MICROPROCESSORS A (17.383)

Fall 2010

Lecture Outline

Class # 03

September 21, 2010

Dohn Bowden

Today's Lecture

- Syllabus review
- Microcontroller Hardware and/or Interface
- Programming/Software
- Lab
- Homework
- Finish Lab # 1
- Start Lab # 2

Course Admin

Administrative

- Admin for tonight ...
  - Lectures 1 and 2 are available on the web site
  - Syllabus Review
    - No changes
    - Lab Report for Lab #1 is next week (September 28, 2010)
    - **Exam #1** in 2 weeks (October 05, 2010)

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		1
5	10/05/10	**Examination 1**	2 con't	
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	LCD Interface and Assembly Language	4	
9	11/09/10	Comparators	4 con't	3
10	11/16/10	Timers and 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	

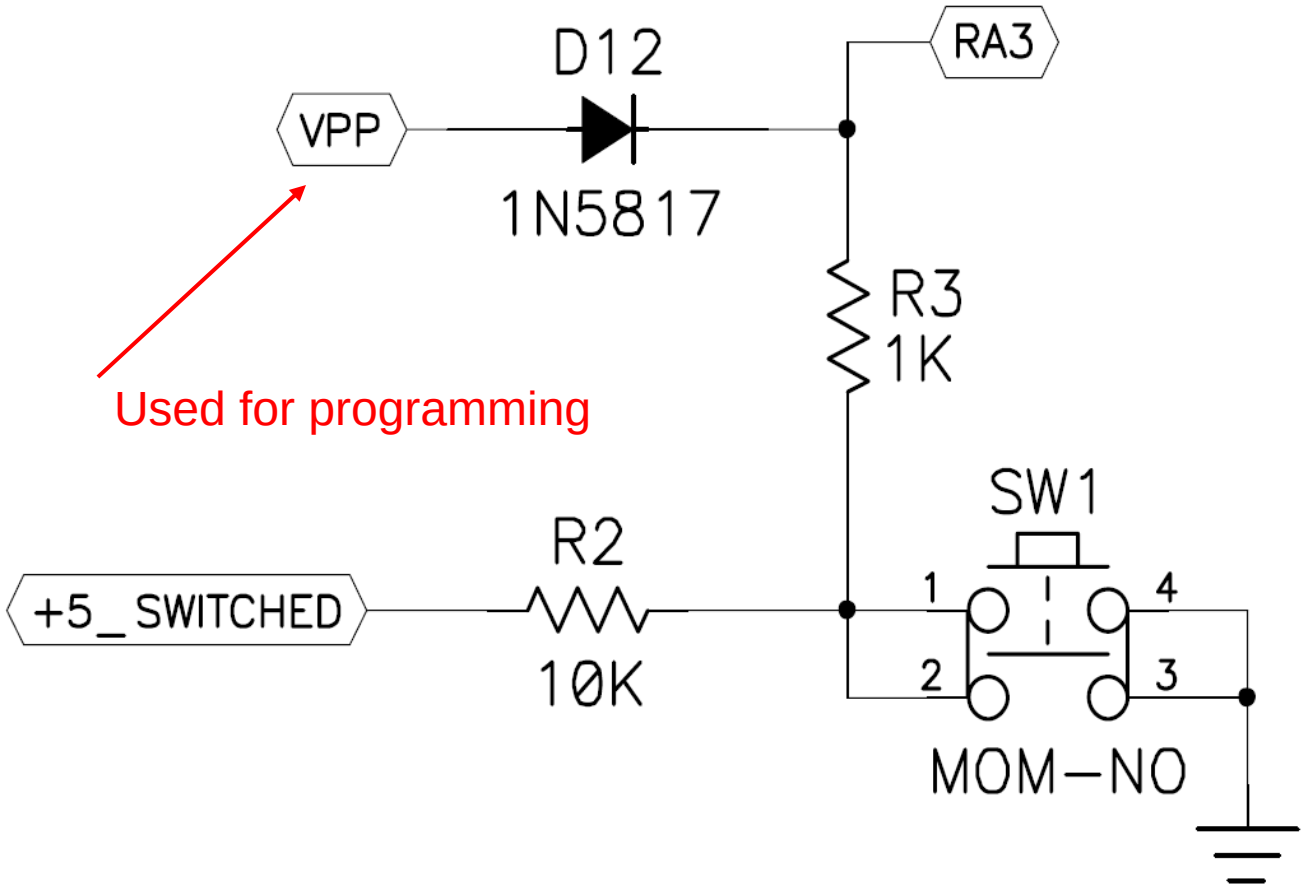
Microcontroller Hardware and / or Interfaces

PIC16F684 Interfacing

- Switches
- LEDs

Switches ...

PICkit 1 Starter Kit - Switch (SW1) Schematic



PIC16F684 Pin Diagram

14-pin PDIP, SOIC, TSSOP

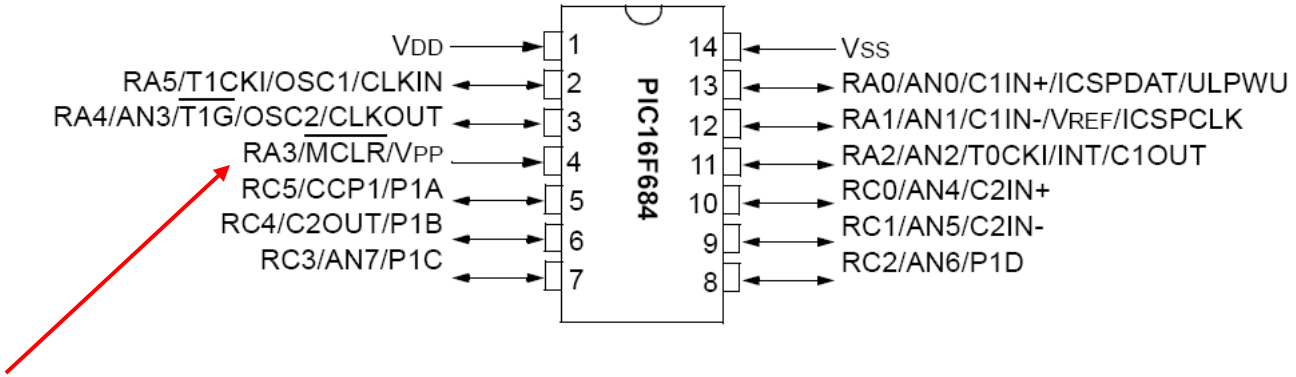


TABLE 1-1: PIC16F684 PINOUT DESCRIPTION

Name	Function	Input Type	Output Type	Description
RA0/AN0/C1IN+/ICSPDAT/ULPWU	RA0	TTL	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	AN0	AN	—	A/D Channel 0 input
	C1IN+	AN	—	Comparator 1 input
	ICSPDAT	TTL	CMOS	Serial Programming Data I/O
	ULPWU	AN	—	Ultra Low-power Wake-up input
RA1/AN1/C1IN-/VREF/ICSPCLK	RA1	TTL	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	AN1	AN	—	A/D Channel 1 input
	C1IN-	AN	—	Comparator 1 input
	VREF	AN	—	External Voltage Reference for A/D
	ICSPCLK	ST	—	Serial Programming Clock
RA2/AN2/T0CKI/INT/C1OUT	RA2	ST	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	AN2	AN	—	A/D Channel 2 input
	T0CKI	ST	—	Timer0 clock input
	INT	ST	—	External Interrupt
	C1OUT	—	CMOS	Comparator 1 output
RA3/$\overline{\text{MCLR}}$/VPP	RA3	TTL	—	PORTA input with interrupt-on-change
	$\overline{\text{MCLR}}$	ST	—	Master Clear w/internal pull-up
	VPP	HV	—	Programming voltage
RA4/AN3/$\overline{\text{T1G}}$/OSC2/CLKOUT	RA4	TTL	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	AN3	AN	—	A/D Channel 3 input
	$\overline{\text{T1G}}$	ST	—	Timer1 gate
	OSC2	—	XTAL	Crystal/Resonator
	CLKOUT	—	CMOS	FOSC/4 output
RA5/T1CKI/OSC1/CLKIN	RA5	TTL	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	T1CKI	ST	—	Timer1 clock
	OSC1	XTAL	—	Crystal/Resonator
	CLKIN	ST	—	External clock input/RC oscillator connection
RC0/AN4/C2IN+	RC0	TTL	CMOS	PORTC I/O
	AN4	AN	—	A/D Channel 4 input
	C2IN+	AN	—	Comparator 2 input
RC1/AN5/C2IN-	RC1	TTL	CMOS	PORTC I/O
	AN5	AN	—	A/D Channel 5 input
	C2IN-	AN	—	Comparator 2 input
RC2/AN6/P1D	RC2	TTL	CMOS	PORTC I/O
	AN6	AN	—	A/D Channel 6 input
	P1D	—	CMOS	PWM output
RC3/AN7/P1C	RC3	TTL	CMOS	PORTC I/O
	AN7	AN	—	A/D Channel 7 input
	P1C	—	CMOS	PWM output
RC4/C2OUT/P1B	RC4	TTL	CMOS	PORTC I/O
	C2OUT	—	CMOS	Comparator 2 output
	P1B	—	CMOS	PWM output
RC5/CCP1/P1A	RC5	TTL	CMOS	PORTC I/O
	CCP1	ST	CMOS	Capture input/Compare output
	P1A	—	CMOS	PWM output
VSS	VSS	Power	—	Ground reference
VDD	VDD	Power	—	Positive supply

Legend: TTL = TTL input buffer, ST = Schmitt Trigger input buffer, AN = Analog input

PIC16F684 PIN 4 – RA3

RA3/$\overline{\text{MCLR}}$/V_{PP}	RA3	TTL	—	PORTA input with interrupt-on-change
	$\overline{\text{MCLR}}$	ST	—	Master Clear w/internal pull-up
	V_{PP}	HV	—	Programming voltage

Setting RA3 for Input

PORTA = 0

TRISA3 is ***ALWAYS*** an input ... it can never be an output

Therefore ... you do not need to set TRISA3 = 1

REGISTER 4-2: TRISA – PORTA TRI-STATE REGISTER (ADDRESS: 85h)

U-0	U-0	R/W-1	R/W-1	R-1	R/W-1	R/W-1	R/W-1
—	—	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0
bit 7							bit 0

bit 7-6: **Unimplemented:** Read as '0'

bit 5-0: **TRISA<5:0>:** PORTA Tri-State Control bit

1 = PORTA pin configured as an input (tri-stated)

0 = PORTA pin configured as an output

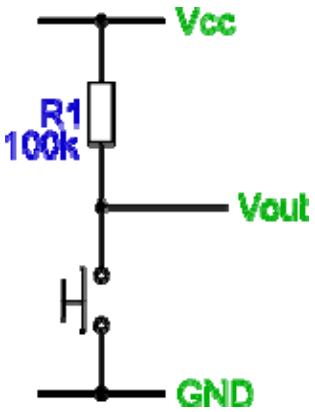
Note 1: TRISA<3> always reads '1'.

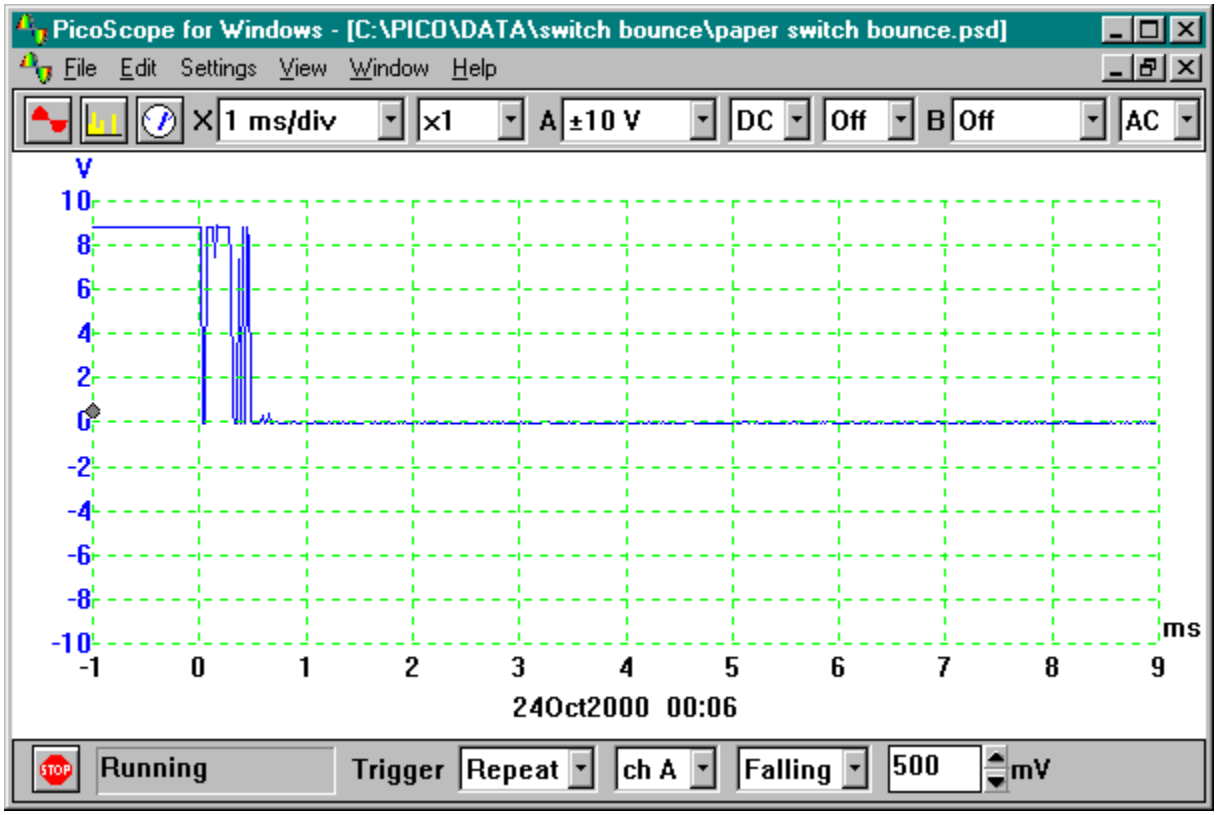
2: TRISA<5:4> always reads '1' in XT, HS and LP OSC modes.

Mechanical Switches

- The switch is connected across a +5 volt supply to the data input pin of the microcontroller via a resister
- An issue arises with mechanical switches ...
  - They do not cleanly open and close
- They can produce spikes which can be interpreted as false switch actions
  - This is referred to as *Switch Bouncing*

Switch Bouncing





Switch Debouncing – Elimination of …

- There are a variety of methods to eliminate switch bouncing
  - One method is the addition of a capacitor across the switch
  - Another method is a software approach
    - Code which can be incorporated into your application which will eliminate switch bouncing

Code to Eliminate Switch Bouncing

- Delays
  - Delays of around 20 msec
- The microcontroller will not look for switch action until the delay is over
  - Thus ... the switch has settled
  - False indications have been eliminated

LEDs ...

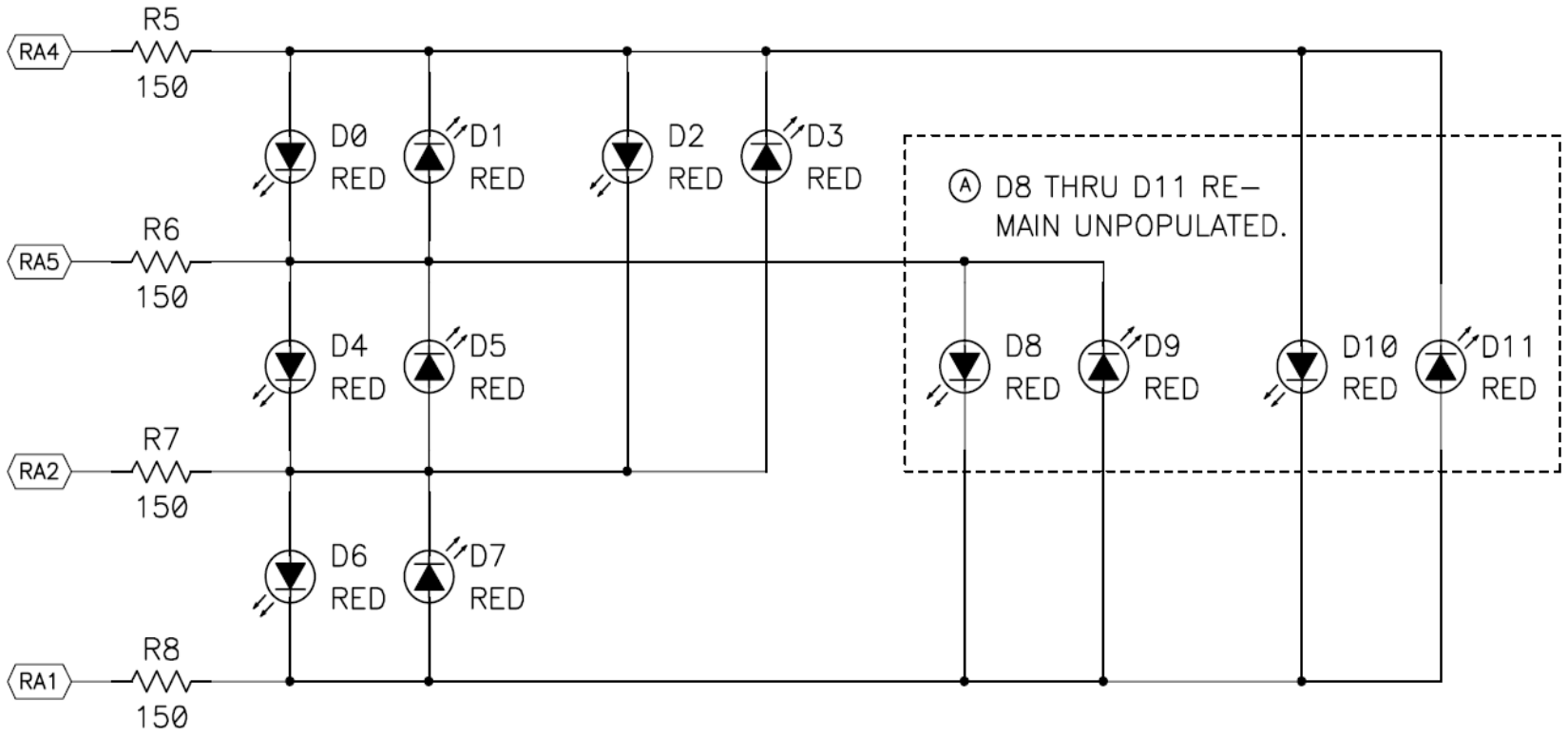
LEDs

- LEDs can be used for a multitude of applications
  - Indications
    - Power available
  - Indicates switch activation
    - By lighting once a switch has been depressed or changed

LEDs

- The PICkit 1 has an array of LEDs for multiple uses ...
  - Indications ... as we have seen in Lab #1
  - Switch actions ... as we shall see in Lab #2
  - Analog to Digital conversion results ... we shall see later in the course

PICkit 1 LED Layout



Programming / Software

Programming

- Commands/instructions that we will encounter tonight
    - C commands
    - PIC16F684 control

C commands – Last Week

- C program structure
- Comments
- ***# include***
- Integer variables
- ***for***
- ***while***
- ***NOP()***
- ***functions***

NEW C Commands

- ***if*** conditional
- ***else***
- ***else if***
- ***switch***
- ***Constant*** Declaration
- ***Bitwise*** Operators

The *if* Statement

- Testing a condition with ... the *if* statement
  - When the condition is True, or a value equal to 1, then C executes the statement that immediately follows, otherwise it is skipped
  - The *if* statement's format is as follows:

```
if (condition)
   Program_Statement;
```

The *if* Statement (Tests)

a == b Tests for equality
(has the value 1 [true] if a is equal to b, and 0 [false] otherwise)

a != b Test for inequality
(has the value 1 [true] if a is not equal to b, and 0 [false] otherwise

a > b Greater than
(has the value 1 [true] if a is greater than b, and 0 [false] otherwise)

a < b Less than
(has the value 1 [true] if a is less than b, and 0 [false] otherwise)

a >= b Greater than or equal to
(has the value 1 [true] if a is greater than or equal to b, and 0 [false] otherwise)

a <= b Less than or equal to
(has the value 1 [true] if a is less than or equal to b, and 0 [false] otherwise)

The *if* Statement (continued)

- Example ...

```
if (x == 1)
        do what is on this line;
```

- If the result of evaluating the condition is nonzero [True] (x is equal to 1), then C executes the statement that immediately follows, otherwise the line is skipped

else Statement

- Another general format for declaring an ***if*** statement is as follows:

```
if (condition)
    Program_Statement1;
else
    Program_Statement2;
```

  - If the condition is nonzero [True], Program_Statement1 is executed
  - Otherwise, if condition is zero [False], Program_Statement2 is executed

- The ***else*** statement provides a way to execute one block of code if a condition is true, another if it is false

else Statement

- Example ...

```
main()
{
    int number = 75;
    int mark;

    if (mark >= number)
        {
            Do something;
        }
    else
        {
            Do something else;
        }
}
```

The *if* - *else if* Statement

- If Program_Statement2 is another if statement, an ***else if*** chain is affected
- We use this when additional statements are being evaluated
- General format for declaring an ***if*** – ***else if*** statement is as follows:

```
if (condition)
   Program_Statement1;
else if (condition)
   Program_Statement2;
else
   Program_Statement3;
```

The *if* - *else if* Statement

- Example ...

```
main()
{
    int number = 75;
    int mark;

    if (mark >= number)
        {
          Do this line if true;
        }
    else if (mark >= 65)
        {
          Do this line if true;
        }
     else
        {
          If other conditions are false, then, do this line;
        }
}
```

The *switch* Statement

- The ***switch*** statement is a multi-way decision statement
- Unlike the multiple decision statement that can be created using if-else ...
- The switch statement evaluates the conditional expression and tests it against numerous constant values
- The branch corresponding to the value that the expression matches is taken during execution
- The value of the expressions in a switch statement must be an ordinal type i.e. integer, char, short, long, etc
  - Float and double are not allowed

The *switch* Statement

– The ***switch*** statement's format is as follows:

```
switch (expression)
{
    case constant_1:
      Program_Statement;
      Program_Statement;
      …
      break;
    case constant_2:
      Program_Statement;
      Program_Statement;
      …
      break;
    default:
      Program_Statement;
      Program_Statement;
      …
      break;
}
```

The *switch* Statement

- No two cases can have the same value
- Omitting the *break* statement from a particular case ...
  - Causes execution to continue into the next case
- If no case value matches the *expression* ...
  - The *default* case (if included) is executed
  - If the *default* case is not included ...
    - No statements contained in the ***switch*** are executed

The *switch* Statement - Example

```
int i = 4;
int Direction;

main()
{

    switch (i)
    {
        case 4:                 //  Go South if Index == 4
            Direction = 180;
            break;              //  Leave Switch Statement
        case 5:                 //  Go North if Index at 5
            Direction = 0;
            break;
        case 7:                 //  Go East if Index at 7
            Direction = 90;
            break;
        default:                //  Go West for Everything Else
            Direction = 270;
    }

    while(1 == 1);

}
```

Type Modifier – *const* (constant)

- The keyword ***const*** can be placed before a type declaration to tell the compiler that a value cannot be modified

- For example ...

```
const int xConstant = 47;
```

- Declares `xConstant` to be a constant integer
  - That is, it will not be set to anything else during the program execution

Type Modifier – *const* (constant)

- The keyword ***const*** converts the declared variable from a variable to a constant

- From our previous example, anytime the label `xConstant` is encountered, the compiler replaces it with the value 47

- You can no longer write to it ... for example, once the above is declared, you cannot do the following:

```
xConstant = 48;
```

Bit Manipulation

- The C language bitwise operators can be used to manipulate the contents of registers using Boolean arithmetic

- Bitwise operators can be used to perform the following on individual bits ...

  – Test
  – Set
  – Clear
  – Toggle

Bitwise Operators

- These operators allow you to easily process bit information

&	bitwise AND ... set if both bits are set
|	bitwise (inclusive) OR ... set if either bit is set
^	bitwise XOR ... set if only one of the two parameter bits are set
-	bitwise negation ... invert each bit

Bit Manipulation – *Testing Bits*

- Test to see if bit 3 is set

```
          0 1 0 0 1 1 0 0
AND (&)
          0 0 0 0 1 0 0 0
          ===============
          0 0 0 0 1 0 0 0
```

- Bit 3 was set, therefore the result has bit 3 set, because ...

  & will set if both bits are set

- If we used an ***if*** statement ... result would be true

Bit Manipulation – *Setting Bits*

- To set bit 4

```
          0 1 0 0 1 1 0 0
OR (|)
          0 0 0 1 0 0 0 0
          ===============
          0 1 0 1 1 1 0 0
```

- We set Bit 4 because ...

Inclusive OR (|) will set if either bit is set

Bit Manipulation – *Toggling Bits*

- To toggle bit 7 (when bit 7 was clear)

```
          0 1 0 0 1 0 0 0
XOR (^)
          1 0 0 0 0 0 0 0
          ===============
          1 1 0 1 1 0 0 0
```

- We toggled Bit 7 because ...

  ^ will set if only one of the two parameter bits are set

Bit Manipulation – *Toggling Bits*

- To toggle bit 7 (when bit 7 was set)

```
          1 1 0 0 1 0 0 0
XOR (^)
          1 0 0 0 0 0 0 0
          ===============
          0 1 0 1 1 0 0 0
```

- We toggled Bit 7 because ...

  ^ will set if only one of the two parameter bits are set

Bitwise Operators - Example

```
RA5 = RA5 ^ 1;
```

- Recall that ...

  bitwise XOR ... set if only one of the two parameter bits are set

  When RA5 = 0 ... RA5 would be changed to 1 because RA5 is zero and the constant is one (only one of the two bits are set)

  When RA5 = 1 ... RA5 would be changed to 0 because RA5 is one and the constant is also one (both bits are set, therefore the bit is cleared)

- So we effectively “toggle” the value of RA5

Information Coding

- Methods of representing information within microcontroller systems
- Data storage and program coding using numbering systems
  - Decimal
  - Hexadecimal
  - Binary
- Programming and data information will use these various numbering systems

Numbering Systems

- Decimal
  - Base 10
- Binary
  - Base 2
- Hexadecimal
  - Base 16

Numbering Formats

System	Format	Example
Decimal	##	0
Hexadecimal	0x##	0x3F
Binary	0b########	0b00111111

PIC16F684 Commands

- The following formats will set all bits in PORTA high

```
PORTA = 63;

PORTA = 0x3F;

PORTA = 0b111111;
```

- What are the advantages of the different formats?
  - Situation dependant ... one format may be easier to interpret based on what the program is doing

Lab

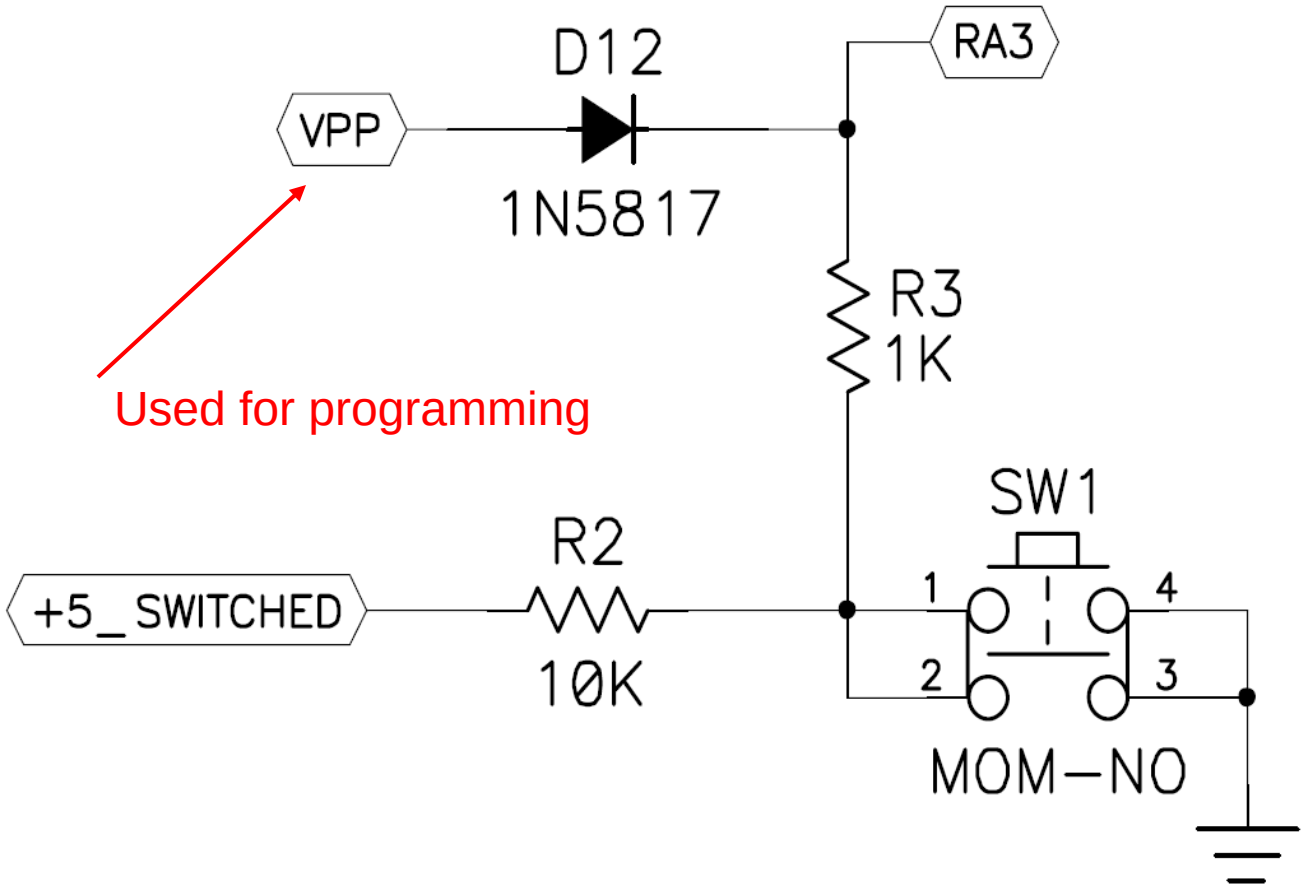
Lab Demonstrations

- Digital Input
- Debouncing switches via code development
- Digital Input and Output control
- Numbering Systems

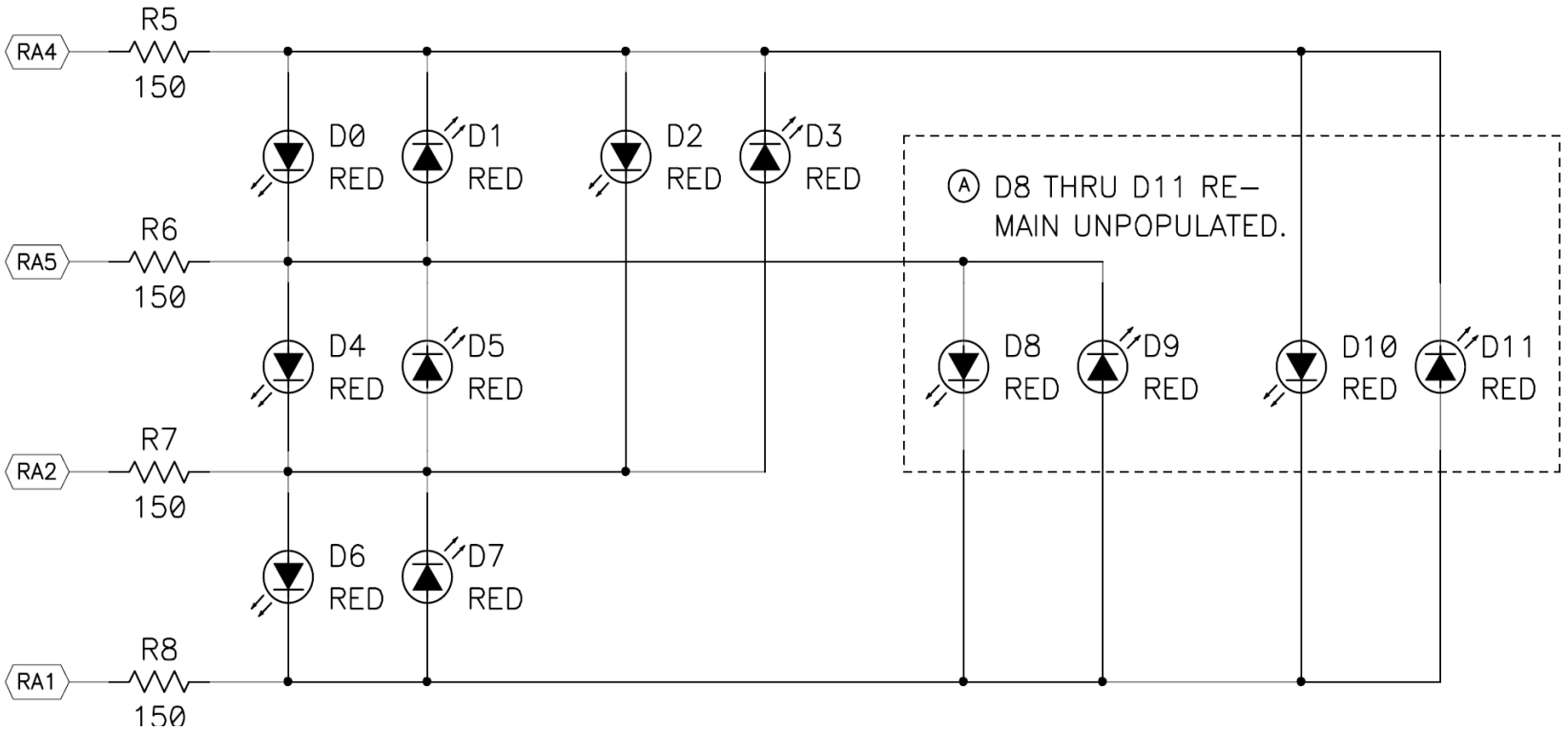
Design Parameters – Step 1

- Using the PICkit switch and LEDs
  - When the switch is off, all of the LEDs will be off
  - When the switch is on, D5 LED will be on
  - When the switch is release, D5 LED will go out

PICkit 1 Starter Kit - Switch (SW1) Schematic



Understand the Hardware



PIC16F684 Pin Diagram

14-pin PDIP, SOIC, TSSOP

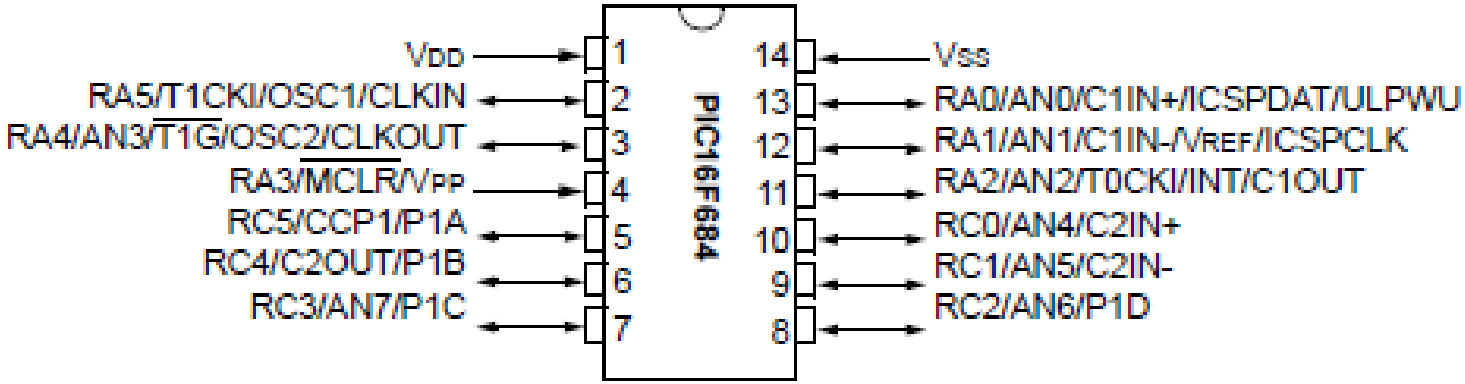


TABLE 1-1: PIC16F684 PINOUT DESCRIPTION

Name	Function	Input Type	Output Type	Description
RA0/AN0/C1IN+/ICSPDAT/ULPWU	RA0	TTL	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	AN0	AN	—	A/D Channel 0 input
	C1IN+	AN	—	Comparator 1 input
	ICSPDAT	TTL	CMOS	Serial Programming Data I/O
	ULPWU	AN	—	Ultra Low-power Wake-up input
RA1/AN1/C1IN-/VREF/ICSPCLK	RA1	TTL	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	AN1	AN	—	A/D Channel 1 input
	C1IN-	AN	—	Comparator 1 input
	VREF	AN	—	External Voltage Reference for A/D
	ICSPCLK	ST	—	Serial Programming Clock
RA2/AN2/T0CKI/INT/C1OUT	RA2	ST	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	AN2	AN	—	A/D Channel 2 input
	T0CKI	ST	—	Timer0 clock input
	INT	ST	—	External Interrupt
	C1OUT	—	CMOS	Comparator 1 output
RA3/$\overline{\text{MCLR}}$/VPP	RA3	TTL	—	PORTA input with interrupt-on-change
	$\overline{\text{MCLR}}$	ST	—	Master Clear w/internal pull-up
	VPP	HV	—	Programming voltage
RA4/AN3/$\overline{\text{T1G}}$/OSC2/CLKOUT	RA4	TTL	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	AN3	AN	—	A/D Channel 3 input
	$\overline{\text{T1G}}$	ST	—	Timer1 gate
	OSC2	—	XTAL	Crystal/Resonator
	CLKOUT	—	CMOS	Fosc/4 output
RA5/T1CKI/OSC1/CLKIN	RA5	TTL	CMOS	PORTA I/O w/programmable pull-up and interrupt-on-change
	T1CKI	ST	—	Timer1 clock
	OSC1	XTAL	—	Crystal/Resonator
	CLKIN	ST	—	External clock input/RC oscillator connection
RC0/AN4/C2IN+	RC0	TTL	CMOS	PORTC I/O
	AN4	AN	—	A/D Channel 4 input
	C2IN+	AN	—	Comparator 2 input
RC1/AN5/C2IN-	RC1	TTL	CMOS	PORTC I/O
	AN5	AN	—	A/D Channel 5 input
	C2IN-	AN	—	Comparator 2 input
RC2/AN6/P1D	RC2	TTL	CMOS	PORTC I/O
	AN6	AN	—	A/D Channel 6 input
	P1D	—	CMOS	PWM output
RC3/AN7/P1C	RC3	TTL	CMOS	PORTC I/O
	AN7	AN	—	A/D Channel 7 input
	P1C	—	CMOS	PWM output
RC4/C2OUT/P1B	RC4	TTL	CMOS	PORTC I/O
	C2OUT	—	CMOS	Comparator 2 output
	P1B	—	CMOS	PWM output
RC5/CCP1/P1A	RC5	TTL	CMOS	PORTC I/O
	CCP1	ST	CMOS	Capture input/Compare output
	P1A	—	CMOS	PWM output
VSS	VSS	Power	—	Ground reference
VDD	VDD	Power	—	Positive supply

Legend: TTL = TTL input buffer, ST = Schmitt Trigger input buffer, AN = Analog input

Design Parameters – Step 2

- Add debouncing Code to the previous code written

Debouncing Code Evaluation (sheet 1 of 1)

```
/*******************************************************
* Function: delay_20ms
*
* Description:  Causes a delay in program execution
*
* Notes:
*
* Delay was determined through trial and error
*
* Returns:  None
*
*******************************************************/
void delay_20ms(void)
{
      const int D20ms = 1150;      //  Declare a Constant for 20 ms Delay
      int i;

      for (i = 0; i < D20ms; i++);

      return;
}
/******** END OF delay_20ms ************************/
```

Design Parameters – Step 3

- Initial power on/programming, all LEDs are off.
- When the switch is depressed for the first time, LED D0 will come on.
- The LED turned on will stay energized until the switch is depressed a second time.
  - At that time, LED D0 will go out and LED D1 will come on.
- The process above will continue until LED D7 is turned on.
- The next time the switch is depressed, all LEDs will be off and the sequence will start all over.

  SUGGESTIONS: Use the ***switch*** or the ***if*** – ***if else*** statements.

- *Do not incorporate* the switch debouncing code into your file

Design Parameters – Step 3 con't

- Next ... incorporate the switch debouncing code into your file

Design Parameters – Step 4

- Rewrite your code using ...
  - Hexadecimal
  - Binary

Next Class …

Next Class Topics

- Transition from the PICkit to a development board
- Seven – Segment LEDs
- Binary operators
  - Shifting Bits

HomeWork

Homework

1. Read, as required, the C commands discussed in today's lecture found in the optional text (Programming in C)

2. Read the PIC16F684 data manual sections for the registers encountered tonight

3. Work on Lab #2

Time To ...

Start the Lab ...

References

1. PIC16F684 Data Sheet 41202F