HCMIU - DEE
Subject: ERTS

RISC MCU Architecture
PIC16F877 Hardware

1

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

2

Microcontrollers vs. Microprocessors

3

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

4

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

5

A little about Microprocessor-based Systems ……

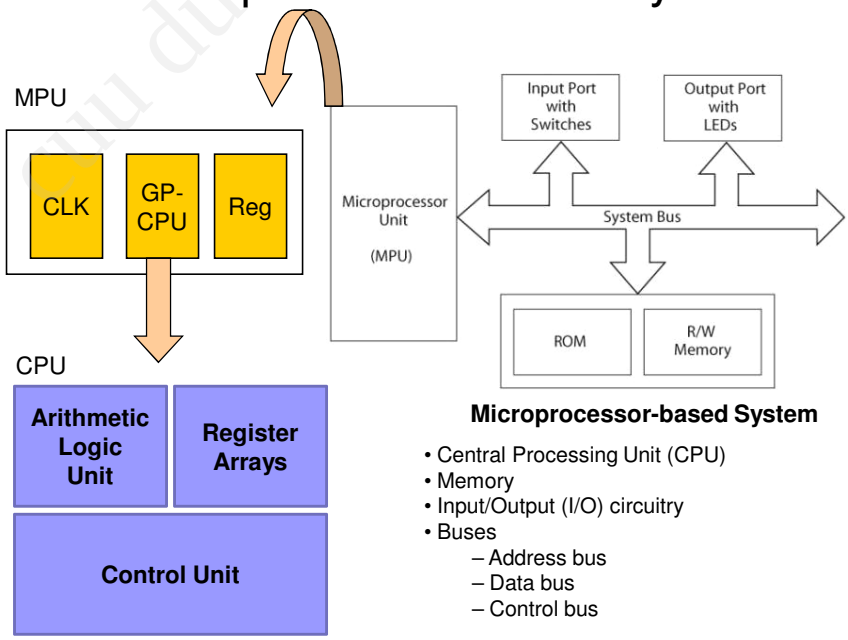
6

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)

7

Microprocessor-Based Systems

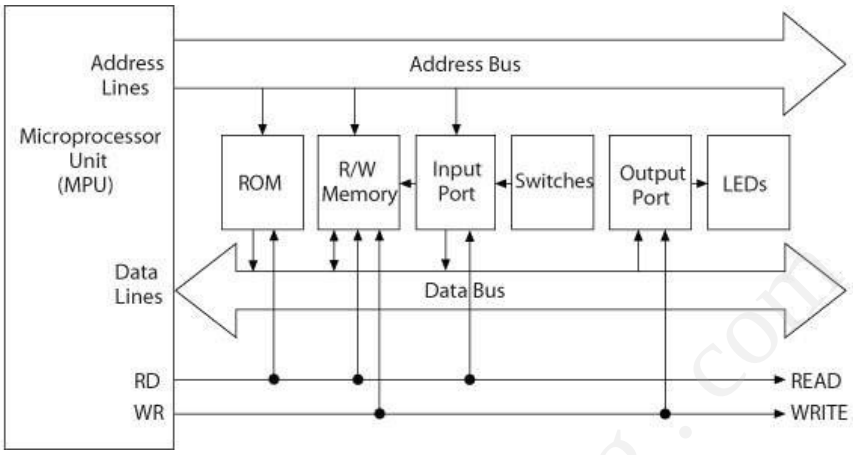


Microprocessor-based System

- Central Processing Unit (CPU)
- Memory
- Input/Output (I/O) circuitry
- Buses
  - Address bus
  - Data bus
  - Control bus

8

Microprocessor-Based System with Buses: Address, Data, and Control



9

Microprocessor-based Systems
Microprocessor

- 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

10

So what are microcontrollers?

11

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

12

Embedded controllers

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

13

Embedded controllers

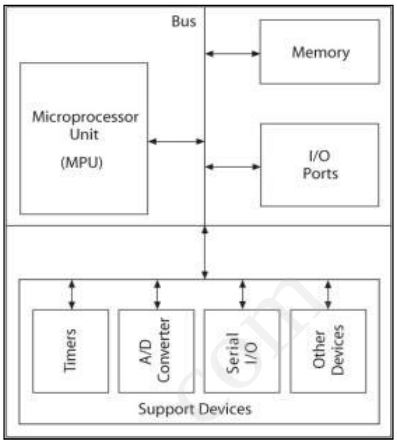
Software Characteristics

- 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

14

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.



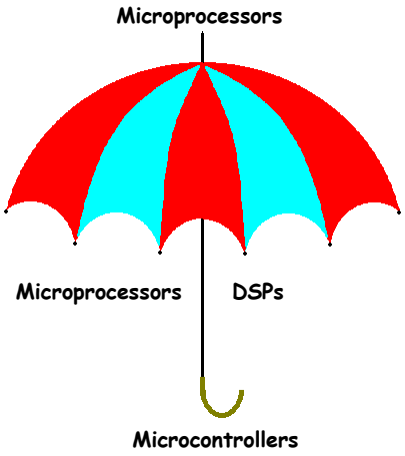
15

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.

17

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

18

High-level Language
Assembly Language
Machine Language

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

19

High-level Language
Assembly Language
Machine Language

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

20

High-level Language
Assembly Language
Machine Language

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

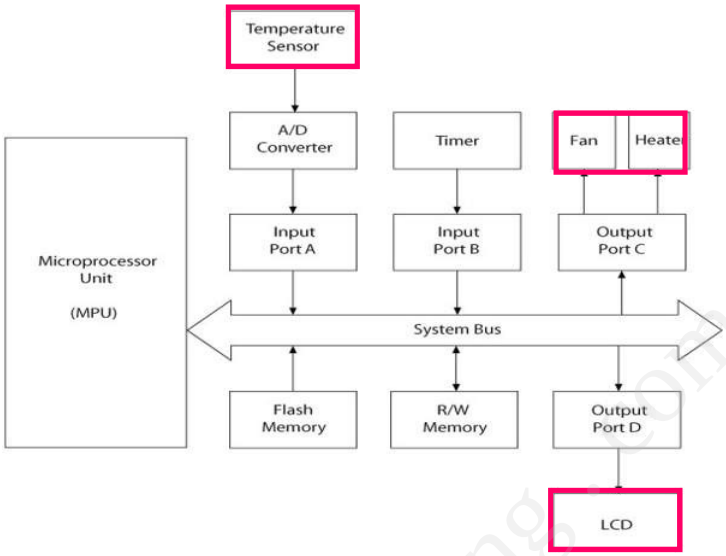
21

Design Examples …..

Microcontrollers vs. Microprocessors

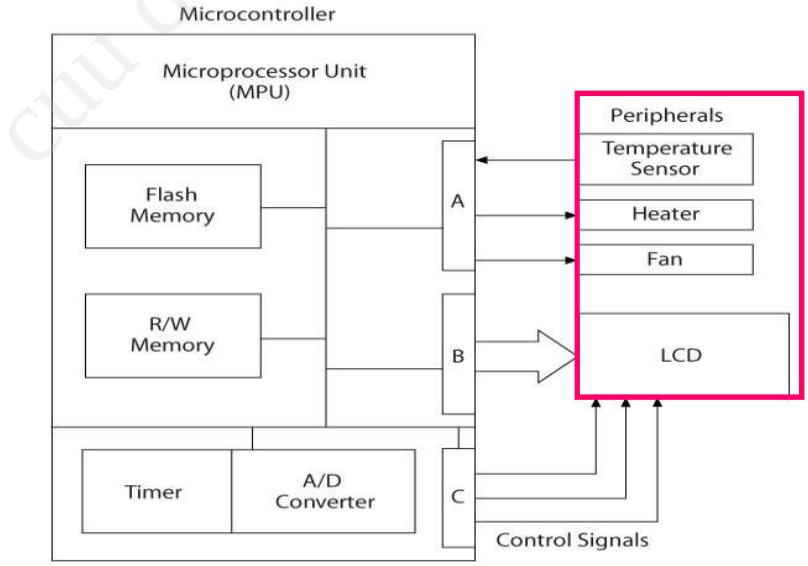
22

MPU-Based Time and Temperature System



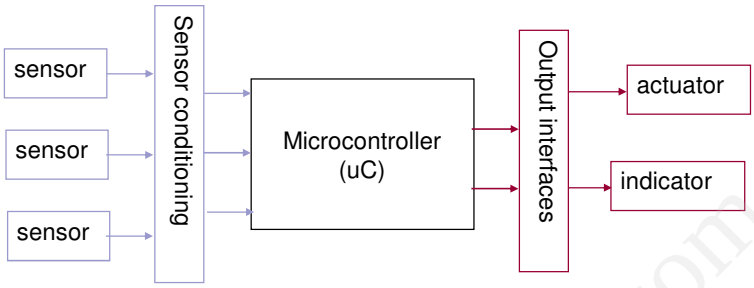
23

MCU-Based Time and Temperature System



24

Embedded System
General Block Diagram



25

Introduction to PIC MCU

26

History of PIC Microcontroller

- In late 1970s, General Instrument had a 16-bit processor (CP1600) which was loosing 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.

27

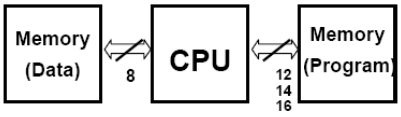
Why PIC?

- 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

28

Two Different Architectures

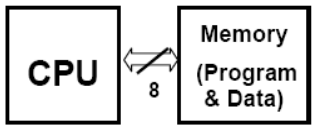
- ***Harvard Architectures***



Harvard Architecture

- 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 8bit registers
  - Program memory (ROM): 12bit, 14bit or 16bit wide (in EPROM, FLASH, or ROM)

- ***Von-Neumann Architecture***



Von-Neumann Architecture

- Used in: 80X86 (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

29

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: 80X86, 8051, 68HC11, etc.
  - Many instructions
    (usually > 100)
  - Several addressing modes
  - Usually takes more than 1 internal clock cycle (Tcyc) to execute

30

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

31

Family Core Architectural Differences

Baseline Core Devices
(ex:12C50x, 16C5x)
- 12 bit wide code memory,
- tiny two level deep call stack.
- **33 instructions**
- PIC10, PIC12 & PIC16 devices
- 6 pin to 40 pin packages.

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

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

32

Family Core Architectural Differences ..

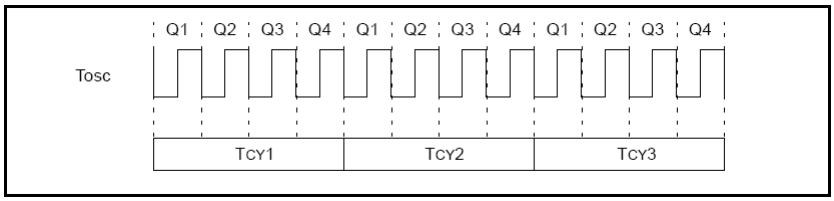
PIC18 High End Core Devices (ex:18Cxxx)
- new high end pic architecture
- It inherits most of the features and instructions of the 17 series,
- 77 instructions, much deeper call stack (31 levels deep)
- the call stack may be read and written
- offset addressing mode
- a new indexed addressing mode in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

33

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.

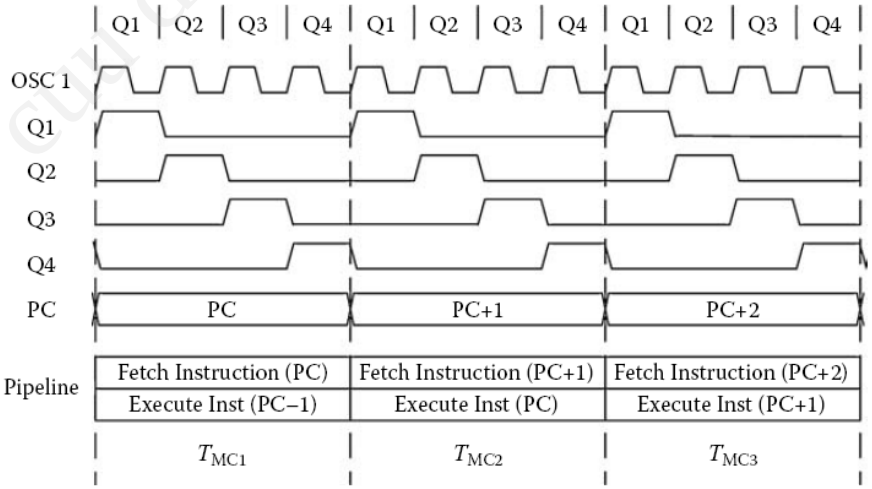


Clock/Instruction Cycle

- 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.

35

Clock/Instruction Cycle



Clock signals in PIC microcontrollers. OSC1 is the main oscillator from which the internal signals Q1, Q2, Q3, and Q4 are derived. These signals synchronize fetching, decode, and execute of instructions. T_{MC} is the duration of a machine cycle. It uses four OSC1 pulses.

36

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.

37

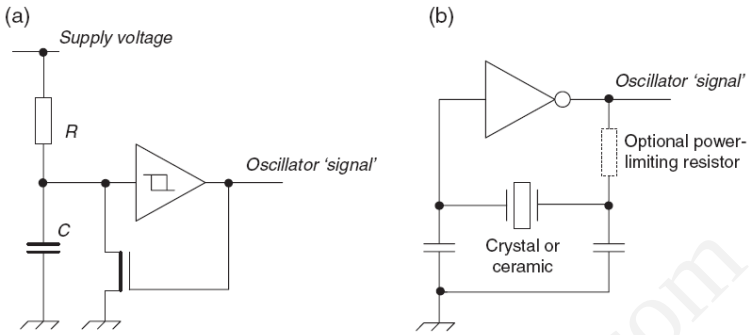
Clock oscillator and instruction cycle

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

Clock frequency	Instruction cycle	
	Frequency	Period
20 MHz	5 MHz	200 ns
4 MHz	1 MHz	1 µs
1 MHz	250 kHz	4 µs
32.768 kHz	8.192 kHz	122.07 µs

38

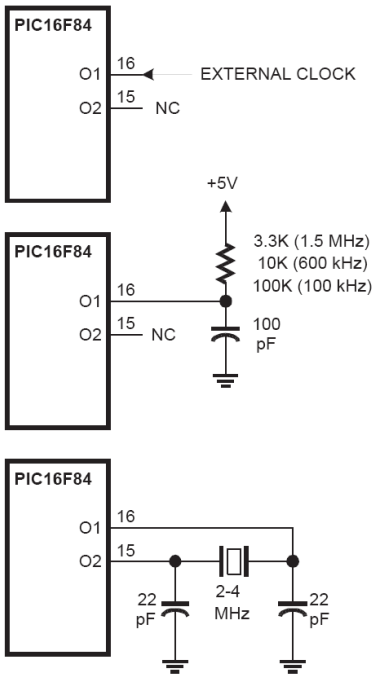
Microcontroller oscillator generator circuits



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

39

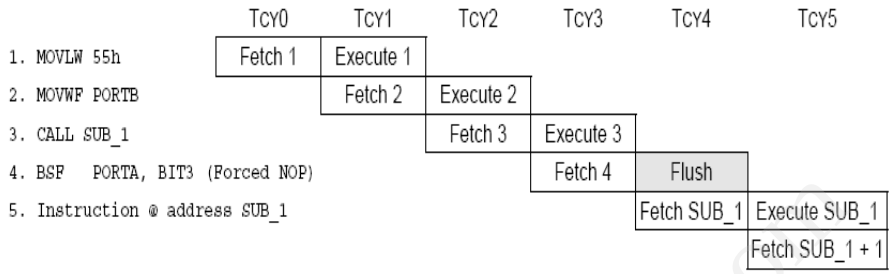
Three ways to provide the clock signal to a PIC



40

Pipelining in PIC

- Instruction Pipeline Flow



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

41

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

  1/(4Mhz/4) = 1 micro-second

42

Reset

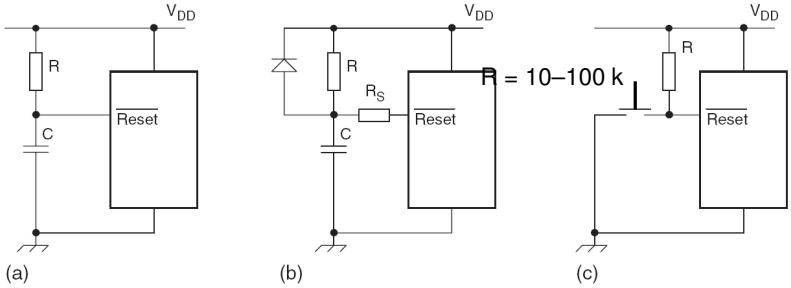


Figure 2.9 External Reset circuits – generic microcontroller with **Reset** input. (a) Power-on Reset, simplest possible. (b) Power-on Reset, with discharge diode and protective resistor. (c) User Reset button.

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.

43

The PIC Family: Program Memory

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

12C508	**512**	**12bit instructions**
16C711	**1024 (1k)**	**14bit instructions**
16F877	**8192 (8k)**	**14bit instructions**
17C766	**16384 (16k)**	**16bit instructions**

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)

45

The PIC Family: Data Memory

- PICs use general purpose "File registers" for RAM (each register is 8bits for all PICs)

  - examples:

12C508	25B RAM
16C71C	36B RAM
16F877	368B RAM + 256B of nonvolatile EEPROM
17C766	902B RAM

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

PIC family	Stack size (words)	Instruction word size	Number of instructions	Interrupt vectors
12CXXX/12FXXX	2	12- or 14-bit	33	None
16C5XX/16F5XX	2	12-bit	33	None
16CXXX/16FXXX	8	14-bit	35	1
17CXXX	16	16-bit	58, including hardware multiply	4
18CXXX/18FXXX	32	16-bit	75, including hardware multiply	2 (prioritised)

48

Some members of the PIC 16 Series family

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

49

Some members of the PIC 16 Series family ..

16F876A	28	DC to 20 MHz	8K program memory 368 bytes RAM, 256 bytes EEPROM	3 parallel ports, 3 counter/timers, 2 capture/compare/PWM modules, 2 serial communication modules, 5 10-bit ADC channels, 2 analog comparators
16F877A	40	DC to 20 MHz	8K program memory 368 bytes RAM, 256 bytes EEPROM	5 parallel ports, 3 counter/timers, 2 capture/compare/PWM modules, 2 serial communication modules, 8 10-bit ADC channels, 2 analog comparators

*For DIP package only.
ADC, analog-to-digital converter; PWM, pulse width modulation.

50

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.

51

PIC16F877 Hardware

52

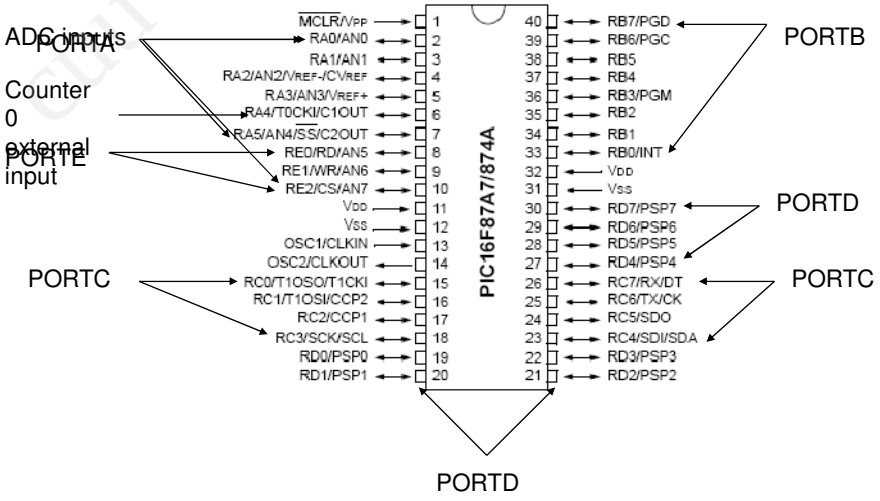
PIC16F877A Features

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

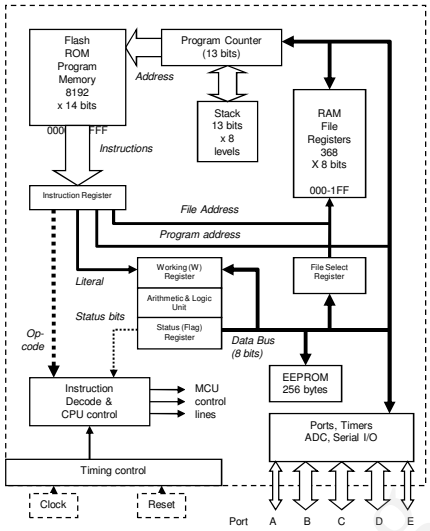
53

PIC16F877A Pin Layout



54

16F877 MCU Block diagram



Shows the main parts of the chip in simplified form

55

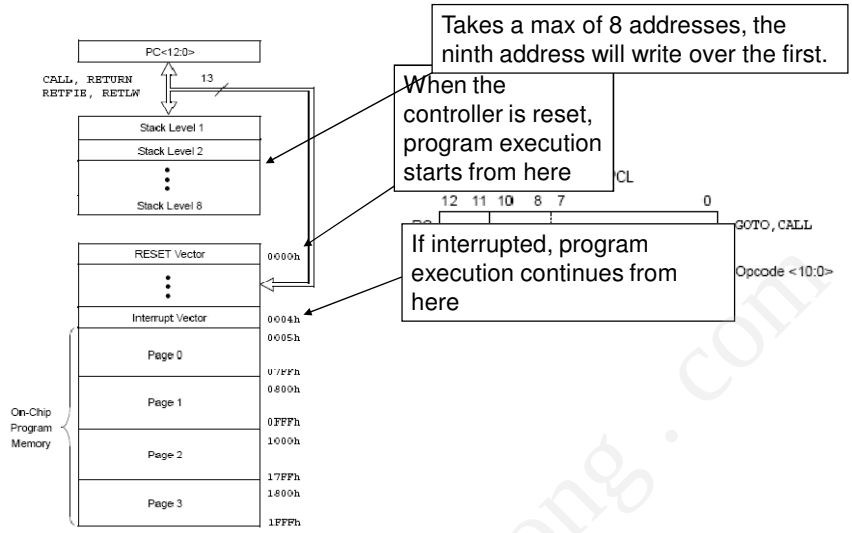
PIC Memory

- 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

56

PIC Program Memory

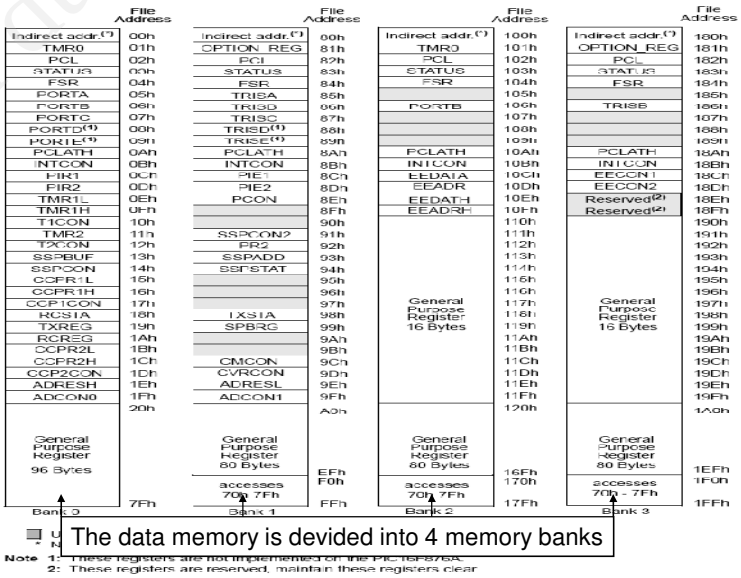
- The PIC16F877 8192 (8k) 14bit instructions



57

PIC Data Memory

The most important registers have addresses in all the four banks

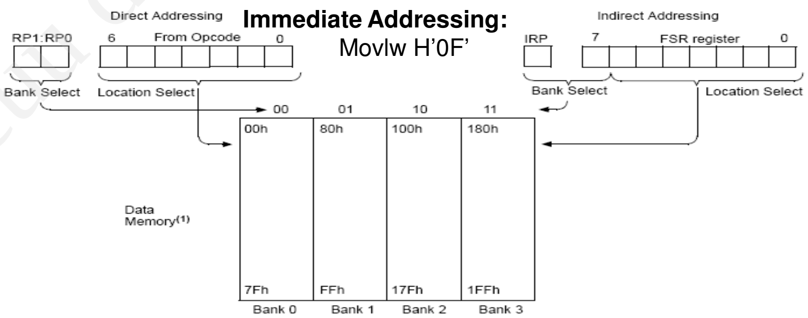


58

PIC16F877 simplified file register map

Bank 0 (000 – 07F)		Bank 1 (080 – 0FF)		Bank 2 (100-180)		Bank 3(180-1FF)	
Address	Register	Address	Register	Address	Register	Address	Register
000h	Indirect	080h	Indirect	100h	Indirect	180h	Indirect
001h	Timer0	081h	Option	101h	Timer0	181h	Option
002h	PC Low	082h	PC Low	102h	PC Low	182h	PC Low
003h	Status Reg	083h	Status Reg	103h	Status Reg	183h	Status Reg
004h	File Select	084h	File Select	104h	File Select	184h	File Select
005h	Port A data	085h	PortA direction	105h	-	185h	-
006h	Port B data	086h	PortB direction	106h	Port B data	186h	PortB direction
007h	Port C data	087h	PortC direction	107h	-	187h	-
008h	Port D data	088h	PortD direction	108h	-	188h	-
009h	Port E data	089h	PortE direction	109h	-	189h	-
00Ah	PC High	08Ah	PC High	10Ah	PC High	18Ah	PC High
00Bh	Interrupt Control	08Bh	Interrupt Control	10Bh	Interrupt Control	18Bh	Interrupt Control
00Ch to 01Fh	20 Peripheral Control Registers	08Ch to 09Fh	20 Peripheral Control Registers	10Ch to 10Fh	4 Peripheral Control Registers	18Ch to 18Fh	4 Peripheral Control Registers
020h to 06Fh	80 General Purpose Registers	0A0h to 0EFh	80 General Purpose Registers	110h to 16Fh	96 General Purpose Registers	190h to 1EFh	96 General Purpose Registers
070h to 07Fh	16 Common Access GPRs	0F0h to 0FFh	Accesses 70h – 7Fh	170h to 17Fh	Accesses 70h – 7Fh	1F0h to 1FFh	Accesses 70h – 7Fh

Register Addressing Modes



Immediate Addressing:

Movlw H'0F'

Direct Addressing:

Uses 7 bits of 14 bit instruction to identify a register file address 8th and 9th bit comes from RP0 and RP1 bits of STATUS register.

i.e. Z equ D'2' ; Z=2

btfss STATUS, Z ; test if the 3rd bit of the STATUS register is set

Indirect Addressing:

- Full 8 bit register address is written the special function register FSR
- INDF is used to get the content of the address pointed by FSR
- Exp : A sample program to clear RAM locations H'20' – H'2F:

```
         MOVLW 0x20 ;initialize pointer
         MOVWF FSR ;to RAM
NEXT     CLRF INDF ;clear INDF register
         INCF FSR,F ;inc pointer
         BTFSS FSR,4 ;all done?
         GOTO NEXT ;no clear next
CONTINUE
: ;yes continue
```

60

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

61

Special Function Register "STATUS Register"

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

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)

bit 6-5 **RP1:RP0:** Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4 **$\overline{TO}$:** Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 **$\overline{PD}$:** 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/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{borrow}$, 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/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF 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

Note: For $\overline{borrow}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high, or low order bit of the source register.

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

62

Special Function Register "INTCON Register"

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE**: Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **TMR0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software).
0 = None of the RB7:RB4 pins have changed state

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

63

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

64

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

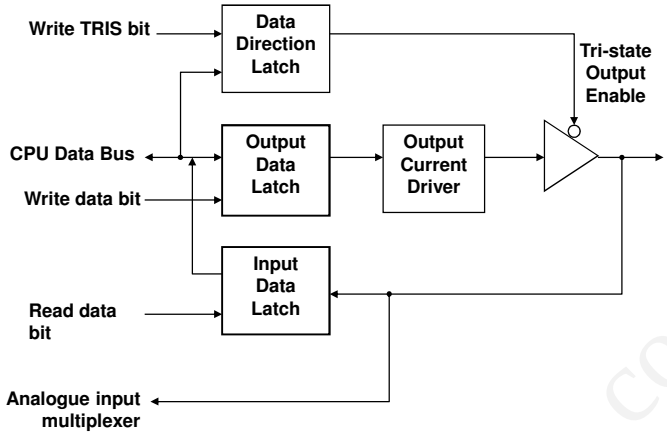
65

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!

66

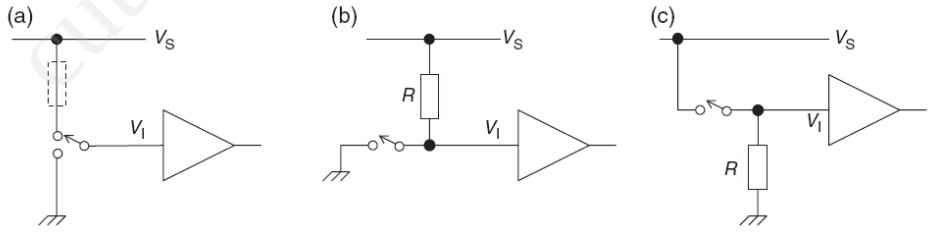
I/O pin operation



The pin can be set for input or output data transfer

67

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.

68

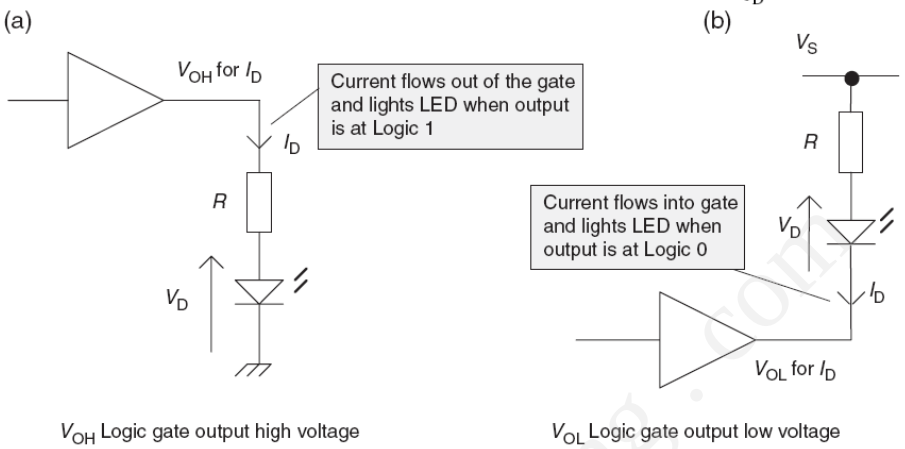
Driving LEDs from logic gates

For current source: $V_{OH} = RI_D + V_D$

$$R = \frac{V_{OH} - V_D}{I_D}$$

For current sink: $V_S = V_{OL} + RI_D + V_D$

$$R = \frac{V_S - V_D - V_{OL}}{I_D}$$



(a) Gate output sourcing current to LED.

(b) Gate output sinking current from LED

69

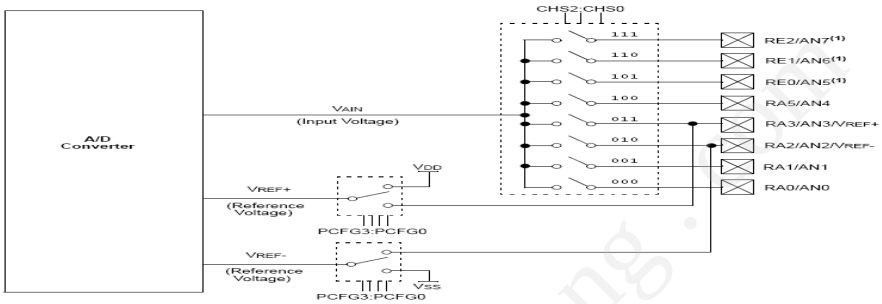
PIC Peripherals: Analogue to Digital Converter

- Only available in 14bit and 16bit cores
- Fs (sample rate) < 54KHz
- the result is a 10 bit digital number
- Can generate an interrupt when ADC conversion is done

70

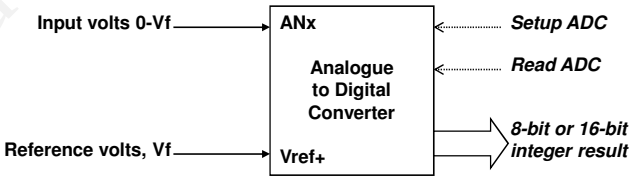
PIC Peripherals: Analogue to Digital Converter

- 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



71

ADC operation



The ADC converts an analog input into a binary code

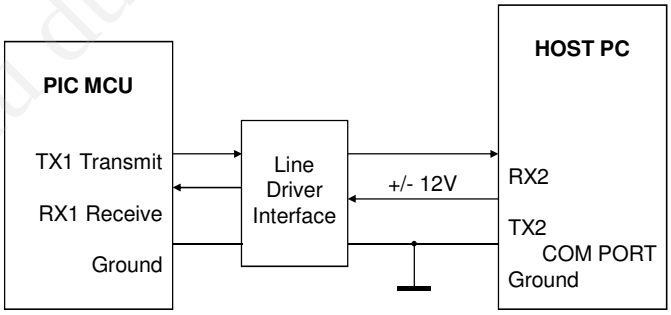
72

PIC Peripherals: USART: UART

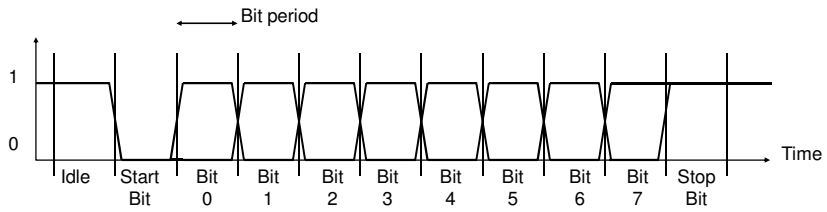
- Serial Communications Peripheral:
Universal Synch./Asynch. Receiver/Transmitter
- Interrupt on TX buffer empty and RX buffer full
- ***Asynchronous communication***: UART (RS-232C serial)
  - Can do 300bps - 115kbps
  - 8 or 9 bits, parity, start and stop bits, etc.
  - Outputs 5V so you need a RS232 level converter (e.g., MAX232)

73

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

74

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

75

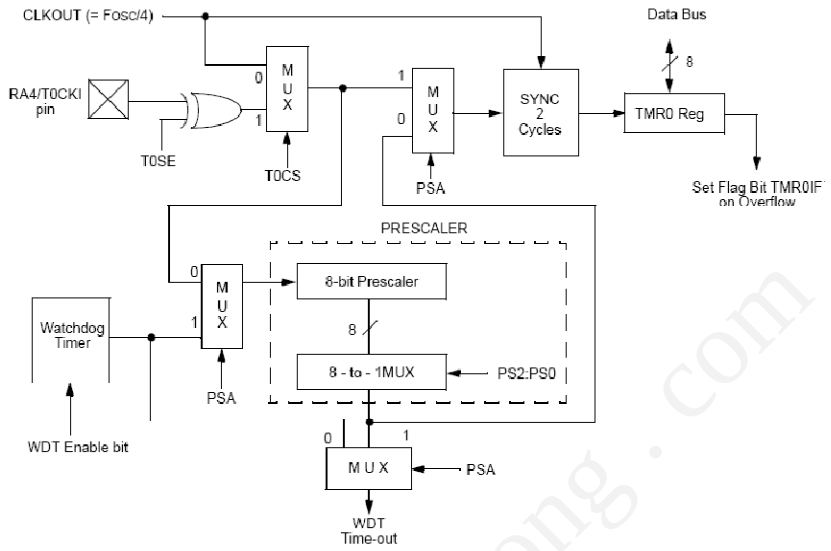
PIC Peripherals: Timers

- 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

(ie, for counting events or using a different Fosc)

76

Timer 0 Block Diagram



77

Special Function Register
OPTION_REG Register

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

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

78

PIC Peripherals: CCP Modules

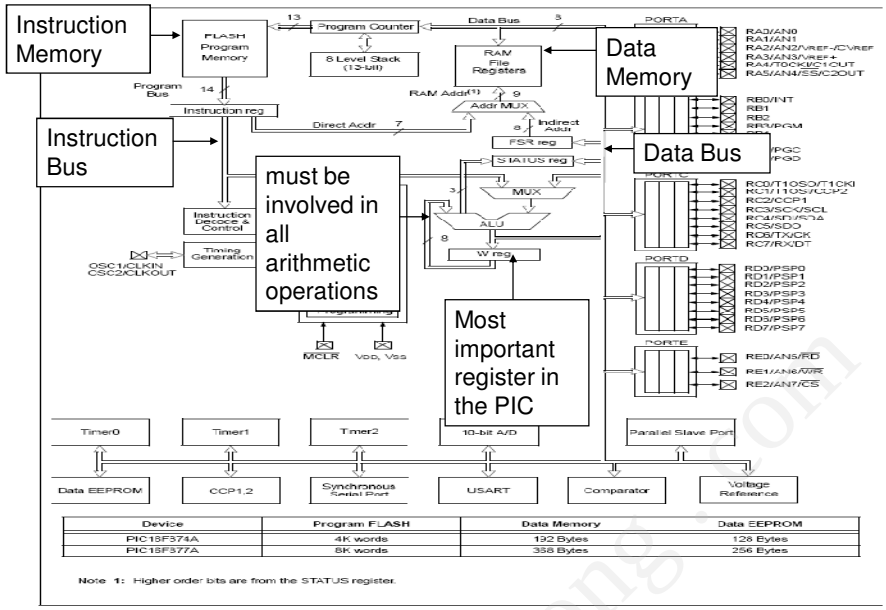
- Capture/Compare/PWM (CCP)
- 10bit PWM width within 8bit PWM period (frequency)
  - Enhanced 16bit cores have better bit widths
- Frequency/Duty cycle resolution tradeoff
  - 19.5KHz has 10bit resolution
  - 40KHz has 8bit resolution
  - 1MHz has 1bit resolution (makes a 1MHz clock!)
- Can use PWM to 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

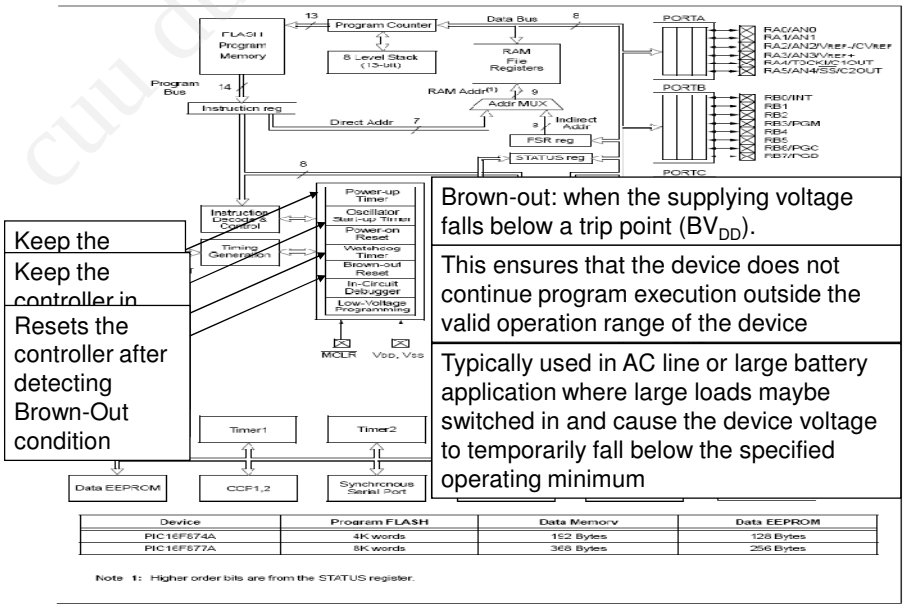


Device	Program FLASH	Data Memory	Data EEPROM
PIC18F374A	4K words	192 Bytes	128 Bytes
PIC18F377A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the STATUS register.

81

PIC16F877 Block Diagram



Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the STATUS register.

82

PIC16 MCU Configuration

PIC16 MCU Configuration:

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

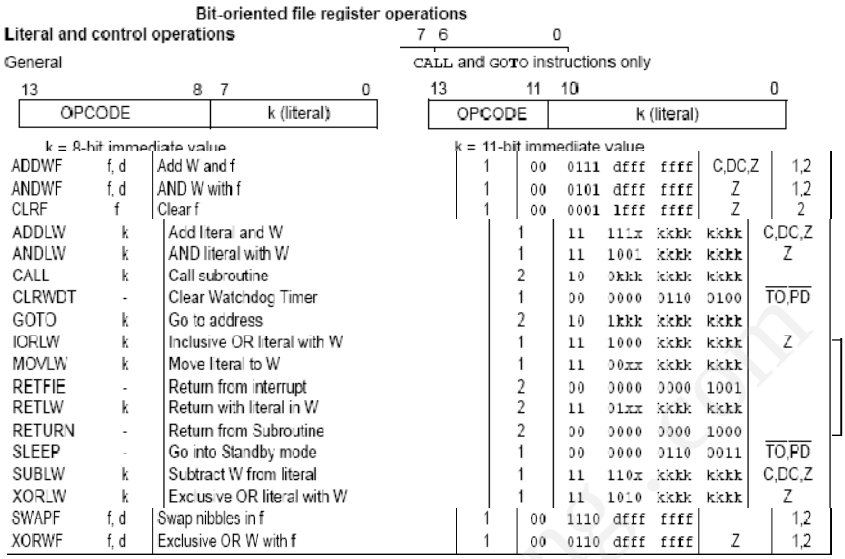
When programming the PIC microcontroller, certain operational modes must be set prior to the main program download. These are controlled by individual bits in a special configuration register separated from the main memory block.

83

PIC16F877 Instruction Set

84

PIC16F877 Instruction Set



ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2

85

Literal and Control Instructions

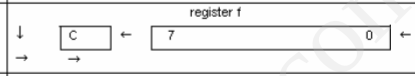
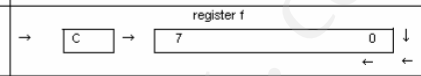
Mnemonic		Description	Function
addlw	k	Add literal to W	k + W → W
andlw	k	AND literal and W	k .AND. W → W
call	k	Call subroutine	PC + 1 → TOS, k → PC
clrwdt		Clear watchdog timer	0 → WDT (and prescaler if assigned)
goto	k	Goto address (k is nine bits)	k → PC (9 bits)
iorlw	k	Incl. OR literal and W	k .OR. W → W
movlw	k	Move Literal to W	k → W
option		Load OPTION register	W → OPTION Register
retfie		Return from Interrupt	TOS → PC, 1 → GIE
retlw	k	Return with literal in W	k → W, TOS → PC
return		Return from subroutine	TOS → PC
sleep		Go into Standby Mode	0 → WDT, stop oscillator
sublw	k	Subtract W from literal	K - W → W
tris	f	Configure port f (downward compat. instr.)	W → I/0 control reg f
xorlw	k	Exclusive OR literal and W	k .XOR. W → W

Key:

Field	Description		
b	Bit address within an 8-bit file register		
d	Destination select;	d = 0	Store result in W
		d = 1	Store result in file register f.
			Default is d = 1.
f	Register file address (0x00 to 0xFF)		
k	Literal field, constant data or label		
W	Working register (accumulator)		

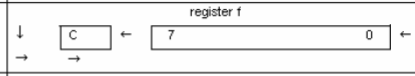
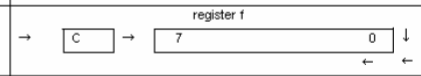
86

Byte-Oriented Instructions

Mnemonic		Description	Function
addwf	f,d	Add W and f	W + f → d
andwf	f,d	AND W and f	W .AND. f → d
clrf	f	Clear f	0 → f
clrw		Clear W	0 → W
comf	f,d	Complement f	.NOT. f → d
decf	f,d	Decrement f	f - → d
decfsz	f,d	Decrement f, skip if zero	f - 1 → d, skip if 0
incf	f,d	Increment f	f + 1 → d
incfsz	f,d	Increment f, skip if zero	f + 1 → d, skip if 0
iorwf	f,d	Inclusive OR W and f	W .OR. f → d
movf	f,d	Move f	f → d
movwf	f	Move W to f	W → f
nop		No operation	
rlf	f,d	Rotate left f	
rrf	f,d	Rotate right f	
subwf	f,d	Subtract W from f	f - W → d
swapf	f,d	Swap halves f	f(0:3) ↔ f(4:7) → d
xorwf	f,d	Exclusive OR W and f	W .XOR. f → d

87

Byte-Oriented Instructions

Mnemonic		Description	Function
addwf	f,d	Add W and f	W + f → d
andwf	f,d	AND W and f	W .AND. f → d
clrf	f	Clear f	0 → f
clrw		Clear W	0 → W
comf	f,d	Complement f	.NOT. f → d
decf	f,d	Decrement f	f - → d
decfsz	f,d	Decrement f, skip if zero	f - 1 → d, skip if 0
incf	f,d	Increment f	f + 1 → d
incfsz	f,d	Increment f, skip if zero	f + 1 → d, skip if 0
iorwf	f,d	Inclusive OR W and f	W .OR. f → d
movf	f,d	Move f	f → d
movwf	f	Move W to f	W → f
nop		No operation	
rlf	f,d	Rotate left f	
rrf	f,d	Rotate right f	
subwf	f,d	Subtract W from f	f - W → d
swapf	f,d	Swap halves f	f(0:3) ↔ f(4:7) → d
xorwf	f,d	Exclusive OR W and f	W .XOR. f → d

88

Bit-Oriented Instructions

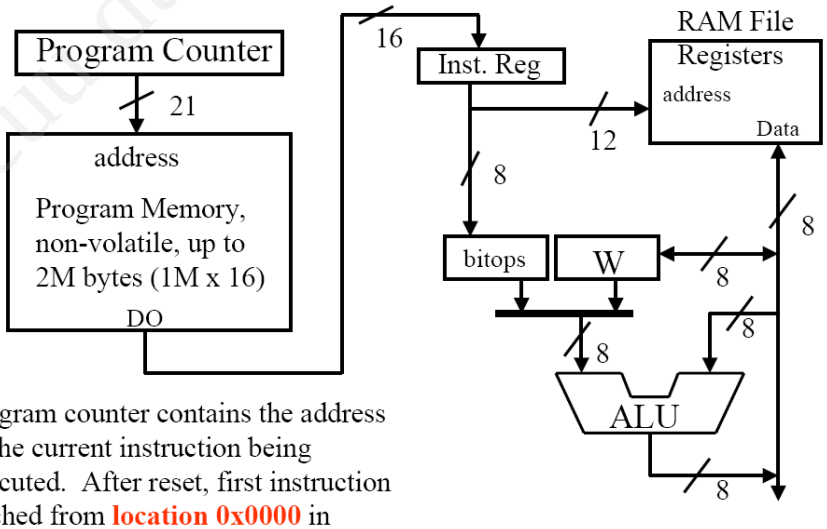
Mnemonic		Description	Function
bcf	f,b	Bit clear f	0 → f(b)
bsf	f,b	Bit set f	1 → f(b)
btfsc	f,b	Bit test, skip next instruction if clear	skip if f(b) = 0
btfss	f,b	Bit test, skip next instruction if set	skip if f(b) = 1

Key:

Field	Description		
b	Bit address within an 8-bit file register		
d	Destination select;	d = 0	Store result in W
		d = 1	Store result in file register f.
			Default is d = 1.
f	Register file address (0x00 to 0xFF)		
k	Literal field, constant data or label		
W	Working register (accumulator)		

89

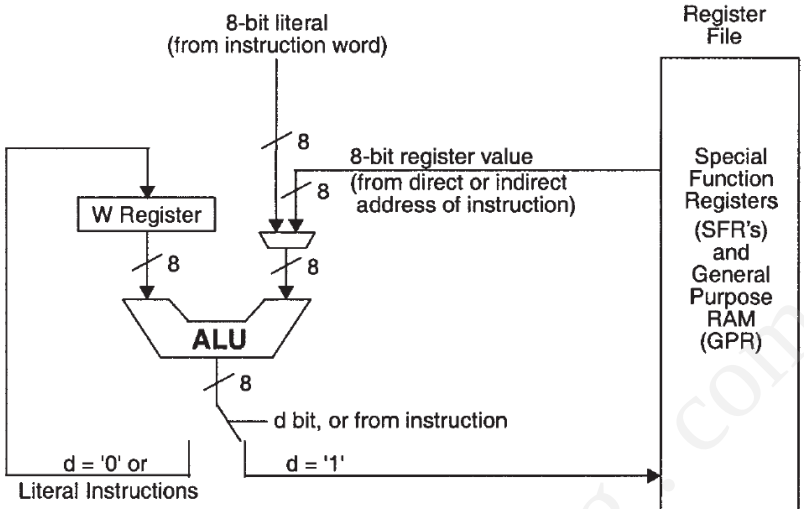
How is the instruction register loaded?



Program counter contains the address of the current instruction being executed. After reset, first instruction fetched from **location 0x0000** in program memory.

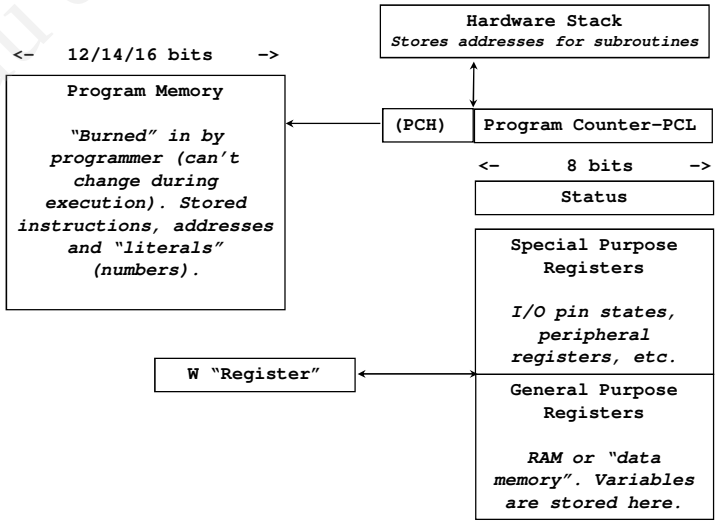
90

Block diagram of the PIC 16 Series ALU



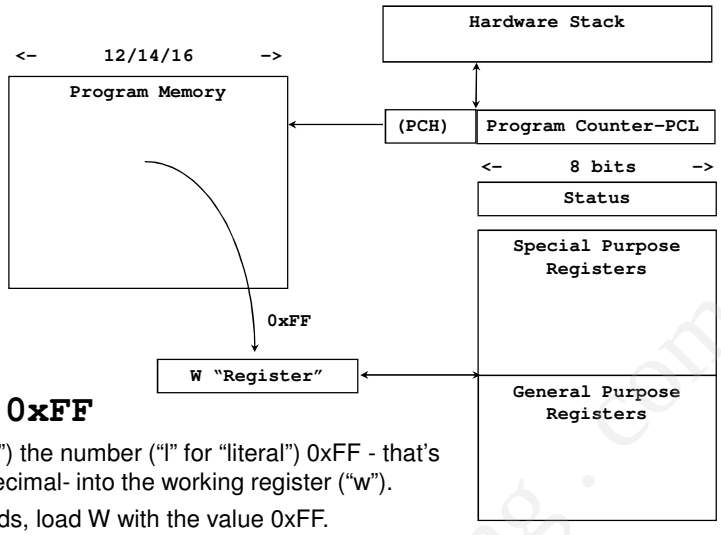
91

Programmer's Model



92

Instruction Example: `movlw 0xFF`



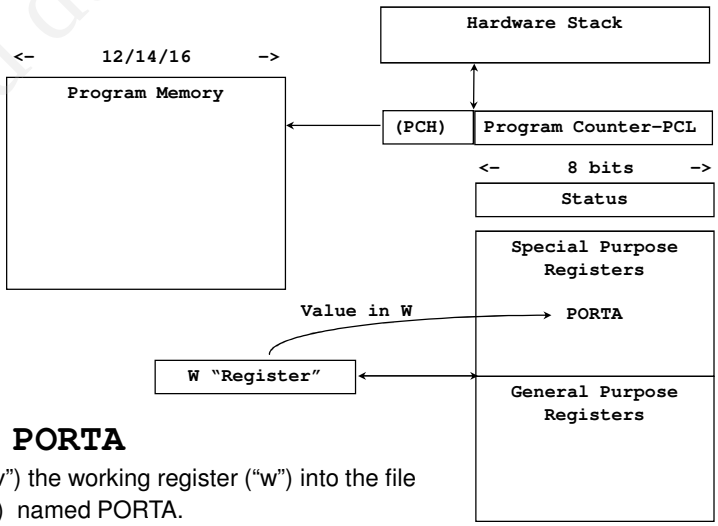
`movlw 0xFF`

Move ("mov") the number ("l" for "literal") 0xFF - that's 256 in decimal- into the working register ("w").

In other words, load W with the value 0xFF.

93

Instruction Example: `movwf PORTA`



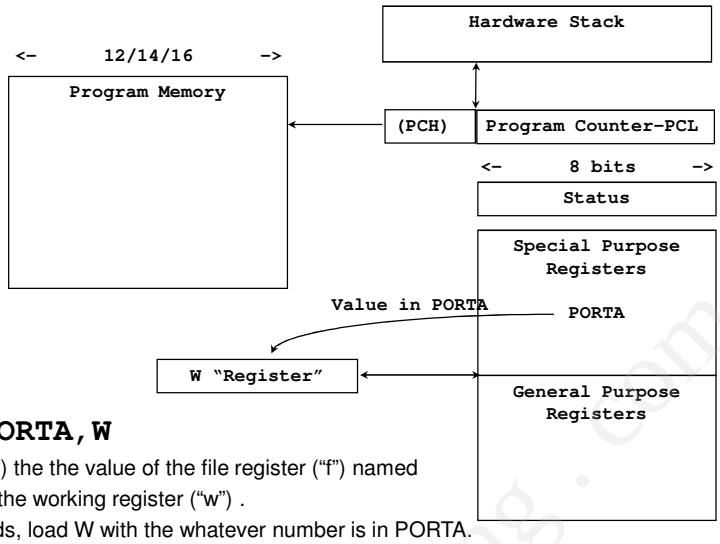
`movwf PORTA`

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.

94

Instruction Example: `movf PORTA,W`



`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.

95

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

        clrf    PORTA           ; Clear PORTA register
        clrf    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
```

96

Branches

- All branches are “Bit Tests”
- All branches only skip one instruction

```
; Set EqualFlag if PORTA = PORTB

        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
```

97

STATUS Register

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
PA2	PA1	PA0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7	6	5	4	3	2	1	bit0

R = Readable bit
W = Writable bit
- n = Value at POR reset

bit 7: **PA2**: This bit unused at this time.
Use of the PA2 bit as a general purpose read/write bit is not recommended, since this may affect upward compatibility with future products.

bit 6-5: **PA1:PA0**: Program page preselect bits (PIC16C56s/CR56s)(PIC16C57s/CR57s)(PIC16C58s/CR58s)
00 = Page 0 (000h - 1FFh) - PIC16C56s/CR56s, PIC16C57s/CR57s, PIC16C58s/CR58s
01 = Page 1 (200h - 3FFh) - PIC16C56s/CR56s, PIC16C57s/CR57s, PIC16C58s/CR58s
10 = Page 2 (400h - 5FFh) - PIC16C57s/CR57s, PIC16C58s/CR58s
11 = Page 3 (600h - 7FFh) - PIC16C57s/CR57s, PIC16C58s/CR58s
Each page is 512 words.
Using the PA1:PA0 bits as general purpose read/write bits in devices which do not use them for program page preselect is not recommended since this may affect upward compatibility with future products.

bit 4: $\overline{TO}$: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3: $\overline{PD}$: 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/$\overline{borrow}$ bit (for ADDWF and SUBWF instructions)
ADDWF
1 = A carry from the 4th low order bit of the result occurred
0 = A carry from the 4th low order bit of the result did not occur
SUBWF
1 = A borrow from the 4th low order bit of the result did not occur
0 = A borrow from the 4th low order bit of the result occurred

bit 0: **C**: Carry/$\overline{borrow}$ bit (for ADDWF, SUBWF and RRF, RLF instructions)

ADDWF	SUBWF	RRF or RLF
1 = A carry occurred	1 = A borrow did not occur	Load bit with LSb or MSb, respectively
0 = A carry did not occur	0 = A borrow occurred	

98

Direct Addressing

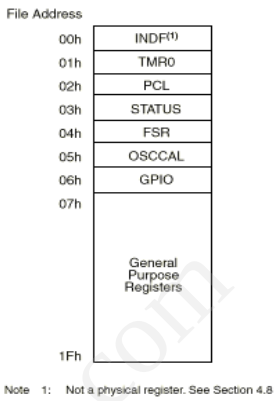
- All file registers (RAM) are accessed by an address. This is called *direct addressing*.
- For example,

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

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

- Thankfully, we can use labels instead of addresses:

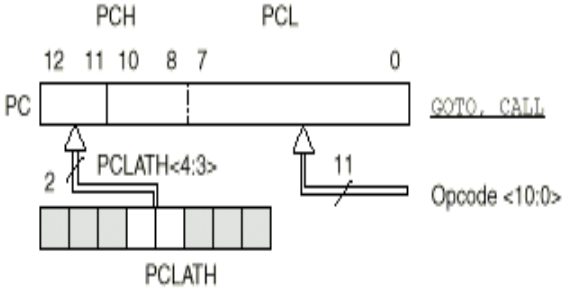
```
GPIO  equ   0x06
      movwf GPIO
```



99

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.



14bit core

100

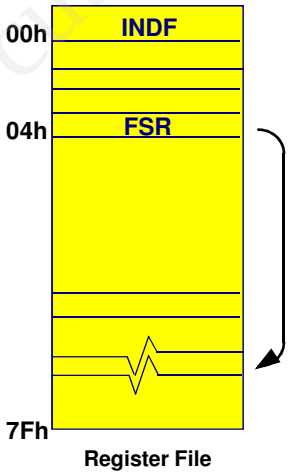
Software: Relative Addressing

- 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   addwf  PCL, W        ; Jump to (current PCL) + W
        retlw  0x00          ; Return with 0x00 in W
        retlw  0x23          ; Return with 0x23 in W
        retlw  0x33          ; etc.
        retlw  0x88
```

101

Indirect Addressing



- Load indirect address into FSR
- Reading/Writing to INDF acts on address stored in FSR
- Example code to clear 0x20 - 7F:

```
        movlw  0x20
        movwf  FSR

loop    clrf   INDF
        incf   FSR,F
        btfss  FSR,7
        goto   loop
```

102

Banking

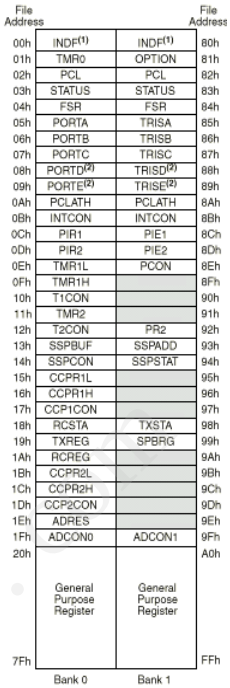
RAM in the PICs is banked, especially special function registers. Use the bank select commands to choose the bank.

Either:

```
bsf    STATUS, RP0
bcf    STATUS, RPO
```

Or use the assembler directive:

```
Banksel      <registername>
```



File Address	Bank 0	Bank 1	File Address
00h	INDF(1)	INDF(1)	80h
01h	TMR0	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	PORTC	TRISC	87h
08h	PORTD(2)	TRISD(2)	88h
09h	PORTE(2)	TRISE(2)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	PIR1	PIE1	8Ch
0Dh	PIR2	PIE2	8Dh
0Eh	TMR1L	PCON	8Eh
0Fh	TMR1H		8Fh
10h	T1CON		90h
11h	TMR2		91h
12h	T2CON	PR2	92h
13h	SSPBUF	SSPADD	93h
14h	SSPCON	SSPSTAT	94h
15h	CCPR1L		95h
16h	CCPR1H		96h
17h	CCP1CON		97h
18h	RCSTA	TXSTA	98h
19h	TXREG	SPBRG	99h
1Ah	RCREG		9Ah
1Bh	CCPR2L		9Bh
1Ch	CCPR2H		9Ch
1Dh	CCP2CON		9Dh
1Eh	ADRES		9Eh
1Fh	ADCON0	ADCON1	9Fh
20h	General Purpose Register	General Purpose Register	A0h
7Fh			FFh

103

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

104

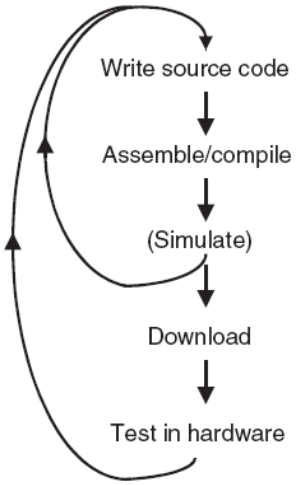
Constants and Syntax Used in Assembler Language

Constant	Syntax	Example	Value
Decimal	D'*decimal_number*' .'*decimal_number*'	D'167' .'167'	0x000000A7
Hexadecimal	H'*hexadecimal*' 0x*hexadecimal* *hexadecimal*H	H'A7' 0xA7 0A7H	0x000000A7
Octal	O'*octal*' *octal*O	O'247' 247O	0x000000A7
Binary	B'*binary*'	B'10100111'	0x000000A7
ASCII	A'*ASCII_char*' `*ASCII*'	A'Z' `Z'	0x0000005A

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

105

The program development process



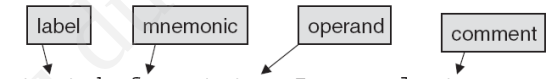
106

Components of MPLAB development system

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

107

Assembler format




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

Column 1	Column 2	Column 3	Column 4
Label	**COMMAND**	**Operand/s**	**; Comment**
Label EQUated to a value, or to indicate a program destination address for jumps.	Mnemonic form of the instruction for the processor to carry out a specific operation. Only mnemonics specified in the instruction set may be used.	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.	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.

108

Table 4.1 Some common MPASM Assembler directives

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

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

Table 4.2 Number representation in MPASM Assembler

Radix	Example representation
Decimal	`D'255'`
Hexadecimal	`H'8d' or 0x8d`
Octal	`O'574'`
Binary	`B'01011100'`
ASCII	`'G' or A'G'`

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.

109

Example of Assembler code

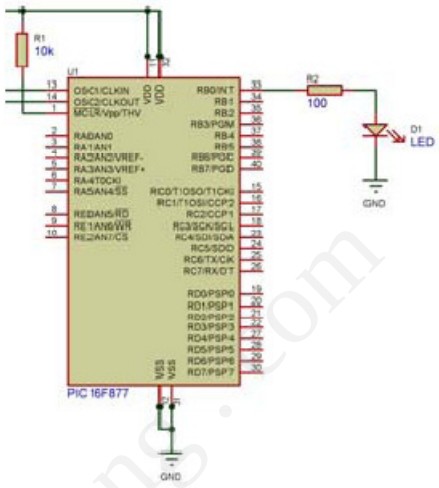
```
Label   Mnemonic        Operand         Comments
        Instruction     Space
        Directive

        Title           "Our first program"     (directive)
        list            p=16f887        ; processor type (directive)
;
; -------------                                 (comment)
; PROGRAM START                                         (comment)
; -------------                                 (comment)
;
        org             0h              ; startup address = 0000 (directive)
start   movlw           0x00            ; simple code (instruction)
        movwf           0x05             (instruction)
        goto            start           ; do this loop forever (instruction)
        end                             (directive)
```

110

PIC Applications

- LED Flasher

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

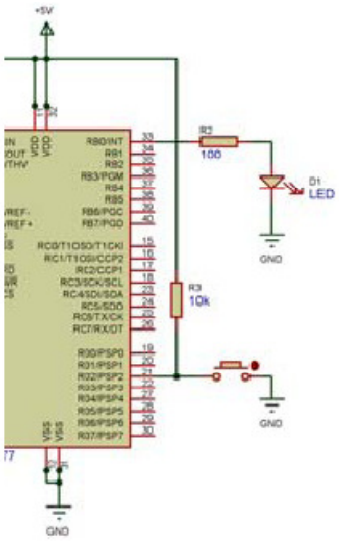


111

PIC Applications

- Button Read

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



112