# PIC Exercises 2014 Autumn

## The Status Register

**STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)**

| R/W-0 | R/W-0 | R/W-0 | R-1 | R-1 | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C |
| bit 7 | | | | | | | bit 0 |

The Status register contains the RP1 and RP0 bits, it also contains some useful information about the last command run. The three bits Z, DC and C bits will change based on the last arithmetic operation run. Zero will change on all logic operation, DC and C will change on Arithmetic operations (ADD, SUB and also RRF and RLF).

| | |
|---|---|
| Z: Zero Bit | 1 when last operation resulted in a 0 |
| C: Carry Bit | 1 when the last addition resulted in a number bigger than 15<br>0 when the last subtraction was less than 0<br>The Carry bit is included in the RRF and RLF rotate commands |

## The PCL Register

The Program Counter is a 13-bit wide number which tracks with command in the Program Memory the PIC is set to execute. When the Program Counter is 0, the next command to execute is the Reset Vector.

The PCL Register (Program Counter Lower bits) is an 8-bit wide register in all four banks which allows read and write access to the 8 lower bits of the Program Counter. We can change this register instead of using a GOTO command to move execution.

If we add a number to the PCL using ADDWF, we can jump to a particular line after this one, based on what was in W. This is described in **Lecture PIC1 Slide 42** and is called a PCL Lookup Table.

# Exercises:

## Exercise 1 – Reading in and Writing out

In this first exercise we set up the lower four bits of PORTA to inputs mode, setup all of PORTB to output mode, then we read PORTA into the working register, add D'5' to it. We are only using the lower 4 LEDs on PORTB, so we will discard the 4 most significant bits, and then write them to PORTB.

After which we will enter SLEEP.

*Assume the input and LEDs are active high for this exercise.*

```
      list p=16f877A
      #include <p16f877A.inc>
      __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF

      ORG 0x00
RESET GOTO  START

      ORG 0x05
START <Instruction1>    ; Move to Bank 1
      <Instruction2>    ; Load W with 0x07
      <Instruction3>    ; Move W to ADCON1 to set PORTA into digital mode
      <Instruction4>    ; Load W with a value
      <Instruction5>    ; Set PORTA to input mode
      <Instruction6>    ; Set PORTB to output mode
      <Instruction7>    ; Move to Bank 0

      <Instruction8>    ; Copy PORTA to W
      <Instruction9>    ; Add 5 to W
      <Instruction10>   ; Discard the four most significant bits
      <Instruction11>   ; Write W to PORTB

      SLEEP
      GOTO  $-1

      END
```

## Exercise 2 – Greater than Check

In this exercise we set up all pins on PORTB to output mode, we then load W with a literal and subtract it from another literal. We find out if this resulted in a negative number by checking the STATUS register. If 10 > 5, turn on the LED at RB0.

After this we enter sleep.

See the Status register explanation above, and check the Instruction Set Summary from the data sheet to find a command to use for Instruction7.

*Assume the LEDs are active high for this exercise.*

```
      list p=16f877A
      #include <p16f877A.inc>
       __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF

            ORG 0x00
RESET       GOTO  START

            ORG 0x05
START       <Instruction1>    ; Turn off all LEDs attached to PORTB
            <Instruction2>    ; Move to Bank 1
            <Instruction3>    ; Set PORTB to output mode
            <Instruction4>    ; Move to Bank 0

            <Instruction5>    ; Load W with 10
            <Instruction6>    ; Subtract W from 5
            <Instruction7>    ; Skip next line if subtraction overflowed
            <Instruction8>    ; Jump to FINISH
            <Instruction9>    ; Turn on LED attached to RB0

FINISH      SLEEP
            GOTO  $-1

            END
```

**After you've finished:**

Experiment with values in Instruction 5 and Instruction 6 and see how the program behaves, observe the changes in STATUS using the MPLAB simulator.

## Exercise 3 – Flashing LED with delay

For this exercise we want this flash our LED on PORTB, pin RB0 every 10ms.

The PIC runs on a 20MHz oscillator, each instruction cycle taking 4 oscillations to fetch and execute an instruction cycle. Don't forget to check the data sheet for the operations that run for two instruction cycles.

You should work out a formula for the clock cycles using CountLow and CountHigh.

Find out how long it takes for Instruction1 to Instruction9. Next find how long it will take for CountLow to do it's first cycle and reach Instruction10. After that, CountLow will overflow, and you can use the same formula with 255. Find out how many cycles a single pass of CountHigh will take.

**TIP:** Use the Debugger->StopWatch to test your formula with small values.

```
      list p=16f877A
      #include <p16f877A.inc>
      __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF

; Define Memory Locations Names
CountHigh   EQU   0x70
CountLow    EQU   0x71

            ORG 0x00
RESET_V     GOTO  START

            ORG 0x05
START       <Instruction1>    ; Clear PORTB
            <Instruction2>    ; Move to Bank 1
            <Instruction3>    ; Set PORTB to output
            <Instruction4>    ; Move to Bank 0

MainLoop    <Instruction5>    ; Load W with decimal 1
            <Instruction6>    ; Toggle PORTB (hint bitwise operation)

            <Instruction7>    ; Load W with a value for CountLow
            MOVWF CountLow
            <Instruction8>    ; Load W with a value for CountHigh
            MOVWF CountHigh

Delay       <Instruction9>    ; Decrease CountLow, skip if zero
            GOTO  Delay
            <Instruction10>   ; Decrease CountHigh, skip if zero
            GOTO  Delay
            GOTO  MainLoop

            END
```

## Exercise 4 – PCL Lookup Tables

This exercise uses a Lookup Table to turn a number of LEDs on based on literal put into W in Instruction 5. The number is loaded into W then added to the PCL, Program Counter Lower bits register, effectively advancing execution by a few lines. Depending on what number was in W, the program will jump to one of the Cases 0-5, and load a number into W. Then that number will be written to PORTB.

Read the explanation of the PCL register, Lecture PIC1 Slide 42 and the PCL portion of the data sheet to fully understand lookup tables.

*Assume the LEDs are active high for this exercise.*

```
      list p=16f877A
      #include <p16f877A.inc>
      __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF

            ORG 0x00
RESET       GOTO  START

            ORG 0x05
START       <Instruction1>    ; Turn all PORTB LEDs off
            <Instruction2>    ; Move to Bank 1
            <Instruction3>    ; Set PORTB to output mode
            <Instruction4>    ; Move to Bank 0

            <Instruction5>    ; Move 3 into W register
            <Instruction6>    ; Put W+PCL in PCL
            <Instruction7>    ; Jump to Case0 when 0 is added to PCL
            <Instruction8>    ; Jump to Case1 when 1 is added to PCL
            <Instruction9>    ; Jump to Case2 when 2 is added to PCL
            <Instruction10>   ; Jump to Case3 when 3 is added to PCL

Case0       <Instruction11>   ; Move a number into W to light 0 LEDs
            GOTO  FINISH
Case1       <Instruction12>   ; Move a number into W to light 1 LED
            GOTO  FINISH
Case2       <Instruction13>   ; Move a number into W to light 2 LEDs
            GOTO  FINISH
Case3       <Instruction14>   ; Move a number into W to light 3 LEDs
            GOTO  FINISH

FINISH      <Instruction15>   ; Write the number in W to PORTB

            SLEEP
            GOTO  $-1

            END
```

## Exercise 5 – Subroutines

In this exercise we move the setup code we've been using for all of our exercises into a subroutine, which we call once at the beginning of our program. We also have a subroutine that adds PORTA to W and writes the result to PORTB.

Subroutines are easier to reuse throughout a program rather than using complicated GOTO jumps. But you only have a limited stack, which means if you keep using CALL within another CALL, the stack will overflow, and the line where CALL was first used will be lost and the RETURN command will not work correctly.

*Assume the input and LEDs are active high for this exercise.*

```
      list p=16f877A
      #include <p16f877A.inc>
      __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF

            ORG 0x00
RESET_V     GOTO  START

            ORG 0x05
START       <Instruction1>    ; Call the Setup Routine

            <Instruction2>    ; Put Decimal 5 in W
            <Instruction3>    ; Call Aplus

            <Instruction4>    ; Put Decimal 7 in W
            <Instruction5>    ; Call Aplus

            SLEEP
            GOTO  $-1

;SUBROUTINE: Setup
Setup       <Instruction6>    ; Goto bank1
            <Instruction7>    ; Put 0x07 in W
            <Instruction8>    ; Set PORTA to Digital I/O Mode
            <Instruction9>    ; Set all PORTA Pins to input
            <Instruction10>   ; Set all PORTA Pins to input
            <Instruction11>   ; Set all PORTB Pins to Output
            <Instruction12>   ; Goto Bank0
            <Instruction13>   ; Return to where CALL occurred

;SUBROUTINE: Aplus
Aplus       <Instruction14>   ; Add PORTA to W
            <Instruction15>   ; Write W to PORTB
            <Instruction16>   ; Return to where CALL occurred

            END
```