RISC MCU Architecture
PIC16F877 Hardware

Outline

- Micriprocessor vs Microcontroller
- Introduction to PIC MCU
- PIC16F877 Hardware:
  - Program Memory
  - Data memory organization: banks, Special Function Registers (STATUS), General Function Registers, W register
  - Direct addressing and indirect addressing (FSR, INDF)
  - On-board Peripherals
- PIC16F877 Instruction Set:
  - bit (bsf, bcf)
  - byte (e.g. movlw, movf, addwf, subwf)
  - conditional branch (e.g. btfsc, btfss incfsz, decfsz)
  - goto
Microcontrollers vs. Microprocessors

A little History

- What is a computer?
  - [Merriam-Webster Dictionary] one that computes; specifically: programmable electronic device that can store, retrieve, and process data.
  - [Wikipedia] A computer is a machine that manipulates data according to a list of instructions.

- Classification of Computers (power and price)
  - Personal computers
  - Mainframes
  - Supercomputers
  - Dedicated controllers – Embedded controllers
Microcontrollers – Embedded Systems

- **An embedded system** is a special-purpose computer system designed to perform one or a few dedicated functions often with real-time
- An integrated device which consists of multiple devices
  - Microprocessor (MPU)
  - Memory
  - I/O (Input/Output) ports
- Often has its own dedicated software

A little about Microprocessor-based Systems …..
Evolution

- First came transistors
- Integrated circuits
  - SSI (Small-Scale Integration) to ULSI
  - Very Large Scale Integration circuits (VLSI)

- 1- Microprocessors (MPU)
  - Microcomputers (with CPU being a microprocessor)
  - Components: Memory, CPU, Peripherals (I/O)
  - Example: Personal computers

- 2- Microcontroller (MCU)
  - Microcomputers (with CPU being a microprocessor)
  - Many special function peripheral are integrated on a single circuit
  - Types: General Purpose or Embedded System (with special functionalities)

---

Microprocessor-Based Systems

- Central Processing Unit (CPU)
- Memory
- Input/Output (I/O) circuitry
- Buses
  - Address bus
  - Data bus
  - Control bus
The microprocessor (MPU) is a computing and logic device that executes binary instructions in a sequence stored in memory.

Characteristics:
- General purpose central processor unit (CPU)
- Binary
- Register-based
- Clock-driven
- Programmable

System software: A group of programs that monitors the functions of the entire system
So what are microcontrollers?

First Microcontrollers

- IBM started using Intel processors in its PC
  - Intel started its 8042 and 8048 (8-bit microcontroller) – using in printers
- Apple Macintosh used Motorola
- 1980 Intel abandoned microcontroller business
- By 1989 Microchip was a major player in designing microcontrollers
  - PIC: Peripheral Interface Controller
Embedded controllers

- Used to control smart machines
- Examples: printers, auto braking systems
- Also called microcontrollers or microcontroller units (MCU)

Embedded controllers
Software Characteristics

- No operating systems
- Execute a single program, tailored exactly to the controller hardware
- Assembly language (vs. High-level language)
  - Not transportable, machine specific
  - Programmer need to know CPU architecture
  - Speed
  - Program size
  - Uniqueness
Microcontroller Unit (MCU) - Block Diagram

- An integrated electronic computing and logic device that includes three major components on a single chip
  - Microprocessor
  - Memory
  - I/O ports
- Includes support devices
  - Timers
  - A/D converter
  - Serial I/O
  - Parallel Slave Port
- All components connected by common communication lines called the system bus.

MCU Architecture

- RISC (Harvard)
  - Reduced instruction set computer
  - Simple operations
  - Simple addressing modes
  - Longer compiled program bust faster to execute
  - Uses pipelining
- CISC (Von Neuman)
  - Complex instruction set computer
  - More complex instructions (closer to high-level language support)

Bench marks: How to compare MCUs together
MIPS: Million Instructions / second (Useful when the compilers are the same)
Microprocessors, Microcontrollers and DSPs

- Microprocessor is an “umbrella” term for all types of processor.
- Microcontrollers and DSPs evolved from the original microprocessors.

**Microcontrollers**
- Processor specifically designed for control applications.

**DSPs**
- Processors specifically designed for digital signal processing.

**Microprocessors**
- Processors for general purpose processing.

Main 8-bit Controllers

- **Microchip**
  - RISC architecture (reduced instruction set computer)
  - Has sold over 2 billion as of 2002
  - Cost effective and rich in peripherals

- **Motorola**
  - CISC architecture
  - Has hundreds of instructions
  - Examples: 68HC05, 68HC08, 68HC11

- **Intel**
  - CISC architecture
  - Has hundreds of instructions
  - Examples: 8051, 8052
  - Many difference manufacturers: Philips, Dallas/MAXIM Semiconductor, etc.

- **Atmel**
  - RISC architecture (reduced instruction set computer) –
  - Cost effective and rich in peripherals
  - AVR
Software: From Machine to High-Level Languages (1 of 3)

- **Machine Language**: binary instructions
  - All programs are converted into the machine language of a processor for execution
  - Difficult to decipher and write
  - Prone to cause many errors in writing

Software: From Machine to High-Level Languages (2 of 3)

- **Assembly Language**: machine instructions represented in mnemonics
  - Has one-to-one correspondence with machine instructions
  - Efficient in execution and use of memory; machine-specific and not easy to troubleshoot
Software: From Machine to High-Level Languages (3 of 3)

- High-Level Languages (such as BASIC, C, and C++)
  - Written in statements of spoken languages (such as English)
    - machine independent
    - easy to write and troubleshoot
    - requires large memory and less efficient in execution

Design Examples .....

Microcontrollers vs. Microprocessors
MPU-Based Time and Temperature System

MCU-Based Time and Temperature System
Embedded System
General Block Diagram

Introduction to PIC MCU
History of PIC Microcontroller

- In late 1970s, General Instrument had a 16-bit processor (CP1600) which was losing its market-share due to increased competition from Intel 8086 and Motorola 68000.
  - Main disadvantage of CP1600 was limited I/O capabilities
- As a solution General Instruments designed a support chip
  - A special purpose processor which was called the Peripheral Interface Controller (PIC) of the CP1600
- By mid 80s the industry found that PIC itself can be used for most control applications.
- General Instruments started a new subsidiary called Microchip which began to develop the PIC as a full featured microcontroller family.

Why PIC?

- Why PIC is popular?
  - low cost, wide availability with high clock speed
  - availability of low cost or free development tools
  - Only 37 instructions to remember
  - serial programming and re-programming with flash memory capability
  - Its code is extremely efficient, allowing the PIC to run with typically less program memory than its larger competitors
  - PIC is very small and easy to implement for non-complex problems and usually accompanies to the microprocessors as an interface
Two Different Architectures

- **Harvard Architectures**
  - Used mostly in RISC CPUs
  - Separate program bus and data bus: can be of different widths
  - For example, PICs use:
    - Data memory (RAM): a small number of 8-bit registers
    - Program memory (ROM): 12-bit, 14-bit or 16-bit wide (in EPROM, FLASH, or ROM)

- **Von-Neumann Architecture**
  - Used in: 8086 (CISC PCs)
  - Only one bus between CPU and memory
  - RAM and program memory share the same bus and the same memory, and so must have the same bit width
  - **Bottleneck**: Getting instructions interferes with accessing RAM

RISC vs. CISC

- **Reduced Instruction Set Computer (RISC)**
  - Used in: SPARC, ALPHA, PIC, Atmel AVR, etc.
  - Few instructions (usually < 50)
  - Only a few addressing modes
  - Executes 1 instruction in 1 internal clock cycle (Tcyc)

- **Complex Instruction Set Computer (CISC)**
  - Used in: 8086, 8051, 68HC11, etc.
  - Many instructions (usually > 100)
  - Several addressing modes
  - Usually takes more than 1 internal clock cycle (Tcyc) to execute
The PIC Family of Microcontrollers

- PIC series of microcontrollers offer a wide range of low cost devices, ranging from a tiny 8 pin device to a feature rich 40 pin device

  - E.g.
    - PIC16C54 18 pin Base line family
    - PIC16F84 18 pin Base line family
    - PIC16C74 28 pin Mid range family
    - PIC17C44 40 pin High end family

Family Core Architectural Differences

<table>
<thead>
<tr>
<th>Baseline Core Devices (ex:12C50x, 16C5x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 bit wide code memory,</td>
</tr>
<tr>
<td>tiny two level deep call stack,</td>
</tr>
<tr>
<td>33 instructions</td>
</tr>
<tr>
<td>PIC10, PIC12 &amp; PIC16 devices</td>
</tr>
<tr>
<td>6 pin to 40 pin packages.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mid-Range Core Devices (ex:12C50x, 16C5x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 bit wide code memory</td>
</tr>
<tr>
<td>improved 8 level deep call stack.</td>
</tr>
<tr>
<td>35 instructions</td>
</tr>
<tr>
<td>increased opcode width allows addressing of more memory</td>
</tr>
<tr>
<td>PIC12 and PIC16 devices.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PIC17 High End Core Devices (Ex:17C4x,17C7xx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>never became popular and superseded by the PIC18 architecture.</td>
</tr>
<tr>
<td>16 bit wide opcodes (allowing many new instructions) : 58 instructions</td>
</tr>
<tr>
<td>16 level deep call stack. Packages of 40 to 68 pins.</td>
</tr>
<tr>
<td>a memory mapped accumulator</td>
</tr>
<tr>
<td>read access to code memory (table reads)</td>
</tr>
<tr>
<td>direct register to register moves</td>
</tr>
<tr>
<td>an external program memory interface to expand the code space</td>
</tr>
<tr>
<td>an 8bit x 8bit hardware multiplier</td>
</tr>
<tr>
<td>auto-increment/decrement addressing</td>
</tr>
</tbody>
</table>
Family Core Architectural Differences ..

**PIC18 High End Core Devices (ex:18Cxxx)**
- new high end pic architecture
- It inherits most of the features and instructions of the 17 series,
- 77 instructions, much deeper call stack (31 levels deep)
- the call stack may be read and written
- offset addressing mode
- a new indexed addressing mode in some devices

**PIC24 and dsPIC 16 bit Microcontrollers**
- architectures differ significantly from prior models.
- *dsPICs* are Microchip’s newest family (started in 2004)
- digital signal processing capabilities.
- Microchip’s first inherent 16-bit (data) microcontrollers.
- hardware **MAC** (multiply-accumulate)
- **barrel shifting**
- bit reversal
- (16x16)-bit multiplication
- other digital signal processing operations.
- Can be efficiently programmed in C

---

Clock and Instruction Cycles

**Instruction Clock**

- Clock from the oscillator enters a microcontroller via OSC1 pin where 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. 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.
Clock/Instruction Cycle

- Clock from the oscillator enters the microcontroller via OSC1 pin.
- Internal circuit divides the clock into four even clocks Q1, Q2, Q3, and Q4 which do not overlap.
- These four clocks make up one instruction cycle during which one instruction is executed.
- 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.
Clock/Instruction Cycle

- Execution of instruction starts by calling an instruction that is next in string.

- **Fetch**
  - Instruction is called from program memory on every Q1 and is written in instruction register on Q4.

- **Execution**
  - Decoding and execution of instruction are done between the next Q1 and Q4 cycles.
  - Data memory is read during Q2 (operand read) and written during Q4 (destination write)

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

---

**Clock oscillator and instruction cycle**

**Table 2.3** PIC 16 Series instruction cycle durations for various clock frequencies

<table>
<thead>
<tr>
<th>Clock frequency</th>
<th>Instruction cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Frequency</td>
</tr>
<tr>
<td>20 MHz</td>
<td>5 MHz</td>
</tr>
<tr>
<td>4 MHz</td>
<td>1 MHz</td>
</tr>
<tr>
<td>1 MHz</td>
<td>250 kHz</td>
</tr>
<tr>
<td>32.768 kHz</td>
<td>8.192 kHz</td>
</tr>
</tbody>
</table>
Microcontroller oscillator generator circuits

(a) Resistor–capacitor (RC). (b) Crystal or ceramic

Three ways to provide the clock signal to a PIC
Exercise

For a system operating from a 4 MHZ crystal oscillator, every instruction would execute in how much time?

Solution

For a system operating from a 4 MHZ crystal oscillator, every instruction would execute in

\[
\frac{1}{(4 \text{Mhz/4})} = 1 \text{ micro-second}
\]
In the PIC16 there is a Reset input, MCLR (‘Master Clear’). As long as this is held low, the microcontroller is held in Reset. When it is taken high, program execution starts.

If the pin is taken low while the program is running, then program execution stops immediately and the microcontroller is forced back into Reset mode.

---

The PIC Family: Program Memory

- Technology: EPROM, FLASH, or ROM
- It varies in size from one chip to another.
  - examples:

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>12C508</td>
<td>512</td>
<td>12bit</td>
</tr>
<tr>
<td>16C711</td>
<td>1024 (1k)</td>
<td>14bit</td>
</tr>
<tr>
<td>16F877</td>
<td>8192 (8k)</td>
<td>14bit</td>
</tr>
<tr>
<td>17C766</td>
<td>16384 (16k)</td>
<td>16bit</td>
</tr>
</tbody>
</table>
The PIC Family: Program Memory

PICs have two different types of program storage:

1- EPROM (Erasable Programmable Read Only Memory)
   • Needs high voltage from a programmer to program (~13V)
   • Needs windowed chips and UV light to erase
   • Note: One Time Programmable (OTP) chips are EPROM chips, but with no window!
   • PIC Examples: Any ‘C’ part: 12C50x, 17C7xx, etc.

2- FLASH
   • Re-writable
   • Much faster to develop on!
   • Finite number of writes (~100k Writes)
   • PIC Examples: Any ‘F’ part: 16F84, 16F87x, 18Fxxx (future)

The PIC Family: Data Memory

- PICs use general purpose “File registers” for RAM (each register is 8bits for all PICs)
  - examples:

<table>
<thead>
<tr>
<th>PIC Code</th>
<th>RAM Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>12C508</td>
<td>25B RAM</td>
</tr>
<tr>
<td>16C71C</td>
<td>36B RAM</td>
</tr>
<tr>
<td>16F877</td>
<td>368B RAM + 256B of nonvolatile EEPROM</td>
</tr>
<tr>
<td>17C766</td>
<td>902B RAM</td>
</tr>
</tbody>
</table>
PIC Programming Procedure

- For example: in programming an embedded PIC featuring electronically erasable programmable read-only memory (EEPROM). The essential steps are:

  - Step 1: On a PC, type the program, successfully compile it and then generate the HEX file.

  - Step 2: Using a PIC device programmer, upload the HEX file into the PIC. This step is often called "burning".

  - Step 3: Insert your PIC into your circuit, power up and verify the program works as expected. This step is often called "dropping" the chip. If it isn't, you must go to Step 1 and debug your program and repeat burning and dropping.

Comparison of PIC families

<table>
<thead>
<tr>
<th>PIC family</th>
<th>Stack size (words)</th>
<th>Instruction word size</th>
<th>Number of instructions</th>
<th>Interrupt vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>12CXXX/12FXXX</td>
<td>2</td>
<td>12- or 14-bit</td>
<td>33</td>
<td>None</td>
</tr>
<tr>
<td>16C5XX/16F5XX</td>
<td>2</td>
<td>12-bit</td>
<td>33</td>
<td>None</td>
</tr>
<tr>
<td>16CXXX/16FXXX</td>
<td>8</td>
<td>14-bit</td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>17CXXX</td>
<td>16</td>
<td>16-bit</td>
<td>58, including hardware multiply</td>
<td>4</td>
</tr>
<tr>
<td>18CXXX/18FXXX</td>
<td>32</td>
<td>16-bit</td>
<td>75, including hardware multiply</td>
<td>2 (prioritised)</td>
</tr>
</tbody>
</table>
## Some members of the PIC 16 Series family

<table>
<thead>
<tr>
<th>Device number</th>
<th>No. of pins</th>
<th>Clock speed</th>
<th>Memory (K = Kbytes, i.e. 1024 bytes)</th>
<th>Peripherals/special features</th>
</tr>
</thead>
<tbody>
<tr>
<td>16F84A</td>
<td>18</td>
<td>DC to 20 MHz</td>
<td>1K program memory, 68 bytes RAM, 64 bytes EEPROM</td>
<td>1 8-bit timer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 5-bit parallel port</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 8-bit parallel port</td>
</tr>
<tr>
<td>16LF84A</td>
<td>As above</td>
<td>As above</td>
<td>As above</td>
<td>As above, with extended supply voltage range</td>
</tr>
<tr>
<td>16F84A-04</td>
<td>As above</td>
<td>DC to 4 MHz</td>
<td>As above</td>
<td>As above</td>
</tr>
<tr>
<td>16F873A</td>
<td>28</td>
<td>DC to 20 MHz</td>
<td>4K program memory, 192 bytes RAM, 128 bytes EEPROM</td>
<td>3 parallel ports,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3 counter/timers,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 capture/compare/PWM modules,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 serial communication modules,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5 10-bit ADC channels,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 analog comparators</td>
</tr>
<tr>
<td>16F874A</td>
<td>40</td>
<td>DC to 20 MHz</td>
<td>4K program memory, 192 bytes RAM, 128 bytes EEPROM</td>
<td>5 parallel ports,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3 counter/timers,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 capture/compare/PWM modules,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 serial communication modules,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8 10-bit ADC channels,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 analog comparators</td>
</tr>
</tbody>
</table>

*For DIP package only.

ADC: analog-to-digital converter; PWM: pulse width modulation.
MCU Features

- The range of microcontrollers now available developed because the features of the MCU used in any particular circuit must be as closely matched as possible to the actual needs of the application. Some of the main features to consider are:
  - Number of inputs and outputs.
  - Program memory size.
  - Data RAM size.
  - Nonvolatile data memory.
  - Maximum clock speed.
  - Range of interfaces.
  - Development system support.
  - Cost and availability.
- The PIC16F877A is useful as a reference device because it has a minimal instruction set but a full range of peripheral features.

Note: The general approach to microcontroller application design followed here is to develop a design using a chip that has spare capacity, then later select a related device that has the set of features most closely matching the application requirements.

PIC16F877 Hardware
PIC16F877A Features

High Performance RISC CPU:
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches, which are two-cycle
- Operating speed: DC - 20 MHz clock input DC - 200 ns instruction cycle

PIC16F877A Pin Layout
PIC Memory

- The PIC16F877A has an 8192 (8k) 14bit instruction program memory

- 368 Bytes Registers as Data Memory:
  - Special Function Registers: used to control peripherals and PIC behaviors
  - General Purpose Registers: used to a normal temporary storage space (RAM)

- 256 Bytes of nonvolatile EEPROM
PIC Program Memory

- The PIC16F877 8192 (8k) 14bit instructions

- Takes a max of 8 addresses, the ninth address will write over the first.

- When the controller is reset, program execution starts from here.

- If interrupted, program execution continues from here.

PIC Data Memory

- The most important registers have addresses in all the four banks.

- The data memory is divided into 4 memory banks.
## PIC16F877 Simplified File Register Map

<table>
<thead>
<tr>
<th>Bank 0 (000 – 07F)</th>
<th>Bank 1 (080 – 0FF)</th>
<th>Bank 2 (100-180)</th>
<th>Bank 3 (180-1FF)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Address</strong></td>
<td><strong>Register</strong></td>
<td><strong>Address</strong></td>
<td><strong>Register</strong></td>
</tr>
<tr>
<td>000h</td>
<td>Indirect</td>
<td>080h</td>
<td>Indirect</td>
</tr>
<tr>
<td>001h</td>
<td>Timer0</td>
<td>081h</td>
<td>Option</td>
</tr>
<tr>
<td>002h</td>
<td>PC Low</td>
<td>082h</td>
<td>PC Low</td>
</tr>
<tr>
<td>003h</td>
<td>Status Reg</td>
<td>083h</td>
<td>Status Reg</td>
</tr>
<tr>
<td>004h</td>
<td>File Select</td>
<td>084h</td>
<td>File Select</td>
</tr>
<tr>
<td>005h</td>
<td>Port A data</td>
<td>085h</td>
<td>Port A direction</td>
</tr>
<tr>
<td>006h</td>
<td>Port B data</td>
<td>086h</td>
<td>PortB direction</td>
</tr>
<tr>
<td>007h</td>
<td>Port C data</td>
<td>087h</td>
<td>PortC direction</td>
</tr>
<tr>
<td>008h</td>
<td>Port D data</td>
<td>088h</td>
<td>PortD direction</td>
</tr>
<tr>
<td>009h</td>
<td>Port E data</td>
<td>089h</td>
<td>PortE direction</td>
</tr>
<tr>
<td>00Ah</td>
<td>PC High</td>
<td>08Ah</td>
<td>PC High</td>
</tr>
<tr>
<td>00Bh</td>
<td>Interrupt Control</td>
<td>08Bh</td>
<td>Interrupt Control</td>
</tr>
<tr>
<td>00Ch to 01Fh</td>
<td>20 Peripheral</td>
<td>08Ch to 09Fh</td>
<td>20 Peripheral</td>
</tr>
<tr>
<td></td>
<td>Control Registers</td>
<td></td>
<td>Control Registers</td>
</tr>
<tr>
<td>020h to 06Fh</td>
<td>80 General Purpose</td>
<td>0A0h to 0EFh</td>
<td>80 General Purpose</td>
</tr>
<tr>
<td></td>
<td>Registers</td>
<td></td>
<td>Registers</td>
</tr>
<tr>
<td>070h to 07Fh</td>
<td>16 Common</td>
<td>0F0h to 0FFh</td>
<td>16 Common</td>
</tr>
<tr>
<td></td>
<td>Access GPRs</td>
<td></td>
<td>Access GPRs</td>
</tr>
<tr>
<td>07Fh</td>
<td>70h – 7Fh</td>
<td>1FFh</td>
<td>70h – 7Fh</td>
</tr>
</tbody>
</table>

### Register Addressing Modes

**Immediate Addressing:**

```
MOVlw H'0F'
```

**Direct Addressing:**

Uses 7 bits of 14-bit instruction to identify a register. 8th and 9th bits from the upper 2 bits of the STATUS register.

**Indirect Addressing:**

```
CLF FSR, STATUS ; clear FSR register
                 ; test if the 3rd bit of the STATUS register is set
                 ; clear next
```

**Example Program:**

```
CLRF INDF ; clear INDF register
CLRF FSR.L, FSR.H ; inc pointer
BTFSS STATUS, FSR.F,4  ; all done?
GOTO NEXT ; no clear next
```

**continued...**
PIC Family Control Registers

- Uses a series of “Special Function Registers” for controlling peripherals and PIC behaviors.

- **STATUS** ➔ Bank select bits, ALU bits (zero, borrow, carry)
- **INTCON** ➔ Interrupt control: interrupt enables, flags, etc.
- **OPTION_REG** ➔ contains various control bits to configure the TMR0 prescaler/WDT postscaler, the External INT Interrupt, TMR0 and the weak pull-ups on PORTB

Special Function Register
“STATUS Register”

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6-5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP0</td>
<td>RP1-2</td>
<td>TO</td>
<td>PD</td>
<td>Z</td>
<td>CY</td>
<td>RS7-0</td>
</tr>
</tbody>
</table>

- **RP0**: Register Bank Select bit (used for indirect addressing)
  - Bank 2: 0 (0000 - 0FFh)
  - Bank 0: 1 (1000 -FFh)
- **RP1**: Register Bank Select bits (used for direct addressing)
  - 0 = Bank 0 (0000 - 0FFh)
  - 1 = Bank 1 (0000 - 1FFh)
- **TO**: Timed-out bit
  - 1 = Timer0 or prescaler instruction, or compare instruction
  - 0 = A WDT time-out occurred
- **PD**: Power down bit
  - 1 = After power-up or by the ClearPD instruction
  - 0 = By execution of the CLRPD instruction
- **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
- **CY**: Carry/Borrow bit (adder, ACMP, ADC, PWM, TIMER instructions)
  - 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
- **RS7-0**: Register Select bits (accumulator, data, stack, instructions)
  - 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

Legend:
- **R**: Readable bit
- **W**: Writable bit
- **U**: Unimplemented bit, read as '0'
- **n**: Value at POR
- **1**: Bit is set
- **0**: Bit is cleared
- **?**: Bit is unknown
Special Function Register
“INTCON Register”

<table>
<thead>
<tr>
<th>RW-0</th>
<th>RW-0</th>
<th>RW-0</th>
<th>RW-0</th>
<th>RW-0</th>
<th>RW-0</th>
<th>RW-0</th>
<th>RW-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHE</td>
<td>PEIE</td>
<td>TMORE</td>
<td>INTE</td>
<td>RBIE</td>
<td>TRMOD</td>
<td>INTF</td>
<td>RBIF</td>
</tr>
</tbody>
</table>

bit 7
CHE: Global interrupt enable bit
0 = Enables all interrupts
1 = Enables all masked interrupts

bit 6
PEIE: Peripheral interrupt enable bit
0 = Enables all peripheral interrupts
1 = Enables all unmasked peripheral interrupts

bit 5
TMORE: TMRO0 overflow interrupt enable bit
0 = Enables the TMRO0 interrupt
1 = Disables the TMRO0 interrupt

bit 4
INTE: RB0INT external interrupt enable bit
0 = Enables the RB0INT external interrupt
1 = Enables the RB0INT external interrupt

bit 3
RBIE: RB Port change interrupt enable bit
0 = Enables the RB port change interrupt
1 = Disables the RB port change interrupt

bit 2
TMRF0: TMRO0 overflow interrupt flag bit
0 = TMRO0 register did not overflow
1 = TMRO0 register has overflowed (must be cleared in software)

bit 1
INTF: RB0INT external interrupt flag bit
0 = The RB0INT external interrupt did not occur
1 = The RB0INT external interrupt occurred (must be cleared in software)

bit 0
RBIF: RB Port change interrupt flag bit
0 = None of the RB0-RA7 pins have changed state
1 = At least one of the RB0-RA7 pins changed state; a mismatch condition will continue to set the bit. Reading POSIF will end the mismatch condition and allow the bit to be cleared (must be cleared in software)

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

PIC Peripherals

- Each peripheral has a set of SFRs to control its operation.
- Different PICs have different on-board peripherals.
- Some common peripherals are:
  - Tri-state ("floatable") digital I/O pins
  - Analog to Digital Converters (ADC) (8, 10 and 12bit, 50ksps)
  - Serial communications: UART (RS-232C), SPI, I^2C, CAN
  - Pulse Width Modulation (PWM) (10bit)
  - Timers and counters (8 and 16bit)
  - Watchdog timers, Brown out detect, LCD drivers
Peripheral Features

- 5 Digital I/O Ports
- Three timer/counter modules
  - Timer0: 8-bit timer/counter with 8-bit pre-scaler
  - Timer1: 16-bit timer/counter with pre-scaler, can be incremented during SLEEP via external crystal/clock
  - Timer2: 8-bit timer/counter with 8-bit period register, pre-scaler and post-scaler
- A 10-bit ADC with 8 inputs
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls

PIC Peripherals: Ports (Digital I/O)

- Ports are basically digital I/O pins which exist in all PICs
- The PIC16F877A have the following ports:
  - PORT A has 6 bit wide, Bidirectional
  - PORT B,C,D have 8 bit wide, Bidirectional
  - PORT E has 3 bit wide, Bidirectional
- Ports have 2 control registers
  - TRISx sets whether each pin is an input (1) or output (0)
  - PORTx sets their output bit levels or contain their input bit levels
- Pin functionality “overloaded” with other features
- Most pins have 25mA source/sink thus it can drive LEDs directly
- **WARNING:** Other peripherals SHARE pins!
I/O pin operation

The pin can be set for input or output data transfer

Connecting switches to logic inputs

(a) SPDT connection.  (b) SPST with pull-up resistor.  (c) SPST with pull-down resistor

For PIC microcontrollers, pull-up values in the range 10–100 kΩ are usually appropriate. The circuit of Figure (b) is very useful and widely applied, as many simple switches (e.g. PCB-mounting slide switches and push-buttons) are only available as SPST.
Driving LEDs from logic gates

For current source: \( V_{OH} = R I_D + V_D \)
\[ R = \frac{V_{OH} - V_D}{I_D} \]

For current sink: \( V_S = V_{OL} + R I_D + V_D \)
\[ R = \frac{V_S - V_D - V_{OL}}{I_D} \]

(a) Gate output sourcing current to LED.

(b) Gate output sinking current from LED

PIC Peripherals: Analogue to Digital Converter

- Only available in 14bit and 16bit cores
- \( F_s \) (sample rate) < 54KHz
- the result is a 10 bit digital number
- Can generate an interrupt when ADC conversion is done
PIC Peripherals: Analogue to Digital Converter

- The A/D module has four registers. These registers are:
  - A/D Result High Register (ADRESH)
  - A/D Result Low Register (ADRESL)
  - A/D Control Register0 (ADCON0)
  - A/D Control Register1 (ADCON1)
- Multiplexed 8 channel inputs
  - Must wait $T_{acq}$ to charge up sampling capacitor
- Can take a reference voltage different from that of the controller

ADC operation

The ADC converts an analog input into a binary code
PIC Peripherals: USART: UART

- Serial Communications Peripheral: Universal Synch./Asynch. Receiver/Transmitter
- Interrupt on TX buffer empty and RX buffer full

- **Asynchronous communication**: UART (RS-232C serial)
  - Can do 300bps - 115kbps
  - 8 or 9 bits, parity, start and stop bits, etc.
  - Outputs 5V so you need a RS232 level converter (e.g., MAX232)

---

**USART RS232**

Line drivers convert the signal to a bipolar, higher voltage

The data bits are timed from the falling edge of the start bit
PIC Peripherals: USART: UART

*Synchronous communication*: i.e., with clock signal

- **SPI = Serial Peripheral Interface**
  - 3 wire: Data in, Data out, Clock
  - Master/Slave (can have multiple masters)
  - Very high speed (1.6Mbps)
  - Full speed simultaneous send and receive (Full duplex)

- **I2C = Inter IC**
  - 2 wire: Data and Clock
  - Master/Slave (Single master only; multiple masters clumsy)
  - Lots of cheap I2C chips available; typically < 100kbps

PIC Peripherals: Timers

- Available in all PICs.
- generate interrupts on timer overflow.
- Some 8bits, some 16bits, some have prescalers and/or postscalers
- Can use external pins as clock in/clock out (i.e., for counting events or using a different Fosc)
Timer 0 Block Diagram

Special Function Register
OPTION_REG Register

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPFS</td>
<td>INTEDG</td>
<td>TOCS</td>
<td>TUSE</td>
<td>PSA</td>
<td>PS2:PS0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>TIMI Rate</th>
<th>WDT Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

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

- Capture/Compare/PWM (CCP)
- 10bit PWM width within 8bit PWM period (frequency)
  - Enhanced 16bit cores have better bit widths
- Frequency/Duty cycle resolution tradeoff
  - 19.5KHz has 10bit resolution
  - 40KHz has 8bit resolution
  - 1MHz has 1bit resolution (makes a 1MHz clock!)
- Can use PWM to do DAC - See AN655
- Capture counts external pin changes
- Compare will interrupt on when the timer equals the value in a compare register

PIC Peripherals: Misc.

- Sleep Mode: PIC shuts down until external interrupt (or internal timer) wakes it up.
- Interrupt on pin change: Generate an interrupt when a digital input pin changes state (for example, interrupt on keypress).
- Watchdog timer: Resets chip if not cleared before overflow
- Brown out detect: Resets chip at a known voltage level
- LCD drivers: Drives simple LCD displays
- Future: CAN bus, 12bit ADC, better analog functions

VIRTUAL PERIPHERALS:

- Peripherals programmed in software. UARTS, timers, and more can be done in software (but it takes most of the resources of the machine)
PIC16F877 Block Diagram

Most important register in the PIC must be involved in all arithmetic operations.

Instruction Memory

Data Memory

Instruction Bus

Data Bus

Brown-out: when the supplying voltage falls below a trip point (BV_{DD}).

This ensures that the device does not continue program execution outside the valid operation range of the device.

Typically used in AC line or large battery application where large loads maybe switched in and cause the device voltage to temporarily fall below the specified operating minimum.

Table:

<table>
<thead>
<tr>
<th>PIC 16F877</th>
<th>An address</th>
<th>128 bytes</th>
<th>256 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC 16F877A</td>
<td>An address</td>
<td>128 bytes</td>
<td>256 bytes</td>
</tr>
</tbody>
</table>

Notes:
- Higher limit data per the PIC16F877 datasheet.
PIC16 MCU Configuration

PIC16 MCU Configuration:
- Clock oscillator types
- Watchdog, power-up, brown-out timers
- Low-voltage programming
- Code protection
- In-circuit debug mode

When programming the PIC microcontroller, certain operational modes must be set prior to the main program download. These are controlled by individual bits in a special configuration register separated from the main memory block.

PIC16F877 Instruction Set
PIC16F877 Instruction Set

Literal and Control Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDW</td>
<td>Add literal to W</td>
<td>k + W → W</td>
</tr>
<tr>
<td>ANDW</td>
<td>AND literal and W</td>
<td>AND, W → W</td>
</tr>
<tr>
<td>CALL</td>
<td>Call subroutine</td>
<td>PC + 1 → TOSS, k → PC</td>
</tr>
<tr>
<td>CLRWDT</td>
<td>Clear Watchdog Timer</td>
<td>0</td>
</tr>
<tr>
<td>GOTO</td>
<td>Go to address</td>
<td>2</td>
</tr>
<tr>
<td>INCF</td>
<td>Increment W</td>
<td>1</td>
</tr>
<tr>
<td>CLRWF</td>
<td>Clear W from Reg</td>
<td>1</td>
</tr>
<tr>
<td>XORWF</td>
<td>Exclusive OR W with F</td>
<td>1</td>
</tr>
</tbody>
</table>

Notes:
- k: Literal or constant data to literal
- f: Register address (F) to literal
- d: Last address within a block of data
- s: Literal initial state of data in a literal
- L: Literal initial state of data in a literal
- EX: Restoring register (accumulator)
## Byte-Oriented Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>addw f,d</td>
<td>Add W and f</td>
<td>W ← W + f</td>
</tr>
<tr>
<td>andwf f,d</td>
<td>AND W and f</td>
<td>W ← W AND f</td>
</tr>
<tr>
<td>clrf f</td>
<td>Clear f</td>
<td>0 → f</td>
</tr>
<tr>
<td>clrw</td>
<td>Clear W</td>
<td>0 → W</td>
</tr>
<tr>
<td>comp f,d</td>
<td>Complement f</td>
<td>NOT f → d</td>
</tr>
<tr>
<td>decf f,d</td>
<td>Decrement f</td>
<td>f ← f - 1</td>
</tr>
<tr>
<td>decfz f,d</td>
<td>Decrement f, skip if zero</td>
<td>f ← f - 1</td>
</tr>
<tr>
<td>incf f,d</td>
<td>Increment f</td>
<td>f ← f + 1</td>
</tr>
<tr>
<td>incfz f,d</td>
<td>Increment f, skip if zero</td>
<td>f ← f + 1</td>
</tr>
<tr>
<td>iswf f,d</td>
<td>Inclusive OR W and f</td>
<td>W OR f ← d</td>
</tr>
<tr>
<td>movf f,d</td>
<td>Move f</td>
<td>f ← d</td>
</tr>
<tr>
<td>movwf f</td>
<td>Move W to f</td>
<td>W ← f</td>
</tr>
</tbody>
</table>

### No Operation

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>nop</td>
<td>No operation</td>
<td></td>
</tr>
<tr>
<td>rlf f,d</td>
<td>Rotate left f</td>
<td></td>
</tr>
<tr>
<td>rlf f,d</td>
<td>Rotate right f</td>
<td></td>
</tr>
<tr>
<td>subwf f,d</td>
<td>Subtract W from f</td>
<td>f ← W - d</td>
</tr>
<tr>
<td>swpft f,d</td>
<td>Swap halves l</td>
<td>(f:3) ← (f:7)</td>
</tr>
<tr>
<td>xorwf f,d</td>
<td>Exclusive OR W and f</td>
<td>W XOR f ← d</td>
</tr>
</tbody>
</table>

---

**Notes:**
- The table above lists byte-oriented instructions commonly used in microcontroller programming.
- Each instruction is followed by its mnemonic, description, and function.
- The table provides a clear overview of each instruction's purpose and its corresponding effect on the program's state.
Bit-Oriented Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcf</td>
<td>Bit clear f</td>
<td>0 ← f(0)</td>
</tr>
<tr>
<td>bcf</td>
<td>Bit set f</td>
<td>1 ← f(0)</td>
</tr>
<tr>
<td>btest, f</td>
<td>Bit test, skip next instruction if clear</td>
<td>skip if f(b) = 0</td>
</tr>
<tr>
<td>btest, f</td>
<td>Bit test, skip next instruction if set</td>
<td>skip if f(b) = 1</td>
</tr>
</tbody>
</table>

Key

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Bit address within an 8-bit file register</td>
</tr>
<tr>
<td>d</td>
<td>Destination select</td>
</tr>
<tr>
<td></td>
<td>d = 0 Store result in W</td>
</tr>
<tr>
<td></td>
<td>d = 1 Store result in file register f.</td>
</tr>
<tr>
<td></td>
<td>Default is d = 1.</td>
</tr>
<tr>
<td>f</td>
<td>Register file address (0x00 to 0xFFF)</td>
</tr>
<tr>
<td>k</td>
<td>Literal field, constant data or label</td>
</tr>
<tr>
<td>W</td>
<td>Working register (accumulator)</td>
</tr>
</tbody>
</table>

How is the instruction register loaded?

Program counter contains the address of the current instruction being executed. After reset, first instruction fetched from location 0x0000 in program memory.
Block diagram of the PIC 16 Series ALU

Programmer's Model
**Instruction Example:** \texttt{movlw \texttt{0xFF}}

Move ("mov") the number ("l" for "literal") \texttt{0xFF} - that's 256 in decimal - into the working register ("w"). In other words, load \texttt{W} with the value \texttt{0xFF}.

**Instruction Example:** \texttt{movwf PORTA}

Move ("mov") the working register ("w") into the file register ("f") named PORTA. In other words, load the register called PORTA with whatever number is in the W register.
Instruction Example: `movf PORTA, W`

MOVE ("mov") the the value of the file register ("f") named PORTA into the working register ("w").
In other words, load W with the whatever number is in PORTA.

Assembly Format

- First column: Labels
- Second column: opcodes and assembler directives
- Third Columns & more: operands

; This is a comments since it starts with a ";";
; This program puts out a square wave on PORTA Pin 0

```
clr PORTA ; Clear PORTA register
clr TRISA ; Make PORTA all outputs
Loop bsf PORTA,0 ; Turn on PORTA Pin 0
nop ; Match 'goto' delay
nop ; " "
bcf PORTA,0 ; Turn off PORTA Pin 0
goto Loop ; If not zero, loop back
```
Branches

- All branches are "Bit Tests"
- All branches only skip one instruction

; Set EqualFlag if PORTA = PORTB

```assembly
bcf    EqualFlag, 7 ; First, clear the flag
movf   PORTA, W ; Move PORTA -> W
subwf  PORTB, W ; W - PORTB -> W
btfsc  STATUS, Z ; Check Z bit (see STATUS)
bsf    EqualFlag, 7 ; Ports equal; set flag
```
Direct Addressing

- All file registers (RAM) are accessed by an address. This is called direct addressing.

- For example,

```
movlw 0xFF
movwf 0x06
```

loads W with FF, and then loads W into GPIO (address 0x06).

- Thankfully, we can use labels instead of addresses:

```
GPIO equ 0x06
movwf GPIO
```

Relative Addressing

- PCL = Low byte of the Program Counter
- Can be read and written.
- Writing to it sets the address of the next instruction to be executed.

```
PCH  PCL
12   11  10   8   7   0
PC    

2  PCLATH<4:3>
11 Opcode <10:0>

14bit core
```
Software: Relative Addressing

Example of Relative Addressing (using a table):

; Here’s a simple lookup table which is called as a
; subroutine. Expects the table offset to be loaded in W.
; An example call looks like this:
; movlw 0x04 ; Load W with 4
; call Table ; Call the table subroutine
; movwf Result ; Store the result from the table

Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>addwf PCL, W</td>
</tr>
<tr>
<td>000h</td>
<td>retlw 0x00</td>
</tr>
<tr>
<td>023h</td>
<td>retlw 0x23</td>
</tr>
<tr>
<td>033h</td>
<td>retlw 0x33</td>
</tr>
<tr>
<td>088h</td>
<td>retlw 0x88</td>
</tr>
</tbody>
</table>

Indirect Addressing

- Load indirect address into FSR
- Reading/Writing to INDF acts on address stored in FSR

Example code to clear 0x20 - 7F:

```asm
movlw 0x20
movwf FSR
loop clrf INDF
incf FSR,F
btfss FSR,7
goto loop
```
Banking

RAM in the PICs is banked, especially special function registers. Use the bank select commands to choose the bank.

Either:

bsf STATUS, RP0
bcf STATUS, RPO

Or use the assembler directive:

Banksel <registername>

Real Code!

- Note: Each PIC has a predefined “.h” file which contains labels for each special file register (so you don’t have to!)

- A working program requires initialization code and option codes set in the program. See .ASM examples for initialization code

- Please see Example.asm
### Constants and Syntax Used in Assembler Language

<table>
<thead>
<tr>
<th>Constant</th>
<th>Syntax</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>D\text{'} \text{decimal_number}' \text{'} \text{decimal_number}'</td>
<td>D'167' \text{'}167'</td>
<td>0x000000A7</td>
</tr>
<tr>
<td>Hexadecimal</td>
<td>H\text{'} \text{hexadecimal}' \text{'} \text{hexadecimal}'</td>
<td>H'A7' \text{'}A7'</td>
<td>0x000000A7</td>
</tr>
<tr>
<td></td>
<td>0x\text{hexadecimal} \text{hexadecimal}'</td>
<td>0xA7 \text{'}A7'</td>
<td>0A7H</td>
</tr>
<tr>
<td>Octal</td>
<td>O\text{'} \text{octal}' \text{'} \text{octal}'</td>
<td>O'247' \text{'}247'</td>
<td>0x000000A7</td>
</tr>
<tr>
<td>Binary</td>
<td>B\text{'} \text{binary}'</td>
<td>B'10100111' \text{'}10100111'</td>
<td>0x000000A7</td>
</tr>
<tr>
<td>ASCII</td>
<td>A\text{'} \text{ASCII_char}' \text{'ASCII_char}'</td>
<td>A'Z' \text{'Z'}</td>
<td>0x0000005A</td>
</tr>
</tbody>
</table>

*Note: The letters D, H, O, and B are used to indicate the type of constant. They can be written in lowercase or uppercase.*

---

### The program development process

1. Write source code
2. Assemble/compile
3. (Simulate)
4. Download
5. Test in hardware
Components of MPLAB development system

<table>
<thead>
<tr>
<th>Software tool</th>
<th>Tool function</th>
<th>Files produced or used</th>
<th>File description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text editor</td>
<td>Used to create and modify source code text file</td>
<td>PROGNAME.ASM</td>
<td>Source code text file</td>
</tr>
<tr>
<td>Assembler</td>
<td>Generates machine code from source code, reports syntax errors, generates list and symbol files</td>
<td>PROGNAME.HEX, PROGNAME.ERR, PROGNAME.LST, PROGNAME.COD</td>
<td>Executable machine code, Error messages, List file with source and machine code, Symbol and debug information</td>
</tr>
<tr>
<td>Simulator</td>
<td>Allows program to be tested in software before downloading</td>
<td>PROGNAME.HEX, PROGNAME.COD</td>
<td></td>
</tr>
<tr>
<td>Programmer</td>
<td>Downloads machine code to chip</td>
<td>PROGNAME.HEX</td>
<td></td>
</tr>
</tbody>
</table>

Assembler format

```
start bsf status,5 ; select memory bank 1
movlw B'00011000'; config pattern for port A
movwf trisa
movlw 53
```

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>COMMAND</td>
<td>Operand/s</td>
<td>Comment</td>
</tr>
<tr>
<td>Label equated to a value, or to indicate a program destination address for jump.</td>
<td>Mnemonic form of the instruction for the processor to carry out a specific operation. Only mnemonics specified in the instruction set may be used.</td>
<td>The data or register contents to be used in the instruction. Registers are usually represented by a label. Some instructions do not need an operand.</td>
<td>Explanatory text to the right of a semicolon on any line of code helps the programmer and user to understand the program. It has no effect on the operation of the program. Full line comments may also be used between program blocks.</td>
</tr>
</tbody>
</table>
All these instructions store the decimal value 167 in the W register:
- `movlw .167`
- `movlw 0a7h`
- `movlw 247O`
- `movlw b'10100111'`

Note how the hexadecimal constants must start with a digit in order to not be misunderstood as labels.

### Table 4.1 Some common MPASM Assembler directives

<table>
<thead>
<tr>
<th>Assembler directive</th>
<th>Summary of action</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Implement a listing option*</td>
</tr>
<tr>
<td>#include</td>
<td>Include additional source file</td>
</tr>
<tr>
<td>org</td>
<td>Set program origin</td>
</tr>
<tr>
<td>equ</td>
<td>Define an assembly constant; this allows us to assign a value to a label</td>
</tr>
<tr>
<td>end</td>
<td>End program block</td>
</tr>
</tbody>
</table>

*Listing options include setting of radix and of processor type.

### Table 4.2 Number representation in MPASM Assembler

<table>
<thead>
<tr>
<th>Radix</th>
<th>Example representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>D’255’</td>
</tr>
<tr>
<td>Hexadecimal</td>
<td>H’8d’ or 0x8d</td>
</tr>
<tr>
<td>Octal</td>
<td>O’574’</td>
</tr>
<tr>
<td>Binary</td>
<td>B’01011100’</td>
</tr>
<tr>
<td>ASCII</td>
<td>‘G’ or A’G’</td>
</tr>
</tbody>
</table>

### Example of Assembler code

<table>
<thead>
<tr>
<th>Label</th>
<th>Mnemonic</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Title</strong></td>
<td></td>
<td>&quot;Our first program&quot;</td>
</tr>
<tr>
<td>list</td>
<td>p=16f887</td>
<td></td>
<td>; processor type (directive)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(comment)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>; PROGRAM START</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(comment)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(comment)</td>
</tr>
<tr>
<td></td>
<td>org</td>
<td>0h</td>
<td>; startup address = 0000 (directive)</td>
</tr>
<tr>
<td>start</td>
<td>movlw</td>
<td>0x00</td>
<td>; simple code (instruction)</td>
</tr>
<tr>
<td></td>
<td>movwf</td>
<td>0x05</td>
<td>(instruction)</td>
</tr>
<tr>
<td></td>
<td>goto</td>
<td>start</td>
<td>; do this loop forever (instruction)</td>
</tr>
<tr>
<td></td>
<td>end</td>
<td></td>
<td>(directive)</td>
</tr>
</tbody>
</table>
## PIC Applications

### LED Flasher

```
Loop:
bsf PORTB, 0
call Delay_500ms
bcf PORTB, 0
call Delay_500ms
goto Loop
```

### Button Read

```
Movlw 0
movwf TRISD, f
bsf TRISD, 2
Loop:
bfsc PORTD, 2
goto light
bfc PORTD, 2
goto No_light
Light:
bsf PORTB, 0
goto Loop
No_light:
bsf PORTB, 0
goto Loop
```