Palestine Polytechnic University

College of Engineering and Technology

Department of Electrical Engineering

Class Notes for The Course

Microcontroller

Prepared by

Dr. Saleh Al-Takrouri

Based on the devices

PIC18F4550 and Arduino

Spring 2018

Course Outline

Course Name:	Microcontroller
Course Number:	5591
Credits:	3 Credit Hours
Semester:	Second Semester 2017\2018
Times and Locations:	Mon \Wed 09:30 to 10:45 B416
Instructor:	Dr. Saleh ALTAKROURI
Office:	B508
Office Hours: (provisional)	Sun 10:00 – 12:00 Mon 11:00 – 12:30 Tue 10:00 – 12:00 Wed 11:00 – 12:30 Thu 10:00 – 12:00
Textbook:	• PIC18F2455/2550/4455/4550 Data Sheet, Microchip Technology Inc, 2009.
Additional Materials:	• PIC Microcontroller and Embedded Systems Using Assembly and C for PIC18. M.A. Mazidi et al. 2008. • PIC Microcontroller: An Introduction to Software and Hardware Interfacing. H.W. Huang. 2005. • Arduino in Action, M. Evans, J. Noble and J. Hochenbaum, 2013. •Arduino: A Quick-Start Guide, Maik Schmidt, 2011.
Prerequisites:	4694 Digital System Design 5587 Digital Systems

Course Description:
This course introduces the student to the hardware and software of microcontrollers. The microcontrollers used in this course are the PIC18 and the Arduino Uno. The C language is used for programming. The student will study and learn how to utilize the various components included in each of these devices.

Course Topics:

Topic	**Hours**
Computer Architecture (Overview)	3
The PIC18 Microcontroller	3
Digital I/O, Delay function	7
ADC	4
Oscillator	3
Timer 0, Timer 2	5
CCP Module (PWM)	4
Interrupts	5
Arduino Microcontroller	8

Grading System:

First Exam:	25%
Second Exam:	25%
Final Exam:	40%
Quizes and Classwork:	10%

Contents

Chapter 1

Introduction to Microcontrollers

References:

Datasheet : 1.0 Device Overview.
5.0 Memory Organization.

Mazidi et al. : Chapter 0 Introduction to Computing.
Chapter 1 The PIC Microcontrollers History and Features.

Huang : Chapter 1 Introduction to the PIC18 Microcontroller.

1.1 Computer Architecture

- **Digital Computer:** a programmable machine (made up of hardware and software) that process binary data. The hardware of the computer consists of: CPU, memory, inputs, outputs.
- **Central Processing Unit (CPU):** the group of circuits that processes data and provides control signals and timing. The CPU consists of at least: ALU, CU, registers.
  - **Arithmetic/Logic Unit (ALU):** the group of circuits that performs arithmetic and logic operations.
  - **Control Unit (CU):** the group of circuits that decodes the instructions, provides timing and control signals to all operations in the computer, and controls data flow.
  - **Register:** A storage location inside the CPU used to hold data and/or a memory address during the execution of an instruction.
- **Memory:** a medium that stores binary information. It consists of a group of registers arranged in sequence to store data.
  - Read Only Memory (ROM): a memory that stores binary information permanently.

  - Random Access Memory (RAM): a memory that stores binary information during the operation of the computer.

- **Input**: a device that transfers information from outside world to the computer.
- **Output**: a device that transfers information from the computer to outside world.

Microprocessor:
a semiconductor device that includes the ALU, CU, and register arrays on a single chip.

Microcontroller:
a device that includes a microprocessor, memory, and I/O signal lines on a single chip. It may also contain:

- Analog to Digital converters.
- Digital to Analog converters.
- Timers.
- Pulse Width Modulator (PWM).

Binary Numbers:

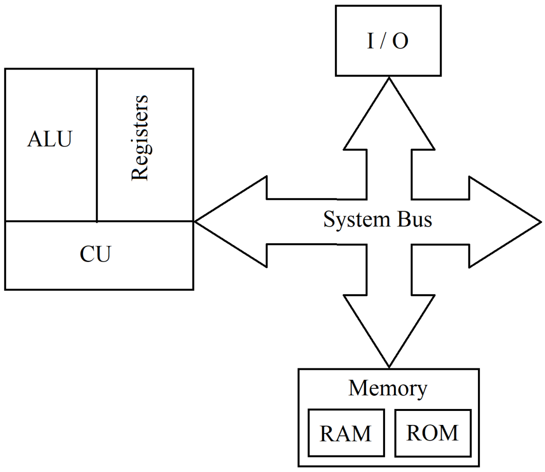
- Bit: a binary digit, 0 or 1.
- Byte: a group of eight bits.
- Word: a group of bits the computer recognizes and processes at a time.

Example: A 16-bit microprocessor has a word length of 2 bytes.

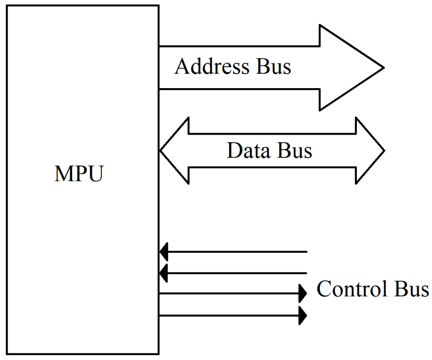
Bus:
a group of lines used to transfer bits between the microprocessor and other components of the computer system.

- **Address Bus:** a unidirectional bus used to send a memory address or a device address from the microprocessor to the memory or the peripheral.
- **Data Bas:** a bidirectional bus used to transfer data between the microprocessor and peripheral or memory.

- **Control Bus:** signal lines that are generated by the microprocessor to provide timing for various operations.

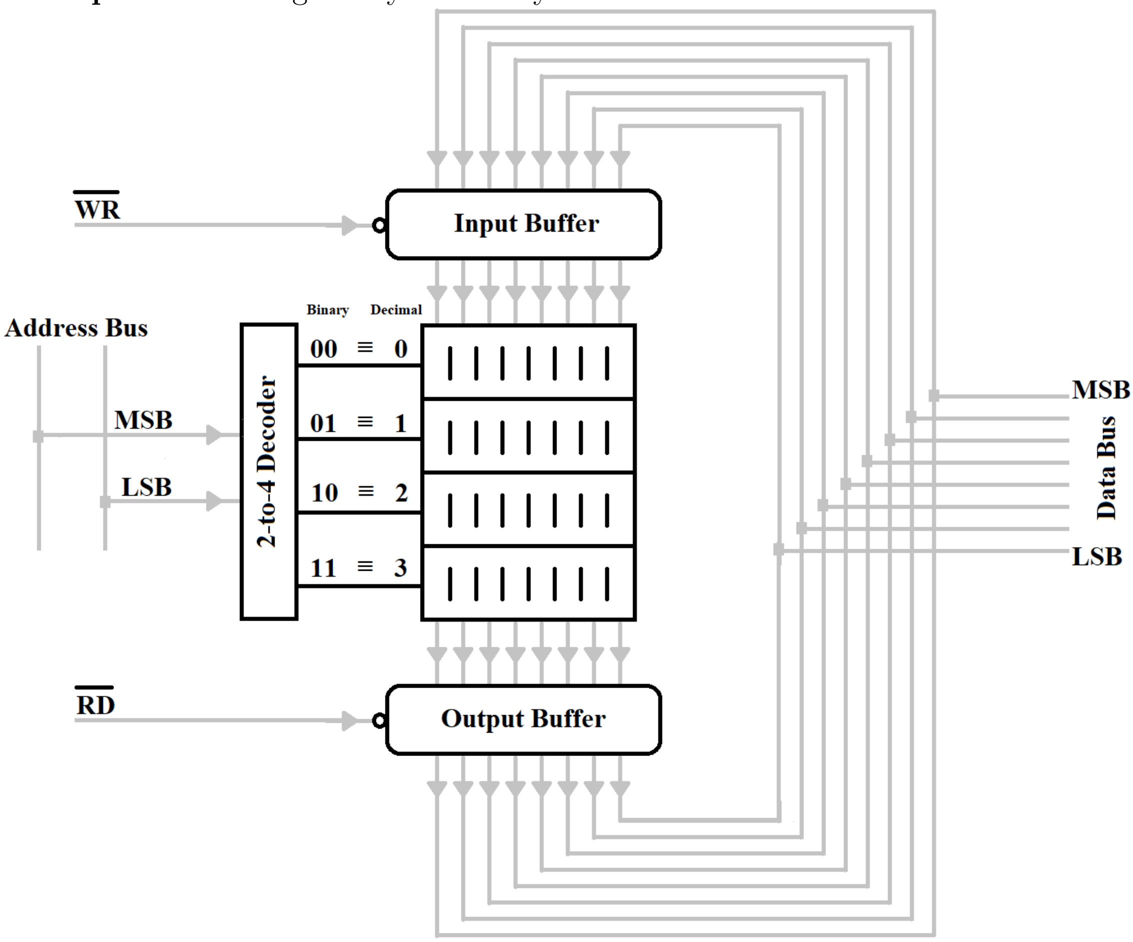


Microprocessor-based system with bus architecture



Microprocessing Unit (MPU) buses

Example: Interfacing a 4-byte memory.



Software:

- **Instruction:** a command in binary that is recognized and executed by the computer to accomplish a task. It can be designed with one or multiple words.
- **Machine Language:** the binary medium of communication with a computer through a designed set of instructions specific to each computer.
- **Mnemonic:** a combination of letters to suggest the operation of an instruction.
- **Assembly Language:** a medium of communication with a computer in which programs are written in mnemonics, specific to a given computer.

Example (8085 MPU):

Machine Language:	0 0 1 1 1 1 0 0 ($3C_H$)	1 0 0 0 0 0 0 0 (80_H)
Assembly Language:	`INR A`	`ADD B`
Instruction:	increment the contents of register A by 1	add the contents of register B to the contents of register A

- Machine language and assembly language are low-level languages.
- **Assembler:** a computer program that translate an assembly language program from mnemonics to the binary machine code of a computer.
- **High-Level Language:** programs are written in English-like words and can be executed using a compiler or interpreter.
- **Compiler:** a program that reads a given program written in English-like words (source code) *in its entirety* and then translates the program into machine language (object code).
- **Interpreter:** a program that translates and executes one statement at a time from a source code.

1.2 PIC18F4550 Microcontroller

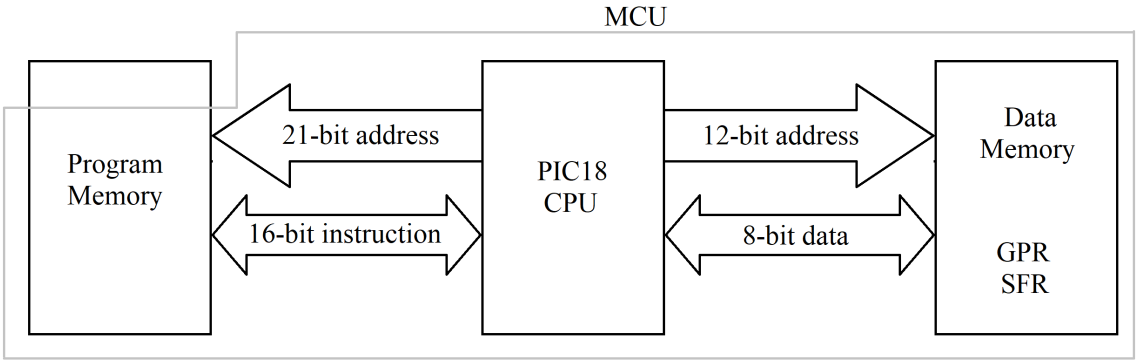
- **PIC:** Peripheral Interface Controller.
- Made by Microchip Technology Inc.
- 8-bit device.

PIC18F4550 Main Features:

- C Compiler Optimized Architecture.
- Pin count: 40 pins.
- I/O ports: 34 pins (A, B, C, D, E).
- Operating frequency: up to 48 MHz.
- Program memory: 32 kB (16348 Instructions).
- Data memory: 2 kB.
- Interrupts: 20 sources, 2 priority levels.
- Instructions: 83.
- Modules: ADC, timers, CCP, serial and parallel ports, USB, comparators.

1.3 PIC18F4550 Memory Organization

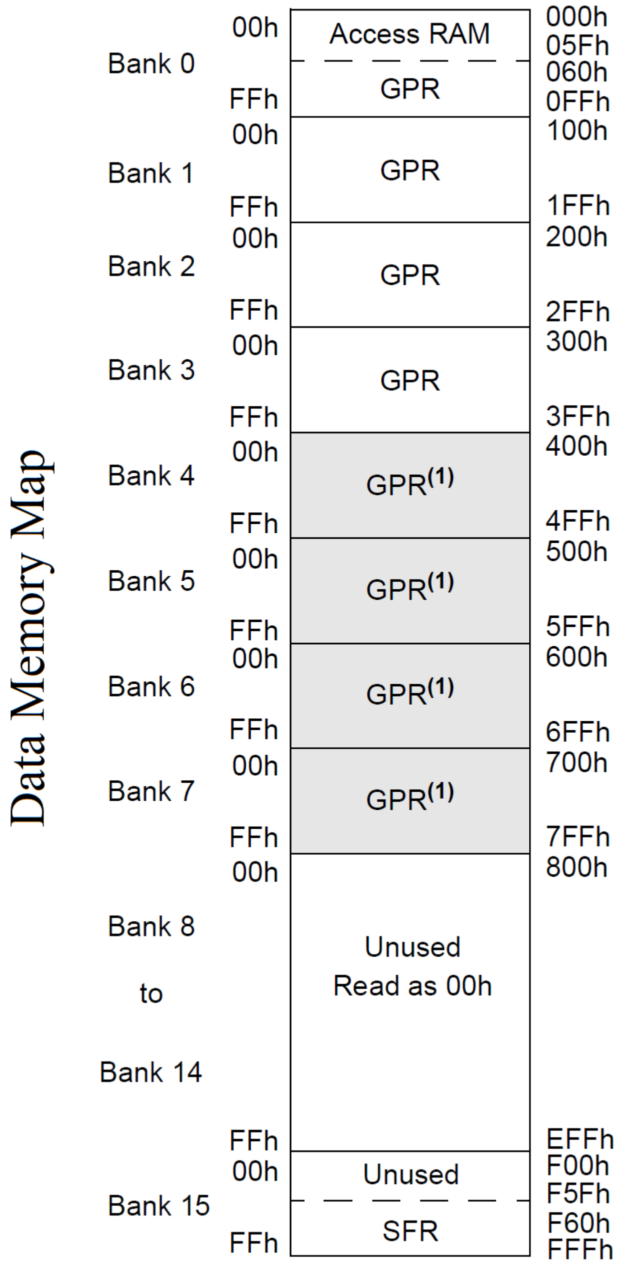
- PIC18 assigns data and program to different memory spaces and provides separate buses to them so that both are available for access at the same time.
- Data memory: 12-bit register address, 8-bit data.
  Note: $2^{12} = 4096 = 4$ kB not all used.
  Note: PIC18 is 8-bit microcontroller.
- Program memory: 21-bit program address, 16-bit instruction.
  Note: $2^{21} = 2$ MB not all included.
  Note: 1 instruction = 2 bytes.
- Data EEPROM (not used in this course).



The PIC18 Memory Spaces

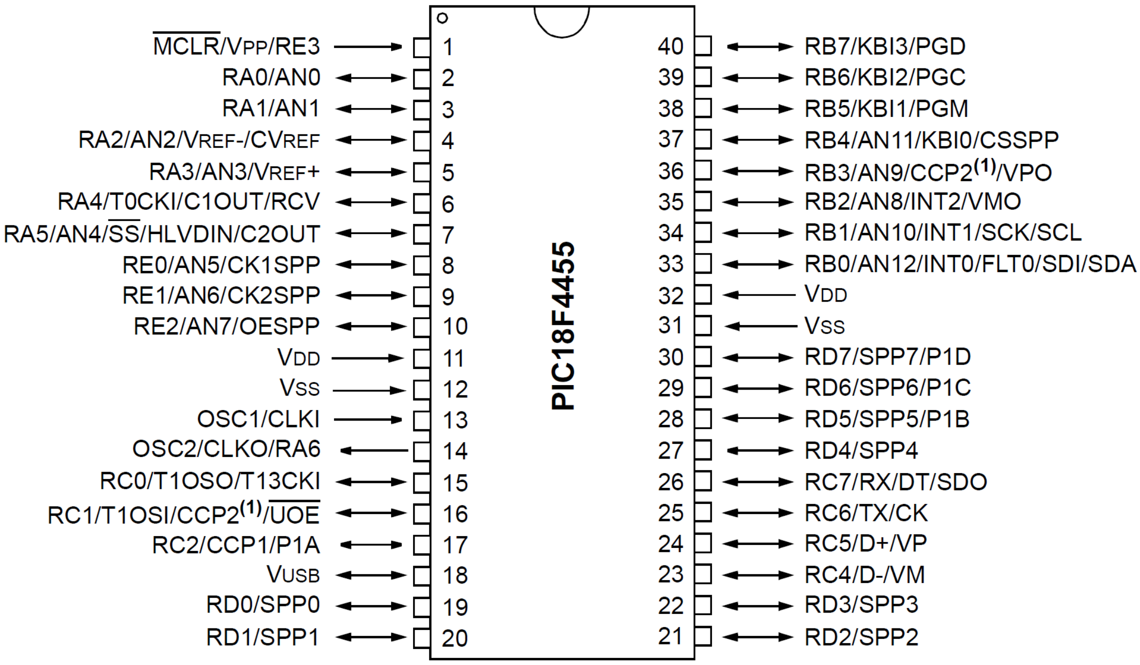
Data Memory:

- Each location is referred to as ”register” or ”file register”.
- 12-bit register address $\Rightarrow$ up to $2^{12} = 4096$ bytes $(000_H - FFF_H)$.
- Data memory space is divided into 16 banks (0 to 15), each bank contains 256 bytes.
- Bank 8 to bank 14 are not used.
- Data memory contains:
  - **General Purpose Registers (GPR):** a group of RAM locations in the data memory used to store data. Addressing starts at 000_H and increases until $7FF_H$.
  - **Special Function Registers (SFR):** a group of RAM locations in the data memory dedicated to specific functions such as ALU status, timers, I/O ports, etc. Addressing starts at FFF_H and decreases until $F60_H$ (160 registers).



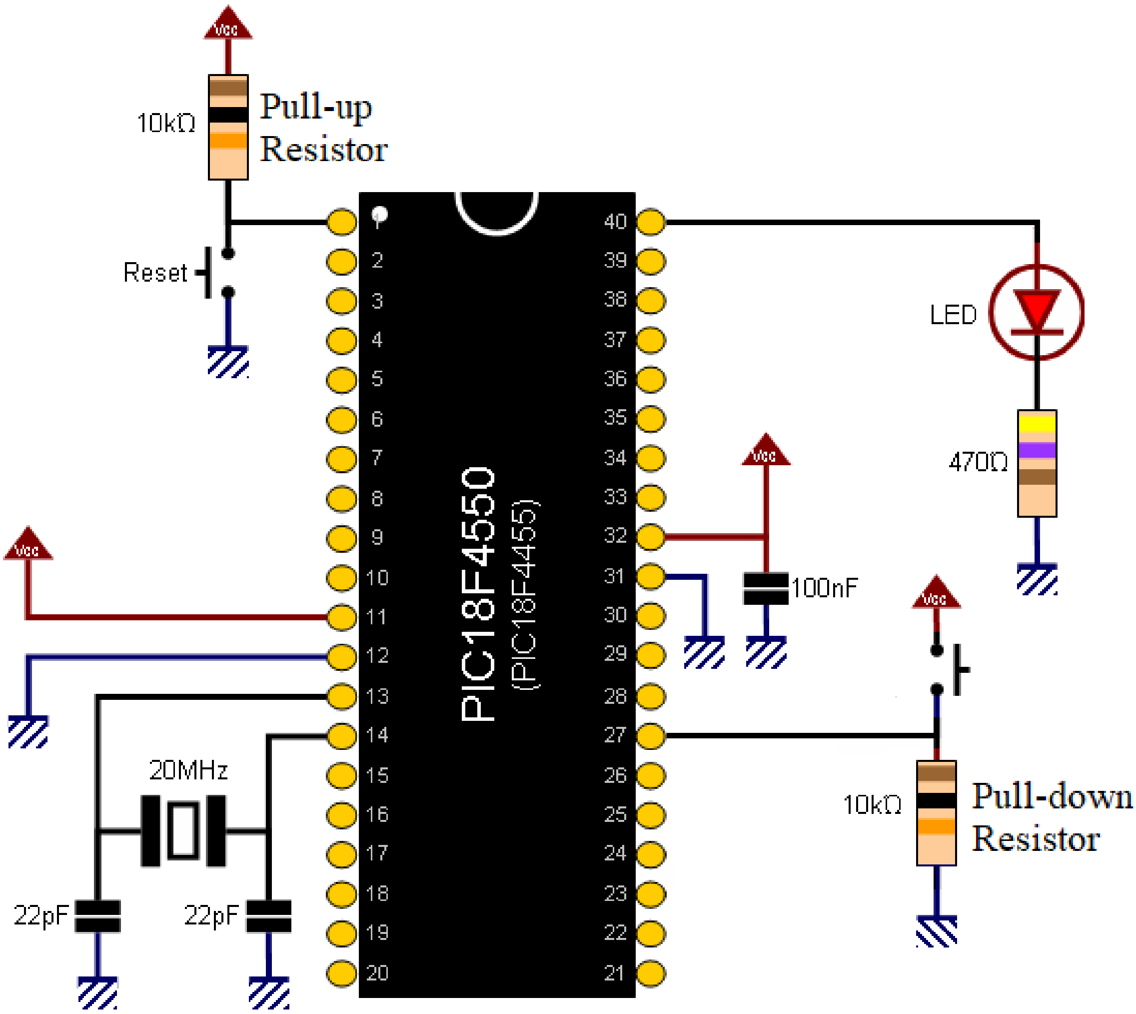
Note: To access specific bits in a SFR register, write "register name" then "bits." then "bit name" (e.g. `PORTAbits.RA2`, `ADCON0bits.GO`).

1.4 PIC18F4550 Pins



- Pins 11 and 32 (V_{DD}) are connected to the power supply (+5 V DC).
- Pins 12 and 31 (V_{SS}) are connected to the ground (0 V DC).
- Pin 1 ($\overline{MCLR}$) is connected to +5 V using pull-up resistor.

Pull-up and pull-down resistors:



- PIC18F4550 has five I/O ports (34 pins).
- Pin name format `RX#` where `X` = `A` – `E`, # = `0` – `7`.
- Each I/O pin is multiplexed with up to six functions.
- Example: Pin 35:
  1. `RB2`: Digital output port B bit 2.
  2. `RB2`: Digital input port B bit 2.
  3. `AN8`: A/D input channel 8.
  4. `INT2`: External Interrupt 2 input.
  5. `VMO`: External USB transceiver VMO data output.
- In general, when a peripheral is enabled, the associated pins may not be used as general purpose I/O pins.

Chapter 2

Digital Input/Output

References:

Datasheet : 10.0 I/O Ports.

Mazidi et al. : Chapter 7 PIC Programing in C.

Huang : Chapter 7 Parallel Ports.

2.1 Ports

- The PIC18F4550 has 5 digital input/output ports:
  1. Port A (7 pins): __, 14, 7, 6, 5, 4, 3 , 2.
  2. Port B (8 pins): 40, 39, 38, 37, 36, 35, 34, 33.
  3. Port C (7 pins): 26, 25, 24, 23, __, 17, 16, 15.
  4. Port D (8 pins): 30, 29, 28, 27, 22, 21, 20, 19.
  5. Port E (4 pins): __, __, __, __, 1, 10, 9, 8.
- Each port has three registers for its operation:
  1. `TRIS` register: data direction register (`0`=output, `1`=input).
  2. `PORT` register: reads the levels on the pins of the device (preferred for input).
  3. `LAT` register: output latch (preferred for output).
- Unimplemented bits are read as `0`.

Port A

Register **PORTA**

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	–	`RA6`	`RA5`	`RA4`	`RA3`	`RA2`	`RA1`	`RA0`

- To use `RA0`, `RA1`, `RA2`, `RA3`, `RA5` the analog inputs should be disabled (`ADCON1 = 0x0F`).
- To use `RA2`, `RA4`, `RA5` the comparator module should be disabled (`CMCON = 0x07`).
- To use `RA6` the oscillator should be configured properly.
- On a Power-on Reset, RA5 and RA3:RA0 are configured as analog inputs; RA4 is configured as a digital input; comparator disabled.

Register **TRISA**

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	–	`TRISA6`	`TRISA5`	`TRISA4`	`TRISA3`	`TRISA2`	`TRISA1`	`TRISA0`

Register **LATA**

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	–	`LATA6`	`LATA5`	`LATA4`	`LATA3`	`LATA2`	`LATA1`	`LATA0`

Port B

Register **PORTB**

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	`RB7`	`RB6`	`RB5`	`RB4`	`RB3`	`RB2`	`RB1`	`RB0`

- To use `RB0`, `RB1`, `RB2`, `RB3`, `RB4` the analog inputs should be disabled (`ADCON1 = 0x0F`).
- On a Power-on Reset, RB4:RB0 are configured as analog inputs; RB7:RB5 are configured as digital inputs.

Register **TRISB**

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	`TRISB7`	`TRISB6`	`TRISB5`	`TRISB4`	`TRISB3`	`TRISB2`	`TRISB1`	`TRISB0`

Register **LATB**

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	`LATB7`	`LATB6`	`LATB5`	`LATB4`	`LATB3`	`LATB2`	`LATB1`	`LATB0`

Port C

Register PORTC

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	RC7	RC6	RC5	RC4	—	RC2	RC1	RC0

- RC4, RC5 can be used as digital inputs only (not outputs).

Register TRISC

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	TRISC7	TRISC6	—	—	—	TRISC2	TRISC1	TRISC0

RegisterLATC

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	LATC7	LATC6	—	—	—	LATC2	LATC1	LATC0

Port D

Register PORTD

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0

Register TRISD

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0

Register LATD

Bit:	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Name:	LATD7	LATD6	LATD5	LATD4	LATD3	LATD2	LATD1	LATD0

2.2 First Program

Example: Write a program to read the value on port B and display this value on port D.

```
#include <xc.h>
void main() {
   TRISB = 0xFF;
   TRISD = 0x00;
   ADCON1 = 0x0F;
   while(1) {
      LATD = PORTB;
   }
}
```

- Eight switches are connected to the pins associated with Port B.
- Eight LEDs are connected to the pins associated with Port D.
- `xc.h` is the header file for the MPLAB XC8 compiler.
- `main()` function has two parts: setup and loop.
- The setup section:
  - All the pins of port B are defined as inputs.
  - All the pins of port D are defined as outputs.
  - The analog inputs on port B are turned off.
- The infinite loop section:
  - The binary value on port B is read.
  - The binary value is written to port D.
  - The The read/write process continues infinitely.

Review of Logic Operations:

	x =	0x0F	0 0 0 0 1 1 1 1
	y =	0xAA	1 0 1 0 1 0 1 0
AND	x && y =	1	0 0 0 0 0 0 0 1
Bitwise AND	x & y =	0x0A	0 0 0 0 1 0 1 0
OR	x || y =	1	0 0 0 0 0 0 0 1
Bitwise OR	x | y =	0xAF	1 0 1 0 1 1 1 1
NOT	!x =	0	0 0 0 0 0 0 0 0
Bitwise NOT	~x =	0xF0	1 1 1 1 0 0 0 0
Bitwise XOR	x^y =	0xA5	1 0 1 0 0 1 0 1
Rotate Right	x >> 3 =	0x01	0 0 0 0 0 0 0 1
Rotate Left	x << 3 =	0x78	0 1 1 1 1 0 0 0

2.3 The Delay Function

Example: Write a program to continuously count from **0x00** to **0xFF** and display the output on port D. Wait 3 seconds between counts.
(Assume the oscillator frequency $F_{OSC} = 20$ MHz)

```
#include <xc.h>
#define _XTAL_FREQ 20000000

void main() {
   TRISD = 0x00;
   LATD = 0x00;
```

```
   while(1) {
      LATD = PORTD + 1;
      __delay_ms(3000);
   }
}
```

Note: The time delay that can be provided by the function `__delay_ms()` is limited by the value of the microcontroller clock frequency F_{OSC}.
(Examples: 48 MHz → 4205 ms, 20 MHz → 10092 ms, 1 MHz → 201852 ms)

Example: Write a program to send the ASCII codes for the characters 0, 1, 2, A, B and C to port D. Wait 6 seconds between counts. (Assume F_{OSC} = 48 MHz)

```
#include <xc.h>
#define _XTAL_FREQ 48000000

void main() {
   unsigned char mycode[] = "012ABC";
   unsigned char k,x;
   TRISD = 0x00;
   while(1) {
      for(k=0; k<6; k++) {
         LATD = mycode[k];
         for(x=0; x<3; x++) __delay_ms(2000);
      }
   }
}
```

Example: Find the output of the following program.

```
#include <xc.h>
#define _XTAL_FREQ 20000000

void main() {
   unsigned char x;
   TRISC = 0;
   TRISD = 0;
   LATC = 0;
   LATD = 0;
   for(;;) {
      LATC = PORTC + 1;
      LATD = PORTD + 1;
      for(x=0; x<5; x++) __delay_ms(3000);
   }
}
```

- The count on port D will be from `0` to `255`.
- The count on port C will be from `0` to `7` (`RC3` is not implemented).
- The time delay between counts is 15 seconds.

Problem: Write a program to toggle all the bits of port B continuously. Wait 15 seconds between toggles and assume the oscillator frequency is 24 MHz. (`01010101` $\Longleftrightarrow$ `10101010`)

Problem: Write an XC8 program to perform the following tasks:

- The program continuously counts from 1 to 7 and displays the number on `Port A`.
- Each number is blinked according to its value (`001` is blinked once, `010` is blinked twice, `011` is blinked three times, ...).
- A blink is 1 second on then 1 second off.
- Assume the oscillator frequency F_{OSC} = 1 MHz.

2.4 Addressing I/O Bits

Example: A push-button and a LED are connected to pins 20 and 22 respectively. Write a program such that the LED is on only when the push-button is pressed down.

```
#include <xc.h>
void main() {
   TRISDbits.TRISD1 = 1;    // pin 20 = RD1 input
   TRISDbits.TRISD3 = 0;    // pin 22 = RD3 output
   while(1)
      LATDbits.LATD3 = PORTDbits.RD1;
}
```

Example: Write a program to read the value on `Port D`. If the value is zero, then turn on the LED on pin `RA2`. Otherwise, turn on the LED on pin `RA5`.

```
#include <xc.h>
void main() {
   TRISD = 0xFF;
   TRISAbits.TRISA2 = 0;
   TRISAbits.TRISD5 = 0;
   ADCON1 = 0x0F;
   COMCON = 0x07;
```

```
   while(1) {
      if(PORTD==0) {
         LATAbits.LATA2 = 1;
         LATAbits.LATA5 = 0;
      }
      else {
         LATAbits.LATA2 = 0;
         LATAbits.LATA5 = 1;
      }
   }
}
```

Problem: Write a program to continuously read the value on `Port B`. If the value is less than or equal to 100, then turn on 2 LEDs on `Port C`. If the value is more than 100 and less than 200, then turn on 4 LEDs on `Port C`. Otherwise, turn on 5 LEDs on `Port C`.

2.5 Masking

Masking is used to change the values of specific bits in a register or variable without affecting the other bits.
To reset a bit value to `0`, use the bitwise AND operation.
To set a bit value to `1`, use the bitwise OR operation.

Example: Let the value of port B to be `0x57`. Reset the bits RB2 and RB4.

Bit number	7 6 5 4	3 2 1 0	
PORTB	0 1 0 1	0 1 1 1	0x57
Mask	1 1 1 0	1 0 1 1	0xEB
LATB = PORTB & 0xEB	0 1 0 0	0 0 1 1	0x43

Example: Let the value of port D to be `0xC1`. Set the bits RD3 and RD5.

Bit number	7 6 5 4	3 2 1 0	
PORTD	1 1 0 0	0 0 0 1	0xC1
Mask	0 0 1 0	1 0 0 0	0x28
LATD = PORTD | 0x28	1 1 1 0	1 0 0 1	0xE9

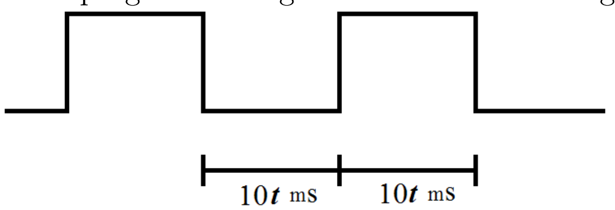
Example: Write a program to add or multiply two four-bit hexadecimal digits. Use port B for input, port D for output and RC0 for selecting the operation (0: add, 1: multiply).

```
#include <xc.h>
#define OP PORTCbits.RC0

void main() {
   unsigned char N1, N2;
```

```
    TRISB = 0xFF;
    TRISD = 0;
    TRISC = 1;
    ADCON1 = 0x0F;
    while(1) {
        N1 = PORTB & 0x0F;
        N2 = PORTB & 0xF0;
        N2 = N2 >> 4;
        if (OP==0)
            LATD = N1 + N2;
        else
            LATD = N1 * N2;
    }
}
```

Problem: Write a program to continuously read three 4-bit numbers and write a 3-bit result. If any input number is larger than 8, then the output is 0. Otherwise the output is the maximum input number. Only use ports B and D.

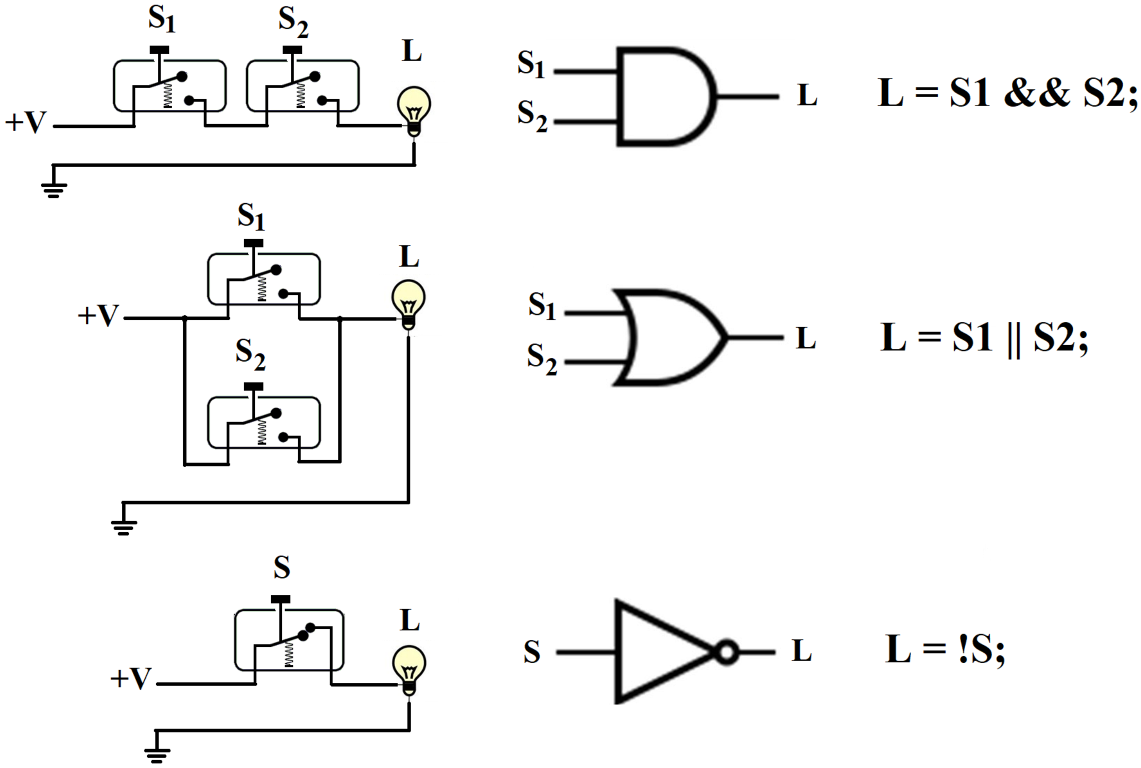
Problem: Write a program to generate the following signal on `RB1`.



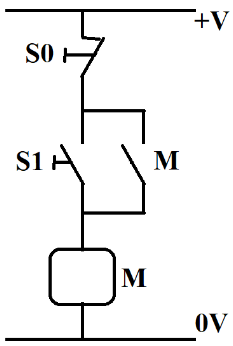
Where ***t*** is a 4-bit number entered by the user using pins `RB2` (least significant) to `RB5` (most significant).

2.6 Electrical Control Circuits

Logic gates and equivalent electrical switching circuits:



Example: Write a program to operate a motor using start/stop push-buttons. The traditional control circuit for the motor using relays is as shown in the figure below.



Inputs/Outputs:

- Stop push-button (`S0`) connected to pin `RD0` (input).
- Start push-button (`S1`) connected to pin `RD1` (input).
- Relay operating the motor (`M`) connected to pin `RD7` (output).

```
#include <xc.h>
#define S0 PORTDbits.RD0
```

```
#define S1 PORTDbits.RD1
#define M LATDbits.LATD7

void main() {
   TRISD = 0x03;
   M = 0;
   while(1)
      M = !S0 && (S1 || M);
}
```

Problem: A machine has three motors and three push-buttons. The machine operates as follows:

- Two push-buttons (S1, S0) are used to start/stop the motor (M1).
- Motor (M2) is on only when the motor (M1) is on and the push-button (S2) is pressed down.
- Motor (M3) is on only when the motor (M1) is on and the motor (M2) is off.

Draw the relay control circuit to control this machine, then write the XC8 program for this machine.

Example: A PIC18 based machine has two motors and operated as follows:

- Each motor has a start push-button.
- One stop push-button turns off the motors.
- The second motor can be turned on only when the first motor is on.
- When the second motor is turned on, the first motor is turned off.

Write a program to control the machine. Use K-maps in your solution.

S0	M2	S1	M1	M1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	**X**	**X**	**X**	**0**

S0	S2	M1	M2	M2
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	**X**	**X**	**X**	**0**

S0M2\S1M1				
	0	1	1	1
	0	0	0	0
	0	0	0	0
	0	0	0	0

$M1 = \bar{S0}\ \bar{M1}\ (S1 + M1)$

S0S2\M1M2				
	0	1	1	0
	0	1	1	1
	0	0	0	0
	0	0	0	0

$M2 = \bar{S0}\ (S2\ M1 + M2)$

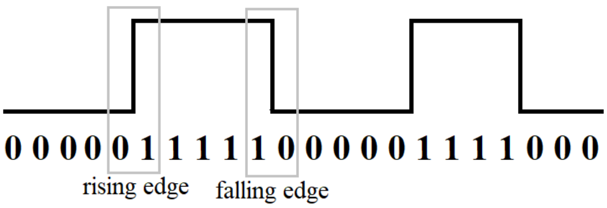
```
#include <xc.h>
#define S0 PORTDbits.RD0
#define S1 PORTDbits.RD1
#define S2 PORTDbits.RD2
#define M1 LATDbits.LATD6
#define M2 LATDbits.LATD7

void main() {
   TRISD = 0b00000111;
   M1 = 0;
   M2 = 0;
   while(1) {
      M1 = (!S0) && (!M2) && (S1 || M1);
      M2 = (!S0) && ((S2 && M1) || M2);
   }
}
```

Problem: A machine has two motors that operate as follows:

- Each motor has a start push-button and a stop push-button.
- Motor 2 can be turned on only while Motor 1 is running.
- If Motor 1 is off, Motor 2 turns off too.

Write an XC8 code to operate the machine. You must use a K-maps in your solution.

Edge Detection



Example: Write a program to use one push-button to turn on/off a LED. Use edge detection in your solution.

```
#include <xc.h>
```

```
#define S PORTDbits.RD0
#define L PORTDbits.RD1

void main() {
   unsigned char S_old;
   TRISDbits.TRISD0 = 1;
   TRISDbits.TRISD1 = 0;
   S_old = 0;
   while(1) {
      if(!S_old && S) L = !L;
      S_old = S;
   }
}
```

Program Blocking

In some cases, the execution of the program is temporarily blocked while a specific condition is satisfied. Once the blocking condition is no longer valid, the execution of the program continues.

Example: Write a program to use one push-button to turn on/off a LED. Use program blocking in your solution.

```
#include <xc.h>
#define S PORTDbits.RD0
#define L PORTDbits.RD1
void main() {
   TRISDbits.TRISD0 = 1;
   TRISDbits.TRISD1 = 0;
   while(1) {
      if (S) {
         L = !L;
         while(S); // Program is blocked until S is released.
      }
   }
}
```

SET / RESET Programming

Priority for Reset:

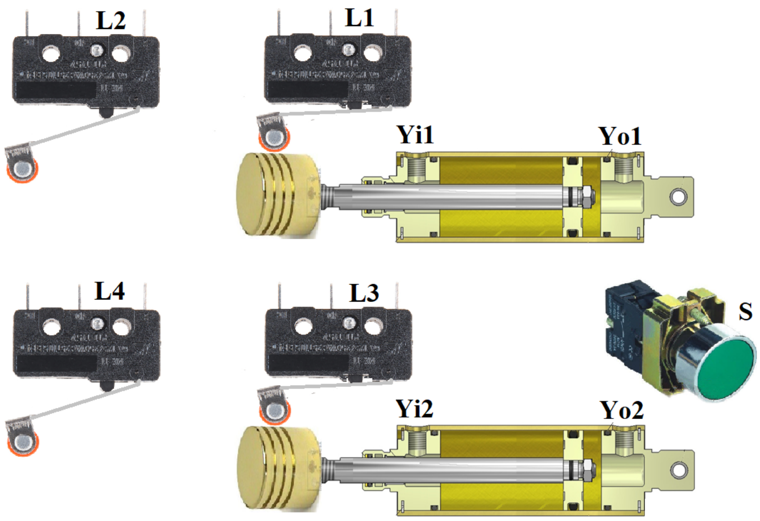
```
if (Reset condition) var = 0;
else if (Set condition) var = 1;
```

Priority for Set:

```
if (Set condition) var = 1;
else if (Reset condition) var = 0;
```

Example: A machine consists of two double-acting cylinders, four limit switches and one push-button. The machine is operated as follows:

- The machine is turned on/off using the push-button (S).
- Each cycle begins with the cylinders at L1 and L3.
- Yo1 is turned on until the first cylinder reaches L2 then it stops.
- Yo2 is turned on until the second cylinder reaches L4 then it stops.
- Yi2 is turned on until the second cylinder returns to L3 then it stops.
- Yi1 is turned on until the first cylinder returns to L1 then it stops, then a new cycle begins.
- When the machine is turned off, both cylinders are returned to L1 and L3.



	SET/ON	RESET/OFF
Yo1	M · L1 · L3	L2 + $\overline{M}$
Yo2	M · L2 · L3 · x	L4 + $\overline{M}$
Yi2	M · L2 · L4	L3 + $\overline{M}$
Yi1	M · L2 · L3 · $\overline{x}$	L1 + $\overline{M}$
x	Yo2	Yi1

```
#include <xc.h>
#define Yo1 LATDbits.LATD0
#define Yo2 LATDbits.LATD1
#define Yi1 LATDbits.LATD2
#define Yi2 LATDbits.LATD3
#define L1 PORTDbits.RD4
#define L2 PORTDbits.RD5
```

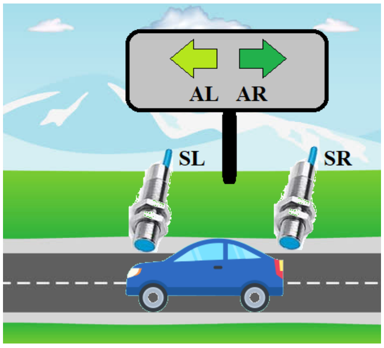
```
#define L3 PORTDbits.RD6
#define L4 PORTDbits.RD7
#define S PORTCbits.RC0

void main() {
   unsigned char M, x, S_old;
   TRISD = 0xF0;
   TRISCbits.TRISC0 = 1;
   LATD = 0;
   M = 0;
   x = 0;
   S_old = 0;
   while(1) {
      if(!S_old && S) M=!M;

      if (L2 || !M) Yo1 = 0;
      else if(M && L1 && L3) Yo1 = 1;

      if (L4 || !M) Yo2 = 0;
      else if(M && L2 && L3 && !x) Yo2 = 1;

      if (L3 || !M) Yi2 = 0;
      else if(M && L2 && L4) Yi2 = 1;

      if (L1 || !M) Yi1 = 0;
      else if(M && L2 && L3 && x) Yi1 = 1;

      if (Yi1) x = 0;
      else if(Yo2) x = 1;

      if (!M && (!L1 || !L3)) {
         while(!L1) Yi1=1; Yi1=0;
         while(!L3) Yi2=1; Yi2=0;
      }

      S_old = S;
   }
}
```

Problem: A security system consists of a smoke detector (SS), a motion sensor (MS), one speaker and a control panel. The control panel has one light and six push-buttons.

- The light (H) is on while the system is on.
- The speaker (A) is on while there is an alarm.
- A push-button (S1) turns on the system and a push-button (S0) turns off the system.
- A push-button (M1) is used to enable motion sensing, a push-button (M0) is used to disable motion sensing, and a push-button (MR) is used to reset the alarm after motion is detected.
- The smoke detector turns on the alarm when there is smoke and stops the alarm when the smoke is removed or the system is turned off.
- A push-button (T) is used to test the speaker.

Write an XC8 code for this system.

Problem: A traffic detection machine consists of two sensors and two arrow-shaped lights. If a car passes from right to left, the left-arrow light is on until the car is away from the sensors. If a car passes from left to right, the right-arrow light is on until the car is away from the sensors. Write the XC8 code for this machine.
(a) Use the Set/Reset method with Edge Detection in the conditions.
(b) Use the Program Blocking method.



Chapter 3

Analog-to-Digital Converter Module

References:

Datasheet : 21.0 10-Bit Analog-to-Digital Converter (A/D) Module.

Mazidi et al. : Chapter 13 ADC, DAC and Sensor Interfacing.

Huang : Chapter 12 Analog-to-Digital Converter.

3.1 ADC Module

- ADC module has 13 inputs to convert analog input signals to a 10-bit digital number.
- 10-bit $\rightarrow 2^{10}$ values $= 1024$ values.
- Input range: $\mathrm{V}_{\mathrm{REF}(-)}$ to $\mathrm{V}_{\mathrm{REF}(+)}$.
  - $\mathrm{V}_{\mathrm{REF}(-)} \equiv$ **`0x000`** $= 0$
  - $\mathrm{V}_{\mathrm{REF}(+)} \equiv$ **`0x3FF`** $= 2^{10} - 1 = 1023$
- V: input voltage in volts $\equiv N$: digital number

$$N = 1023\,\frac{V - \mathrm{V}_{\mathrm{REF}(-)}}{\mathrm{V}_{\mathrm{REF}(+)} - \mathrm{V}_{\mathrm{REF}(-)}} \qquad\qquad V = N\,\frac{\mathrm{V}_{\mathrm{REF}(+)} - \mathrm{V}_{\mathrm{REF}(-)}}{1023} + \mathrm{V}_{\mathrm{REF}(-)}$$

- Resolution $= \dfrac{\mathrm{V}_{\mathrm{REF}(+)} - \mathrm{V}_{\mathrm{REF}(-)}}{1023}$

Example: Let $V_{REF(-)} = 1$ V and $V_{REF(+)} = 4$ V.

N =	0	≡	V =	1.00 V
N =	25	≡	V =	1.07 V
N =	500	≡	V =	2.47 V
N =	900	≡	V =	3.64 V
N =	1023	≡	V =	4.00 V

The ADC module has five registers:

- `ADRESH`: A/D Result High Register.
- `ADRESL`: A/D Result Low Register.
- `ADCON0`: A/D Control Register 0.
- `ADCON1`: A/D Control Register 1.
- `ADCON2`: A/D Control Register 2.

Acquisition Time and Conversion Time:

- The analog to digital conversion is performed in two steps: sampling and conversion.
- **Acquisition Time (T_{ACQ}):** is the time needed to sample the analog input. For simplicity assume $T_{ACQ} \geq 2.5\mu s$.
- **A/D Conversion Time (T_{AD}):** is the conversion time for each bit. For simplicity assume $T_{AD} \geq 0.8\mu s$.
- T_{ACQ} is defined as multiples of T_{AD} (how many T_{AD} values to exceed $2.5\mu s$).
- T_{AD} is defined as multiples of T_{OSC} (how many T_{OSC} values to exceed $0.8\mu s$).
- $T_{OSC} = \frac{1}{F_{OSC}}$ is the oscillator period.

`ADCON0` Register

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
–	–	`CHS3`	`CHS2`	`CHS1`	`CHS0`	`GO`	`ADON`
		Channel Select					

Reset Value: `00000000`

- Channel Select: Select the analog channel input.

`0000`	= AN0	`0101`	= AN5	`1001`	= AN9
`0001`	= AN1	`0110`	= AN6	`1010`	= AN10
`0010`	= AN2	`0111`	= AN7	`1011`	= AN11
`0011`	= AN3	`1000`	= AN8	`1100`	= AN12
`0100`	= AN4				

- `GO`: A/D conversion start/status bit.
  `0` : idle / conversion finished.
  `1` : start conversion / conversion in progress.

- `ADON`: ADC module on/off bit.
  `0` : module is disabled.
  `1` : module is enabled.

`ADCON1` Register

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
–	–	`VCFG1`	`VCFG0`	`PCFG3`	`PCFG2`	`PCFG1`	`PCFG0`
		$V_{REF(-)}$	$V_{REF(+)}$	Analog/Digital Ports			

Reset Value: `00000000`

- $V_{REF(-)}$: Minimum analog input value.
  `0` : $V_{REF(-)} = V_{SS} = 0V$.
  `1` : $V_{REF(-)}$ = voltage on AN2 [pin 4, RA2].

- $V_{REF(+)}$: Maximum analog input value.
  `0` : $V_{REF(+)} = V_{DD} = 5V$.
  `1` : $V_{REF(+)}$ = voltage on AN3 [pin 5, RA3].

- Analog/Digital Ports: Configure I/O ports to be analog inputs or digital I/O.

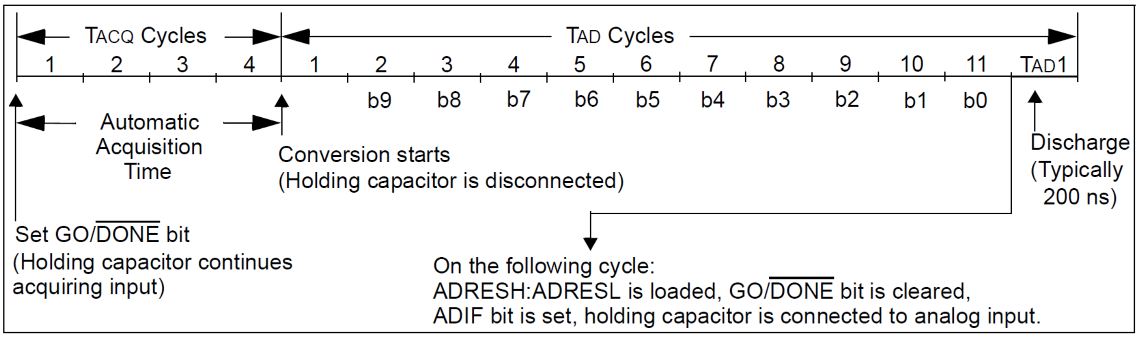
	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
`0000`	A	A	A	A	A	A	A	A	A	A	A	A	A
`0001`	A	A	A	A	A	A	A	A	A	A	A	A	A
`0010`	A	A	A	A	A	A	A	A	A	A	A	A	A
`0011`	D	A	A	A	A	A	A	A	A	A	A	A	A
`0100`	D	D	A	A	A	A	A	A	A	A	A	A	A
`0101`	D	D	D	A	A	A	A	A	A	A	A	A	A
`0110`	D	D	D	D	A	A	A	A	A	A	A	A	A
`0111`	D	D	D	D	D	A	A	A	A	A	A	A	A
`1000`	D	D	D	D	D	D	A	A	A	A	A	A	A
`1001`	D	D	D	D	D	D	D	A	A	A	A	A	A
`1010`	D	D	D	D	D	D	D	D	A	A	A	A	A
`1011`	D	D	D	D	D	D	D	D	D	A	A	A	A
`1100`	D	D	D	D	D	D	D	D	D	D	A	A	A
`1101`	D	D	D	D	D	D	D	D	D	D	D	A	A
`1110`	D	D	D	D	D	D	D	D	D	D	D	D	A
`1111`	D	D	D	D	D	D	D	D	D	D	D	D	D

`ADCON2` Register

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
`ADFM`	–	`ACQT2`	`ACDT1`	`ACQT0`	`ADCS2`	`ADCS1`	`ADCS0`
Result		T_{ACQ}			T_{AD}		

Reset Value: `00000000`

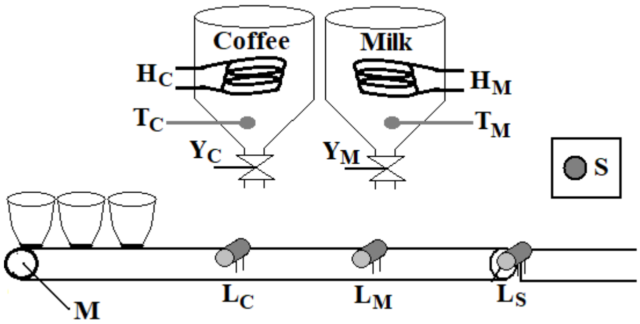
- Result: How to store the 10-bit result in the 16-bit ADRES register pair.
  `0` : Left justification: `|#|#|#|#|#|#|#|#| |#|#|0|0|0|0|0|0|`
  `1` : Right justification: `|0|0|0|0|0|0|#|#| |#|#|#|#|#|#|#|#|`
- T_{ACQ}:
  `000` $= 0\ T_{AD}$ `100` $= 8\ T_{AD}$
  `001` $= 2\ T_{AD}$ `101` $= 12\ T_{AD}$
  `010` $= 4\ T_{AD}$ `110` $= 16\ T_{AD}$
  `011` $= 6\ T_{AD}$ `111` $= 20\ T_{AD}$
- T_{AD}:
  `000` $= 2\ T_{OSC}$ `100` $= 4\ T_{OSC}$
  `001` $= 8\ T_{OSC}$ `101` $= 16\ T_{OSC}$
  `010` $= 32\ T_{OSC}$ `110` $= 64\ T_{OSC}$
  `011` = internal `111` = internal



Example: A machine that fills cups with hot coffee and milk operates as follows:

- When the push-button (S) is pressed one cycle is executed to fill one cup.
  1. The motor (M) is on until the cup is under the coffee tank at sensor (L_C).
  2. The motor (M) is off and the valve (V_C) is opened for 3 seconds.
  3. The motor (M) is on until the cup is under the milk tank at sensor (L_M).
  4. The motor (M) is off and the valve (V_M) is opened for 5 seconds.
  5. The motor (M) is on until the cup reaches sensor (L_S) and the cycle ends.
- The heaters (H_C) and (H_M) are used to heat the coffee and milk respectively.
- The sensors (T_C) and (T_M) are used to measure the temperatures of the coffee and milk respectively.

- The coffee temperature should be 70°C ± 2°C. The milk temperature should be 60°C ± 2°C.
- The input range for the temperature sensors is 0 – 5 V, and the output range is 0 – 100°C.
- Assume F_{OSC} = 20 MHz.



Inputs		Outputs	
S	RD0	M	RC0
LC	RD1	VC	RC1
LM	RD2	VM	RC2
LS	RD3	HC	RC6
TC	AN8 [RB2]	HM	RC7
TM	AN9 [RB3]		

ADC Setup:

$$T_{OSC} = \frac{1}{20\text{ MHz}} = 0.05\ \mu\text{s}.$$

$$\frac{0.8\ \mu s}{0.05\ \mu s} = 16 \quad \Rightarrow \quad T_{AD} = 16\ T_{OSC}$$

$$\frac{2.5\ \mu s}{16 \times 0.05\ \mu s} = 3.125 \ \Rightarrow \ T_{ACQ} = 4\ T_{AD}$$

ADCON0 (AN8) = 0x21

0	0	1	0	0	0	0	1
−−		AN8				idle	on

ADCON0 (AN9) = 0x25

0	0	1	0	0	1	0	1
−−		AN9				idle	on

ADCON1 = 0x05

0	0	0	0	0	1	0	1
−−		V_{REF}		AN9 analog			

ADCON2 = 0x95

1	0	0	1	0	1	0	1
right	−	T_{ACQ}			T_{AD}		

0°C – 100°C ≡ 0 – 1023

58°C	≡ 593	68°C	≡ 696
62°C	≡ 634	72°C	≡ 737

```
#include <xc.h>
#define _XTAL_FREQ 20000000
#define S  PORTDbits.RD0
#define LC PORTDbits.RD1
#define LM PORTDbits.RD2
#define LS PORTDbits.RD3
#define M  LATCbits.LATC0
#define VC LATCbits.LATC1
#define VM LATCbits.LATC2
```

```
#define HC LATCbits.LATC6
#define HM LATCbits.LATC7

void main() {
   unsigned char x,S_old;
   int TC, TM;
   TRISB = 0xFF;
   TRISC = 0;
   TRISD = 0xFF;
   ADCON1 = 0x05;
   ADCON2 = 0x95;
   M = 0; VC = 0; VM = 0; HC = 0; HM = 0;
   S_old = 0;

   while(1) {
      ADCON0 = 0x21;
      ADCON0bits.GO = 1;
      while(ADCON0bits.GO);
      TC = ADRESH;
      TC = (TC << 8) + ADRESL;
      if (TC < 696) HC = 1;
      if (TC > 737) HC = 0;

      ADCON0 = 0x25;
      ADCON0bits.GO = 1;
      while(ADCON0bits.GO);
      TM = ADRESH;
      TM = (TM << 8) + ADRESL;
      if (TM < 593) HM = 1;
      if (TM > 634) HM = 0;

      if(!S_old && S) {
         M = 1;
         while(!LC);
         M = 0; VC = 1;
         __delay_ms(3000);
         VC = 0; M = 1;
         while(!LM);
         M = 0; VM = 1;
         __delay_ms(5000);
         VM = 0; M = 1;
         while(!LS);
         M = 0;
      }
```

```
        S_old = S;
    }
}
```

Problem: A water tank is filled using three pumps (P1, P2 and P3) as follows:

- The water level is measured using a sensor with input range of 0m to 3m, and output range of 0.4V to 4.0V.
- If the water level is less than 75cm, the three pumps are turned on.
- If If the water level is more than 1.25m and less than 1.75m, P1 and P2 are turned on and P3 is turned off.
- If If the water level is more than 2.25m and less than 1.75m, P1 and P2 are turned off and P3 is turned on.
- The water level should not exceed 3m.

F_{OSC} = 13.33MHz. Write the program.

Problem: Complete the following code for an Analog-Input Digital-Output calculator. The calculator has three analog inputs (`AN1`, `AN2`, and `AN3`) connected to potentiometers (range = 0V to 5V). The analog inputs represent integer numbers between 0 and +85. The total of the three numbers is displayed on `Port B` and the maximum number is displayed on `Port D`.

```
#include <xc.h>
#pragma config PLLDIV = 6
#pragma config CPUDIV = OSC3_PLL4
#pragma config FOSC = ECPLL_EC
void main() {
int N[3], x;
```

Chapter 4

Oscillator Configuration

References:

Datasheet : 2.0 Oscillator Configurations.

Mazidi et al. : Chapter 8 PIC18F Hardware Connection and ROM Loaders.

4.1 Clock Sources

There are three clock sources for the PIC18F4550:

- Primary oscillators:
  - Can be: crystal, ceramic resonator, or external clock input.
  - Maximum F_{OSC} = 48 Mhz.
  - Can be with or without Phase Locked Loop (PLL).
  - For the PLL block, the input frequency is 4 Mhz and the output frequency is 96 Mhz.
- Secondary oscillators: Will not be covered in this course.
- Internal oscillators:
  - Maximum F_{OSC} = 8 Mhz.
  - No external connections needed.

Oscillator modes: 12 modes.

Configuration bits: `PLLDIV`, `CPUDIV`, `FOSC`.

Control registers: `OSCCON`, `OSCTUNE`.

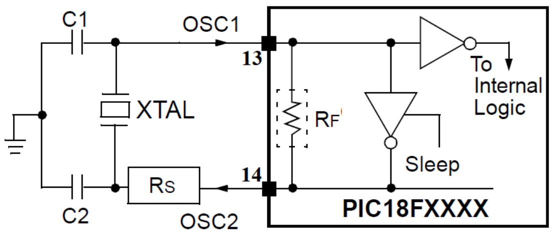
USB frequency: 48 MHz or 6 MHz.

- XT
  - No PLL
  - With PLL
- HS
  - No PLL
  - With PLL
- EC
  - No PLL
    - CLKO
    - RA6
  - With PLL
    - CLKO
    - RA6
- INT
  - USB-EC
    - CLKO
    - RA6
  - USB-XT
  - USB-HS

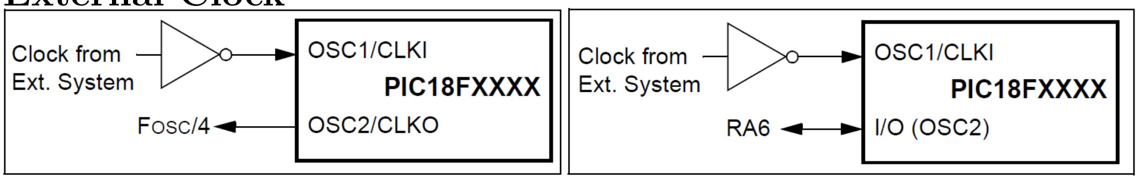
```
Primary
                      with PLL          4MHz      96MHz
         OSC1---+---+-----------PLLDIV------PLL-------CPUDIV---FOSC
                |   |
OSC2/CLKO/RA6---+   +---------------------------------CPUDIV---FOSC
                      without PLL
Secondary

Internal
         8MHz  -----IRCF----------------------------------------FOSC
                    |
     RC 31kHz  ------+
```

Crystal Oscillator

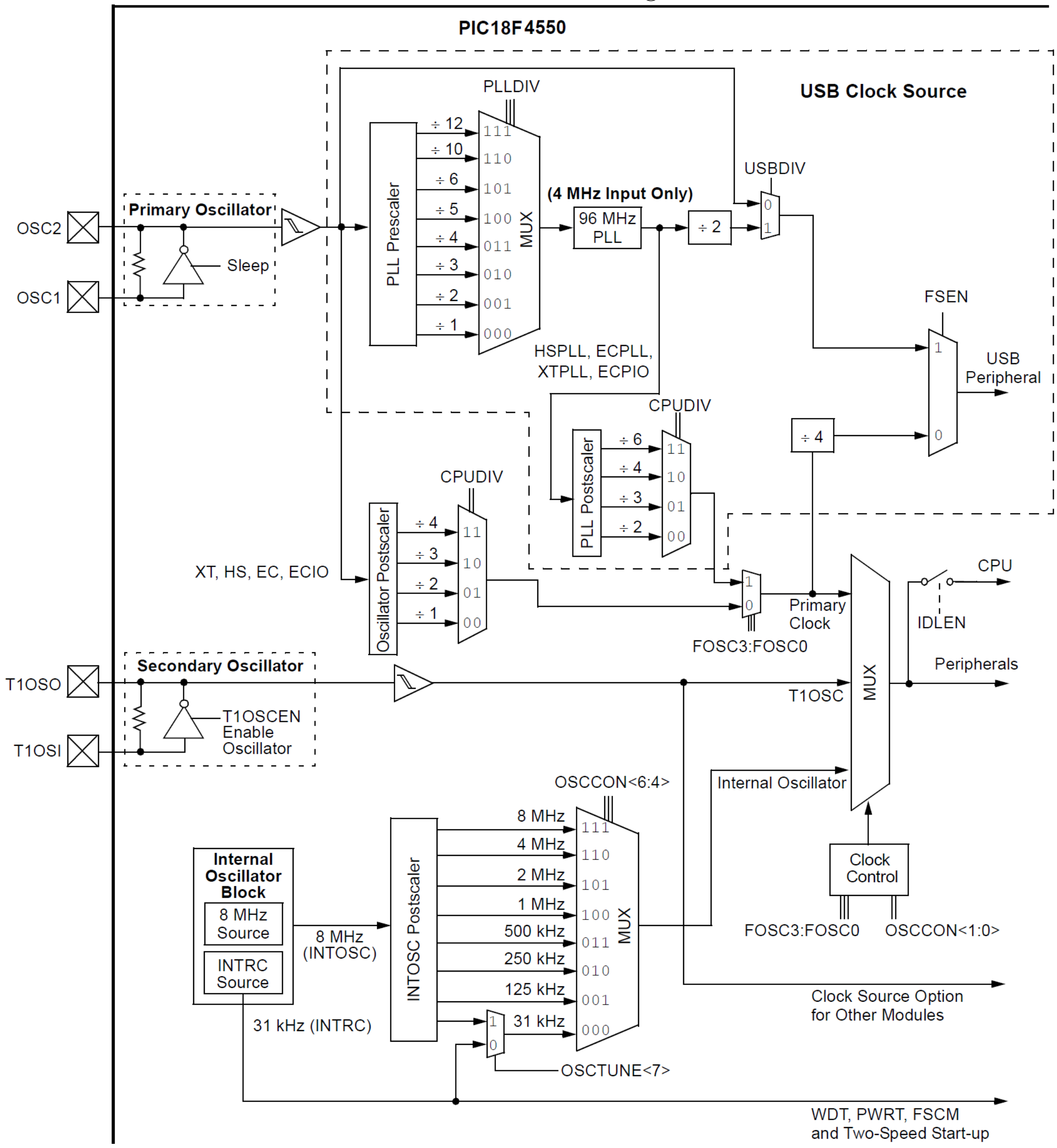
Osc Type	Crystal Freq	Typical Capacitor Values Tested:	
		C1	C2
XT	4 MHz	27 pF	27 pF
HS	4 MHz	27 pF	27 pF
	8 MHz	22 pF	22 pF
	20 MHz	15 pF	15 pF



External Clock



PIC18F4550 Clock Diagram



4.2 Oscillator Configuration and Control

`PLLDIV`: PLL Prescaler Selection bits:

PLLDIV = 1	No prescale [4 MHz input]
PLLDIV = 2	Divide by 2 [8 MHz input]
PLLDIV = 3	Divide by 3 [12 MHz input]
PLLDIV = 4	Divide by 4 [16 MHz input]
PLLDIV = 5	Divide by 5 [20 MHz input]
PLLDIV = 6	Divide by 6 [24 MHz input]
PLLDIV = 10	Divide by 10 [40 MHz input]
PLLDIV = 12	Divide by 12 [48 MHz input]

Default value: `PLLDIV = 1`

`CPUDIV`: System Clock Postscaler Selection bits:

CPUDIV = OSC1_PLL2	Divide primary by 1	$96 \div 2 = 48$
CPUDIV = OSC2_PLL3	Divide primary by 2	$96 \div 3 = 32$
CPUDIV = OSC3_PLL4	Divide primary by 3	$96 \div 4 = 24$
CPUDIV = OSC4_PLL6	Divide primary by 4	$96 \div 6 = 16$

Default value: `CPUDIV = OSC1_PLL2`

`FOSC`: Oscillator Selection bits:

FOSC = XT_XT	4 MHz Crystal
FOSC = XTPLL_XT	4 MHz Crystal with PLL
FOSC = HS	Crystal
FOSC = HSPLL_XT	Crystal with PLL
FOSC = EC_EC	External clock, CLKO
FOSC = ECIO_EC	External clock, RA6
FOSC = ECPLL_EC	External clock with PLL, CLKO
FOSC = ECPLLIO_EC	External clock with PLL, RA6
FOSC = INTOSC_EC	Internal clock, CLKO, EC for USB
FOSC = INTOSCIO_EC	Internal clock, RA6, EC for USB
FOSC = INTOSC_XT	Internal clock, 4 MHz Crystal for USB
FOSC = INTOSC_HS	Internal clock, Crystal for USB

Default value: `FOSC = EC_EC`

Note: Configuration bits do not change with a Reset.

`OSCCON` Register

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1	SCS0
	Internal Clock			Read only		Clock Source	

Reset Value: `0100x000`

- `IDLEN`: Used in power management, not used in this course.

- Internal Clock: Select the frequency of the internal oscillator.

`000`	= 31 kHz	`100`	= 1 MHz
`001`	= 125 kHz	`101`	= 2 MHz
`010`	= 250 kHz	`110`	= 4 MHz
`011`	= 500 kHz	`111`	= 8 MHz

- The read only bits are not used in this course.

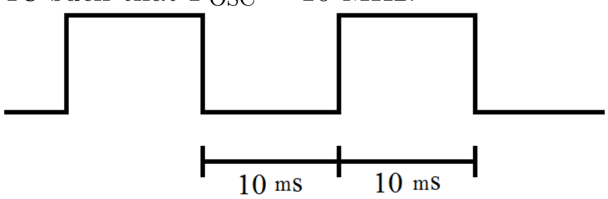
- Clock source: Select the system clock source.
  `00` Primary
  `01` Secondary
  `1x` Internal

`OSCTUNEbits.INTSRC`: Select the source for the 31 kHz internal frequency.
`0` 31.25 kHz derived from the 8 MHz oscillator (8 MHz ÷ 256)
`1` 31 kHz derived from the internal RC oscillator.
Reset value: `0`

Input Oscillator Frequency	PLL Division (PLLDIV2:PLLDIV0)	Clock Mode (FOSC3:FOSC0)	MCU Clock Division (CPUDIV1:CPUDIV0)	Microcontroller Clock Frequency
48 MHz	N/A[1]	EC, ECIO	None (00)	48 MHz
			÷2 (01)	**24 MHz**
			÷3 (10)	16 MHz
			÷4 (11)	12 MHz
48 MHz	÷12 (111)	EC, ECIO	None (00)	48 MHz
			÷2 (01)	**24 MHz**
			÷3 (10)	16 MHz
			÷4 (11)	12 MHz
		ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz
40 MHz	÷10 (110)	EC, ECIO	None (00)	40 MHz
			÷2 (01)	20 MHz
			÷3 (10)	13.33 MHz
			÷4 (11)	10 MHz
		ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz
24 MHz	÷6 (101)	HS, EC, ECIO	None (00)	**24 MHz**
			÷2 (01)	12 MHz
			÷3 (10)	8 MHz
			÷4 (11)	6 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz

Input Oscillator Frequency	PLL Division (PLLDIV2:PLLDIV0)	Clock Mode (FOSC3:FOSC0)	MCU Clock Division (CPUDIV1:CPUDIV0)	Microcontroller Clock Frequency
20 MHz	÷5 (100)	HS, EC, ECIO	None (00)	20 MHz
			÷2 (01)	10 MHz
			÷3 (10)	6.67 MHz
			÷4 (11)	5 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz
16 MHz	÷4 (011)	HS, EC, ECIO	None (00)	16 MHz
			÷2 (01)	8 MHz
			÷3 (10)	5.33 MHz
			÷4 (11)	4 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz
12 MHz	÷3 (010)	HS, EC, ECIO	None (00)	12 MHz
			÷2 (01)	6 MHz
			÷3 (10)	4 MHz
			÷4 (11)	3 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz
8 MHz	÷2 (001)	HS, EC, ECIO	None (00)	8 MHz
			÷2 (01)	4 MHz
			÷3 (10)	2.67 MHz
			÷4 (11)	2 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz
4 MHz	÷1 (000)	XT, HS, EC, ECIO	None (00)	4 MHz
			÷2 (01)	2 MHz
			÷3 (10)	1.33 MHz
			÷4 (11)	1 MHz
		HSPLL, ECPLL, XTPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz

Example: Write a program to generate the following signal on `RD0`. Select a crystal and configure the PIC such that $F_{OSC} = 10$ MHz.



10 MHz options: 40 MHz ÷ 4 without PLL, 20 MHz ÷ 2 without PLL.

Crystal cannot be 40 MHz $\Rightarrow$ use 20 MHz crystal.

OSCCON =

0	0	0	0	0	0	0	0
X	X			X		Primary	

= 0x00

```
#include <xc.h>
#define _XTAL_FREQ 10000000
#pragma config CPUDIV = OSC2_PLL3
#pragma config FOSC = HS
void main() {
   TRISD = 0;
   OSCCON = 0;
   while(1) {
      LATDbits.LATD0 = !PORTDbits.RD0;
      __delay_ms(10000);
   }
}
```

Example: Write a program to continuously read a voltage value at `AN0`. The voltage range is 0 to 3 V. If the input voltage is less than 1 V, a red light at `RA6` is turned on. Configure the PIC such that F_{OSC} = 32 MHz using an input oscillator frequency of 16 MHz.

16 MHz $\div$ 4 = 4 MHz, with PLL 96 MHz $\div$ 3 = 32 MHz.

Available modes: `HSPLL_HS`, `ECPLL_EC`, `ECPLLIO_EC`.

RA6 is digital output $\Rightarrow$ use `ECPLLIO_EC`.

Inputs		Outputs	
Voltage	AN0 [RA0]	Red light	RA6
$V_{REF(+)}$	AN3 [RA3]		

$$T_{OSC} = \frac{1}{32\ \text{MHz}} = 0.03125\ \mu\text{s}.$$

$$\frac{0.8\ \mu\text{s}}{0.03125\ \mu\text{s}} = 25.6 \quad \Rightarrow \quad T_{AD} = 32\ T_{OSC}$$

$$\frac{2.5\ \mu\text{s}}{32 \times 0.03125\ \mu\text{s}} = 2.4 \quad \Rightarrow \quad T_{ACQ} = 4\ T_{AD}$$

ADCON0 =

0	0	0	0	0	0	0	1
−−		AN0				idle	on

= 0x01

ADCON1 =

0	0	0	1	1	0	1	1
−−		V_{REF}		AN3 analog			

= 0x1B

ADCON2 =

1	0	0	1	0	0	1	0
right	−	T_{ACQ}			T_{AD}		

= 0x92

0 − 3 V ≡ 0 − 1023 1 V ≡ 341

```
#include <xc.h>
#pragma config PLLDIV = 4
#pragma config CPUDIV = OSC2_PLL3
```

```
#pragma config FOSC = ECPLLIO_EC

void main() {
   int V;
   TRISA = 0b00001001;
   CMCON = 7;
   OSCCON = 0;
   ADCON0 = 0x01;
   ADCON1 = 0x1B;
   ADCON2 = 0x92;
   while(1) {
      ADCON0bits.GO = 1;
      while(ADCON0bits.GO);
      V = ADRESH;
      V = (V << 8) + ADRESL;
      LATAbits.LATA6 = (V < 341);
   }
}
```

Problem: Write an XC8 program to control eight LEDs using one push-button as follows:

- Initially, all the LEDs are off.
- If the push-button is pressed and released, the right-most LED is turned on.
- The on LED is moved one step to the left every three seconds. (Only one LED is on at a time)
- After the left-most LED is on for three seconds, all the LEDs are off again.

Configure the PIC such that F_{OSC} = 2 MHz derived from the internal oscillator.

Chapter 5

Timers

References:

Datasheet : 11.0 Timer0 Module.
13.0 Timer2 Module.

Mazidi et al. : Chapter 9 PIC18 Timer Programming in Assembly and C.

Huang : Chapter 8 Timers and CCP Modules.

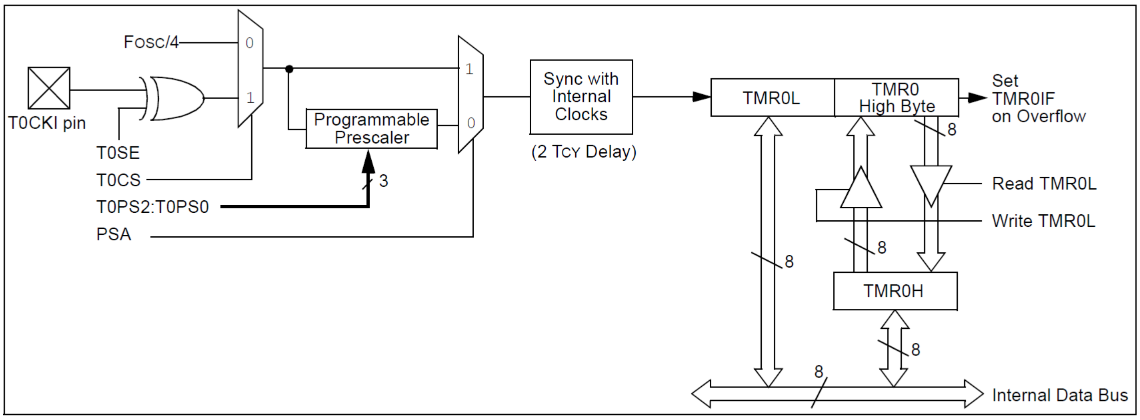
5.1 Timers in PIC18F4550

The PIC18F4550 has four timers/counters.

	Timer 0	Timer 1	Timer 2	Timer 3
Counter size	8-bit/16-bit	16-bit	8-bit	16-bit
Function	timer/counter	timer/counter	timer	timer/counter
Read/write registers	`TMR0H, TMR0L`	`TMR1H, TMR1L`	`TMR2, PR2`	`TMR3H, TMR3L`
Clock source	F_{OSC}/`T0CKI`	F_{OSC}/`T13CKI`	F_{OSC}	F_{OSC}/`T13CKI`
Prescaler	8 options	4 options	3 options	4 options
Postscaler	–	–	16 options	–

5.2 Timer 0 Module

- Internal clock source = $\frac{F_{OSC}}{4}$.
- 8-bit or 16-bit timer/counter [`TMR0H:TMR0L`], readable and writable.
- For counter: connect signal source to pin 6 (`RA4/T0CKI/C1OUT/RCV`).
- Write `TMR0H` then `TMR0L`. Read `TMR0L` then `TMR0H`.
  - The value of `TMR0H` is written to the timer only when `TMR0L` is updated.
  - The value of `TMR0H` is updated from the timer only when `TMR0L` is read.



`T0CON` Register

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
`TMR0ON`	`T08BIT`	`T0CS`	`T0SE`	`PSA`	`T0PS2`	`T0PS1`	`T0PS0`
					Prescaler value		

Reset Value: `11111111`

- `TMR0ON`: Run or stop Timer 0.
  `0` Stop Timer 0.
  `1` Enable Timer 0.
- `T08BIT`: 8-bit or 16-bit timer/counter.
  `0` 16-bit [`TMR0H:TMR0L`].
  `1` 8-bit [`TMR0L`].
- `T0CS`: Timer 0 clock source.
  `0` $F_{OSC}/4$ (timer).
  `1` `T0CKI` (timer/counter).
- `T0SE`: Timer 0 source edge in counter mode.
  `0` Count at rising edge on `T0CKI`.
  `1` Count at falling edge on `T0CKI`.
- `PSA`: Enable/disable prescaler.
  `0` Prescaler enabled.
  `1` prescaler disabled.
- Prescaler value: Timer 0 prescaler value.
  `000` $= 1:2$ `100` $= 1:32$
  `001` $= 1:4$ `101` $= 1:64$
  `010` $= 1:8$ `110` $= 1:128$
  `011` $= 1:16$ `111` $= 1:256$

`INTCONbits.TMR0IF`: Timer 0 interrupt flag.
When Timer 0 timer/counter value is `0xFFFF` for 16-bit (or `0xFF for 8-bit`), the *next* count resets the count to `0x0000` (or `0x00`) and sets the bit `INTCONbits.TMR0IF` to `1` *automatically*. The programmer should reset the bit `INTCONbits.TMR0IF` to `0` *in the code*.

Example: Write a program to toggle `RB4` continuously every 100 ms. Use Timer 0 in 16-bit mode. Assume F_{OSC} = 10 MHz is readily configured.

$$T_{cy} = \frac{4}{F_{OSC}} = 0.4\mu s$$

Total time = 100 ms

$$\text{count} = \frac{100\text{ms}}{0.4\mu s} = 250000 > 65535$$

$$250000 \div 65535 = 3.815 \quad \Rightarrow \quad 1:4 \text{ prescaler}$$

$$\text{count} = \frac{100\text{ms}}{4 \times 0.4\mu s} = 62500 = F424_H$$

$$10000_H - F424_H = 0BDC_H$$

`T0CON` =

1	0	0	0	0	0	0	1
run	16-bit	timer	X	use prescaler	1:4		

= `0x81`

```
#include <xc.h>
void main() {
   TRISBbits.TRISB4 = 0;
   ADCON1 = 0x0F;
   T0CON = 0x81;
   while(1) {
      LATBbits.LATB4 = !PORTBbits.RB4;
      TMR0H = 0x0B;
      TMR0L = 0xDC;
      while(INTCONbits.TMR0IF == 0);
      INTCONbits.TMR0IF = 0;
   }
}
```

Example: A simple function generator based on PIC18F4550 operates as follows: a switch is connected to pin `RB7`. If the switch is on, a 5 Hz square wave is generated on pin `RB0`,if the switch is off, a 1.5 Hz square wave is generated on pin `RB0`. Use Timer 0 to write the program and configure the oscillator to provide a 32 MHz clock in XT mode.

XT mode: 4 MHz crystal ÷ 1 = 4 MHz with PLL 96 MHz ÷ 3 = 32 MHz.

$$T_{cy} = \frac{4}{F_{OSC}} = 0.125\mu s$$

For 5 Hz: half a period = 0.5 ÷ 5 Hz = 0.1 s

$$\text{count} = \frac{0.1\text{s}}{0.125\mu s} = 800000 > 65535$$

$$800000 \div 65535 = 12.207 \quad \Rightarrow \quad 1:16 \text{ prescaler}$$

For 1.5 Hz: half a period = 0.5 ÷ 1.5 Hz = 0.333 s

$$\text{count} = \frac{0.333\text{s}}{0.125\mu\text{s}} = 2666667 > 65535$$

$$2666667 \div 65535 = 40.69 \quad \Rightarrow \quad 1:64 \text{ prescaler}$$

We can use 1:64 prescaler for both frequencies.

For 5 Hz:

$$\text{count} = \frac{0.1\text{s}}{64 \times 0.125\mu\text{s}} = 12500 = 30\text{D}4_\text{H}$$

$$10000_\text{H} - 30\text{D}4_\text{H} = \text{CF}2\text{C}_\text{H}$$

For 1.55 Hz:

$$\text{count} = \frac{0.333\text{s}}{64 \times 0.125\mu\text{s}} = 41667 = \text{A}2\text{C}2_\text{H}$$

$$10000_\text{H} - \text{A}2\text{C}2_\text{H} = 5\text{D}3\text{E}_\text{H}$$

T0CON =

1	0	0	0	0	1	0	1
run	16-bit	timer	X	use prescaler	1:64		

= 0x85

```
#include <xc.h>
#pragma config PLLDIV = 1
#pragma config CPUDIV = OSC2_PLL3
#pragma config FOSC = XTPLL_XT
#define SW PORTBbits.RB7
#define FG LATBbits.LATB0

void func(void);

void main() {
   TRISB = 0x80;
   ADCON1 = 0x0F;
   OSCCON = 0;
   T0CON = 0x85;
   FG = 1;
   func();
   while(1) {
      if(INTCONbits.TMR0IF) {
         func();
         INTCONbits.TMR0IF = 0;
      }
   }
}

void func() {
  FG = !FG;
  if(SW) {
    TMR0H = 0xCF;
    TMR0L = 0x2C;
  }
  else {
```

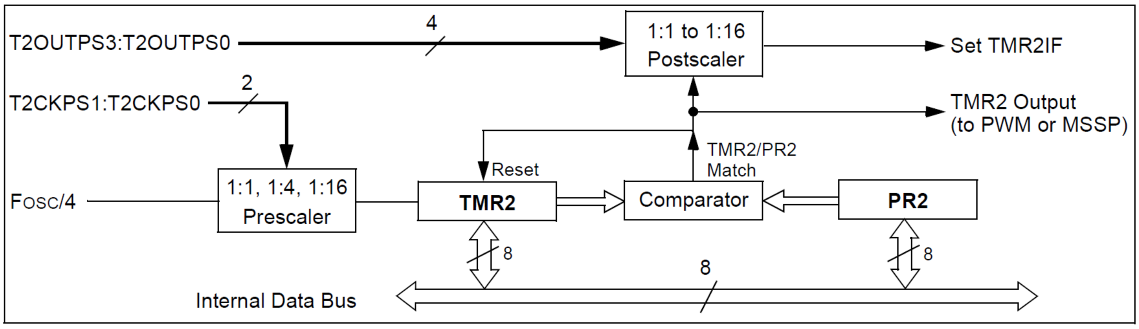
```
    TMR0H = 0x5D;
    TMR0L = 0x3E;
  }
}
```

Problem: The door of an industrial oven is operated using two push-buttons as follows:

- The door of the oven is driven by a single-acting cylinder (`0`: close, `1`: open).
- The door of the oven is opened if the OPEN push-button is pressed and the temperature is below 250°C. (If the temperature is above 250°C the door does not open)
- The door is closed manually by pressing the CLOSE push-button, or automatically after 3 seconds.
- The temperature of the oven is measured using a sensor with input range of 20°C to 1000°C, and output range of 0.1V to 5.0V.
- F_{OSC} = 1 MHz is readily configured.

5.3 Timer 2 Module

- Internal clock source = $\frac{F_{OSC}}{4}$.
- 8-bit timer [`TMR2`], readable and writable.
- 8-bit period register [`PR2`], readable and writable.
- When `TMR2` = `PR2`, the value of `TMR2` is reset to `0x00`.
- Used for CCP (in PWM mode) and for MSSP.
  (CCP: capture/compare/pwm, MSSP: master synchronous serial port)
- Has a prescaler and a postscaler.
- Postscaler is used for timer only (not for CCP or MSSP).



`T2CON` Register

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
–	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
	Postscaler value					Prescaler value	

Reset Value: `00000000`

- Postscaler value: Timer 2 postscaler value (not used in CCP or MSSP).

`0000` = 1:1	`0100` = 1:5	`1000` = 1:9	`1100` = 1:13
`0001` = 1:2	`0101` = 1:6	`1001` = 1:10	`1101` = 1:14
`0010` = 1:3	`0110` = 1:7	`1010` = 1:11	`1110` = 1:15
`0011` = 1:4	`0111` = 1:8	`1011` = 1:12	`1111` = 1:16

- `TMR2ON`: Run or stop Timer 2.
  - `0` Stop Timer 2.
  - `1` Enable Timer 2.

- Prescaler value: Timer 2 prescaler value.
  - `00` = 1 : 1
  - `01` = 1 : 4
  - `1x` = 1 : 16

`PIR1bits.TMR2IF`: Timer 2 interrupt flag.
When Timer 2 timer value `TMR2` equals the period value `PR2`, the next count resets `TMR2` to `0x00` and sets the bit `PIR1bits.TMR2IF` to `1` *automatically*. The programmer should reset the bit `PIR1bits.TMR2IF` to `0` *in the code*.

Example: Write a program for a digital clock based on Timer 2. The clock should count in 10 ms. The minutes are displayed on port B and the hours are displayed on port D. Configure the oscillator to provide a 8 MHz derived from the internal oscillator.

8 MHz internal, USB and `RA6` are not used ⇒ any `INTOSC` mode can be used .

`OSCCON` =

0	1	1	1	0	0	1	0
X	8 MHz			X		Internal	

= `0x72`

$$T_{cy} = \frac{4}{F_{OSC}} = 0.5\mu s$$

counting time = 0.01 s

$$\text{count} = \frac{0.01s}{0.5\mu s} = 20000 > 255$$

$20000 \div 255 = 78.4$ ⇒ 1:16 prescaler, 1:5 postscaler

$$\text{count} = \frac{0.01s}{5 \times 16 \times 0.5\mu s} = 250$$

`T2CON` =

0	0	1	0	0	1	1	0
X	1:5 post				run	1:16 pre	

= `0x46`

```
#include <xc.h>
```

```
#pragma config FOSC = INTOSC_HS
#define M LATB
#define H LATD

void main() {
   unsigned char ten_ms, s;
   TRISB = 0;
   TRISD = 0;
   ADCON1 = 0x0F;
   OSCCON = 0x72;
   PR2 = 250;
   T2CON = 0x46;
   M = 0; H = 0; s = 0; ten_ms = 0;
   while(1) {
      if(PIR1bits.TMR2IF) {
         PIR1bits.TMR2IF = 0;
         ten_ms++;
         if(ten_ms == 100) {
            ten_ms = 0;
            s++;
            if(s == 60) {
               s = 0;
               M++;
               if(M == 60) {
                  M = 0;
                  H++;
                  if(H == 24) H = 0;
               }
            }
         }
      }
   }
}
```

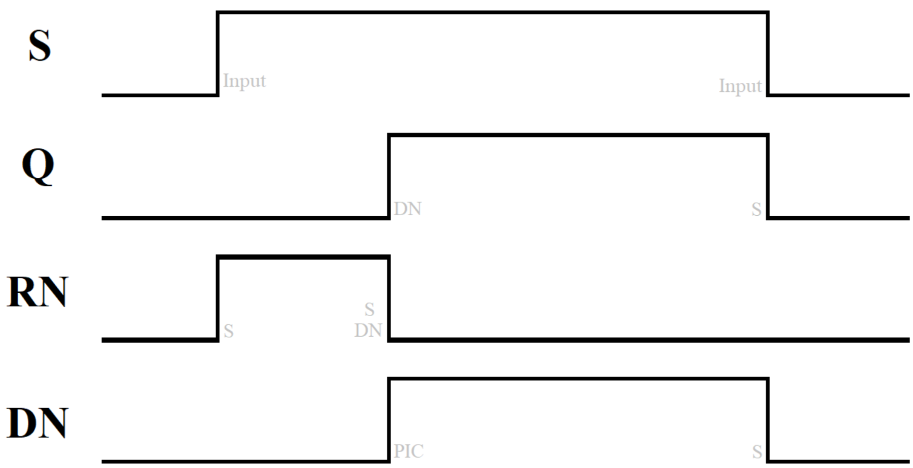
Problem: Write a program to operate a motor using one push-button as follows: When the push-button is pressed and released the motor turns on and runs until a specified time is over or when the push-button is pressed and released again. The running time is between 1 second and 5 seconds (in 25 ms increments) and is selected using a potentiometer with voltage input range of 0.8 V to 4 V. Configure the PIC such that F_{OSC} = 8 MHz is derived from a 8 MHz crystal.

Problem: A machine that has two motors and one push-button operates as follows:

- If the push-button is pressed for the first time, the first motor turns on.
- While the first motor is on, the second motor turns on after 3 seconds OR if the push-button is pressed again.
- The motors turn off if the push-button is pressed again.
- $F_{OSC} = 500$ kHz is readily configured.

Write an XC8 code to operate the machine and use Timer 2 in your solution.

5.4 Application: The On-Delay Timer



S = 0			
RN	DN		
0	0		
0	1	···	RN = 0
1	0	···	DN = 0
1	1	···	Q = 0

S = 1		
RN	DN	
0	0	RN = 1
0	1	
1	0	
1	1	··· RN = 0, Q = 1

Assume:
Delay time = 1.5 seconds.
$F_{OSC} = 500$ kHz is readily configured.

$T_{cy} = \dfrac{4}{F_{OSC}} = 8\mu s$

Delay time = 1.5 s

$\text{count} = \dfrac{1.5s}{8\mu s} = 187500 > 65535$

$187500 \div 65535 = 2.86 \quad \Rightarrow \quad 1:4$ prescaler

$$\text{count} = \frac{1.5\text{s}}{4 \times 8\mu\text{s}} = 46875 = \text{B71B}_\text{H}$$

$$10000_\text{H} - \text{B71B}_\text{H} = \text{48E5}_\text{H}$$

`T0CON` =

0	0	0	0	0	0	0	1
stop	16-bit	timer	X	use prescaler	1:4		

= `0x01`

```
#include <xc.h>
#define S PORTDbits.RD0
#define Q LATDbits.LATD1
#define RN T0CONbits.TMR0ON
#define DN INTCONbits.TMR0IF

void main() {
   TRISD = 1;
   T0CON = 0x01;
   Q = 0;
   while(1) {
      if(S) {
         if(!RN && !DN) {
            TMR0H = 0x48;
            TMR0L = 0xE5;
            RN = 1;
         }
         else if(RN && DN) {
            RN = 0;
            Q = 1;
         }
      }
      else
         if(RN || DN) {
            RN = 0;
            DN = 0;
            Q = 0;
         }
   }
}
```

Chapter 6

Pulse Width Modulation

References:

Datasheet : 15.0 Capture/Compare/PWM (CCP) Modules.

Mazidi et al. : Chapter 15 CCP and ECCP Programming.

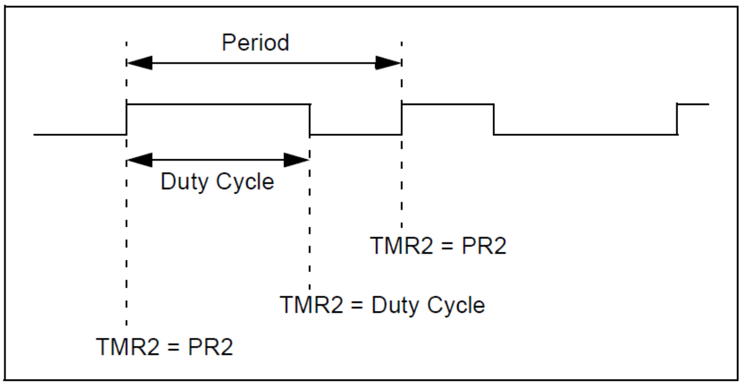
Huang : Chapter 8 Timers and CCP Modules.

6.1 The CCP Module

- **Capture:** save the value of the timer (Timer 1 or Timer 3) in the CCP registers when an event occures on the input pin (`RC1` or `RB3`).
- **Compare:** when the value of the timer (Timer 1 or Timer 3) equals the CCP registers, do an action on the output pin (`RC1` or `RB3`).
- **PWM:** up to 10-bit resolution Pulse Width Modulation (Timer 2) on output pin (`RC1` or `RB3`).

The PIC18F4550 has one CCP module (`CCP2`) and one Enhanced CCP module (`CCP1`).

6.2 CCP2 Module in PWM Mode



- The CCP2 module in PWM mode produces a 10-bit resolution PWM output on bit RC1 or RB3.
- The appropriate TRIS bit must be 0 to make the CCP2 pin an output.
- The PWM period is specified by the PR2 register.
  Period $= (\mathtt{PR2}+1) \times 4 \times T_{OSC} \times$ **Prescaler**
- The PWM duty cycle is specified by the CCPR2L register (whole number) and bits <5:4> in the CCP2CON register (binary fraction).
  On Time $= (\mathtt{CCPR2L.CCP2CON<5:4>}) \times 4 \times T_{OSC} \times$ **Prescaler**
- Timer 2 postscaler is not used in PWM.

CCP2CON Register

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
–	–	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0
		DC bits 1 and 0		CCP2 Mode			

Reset Value: 00000000

- DC bits 1 and 0: The two *binary fraction* bits of the PWM duty cycle.
  00 ≡ 0.00   10 ≡ 0.50
  01 ≡ 0.25   11 ≡ 0.75
- CCP2 Mode: Select the mode of operation for the CCP2 module.
  0000 : CCP2 is disabled
  11xx : PWM mode.

CCP2MX: CCP2 Multiplexed bit:

CCP2MX = OFF	CCP2 input/output is multiplexed with RB3
CCP2MX = ON	CCP2 input/output is multiplexed with RC1

Default value: CCP2MUX = ON

Example: Find the values of `PR2`, `CCP2RL` and `CCP2CON` for 75% duty cycle PWM with a frequency of 1 kHz. $F_{OSC} = 10$ MHz.
Prescaler 1:16

$$\texttt{PR2} = \frac{F_{OSC}}{4 \times F_{PWM} \times \text{Prescaler}} - 1 = \frac{10\text{MHz}}{4 \times 1\text{kHz} \times 16} - 1 = 156 - 1 = 155$$

$$\begin{aligned} \texttt{CCPR2L.CCP2CON<5:4>} &= (\text{On Time})\frac{F_{OSC}}{4 \times \text{Prescaler}} \\ &= \frac{0.75}{1\text{kHz}}\frac{10\text{MHz}}{4 \times 16} = 117.1875 \approx 117.25 \end{aligned}$$

`CCPR2L` = 117

`CCP2CON` = `0x1C`

0	0	0	1	1	1	0	0
X		0.25		PWM			

Example: Given that `PR2 = 20`, `CCP2RL = 10`, `CCP2CON = 0x2C`, and `T2CON = 0x01`, find the frequency and duty cycle for the PWM. $F_{OSC} = 40$ MHz.

`T2CON = 0x01 = 0b00000001` ⇒ 1:4 prescaler.

$$\text{Period} = \frac{(\texttt{PR2} + 1) \times 4 \times \text{Prescaler}}{F_{OSC}} = \frac{(20 + 1) \times 4 \times 4}{40\text{MHz}} = 8.4\mu s$$

$$\text{PWM frequency} = \frac{1}{\text{Period}} = 119 \text{ kHz}$$

`CCP2CON = 0x2C = 0x00101100` ⇒ Binary fraction = 0.50

$$\text{On Time} = \frac{(\texttt{CCPR2L.CCP2CON<5:4>}) \times 4 \times \text{Prescaler}}{F_{OSC}} = \frac{(10.50) \times 4 \times 4}{40\text{MHz}} = 4.2\mu s$$

$$\text{Duty cycle} = \frac{\text{On Time}}{\text{Period}} \times 100\% = \frac{4.2\mu s}{8.4\mu s} \times 100\% = 50\%$$

Example: A PIC18 based air-conditioning system operates as follows:

- The temperature is measured ten times every second. [`AN8`]
- For the temperature sensor: Input range: −20°C to 60°C; Output range: 0 V to 5 V
- The average of the last 15 measurements is calculated. [`Tavg`]
- Set points:

Set points:	-20°C < `Tavg` < 8°C	Heater at 100%
	12°C < `Tavg` < 23°C	Heater at 50%
	27°C < `Tavg` < 60°C	Heater at 0%

- Configure the internal oscillator for F_{OSC}= 4 MHz.
- For 50% Heater operation, use a 600 Hz signal.

Hint: Use Timer 0 for timing temperature readings, and CCP2 as PWM for 50% Heater operation.

Oscillator Setup:

4 MHz internal, USB and `RA6` are not used ⇒ any `INTOSC` mode can be used .

`OSCCON` =

0	1	1	0	0	0	1	0
X	4 MHz			X		Internal	

= `0x62`

ADC Setup:

$$T_{OSC} = \frac{1}{4\text{ MHz}} = 0.25\ \mu s.$$

$$\frac{0.8\ \mu s}{0.25\ \mu s} = 3.2 \quad \Rightarrow \quad T_{AD} = 4\ T_{OSC}$$

$$\frac{2.5\ \mu s}{4 \times 0.25\ \mu s} = 2.5 \ \Rightarrow\ T_{ACQ} = 4\ T_{AD}$$

`ADCON0` =

0	0	1	0	0	0	0	1
−−		AN8				idle	on

= `0x21`

`ADCON1` =

0	0	0	0	0	1	1	0
−−		V_{REF}		AN8 analog			

= `0x06`

`ADCON2` =

1	0	0	1	0	1	0	0
right	−	T_{ACQ}			T_{AD}		

= `0x94`

$N = 12.7875\ T + 255.75$

T	−20	8	12	23	27	60
N	0	358	409	550	601	1023

Timer 0 Setup:

$$T_{cy} = \frac{4}{F_{OSC}} = 1\mu s$$

Sampling time = 100 ms

$$\text{count} = \frac{100\text{ms}}{1\mu s} = 100000 > 65535$$

$$100000 \div 65535 = 1.526 \quad \Rightarrow \quad 1:2 \text{ prescaler}$$

$$\text{count} = \frac{100\text{ms}}{2 \times 1\mu s} = 50000 = \text{C350}_H$$

$$10000_H - \text{C350}_H = \text{3CB0}_H$$

`T0CON` =

1	0	0	0	0	0	0	0
run	16-bit	timer	X	use prescaler	1:2		

= `0x80`

CCP2 Setup:

Prescaler 1:16

$$\texttt{PR2} = \frac{F_{OSC}}{4 \times F_{PWM} \times \text{Prescaler}} - 1 = \frac{4\text{MHz}}{4 \times 600\text{Hz} \times 16} - 1 = 104.17 - 1 = 103.17$$

$$\begin{aligned} \texttt{CCPR2L.CCP2CON<5:4>} &= (\text{On Time})\frac{F_{OSC}}{4 \times \text{Prescaler}} \\ &= \frac{0.50}{600\text{Hz}}\frac{4\text{MHz}}{4 \times 16} = 52.08 \approx 52.00 \end{aligned}$$

`CCPR2L` = 52

`CCP2CON` =

0	0	0	0	1	1	0	0
X		0.00		PWM			

= `0x0C`

`T2CON` =

0	0	0	0	0	0	1	0
X	unused				stop	1:16 pre	

= `0x02`

```
#include <xc.h>
#pragma config FOSC = INTOSC_HS
#pragma config CPP2MX = ON
#define H LATCbits.LATC1

void main() {
   int Tadc, Tavg, x;
   int T[15] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
   TRISBbits.TRISB2 = 1;   //AN8
   TRISCbits.TRISC1 = 0;   //PWM
   OSCCON = 0x62;
   ADCON0 = 0x21;
   ADCON1 = 0x06;
   ADCON2 = 0x94;
   TMR0H = 0x3C;
   TMR0L = 0xB0;
   T0CON = 0x80;
   PR2 = 103;
   CCPR2L = 52;
   CCP2CON = 0x00;   // start off
   T2CON = 0x02;
   H = 0;
   while(1) {
      if (INTCONbits.TMR0IF) {
         TMR0H = 0x3C;
         TMR0L = 0xB0;
         INTCONbits.TMR0IF = 0;
         ADCON0bits.GO = 1;
         while(ADCON0bits.GO);
         Tadc = ADRESH;
         Tadc = (Tadc << 8) + ADRESL;
         Tavg = 0;
         for(x=0; x<14; x++) {
            T[x] = T[x+1];
            Tavg = Tavg + T[x];
         }
         T[14] = Tadc;
         Tavg = (Tavg + T[14]) / 15;

         if (Tavg < 358) {
            T2CONbits.TMR2ON = 0;
            CCP2CON = 0x00;
            H = 1;
         }
```

```
            else if(!T2CONbits.TMR2ON && (Tavg>409) && (Tavg<550)){
               CCP2CON = 0x0C;
               T2CONbits.TMR2ON = 1;
            }
            else if (Tavg > 601) {
               T2CONbits.TMR2ON = 0;
               CCP2CON = 0x00;
               H = 0;
            }
         }
      }
   }
```

Problem: A toy car is controlled by a PIC18 microcontroller as follows:

- The motor of the car is controlled by the CCP2 module.
- Two switches are connected to the PIC18 to select the speed of the car:
  - `0x` : the car is stopped.
  - `10` : the car moves forward at half the maximum speed.
  - `11` : the car moves forward at maximum speed.
- The frequency of the CCP is set to 2kHz.

Configure the PIC such that F_{OSC} = 2 MHz internally, and write the XC8 program to control the car.

Chapter 7

Interrupts

References:

Datasheet : 9.0 Interrupts.

Mazidi et al. : Chapter 11 Interrupt Programming in Assembly and C.

Huang : Chapter 6 Interrupts, Resets and Configuration.

7.1 Interrupt Registers

- PIC18F4550 has multiple interrupt sources.
- Interrupt may be high-priority or low-priority.
- High-priority interrupts low-priority.
- High-priority / No priority function: `void interrupt high_isr(void)`{ }
- Low-priority function: `void interrupt low_priority low_isr(void)`{ }
- Ten registers: 80 bits total, 75 bits implemented, 30 bits in this course.

Registers:

- `RCON`: enable priority system, reset bits.
- `INTCON, INTCON2, INTCON3`: general enable, Timer 0, pin interrupts, RB.
- `PIR1, PIR2`: peripheral interrupt flags (`IF`).
- `PIE1, PIE2`: peripheral interrupt enable (`IE`).
- `IPR1, IPR2`: peripheral interrupt priority (`IP`).

Peripherals:

SPP	AD	RC	TX	SSP	CCP1	TMR2	TMR1
OSC	CM	USB	EE	BCL	HLVD	TMR3	CCP2

7.2 General Bits

`RCONbits.IPEN`:
`0`: disable priority levels.
`1`: enable priority levels.

`INTCONbits.GIE` / `INTCONbits.GIEH`:
without priority → enable/disable all interrupts.
with priority → enable/disable high priority interrupts.

`INTCONbits.PEIE` / `INTCONbits.GIEL`:
without priority → enable/disable peripheral interrupts.
with priority → enable/disable low priority interrupts (`GIE/GIEH` must be enabled).

Note: (`IF`) flags are set at interrupt condition regardless of (`IE`) state and must be reset by software.

7.3 INTx Pin Interrupt

- `INTEDGx`: `1` = rising edge, `0` = falling edge.
- `INT0` is always high priority.
- Configure related pins in port B as digital inputs (`TRISB` and `ADCON1`).

	INT0	INT1	INT2
Pin	RB0	RB1	RB2
Priority	–	INTCON3bits.INT1IP	INTCON3bits.INT2IP
Enable	INTCONbits.INT0IE	INTCON3bits.INT1IE	INTCON3bits.INT2IE
Flag	INTCONbits.INT0IF	INTCON3bits.INT1IF	INTCON3bits.INT2IF
Edge	INTCON2bits.INTEDG0	INTCON2bits.INTEDG1	INTCON2bits.INTEDG2

7.4 Timers, CCP2, AD

	TMR0	TMR2	CCP2	AD
IF	INTCON	PIR1	PIR2	PIR1
IE	INTCON	PIE1	PIE2	PIE1
IP	INTCON2	IPR1	IPR2	IPR1

e.g. `INTCONbits.TMR0IF`   `IPR2bits.CCP2IP`

7.5 Port B Interrupt-on-Change

- Pins `RB7, RB6, RB5, RB4`.
- Interrupt on any change in one or more pins.
- Configure related pins in port B as digital inputs (`TRISB` and `ADCON1`).
- `INTCONbits.RBIE`   `INTCONbits.RBIF`   `INTCON2bits.RBIP`
- After interrupt:
  1. read port B   2. wait 1 T_{cy} 3. reset `RBIF = 0`

`INTCON2bits.RBUP` (not needed for this course)
`0`: Pull-up resistors enabled.
`1`: disconnect pull-up resistors.
Disabled if output or reset.

7.6 Interrupt Example

Example: A PIC18 based frequency measurement system operates as follows:

- The frequency is measured two times every minute.
- The first rising edge starts the measurement, and the second rising edge ends the measurement.
- The input frequency range is 250 Hz to 500 Hz.
- If the input frequency is more than or equal to 390 Hz, a LED on `RA6` is turned on.

Configure the primary oscillator such that $F_{OSC} = 1$ MHz.
Oscillator Setup:
1 MHz primary, `RA6` is used ⇒ 4 MHz crystal.

$4\text{ MHz} \div 4 = 1\text{ Mhz without PLL} \quad \Rightarrow \quad$ use `ECIO_EC` mode.

Timer 0 Setup:

$\mathrm{T_{cy}} = \dfrac{4}{\mathrm{F_{OSC}}} = 4\mu s$

Measurement every 30 second

$\text{count} = \dfrac{30s}{4\mu s} = 7500000 > 65535$

$7500000 \div 65535 = 114.4 \quad \Rightarrow \quad 1:128 \text{ prescaler}$

$\text{count} = \dfrac{30s}{128 \times 4\mu s} = 58594 = \mathrm{E4E2_H}$

$\mathrm{10000_H - E4E2_H = 1B1E_H}$

T0CON =

1	0	0	0	0	1	1	0
run	16-bit	timer	X	use prescaler	1:128		

= `0x86`

Timer 2 Setup:

$\mathrm{T_{cy}} = \dfrac{4}{\mathrm{F_{OSC}}} = 4\mu s$

$\text{max period} = \dfrac{1}{\mathrm{F_{min}}} = 4\text{ ms}$

$\text{max count} = \dfrac{4ms}{4\mu s} = 1000 > 255$

$1000 \div 255 = 3.92 \quad \Rightarrow \quad 1:4 \text{ prescaler}$

$\text{min period} = \dfrac{1}{\mathrm{F_{max}}} = 2\text{ ms}$

$\text{min count} = \dfrac{2ms}{4 \times 4\mu s} = 125 \text{ (acceptable)}$

T2CON =

0	0	0	0	0	0	0	1
X	post not used				stop	1:4 pre	

= `0x01`

$\mathrm{F} = 390\text{ Hz} \rightarrow \mathrm{T} = 2.564\text{ ms} \rightarrow \text{count} = \dfrac{2564ms}{4 \times 4\mu s} = 160$

Use `INT0` (`RB`) for signal input, and `RB1` for LED output.

```
#include <xc.h>
#pragma config CPUDIV = OSC4_PLL6
#pragma config FOSC = ECIO_EC

unsigned char measure;

void main() {
   TRISB = 0x01;
   ADCON1 = 0x0F;
   OSCCON = 0x00;
   TMR0H = 0x1B;
   TMR0L = 0x1E;
   T0CON = 0x86;
   T2CON = 0x01;
```

```
    measure = 0;
    LATAbits.LATA6 = 0;
    RCONbits.IPEN = 0;
    INTCONbits.GIE = 1;
    INTCONbits.TMR0IE = 1;
    INTCONbits.INT0IE = 0;
    INTCON2bits.INTEDG0 = 1;
    while(1);
}

void interrupt ISR(void){
    if (INTCONbits.TMR0IF && INTCONbits.TMR0IE) {
        TMR0H = 0x1B;
        TMR0L = 0x1E;
        INTCONbits.TMR0IF = 0;
        INTCONbits.INT0IE = 1;
            INTCONbits.INT0IF = 0;
    }
    if (INTCONbits.INT0IF && INTCONbits.INT0IE) {
        INTCONbits.INT0IF = 0;
        if (!measure) {
            TMR2 = 0;
            T2CONbits.TMR2ON = 1;
            measure = 1;
        }
        else {
            LATAbits.LATA6 = (TMR2 <= 160);
            INTCONbits.INT0IE = 0;
            T2CONbits.TMR2ON = 0;
            measure = 0;
        }
    }
}
```

Problem: Write a program for a stop-watch using a PIC18 microcontroller. The stop-watch has two buttons:

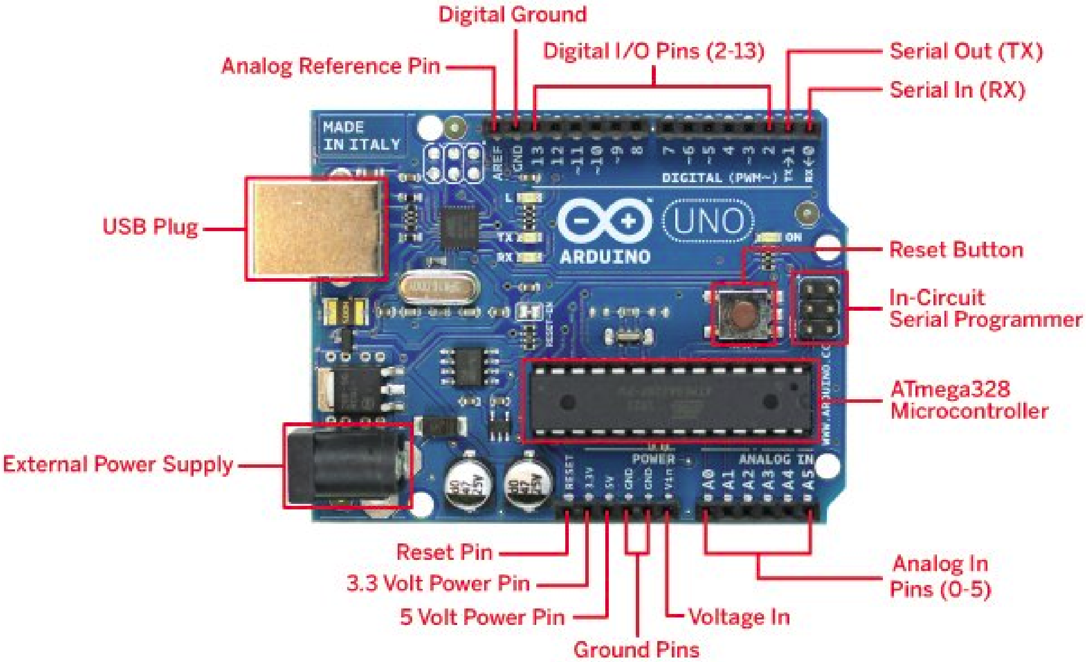
- Start/Stop button: start or stop time counting.
- Reset button: reset the time count to zero.

The stop-watch counts in tenths of a second, and the count is displayed in minutes and seconds (up to 255 minutes and 59 seconds). You must use interrupts for the buttons and timing.
(F_{OSC} = 8 MHz readily configured).

Chapter 8

Arduino

8.1 Arduino Uno



<u>**Structure:**</u>

```
void setup()
{
   // preparation
}
void loop()
{
   // read, process, write
}
```

Digital I/O:
`pinMode(`pin, mode`);`

- pin: `2` to `13` (pin `0` and pin `1` are used for serial I/O)
- mode: `INPUT`, `OUTPUT`, or `INPUT_PULLUP`

`digitalWrite(`pin, value`);`
value = `digitalRead(`pin`);`

- pin: the pin number
- value: `HIGH` or `LOW`

Pin `13` has LED connected.

Analog Input:
`analogReference(`type`);`

- type: `DEFAULT` (5V), `INTERNAL` (1.1V), or `EXTERNAL` (0 to 5V on `AREF` pin)

value = `analogRead(`pin`);`

- pin: 0 to 5
- value: int (0 to 1023)

toValue = `map(`fromValue, fromLow, fromHigh, toLow, toHigh`);`

$$\text{signed int toValue} = \frac{\text{toHigh} - \text{toLow}}{\text{fromHigh} - \text{fromLow}}(\text{fromValue} - \text{fromLow}) + \text{toLow}$$

PWM:
`analogWrite(`pin, value`);`

- pin: digital pins `3, 5, 6, 9, 10, 11`
- value: the duty cycle between 0 (always off) and 255 (always on)

pinMode() is not needed
Pins `3, 9, 10, 11` PWM frequency of 490 Hz.
Pins `5, 6` PWM frequency of 980 Hz. higher-than-expected duty cycles due to time functions.

Time:
`delay(`ms`);`
ms = `millis();`

- ms: milliseconds (unsigned long 0 to 4,294,967,295) `millis()` returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

Interrupts:
`attachInterrupt(interrupt, ISR, mode);`

- interrupt: `0` ≡ pin `2`   `1` ≡ pin `3`
- ISR: the function name to call when an interrupt occurs
- mode: interrupt condition `LOW`, `CHANGE`, `RISING`, `FALLING`.

Global variables modified by ISR should be `volatile`.
When one interrupt is running, the other is ignored.
Inside ISR function, `delay()` will not work and `millis()` will not increment.

8.2 Examples

Example: Eight LEDs are controlled by one push-button. When the push-button is pressed down, all LEDs are turned on, then one LED is turned off every 3 seconds. If the push-button is released before 7 seconds all the LEDs are turned off, otherwise the push-button is ignored until all LEDs are off.

Input: push-button at pin `2`
Outputs: LEDs at pins `3` to `13`

```
int pb, pb_old, ON, LED;
unsigned long TLED, TPB;

void setup() {
   pinMode(2, INPUT);
   for(int x=3; x<14; x++)
      pinMode(x, OUTPUT);
   pb_old = 0;
   ON = 0;
}

void loop() {
   pb = digitalRead(2);
```

```
    if(!pb_old && pb && !ON) {   // Rising edge
       for(int x=3; x<14; x++)
          digitalWrite(x,HIGH);
       LED = 3;
       ON = 1;
       TLED = millis();
       TPB = millis();
    }

    if(pb_old && !pb && ((millis()-TPB)<7000)) { // Falling edge
       for(int x=3; x<14; x++)
          digitalWrite(x,LOW);
       ON = 0;
    }

    if(ON && ((millis()-TLED)>3000)) {
       TLED = millis();
       digitalWrite(L, LOW);
       L++;
       if(L>13) ON = 0;
    }

    pb_old = pb;
 }
```

Example: A temperature measurement system reads the temperature once every 10 minutes. The sensor input range is $-10°$C to $70°$C, and the output range is 0V to 4V. Three LEDs are used to indicate the measured temperature as follows:

Temperature Range	LEDs On
$-10°$C to $10°$C	None
$11°$C to $30°$C	Dark
$31°$C to $50°$C	Dark and medium
$51°$C to $70°$C	Dark, medium and bright

Input: temperature sensor at pin `A0`.
Outputs: LEDs at pins `10` (dark), `11` (medium), `12` (bright).

```
int L1 = 10;    // PWM dark
int L2 = 11;    // PWM medium
int L3 = 12;    // Digital bright

unsigned long T;

void setup() {
   pinMode(12, OUTPUT);
   T = millis();
```

```
   analogReference(EXTERNAL);
}

void loop() {
   if(millis() > 600000) {
      T = millis();
      int Temp = analogRead(0);
      Temp = map(Temp, 0, 1023, -10, 70);
      if(Temp < 10) {
         analogWrite(L1, 0);
         analogWrite(L2, 0);
         digitalWrite(L3, LOW);
      }
      else if(Temp < 30) {
         analogWrite(L1, 85);   // 255x1/3=85
         analogWrite(L2, 0);
         digitalWrite(L3, LOW);
      }
      else if(Temp < 50) {
         analogWrite(L1, 85);
         analogWrite(L2, 170);   //255x2/3=170
         digitalWrite(L3, LOW);
      }
      else {
         analogWrite(L1, 85);
         analogWrite(L2, 170);
         digitalWrite(L3, HIGH);
      }
   }
}
```

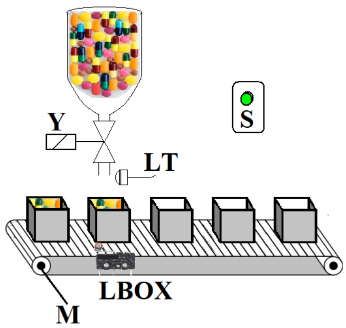
Example: A LED is is controller by a push-button and a potentiometer. When the push-button is pressed and released, the LED is on for 7 seconds. The LED light intensity is selected by the potentiometer (0 to 5V). Write the Arduino program.
Inputs: push-button `S` at pin `8`, potentiometer `P` at pin `A2`.
Output: LED `L` at pin `9`.
Variable: time `T`.

```
int S, P, L, S_old;
unsigned long T;

void setup() {
   pinMode(8, INPUT);
   S_old = 0;
}
```

```
void loop() {
   S = digitalRead(8);
   if(S_old && !S) {   // falling edge
      T = millis();
      while((millis()-T)<=7000) {
         P = analogRead(2);
         L = map(P, 0, 1023, 0, 255);
         analogWrite(9, L);
      }
      analogWrite(9, 0);
   }
   S_old = S;
}
```

Example: A medical tablets filling machine operates as follows:

- The switch S turns the machine on/off.
- The motor M is on until a box is under the tablet container detected by the limit switch LBOX.
- The motor is off and the valve Y opens to drop 5 tablets in the box. The tablets are detected by the proximity sensor LT.
- The motor moves the full box away from the container and a new cycle begins.
- The machine does not turn off until the box being filled is full and moved away.



Inputs: switch S at pin 2, proximity sensor LT at pin 3, limit switch LBOX at pin 4.
Output: motor M at pin 5, valve Y at pin 6.
Variables: ON, count.

```
int LBOX;
volatile int ON, count;

void setup() {
```

```
   pinMode(4, INPUT);
   pinMode(5, OUTPUT);
   pinMode(6, OUTPUT);
   attachInterrupt(0, onOff, RISING);
   attachInterrupt(1, addOne, RISING);
   M = 0;
   Y = 0;
   ON = 0;
   count = 0;
}

void loop() {
   while(!ON);
   digitalWrite(5, HIGH);   // Motor on
   while(!LBOX) LBOX = digitalRead(4);
   digitalWrite(5, HIGH);   // Motor off
   digitalWrite(6, HIGH);   // Valve on
   while(count <= 5);
   digitalWrite(6, HIGH);   // Valve off
   digitalWrite(5, HIGH);   // Motor on
   while(LBOX) LBOX = digitalRead(4);
   count = 0;
}

void onOff() {
   ON = !ON;
}

void addOne() {
   count++;
}
```

Problem: Write an Arduino program to control eight LEDs arranged in a circle using one push-button as follows:

- Initially, all the LEDs are off.
- If the push-button is pressed and held down, only one LED is on at a time and the on LED moves one step clockwise every 1.5 seconds.
- If the push-button is released, the on LED stops moving and turns off immediately.

Problem: A LED is operated using two push-buttons as follows:

- When the ON push-button is pressed for the first time, the LED turns on after 2 seconds with full brightness.
- If the ON push-button is pressed for the second time during the 2 seconds, the brightness will be 50%.
- If the ON push-button is pressed for the third time during the 2 seconds, the brightness will be 25%.
- The OFF push-button turns the LED off (or cancels the ON push-button effect during the 2-second period).

Write the Arduino program for this controller:
(a) by polling the two push-buttons (without interrupts).
(b) using interrupts for both push-buttons.