MICROPROCESSORS A (17.383)

Fall 2010

Lecture Outline

Class # 09

November 09, 2010

Dohn Bowden

Today's Lecture

- Syllabus review
- Microcontroller Hardware and/or Interface
  - LCD interface
- Programming/Software
  - LCD interface
  - Assembly Language Basics
- Lab
  - Work on Lab #4
- Homework

Course Admin

Administrative

- Admin for tonight ...
  - Course Project
    - Submit your Project topic NLT Monday November 15th (next week)
    - Any questions?
  - Syllabus Highlights
    - Lab report for Lab # 3 is due tonight
    - For planning purposes ...
      - Exam #2 is November 30th ... 3 weeks from today

Syllabus Review

Week	Date	Topics	Lab	Lab Report Due
1	09/07/10	Intro, Course & Lab Overview, Microcontroller Basics	1	
2	09/14/10	PIC16F684 Overview and General Input/Output	1 con't	
3	09/21/10	Switches	2	
4	09/28/10	Seven Segment LEDs	2 con't	1
5	10/05/10	**Examination 1**		
X	10/12/10	**No Class – Monday Schedule**		
6	10/19/10	Analog to Digital Conversion	3	2
7	10/26/10	Analog to Digital Conversion con't	3 con't	
8	11/02/10	Lab Work (Finish Lab #3 and start Lab #4)	3, 4	
9	11/09/10	LCD Interface and Assembly Language	4 con't	3
10	11/16/10	Comparators, Timers, Pulse Width Modulation (PWM)	5	
11	11/23/10	Mixed C & Assembly Programming/Course Project	Project	4
12	11/30/10	**Examination 2**		
13	12/07/10	Course Project	Project	5
14	12/14/10	Final Exam/Course Project Brief and Demonstration	Demo	

Microcontroller Hardware and / or Interfaces

LCD Interface ...

PIC16F684/LCD Display Interface

- Many microcontroller applications require display of ...
    - Messages
    - Data Values
- Typical types of displays:
    - LEDs (previously discussed)
    - 7-segment LED Displays (previously discussed)
    - LCD Displays
    - Video Displays (requires complex interfaces & are costly)
    - Touch Screens

PIC16F684/LCD Display Interface

- LCD Display Advantages ...
    - Low cost
    - Low power consumption ... therefore ideal for:
        - Low power
        - Battery operated portable applications
    - Alphanumeric
    - Some LCDs are 40 characters wide
    - LCDs can be single or multiple rows
    - Some can display graphic images

Types of LCDs

- Serial
- Parallel

Serial LCDs

- Connected to the PIC using one data line
    - Only one wire from the microcontroller
        - Saves I/O pins
- Data is transferred to the LCD via the standard RS232 asynchronous data communication protocols
- Cost more than parallel
    - Cost of serial interface hardware
- Can be challenging to program (timing and the RS232 protocol)

Parallel LCDs

- Programming requires an understand of the internal operation of the LCD module, including timing
- Many parallel LCD modules are HD44780 types
- Line lengths come in ...
    - 8, 16, 20, 24, 32, and 40 characters
- Depending on the model ...
    - 1, 2, or 4 display rows can be selected

Parallel LCDs

- The display has either a 14 or 16 pin connector for interfacing to the microcontroller
    - 14 pin for non-back lighted displays
    - 16 pin for back lighted displays

Parallel LCDs Pin Assignments

Pin Number	Symbol
1	V_{ss}
2	V_{cc}
3	V_{ee}
4	RS
5	R/W
6	E
7	DB0
8	DB1
9	DB2
10	DB3
11	DB4
12	DB5
13	DB6
14	DB7

Parallel LCDs Pin Assignments

Signal name	No. of Lines	Input/Output	Connected to	Function
DB4 ~ DB7	4	Input/Output	MPU	4 lines of high order data bus. Bi-directional transfer of data between MPU and module is done through these lines. Also DB_7 can be used as a busy flag. These lines are used as data in 4 bit operation.
DB0 ~ DB3	4	Input/Output	MPU	4 lines of low order data bus. Bi-directional transfer of data between MPU and module is done through these lines. In 4 bit operation, these are not used and should be grounded.
E	1	Input	MPU	Enable - Operation start signal for data read/write.
R/W	1	Input	MPU	Signal to select Read or Write "0": Write "1": Read
RS	1	Input	MPU	Register Select "0": Instruction register (Write) : Busy flag; Address counter (Read) "1": Data register (Write, Read)
Vee	1		Power Supply	Terminal for LCD drive power source.
Vcc	1		Power Supply	+5V
Vss	1		Power Supply	0V (GND)

Pins (1 – 4)

- Pin 1 – V_{SS} is 0 volts or ground
- Pin 2 – V_{DD} is positive voltage supply
- Pin 3 – V_{EE} is contrast control pin.
  - Tie to a variable voltage source
- Pin 4 – Register Select (RS)
  - When low ... data transferred to the display
  - When high ... character data can be transferred to/from the display

Pin (5)

- Pin 5 – Read/Write (R/W)
  - Low to write commands or data to the LCD
  - When High, character data or status information can be transferred
- **NOTE:**
  - Pin 5 is usually connected to ground ... LCD in write mode only

Pins (6 – 14)

- Pin 6 – Enable (E) pin
  - Used to initiate the transfer of commands or data between the LCD and the microcontroller
  - When writing to the display ... data is transferred only on the High to Low transition of this pin
- Pins 7 to 14 – Data bus lines (D0 to D7)
  - Data can be transferred either in 4 or 8 bit interface
  - In 4 bit interface, only D4 to D7 are used

Pins (15 -16)

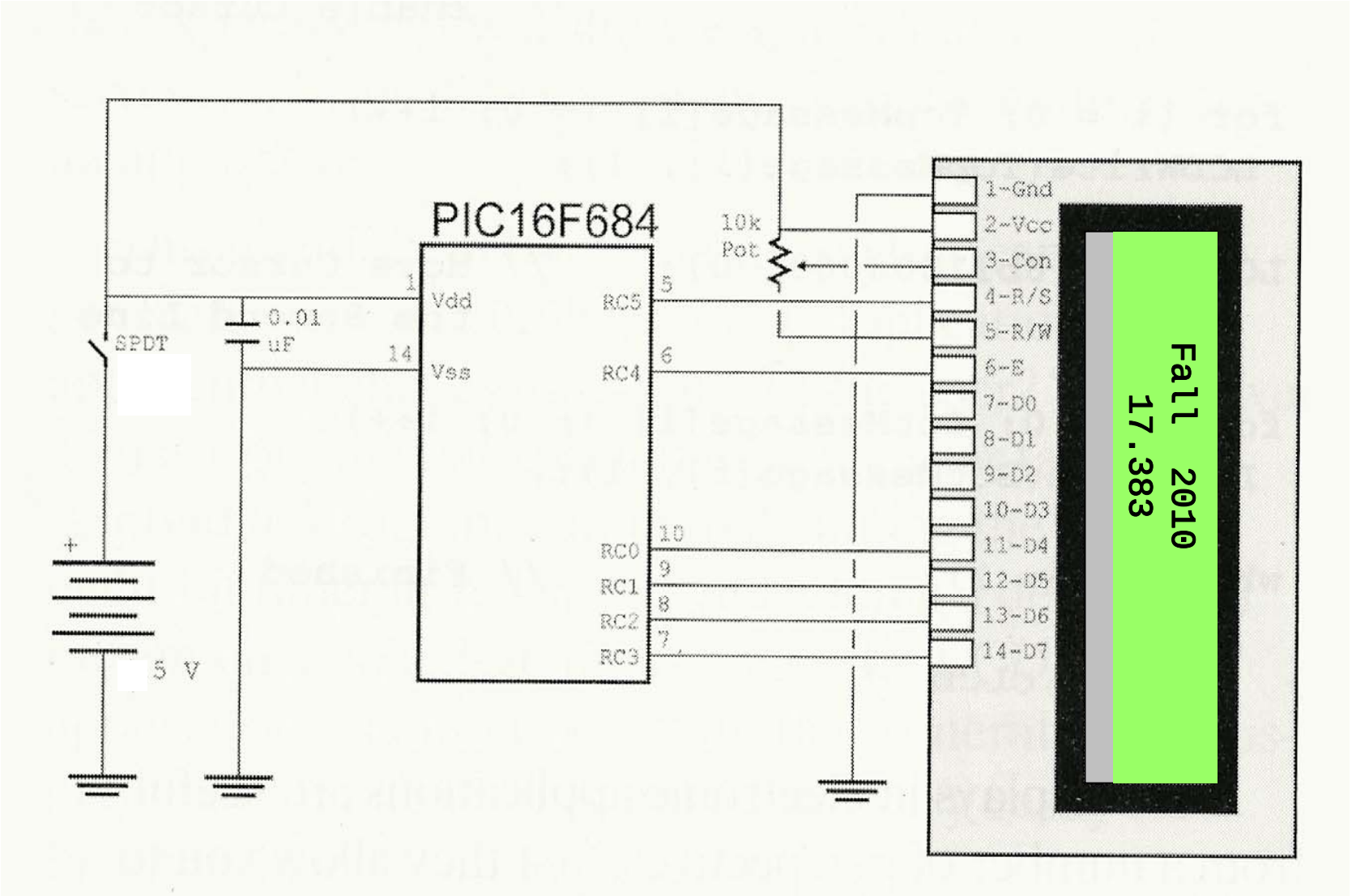
- Pin 15 – LED+ (Backlighting), if available
  - V_{CC}
- Pin 16 – LED- (Backlighting), if available
  - Gnd

Connecting the LCD to the PIC16F684

Circuit connections (see below and diagram on next slide) ...

Circuit	PIC	LCD
Gnd		Pin 1
VCC		Pin 2
10k pot wiper		Pin 3
Gnd		Pin 5
	RC5 (5)	Pin 4
	RC4 (6)	Pin 6
	RC0 (10)	Pin 11
	RC1 (9)	Pin 12
	RC2 (8)	Pin 13
	RC3 (7)	Pin 14

Connecting the LCD to the PIC16F684



Programming – LCD/PIC Interface

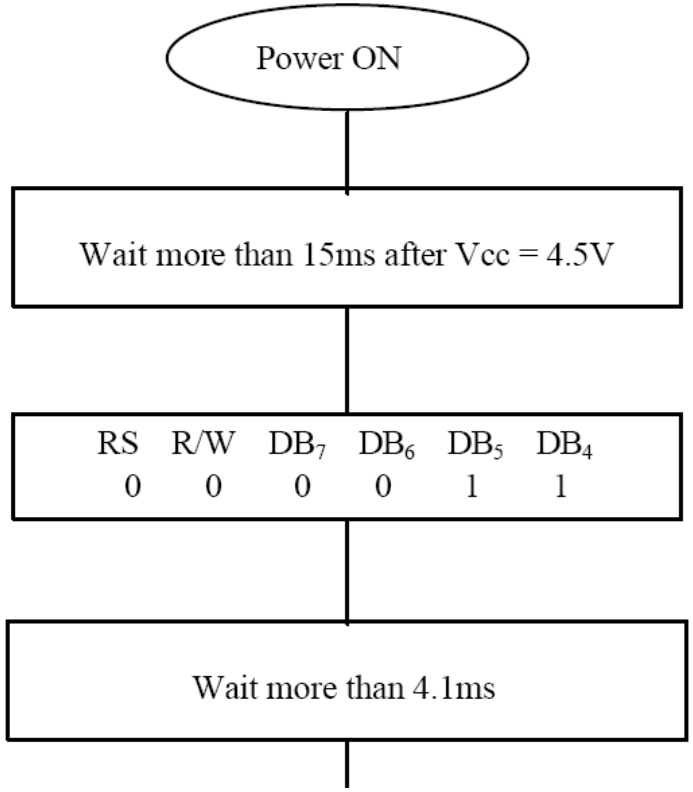
- Each character display can be operated in either 4 or 8 bit mode
- When operating in 4 bit mode, data is transferred in two 4 bit operations using data bits DB4 - DB7 ...
  - DB0 - DB3 are not used and should be tied low.
  - When using 4 bit mode, data is transferred twice before the instruction cycle is complete
    - First the high order nibble is transferred then the low order nibble
- When operating in 8 bit mode, data is transferred using the full 8 bit bus DB0 - DB7

Initializing the LCD Module

- **Software Initialization**
  - Although software initialization is not mandatory ...
    - Software initialization *is recommended*
- The next three slides describe the software initialization
  - NOTE: **BF** is **B**usy **F**lag

Software Initialization (Slide 1 of 3)

4 - Bit Initialization:



No data should be transferred to or from the display during this time.

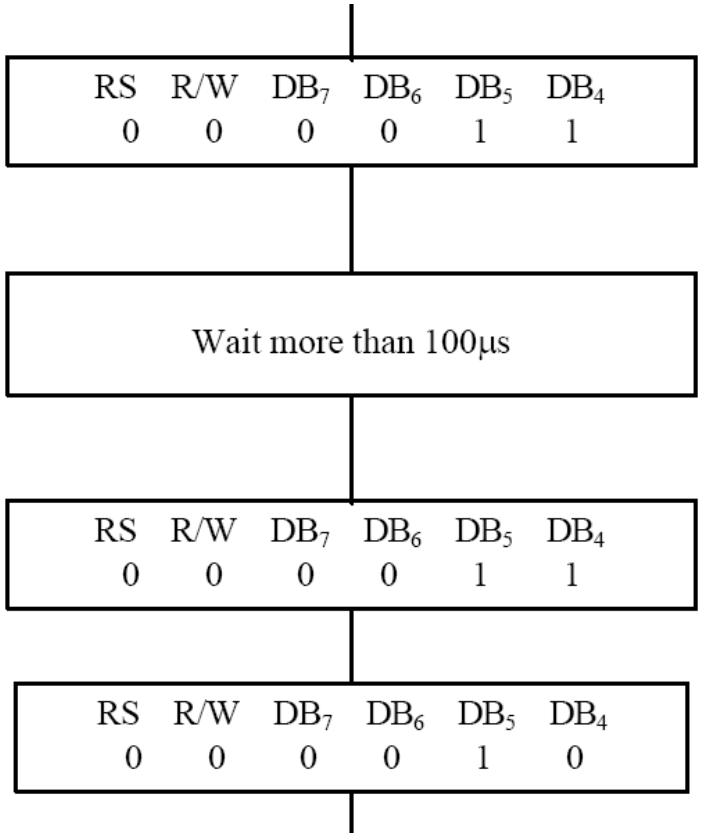
Function Set Command: (8-Bit interface)

No data should be transferred to or from the display during this time.

Continued … next slide

Software Initialization (Slide 2 of 3)

Continued from previous slide



Function Set Command: (8-Bit interface)

No data should be transferred to or from the display during this time.

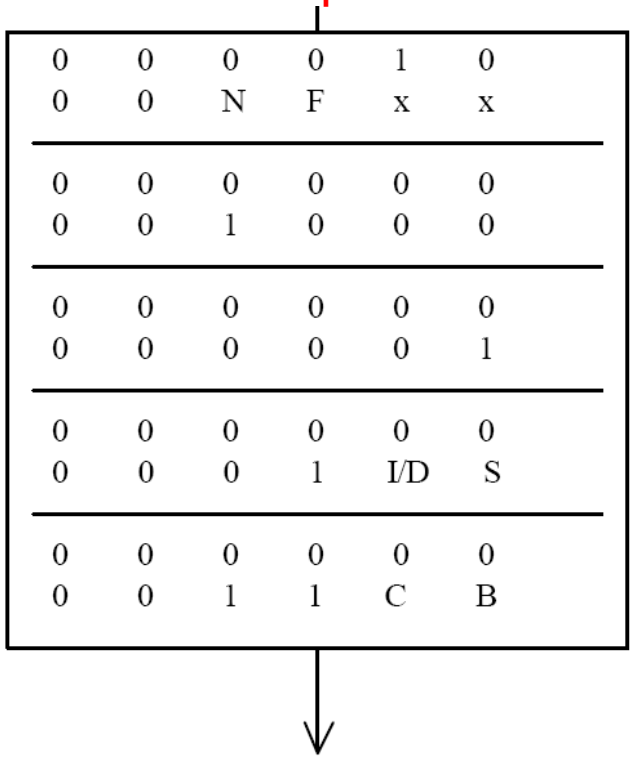
Function Set Command: (8-Bit interface)
After this command is written, BF can be checked.

Function Set: Sets interface to 4 -bit

Continued … next slide

Software Initialization (Slide 3 of 3)

Continued from previous slide



Function Set (Interface = 4 -bit, Set N and F for number of lines and character font)

Display OFF

Clear Display

Entry Mode Set:

Display ON (Set C and B for cursor/Blink options.)

Note: BF should be checked before each of the instructions starting with Display OFF.

Reference Material

- On the Course Web Page ...
  - OPTREX CORPORATION ...

    DOT MATRIX CHARACTER LCD MODULE USER'S MANUAL
- An excellent reference source

Sample PIC16F684/LCD Code

- The course Web Page contains LCD_4_Bit.c file
  - A good starter for basic interface
  - Uses the pin-out previously identified
  - Can format the output using C sprintf formatting

Sample PIC16F684/LCD Code

- LCD_4_Bit.c code requires the stdio.h header
  - Therefore ... use the INCLUDE command
- This file inclusion is required for the sprintf command
  - Stdio.h uses large amount of the 684 memory

Available Hardware

- We have 2x16 parallel LCD modules available (JHD 162A)
    - **They must be turned in at the end of the semester**

Programming / Software

Programming

- Commands/instructions that we will encounter tonight
  - C commands - ***No new C commands tonight***
  - Assembly Language – ***Fundamentals***
  - PIC16F684 control – ***LCD interface***

LCD_4_Bit.c

Review of the software

LCD_4_Bit.c

- Recommend ...

  1. Review the ... OPTREX CORPORATION

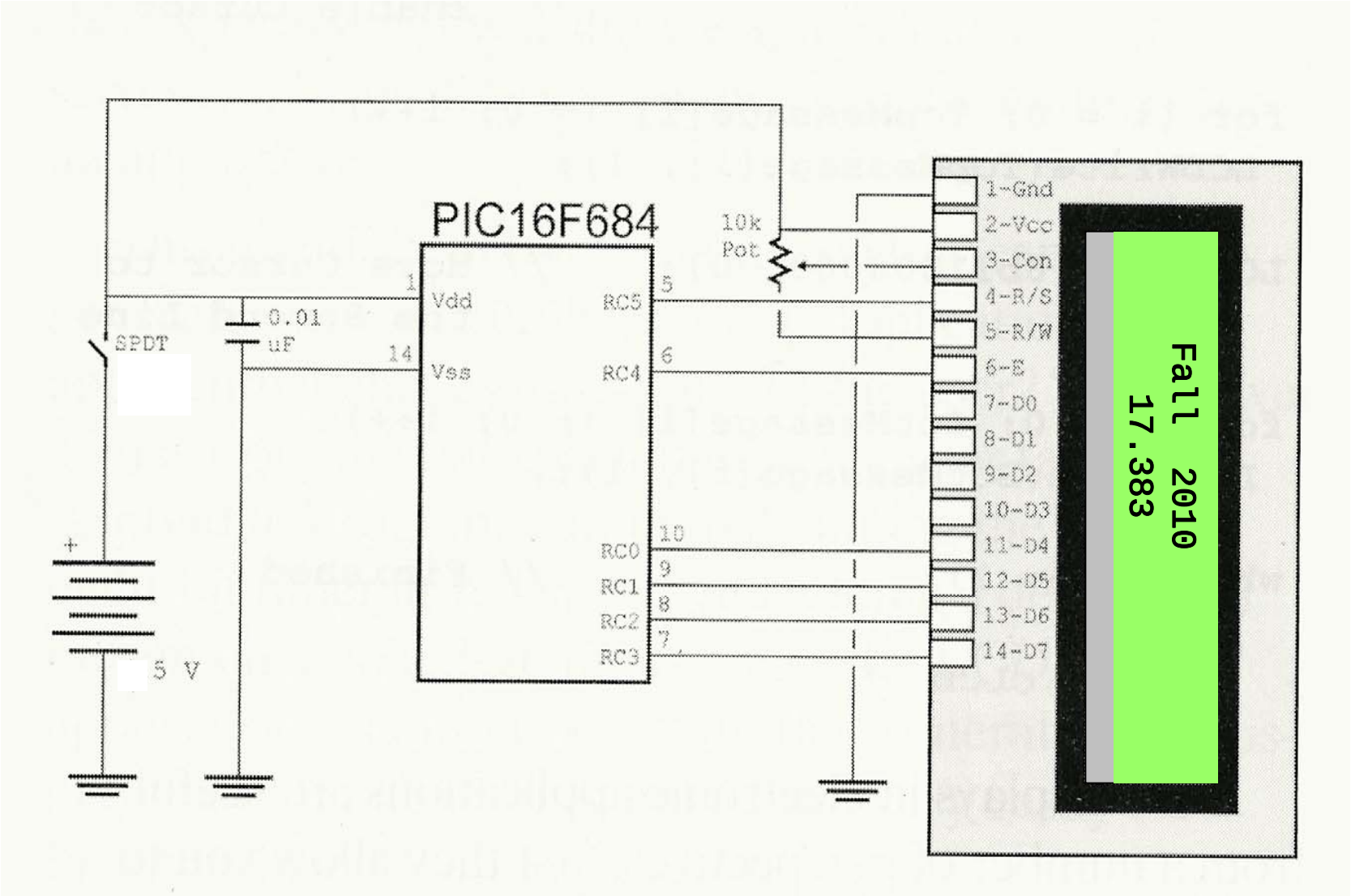
     “DOT MATRIX CHARACTER LCD MODULE USER’S MANUAL“

  2. Exam the schematic (next slide)

  3. Study the LCD_4_Bit.c code provided on the course web page

- An excellent starting point for applications requiring the use of a LCD display

Connecting the LCD to the PIC16F684



Assembly Language ...

Introduction to Assembly Language

- Assembly Language
- Resources
  - MPASM Users Guide
  - PIC16F684 Data Sheet
  - Easy Microcontrol'n by David Benson

Assembly Language

- Will approach assembly language as we did with C
  - We will learn it as we encounter instructions and formats
- Tonight ...
  - I will throw a lot at you ...
  - It is an intro, digest the information ...
    - We can go over it again next week if you have any question

Introduction to PIC MCU Assembly Language Programming

- Assembler ... the program that converts an assembly language program into a .hex file
- The assembler is analogous to the compiler that converts C into a .hex file
- The assembler we will use is the MPASM
  - Which is built into the MPLAB IDE
  - HI-TECH ANSI C compiler ... what we used for C programs ...
    - Was not built into MPLAB IDE

Introduction to PIC MCU Assembly Language Programming

- Assembly language is no more difficult than C
  - It is ‘just’ different
    - The differences is in the statements used to write the code
  - Assembly code tends to be longer ...
    - Assembly language does smaller tasks
    - C commands does many steps at one time
  - Both accomplish the same result ... only assembly takes smaller steps to get to the same end point

Introduction to PIC MCU Assembly Language Programming

- Poorly written code will be no more efficient in assembly than written in a higher level language
- The program needs to be easily read
- The program needs to be efficient
- Use comments
  - Makes debugging easier

MPASM Assembler

- The MPASM assembler is a full-featured, universal macro assembler for all PICmicro MCUs
- The MPASM assembler features include:
  - Integration into MPLAB IDE project
  - User defined macros to streamline assembly code
  - Conditional assembly for multi-purpose source files
  - Directives that allow complete control over the assembly process

Source Code For the Assembler

- Code can be created with MPLAB Text Editor
- Assembly Language Features are as follows ...

Assembly Instruction Fundamentals

- Instructions may be unfamiliar to you ... the following fundamentals will help understand most instructions ...
  - The letter **f** in an instruction ... refers to a **f**ile register
  - A **w** means the **w**orking register, also known as the W register
  - A **b** represents a **b**it
  - A **l** indicates **l**iteral (usually the number that follows)

A Sample Assembly Instruction

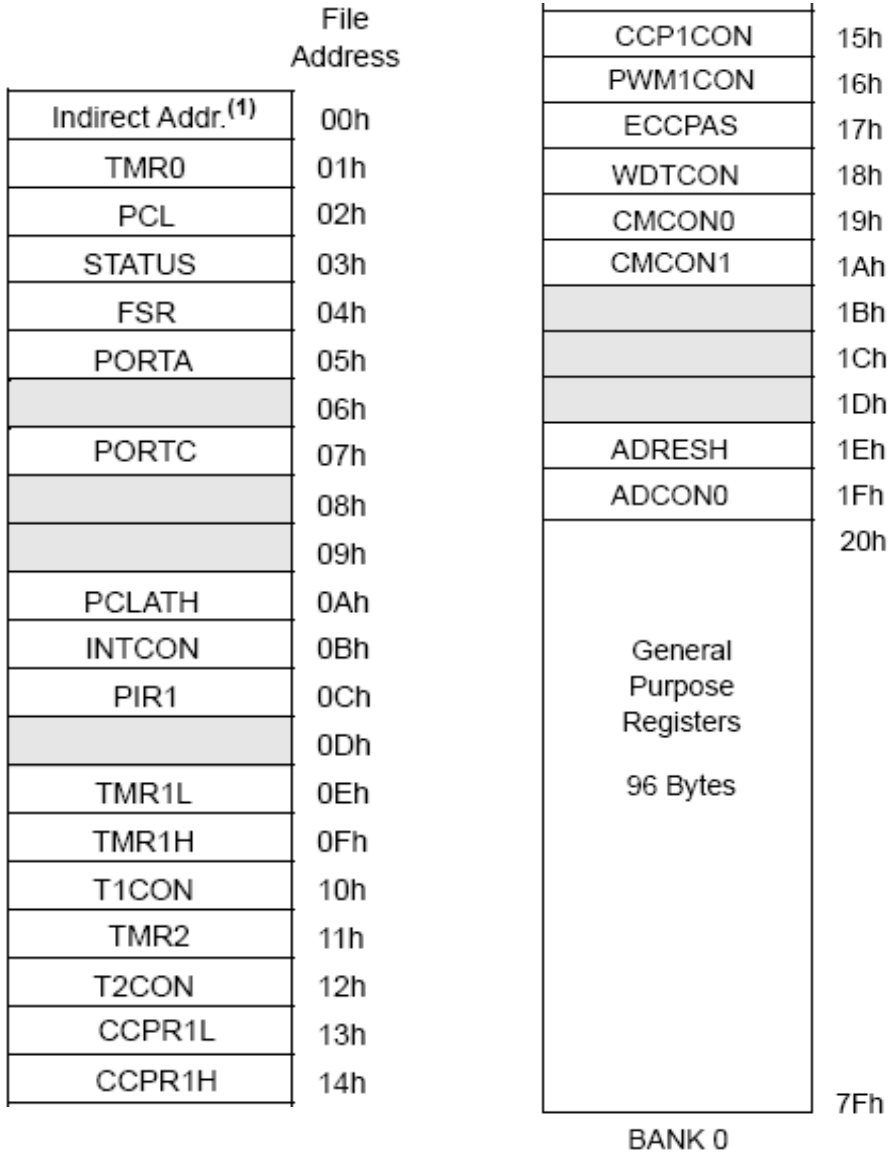
```
BSF    PORTA, 0      ;sets bit 0 of file register - PORTA
```

- The above is just a sample of what a typical instruction would look like
  - BSF ... is what you are doing, the instruction
  - PORTA is the file register
  - 0 is bit zero of that register
  - The ; indicates that a comment follows
- We will get into more details of how to construct an actual program

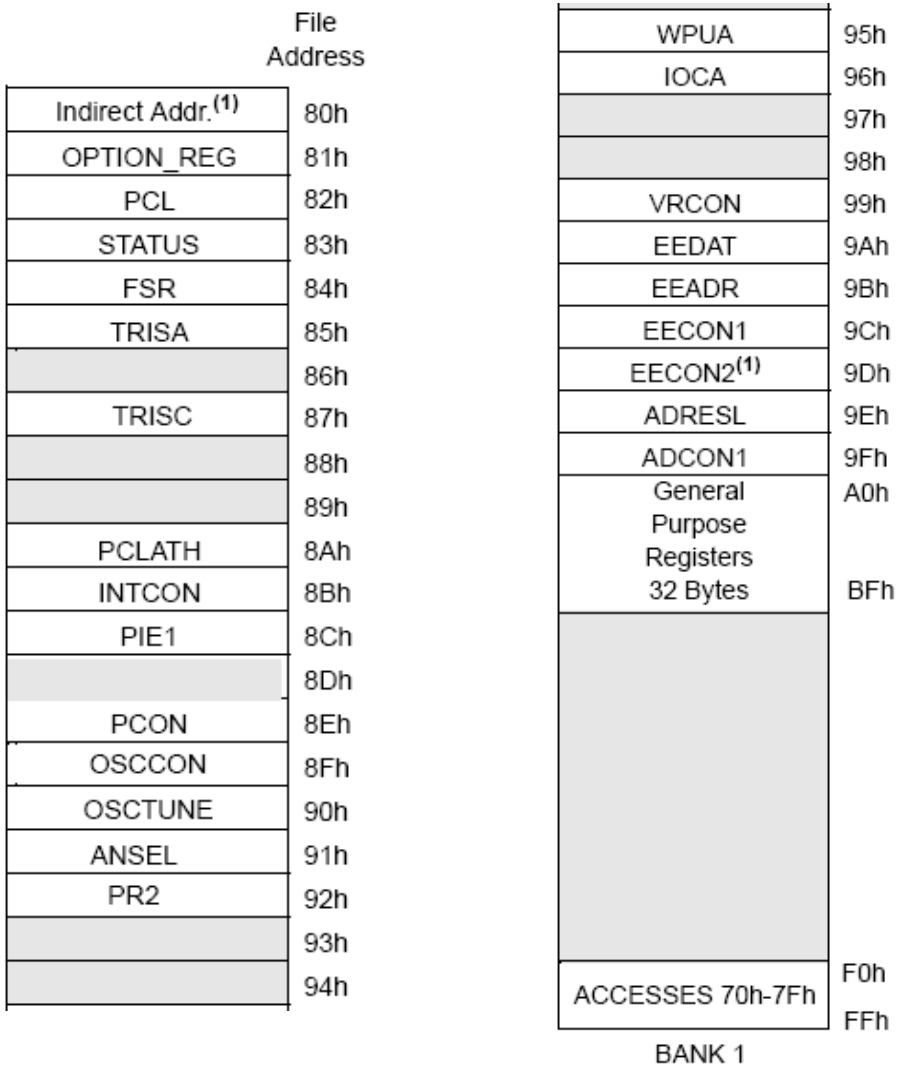
Need to Understand File Registers

- Assembly requires knowledge of the File Register locations
- Need to be aware of which Bank the individual registers are located in
- The next two slides contains the Registers
    - Bank 0
    - Bank 1

Map of the File Registers for the PIC16F684
Bank 0



Map of the File Registers for the PIC16F684
Bank 1



Directives

- Directives ...
  - Commands to the assembler that are used to control the assembly of the program
  - They appear in the source code but are not usually translated directly into opcodes
  - They are used to control the assembler ...
    - Its input
    - Output ...and ...
    - Data allocation

Directives By Type

- There are six basic types of directives provided by the assembler ...
  - Control Directives (#include, equ, org)
  - Conditional Assembly Directives (if, else)
  - Data Directives (__config)
  - Listing Directives (title, list)
  - Macro Directives
  - Object File Directives

title Directive

- `title` – SPECIFY PROGRAM TITLE
- **Syntax**
  - title "*title_text*" NOTE ... there is a space before `title`
- **Description and Usage**
  - *title_text* is a printable ASCII string enclosed in double quotes
  - It must be 60 characters or less
  - Establishes the text to be used in the top line of each page in the listing file
- **Example**

```
title "operational code, rev 5.0"
```

`list` – LISTING OPTIONS

- **Syntax**
  - `list [`*`list_option`*`, ..., `*`list_option`*`]`
- **Description**
  - Occurs on a line by itself
  - Has the effect of turning listing output on ... if it had been previously turned off
  - Otherwise, one of a list of options can be supplied to control the assembly process or format the listing file

`list` – LISTING OPTIONS

- Usage
  - Options that may be used with the list directive are specified in the following table (next slide)

list – LISTING OPTIONS

Option	Default	Description
b=nnn	8	Set tab spaces.
c=nnn	132	Set column width.
f=*format*	INHX8M	Set the hex file output. *format* can be INHX32, INHX8M or INHX8S. **Note:** Hex file format is set in MPLAB® IDE (Build Options dialog).
free	FIXED	Use free-format parser. Provided for backward compatibility.
fixed	FIXED	Use fixed-format parser.
mm={ON|OFF}	On	Print memory map in list file.
n=nnn	60	Set lines per page.
p=*type*	None	Set processor type; for example, PIC16F54. See also `processor`. **Note:** Processor type is set in MPLAB IDE (*Configure>Device*).
pe=*type*	None	Set processor type and enable extended instruction set; for example, `LIST pe=PIC18F4620` Only valid with processors which support the extended instruction set and the generic processor PIC18XXX. Is overridden by command-line option `/y-` (disable extended instruction set). **Note:** Processor type is set in MPLAB IDE (*Configure>Device*).
r=*radix*	hex	Set radix: hex, dec, oct. See also `radix`.
st={ON|OFF}	On	Print symbol table in list file.
t={ON|OFF}	Off	Truncate lines of listing (otherwise wrap).
w={0|1|2}	0	Set the message level. See also `errorlevel`.
x={ON|OFF}	On	Turn macro expansion on or off.

`list` – LISTING OPTIONS
set radix

- Example:

```
list r=dec  ;set default to decimal
```

- Radix forms for constants are:
  - Hexadecimal, decimal, octal, binary, and ASCII

- The default radix is hexadecimal

  – The default radix determines what value will be assigned to constants in the object file when a radix is not explicitly specified by a base descriptor

`list` – LISTING OPTIONS
Set Processor Type

- Processor type is set in MPLAB IDE
    - *Configure>Device*
- Therefore, we will not need to set the type within the code
- If using another IDE ... you may see the following:

```
list p=16f684
```

`__config` – SET PROCESSOR CONFIGURATION BITS

- Syntax (NOTE: there are two underscore characters)

  `__config` *`expr`*

- Sets the processor's configuration bits

- Before this directive is used ...

  – the processor must be declared through
    - *Configure>Select Device* if using MPLAB IDE ... or ... one of the following ...
      - » the command line
      - » the list directive
      - » the processor directive

`__config` – SET PROCESSOR CONFIGURATION BITS

- Place configuration bit assignments at the beginning of your code
- Use the configuration options (names) in the standard include (*.inc) file
    - These names can be bitwise ANDed together (**&**) to declare multiple configuration bits
- For the PIC16F684 ... we have the following:

```
        INCLUDE "p16f684.inc"

        __CONFIG _FCMEN_OFF & _IESO_OFF & _BOD_OFF & _CPD_OFF & _CP_OFF
        & _MCLRE_OFF & _PWRTE_ON & _WDT_OFF & _INTOSCIO
```

REGISTER 12-1: CONFIG – CONFIGURATION WORD (ADDRESS: 2007h)

—	—	FCMEN	IESO	BODEN1	BODEN0	$\overline{CPD}$	$\overline{CP}$	MCLRE	$\overline{PWRTE}$	WDTE	FOSC2	FOSC1	FOSC0
bit 13													bit 0

bit 13-12 **Unimplemented**: Read as '1'

bit 11 **FCMEN:** Fail-Safe Clock Monitor Enabled bit
- 1 = Fail-Safe Clock Monitor is enabled
- 0 = Fail-Safe Clock Monitor is disabled

bit 10 **IESO:** Internal External Switchover bit
- 1 = Internal External Switchover mode is enabled
- 0 = Internal External Switchover mode is disabled

bit 9-8 **BODEN<1:0>:** Brown-out Detect Selection bits(1)
- 11 = BOD enabled
- 10 = BOD enabled during operation and disabled in Sleep
- 01 = BOD controlled by SBODEN bit (PCON<4>)
- 00 = BOD disabled

bit 7 **$\overline{CPD}$:** Data Code Protection bit(2)
- 1 = Data memory code protection is disabled
- 0 = Data memory code protection is enabled

bit 6 **$\overline{CP}$:** Code Protection bit(3)
- 1 = Program memory code protection is disabled
- 0 = Program memory code protection is enabled

bit 5 **MCLRE:** RA3/$\overline{MCLR}$ pin function select bit(4)
- 1 = RA3/$\overline{MCLR}$ pin function is $\overline{MCLR}$
- 0 = RA3/$\overline{MCLR}$ pin function is digital input, $\overline{MCLR}$ internally tied to V_{DD}

bit 4 **$\overline{PWRTE}$:** Power-up Timer Enable bit
- 1 = PWRT disabled
- 0 = PWRT enabled

bit 3 **WDTE:** Watchdog Timer Enable bit
- 1 = WDT enabled
- 0 = WDT disabled and can be enabled by SWDTEN bit (WDTCON<0>)

bit 2-0 **FOSC<2:0>:** Oscillator Selection bits
- 111 = RC oscillator: CLKOUT function on RA4/OSC2/CLKOUT pin, RC on RA5/OSC1/CLKIN
- 110 = RCIO oscillator: I/O function on RA4/OSC2/CLKOUT pin, RC on RA5/OSC1/CLKIN
- 101 = INTOSC oscillator: CLKOUT function on RA4/OSC2/CLKOUT pin, I/O function on RA5/OSC1/CLKIN
- 100 = INTOSCIO oscillator: I/O function on RA4/OSC2/CLKOUT pin, I/O function on RA5/OSC1/CLKIN
- 011 = EC: I/O function on RA4/OSC2/CLKOUT pin, CLKIN on RA5/OSC1/CLKIN
- 010 = HS oscillator: High-speed crystal/resonator on RA4/OSC2/CLKOUT and RA5/OSC1/CLKIN
- 001 = XT oscillator: Crystal/resonator on RA4/OSC2/CLKOUT and RA5/OSC1/CLKIN
- 000 = LP oscillator: Low-power crystal on RA4/OSC2/CLKOUT and RA5/OSC1/CLKIN

Source Code Guidelines ...

Source Code Guidelines

- Your source code should conform to the following basic guidelines ...
  - Each line of the source file may contain up to four types of information ...
    - *Labels*
    - *Mnemonics*, *Directives* and *Macros*
    - *Operands*
    - *Comments*

Source Code Guidelines

- *Labels* ...
  - Used to represent a line or group of code, or a constant value
  - It is needed for branching instructions

Source Code Guidelines

- *Mnemonics* ...

  – Tell the assembler what machine instructions to assemble

    - For example ...
      – addition (add), branches (goto) or moves (movwf)

    - Unlike labels that you create yourself ...

      – Mnemonics are provided by the assembly language

      – Mnemonics are not case sensitive

Source Code Guidelines

- *Directives* ...
  - Are assembler commands that appear in the source code but are not usually translated directly into opcodes
  - They are used to control the assembler ...
    - Its input ...
    - Its output ...and ...
    - Data allocation
  - Directives are not case sensitive

Source Code Guidelines

- *Macros* ...
  - Are user defined sets of instructions and directives that ...
    - Will be evaluated in-line with the assembler source code whenever the macro is invoked

Source Code Guidelines

- *Operands* ...
  - Provide information to the instruction on the data that should be used ...and ...
  - The storage location for the instruction
- Operands must be separated from mnemonics by one or more spaces, or tabs
- Multiple operands must be separated by commas

Source Code Guidelines

- *Comments* ...
  - Comments are text explaining the operation of a line or lines of code
  - The MPASM assembler treats anything after a semicolon as a comment
    - All characters following the semicolon are ignored through the end of the line
  - String constants containing a semicolon are allowed and are not confused with comments

Source Code Guidelines

- The order and position of the information is important
  - *Labels* start in column one
  - *Mnemonics* start in column two or beyond
  - *Operands* follow the mnemonic
  - *Comments* may follow the operands, mnemonics or labels, and can start in any column
- The maximum column width is 255 characters

Source Code Guidelines

- White space or a colon must separate the label and the mnemonic
  - *White space* is one or more spaces or tabs
  - White space should be used to make your code easier for people to read
  - Unless within character constants, any white space means the same as exactly one space.
- White space must separate the mnemonic and the operand(s)
  - Multiple operands must be separated by commas
- White space is used to separate pieces of a source line

Source Code Guidelines

Labels	Mnemonics Directives Macros	Operands	Comments
↓	↓	↓	↓

```
      list      p=18f452
      #include  p18f452.inc

Dest  equ       0x0B    ;Define constant

      org       0x0000  ;Reset vector
      goto      Start

      org       0x0020  ;Begin program

Start
      movlw     0x0A
      movwf     Dest

      bcf       Dest, 3 ;This line uses 2 operands
      goto      Start
      end
```

Labels ...

Labels

- Rules for defining labels ...
    - Place in the label column (start at the 1st position)
    - Must begin with alpha character or underscore bar
    - Labels may be up to 31 characters long
    - Labels are case sensitive
- The underscore (_) is a useful means of separating words as ...
    - Spaces are not allowed

Labels

- A label is used to represent a line or group of code, or a constant value
- It is needed for branching instructions
- Labels should start in column 1
- They may be followed by a colon (:), space, tab or the end of line

Labels

- Labels *must* ...
  - Begin with an alpha character or an under bar (_)
  - They may contain alphanumeric characters, the under bar and the question mark

- Labels *must* ***not*** ...
  - Begin with two leading underscores, e.g., __config
  - Begin with a leading underscore and number, e.g., _2NDLOOP
  - Be an assembler reserved word
  - Labels may be up to 32 characters long
    - By default they *are* case sensitive

Equate ...

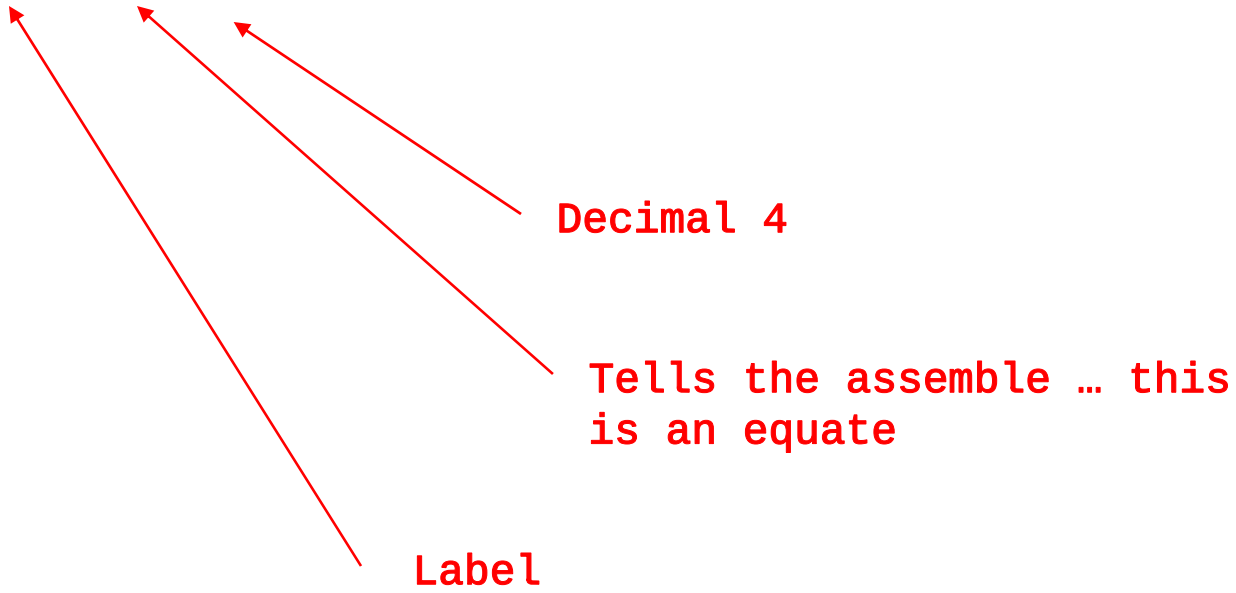
Equate

- Equate is used to assign the value of *expr* to the *label*
  - *label* equ *expr*

Equates

- Example

```
four equ 4 ; assigned the numeric value of 4 to label four
```



Literals ...

Literals

- Literals are constants or numbers
  - Usually hexadecimal numbers

Origin ...

Origin (ORG)

Column 1	Column 2 **This Is An ORG**	Column 3 **Hex Address**
not used	org	

`org` – SET PROGRAM ORIGIN

- Syntax

  [*label*] `org` *expr*

- Set the program origin for subsequent code at the address defined in *expr*

- If *label* is specified, it will be given the value of the *expr*

- If no org is specified, code generation will begin at address zero

org – SET PROGRAM ORIGIN

- *org* usages ...
  - Defines the address where program code starts
  - Establish the location of a table
  - Establish the start of an interrupt service routine
- Example:

```
    org   0x000
```

Program ...

Program

Column 1	Column 2	Column 3
Label	**Instruction**	**Literal Or Label**

Program

- The code ...
  - Main portion of the program

End ...

End

Column 1	Column 2	Column 3
	This Is **End**	
	end	

end – END PROGRAM BLOCK

- Syntax

  end

- Indicates the end of the program
- You will need at least one end directive in any assembly program to indicate the end of a build
- In a single assembly file program, one and only one end must be used

end – END PROGRAM BLOCK

- Example:

```
#include p18f684.inc
: ; executable code
: ;
end ; end of instructions
```

Include ...

#include – INCLUDE ADDITIONAL SOURCE FILE

- The specified file is read in as source code
  - The effect is the same as if the entire text of the included file were inserted into the file at the location of the include statement
- Upon end-of-file, source code assembly will resume from the original source file
  - Up to 5 levels of nesting are permitted
  - Up to 255 include files are allowed

#include – INCLUDE ADDITIONAL SOURCE FILE

- Syntax
  - **Preferred:**

```
#include include_file
#include "include_file"
#include <include_file>
```

  - **Supported:**

```
include include_file
include "include_file"
include <include_file>
```

#include – INCLUDE ADDITIONAL SOURCE FILE

- If *include_file* contains any spaces
  - It must be enclosed in quotes or angle brackets
- If a fully qualified path is specified ...only that path will be searched
  - Otherwise, the search order is:
    - current working directory
    - source file directory
    - MPASM assembler executable directory

#include – INCLUDE ADDITIONAL SOURCE FILE

- You should use the include directive once to include that standard header file for your selected processor
- This file contains defined register, bit and other names for a specific processor, so there is no need for you to define all of these in your code
- Example:

```
#include “p16f684.inc”
```

cblock ...

cblock – DEFINE A BLOCK OF CONSTANTS

- Defines a list of named sequential symbols
- Use this directive in place of or in addition to the equ directive
- The purpose of this directive is to assign address offsets to many labels
- The list of names ends when an *endc* directive is encountered
- *expr* indicates the starting value for the first name in the block
  - If no expression is found, the first name will receive a value one higher than the final name in the previous cblock
  - If the first cblock in the source file has no *expr*, assigned values start with zero

cblock – DEFINE A BLOCK OF CONSTANTS

- If *increment* is specified, then the next *label* is assigned the value of *increment* higher than the previous *label*
- Multiple names may be given on a line, separated by commas
- cblock is useful for defining constants in program and data memory for absolute code generation.
- Syntax

```
cblock [expr]
        label[:increment][,label[:increment]]
endc
```

Output files ...

Files Created by the Assembler

- .LST and .HEX files
- The .LST file ..
    - Provides a mapping of source code to object code
    - It also provides a list of symbol values, memory usage information and the number of errors, warnings and messages generated
    - This file may be viewed in MPLAB IDE by:
        1. Selecting *File>Open* to launch the Open dialog
        2. Selecting “List files (*.lst)” from the “Files of type” drop-down list
        3. Locating the desired list file
        4. Clicking on the list file name
        5. Clicking **Open**

Files Created by the Assembler

- The .HEX file ...
  - Hexadecimal object code which will be used by the programmer to program the PIC chip

Instruction Set ...

PIC16F684 Instruction Set Summary

- There are 35 instructions ...
- Plus two obsolete instructions that we will still use
    - `OPTION` and `TRIS` instructions

*** We will get a message stating that they are obsolete ***

- To maintain upward compatibility with future products these two instruction will need to be modified

PIC16F684 Instruction Set Summary

- The PIC16F684 instruction is comprised of three basic categories:
    - **Byte-oriented** operations
    - **Bit-oriented** operations
    - **Literal and control** operations

- Each PIC16 instruction is a 14-bit word divided into ...
    - An **opcode**
        - – which specifies the instruction type
    - And one or more **operands**
        - – which further specify the operation of the instruction

TABLE 13-2: PIC16F684 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C, DC, Z	**1, 2**
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	**1, 2**
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	**2**
CLRW	–	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	**1, 2**
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	**1, 2**
DECFSZ	f, d	Decrement f, Skip if 0	1**(2)**	00 1011 dfff ffff		**1, 2, 3**
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	**1, 2**
INCFSZ	f, d	Increment f, Skip if 0	1**(2)**	00 1111 dfff ffff		**1, 2, 3**
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	**1, 2**
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	**1, 2**
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	–	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	**1, 2**
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	**1, 2**
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C, DC, Z	**1, 2**
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		**1, 2**
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	**1, 2**

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF GPIO, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

TABLE 13-2: PIC16F684 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1, 2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1, 2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF GPIO, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

TABLE 13-2: PIC16F684 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C, DC, Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call Subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	–	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$, PD	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	–	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	–	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	–	Go into Standby mode	1	00 0000 0110 0011	$\overline{TO}$, PD	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C, DC, Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF GPIO, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

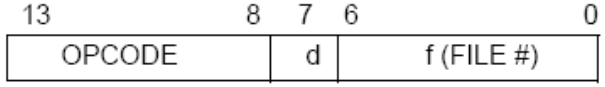
3: If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

TABLE 13-1: OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1.
PC	Program Counter
$\overline{TO}$	Time-out bit
PD	Power-down bit

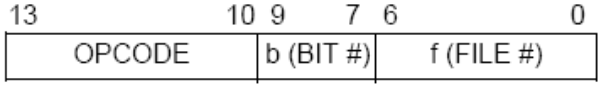
FIGURE 13-1: GENERAL FORMAT FOR INSTRUCTIONS

Byte-oriented file register operations



d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

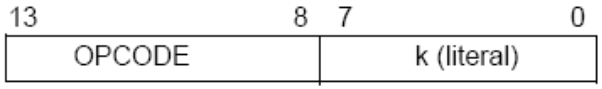
Bit-oriented file register operations



b = 3-bit bit address
f = 7-bit file register address

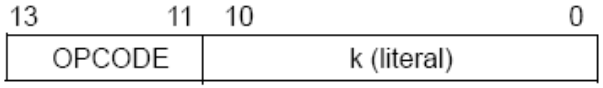
Literal and control operations

General



k = 8-bit immediate value

CALL and GOTO instructions only



k = 11-bit immediate value

Commands that we shall use today

- comments ... semicolon (;)
- MOVLW
- MOVWF
- CLRF
- BCF
- BSF
- DECFSZ
- CALL
- GOTO
- END

Semicolon

- Semicolon (;)
  - Used for comments
  - Ignore everything following the semicolon on the line

MOVLW

MOVLW	**Move literal to W**
Syntax:	[*label*] MOVLW k
Operands:	$0 \leq k \leq 255$
Operation:	$k \rightarrow (W)$
Status Affected:	None
Description:	The eight bit literal 'k' is loaded into W register. The don't cares will assemble as '0's.
Words:	1
Cycles:	1
Example	`MOVLW 0x5A` After Instruction W = 0x5A

MOVWF

MOVWF	**Move W to f**
Syntax:	[*label*] MOVWF f
Operands:	0 ≤ f ≤ 127
Operation:	(W) → (f)
Status Affected:	None
Description:	Move data from W register to register 'f'.
Words:	1
Cycles:	1
Example	`MOVW OPTION F` Before Instruction OPTION = 0xFF W = 0x4F After Instruction OPTION = 0x4F W = 0x4F

CLRF

CLRF	**Clear f**
Syntax:	[*label*] CLRF f
Operands:	$0 \leq f \leq 127$
Operation:	$00h \rightarrow (f)$ $1 \rightarrow Z$
Status Affected:	Z
Description:	The contents of register 'f' are cleared and the Z bit is set.

BCF

BCF	**Bit Clear f**
Syntax:	[*label*] BCF f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	$0 \rightarrow (f<b>)$
Status Affected:	None
Description:	Bit 'b' in register 'f' is cleared.

BSF

BSF	**Bit Set f**
Syntax:	[*label*] BSF f,b
Operands:	$0 \le f \le 127$ $0 \le b \le 7$
Operation:	$1 \rightarrow (f<b>)$
Status Affected:	None
Description:	Bit ‘b’ in register ‘f’ is set.

DECFSZ

DECFSZ	Decrement f, Skip if 0
Syntax:	[*label*] DECFSZ f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) - 1 → (destination); skip if result = 0
Status Affected:	None
Description:	The contents of register 'f' are decremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'. If the result is '1', the next instruction is executed. If the result is '0', then a NOP is executed instead, making it a 2-cycle instruction.

CALL

CALL	Call Subroutine
Syntax:	[*label*] CALL k
Operands:	$0 \le k \le 2047$
Operation:	(PC)+ 1→ TOS, k → PC<10:0>, (PCLATH<4:3>) → PC<12:11>
Status Affected:	None
Description:	Call Subroutine. First, return address (PC + 1) is pushed onto the stack. The eleven-bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. CALL is a two-cycle instruction.

GOTO

GOTO	Unconditional Branch
Syntax:	[*label*] GOTO k
Operands:	$0 \le k \le 2047$
Operation:	$k \rightarrow PC<10:0>$ $PCLATH<4:3> \rightarrow PC<12:11>$
Status Affected:	None
Description:	GOTO is an unconditional branch. The eleven-bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two-cycle instruction.

END

`end` – END PROGRAM BLOCK

4.22.1 Syntax

`end`

4.22.2 Description

Indicates the end of the program.

4.22.3 Usage

This directive is used in the following types of code: absolute or relocatable. For information on types of code, see **Section 1.6 "Assembler Operation"**.

You will need at least one `end` directive in any assembly program to indicate the end of a build. In a single assembly file program, one and only one `end` must be used.

Assembly Language Code ...

Developing Code Using Assembly Language

- First program in assembly will be to flash D0 on the PICkit 1 Starter Kit ... just as we did with our first lab using the PICkit

- Will light DO ... 2 times per second

Flash_D0.asm – An “Eye Chart”

```
 title “Flash_D0.asm – Flash D0 LED”

; ---------------------------------
; Flash_D0.asm
;
; Microprocessors B 17.384
;
; xxxxxxxx - Put in Semester (i.e. Spring 2008) here
;
; xxxxxxxx - Put in your name here
;
; xx/xx/xx - Put date here
;
; ---------------------------------

 list r=dec

#include "p16f684.inc"

 __CONFIG _FCMEN_OFF & _IESO_OFF & _BOD_OFF & _CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON & _WDT_OFF & _INTOSCIO

; ---------------------------------
; Defining constants or variables in data memory space

 Cblock 0x20                                                                    ; start of GPR

count                                                                           ;

 endc                                                                           ; This directive must be
                                                                                ; supplied to terminate the
                                                                                ; cblock list

;
; ----- main program --------------

                                        org             0x000                   ; Hex address 0x000, the first
                                                                                ; program memory location
;
start                                   BCF             STATUS,RP0              ; Select Bank 0
;
                                        CLRF            PORTA                   ; Initialize PORTA (to all zeros)
;
                                        MOVLW           7                       ; Load w with 7
                                        MOVWF           CMCON0                  ; Load CMCON0 with 7
                                                                                ; Turns off comparators
;
                                        BSF             STATUS,RP0              ; Select Bank 1
;
                                        CLRF            ANSEL                   ; Shut off ADC (digital I/O)
;
                                        MOVLW           b'001111'               ; Load w – RA4 and RA5 outputs
                                        MOVWF           TRISA                   ; copy w to TRIS PORTA
;
                                        BCF             STATUS,RP0              ; Select Bank 0
;
Loop                                    BSF             PORTA,4                 ; Make RA4 high -- D0 ON
;
                                        CALL            Delay                   ; Goto the delay routine
;
                                        BCF             PORTA,4                 ; Make RA4 low -- D0 OFF
;
                                        CALL            Delay                   ; Goto the delay routine
;
                                        GOTO            Loop                    ; Do it again
;
;
Delay                                   MOVLW           10                      ; Decimal 10
                                        MOVWF           count                   ; Initialize counter to 10
Repeat                                  DECFSZ          count,f                 ; Decrement counter
                                        GOTO            Repeat                  ; If counter <> 0
                                        RETURN                                  ; If counter = 0 (end delay)
;
 end
```

Flash_D0.asm (Slide 1 of 9)

- Identify the title (do not start in position1 _):

```
title “Flash_D0.asm – Flash D0 LED”
```

Flash_D0.asm (Slide 2 of 9)

- Add our required class header information

```
; -------------------------------
; Flash_D0.asm
;
; Microprocessors B 17.384
;
; xxxxxxxx - Put in Semester (i.e. Spring 2009) here
;
; xxxxxxxx - Put in your name here
;
; xx/xx/xx - Put date here
;
; -------------------------------
```

Flash_D0.asm (Slide 3 of 9)

- Set radix to decimal, vice hexadecimal
- Include the PIC16F684 file
- Add the configuration word

```
 list r=dec

#include "p16f684.inc"

 __CONFIG _FCMEN_OFF & _IESO_OFF & _BOD_OFF & _CPD_OFF & _CP_OFF
& _MCLRE_OFF &      _PWRTE_ON & _WDT_OFF & _INTOSCIO
```

Flash_D0.asm (Slide 4 of 9)

- Make provisions for constants or variables in data memory space

```
 ; ---------------------------------
; Defining constants or variables in data memory space

 Cblock 0x20                    ; start of GPR

count                           ;

 endc                           ; This directive must be
                                ; supplied to terminate the
                                ; cblock list
;
```

Flash_D0.asm (Slide 5 of 9)

- Define the address where the program code starts

```
; ----- main program --------------

                    org         0x000               ; Hex address 0x000, the first
                                                    ; program memory location
;
```

Flash_D0.asm (Slide 6 of 9)

- Establish initial conditions

```
 start          BCF     STATUS,RP0      ; Select Bank 0
;
                CLRF    PORTA           ; Initialize PORTA (to all zeros)
;
                MOVLW   7               ; Load w with 7
                MOVWF   CMCON0          ; Load CMCON0 with 7
                                        ; Turns off comparators
;
                BSF     STATUS,RP0      ; Select Bank 1
;
                CLRF    ANSEL           ; Shut off ADC (digital I/O)
;
                MOVLW   b'001111'       ; Load w – RA4 and RA5 outputs
                MOVWF   TRISA           ; copy w to TRIS PORTA
;
                BCF     STATUS,RP0      ; Select Bank 0
;
```

Flash_D0.asm (Slide 7 of 9)

- Loop

```
 Loop            BSF     PORTA,4         ; Make RA4 high -- D0 ON
;
                 CALL    Delay           ; Goto the delay routine
;
                 BCF     PORTA,4         ; Make RA4 low -- D0 OFF
;
                 CALL    Delay           ; Goto the delay routine
;
                 GOTO    Loop            ; Do it again
;
```

Flash_D0.asm (Slide 8 of 9)

- Create a Delay Routine

```
;
Delay           MOVLW 10                ; Decimal 10
                MOVWF count             ; Initialize counter to 10
Repeat          DECFSZ          count,f ; Decrement counter
                GOTO    Repeat          ; If counter <> 0
                RETURN                  ; If counter = 0 (end delay)
;
```

Flash_D0.asm (Slide 9 of 9)

- Ending the program

```
;
end
```

Lab

Lab

- Finish Lab # 4
- Lab # 5 ... is optional ... will be available next week... however ...

  » It is recommended that you do the lab

  – There is no report for Lab #5

Next Class …

Next Class Topics

- Comparators, Timers, and Pulse Width Modulation

HomeWork

Homework

1. Lab # 4 Report ... due November 23, 2010

2. Read the material from Today's class ...

    - MPASM Users Guide

3. Start preparing for Exam #2 ... Scheduled for November 30th

Time To ...

Start the Lab ...

References

1. PIC16F684 Data Sheet 41202F

2. OPTREX CORPORATION

   “DOT MATRIX CHARACTER LCD MODULE USER’S MANUAL“

3. MPASM Users Guide