VANIER COLLEGE – Computer Engineering Technology – Autumn 2017

Introduction to Microprocessors (247-302)

Lab 3

Internal clock
&
Software delay loop

NOTE:
To be completed in ONE lab sessions of 3 hrs.
One report has to be submitted **not later than one week** after the last lab session.
This exercise is to be done **individually** except where specified in the procedure. **Each** student must submit a lab report with original design, observations and conclusions.

OBJECTIVES:

After completing this lab, the student will be able to:

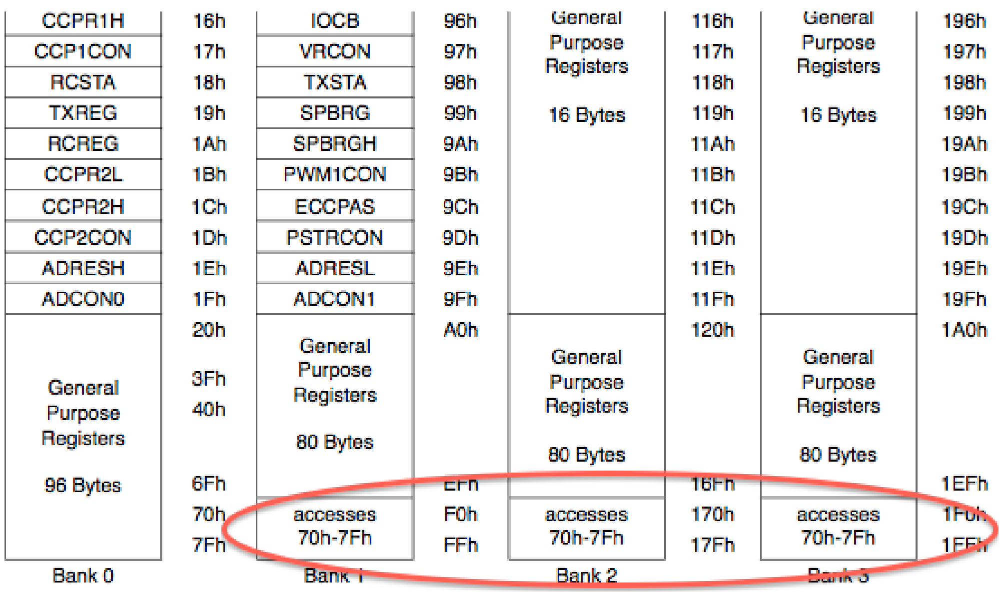
1. Utilize GPRs (general purpose registers) as variables in coding
2. Configure internal clock for different modes
3. Use software loops to create delays
4. Understand basic operation and syntax of PIC16F887's instructions

THEORY:

PIC16F887 General Purpose Registers

PIC16F887 register map and concept of banked register access has been described in previous lab.

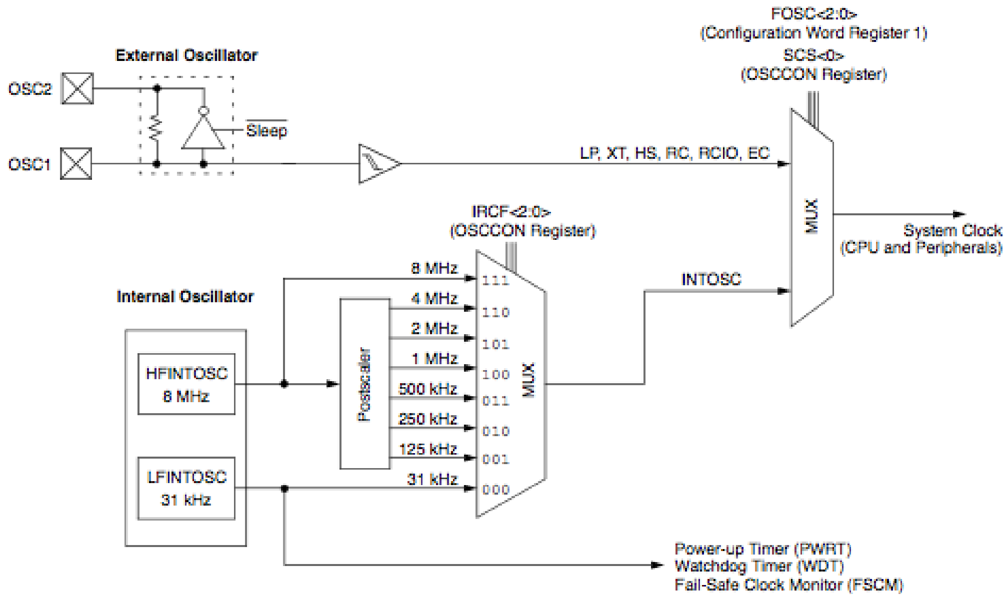
Each bank consists of 128 registers. The first upto 32 addresses in each register bank are used for special function registers (SFRs), while the remaining addresses in each bank are available for general-purpose registers (GPRs). Some of these GPR (70h-7Fh) are mapped into all four banks, meaning that they are shared, not banked.



GPRs can be used as variables for various arithmetic and logic operations. Their names and addresses in a program must be defined at the beginning of the program. It is not necessary to specify the address of continuous variable in the program. Instead, the code only required specifying the first one using directive CBLOCK and listing all others afterwards. The compiler automatically assigns these variables the corresponding addresses as per the order they are listed. Lastly, the directive ENDC indicates the end of the list of variables.

```
CBLOCK    0x20
          START    ; address 0x20
          RELE     ; address 0x21
          STOP     ; address 0x22
          LEFT     ; address 0x23
          RIGHT    ; address 0x24
ENDC
```

PIC16F887 internal clock sources



Internal oscillator consists of 2 sources:

a) **HFINTOSC** : a high-frequency internal oscillator which operates at 8MHz. The microcontroller can use clock source generated at that frequency or after being divided in prescaler. One of these seven frequencies can be selected via software using the IRCF2, IRCF1 and IRCF0 bits of the OSCCON register.

b) **LFINTOSC** : a low-frequency internal oscillator which operates at 31 kHz. Its clock sources are used for watch-dog and power-up timing but it can also be used as a clock source for the operation of the entire microcontroller. It is enabled by selecting this frequency (bits of the OSCCON register) and setting the SCS bit of the same register.

REGISTER 4-1: OSCCON: OSCILLATOR CONTROL REGISTER

U-0	R/W-1	R/W-1	R/W-0	R-1	R-0	R-0	R/W-0
—	IRCF2	IRCF1	IRCF0	OSTS(1)	HTS	LTS	SCS
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7	**Unimplemented:** Read as '0'
bit 6-4	**IRCF<2:0>:** Internal Oscillator Frequency Select bits 111 = 8 MHz 110 = 4 MHz (default) 101 = 2 MHz 100 = 1 MHz 011 = 500 kHz 010 = 250 kHz 001 = 125 kHz 000 = 31 kHz (LFINTOSC)
bit 3	**OSTS:** Oscillator Start-up Time-out Status bit(1) 1 = Device is running from the external clock defined by FOSC<2:0> of the CONFIG1 register 0 = Device is running from the internal oscillator (HFINTOSC or LFINTOSC)
bit 2	**HTS:** HFINTOSC Status bit (High Frequency – 8 MHz to 125 kHz) 1 = HFINTOSC is stable 0 = HFINTOSC is not stable
bit 1	**LTS:** LFINTOSC Stable bit (Low Frequency – 31 kHz) 1 = LFINTOSC is stable 0 = LFINTOSC is not stable
bit 0	**SCS:** System Clock Select bit 1 = Internal oscillator is used for system clock 0 = Clock source defined by FOSC<2:0> of the CONFIG1 register

Software delay loop

To generate software delay, we need to make the PIC "do nothing" for some amount of time, which is known as delay loops. A loop needs a loop counter: a variable which is incremented or decremented on every pass through the loop.

The following example declare one single byte variable, 'counter1' in GPR.

```
;************** DEFINING VARIABLES *****************************************
       cblock  0x20            ; Block of variables starts at address 20h
       counter1                ; Variable "counter1" at address 20h
       endc
;**************************************************************************
```

Here's an example of a basic delay loop :

```
       movlw   h'FF'           ; Number hFF is moved to W
       movwf   counter1        ; Number is moved to variable "counter1"

loop1
       decfsz  counter1        ; Variable "counter1" is decremented by 1
       goto    loop1           ; If result is 0, continue. If not,
                               ; remain in loop1
```

Instruction clock of PIC processor is ¼ of the processor clock rate. Example, if the processor is running at 4 MHz processor clock, the instruction clock runs at 1 MHz, or 1 µs per instruction. So the above sample code will generate a total delay of ((254 x 3) +2 + 2) µs.

In order to generate a longer delay loop, a loop can be wrap inside another loop, which is known as *nested loop.* Following is another basic sample of nested loop.

```
       movlw       h'FF'        ; Number hFF is moved to W
       movwf       counter2     ; Number is moved to variable "counter2"

loop2
       movlw       h'FF'        ; Number hFF is moved to W
       movwf       counter1     ; Number is moved to "counter1"
loop1
       decfsz      counter1     ; Decrements "counter1" by 1. If result is 0
       goto        loop1        ; skip next instruction

       decfsz      counter2     ; Decrements "counter2" by 1. If result is 0
       goto        loop2        ; skip next instruction
```

Lab procedures

PART A: Flashing LEDs

> ***Note*** : *This lab is based on your PIC16F887 prototype board built in lab 2. Make sure your circuit is correctly designed and connected before you proceed with this lab.*

1. Modify your circuit so that RA0, RA1, RA2, RA3 each drives an LED and an appropriate current limiting resister.
2. Start a new project in MPLABX and name the project *Lab3a*.
3. Setup your code with the following configuration settings:
   - To use the default internal clock, with clock out.
   - Disable watchdog timer.

   What are the configuration directive settings that correspond to the above requirements?
4. Write your code to flash all the 4 LEDs at the same time, at 50% duty cycle. Use a simple delay loop (loop for 255 times) to generate the delay between turning on and off the LEDs.
   a) Is that possible to generate a simple delay loop (without nested) of more than 255 times? If yes, how? If no, why?
5. Compile and download your code to your circuit.
   a) What is the frequency of the waveform observed at clock out pin? Is this what you are expecting?
   b) What is the LED flashing frequency generated by your code? Does this match the calculation of delay loop based on your code?
   c) Do your LEDs flash? If no, why?
6. Demonstrate your working board and software to your teacher.

PART B: Flashing LEDs at slower speed

7. Start a new project in MPLABX and name the project *Lab3b*.
8. Based on your code in part A, reduce the flashing speed of your LED by modify your code to select the slowest internal clock available.
   a) What are the settings needed to select the slowest internal clock?
   b) What would be the expected clock out frequency and LED flashing frequency? Show your calculation.
9. Compile and download your code to your circuit. Check if it works as expected.
10. Further reduce the flashing speed by around 2 times, using nested loop technique. Compile and download your code to the board. Check if it works as expected. Show your working board and software to your teacher.
11. Include a copy of your flowchart, all source codes, full detailed schematic, and answer all the question in your report. Also includes any observation, analysis if any.