high-priority) or
    000018 (low-priority)
- When multiple requests are directed to these locations, the interrupt source must be identified by checking the interrupt flag through software instructions.

PIC18 Resets

- When the reset signal is activated:
  - The MPU goes into a reset state during which the initial conditions are established.
  - The program counter is cleared to 000000 which is called the reset vector.
  - The MPU begins the execution of instructions from location 000000.
- PIC18 MCU can be reset by external source such as the push-button key, or when power is turned-on, or by various internal sources.
  - Resets categorized as follows:
    - External Manual Reset Key
    - Power-on Reset (POR)
    - Watchdog Timer Reset (WDT)
    - Programmable Brown-Out Reset (BOR)
    - RESET and SLEEP Instructions
    - Stack Full and Underflow Reset

****Note there are two errors to mention on Page 330. The 10.3.3 Program contains an error in section MAIN: on line 9 and it should be

```
BSF    INTCON3, INT1IE    ;Enable Interrupt 1-
                          ;INTCON3 <3>
```

In line 14 it should be BSF PIE1, TMR2IE

<u>**Chapter 11 Lecture/Discussion**</u>

Basic Concepts in Counters and Timers

- In digital systems
    - Counting is a fundamental concept.
    - Clock is an essential element.
    - Count is in synchronization with the clock.
    - Count is converted in time by multiplying the count and the clock period.

Hardware Counters and Timers

- Counter is a register that can be loaded with a binary number (count) which can be decremented or incremented per clock cycle.
- Time is calculated as follows:
    - Find the difference between the beginning count and the last count
    - Multiply the count difference by the clock period

- The register can also be used as a counter by replacing the clock with a signal from an event.
- When a signal from an event arrives, the count in the register is incremented (or decremented); thus, the total number of events can be counted.

Types of Counters

- Up-counter
    - Counter is incremented at every clock cycle
    - When count reaches the maximum count, a flag is set
    - Counter can be reset to zero or to the initial value
- Down-counter
    - Counter is decremented at every clock cycle
    - When count reaches zero, a flag is set
    - Counter can be reset to the maximum or the initial value
- Free-running counter
    - Counter runs continuously and only readable
    - When it reaches the maximum count, a flag is set

Timer Applications

- Time delay
- Pulse wave generation
- Pulse width or frequency measurement
- Timer as an event counter

Capture, Compare, and PWM (CCP) Modules

- CCP modules are commonly found in recent microcontrollers
    - 16-bit (or two 8-bit) registers specially designed to perform the following functions in conjunction with timers
        - Capture: The CCP pin can be set as an input to record the arrival time of a pulse.

- Compare: The CCP pin is set as an output, and at a given count, it can be driven low, high, or toggled.
- Pulse width modulation (PWM): The CCP pin is set as an output and the duty cycle of a pulse can be varied.

Pulse Width Modulation (PWM)

- Duty cycle is defined as the percentage ratio of on time of a pulse to its period, and the changing of the duty cycle is defined as PWM
  - In the PWM mode, CCP pin is set as an output
  - The count for the period and the duty cycle are loaded into CCP registers.
  - In this mode, the duty cycle of the output pulse can be varied.

PIC18 Timers

- The PIC18 microcontroller has multiple timers, and all of them are up-counters.
- Timers are divided into two groups: 8-bit and 16-bit
- Labeled as Timer0 to Timer3 or Timer4 (if available)
  - Timer0 can be set up as an 8-bit or 16-bit timer.
  - Timer1 and Timer3 are 16-bit timers.
  - Timer2 and Timer4 (if available) are 8-bit timers.
- Each timer associated with its Special Function Register (SFR): T0CON-T3CON or T4CON

Timer0

- **Can be set up as an 8-bit or 16-bit timer (Figures 11-2 (a) and (b)**
- Is readable and writable
- Parameters are set up by bits in T0CON register
- **Has eight options of pre-scale values (Bit2-Bit0-Figure 11-3)**
- Can run on internal clock source (instruction cycle) or external clock connected to pin RA4/T0CK1
- Generates an interrupt or sets a flag when it overflows from FFH to 00 in the 8-bit mode and from FFFFH to 0000 in the 16-bit mode
- Can be set up on either rising edge or falling edge (Bit4 –Figure 11-3) when an external clock is used

Timer0 Control Register (Figure 11-3)

- Timer0 as timer
  - Bit5 must be cleared to use the internal clock.
  - At each instruction cycle (four clock cycles), the timer register is incremented.
- Timer0 as a counter
  - Bit5 must be set 1 to use an external clock.
  - In this mode, input signal at PORTA-pin RA4/T0CK used as a clock.
  - When Bit4 = 1, register is incremented on the falling edge, and when Bit4 = 0, the register is incremented on the rising edge.
- Prescaler
  - Divides clock frequency by a specified ratio.

  - To use prescaler, Bit3 = 0, and three bits Bit2-Bit0 specify scaler ratio from 1:2 to 1:256
- Interrupt
  - When Timer0 overflows from FFH to 00 in the 8-bit mode and from FFFFH to 0000 in the16-bit mode, it sets TMR0IF (Timer0 Interrupt Flag) –Bit2 in the INTCON register.
    - Flag can be used two ways: 1) a software loop can be set up to monitor the flag, or 2) an interrupt can be generated.
    - Flag must be cleared to start the timer again.
- 16-bit mode
  - When Timer0 is set in the 16-bit mode, it uses two 8-bit registers TMR0L and TMR0H.

****Note on Page 345 in Example 11.2 Line 3 from the end "Therefore, the count should be 65, 536 – 19531 = 46,005 = $B3B5_H$"

In 11.2.2 Program under section DELAY_1s: Line 1 should be
MOVLW 0xB3 ;High count of $B3B5_H$

Also in 11.2.2 Program, last line MOVLW 0xB5 ;Low count of $B3B5_H$

On Page 346 11.2.2 Program continues and in the section TMR0_ISR in line 1 the comment should be ;High count of $B3B5_H$

And in line 3 it should be MOVLW 0xB5 ;Low count of $B3B5_H$

Timer1
- Figure 11-5
  - A 16-bit counter/timer with two 8-bit registers (TMR1H and TMR1L); both registers are readable and writable
  - Four options of prescale value (Bit5-Bit4)
  - Clock source (Bit1) can be internal (instruction cycle) or external (pin RC0/T13CK1) on rising edge
  - Sets flag or generates an interrupt when it overflows from FFFFH to 0000
- Timer1 Operation
  - Can operate in three modes: timer, synchronous counter, and asynchronous counter
  - Bit0 enables or disables the timer
  - When Bit1 = 0, it operates as a timer and increments count at every instruction cycle.
  - When Bit1 = 1, it operates as a counter and increments count at every rising edge of the external clock.
  - When Bit3 = 1, Timer1 oscillator is enabled which is used for low frequency operations.

- Interrupt
  - When the Timer1 overflows from FFFFH to 0000, it sets the TMR1IF (Timer1 Interrupt Flag) –Bit0 in the Peripheral Interrupt Register1 (PIR1).
  - This flag can be used two ways
    - A software loop can be set up to monitor the flag.
    - An interrupt can be generated.
  - Flag must be cleared to start the timer again.
- Resetting Timer1 using CCP module
  - When a CCP module associated with Timer1 is loaded with a 16-bit number and setup in the Compare mode, the count in Timer1 and the number in the CCP module are compared at every cycle.
  - When a match is found, Timer1 is reset.

Timer2
- Figure 11-6
  - An 8-bit timer (TMR2) with an 8-bit period register (PR2)
  - Registers (TMR2 and PR2) are readable and writable
  - Three options of prescale values (Bit1-Bit0)
  - 16 options of postscale values (Bit6-Bit3)
  - A flag is set when TMR2 value matches that of PR2, which can generate an interrupt
- Timer2 operation
  - Figure 11-7 shows two 8-bit registers (TMR2 and PR2)
  - An 8-bit number is loaded in PR2 and the timer is turned on, which is incremented every instruction cycle.
  - When the count in the timer register and the PR register match, an output pulse is generated and the timer register is set to zero.
  - The output pulse goes through a post scaler that divides the frequency by the scale factor and sets the flag TMR2IF- Bit1 in the Peripheral Interrupt Register1 (PIR1) that can be used to generate an interrupt.

Timer3
- Figure 11-8
  - 16-bit counter/timer with two 8-bit registers (TMR3H and TMR3L); both registers are readable and writable
  - Can operate as a timer, a synchronous or an asynchronous counter
  - Four options of prescale value (Bit5-Bit4)
  - Clock source (Bit1) can be internal (instruction cycle) or external Timer1 oscillator or T1CK1 (on the rising edge after the first falling edge)
  - Sets a flag or generates an interrupt when it overflows from $FFFF_H$ to 0000

CCP (Capture, Compare, and PWM) Modules
- Each CCP module is comprised of two 8-bit registers: CCPR1H (high) and CCPR1L (low)
- Can operate as 16-bit Capture register, 16-bit Compare register, or duty-cycle PWM register

- Timer1 and Timer3 are used as clock resources for Capture and Compare registers
- Timer2 and Timer4 (if available) are used as clock sources as PWM modules

CCP in the Capture Mode

- CCPR1 register captures the 16-bit value of Timer1 (or Timer3) when an event occurs on pin RC2/CCP1.
- When a capture occurs, the interrupt request flag bit CCP1IF (Bit2 in PIR1) is set and must be cleared for the next operation.
- To capture an event:
  - Set up pin RC2/CCP1 of PORTC as the input.
  - Initialize Timer1 in the timer mode or synchronized counter mode by writing to T1CON register.
  - Initialize CCP1 by writing to the CCP1CON register.
  - Clear the CCP1IF flag to continue the next operation when a capture occurs.
  - Clear CCP1IE and CCP1IF to avoid a false interrupt when capture mode is changed.

CCP in the Compare Mode

- 16-bit value loaded by the user in CCPR1 (or CCPRx) is constantly compared with the TMR1 (or TMR3) register when the timers are running in either timer mode or synchronized counter mode.
- When a match occurs, the pin RC2/CCP2 on PORTC is driven high, low, or toggled based on mode select bits in the CCP1CON (Bit3-Bit0 in CCP1 control register), and the interrupt flag bit CCP1IF is set.
- To set up CCP1 in the Compare mode:
  - Set up pin RC2/CCP1 of PORTC as output.
  - Initialize Timer1 in the timer mode or the synchronized counter mode by writing to the T1CON register.
  - Initialize CCP1 by writing to the CCP1CON register.
  - Clear the flag CCP1IF, which is set when a compare occurs, and must be cleared to continue to the next operation.
  - For a special event trigger, an internal hardware trigger is generated that can be used to initiate an action.
  - The special event trigger output resets Timer1.

CCP in the Pulse Width Modulation (PWM) Mode

- A CCP module in conjunction with Timer2 can be set up to output a pulse wave form for a given frequency and a duty cycle.
- The CCP module uses a 10-bit number to specify the duty cycle.
- The 8-bit number loaded into the PR2 register specify the PWM period.
- When TMR2 is equal to PR2, the following three events occur in the next increment cycle:
  - TMR2 is cleared.
  - Pin RC2/CCP1 of PORTC is set high.
  - The PWM duty-cycle byte is latched from CCPR1L into CCPR1H.

        - When CCPR1H and TMR2 match again for the specified duty cycle, the CCP1 pin is cleared.
- To Initialize CCP1 in the PWM mode:
    - Set up pin RC2/CCP1 of PORTC as output.
    - Set up PWM period by writing to the PR2 register.
    - Set up PWM duty cycle by writing to CCPR1L register and Bit5-Bit4 of CCP1CON register.
    - Set up TMR2 prescale value and Timer2 in timer mode by writing to T2CON register.
    - Enable CCP1 module in the PWM mode.
    - Set up CCP1 by writing to the CCP1CON register.

Timer Calculations
- Example 11.8
-

$$\text{Instruction Cycles} = \frac{1.0\ \text{s}}{0.4\ \mu\text{s}} = 2{,}500{,}000$$

$$\frac{1}{\text{Fw}} = \frac{4 \times \text{Prescale} \times \text{Count} + 1}{\text{Foc}} \text{ therefore, Count} + 1 = \frac{\text{Foc}}{\text{Fw} \times 4 \times \text{Prescale}} = \frac{10 \times 10^6}{10 \times 10^3 \times 4 \times \text{Prescale}}$$

****Note on Page 363 11.5.3 Program contains an error. In the description section above the coding section, the 3rd line has ;characters in ;data registers. The second set of semicolons should be removed so it should read as ;characters in data registers.

Lesson 8114 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. A(n) ______________ is a communication process between internal or external devices and the MPU to stop processing.
   (1) interrupt (3) routine
   (2) subroutine (4) program

2. A(n) _____________ is a set of instructions that accomplishes a task but is terminated with a special return instruction.
   (1) interrupt (3) interrupt service routine
   (2) subroutine (4) program

3. A(n) ____________ is an address where the MPU is directed after the acknowledgment of an interrupt.
   (1) interrupt service routine (3) interrupt hardware
   (2) interrupt vector (4) interrupt subroutine

4. Which one of the following is not a type of interrupt?
   (1) Un-maskable (3) Software
   (2) Hardware (4) Maskable

5. The type of interrupts that can be disabled and must be enabled after a system reset is the _______________.
   (1) un-maskable (3) software
   (2) hardware (4) maskable

6. The PIC18 microcontroller has ____________ interrupts divided into two groups.
   (1) un-maskable (3) hardware
   (2) maskable (4) software

7. The two groups of interrupts the PIC18 microcontroller has are the _____________ and _______________.
   (1) high priority; medium priority (3) high priority; low priority
   (2) high priority; lesser priority (4) low priority; micro priority

8. The instructions to disable the interrupt priority scheme is BCF RCOM, IPEN
   (1) True (2) False

9. The registers that are saved on the stack when an interrupt source with high priority interrupts the MCU are the ___________________________?
   (1) WREG, STATUS, and BSR (3) STATUS and BSR
   (2) WREG and STATUS (4) WREG and BSR

Questions 10, 11 and 12 are based on the following information:

A free-running 16-bit timer has a clock frequency of 5-MHz. The counter register in the timer is incremented every clock cycle. When the counter reaches $FFFF_H$ it rolls over to 0000_H and continues to count. The timer reading at the beginning of the event is $1FF8_H$ and at the end of the event is 3380_H.

10. Based on the information above, the clock period would be calculated as _________?
    (1) 0.5-µs (3) 1.0-µs
    (2) 0.2-µs (4) 1.5-µs

11. What would be the count between the two events expressed as a decimal value?
    (1) 13192_{10} (3) 5000_{10}
    (2) 8184_{10} (4) 500_{10}

12. The time delay between the two events would be ________?
    (1) 2.6 ms (3) 1.4 ms
    (2) 1.6 ms (4) 1.0 ms

13. In a pulse waveform, the on-time is 150-µs and the off-time is 300-µs. Calculate the duty cycle of the waveform.
    (1) 33% (3) 55%
    (2) 41% (4) 60%

14. Based on the information given in Question 13, calculate the frequency of the waveform.
    (1) 2.22-mHz (3) 2.22-kHz
    (2) 222.2-Hz (4) 22.2-kHz

15. The duty cycle of a square wave is 60%.
    (1) True (2) False

End of Examination

Lesson 8116-Data Converters: A/D and DAC Microcontrollers and Embedded Systems

Assignment Checklist

- ✓ Read Chapter 12 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Lesson 8116 in this Study Guide
- ✓ Take the Examination for this lesson

Chapter 12 Lecture/Discussion

Data Converters-Basic Concepts

- Analog signals are continuous, with infinite values in a given range.
  - Examples: A clock face with hands, a voltmeter with a needle, and audio signals.
- Digital signals have discrete values such as on/off or 0/1.
  - Examples: A digital clock or a digital voltmeter.
- Limitations of analog signals
  - Analog signals pick up noise as they are being amplified.
  - Analog signals are difficult to store.
  - Analog systems are more expensive in relation to digital systems.
- Advantages of digital systems (signals)
  - Noise can be reduced by converting analog signals in 0s and 1s.
  - Binary signals of 0s/1s can be easily stored in memory.
  - Technology for fabricating digital systems has become so advanced that they can be produced at low cost.
- The major limitation of a digital system is how accurately it represents the analog signals after conversion.

Embedded System

- A typical system that converts signals from analog to digital and back to analog (Figure 12-1) includes:
  - A transducer that converts non-electrical signals into electrical signals
  - An A/D converter that converts analog signals into digital signals
  - A digital processor that processes digital data (signals)
  - A D/A converter that converts digital signals into equivalent analog signals
  - A transducer that converts electrical signals into real life non-electrical signals (sound, pressure, and video)

Analog-to-Digital (A/D, ADC, or A-to-D) Conversion

- Process of converting a continuous varying signal, such as voltage or current, into discrete digital quantities that represent the magnitude of the signal compared to standard or reference voltage
- Figure 12-2a shows a simple hypothetical A/D converter circuit with one analog input signal and three digital output lines with eight possible binary combinations: 000 to 111
- Figure 12-2b shows the graph of digital output for 1 V analog input known as the quantization process.

- From Figure 12-2b, following points can be summarized:
  - Maximum value this quantization process reaches is 7/8 V for a 1 V analog signal; includes 1/8 V an inherent error
  - 1/8 V (an inherent error) is also equal to the value of the Least Significant Bit (LSB) = 001.
  - Resolution of a converter is defined in terms of the number of discrete values it can produce
  - Resolution is also expressed in the number of bits used for conversion or as $1/2^{\mathbf{n}}$ where n = number of bits
- The value of the most significant bit (MSB) -100- is equal to ½ the voltage of the full-scale value of 1 V.
- The value of the largest digital number 111 is equal to full-scale value minus the value of the LSB.
- The quantization error can be reduced or the resolution can be improved by increasing the number of bits used for the conversion.
- Can be classified in four groups:
  - Flash: uses multiple comparators in parallel as shown in Figure 12-3. The known signal is connected to one side of the comparator and the analog signal to be converted to the other side of the comparator. The output of the comparators provides the digital value. This is a high-speed, high cost converter.
  - Integrator: charges a capacitor for a given amount of time using the analog signal. It discharges back to zero with a known voltage and the counter provides the value of the unknown signal.
  - Successive approximation: Includes a D/A (digital to analog) converter and a comparator. An internal analog signal is generated by turning on successive bits in the D/A converter. Signals are compared in the comparator until the voltages match. This converter is more frequently used in industrial applications and found in many MCUs than any other type of converter.
  - Counter: Similar to a successive approximation converter except that the internal analog signal is generated by a counter starting at zero and feeding it to the D/A converter.

Successive Approximation A/D Converter Circuit

- Figure 12-4 shows a block diagram of a successive approximation A/D converter
  - The SAR (successive approximation register) begins by turning on the MSB Bit7.
  - V_o of the D/A converter is compared with the analog input voltage V_{in} in the comparator.
  - If analog voltage is less than the digital voltage, Bit7 is turned off and Bit6 is turned on.
  - If analog voltage is greater than the digital voltage, Bit7 is kept on and Bit6 is turned on.
  - The process of turning bit on/off is continued until Bit0.
  - Now the 8-bit input to the D/A converter represents the digital equivalent of the analog signal V_{in}.
- To find the digital equivalent of an analog signal connected to the ADC that is interfaced with the MCU, the MPU should:

  - Access the ADC to start a conversion.
  - Wait in a loop until the conversion is complete by checking the flag set by the ADC at the end of the conversion or respond to the interrupt generated by the ADC.
  - Read the converted digital reading.

Sample and Hold Circuit

- If the input voltage to an A/D converter is variable, the digital output is likely to be unreliable and unstable. Therefore, the varying voltage source is connected to the ADC through a sample and hold circuit.
- When the switch is connected, it samples the input voltage.
- When the switch is open, it holds the sampled voltage by charging the capacitor.
- Acquisition time: time to charge the capacitor after the switch is open and settle the output.
- Conversion time: total time needed from the start of a conversion (turning on the MSB in the SAR) until the end of the conversion (turning on/off Bit0 in the SAR)
- TAD: conversion time per bit.

PIC18F4520 Analog-to-Digital (A/D) Converter Module

- The PIC184520 microcontroller includes:
  - 10-bit A/D converter
  - 13 channels AN0 – AN12
  - Three control registers
    - ADCON0, ADCON1, and ADCON2
- Three control registers are used to:
  - Select a channel: Figure 12–6 shows a 10-bit A/D converter with a multiplexer that can accept analog signals from 13 channels.
  - Set up the I/O pins for analog signals from ports A, B, and E that are used as inputs for A/D conversion.
  - Set up pins RA2 and RA3 to connect external V_{REF} + and V_{REF} - if specified in the control register ADCON1.
  - Select an oscillator frequency divider through the control register ADCON2.
  - Select an acquisition time through the control register ADCON2.

A/D Control Register0 (ADCON0)

- Primary function of the ADCON0 register:
  - Select a channel for input analog signal
  - Start a conversion
  - Indicate the end of the conversion
- Bit1 is set to start the conversion, and at the end of the conversion this bit is reset.

A to D Control Register1 (ADCON1)

- ADCON1 is primarily used to set up the I/O pins either for analog signal or for digital signals (see Table 12.2) and select V_{REF} voltages (see Table 12.1).

A to D Control Register2 (ADCON2)

- Used to:
  - Select an acquisition time and clock frequency
  - Right or left justify output reading
- The output reading, after a conversion, is stored in the 16-bit register ADRESH and ADRESL. However, this is a 10-bit A/D converter leaving six bit positions unused.
- Bit7 ADFM enables the user either to right justify or left justify the 16-bit reading leaving the unused positions as 0s.

Interfacing a Temperature Sensor

- Temperature sensor
  - Transducer that converts temperature into an analog electrical signal
  - Many are available as integrated circuits, and their outputs (voltage or current) are, in general, linearly proportional to the temperature
  - However, output voltage ranges of these transducers may not be ideally suited to reference voltages of A/D converters
  - Therefore, it is necessary to scale the output of a transducer to range of the reference voltages of an A/D converter
  - Scaling may require amplification or shifting of voltages at a different level
- Problem statement
  - Interface the National Semiconductor LM34 temperature sensor to channel 0 (AN0) of the A/D converter module as shown in Figure 12.11.
  - Assume the output voltage of LM34 for the temperature range from 0ºF to 100ºF is properly scaled to 0 to +5 V.
  - Write instructions to start a conversion, read the digital reading at the end of the conversion, calculate the equivalent temperature reading in degrees Fahrenheit, convert it into BCD, and store the reading in ASCII code to the accuracy of one decimal point.
    - The expected range of temperatures is 0ºF to 99.9ºF.
- Figure 12-11 Interfacing LM34—Temperature Sensor to PIC18 A/D Conversion Module
- Hardware
  - Temperature transducer LM34
    - Three-terminal integrated circuit device that can operate in the +5 V to +30 V power supply range
    - Outputs 10 mV/ºF linearly
  - For the temperature range from 0ºF to +99.9ºF; the output voltage range is 0 to 1 V (rounded off to 100ºF).
- Scaling circuit
  - To get the full dynamic range of the A/D conversion for the output voltage range 0 to 1V of LM34:
    - We can connect $+V_{REF}$ to +1 V or
    - Scale the output voltage +1V to the voltage of the power supply +5 V as shown in Figure 12.11
  - This scaling enables us to connect PIC18 power supply V_{DD} as voltage reference $+V_{REF}$ and ground V_{ss} as $-V_{REF}$.

- Temperature calculations
  - A/D converter has 10-bit resolution
  - For temperature range 0ºF to +100ºF, the digital output should be divided into 1023 steps (0 to $3FF_H$).
  - Therefore, the digital value per degrees Fahrenheit is 10.23 ($1023/100 = 10.23_{10}$).
  - To obtain temperature reading from a digital reading of the A/D converter, the digital reading must be divided by the factor of 10.23.
- Software modules
  - Program should be divided into the following:
    - Start a conversion and read the digital reading at the end of the conversion.
    - Calculate the equivalent temperature reading.
    - Convert the result in BCD.
    - Convert the BCD numbers in ASCII code.

Digital to Analog (D/A, DAC, or D-to-A) Conversion

- Converting discrete signals into discrete analog values that represent the magnitude of the input signal compared to a standard or reference voltage
  - The output of the DAC is discrete analog steps.
  - By increasing the resolution (number of bits), the step size is reduced, and the output approximates a continuous analog signal.
- Figure 12–13a shows a simple hypothetical D/A converter circuit: three digital input lines with eight possible values: 000 to 111 and one output signal.
- The resolution of a DAC is defined in terms of bits—the same way as in ADC.
- The values of LSB, MSB, and full-scale voltages calculated the same way as in the ADC.
- Figure 12–13b shows the largest input signal 111 is equivalent of 7/8 of the full-scale analog value.

D/A Converter Circuits

- Can be designed using an operational amplifier and appropriate combination of resistors as shown in Figure 12–14a.
- Resistors connected to data bits are in binary weighted proportion, and each is twice the value of the previous one.
- Each input signal can be connected to the op amp by turning on its switch to the reference voltage that represents logic 1.
  - If the switch is off, the input signal is logic 0.
- If the reference voltage is 1 V, and if all switches are connected, the output current can be calculated as follows:

$$I_o = I_T = I_1 + I_2 + I_3 = \frac{V_{REF}}{R_1} + \frac{V_{REF}}{R_2} + \frac{V_{REF}}{R_3} = \frac{V_{REF}}{1k}\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8}\right) = 0.875\,\text{mA}$$

- Output voltage

$$V_O = -R_f I_T = -(1k)\times(0.875\text{ mA}) = -0.875\text{ V} = \left| {}^{7}\!/_{8}\,\text{V} \right|$$

- Now the formula can be extended to any number of bits as follows:

$$I_O = \frac{V_{REF}}{R_{REF}}\left(\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \ldots\ldots + \frac{A_n}{2^n}\right)$$

where R_{REF} is a reference resistor and A_1 to A_n can be either 0 or 1, A_1 is the MSB and A_n is the LSB. However, in a microprocessor, bit B7 (or D7) is the MSB, and in the above formula A1/2 will be replaced by B7/2.

D/A Converters as Integrated Circuits

- D/A converters are available commercially as integrated circuits
- Can be classified in three categories.
  - Current output, voltage output, and multiplying type
    - Current output DAC provides the current I_O as output signal
    - Voltage output D/A converts I_O into voltage internally by using an op amp and provides the voltage as output signal
    - In multiplying DAC, the output is product of the input voltage and the reference source V_{REF}.
  - Conceptually, all three types are similar
- Converter in Figure 12-15 is microprocessor-compatible 8-bit converter
- Has two control signals: WR (Write) and CS (Chip Select) and 8-input data lines
- In interfacing this converter with a microcontroller, no additional circuitry is required.
- To convert a binary code, the MPU needs to place a code on input data lines, access the converter by sending a low signal to CS, and write the code (or input code) by sending a low signal to WR signal

Generating a Ramp Waveform Using a D/A Converter

- Problem statement
  - Write a subroutine to generate a ramp waveform at the output of the D/A converter AD558 shown in Figure 12–16. It is a voltage-output 8-bit DAC.
  - The slope of the ramp should be variable based on the delay count provided by the caller.
- Hardware
  - The AD558 DAC requires ten signals from the microcontroller: 8 bits for data input and two bits for the control signals.
    - 8 bits of PORTC are connected as data input and the two bits RE0 and RE1 of PORTE are used as control signals.
    - All bits should be initialized as outputs
- Software
- To generate a ramp: the Output should begin with zero and the waveform should be increased incrementally.
  - Can be done by setting a register as an up-counter starting from zero and outputting that count.
  - The output will begin with zero volts. As counter is incremented and each count is outputted, the output waveform will increase in magnitude in a straight line with a slope depending upon the delay between counts.
  - When counter reaches the maximum value FF_H, it will be reset to 00 and start again. Therefore, the output will reach maximum value and start again from zero, generating a ramp waveform.

Lesson 8116 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. A __________ converts non-electrical signals into electrical signals and a __________ converts electrical signals into real life non-electrical signals.
   (1) transducer; A/D converter
   (2) A/D converter; D/A converter
   (3) D/A converter; transducer
   (4) transducer; transducer

2. A ________________ converts a digital signal into equivalent discrete analog signal.
   (1) transducer
   (2) A/D converter
   (3) D/A converter
   (4) analog

3. The resolution of a 16-bit A/D converter is ________________?
   (1) 15.2×10^{-6}
   (2) 1/2
   (3) 1/16
   (4) 15.2×10^{6}

4. If a 12-bit A/D converter has an analog signal from 0 to +10-V determine the LSB.
   (1) 5-V
   (2) 24.4-mV
   (3) 9.975-V
   (4) 10-V

5. If a 12-bit A/D converter has an analog signal from 0 to +10-V determine the MSB.
   (1) 5-V
   (2) 24.4-mV
   (3) 9.975-V
   (4) 10-V

6. If a 12-bit A/D converter has an analog signal from 0 to +10-V determine the full-scale output voltage.
   (1) 5-V
   (2) 24.4-mV
   (3) 9.975-V
   (4) 10-V

7. If the voltage range in Question 4 changed from -5-V to +5-V the value of the LSB would _______________?
   (1) increase
   (2) decrease
   (3) remain the same
   (4) become 0-V.

8. If the reference voltages in PIC18 A/D modules are V_{REF-} = 0-V and V_{REF+} = +5-V, what is the output voltage for the digital value of 80_H?
   (1) 625-mV
   (2) 4.97-V
   (3) 2.5-V
   (4) 0-V

9. Based on the information provided in Question 8 what is the output voltage for the digital value of $03FA_H$?

(1) 4.97-V	(3) 2.5-V
(2) 625-mV	(4) 0-V

10. An A/D converter will convert an energy signal into its digital signal equivalent.

(1) True	(2) False

11. If the reference voltages in PIC18 A/D module are V_{REF}- = -1-V and V_{REF+} = +5-V, what would be the value of the voltage per bit?

(1) -1-V	(3) 5.859-mV
(2) +5-V	(4) 2.5-mV

12. If a 12-bit A/D converter has an analog signal from -5-V to +5-V determine the MSB.

(1) -5 V	(3) 4.975 V
(2) 0 V	(4) 10 V

13. Referencing Question 12, determine the full-scale output voltage.

(1) -5 V	(3) 4.975 V
(2) 0 V	(4) 10 V

14. If the reference voltages in PIC18 A/D module are V_{REF}- = -1-V and V_{REF+} = +5-V, determine the output voltage for the digital value of 55H.

(1) -502 mV	(3) -3.9 mV
(2) -498 mV	(4) -1 V

15. If the reference voltages in PIC18 A/D module are V_{REF}- = -1-V and V_{REF+} = +5-V, determine the output voltage for the digital value of AAH.

(1) -498 mV	(3) +4.9 V
(2) -3.9 mV	(4) +6 V

End of Examination

Lesson 8118-Input/Output (I/O) Ports, Interfacing, and Serial I/O
Microcontrollers and Embedded Systems

Assignment Checklist

- ✓ Read Chapter 9 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Chapter 13 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Lesson 8118 in this Study Guide
- ✓ Take the Examination for this lesson

<u>Chapter 9 Lecture/Discussion</u>

Basic Concepts in I/O Interfacing and PIC18 I/O Ports

- I/O devices (or peripherals) such as LEDs and keyboards are essential components of the microprocessor-based or microcontroller-based systems.
  - Classified into two groups
    - input devices
    - output devices
- Input devices
  - Provide digital information to an MPU
  - Examples: switch, keyboard, scanner, and digital camera
- Output devices
  - Receive digital information from an MPU
  - Examples: LED, seven-segment LED, LCD, and printer
- I/O devices interfaced (connected) to an MPU using I/O ports as shown in Figure 9-1.

I/O Ports: Interfacing and Addressing

- I/O ports
  - Buffers and latches on the MCU chip
  - Assigned binary addresses by decoding the address bus
    - MPU accesses I/O ports through these addresses.
  - Peripherals such as keyboards/LEDs connected to the MCU through I/O ports
- Generally bidirectional meaning they can be set up either as input ports (buffers) or output ports (latches) through internal data direction registers
  - Input peripherals connected to the MCU using buffers
  - Output peripherals connected using latches
- Most designed to perform multiple functions
  - Functions determined by writing specific bits to their control registers
- To read (receive) binary data from an input peripheral
  - MPU places the address of an input port on the address bus, enables the input port by asserting the RD signal, and reads data using the data bus.
- To write (send) binary data to an output peripheral
  - MPU places the address of an output port on the address bus, places data on data bus, and asserts the WR signal to enable the output port.

Writing to and Reading from I/O Ports

- Writing to the port
  - When the MPU sends out or transfers data to an output port
- Reading from the port
  - When the MPU receives data from an input port
- To write to an output port
  - The MPU places address of the port on the address bus, selects the port using the decoder, places the data to be sent out on the data bus, and asserts the Write control signal to enable the output port (latch).
    - The latch latches the data and displays at the LEDs as shown in Figure 9-1.
- To read from an input port
  - The MPU places address of the port on the address bus, selects the port using the decoder, asserts the Read control signal to enable the input port, and receives the data via the data bus.

PIC18F452/4520 I/O Ports

- MCU includes five I/O ports
  - PORTA, PORTB, PORTC, PORTD, and PORTE
- Ports are multiplexed meaning they can be set up by writing instructions to perform various functions as shown in Figure 9-2.

PORTA: Example of Multiple Functions

- Digital I/O: RA6-RA0
- Analog Input: AN0-AN4
- $V_{REF}+$: A/D Reference Plus Voltage
- $V_{REF}-$: A/D Reference Minus Voltage
- TOCK1: Timer0 Ext. Clock
- SS: SPI Slave Select Input
- LVDIN: Low voltage Detect Input
- Each I/O port is associated with the special functions registers (SFRs) to setup various functions.
  - Can be set up as entire ports or each pin can be set up.
    - PORT: This register functions as a latch or a buffer determined by the logic levels written into the associated TRIS register.
    - TRIS: This is a data direction register. Writing logic 0 to a pin sets up the pin as an output pin, and logic 1 sets up the pin as an input pin.
    - LAT: This is an output latch similar to PORT.
- Figure 9-3 shows the internal block diagram of PORTB in the simplified form. It includes:
  - Three internal D flip-flops (latches)
    - Data latch to output data
    - TRIS latch to setup data direction
    - Input latch for input data

Interfacing Output Peripherals

- Commonly used output peripherals in embedded systems are
  - LEDs, seven-segment LEDs, and LCDs; the simplest is LED

- Two ways of connecting LEDs to I/O ports:
  - LED cathodes are grounded and logic 1 from the I/O port turns on the LEDs - Figure 9-4 (a). The current is supplied by the I/O port called current sourcing.
  - LED anodes are connected to the power supply and logic 0 from the I/O port turns on the LEDs - Figure 9-4 (b). The current is received by the chip called current sinking.

Interfacing Seven-Segment LEDs as an Output

- Seven-segment LEDs
  - Often used to display BCD numbers (1 through 9) and a few alphabets
  - A group of eight LEDs physically mounted in the shape of the number eight plus a decimal point as shown in Figure 9-5 (a)
  - Each LED is called a segment and labeled as 'a' through 'g'.
- Two types of seven-segment LEDs
  - Common anode
  - Common cathode
- In a common anode seven-segment LED
  - All anodes are connected together to a power supply and cathodes are connected to data lines
- Logic 0 turns on a segment.
- Example: To display digit 1, all segments except b and c should be off.
- Byte 11111001 = F9H will display digit 1.

****Note Figure 9-5 on Page 266 mislabeled the Logic Diagram as it should say "Common Anode" at the top of the diagram rather than common cathode.

- In a common cathode seven-segment LED
  - All cathodes are connected together to ground and the anodes are connected to data lines
- Logic 1 turns on a segment.
- Example: To display digit 1, all segments except b and c should be off.
- Byte 00000110 = 06H will display digit 1.

Interfacing Input Peripherals

- Commonly used input peripherals in embedded systems are: DIP switches, push-button keys, keyboards, and A/D converters.
- DIP switch: One side of the switch is tied high (to a power supply through a resistor called a pull-up resistor), and the other side is grounded. The logic level changes when the position is switched.
- Push-button key: The connection is the same as in the DIP switch except that contact is momentary.

Internal Pull-Up Resistor

- Figure 9-8 (a) shows that the pull-up resistors are connected externally. However, PORTB can provide equivalent resistors internally through initialization.

- FIGURE 9-9 (a) shows that turning off the internal FET is equivalent to providing a pull-up resistor.
- Bit7 (RBPU) in the INTCON2 register enables or disables the pull-up resistor (Figure 9-9 b).
    - Instruction to Enable Pull Up Resistors:
      BCF INTCON2 7, 0

Interfacing Push-Button Keys

- Electrical connection of a push-button key is same as that of a DIP switch (Figure 9-10 a) except that the connection is temporary when the key is pressed.
    - When a key is pressed (or released), mechanical metal contact bounces momentarily as shown in Figure 9-10 (b) and can be read as multiple inputs.
    - The reading of one contact as multiple inputs can be eliminated by a key-debounce technique, using either hardware or software.

****Note on Page 279 of Figure 9-10 (b) the caption is incorrect as it should read “Push-Button Key Bounce”

Key Debounce Techniques

- Hardware technique
    - Figure 9-11 shows two circuits, based on the principles of generating a delay and switching the logic level at a certain threshold level.
    - Figure 9-11 (a) shows two NAND gates connected back to back, equivalent of a S-R latch. The output of the S-R latch is a pulse without a bounce.
    - Figure 9-11 (b) shows an integrated circuit (MAX 6816) that bounces the key internally and provides a steady output.

Illustration: Interfacing Push-Button Keys

- Problem statement
    - A bank of push-button keys are connected as inputs to PORTB (Figure 9-12). The pull-up resistors are internal to PORTB.
    - Write a program to recognize a key pressed, debounce the key, and identify its location in the key bank with numbers from 0 to 7.
- Hardware (Figure 9-12)
    - PORTB should be set up as input port
    - Internal pull-up resistors should be enabled
- Software
    - Checking a key closure (Figure 9-12)
    - Debouncing the key
    - Encoding the key
- Checking a key closure
    - When a key is open, the logic level is one and when it is closed, the logic level is zero.
    - When all keys are open, the reading will be FFH, and when a key is closed, the reading will be less than FFH.
        - Therefore, any reading less than FFH indicates a key closure.

- Debouncing the key
  - Software technique
    - Wait for 20 ms.
    - Read the port again.
    - If the reading is still less than FFH, it indicates that a key is pressed.
- Encoding the key
  - No direct relationship between the binary reading generated when a key is pressed and its position in the key bank
  - Key closure can be identified by rotating the reading right and looking for 'No Carry' and counting the rotations
    - Number of rotations will indicate the key position

****Note on Page 284 line 3 of 9.5.3 Program contains an error. Line 3 should be

```
        BCF     INTCON2, 7, 0
```

Interfacing LCD (Liquid Crystal Display)

- Problem statement
  - Interface a 2-line x 20 character LCD module with the built-in HD44780 controller to I/O ports of the PIC18 microcontroller.
  - Explain the control signals necessary to read from and write to the LCD.
  - Write a program to display ASCII characters.
- Hardware
  - 20 x 2-line LCD displays (two lines with 20 characters per line)
  - LCD has a display Data RAM (registers) that stores data in 8-bit character code.
  - Each register in Data RAM has its own address that corresponds to its position on the line.
    - The address range for Line 1 is 00 to 13H and Line 2 is 40H to 53H.
- Driver HD77480 (Figure 9.15 a)
  - 8-bit data bus (RD7-RD0)
  - Three control signals:
    - RS – Register Select (RA3)
    - R/W – Read/Write (RA2)
    - E – Enable (RA1)
  - Three power connections
    - Power, ground, and the variable register to control the brightness
- Can be interfaced either in the 8-bit mode or the 4-bit mode
  - In the 8-bit mode, all eight data lines are connected for data transfer
  - In the 4-bit mode, only four data lines (DB7-DB4 or DB3-DB0) are connected and two transfers per character (or instruction) are needed
- Driver has two 8-bit internal registers
  - Instruction Register (IR) to write instructions to set up LCD
  - Data Register (DR) to write data (ASCII characters)
- LCD Operation
  - When the MPU writes an instruction to IR or data to DR, the controller:
    - Sets the data line DB7 high as a flag indicating that the controller is busy completing the operation

        - Sets the data line DB7 low after the completion of the operation
    - The MPU should always check whether DB7 is low before sending an instruction or a data byte
    - After the power up, DB7 cannot be checked for the first two initialization instructions.
- Writing to or reading from LCD (Figure 9-15 b)
    - The MPU:
        - Asserts RS low to select IR
        - Asserts RS high to select DR
        - Reads from LCD by asserting the R/W signal high
        - Writes into LCD by asserting the R/W signal low
        - Asserts the E signal high and then low (toggles) to latch a data byte or an instruction
- Software
    - To write into the LCD, the program should:
        - Send the initial instructions (commands) before it can check DB7 to set up the LCD in the 4-bit or the 8-bit mode.
        - Check DB7 and continue to check until it goes low.
        - Write instructions to IR to set up the LCD parameters such as the number of display lines and cursor status.
        - Write data to display a message.

****Note on Page 294 of 9.6.3 Program under section LCD_DATA: line 1 should read
    ;Save ASCII code in TEMP register

Interfacing a Matrix Keyboard
- Problem statement
    - Interface a 4 x 4 Hex keypad to PORTB.
    - Write a program to recognize a key pressed and encode the key in its binary value.
- Hardware
    - 4 x 4 matrix keypad organized in the row and column format
        - Four rows and four columns (Figure 9-16)
    - Four rows are connected to the lower half of PORTB (RB0-RB3)
    - Four columns are connected to upper half of PORTB (RB4-RB7)
    - Internal pull-up resistors are turned on
    - When a key is pressed, it makes a contact with the corresponding row and column
- Software
    - To recognize and encode the key pressed, the program should:
        - Ground all the columns by sending zeros.
        - Check each key in a row for logic zero.
        - Ground one column at a time and check all the rows in that column.
        - Once a key is identified, it is encoded based on its position in the column.

****Note on Page 298 of 9.7.3 Program under section SETUP: line 3 should be

```
BCF    INTCON2, 7, 0
```

Interfacing Seven Segment LEDs—Time Multiplex Scanning Technique

- Problem statement (****Note this is how 9.8.1 Problem Statement should read****)
  - Interface four common **cathode** seven-segment LEDs to PORTB and PORTC for display using the time multiplex scanning technique.
  - Write instructions to display a four-digit number stored in data registers.
- Hardware (Figure 9-17)
  - Eight data lines of PORTB are connected to the anodes of eight segments (a-g) of each LED
  - Each cathode is connected to PORTC (RC0-RC3) through a transistor
  - Transistors (and LEDs) can be turned on by sending logic 1 to the base of a transistor
  - Each LED is turned on and off in a sequence to display a digit
- Software

  To turn on the four LEDs in a sequence for a continuous display:
  - Codes of the numbers to be displayed are stored in data registers in a sequence.
  - The program gets the codes from the data registers by using the pointer (FSR) and sends them out to the LED segments through PORTB.
  - One LED at a time is turned on by sending logic 1 to the corresponding transistor connected to PORTC.
  - After an appropriate delay, the first LED is turned off and the next LED is turned on.
  - Turning LEDs on/off is repeated in a sequence.

Chapter 13 Lecture/Discussion

Serial I/O

- Parallel I/O
  - The 8-bit microcontroller processes data in groups of 8 bits and transfers (copies) data to peripherals using the entire data bus.
    - This is called parallel I/O or parallel communication.
  - However, it is not always possible to use the entire data bus to transfer data to and from a peripheral or over long distance communication such as telephone lines.
- In serial I/O (or communication) one bit is transferred at a time over one line.
- To transfer data serially, one bit at a time, the microcontroller must convert its parallel word into a stream of serial bits.
  - Called parallel-to-serial conversion
- Similarly, to receive data serially, the microcontroller must receive one bit at a time and convert a stream of serial bits into a parallel word.
  - Called serial-to-parallel conversion

Basic Concepts in Serial Communication

- For serial communication, a parallel word must be converted into a stream of bits to send the data out over one line.
- Transmitter
  - The device that initiates the data transfer
- Receiver
  - The device that receives the data
- This transmission process raises various issues as follows:
  - Synchronous versus Asynchronous Serial Communication
  - Simplex versus Duplex Transmission
  - Rate of Transmission
  - Transmission Errors and Error Checks
  - Standards and Protocol

Synchronous Versus Asynchronous Serial Communication

- The serial communication between a receiver and a transmitter is classified primarily into two formats.
  - Synchronous and asynchronous
- Synchronous
  - In this format, the receiver and the transmitter have the same clock. Figure 13–1 (a) shows a typical synchronous transmission format that begins with sync characters or clock followed by data.
  - Until the 1980s, the synchronous format was used primarily in high- speed transmission. Since the advent of the microcontroller, the synchronous format is commonly used for the serial communication between the microcontroller and its peripherals.
- Asynchronous
  - Format shown in Figure 13–1 (b)
  - Communication can happen any time.

        - Transmitter must let receiver know when it begins to transmit and when it ends.
        - Figure 13–1 (b) shows that the “Start” and “Stop” bits are included as a part of the transmission.
    - Typical application includes communication between a PC (personal computer) and a modem at an agreed upon clock rate.
- Figure 13-1 (a) Synchronous Data Transfer
- Figure 13-1 (b) Asynchronous Data Transfer

Simplex Versus Duplex Transmission

- Terms refer to the direction and simultaneity of data flow
- Simplex
    - Data flow in only one direction, such as from a PC to its peripheral, in serial format
- Full duplex
    - Data flow in both directions simultaneously, such as a telephone conversation or communication via a modem
- Half duplex
    - Data flow in both directions, but only one direction at a time, such as a conversation over a CB radio

Rate of Transmission

- Defined either in terms of bits per second or the unit called “baud”; sometimes they are equal
- Baud and bits per second (BPS)
    - In telecommunications and electronics, baud is a measure of the “signaling rate,” which is the number of changes to the transmission medium per second in a modulated signal.
    - At slow speeds, only one bit of information is encoded in each electrical change. In this case, the baud and the bits per second are equal.
    - At higher speeds, multiple bits are encoded in one electrical change. Therefore, data transmission rates are generally expressed in bits per second (bps).

Transmission Errors and Error Checks

- Errors commonly checked
    - Framing error
        - Occurs when Start and Stop bits improperly frame a character
        - Recognized when the Stop bit is zero which is expected to be one
        - Serial I/O chip, generally, provides bit or flag in its control register to check for this error
    - Overrun error
        - Occurs when a new byte overwrites the earlier byte before the receiver completes the reading of the earlier byte
        - Can also be verified in the control register of a serial I/O chip
- Parity error
    - In some transmission, last bit is used as a parity check

  - Technique is based on the principle that the characters (data) are transmitted as either an even or odd number of 1s
  - ASCII is a seven-bit alphanumeric code; the last bit, B7, of a byte is always zero, and in some systems, the last bit is used for parity information.

    Example: Parity Check
    - The ASCII letter "Y—" 0101 1001 (59H)—has four 1s, which is even.
    - If the transmission is set up with even parity, the letter "Y" is transmitted as 59H. If the transmission is set up with odd parity, odd number of 1s must be transmitted. Bit B7 is changed to 1 and the byte is sent as 1101 1001 (D9H).
- CheckSum
  - Used when blocks of data are transmitted
  - All bytes to be transmitted are added (without a carry), and 2's complement of sum is calculated, which is sent as the last byte, called checksum.
  - Receiver program adds all the bytes including the checksum (2's complement), and if the sum is zero, the receiver accepts the data. Otherwise, the receiver program generates an error message.

Standards and Protocols in Serial Data Transfer
- Equipment needed for serial data transfer are designed and manufactured by various companies
- Common understanding must exist that guarantees compatibility among different types of equipment
  - Common understanding or guideline is known as a standard.
  - A defined operational procedure is known as protocol.
- Standard
  - May include various aspects such as assignments of pin connections, speed of transmission, format of transmission, and mechanical specifications
- Protocol
  - May include a certain sequence of steps or a procedure to follow.
- EIA-232 (formerly known as RS-232)
- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I^2C)
- One-wire (1-Wire®) Bus
- Controller Area Network Bus (CAN)
- Local Interconnect Network (LIN)

EIA-232 AND PIC18 Serial Communication Module USART
- Includes two serial communication modules
  - Addressable Universal Synchronous Asynchronous Receiver Transmitter (USART) and Master Synchronous Serial Port (MSSP)
  - USART is also known as Serial Communications Interface (SCI).
  - MSSP is also known as Serial Peripheral Interface (SPI).

EIA-232 (RS-232) Serial I/O Standard
- RS-232 (Revised Standard 232, now known as EIA-232)

    - Standard for serial binary data interconnection between DTE (data terminal equipment) such as a computer or a terminal, and DCE (data communication equipment) such as a modem or a router
    - Standard defines electrical, mechanical, and functional characteristics for interfacing communication equipment
- Negative logic
    - Logic 1: –3 V to –25 V or Mark
    - Logic 0: +3 V to + 25 V or Space.
- Connector
    - DB25 subminiature with twenty-five pins
- Signals are divided into four groups
    - Data, control, timing, and ground
- Signals are defined in reference to DTE (data terminal equipment)
- Modern equipment, including the PC, uses positive TTL/CMOS logic levels, and +5 V power supply.
    - These signals cannot be connected directly to EIA-232 interface circuitry.
    - Signals need to be converted into negative logic and shifted to higher voltages by using line drivers and line receivers called transceivers.
    - Figure 13–2 shows minimum connections—transmit, receive, and ground—required to connect a DTE and a DCE.
- Figure 13–2 shows that the transceiver changes:
    - TTL voltage +3.4 V to EIA-232 level –9 V and again back to the TTL level at the receiver side.
    - Similarly, 0.2 V (logic 0) is changed to +9 V and back to 0.2 V at the receiver side.
- The PC serial communication is primarily asynchronous.
- Many of the signals on the DB25 connector are unnecessary in asynchronous communication.
- Seven to nine signals are necessary in asynchronous communication.
- DB9 connector has nine pins.
- Figure 13–3 shows the 25-pin and 9-pin connectors with the associated signals.

EIA-232 Transmission Format (Framing) in the Asynchronous Mode

- Transmission consists of a Start bit, seven or eight data bits, optional parity bit, and one or two Stop bits.
- Figure 13-4 shows transmission format of the ASCII character "Y" (59H).
    - It begins with "Start" bit low (Logic 0), followed by bits of the ASCII character "Y" beginning with LSB B0,and terminates with "Stop" bit(s) high (Logic 1).
    - The number of "Stop" bits used to be either 1, 1½, or 2. Currently, one Stop bit is almost universal.
    - Format shown in Figure 13-4 is called Framing; logic 1 is called Marking; and logic 0 is called Space.

PIC18 USART Module

- The PIC18 family of microcontrollers includes the USART module.
- Newer versions include an enhanced version of the USART module, called EUSART.

- Serial USART module can be configured in the following modes:
  - Asynchronous: full duplex
  - Synchronous:– master half duplex
  - Synchronous:– slave half duplex
- Commonly used in PC serial communication in the asynchronous mode with the EIA-232 protocol
- Also used in the half duplex synchronous mode to communicate with peripherals, such as A/D and D/A converters, and with serial EEPROM
- Registers used to set up and control communication are:
  - SPBRG: Baud Rate Generator
  - TXSTA: Transmit Status and Control
  - RCSTA: Receive Status and Control
  - BAUDCON: Baud Rate Control

Data Transmission

- Serial Bit Transmission (Figure 13–5)
  - Uses pin RC6 (TX) of PORTC
  - Two types of protocols – 8-bit and 9-bit
  - Once a data byte (character) is written into TXREG, bits are moved into transmit shift register and clocked out onto TX pin
  - Each byte is preceded by a Start bit and followed by a Stop bit
  - Transmit shift register enables the MPU to write new data while the previous data bits are being transmitted.
  - TXSTA register (Figure 13–6) controls data transmission
  - 9-bit protocol:
    - Used in multiprocessor communication where a PC as a host communicates with many microcontrollers
    - Ninth bit must be placed in TX9D bit position of TXSTA register (Figure 13–6) before writing eight bits into TXREG register

Data Reception

- Serial Bit Reception (Figure 13–7)
  - Uses pin RC7 of PORTC
  - Can be either eight or nine data bits
- When USART detects Start bit, receive shift register receives either eight or nine bits, one bit at a time.
- After checking Stop bit, USART moves data bits into buffer and then into RCREG.
- RCSTA register controls data reception (Figure 13–8).
- For the nine-bit reception, ninth bit is placed in RX9D-bit position of the RCSTA register.
  - The receive shift register and buffer enable the MPU to read data from RCREG while receiving data in the receive shift register.

Baud Rate Control Register (BAUDCON)

- Is used to detect and calibrate baud rate automatically
- Can also specify 16-bit register for baud rate generation

- Only available in EUSART (Figure 13–9)
- Baud (the rate of data transfer) must be specified in the 16-bit register SPBRGH:SPBRG.

****Note on Page 415 of Example 13.2 in section EUSART line 1 should be
;Set RC7 and RC6 <7-6> as inputs

Data Transmission and Reception Methods: Checking Flags Versus Interrupts

- Two methods of implementing serial communication
  - Checking flags
  - Interrupts
- Flags used
  - TXIF (Bit4) and RCIF (Bit5) in Peripheral Interrupt Request Register1 (PIR1)
    - TXIF (Bit4): USART Transmit Interrupt Flag – this bit is 1 when TXREG (Transmit Register) is empty. When a byte is written in TXREG, this flag is set to 0 indicating that TXREG is full.
    - RCIF (Bit5): USART Receive Interrupt Flag – this bit is 1 when RCREG (Receive Register) is full. When a byte is read from RCREG, this flag is set to 0 indicating that RCREG is empty.
- Flags can be checked in software loops or used to interrupt the MPU.
- In the software loops, the MPU will be kept busy waiting in the loops and unavailable for the execution of any other instructions.
- In the interrupt process, the reception and the transmission can be done in the background.
- Registers used to set up the interrupt process:
  - Interrupt Control Register (INTCON)
  - Peripheral Interrupt Enable Register1 (PIE1)
  - Reset Control Register (RCON) to set priorities
- Bits used to set up the interrupt process are as follows:

  INTCON Register

****Note on Page 416 the textbook has an error in this section. In the textbook the word "bit" was replaced with (H).

  - GIE/GIEH (Bit7)
    - Global Interrupt Enable bit: When Bit7 is 1, it enables the global interrupts, and when the bit 0, it disables all interrupts.
  - PEIE/ GIEL (Bit6)
    - Peripheral Interrupt Enable bit: When Bit6 is 1, it enables the peripheral interrupts, and when the bit 0, it disables the peripheral interrupts.
  - IPEN Bit7
    - If the Reset Control Register (RCON) is set to 1, the priority levels for all interrupts are enabled.

  PIE1

  - RCIE (Bit5): USART Receive Interrupt Enable
    - When Bit5 is 1, it enables the interrupt, and when this bit is 0, it disables the interrupt.
  - TXIE (Bit4): USART Transmit Interrupt Enable

        - When Bit4 is 1, it enables the interrupt, and when this bit is 0, it disables the interrupt.

Serial Peripheral Interface (SPI)

- SPI is a serial synchronous data exchange protocol between a master and a slave device.
    - Commonly used for high-speed serial communication between a microcontroller and its peripheral devices such as EEPROMs, data converters, and display drivers.
    - Basic concepts of the SPI are as follows:
        - Four-wire Interface—includes four signals: clock, data in, data out, and slave select
        - Synchronous protocol—clock signal is provided and controlled by the master device, and data are sent with clock pulses
        - Master-Slave protocol—master device controls the clock line (SCK), and can communicate with multiple slave devices; only one master in the SPI protocol
        - Data Exchange protocol—each device (master and slave) has two data lines: one for input and one for output
    - Data always exchanged between the devices; as data being transmitted with the MSB first, new data are being received.
        - A device cannot be just a transmitter or a receiver.
- Master selects a slave.
    - Slave device is selected by a signal from the master, and signal is generally known as Chip Select (CS) or Slave Select (SS).

- Data exchange and clock
    - Data typically change during rising or falling edge of the clock. Sampling and data change take place on opposite edge of clock.

PIC18 Master Synchronous Serial Port (MSSP) Module

- A serial interface used in communicating with other peripheral devices
- Can operate in one of the two modes
    - Serial Peripheral Interface (SPI)
    - Inter-Integrated Circuit (I^2C)

SPI Mode

- The SPI module transmits and receives data simultaneously in the synchronous mode.
- Creates a loop (Figure 13.10)
- Uses following first three pins, and fourth pin in the Slave mode of operation:
    - Serial Data Out (SDO) – pin RC5/SDO on PORTC
    - Serial Data In (SDI) – pin RC4/SDI/SDA on PORTC
    - Serial Clock (SCK) – pin RC3/SCK/SCL on PORTC
    - Slave Select (/SS) – Available pin on a PORT
        - The SS pin is active low. It selects a slave device to communicate with; it is generally used in configurations that call for multiple slaves. This pin can be any available pin on an I/O port.
- Figure 13.10 is a simplified block diagram of the MSSP module.

- SSPSR
  - It is a shift register; it shifts data out serially on SDO pin and receives data in on SDI pin.
- SSPBUF
  - It is a buffer register, used to write a byte to or read a byte from.
- After the exchange of a byte between the master and the slave, the module copies the byte in the SSPBUF register, and the user program must read the SSPBUF register before transmitting the next byte.
- The Write and Read operations must be performed for each byte; some of the data bytes may not have any use in a given application. Therefore, the data transmission in the SPI mode can be divided in three categories:
  - Transmission: the master sends data and receives dummy data from the slave
  - Transmission and Reception: both exchange data
  - Reception: the master receives data and sends dummy data to the slave

Control of Data Exchange in MSSP Module

- Data exchange operation in the SPI mode is controlled by two registers:
  - SSPCON1: MSSP Control Register1 (Figure 13-11)
    - This control register is used primarily to enable the pins for serial communication <SSPEN -Bit5>, to select the clock polarity <CKP – Bit4>, and the serial mode (such as master or slave <Bit3 – Bit0>).

  - SSPOV <Bit6>
    - Used to indicate an overflow in the Slave while receiving data. An overflow occurs whenever the master writes a byte before the previous byte has been read from the SSPBUF. If the SSPOV bit is set, it must be cleared by the user program.

- SSPSTAT: MSSP Status Register (Figure 13.12)
  - Used in both SPI and I^2C modes
- SPI mode
  - SSPSTAT register is used to specify when to sample data (at the middle or at the end of the output line, <Bit7>) when to transmit data (CKE – <Bit6>), and to indicate the status of the buffer register (<BF – Bit0>)
- I^2C mode
  - Five bits of this register are used exclusively in the I^2C mode

****Note on Page 423 of Example 13.5 in the section SPI_MS in the comments section move the word pulse up one line to ;rising pulse. The word “rising” in line 7 should not be alone.

SPI Applications

- The SPI
  - A synchronous serial protocol and requires only four signals (pins)
  - Slower than the parallel mode, but many devices do not need high-speed communication
- Many manufacturers have designed peripherals that are compatible with the SPI.

- SPI-compatible peripherals include
  - Converters (ADC and DAC), memories (EEPROM and Flash), sensors (temperature and pressure), and chips such as Real Time Clock (RTC), display (LCD), and shift registers
- SPI operates in the master-slave configuration—one master and multiple slaves.
- Slaves can be connected in two different formats as shown in Figure 13–14a and b.
- Figure 13–14a shows one master with independent slaves.
  - Each slave is supplied with a clock and connected with the same SDO and SDI signals.
  - Each slave is accessed separately by using separate lines for SS.
  - These lines are shown as R0 to Rn where R represents an output line of an I/O port and n represents the number of output lines used for n slave devices.
  - When a slave is not selected, it remains high, and the master can supply the data only to the selected slave.
- The configuration in Figure 13–14 (b) is used when the same data should be supplied to all the slaves.
- Data are sent to k devices – from one device to the next.

The Inter-integrated Circuit (I^{2}C) Protocol

- The features of the protocol are as follows:
  - Two-wire interface
    - This is a two-wire interface that has two open drain/collector lines, one for clock and one for data.
  - Synchronous
    - Data are transferred with a synchronous clock initiated by the master device. The rate of data transfer is 100 kbps in the standard mode and 400 kbps in the fast mode.
  - Master/Slave or Many Masters
    - Bus may have one master and many slaves, or multiple masters.
- SCL (clock) and SDA (data) lines
  - Lines are pulled high by resistors and stay high in the idle state
  - Clock controls the data transfer, and data transfer is bidirectional. All changes in the SDA (data) line must occur when the SCL (clock) line is low.
- 8-bit data transfer
  - The master initiates all data transfers and transmits 8-bits starting with the MSB.
- Addressing
  - Two types of addresses are used: 7-bit or 10-bit. The SDA line is also used to send the addresses of the slaves.
- Framing
  - Data transfer from master includes the following bits, Start (S), Address or Data (D), Stop (P), and the slave sends Acknowledge (A) or does not acknowledge (A).
- Figure 13.16 (a) shows data transfer from the master to the slave, and acknowledge (A or A) signal from the slave to the master.
- Figure 13.16b shows the master reading the slave, and data transfer from the slave to the master.

The PIC18 MSSP Module in the I^2C Mode

- Implements all master and slave functions and provides interrupts on Start and Stop bits in hardware to determine whether the bus is free when the multimasters are using the bus
- Supports both the 7-bit and the 10-bit addressing mode.
- Uses the following pins:
  - SCL (Serial Clock): pin RC3/SCK/SCL on PORTC
  - SDA (Serial Data) : pin RC4/SDI/SDA on PORTC
- These pins must be setup as inputs so that the I^2C control circuitry can control the directions as necessary.
- The following control and status registers are used to configure the MSSP module in the I^2C mode.
  - SSPCON1) (MSSP Control Register1)
  - SSPCON2 (MSSP Control Register2)
  - SSPSTAT (MSSP Status Register)
- Figure 13.17 (a), (b), and (c) show the definitions of the bits in these three registers.
- The other registers necessary to implement the I^2C mode are shown in Figure 13.18.
  - SSPADD (MSSP Address Register)
    - Holds the slave device address when the SSP is in the slave mode.
    - In the master mode, it holds baud rate generator reload value, which is only seven bits (without the MSB).
  - SSPSR (Serial Receive/Transmit Buffer Register)
    - This is a shift register to transmit and receive bytes.
  - SSPBUF (Buffer Register)
    - The MPU writes bytes to this register for transmission and reads bytes for reception.

****Note on Page 436 in Example 13.9 the section TRANSMT has a small tab issue in line 4 of the code. It has HERE: BRA HERE when it is more correct to express this line as

```
HERE:       BRA HERE
```

Interfacing Serial EEPROM to the PIC18 MSSP Module in the SPI Mode

- In embedded systems, a microcontroller with limited memory can expand its memory size by adding a serial EEPROM.
- Advantages
  - Small size, low number of I/O pin requirement, byte level flexibility, low power consumption, and low cost
  - The primary benefit is that signal timings are handled by hardware rather than software.
    - This enables system to continue program execution while serial communication is handled in the background.
- Problem statement
  - Interface Microchip 25XX040 Serial EEPPROM to the PIC18F452/4520 MSSP module in the SPI mode.
  - Explain the serial communication process between the memory chip and the microcontroller module.

  - Write instructions to transfer 16 bytes from the microcontroller data registers to the EEPROM.
- Hardware: Electrically Erasable PROM (EEPROM)
- The Microchip 25XX040 is an 8-pin device with 4 Kb memory organized as 512 x 8.
  - The address range for 512 bytes is 000 to 1FFH, and the size of the page is 16 bytes.
  - The data transfer between memory and the microcontroller MSSP module takes place via a Serial Peripheral Interface (SPI) bus.
    - The bus requires only three signals (Figure 13–19): Serial Clock (SCK), Serial Input (SI), and Serial Output (SO).
- The memory chip is accessed through Chip Select (CS) signal.
- Its HOLD signal is used to stop the data transfer.
- The Write Protect (WP) can prevent from writing to the chip.
- The power supply V_{CC} range for 25XX040 is 2.5 V – 5.5 V.
- The memory chip has an 8-bit instruction register with six instructions (commands).
- When the CS is low, data bits are clocked in on the rising edge of the clock (SCK) assuming HOLD and WP are inactive (high).
- The chip includes the Read Status Register (RDSR), and bits of this register are defined in Figure 13–20.

- Steps in Write Operation
  - Enable latch by sending the Write Enable (WREN) instruction. Assert CS low, send the WREN instruction, and disable the CS by setting it high (Figure 13–21).
    - The MSSP module asserts CS low, and latch-enable command (06_H) is shifted in the memory chip on the SI line synchronized with the clock.
  - Read the status register to check that Write-in-Process (WIP) is no longer active.
  - Program must stay in the loop until the WIP bit goes low.
    - Figure 13–21b shows that the command to read status (05_H) is on the SI line, and the data from the status register is sent out on the SO line.
- Write Operation (cont'd)
  - Send a byte (or a page, which = 16 bytes) to EEPROM.
  - Assert CS low, and send the WRITE instruction, followed by the address and data (Figure 13–21c).
  - 16 bytes can be written before deactivating the CS signal.
- During the Write Operation, status register RDSR can be read to check any of the status bits. When the write cycle is completed, the Write Enable latch is reset.
- Read Operation
  - Access the memory chip. Assert CS low.
  - Send Read Instruction followed by the address.
  - Data shifted out over the SO pin, and the operation continues until the CS is asserted high.
  - Timing diagram of a read operation is the same as in Figure 13-21c except that the command byte is 03_H.

Interfacing Serial EEPROM to the PIC18 MSSP Module in the I^2C Mode

- The I^2C Mode
  - Two-wire interface consisting of SCL (Clock) and SDA (Data)
  - Requires fewer pins than the SPI mode
  - Advantages similar to those of the SPI mode
  - Many devices, such as the I/O expander, D/A converter, and digital thermometers, compatible with this bus, are available
  - Primary benefit of using the MSSP module is the signal timings handled by hardware rather than software
- Problem statement
  - Interface the Microchip 24LC024 Serial EEPPROM to the PIC18F452/4520 MSSP module in the I^2C mode.
  - Explain the serial communication process between the memory chip and the microcontroller module.
  - Write instructions to transfer a byte from the microcontroller to the EEPROM.
- I^2C Compatible Electrically Erasable PROM (EEPROM):
  - The Microchip 24LC024 is an 8-pin device with 2 Kb memory organized as 256 x 8.
  - The address range for 256 bytes is 00 to FF_H, and the size of the page is 16 bytes.
  - The data transfer between memory and the microcontroller MSSP module takes place via the I^2C bus.
  - The bus requires only two signals (Figure 13–23): Serial Clock (SCL) and Serial Data/Address (SDA).
  - The power supply V_{CC} range varies based on the version of the chip; the V_{CC} range for 24LC024 is 2.5 V – 5.5 V.
- It has three address lines A2–A0 that enable the user to address eight different EEPROM chips in the circuit.
- The WP is a Write Protection pin; it should be tied to V_{CC} to enable the write protection or grounded to disable the function.
- The SDA and SCL are open drain pins, and require pull-up resistors as shown in Figure 13–23.
- The SDA is bidirectional, and data flow in both directions, from the master to a slave and from a slave to the master.
- The master controls the clock and the direction of the data flow.
- Steps in Write Operation
  - Send a Start bit
    - Before sending the Start bit:
      - **Set the bit SEN <Bit 0> in the SSPCON2 register.**
      - **Clear the SSPIF flag in the PIR1.**
      - **Wait in the loop until the SSPIF flag is set.**
  - Send the control byte.
    - Before sending the byte:
      - **Clear the SSPIF flag.**
      - **Load the control byte ($A0_H$) in SSPBUF.**
      - **Wait in the loop until the SSPIF flag is set.**
  - Check if the Acknowledge bit (Bit ACKSTAT <6>) is cleared.

- The EEPROM device must acknowledge by pulling the SDA line low for the ninth clock cycle.
- The control byte $A0_H$ is configured as follows (Figure 13-24):
  - The code for the high-order nibble for this memory is A.
  - The address bits are 0 because they are grounded.
  - The bit <0> is low for the write operation.
- Send the address
  - The 24LC024 EEPROM has only 256 bytes, and requires only one byte address. Memory chips larger than 2 Kb would require a 16-bit or 2-byte address. To send the address, follow the same procedure as in “Send the Control Byte”.
- Send the data byte
  - Once the address has been sent and acknowledged, the data byte can be sent by following the same procedure as before.
- Send the Stop bit
  - The procedure to send the Stop is similar to that of the Start bit except that the bit set in the SSPCON2 register is PEN (Stop condition – Bit <2>).

- Steps in Read Operation
  - Send Start bit, followed by the write command and the address of the memory location to be read. Steps are identical to write operation.
  - Send Restart bit, necessary to send the read command.
  - Send the read command from the master to the slave. Therefore, this is also a write operation.
  - Read the byte and send the Stop bit.

Lesson 8118 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. The difference between serial I/O and parallel I/O is that serial I/O transfers _______ bit or bits at a time and parallel I/O uses the entire ___________ to transfer data.
   (1) one; MPU (3) several, data bus
   (2) four; MPU (4) one, data bus

2. One advantage of serial I/O is that there are less bus lines resulting into smaller chips.
   (1) True (2) False

3. Asynchronous data transmission has separate clocks with similar frequencies.
   (1) True (2) False

4. A walkie-talkie would be classified as which type of conversation mode?
   (1) Simple (3) Half-duplex
   (2) Simplex (4) Full-duplex

5. The bit time for a fax machine that is transmitting data at 1200 bps is ______________?
   (1) 0.83 ms (3) 83.3 ms
   (2) 8.33 ms (4) 833 ms

6. A ________________ occurs when the Start and Stop bits improperly frame a character.
   (1) framing error (3) transmission error
   (2) parity error (4) overrun error

7. If a receiver receives the bits 0100 0001, and the transmission was set for even parity, the receiver would generate a parity error message.
   (1) True (2) False

8. The EIA-232 standard was once known as _____________?
   (1) SPI (3) BPS
   (2) RS-232 (4) LIN

9. In EIA-232 the voltage range of _____ to ______ is defined as Mark.
   (1) -3V, -25V (3) 0V, +5V
   (2) +3V, +25V (4) Mark is not defined

10. In EIA-232 the voltage range of _______ to ______ is defined as Space.
    (1) -3V, -25V
    (2) +3V, +25V
    (3) 0V, +5V
    (4) Space is not defined

11. The Hex code used to transmit the 7-bit character “a” with odd parity is _________?
    (1) 61_H
    (2) $C1_H$
    (3) $E1_H$
    (4) None of the above

12. In reference to Question 11, if the receiver receives the following bits, 0110 0001, then it would generate a parity error message.
    (1) True
    (2) False

13. When a receiver and a transmitter operate on the same clock then they are in _____________________ transmission.
    (1) Asynchronous
    (2) Synchronous
    (3) Simplex
    (4) Full Duplex

14. If the ASCII character “Z” is being transmitted at 28k baud, the bit time is __________?
    (1) 17.8 μs
    (2) 35.7 μs
    (3) 0.357 ms
    (4) More information is needed

15. In the PIC18F microcontroller, PORTA is assigned the address of $F81_H$.
    (1) True
    (2) False

End of Examination

Lesson 8120-Embedded Systems Design
Microcontrollers and Embedded Systems

Assignment Checklist

- ✓ Read Chapter 14 of *Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 Microcontroller Family)*
- ✓ Read Lesson 8120 in this Study Guide
- ✓ Take the Examination for this lesson

<u>Chapter 14 Lecture/Discussion</u>

Features of Embedded Systems

- Specific dedicated purpose
- Space constraints
- Memory constraints
- Power constraints
- Trade-off between power and frequency
- Cost sensitivity
- Protection against software failure
- Real-time constraints

Designing Embedded Systems

- In embedded systems, hardware and software are closely interlinked.
- A decision in hardware components may have considerable impact on software, and vice versa.
- In designing embedded systems, hardware and software designs begin almost simultaneously, unlike the design processes of other computer systems.
- Generally follows the following steps:
  - Establishing system specifications
  - Partitioning tasks between hardware and software
  - Reevaluating partitioning decisions
  - Designing hardware and software components
  - Integrating hardware and software
  - Testing and releasing to production
  - Troubleshooting during production
  - Upgrading and maintenance

System Specifications

- The project should include the following features:
  - – Monitor time and room temperature
  - – Display the time and temperature on an LCD.
    - If the temperature falls below a certain limit, the system should turn on a heater.
    - If the temperature goes above a certain limit, it should turn on a fan.

- Project should specify the following parameters:
  - Temperature range and limits
  - Time format: 24-hour versus 12-hour clock
  - Display: Devices for display
  - Temperature input
  - Power and frequency requirements
  - User's ability to interrupt the system to set the time.
  - Control of I/O peripherals such as the fan and the heater
- Temperature
  - An input to the TTMS
  - Commonly used temperature sensors
    - Thermocouple
    - Resistance temperature device (RTD)
    - Thermistor
    - Integrated silicone-based chips.
- Selection of a sensor depends on temperature range, accuracy, linearity, ruggedness, and price.
  - Indoor room temperature: 40ºF to 90ºF
  - A silicone-based IC
  - Analog signal from IC must be converted into digital signal using an ADC
  - May need a scaling circuit to adjust the output of a sensor to take the advantage of the full-scale range of an A/D converter
- Time
  - Hardware timers or software delay loops
    - Hardware timers more accurate than the software delay loops
  - 12-hour clock versus 24-hour clock
    - Customer preference
    - Affects only software
- Display
  - This is an output.
  - Options:
    - Seven-segment LEDs
    - Matrix LEDs
    - LCDs
- Seven-segment LEDs
  - Least expensive
  - Limited in displaying various characters
  - High power consumption
    - To minimize the power consumption:
      - **Multiplex LED circuit with two I/O ports**
  - Display of four versus six LEDs
  - Hardware decision affects software decisions considerably.

- Matrix LEDs
  - Bright and very attractive displays
  - Need display driver chip or driver would have to be written in software
  - More expensive than other options
  - Consume more power
- LCD display
  - Comprise between price and the attractiveness
  - An 8-bit or 4-bit parallel interface
  - LCDs with a serial interface
- Power and frequency
  - Microcontrollers are TTL logic compatible, and can run on a 5 V power supply. Alternately, the unit can be battery operated.
  - 5 V power supply
    - The power line voltages are different in different countries.
  - The transformers used to step down the power line voltages will have to be selectable.
  - A common practice is to step down the line voltage to unregulated 9 V and use a semiconductor regulator chip to supply 5 V DC.
  - In a battery-operated unit:
    - Power consumption should be minimized
      - Use a lower clock frequency
  - Oscillator circuits
    - An RC circuit and a crystal-driven circuit
      - RC circuit is less expensive, but less reliable.
      - Crystal driven is more reliable and stable.
- Final Specifications
  - Temperature range: 0ºF to 99ºF
  - Temperature sensor: semiconductor chip
  - Clock: 12-hour
  - Display: LCD
  - Power supply: 5 V regulated
  - Frequency: 10 MHz—crystal driven oscillator
  - Clock setting: two switches
- Figure 14–1 shows block diagram of the above system.

Partition of Tasks Between Hardware and Software

- Before partitioning of the tasks, a microcontroller must be selected based on:
  - The number of I/O pins required
  - The memory requirements
  - Cost
  - Availability of troubleshooting tools
  - Technical support from the manufacturer
  - In-house capability

Microcontroller Selection

- Pins requirement
  - Based on specifications, the TTMS project includes an LCD module, a temperature sensor, a fan, and a heater as output peripherals and switches for interrupts.
- All of these devices require one pin each except the LCD.
- LCD interfacing: 8-bit interface, 4-bit interface, and serial LCD
- Size of the memory requirement
  - Assembly language
  - High-level language such as “C”
- Memory: R/W memory and read-only-memory (ROM)
- Questions:
  - Once the product is shipped, does the user need to modify any information to run the unit?
  - Does the MPU need any memory for temporary storage during the execution of the program?
  - Is there a need for the user input?
- The TTMS unit
  - Requires input from the user to set the time
  - Is expected to include subroutines and the use of data registers
- Both types of memory needed
  - ROM to store permanent programs
  - R/W to store the inputs from the user
- ROM choices
  - Masked ROM becomes cost effective
    - **If the production quantities are large (in thousands)**
    - **The product does not anticipate software changes in near future**
  - Product designer needs to depend on a memory manufacturer that may require a six- to twelve-week waiting period.
- OTP - one-time programmable
  - Can be programmed in-house
  - Somewhat cheaper than flash memory
  - Cannot be reprogrammed
  - Flash memory
    - Can be programmed in-house
    - Can be reprogrammed (1000 to 100 k erase/write cycles)
  - EPROM: erasable programmable read-only-memory
    - Almost obsolete because it needs to be taken out of circuit board to be reprogrammed and exposed to ultraviolet light to be erased
  - EEPROM: electrically erasable PROM
    - Can be easily reprogrammed a byte at a time in the field from a remote location
    - More expensive than flash
- TTMS project may need:
  - Less than 1 K of flash memory
  - Fewer than 64 data registers (R/WM)

  - No EEPROM
- Cost
  - Presently, the price of an 8-bit microcontroller in quantity ranges from less than fifty cents to more than six dollars.
  - If a microcontroller meets the pin and memory requirements, the least expensive microcontroller can be selected.
- Other considerations
  - Availability of troubleshooting tools and technical support from the manufacturer of the selected microcontroller
  - In-house capability
  - Availability of secondary sources

Integration of Hardware and Software and Testing

- Hardware
  - Checking with a meter and an oscilloscope:
    - Voltages at various points
    - Grounds
    - System clock frequency
- I/O pins
  - Checking with instructions in a continuous loop
- Software testing generally done with a simulator:
  - Each software module should be tested independently.
  - When modules are combined, the interaction of the independent modules should be tested.
  - After all the errors are corrected, software can be downloaded in hardware target system.
- Hardware and software can be tested together.
  - Power-on reset test.
  - LCD should display the time 00:00 and the room temperature.
  - Various elements of the system can be tested with the LCD display.
  - Interrupt switches can be tested by observing LCD displays.
  - Similarly, the system can be tested for the range of temperatures by cooling and heating the sensor within the limits.

Quality Control, Maintenance, and Upgrading

- Production testing
  - What works in the development lab may not work when the unit is mass produced on the production floor.
  - The quality control tests must be devised.
- Maintenance and upgrading
  - If it is possible to fix or upgrade the software code from remote location, the unit should be available online, and memory that is used in the product should be easily available for remote access.
  - Code in EEPROM can be serially accessed and changed very easily.
  - Code in flash memory can also be changed, but it should be built into the design process.

TTMS Project Design: Hardware

- Hardware
  - Section divided into the following components:
    - Power supply and oscillator (clock)
    - Temperature sensor as an input device
    - Liquid crystal display (LCD) as an output device
    - Switches to interrupt the MCU and set time
    - Fan and heater as controlled output devices

Power Supply and Oscillator (Clock)

- Power supply components required:
  - Wall transformer that converts 120 V into 9 V DC and provides 200 mA current
  - Voltage regulator at 5 V with 1 A current capacity
  - Capacitors
  - LED
  - 9 V battery
- Wall transformer
  - Converts 120 V AC from the wall socket, using a full-wave rectifier, into 9 V unregulated DC with 120-cycle ripple
- Voltage regulator
  - Three-terminal integrated circuit provides a constant output voltage for varying load
- Capacitors
  - Large electrolytic input capacitor provides constant input voltage to the voltage regulator and filters out the 120-cycle ripple generated by 60-cycle AC voltage
  - Small capacitor filters out high-frequency
- LED
  - Turned on whenever the power supply is plugged in, indicating that the power is on
- 9 V battery
  - Can be connected in place of the power supply, and the voltage regulator is protected by the diode if a battery were to be connected with the voltage reversed
- Oscillator
  - PIC18F452 has eight and 4520 has ten different oscillator modes.
  - Mode can be programmed by setting appropriate logic levels of the bits FOSC3:FOSC0 in Configuration Register 1 H.
  - Modes can be grouped in four categories:
    - Using a crystal or a ceramic resonator
    - RC oscillator circuit
    - Internal oscillator
    - External clock
- Figure 14–3a shows an oscillator circuit with a 10-MHz crystal connected to pins OSC1 and OSC2 with two 22-pico-farad capacitors.
- Figure 14–3b shows an RC circuit connected to generate clock frequency, which is more cost effective but provides less stable frequency.

- Clock frequency directly affects power consumption, and many special features are available in this microcontroller family to reduce the power consumption.

Temperature Sensor as an Input Device

- The TTMS temperature range is 0ºF to 99ºF.
- A sensor with the voltage output proportional to Fahrenheit is an ideal choice.
- LM34 temperature sensor
  - A three-terminal device and its output changes linearly 10 mV per degree Fahrenheit starting with 0 volts at zero-degree temperature
- The output voltage range of the LM34 will be 0 to 0.99 V for the temperature range of the system.
- Channel AN0 (pin RA0) is used as shown in Figure 14–4.
- The reference voltages of the A/D converter V_{REF+} and V_{REF-} can be the same as power supply voltage +5 V and ground.
- Input voltage from the temperature sensor to the A/D converter module ranges from 0 to 0.99 V.
- Input voltage range must be scaled to match the entire range of the reference voltages.
- This can be accomplished by using an operational amplifier with the gain of five as shown in Figure 14–4.

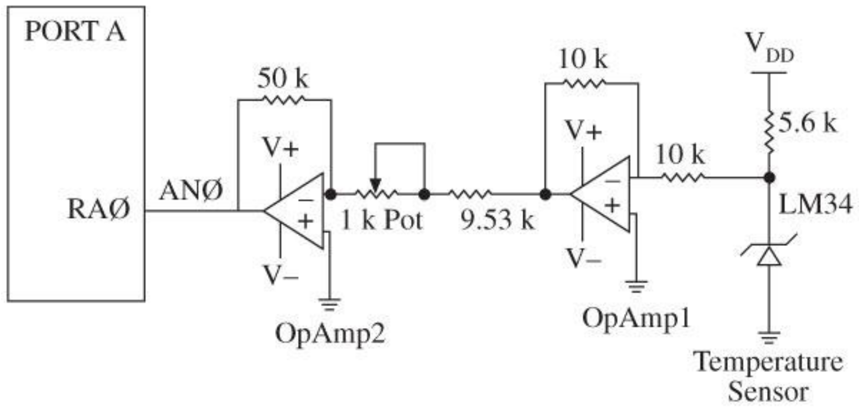


Figure 14-4 Connecting a Temperature Sensor to A/D Converter Input through a **<u>Scaling</u>** Circuit
****Note on Page 471 the caption for Figure 14-4 is incorrect. This is the correct wording.

- Figure 14–4 has two precision operational amplifiers.
  - First (Op Amp1) functions as a unity gain amplifier and changes the polarity of input voltage range from positive to negative
  - Second (Op Amp2) amplifies the input signal with the necessary gain (approximately five) and changes the polarity of the signal back to positive voltage range
    - Output of scaling circuit ranges from 0 V to +4.99 V.

Liquid Crystal Display (LCD) as an Output

- LCD display
  - Hitachi LM032L (2-line x 20 Character)
  - Display module can operate in three modes:

        - 8-bit interface
        - 4-bit interface using high-order four data lines <7-4>
        - 4-bit interface using low-order four data lines <3-0>
    - 8-bit data interface requires less programming, but uses 11 (8 data lines and 3 control signals) I/O pins.
    - 4-bit data interface uses only 7 I/O pins, but requires more programming.
    - Figure 14-5 shows 8-bit interface.
        - Circuit uses eight pins of PORTD for LCD data lines, and three pins from PORTA for control signals.

Reset and Interrupts

- When user plugs unit in wall socket, power-on reset will establish initial conditions for the TTMS.
- The MCLR can be tied to power supply voltage V_{DD} as shown in Figure 14–6.
    - Figure 14–6 shows two push-button switches—one to set hours and the other to set minutes—connected to pins RB1 and RB2.
    - Pins can accept interrupts from outside sources.
        - The ISR associated with each switch should increment the digits for hours and minutes.

Fan and Heater as Controlled Output Devices

- Fan and heater are controlled by two output bits
    - RB7 and RB6
- Fan and heater are high-voltage devices
    - Should not be connected directly to low-voltage digital systems; ground connections must be isolated
- Figure 14–7 shows the fan and the heater connected to bits RB7 and RB6, respectively, through optically isolated triac drivers.
    - Logic 0 at bit RB7 (and at RB6) turns on the LED of the opto-isolator and the current flows in the LED, a minimum of 10 mA, turns on the triac driver, which, in turn, switches on the power triac 2N6071.

Complete TTMS System

- Figure 14–8 shows the complete schematic of TTMS system using PIC18F452/4520 microcontroller.
- When user plugs the system in to 120V AC, the power supply turns on the LED, the power-on reset (POR) establishes initial conditions of the system, and the LCD module displays time 00:00 AM and temperature 00.0ºF.
- If user pushes hour switch, system is interrupted, and Interrupt Service Routine (ISR) of that switch increments hour digit at each contact of switch until 12 PM and begins again at 1 PM.
- User can set minutes by pressing minute switch, which also interrupts the system, and its ISR increments the minutes until 59 and restarts from 00.

TTMS Project Design – Software

- Software segment of the project involves the following tasks:
    - Reading the room temperature
    - Checking the reading against the set values of high and low to turn on/off the fan and the heater
    - Displaying the temperature on line 2 of the LCD
    - Setting a timer for a one-second delay
    - Updating the clock every second and display the time on line 1 of the LCD

Project Flowchart and Interrupts

- Project software can be divided into two independent segments:
    - Main segment consists of subroutines that read temperature from the temperature sensor, convert analog readings into digital equivalents, and display the temperature at the LCD.
    - Clock Interrupt segment consists of interrupt service routines (ISRs) that update the clock when an interrupt is generated.

Flowcharts and Software Modules

- MAIN Segment—consists of four modules as shown in Figure 14–9:
    - Initialization Module
        - Initialize various I/O ports and modules such as the A/D converter
    - Temperature Module
        - Start an A/D conversion, read temperature at the end of the conversion, convert the reading in a proper format, and turn on/off fan and heater
    - Display Module
        - Display the time and temperature at the LCD
    - Time Delay Module
        - Provide delays to stabilize LCD display
- Initialization Module
    - Calls SETUP subroutine to initialize various I/O pins either as inputs or outputs as well as the A/D converter module in the 18F4520 microcontroller.
    - SETUP subroutine initializes:
        - Output pins
            - All eight pins of PORTD used as data lines to the LCD
            - LCD control signals RA1 and RA2 of PORTA
            - RB7 and RB6 pins of PORTB that control the fan and the heater.
        - Input pins
            - RA0/AN0 for analog signal from temperature sensor and RB1 and RB2 (as external interrupts from switches) to set the time

****Note on Page 477 this section needs the word “for” removed****

- A/D Module
    - Selects channel AN0, sets it up for analog input, and turns on the A/D module
    - Selects V_{DD} and V_{SS} as reference voltages, right justified display, and appropriate conversion frequency F_{OC} and bit conversion time T_{AD}

- Temperature Module—consists of multiple subroutines:
  - A-TO-D subroutine to start a conversion and read the temperature at the end of the conversion
  - MULTIPLY10 and DIVIDE100 subroutines to convert the temperature reading into its equivalent integer and fraction
  - BINBCD subroutine to convert the reading into BCD numbers and their ASCII equivalent
    - These readings are stored in data registers for the LCD module
  - Also checks low and high temperature limits
    - If the temperature reading is higher than 80°F, the module calls the FANON subroutine to turn on the fan.
    - If temperature reading is lower than 60°F, the module calls the HEATERON subroutine to turn on the heater.
- Display Module
  - Initializes the LCD, sends the necessary commands, and displays ASCII characters stored in data registers at TIME0, TIME, and DEGREE
  - When the TTMS system is turned on, it should display time as 00:00.
  - Five ASCII characters for 00:00 are stored at location TIME0 that can be displayed by this module.
- Temperature module saves temperature readings at data registers defined as DEGREE, and the ISR of the timer saves the time at data registers defined as TIME.
- Delay Module
  - Subroutine provides multiples of a 20-millisecond delay, and multiple is determined by the count in the data register MULTIPLE.
  - This delay is used between outputting consecutive displays and debouncing the interrupt keys.
- Clock Interrupt Segment
  - Consists of three interrupts
    - One high-priority interrupt from Timer0, which is used for the clock
    - Two low-priority interrupts from the two external push-button switches connected to pins RB1 and RB2 to set time.
- Timer0 Interrupt
  - Set up to generate an interrupt every second, the interrupt service routine CLK_ISR updates the time, and BCDASCII subroutine converts hours, minutes, and seconds into equivalent ASCII characters which are saved in data registers labeled as TIME0 to TIME+7
    - (Refer to Chapter 11, Section 11.5)
- Push-Button Interrupts
  - Two push-button keys are connected to pins RB1 and RB2 to set minutes and hours respectively
  - When a key is pushed, an interrupt signal is generated, the MPU jumps to the low-priority interrupt memory location 00018H and then to KEY_ISR.
    - The flowchart of the KEY_ISR interrupt is shown in Figure 14–10.

- Push-Button Interrupts (cont’d)
  - The interrupt KEY_ISR first waits in a loop for 20-millisecond to debounce the key and then checks RB1.
    - If it is low, the service routine updates the minutes and returns to the place where it was interrupted.
    - If RB1 is high, the KEY_ISR checks RB2 next, and if it is low, the KEY_ISR updates hours and returns to the place where the MPU was interrupted.
    - If none of the keys is low, the KEY_ISR assumes it is a false alarm and returns to the place of the interrupt.

Subroutines – FANON and FANOFF

- Figure 14–9 shows the flowchart of the TTMS software.
  - Flowchart of the subroutine A-TO-D shows one decision point checking whether the temperature is higher than 80ºF, followed by the second decision point checking whether temperature is less than or equal to 80ºF. This is somewhat unusual.
- If we do not ask the second question, the fan will be turned off, even if it has just been turned on by the previous subroutine because temperature was higher than 80ºF.
- The same logic also applied to the process of running the heater.

****Note on Page 481 the Instructions check listed has a couple of errors. In the section FANON: on line 2 it should be   BCF   PORTB, RB7

In section FANOFF: line 1 should be   BTFSC   FANFLAG, 0
Also in FANOFF: line 2 should be   BSF   PORTB, RB7

Lesson 8120 Examination
Microcontrollers and Embedded Systems

Answer the following questions based on what has been presented or discussed in the textbook and study guide. To submit the exam please use either Word or Excel to create a two column answer sheet. The left column is the question number and the right column would be your answer. On the answer sheet please include your name, student number, examination/lesson number and an email address so we can return your results. If you have questions on creating an answer sheet or attaching the answer sheet please contact the Instruction Department at 1-800-243-6446 or faculty@cie-wc.edu.

1. Which one of the following is not a feature of embedded systems?
   (1) Not accessible to the user
   (2) Not operated in real-time constraints
   (3) Designed for specific dedicated tasks
   (4) Constrained by space, memory size, power, and cost

2. The constraint of space with the design of embedded systems stems from _________.
   (1) the area the system is stored in
   (2) limits with the printed circuit board
   (3) the placement of the power supply
   (4) the physical size limitations

3. The real-time embedded systems are divided into which two categories?
   (1) time-critical & time-stamp
   (2) time-sensitive & time-critical
   (3) strict time limit & time sensitive
   (4) real & time

4. One of the design steps that are required in the development of embedded systems that is different from designing other electronic systems is ________________________?
   (1) the partitioning of tasks between hardware and software at the beginning stage of the design cycle
   (2) the integration of accessibility for the end user
   (3) maximizing the power requirements
   (4) designing the system for memory sizes approaching 1 Giga bytes

The following information will cover Questions 5 to 10.

The operating range of the temperature sensor LM34 is from -50 degrees Fahrenheit to +300 degrees Fahrenheit. The output voltage range of the sensor is -500 mV to +3000 mV, which is scaled to the range of 0 to 5 V as an input to the PIC18 A/D converter module with the 10-bit resolution. [If necessary, review Figure 14-4]

5. From the information provided, what would be the sensor output voltage?
   (1) -500 mV
   (2) +2500 mV
   (3) +3000 mV
   (4) +3500 mV

6. What is the gain of the A/D converter when the temperature is 0 °F?
   (1) 1.428
   (2) 1.667
   (3) 2
   (4) 10

7. What is the input voltage to the A/D converter when the temperature is 0 °F?
   (1) 714 mV   (3) 1 V
   (2) 833 mV   (4) 4.28 V

8. What is the temperature reading when the converted digital reading is 32_H?
   (1) -50 °F   (3) +125 °F
   (2) -32.9 °F   (4) +300 °F

9. What is the temperature reading when the converted digital reading is 200_H?
   (1) -50 °F   (3) +125 °F
   (2) -32.9 °F   (4) +300 °F

10. When the temperature is +100 °F what is the digital Hex reading?
   (1) 92_H (3) $1B6_H$
   (2) 124_H (4) $2DB_H$

11. The PIC184520 has _______ different oscillator modes.
   (1) 8   (3) 12
   (2) 10   (4) 16

12. Another consideration for the design of the TTMS (Time and Temperature Monitoring System) unit is to change the transformer ratio and the shape of the wall socket if marked outside of the United States.
   (1) True   (2) False

13. The TTMS operates the fan and heater and these two devices are controlled by two output bits. The bit ______ controls the fan and the bit _______ controls the heater.
   (1) RB7, RB1
   (2) RB1, RB7
   (3) RB6, RB7
   (4) RB7, RB6

14. In Figure 14-2, the function of the diode connected to a battery source is __________?
   (1) to filter the AC from the bridge circuit
   (2) to protect the fuse
   (3) to protect the circuitry in the event the battery is connected with the reversed polarity as the diode will conduct
   (4) to protect the output voltages (V_{DD} and V_{SS})

15. A watchdog timer is used to prevent hardware failures.
   (1) True   (2) False

End of Examination

CLEVELAND INSTITUTE OF ELECTRONICS

1934