PIC Discussion
By Eng. Tamar Jomaa
Grades distribution:

- Homework: 5%
- Quizzes: 10%
- Searching: 5%
Chapter#1
Microcontroller Basics

Outlines:
1.1 Simple computer architecture
1.2 Instruction handling sequence
1.3 Pipelining
1.4 Microcontroller versus microprocessors
1.5 Harvard versus Von-Neumann’s
1.6 PIC 16F84A
1.7 Implementing your first application
1.1 Simple computer architecture

- Every computer system is designed around three basic circuits:
  - A central processing unit (CPU)
  - A memory
  - Input/output ports (I/O)

- The various circuits inside a typical computer are connected to one another through three sets of electrically conductive paths called busses, which are:
  - Data bus
  - Control bus
  - Address bus
1.1 Simple computer architecture

**Note** that both the data and the control buses are bidirectional but the address bus is unidirectional.
1.1 Simple computer architecture

CPU:

- The CPU, or so called microprocessor, is the brain of a computer.
- It performs all the arithmetic operations in the computer, does all the logic, and directs all the input and output.
- Major components of the CPU are:
  - some registers
  - a control unit
  - an arithmetic and logic unit (ALU)
1.1 Simple computer architecture

- The ALU does the actual computation or processing of data.

- The control unit controls the movement of data and instructions into and out of the CPU and controls the operation of the ALU.

- The CPU is connected to memory and peripheral I/O devices through a data bus, which carries data to and from the CPU; an address bus, which carries addresses out from the CPU; and a control bus, which carries coordinating signals among the various circuits.
The memory:

- It is used to hold user data and programs.
- It is composed of cells each of which contains a bit of information.
- Bits are organized into sets called memory locations or words, each of which is assigned a unique sequentially numbered address.
- A memory containing $2^n$ words has $n$ address lines. The size of word determines the size of the data bus of the memory.
- Therefore the size of the memory is determined by two parameters: the word width and the number of words.
1.1 Simple computer architecture

- One may imagine a memory as a building consisting of floors each of which has a number $2^n$, each floor from these floors contain a number of flats which equal the size of data bus.

- Example: assume a memory has a size 14x64, so we contain 64 floors begin from 0 to 63, therefore we need to 6 address lines to reach to each floor from these floors, because $2^6 = 64$.

  - the size of data bus is 14, so:
    - # of words(floors) = 64.
    - # of address lines = 6.
    - The size of each word = size of data bus = 14.
1.2 Instruction handling sequence

- A program is a sequence of instructions describing how to perform a certain task.

- While programs are written in different computer languages, they should be converted to the machine language before being executed.

- The machine language is the computer primitive instructions consisting of zeros and ones.

- Programs in machine language code are loaded into memory upon execution during which memory read, and sometimes write, cycles are encountered.
When the CPU is prepared to read an instruction or a byte of data from memory, it follows the following steps:

- It puts the address of that instruction or that byte of data on the address bus.
- Putting out an "active" signal on the memory READ line in the control bus.
- Memory responds by putting the contents of the addressed byte onto the data bus.
- The CPU reads the data bus.
1.2 Instruction handling sequence

How to write?

When the CPU is prepared to write a byte of data to memory, it is follow the following steps:

- It puts the address of the byte to be written on the address bus.
- It puts the contents of the byte on the data bus.
- It sends out a signal on the memory WRITE line of the control bus.
- Memory responds by recording the contents of the data bus at the memory location specified by the address bus.
1.2 Instruction handling sequence

**Note:**
In many cases, the CPU is faster than the memory; therefore a READY signal is used to stretch the memory cycle time. The CPU initiates a memory transaction, and then freezes until the READY indicates that it is safe to continue.
1.2 Instruction handling sequence

- The instruction pointer register (IP) holds the address of the instruction to be executed.

- The basic instruction processing cycle begins with a memory fetch or read in which the instruction machine code is loaded in a special register of the CPU called instruction register (IR).

- The instruction decoder in the control unit generates the hardware signals that insure the execution of the instruction.

- In summery, the IP points to the instruction to be executed. This instruction is fetched, decoded and executed.

- Execution of some instructions may generate memory read or write cycles.

- After processing an instruction the instruction pointer is updated to point to the next instruction to be processed.
1.2 Instruction handling sequence

Instruction execution steps:

1. Fetch an instruction
2. Decode
3. Execute
4. Generate READ or WRITE during execution
5. Upgrade Instruction Pointer
6. READ or WRITE
1.3 Pipelining

- Clock is microcontroller's main starter, and is obtained from an external component called an "oscillator".

- Internal circuit of a microcontroller divides the clock into four even clocks Q1, Q2, Q3, and Q4 which do not overlap.

- These four clocks make up one instruction cycle (also called machine cycle) during which one instruction is executed.

- Execution of instruction starts by calling an instruction that is next in string. Instruction is called from program memory on every Q1 and is written in instruction register on Q4.

- Decoding and execution of instruction are done between the next Q1 and Q4 cycles.
1.3 Pipelining

- On the following diagram we can see the relationship between instruction cycle and clock of the oscillator (OSC1) as well as that of internal clocks Q1-Q4.

- Program counter (PC) holds information about the address of the next instruction.
1.3 Pipelining

- Cycles of calling and executing instructions are connected in such a way that in order to make a call, one instruction cycle is needed, and one more is needed for decoding and execution.

- Due to pipelining, each instruction is effectively executed in one cycle.

- If instruction causes a change on program counter, and PC doesn't point to the following but to some other address (which can be the case with jumps or with calling subprograms), two cycles are needed for executing an instruction.

- Cycle of calling begins with Q1 clock, by writing into instruction register (IR). Decoding and executing begins with Q2, Q3 and Q4 clocks.
1.3 Pipelining

- **TCY0** reads in instruction MOVLW 55h.
- **TCY1** executes instruction MOVLW 55h and reads in MOVWF PORTB.
- **TCY2** executes MOVWF PORTB and reads in CALL SUB_1.
- **TCY3** executes a call of a subprogram CALL SUB_1, and reads in instruction BSF PORTA, BIT3.

All instructions are single cycle except for any program branches. These take two cycles since the fetch instructions is "flushed" from the pipeline while the new instruction is being fetched and then executed.
As this instruction (TCY3) is not the one we need, or is not the first instruction of a subprogram SUB_1 whose execution is next in order, instruction must be read in again. This is a good example of an instruction needing more than one cycle.

- **TCY4** instruction cycle is totally used up for reading in the first instruction from a subprogram at address SUB_1.

- **TCY5** executes the first instruction from a subprogram SUB_1 and reads in the next one.
1.4 Microcontroller versus microprocessors

- Microcontroller differs from a microprocessor in its functionality.

- In order for a microprocessor to be used, other components such as memory, or components for receiving and sending data must be added to it.

- Microcontroller is designed to be all of that in one. No other external components are needed for its application because all necessary peripherals are already built into it. Thus, we save the time and space needed to construct devices.

- In summary, microprocessor is included in the microcontroller with the memory and I/O devices, so microcontroller consists of microprocessor + memory + I/O devices.
In Harvard architecture, data bus and address bus are separate, but in Von-Neumann’s architecture these are merged.

The PIC microcontrollers do not use the conventional von Neumann architecture but use Harvard architecture. Harvard rose out of the need to speed up the work of a microcontroller.

Separating a program from data memory makes it further possible for instructions not to have to be 8-bit words.

It is also typical for Harvard architecture to have fewer instructions than von-Neumann's, and to have instructions usually executed in one cycle.
1.5 Harvard versus Von-Neumann’s

- Microcontrollers with Harvard architecture are also called "RISC Microcontrollers". RISC stands for Reduced Instruction Set Computer.

- Microcontrollers with von-Neumann's architecture are called 'CISC microcontrollers'. Title CISC stands for Complex Instruction Set Computer.
1.6 PIC 16F84A

- PIC is abbreviated to: Peripheral Interface Controller for example: PIC16F84A, where F = Flash.

- The 16F84A microcontroller is a simple microcontroller consists of the following modules:
  - An Arithmetic Logic Unit (ALU)
  - The control unit.
  - One or more working registers
  - Program memory (ROM) and data memory (RAM)
  - A program counter
  - An instruction registers with instruction decoder.
  - A stack
  - Timer/Counter.
  - Watchdog Timer.
  - I/O Ports.
An Arithmetic Logic Unit (ALU):

- The ALU is responsible for performing all arithmetic operations such as addition, subtraction and Boolean logical operations.

- The working registers are used by the ALU as temporary "scratchpad" memory, for example, for holding intermediate results of arithmetic operations.

The control unit:

- The control unit controls the timing and sequencing of all operations necessary to correctly schedule and execute instructions.
One or more working registers:
- called accumulators in the past
- for temporary storage during computations.
- A register is a small block of memory, often the size of a byte, where data is stored.

Program memory (ROM) and data memory (RAM):
- A microcontroller needs a memory to store its program in such a way that it will not be lost when the circuit’s power supply is switched off. This type of memory is called nonvolatile and is implemented as Read Only Memory (ROM) because the microcontroller can only read data from it.

- There are different types of ROM:
  - An EPROM (Erasable Programmable ROM).
  - An EEPROM is an electrically erasable PROM.
During program execution, the processor needs memory space where it can temporarily store and retrieve data.

This data is stored in a volatile memory known as Random Access Memory (RAM).

Information in RAM is lost if power to the memory circuits is removed.

A program counter:
- The program counter is a register used to store the address of the next instruction to be executed.
- Because the program consists of instructions stored sequentially in program memory, the address of the next instruction is obtained by simply incrementing the number (that is, the address), contained in the program counter.
An instruction registers with instruction decoder:

- The instruction register contains the actual binary instruction that needs to be executed.
- The instruction decoder takes the binary instruction and decodes it to determine what operation the instruction must perform and which data it must use.

A stack:

- The stack is an area of memory used to keep track of the contents of the program counter when subroutines are called.
- When data is written to the stack, it is stored at the ‘top’ of the stack. This operation is referred to as pushing data onto the stack.
- When data is removed from the top of the stack, the stack is said to be popped.
Timer/Counter:
- The timer/counter module can be used for either timing or counting operations.
- It can be configured to increment its value via clock pulses originating from the internal clock, or from an external source applied to the pin.
- Incrementing occurs on either the rising or falling edge of the input signal.

Watchdog Timer:
- A watchdog timer is an internal timer running independently of the system clock.
- It resets the device in the event of a program or circuit malfunction or if an unknown logical state is encountered.
- For example, if the program hangs, the watchdog timer will time out and reset the processor.
**1.6 PIC 16F84A**

*I/O Ports:*

- Ports provide PICs access to the outside world and are mapped to physical pins on the device.
We can simplify the previous figure in the figure below:
1.6 PIC 16F84A

Memory organization of PIC16F84A:

- **Memory**
  - **Data memory**
    - RAM (GPR) 68x8
  - **Program memory**
    - 1K x 14 bits
  - **Flash memory**
    - EEPROM 64 x8
The 16F84A program memory:
The figure shows three things:
- the Program Counter
- the Stack
- actual program memory
The 16F84A program memory:

- The program memory is loaded with the program code that the microcontroller executes.

- The program is in the form of a list of instructions, and the Program Counter holds the address of the next instruction.

- The value of the Program Counter can also be moved onto the Stack. This occurs when either a subroutine or an interrupt occurs. The instructions indicated in the diagram, CALL, RETURN, RETFIE and RETLW, all relate to subroutines and interrupts.

- With its 13-bit Program Counter, the microcontroller can theoretically address a range from 0000 to 1FFFH.

- The extra address space is shown (in grey), although it is of no use here.

- The very first location in the program memory is labelled the reset vector. When the program starts running for the first time for example on power-up, the Program Counter is set to 0000.

- The peripheral interrupt vector acts in a similar way for interrupt service routines is set to location 0004.
The 16F84A data and Special Function Register memory “RAM”:

- The RAM memory map is shown in Figure.
- The memory area is divided into two important areas:
  1) the general-purpose data memory, which occupies locations 0CH to 4FH.
  2) the Special Function Registers (SFRs), which occupies locations 00H to 0BH.
1.6 PIC 16F84A

- The Special Function Registers are used by the CPU and peripheral functions to control the device operation. These registers are static RAM.

<table>
<thead>
<tr>
<th>Bank 0</th>
<th>Bank 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>80h</td>
</tr>
<tr>
<td>01h</td>
<td>81h</td>
</tr>
<tr>
<td>02h</td>
<td>82h</td>
</tr>
<tr>
<td>03h</td>
<td>83h</td>
</tr>
<tr>
<td>04h</td>
<td>84h</td>
</tr>
<tr>
<td>05h</td>
<td>85h</td>
</tr>
<tr>
<td>06h</td>
<td>86h</td>
</tr>
<tr>
<td>07h</td>
<td>87h</td>
</tr>
<tr>
<td>08h</td>
<td>88h</td>
</tr>
<tr>
<td>09h</td>
<td>89h</td>
</tr>
<tr>
<td>0Ah</td>
<td>8Ah</td>
</tr>
<tr>
<td>0Bh</td>
<td>8Bh</td>
</tr>
</tbody>
</table>

Special function register
Now we explain some of Special function registers:

1) PORTA and TRISA Registers

- PORTA is a 5-bit wide, bi-directional port.
- The corresponding data direction register is TRISA.
- Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input.
- Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output.
- Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch.
- Pin RA4 is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin.
2) PORTB and TRISB Registers

- PORTB is an 8-bit wide, bi-directional port.
- The corresponding data direction register is TRISB.
- Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input.
- Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an input.
- Four of PORTB’s pins, B7:RB4, have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt on-change comparison).
3) STATUS Register (Address 03h, 83h)

- The STATUS register contains:
  - the arithmetic status of the ALU
  - the RESET status
  - the bank select bit for data memory

- As with any register, the STATUS register can be the destination for any instruction.
- If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to device logic.
- The TO and PD bits are not writable.
For example, CLRF STATUS will clear the upper three bits and set the Z bit. This leaves the STATUS register as 000u u1uu (where u = unchanged).

- Only the BCF, BSF, SWAPF and MOVWF instructions should be used to alter the STATUS register, because these instructions do not affect any status bit.

| bit 7-6 | Unimplemented: Maintain as ‘0’ |
| bit 5   | RP0: Register Bank Select bits (used for direct addressing) |
|        | 01 = Bank 1 (80h – FFh) |
|        | 00 = Bank 0 (00h - 7Fh) |
| bit 4  | T: Time-out bit |
|        | 1 = After power-up, CLRWDT instruction, or SLEEP instruction |
|        | 0 = A WDT time-out occurred |
| bit 3  | P: Power-down bit |
|        | 1 = After power-up or by the CLRWDT instruction |
|        | 0 = By execution of the SLEEP instruction |
| bit 2  | Z: Zero bit |
|        | 1 = The result of an arithmetic or logic operation is zero |
|        | 0 = The result of an arithmetic or logic operation is not zero |
| bit 1  | DC: Digit carry/borrow bit (ADDWF, ADDLW,SUBLW,SUBWF instructions) (for borrow, the polarity is reversed) |
|        | 1 = A carry-out from the 4th low order bit of the result occurred |
|        | 0 = No carry-out from the 4th low order bit of the result |
| bit 0  | C: Carry/borrow bit (ADDWF, ADDLW,SUBLW,SUBWF instructions) (for borrow, the polarity is reversed) |
|        | 1 = A carry-out from the Most Significant bit of the result occurred |
|        | 0 = No carry-out from the Most Significant bit of the result occurred |
4) **OPTION Register** (*Address 81h*):
- The OPTION register is a readable and writable register which contains various control bits to configure:
  - the TMR0/WDT prescaler
  - the external INT interrupt
  - TMR0
- the weak pull-ups on PORTB

5) **INTCON Register** (*Address 0Bh, 8Bh*):
- The INTCON register is a readable and writable register that contains the various enable bits for all interrupt sources.
The program counter (PC) specifies the address of the instruction to fetch for execution.

- The PC is 13 bits wide.
- The low byte is called the PCL register. This register is readable and writable.
- The high byte is called the PCH register. This register contains the PC<12:8> bits and is not directly readable or writable.
- If the program counter (PC) is modified or a conditional test is true, the instruction requires two cycles.
- The second cycle is executed as a NOP.
- All updates to the PCH register go through the PCLATH register.
The EEPROM data memory is readable and writable.
This memory is not directly mapped in the register file space. Instead it is indirectly addressed through the Special Function Registers.
There are four SFRs used to read and write this memory. These registers are:
• EECON1
• EECON2 (not a physically implemented register)
• EEDATA
• EEADR
EEDATA holds the 8-bit data for read/write
EEADR holds the address of the EEPROM location being accessed.
PIC16F84A devices have 64 bytes of data EEPROM with an address range from 0h to 3Fh.
Think….why there are two banks in PIC16F84A??

Solution:

- A problem with any memory space is that the larger the memory is, the larger the address bus must be.
- One way of avoiding big address buses is to divide the memory into a number of smaller blocks - called banks - each identical in size.
- Now a smaller address bus can be used. It can access all banks in an identical way, with just one of the banks being identified at any one time as the target of the address specified.
- PIC microcontrollers adopt a banked structure for their RAM, with the 16F84A having just two banks.
- The address of either bank is the 7-bit RAM address.
1.7 Implementing your first application

**Oscillator Types:**
- The PIC16F84A can be operated in four different oscillator modes.
- The user can program two configuration bits (FOSC1 and FOSC0) to select one of these four modes:
  - LP Low Power Crystal
  - XT Crystal/Resonator
  - HS High Speed Crystal/Resonator
  - RC Resistor/Capacitor

**RESET**
- The PIC16F84A differentiates between various kinds of RESET:
  - Power-on Reset (POR)
  - MCLR during normal operation
  - MCLR during SLEEP
  - WDT Reset (during normal operation)
  - WDT Wake-up (during SLEEP)
1.7 Implementing your first application

- The Figure describes a simple circuit for PIC16F84A:

Note: VDD and VSS are connected internally in Protus.
1.7 Implementing your first application

**.asm code:**

```
PROCESSOR P16F84A
    INCLUDE <P16F84A.INC>

ORG 0X00
GOTC START
ORG 0X04
GOTC START

START   BSEF STATUS,RPO
        CLRF TRISB
        BCF STATUS,RPO

LOOP   MOVLL 0XFF
        MOVWF PORTB
        GOTC LOOP

END
```
1.7 Implementing your first application

After loading the hex file in Protus:

Flasher
Assignment#1:

Connect the flasher circuit practically

*Remember:*

Prepare yourself to quiz in next lecture.
Be free to ask any question