Getting Started with Embedded Systems

Chapter 1
Sections 1-6

Dr. Iyad Jafar

Outline

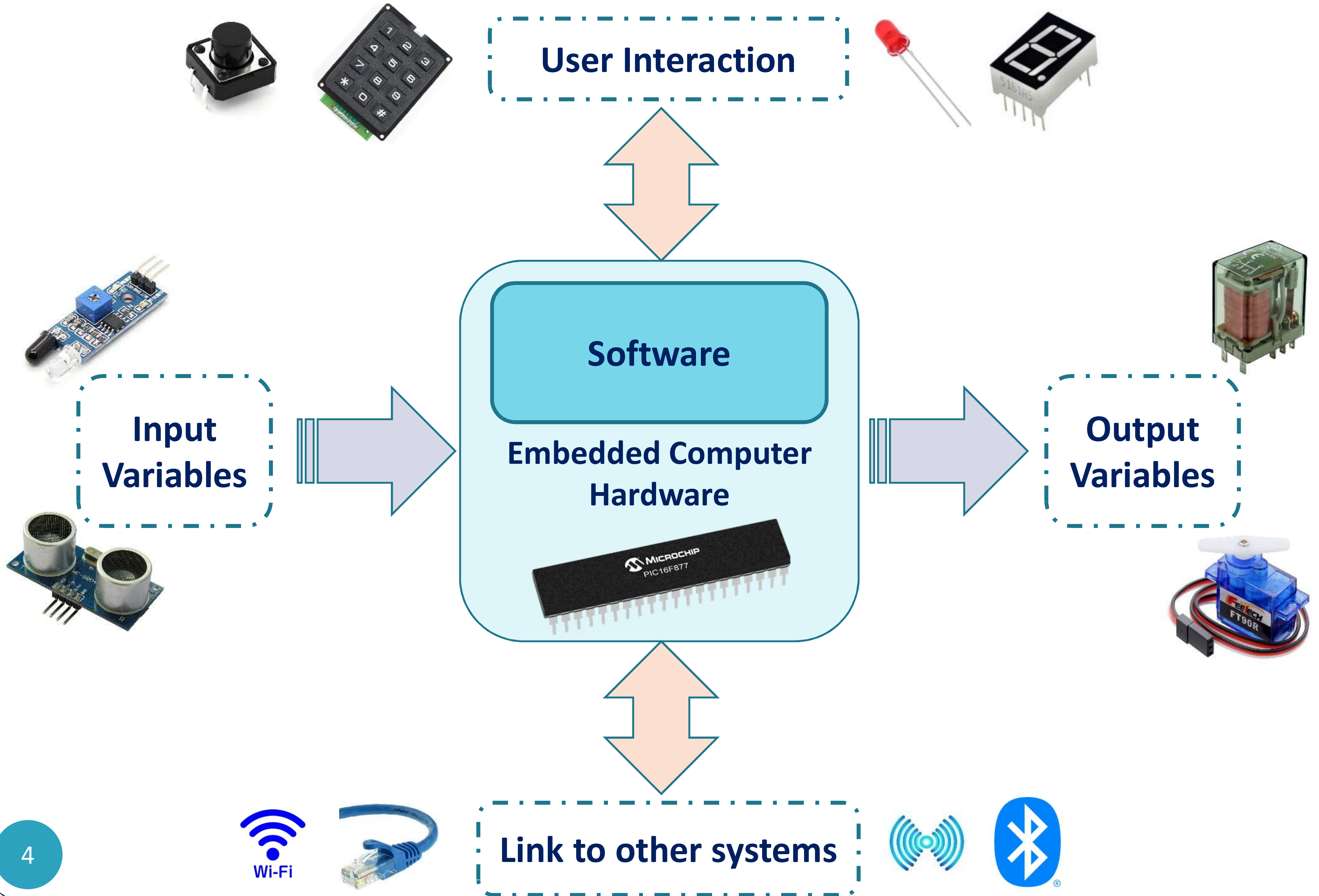
- What is an Embedded System?
- The Essence of Embedded Systems
- Embedded Systems Examples
- Some Computer Essentials
- Microprocessors vs. Microcontrollers
- The PIC Microcontroller
- The PIC 12 Series as an Example

What is an Embedded System?

- **An *embedded system* is a computer system that is**
  - designed to perform one or a few dedicated functions often with real-time computing constraints
  - *embedded as part of a complete* device often including hardware and mechanical parts.

- By contrast, a general-purpose computer, such as a personal computer, is designed to be flexible and to meet a wide range of end-user needs.

The Essence of Embedded Systems



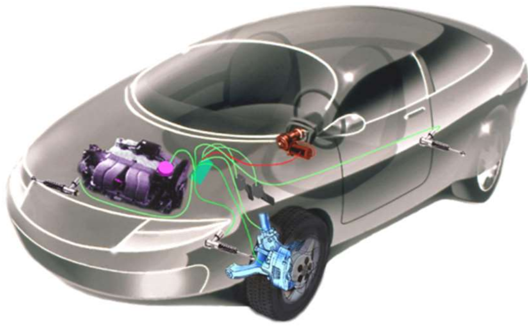
The Essence of Embedded Systems

- **Characteristics**
  - *Software driven*
  - *Reliable*
  - *Real-time control system*
  - *Microcontroller or DSP based*
  - *Autonomous / human interactive / network interactive*
  - *Operate on diverse input variables and in diverse environments*

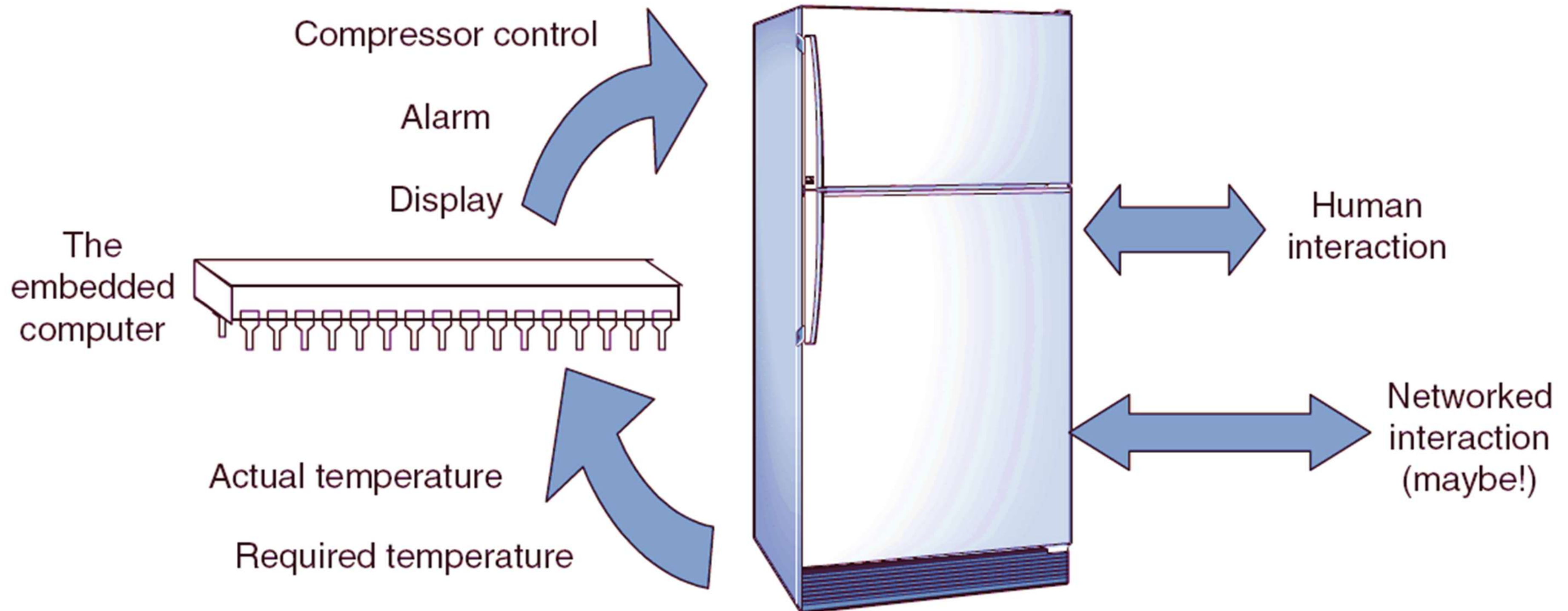
Examples

- Automotive
- Avionics/Aerospace/Defence
- Industrial Automation
- Telecommunications
- Consumer Electronics & Intelligent Homes & Retail (Thin Clients/POS)
- Scientific & Medical Equipment
- Computer peripherals

Examples



Examples

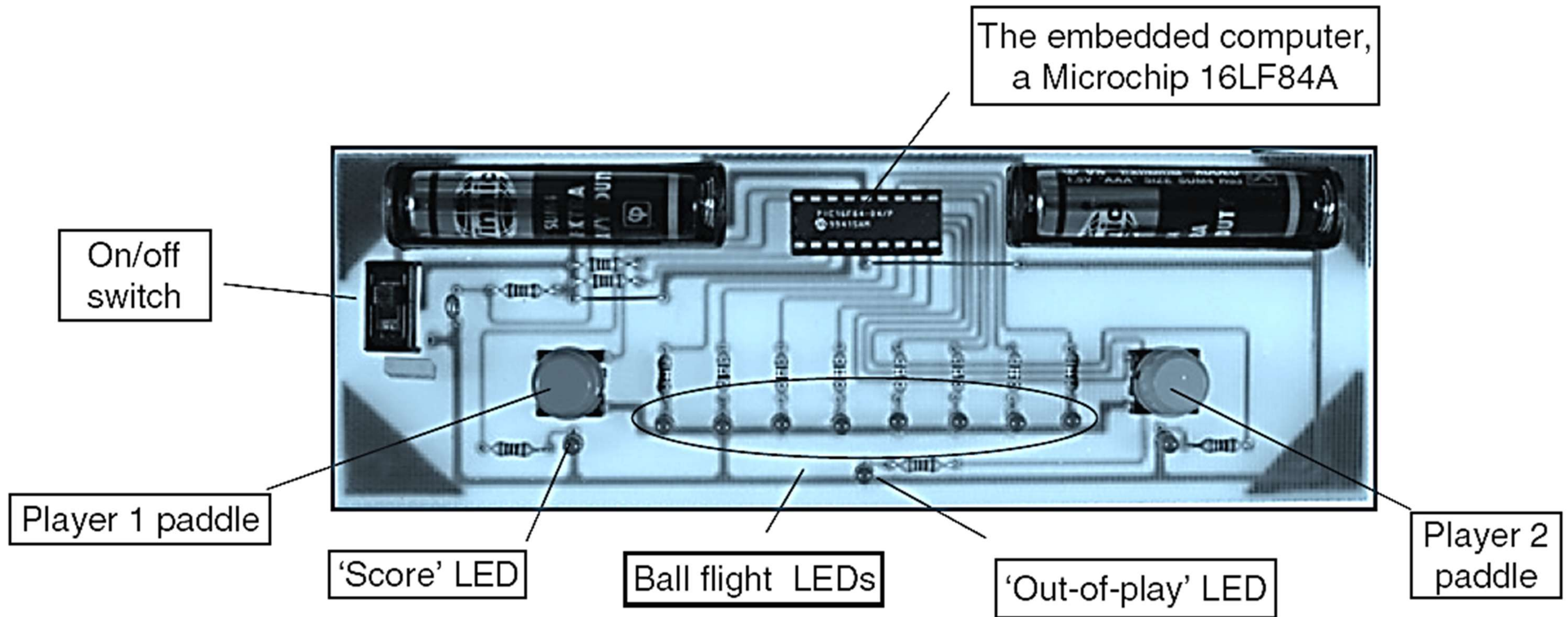


- The refrigerator is required to maintain low temperature by reading the current value and controlling the compressor accordingly

Examples

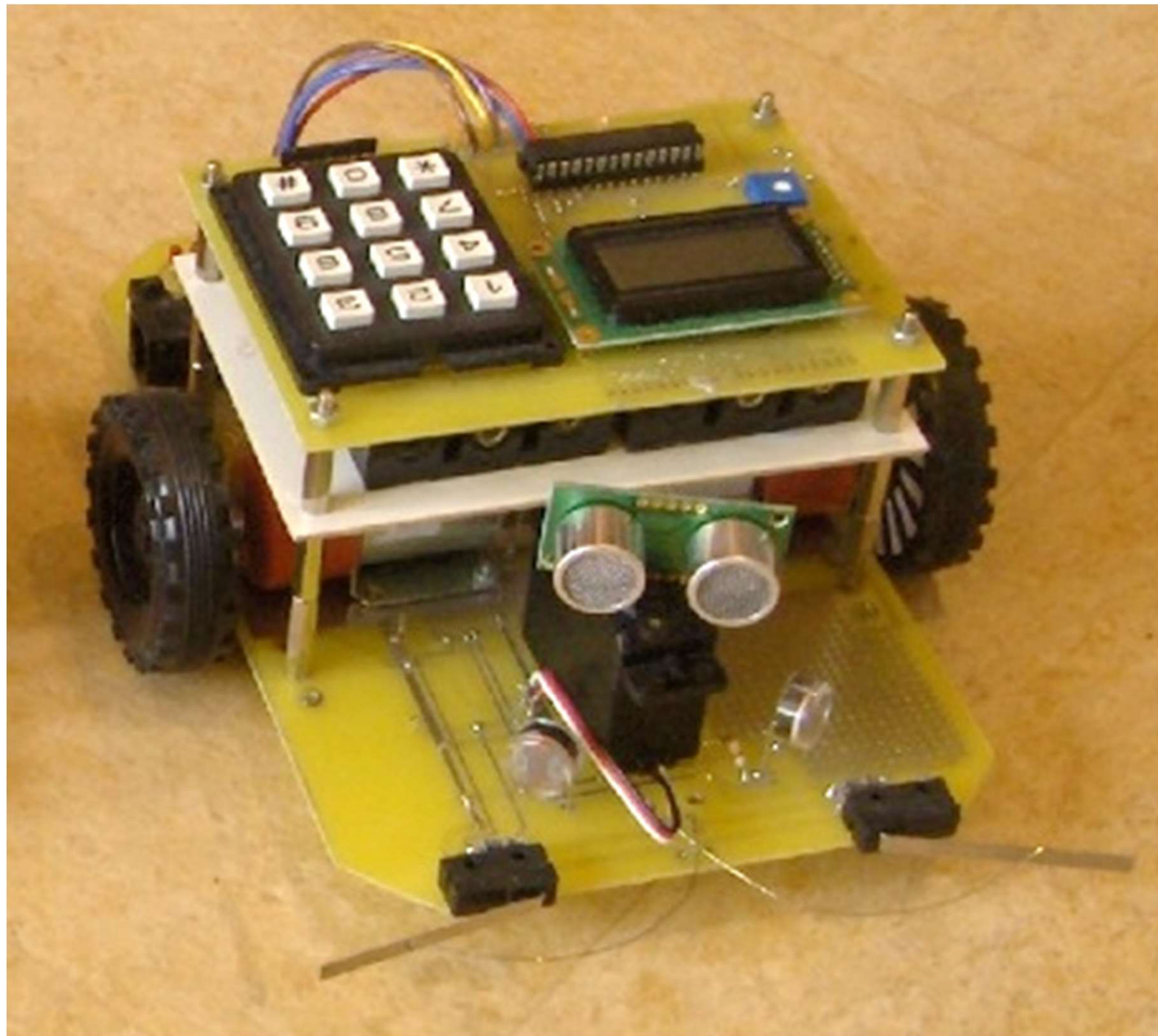
- Different sensors in the car door produce signals that are of great importance when integrated with the rest of the car functionality

Examples



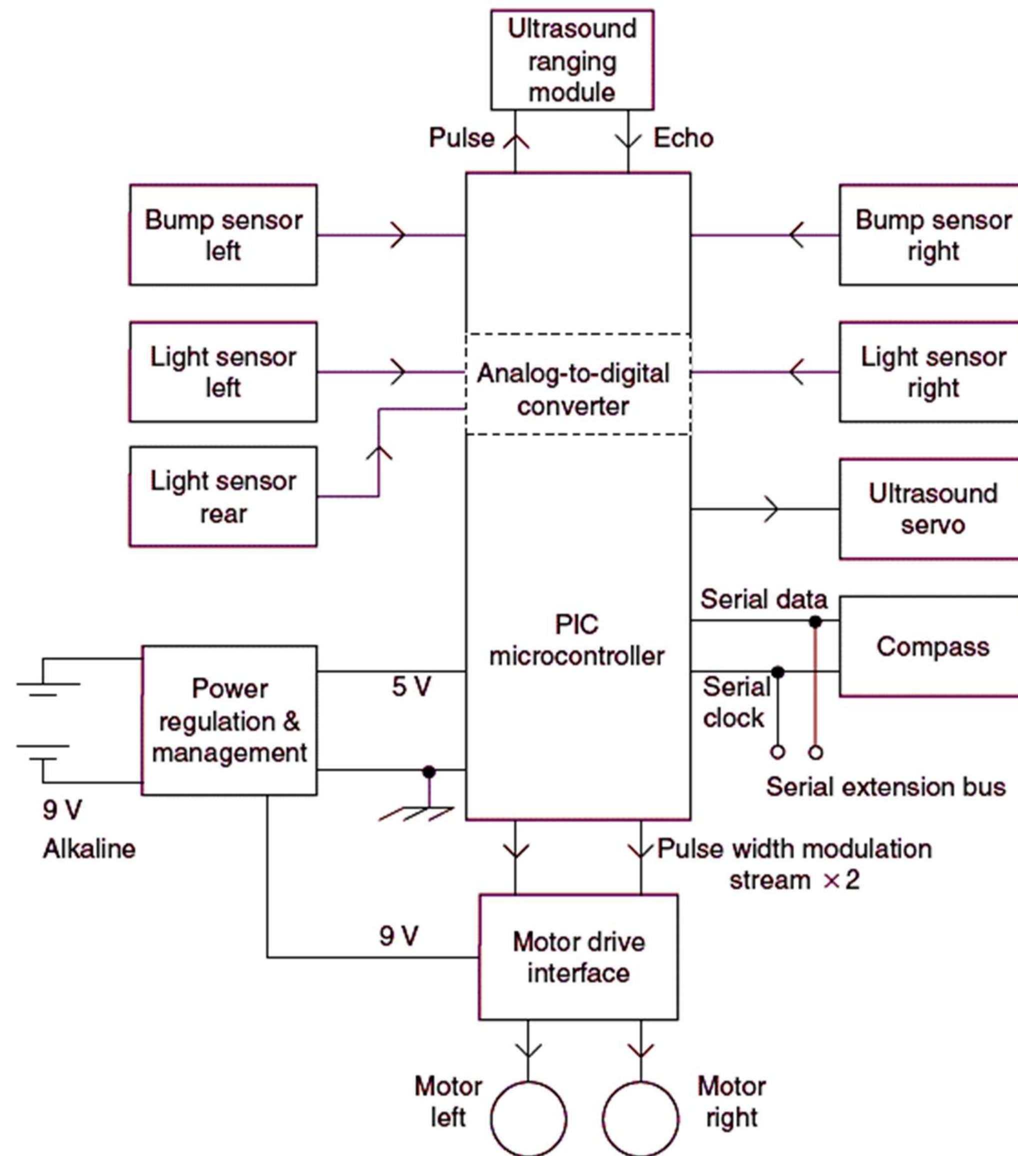
- The Electronic 'ping-pong'

Examples



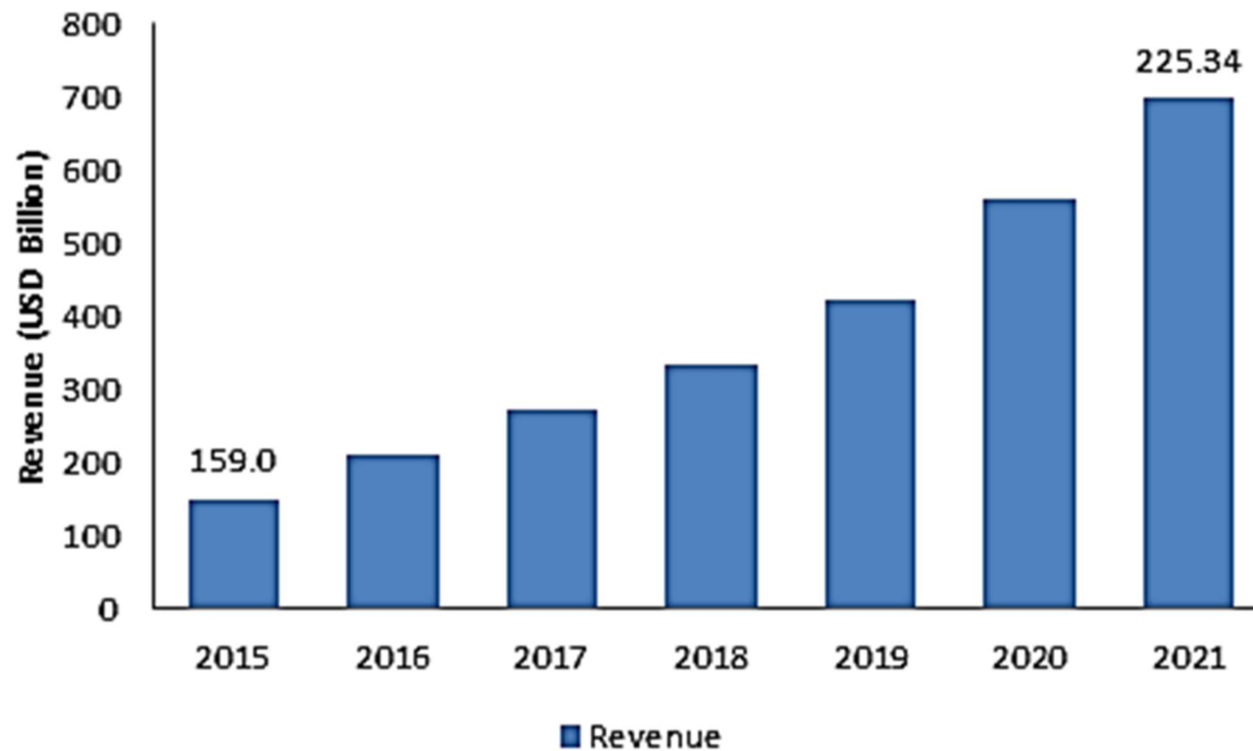
- The Derbot Autonomous Guided Vehicle
- More sensors and powerful microcontroller

Examples



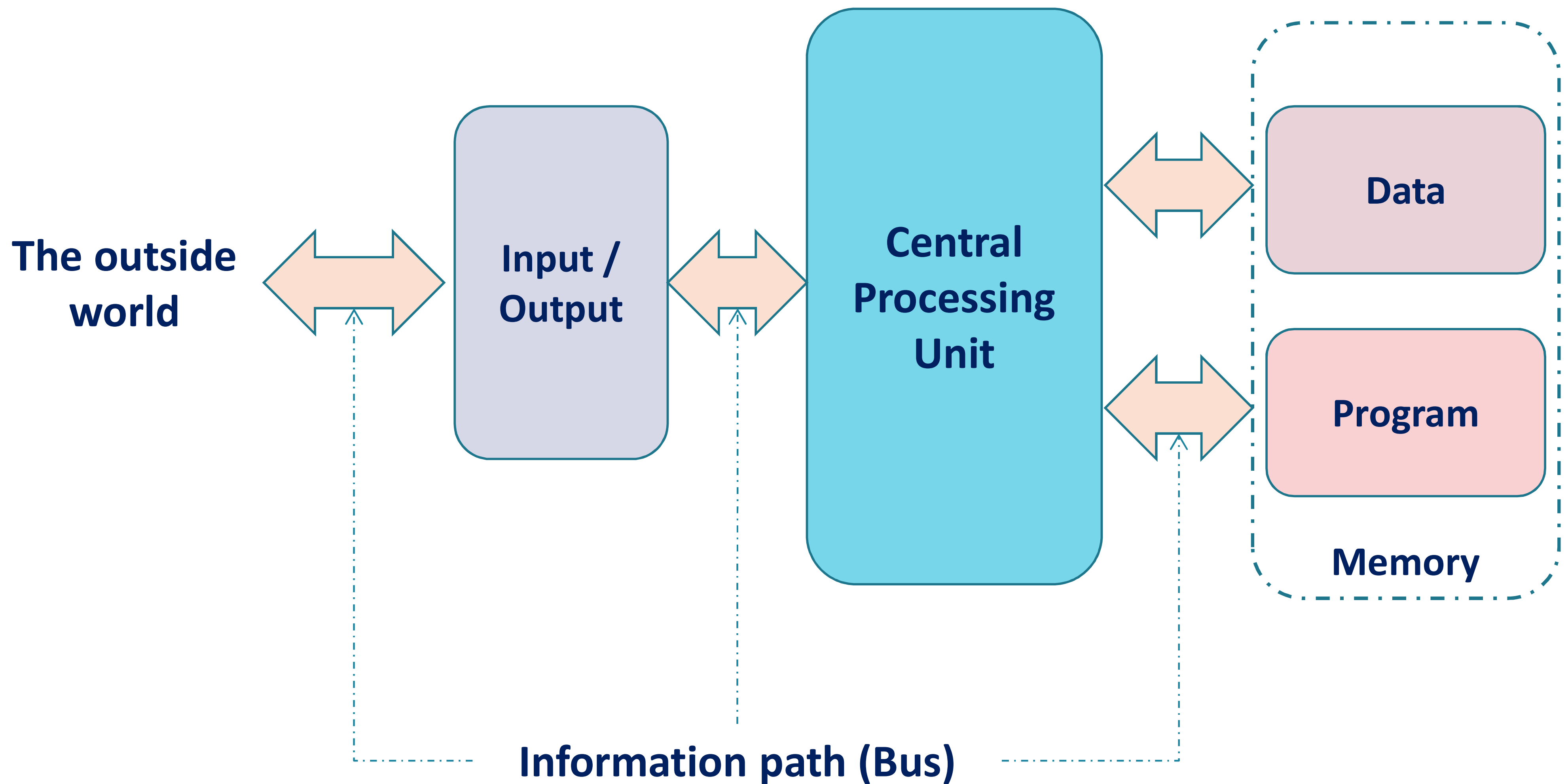
- The Derbot Autonomous Guided Vehicle

Embedded Systems Market



Some Computer Essentials

- **Elements of a Computer**

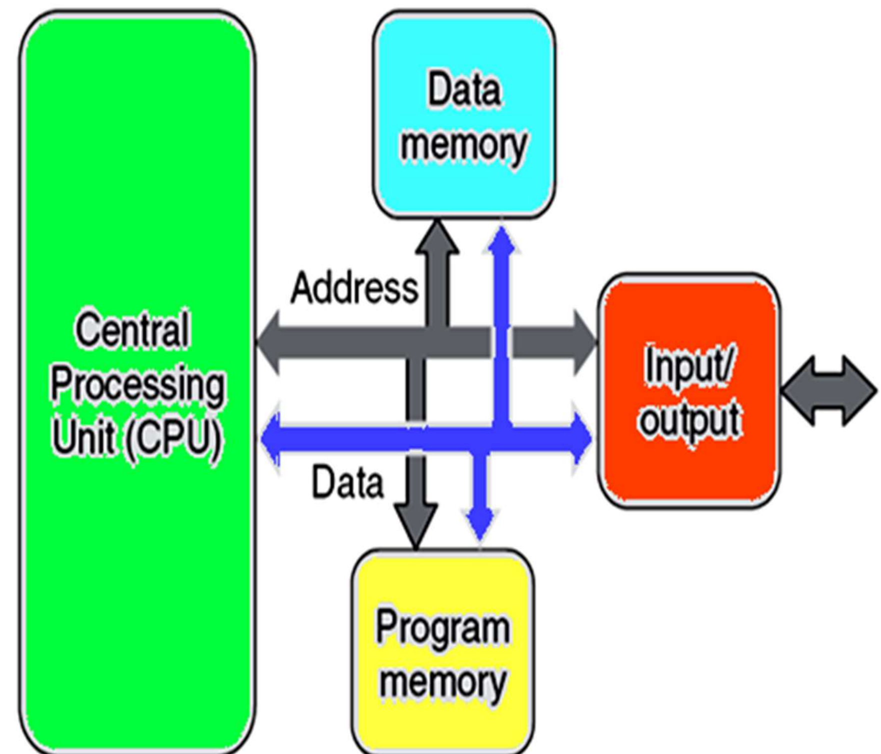


Some Computer Essentials

Memory Organization

The Von Neumann Architecture

- One address bus and one data bus
- I/O may be also connected to these busses
- Simple and logical architecture, however
  - Same memory width for instruction and data ?!
  - Shared busses ?!

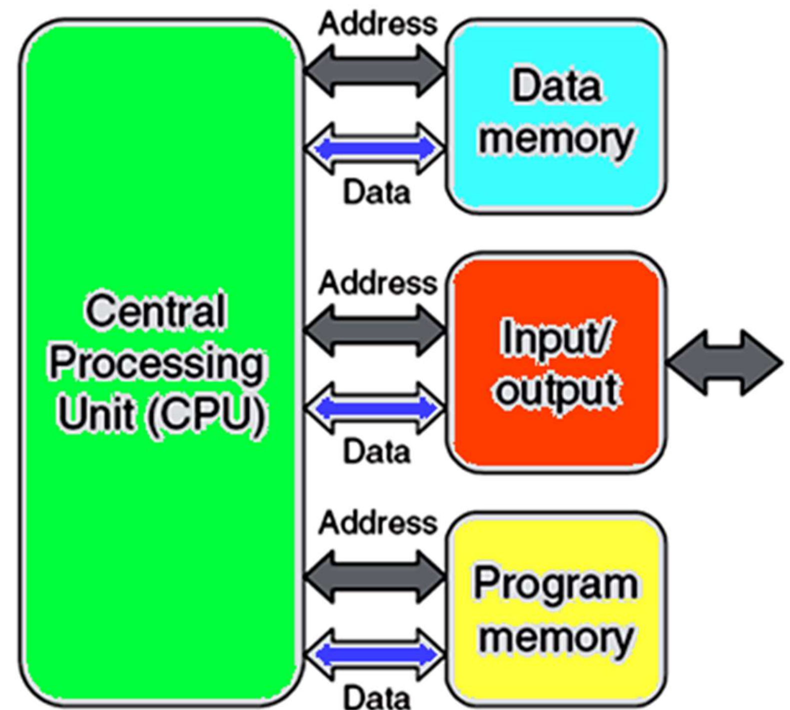


Some Computer Essentials

Memory Organization

The Harvard Architecture

- Separate address and data bus for program memory and data memory
- More flexibility;
  - Different memory width
  - Simultaneous access of data and program memories
- Complex ?!



Some Computer Essentials

Instruction Sets

- Every CPU has a set of instructions that it can recognize and execute
- There are different approaches in designing instructions for the CPU in attempt to speed up program execution
  - **CISC (Complex Instruction Set Computers)**
    - Many instructions and addressing modes
    - Instructions have different levels of complexity (different size and execution time)
    - Relatively slow
    - Shorter programs
  - **RISC (Reduced Instruction Set Computers)**
    - Few instructions and addressing modes
    - Simple instructions of fixed size
    - Relatively fast
    - Longer programs

Some Computer Essentials

- **Memory Types**
  - **Volatile**
    - Holds its contents as long as power is ON
    - Used as temporary *storage to hold data*
    - Easy to write
    - RAM
  - **Non-volatile**
    - Retains its values on power out
    - More difficult to write in terms of time and power
    - In embedded systems, it is *usually used to store programs*
    - ROM

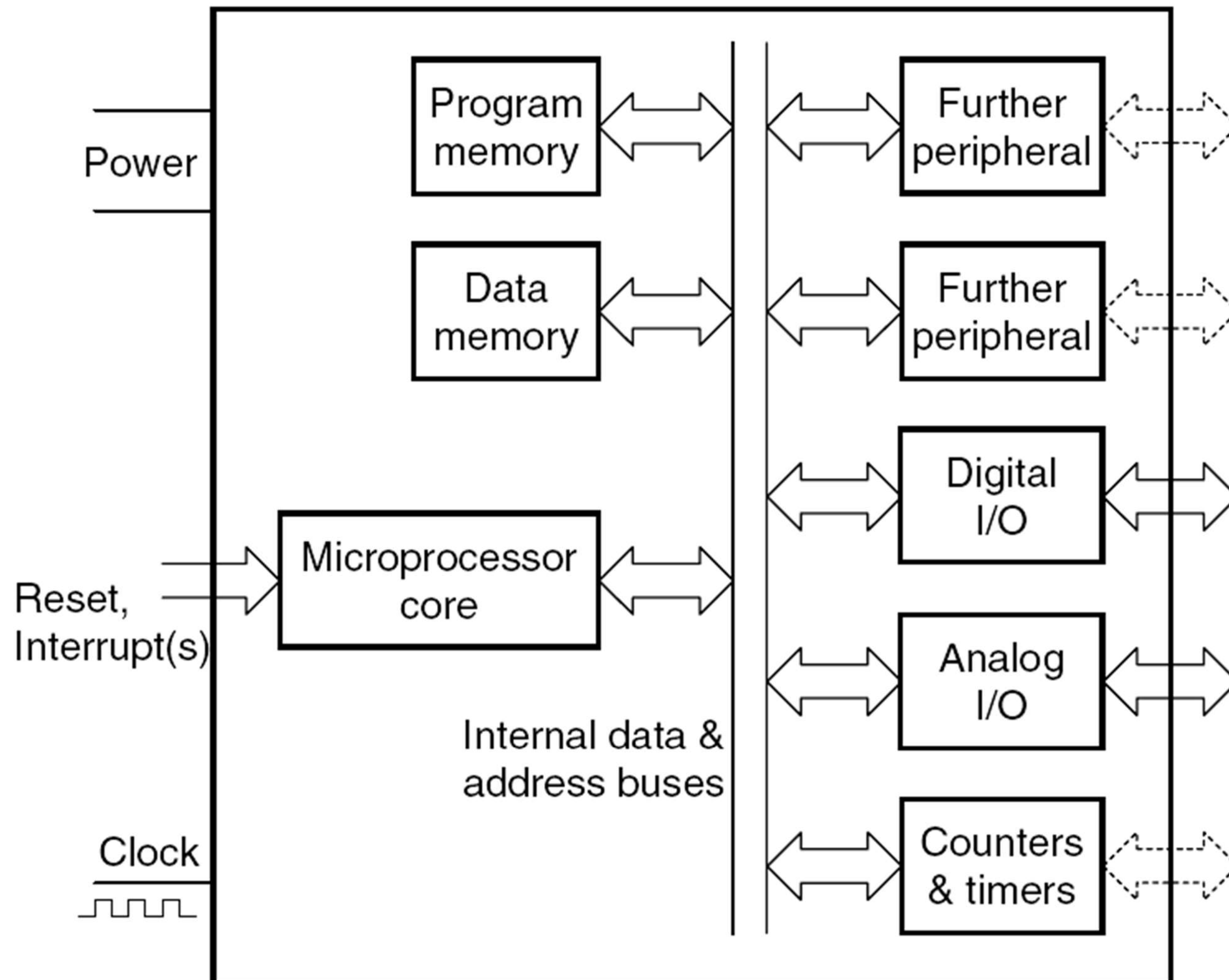
Microprocessors and Microcontrollers

- **First microprocessors in the 1970s**
  - The computer CPU on a single chip
  - Initially, memory and I/O interfacing outside the CPU
  - As technology evolved, the microprocessor became more self-contained, powerful, and faster

- **A special category of microprocessors emerged**
  - Microcontrollers
  - Intended for control purposes
  - *No high computational power, huge memories, or high speed is required*
  - *Has excellent I/O capabilities*
  - Small, low cost, and self contained

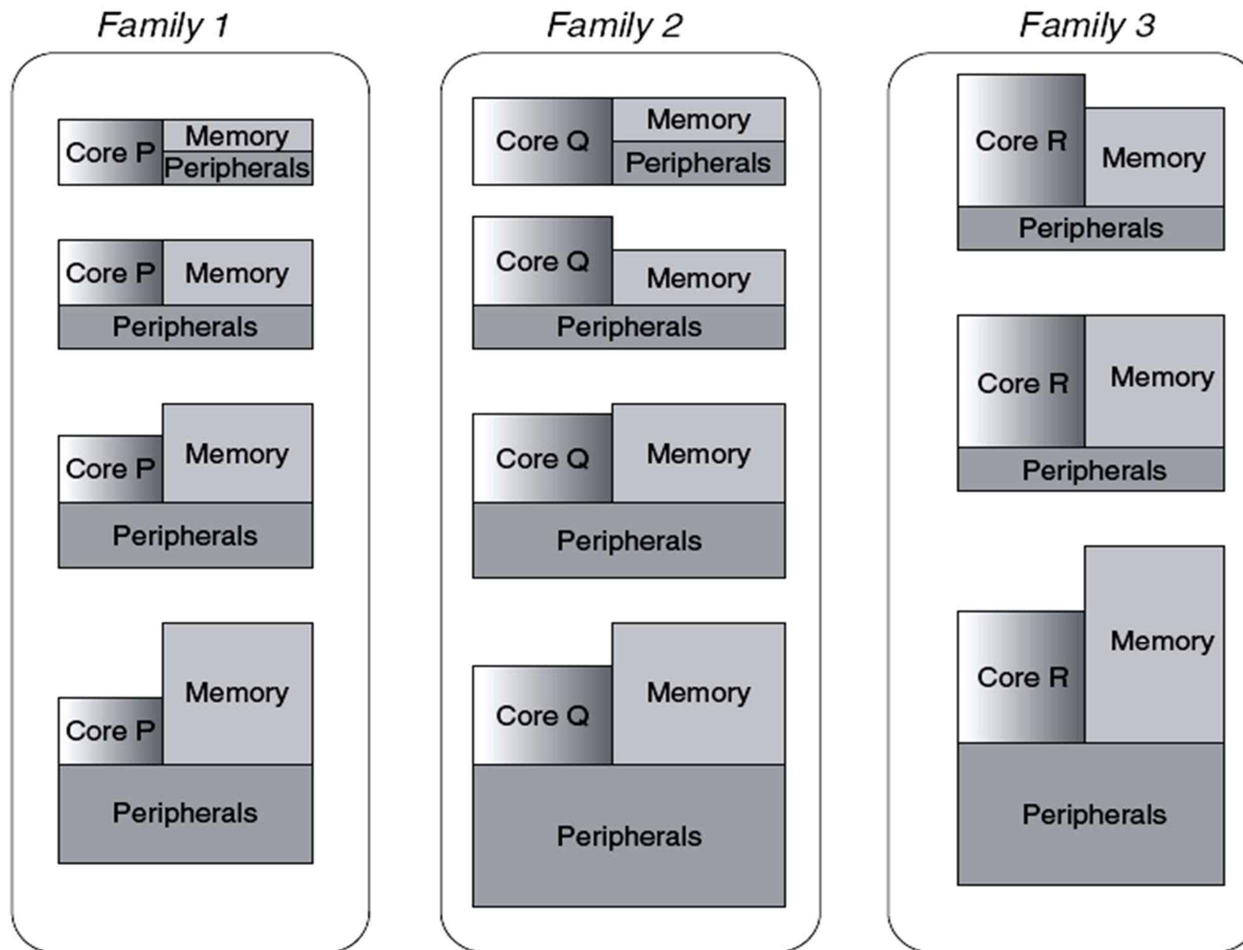
Microprocessors and Microcontrollers

- **A generic microcontroller**



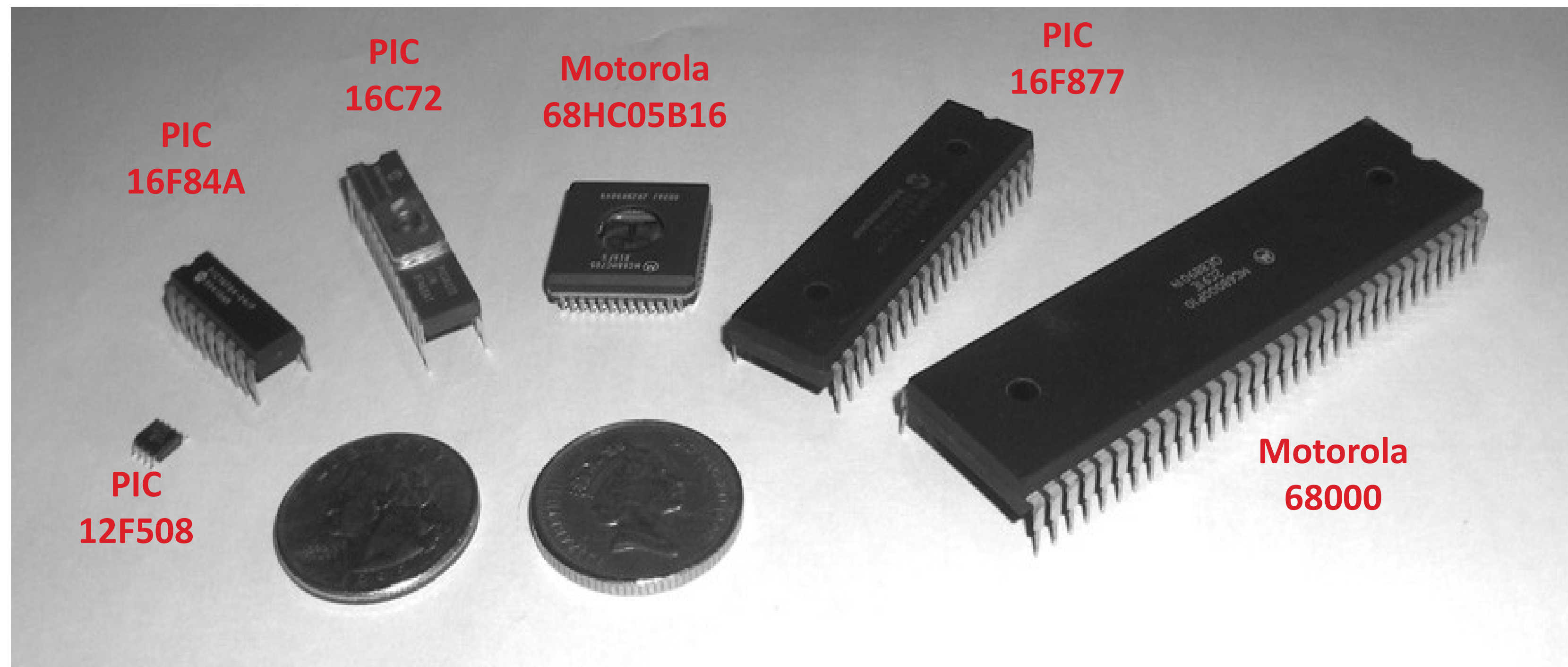
Microprocessors and Microcontrollers

- Microcontroller Families
  - Different families with *each family built around the same core*
  - Family members differ in *memory size* and *peripheral capabilities*



Microprocessors and Microcontrollers

- Microcontroller Packaging
  - Plastic packaging
  - Pins for I/O, clock, communication, and Power.
  - The number of pins usually determines the size of the chip



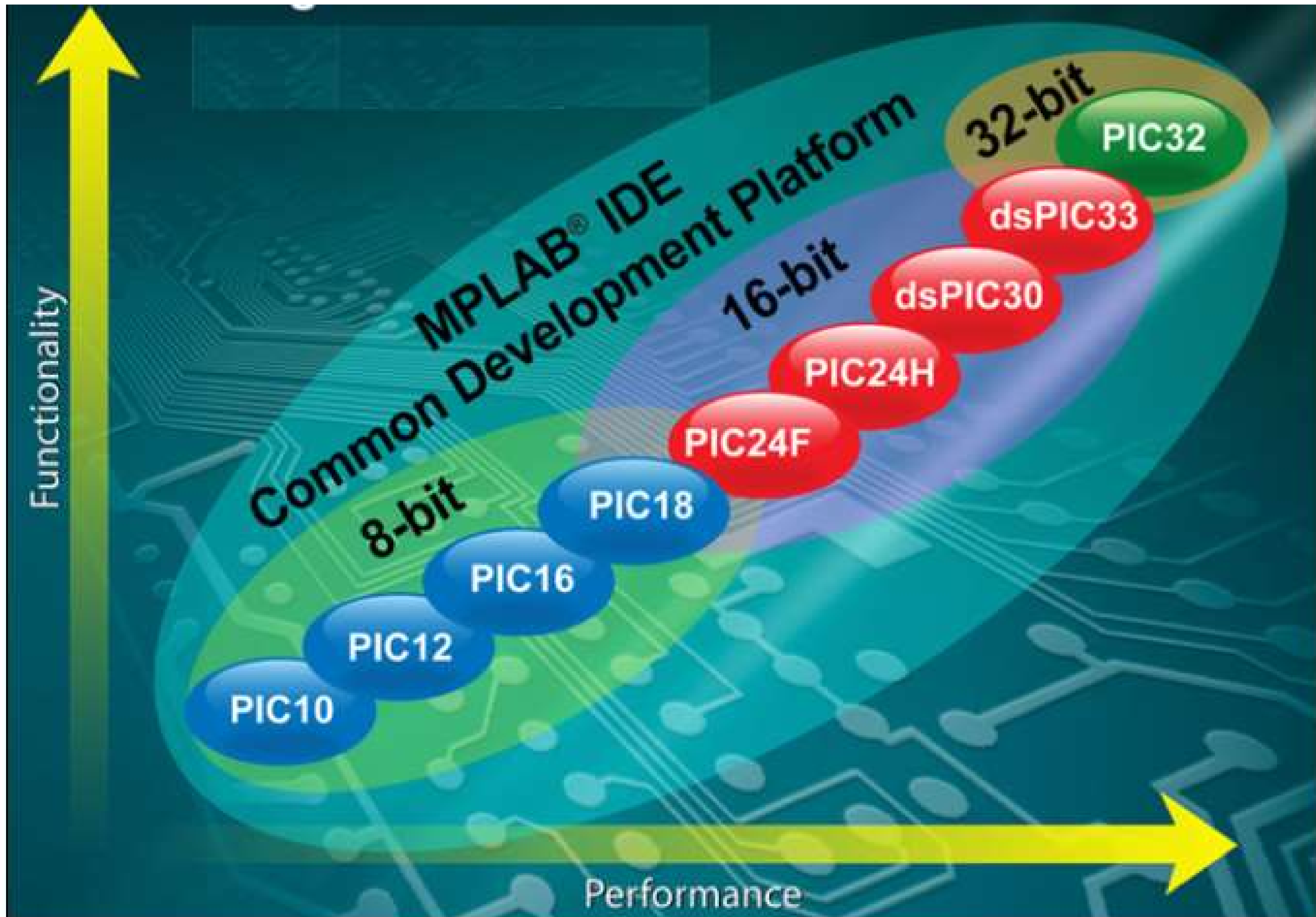
Microchip and the PIC Microcontrollers

- Peripheral Interface Controller (PIC) was originally a design by General Instruments (GI) intended for simple control applications

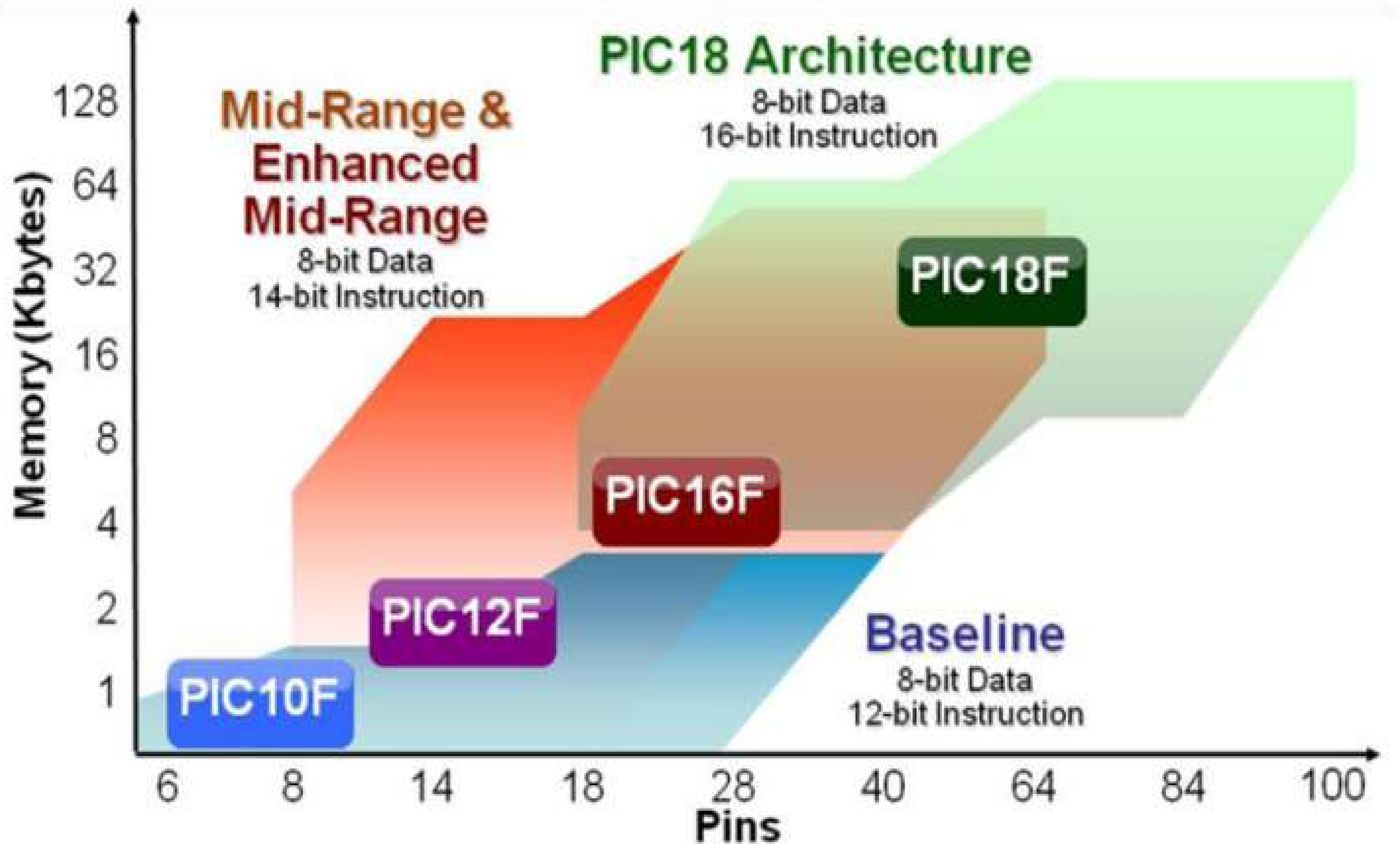
- In the late 1970s, GI introduced PIC® 1650 and 1655
  - Standalone design
  - RISC with 30 instructions
  - Single working register (accumulator)
  - Many attractive features

- PIC was sold to Microchip

Microchip and the PIC Microcontrollers



Microchip and the PIC Microcontrollers



Microchip and the PIC Microcontrollers

- **PIC Families**

PIC Family	Stack Size (words)	Instruction Word Size	No. of Instructions	Interrupt Vectors
12CX/12FX	2	12- or 14-bit	33	None
16C5X/16F5X	2	12-bit	33	None
16CX/16FX	8	14-bit	35	1
17CX	16	16-bit	58	4
18CX/18FX	32	16-bit	75	2

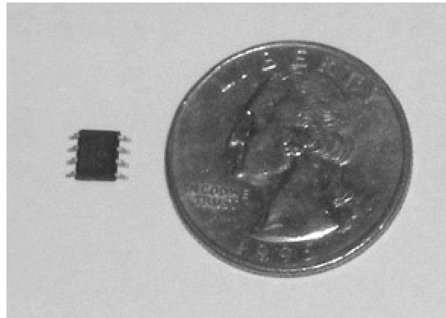
- Example: the 16**C**84 was the first of its kind built using CMOS technology. It was later reissued as 16**F**84**A** incorporating flash memory and other technological features

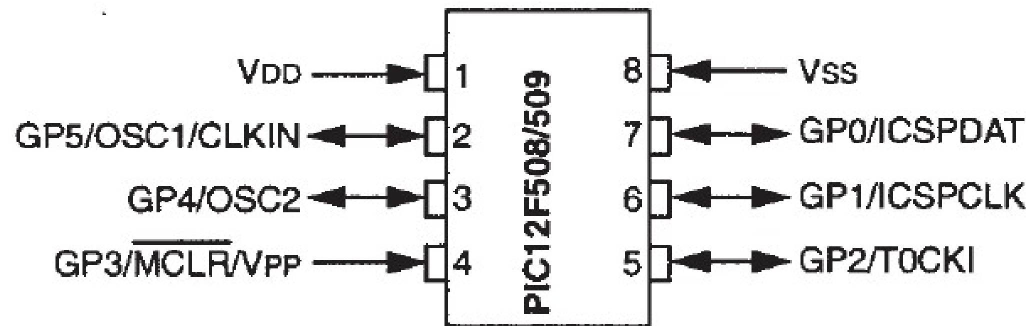
Microchip and the PIC Microcontrollers

- **PIC 16 Series Characteristics**
  - Low-cost
  - Self-contained
  - 8-bit
  - Harvard architecture
  - RISC
  - Pipelined
  - Single accumulator (the working or W register)
  - Fixed reset and interrupt vectors

The PIC 12 Series

- PIC 12F508/509
- The smallest and simplest PIC



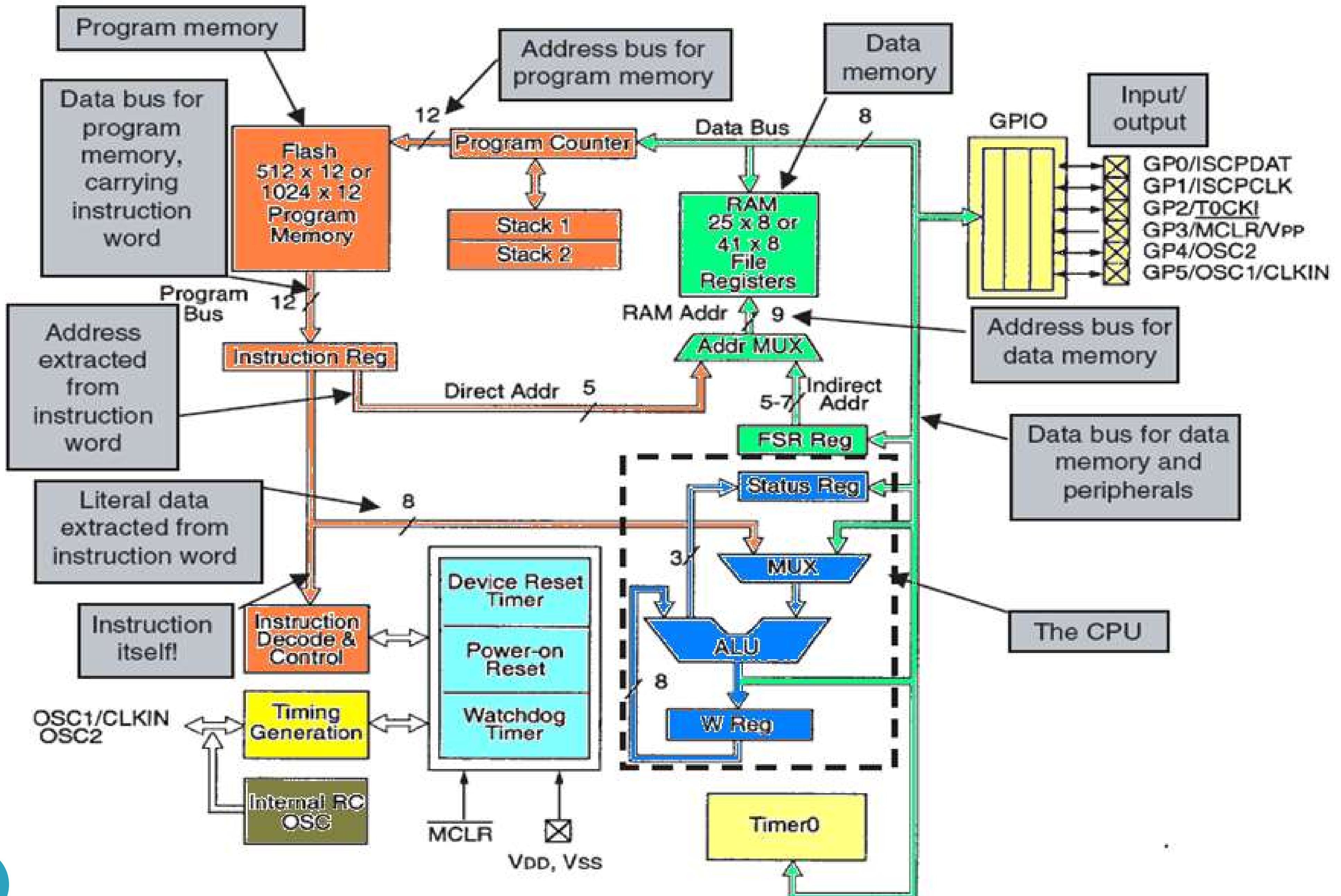


Key

V_{DD}:	Power supply	V_{SS}:	Ground
V_{PP}:	Programming voltage input	MCLR:	Master clear
OSC1, OSC2:	Oscillator pins	CLKIN:	External clock input

GP0 to GP5: General-Purpose input/output pins (bidirectional except GP3)
CSPDAT: In-Circuit Serial Programming™ data pin.
CSPCLK: In-Circuit Serial Programming™ clock pin.

The PIC 12 Series Architecture



Summary

- *An embedded system has one or more computers embedded within it that perform control operations*

- *A microcontroller is at the heart of embedded systems. It is basically a microprocessor with extended I/O capabilities*

- *Microchip is one of the popular vendors for a large variety of microcontrollers with different features*

Introducing the PIC 16 Series and the 16F84A

Chapter 2
Sections 1-8

Dr. Iyad Jafar

Outline

- Overview of the PIC 16 Series
- An Architecture Overview of the 16F84A
- The 16F84A Memory Organization
- Memory Addressing
- Some Issues of Timing
- Power-up and Reset
- The 16F84A On-chip Reset Circuit

Overview of the PIC 16 Series

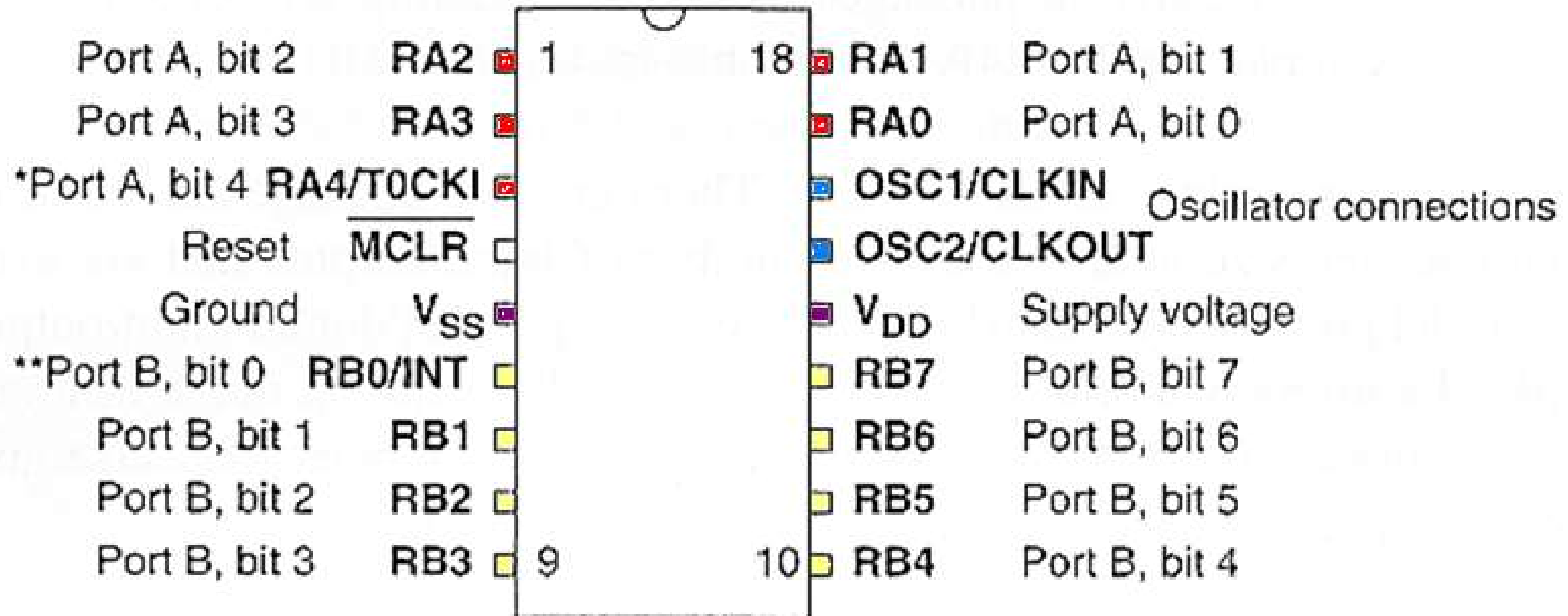
- The PIC 16 series is classified as a midrange microcontroller
- The series has different members all built around the same core and instruction set, but with different memory, I/O features, and package size

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
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

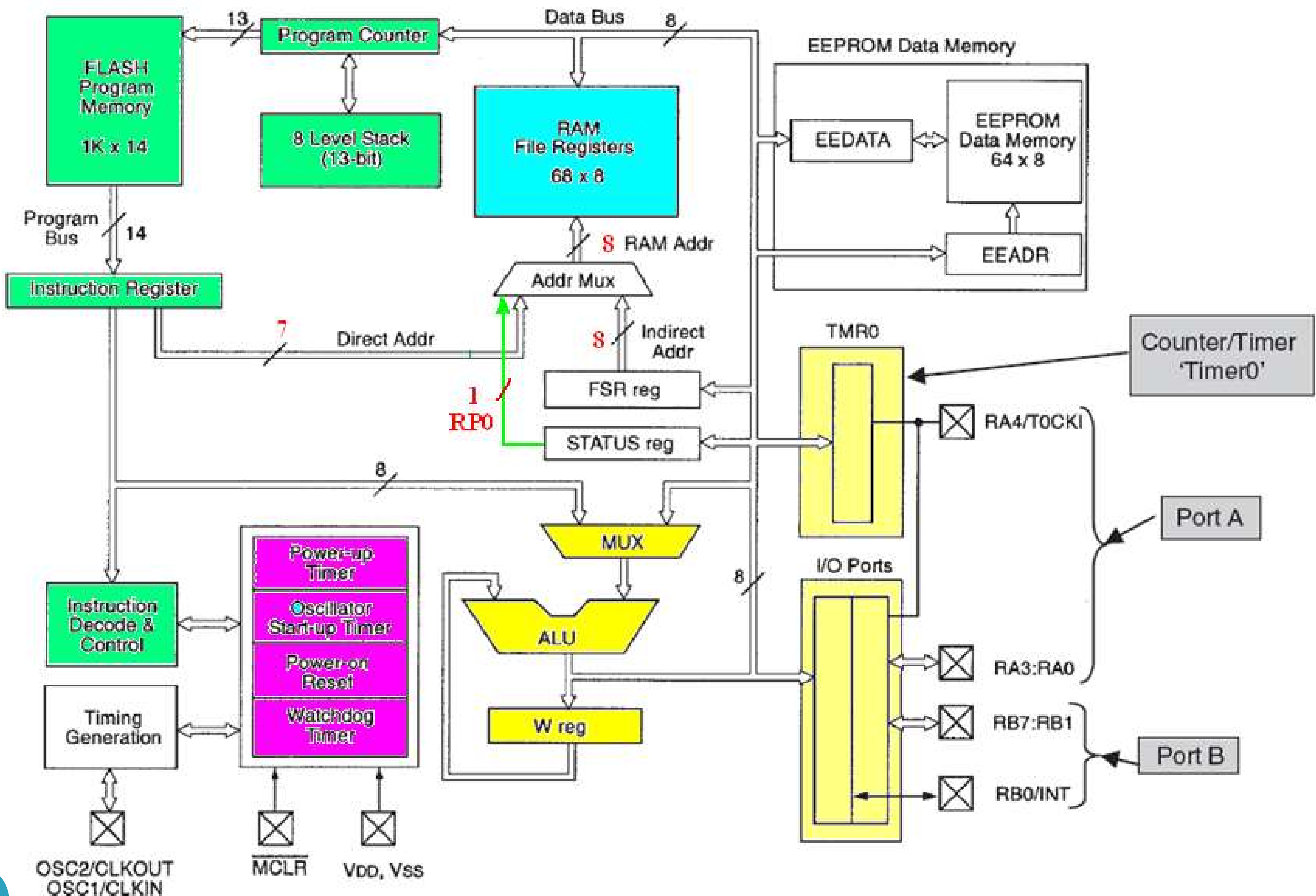
*For DIP package only.

An Architecture Overview of the 16F84A



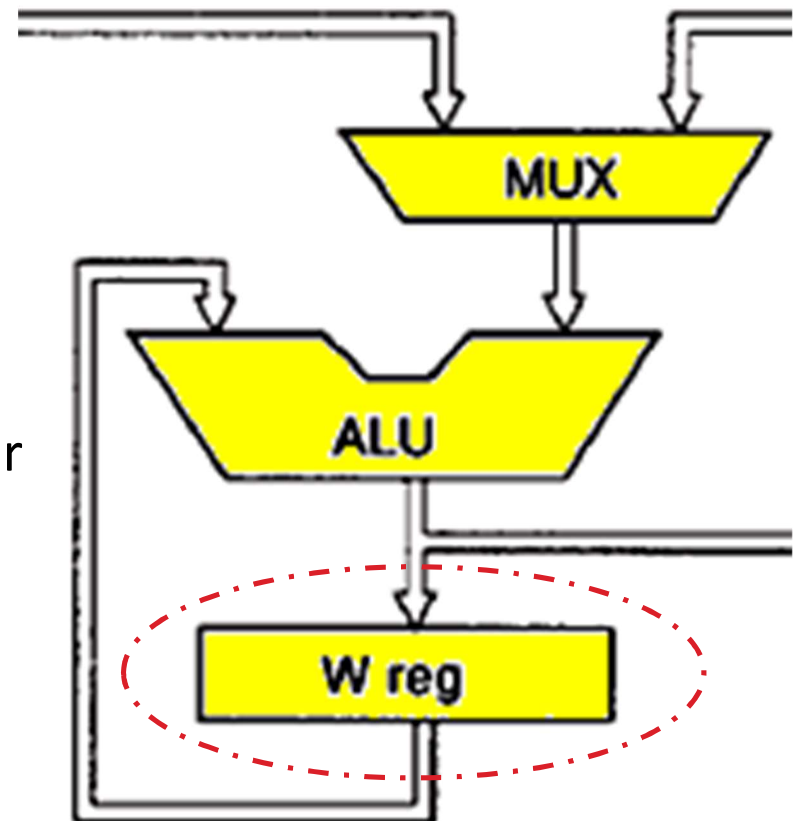
- 18 Pins / DC to 20MHz / 1K program Memory/ 68 Bytes of RAM / 64 Bytes of EEPROM / 1 8-bit Timer / 1 5-bit Parallel Port / 1 8-bit Parallel Port

An Architecture Overview of the 16F84A



The PIC 16F84A ALU and Working Register

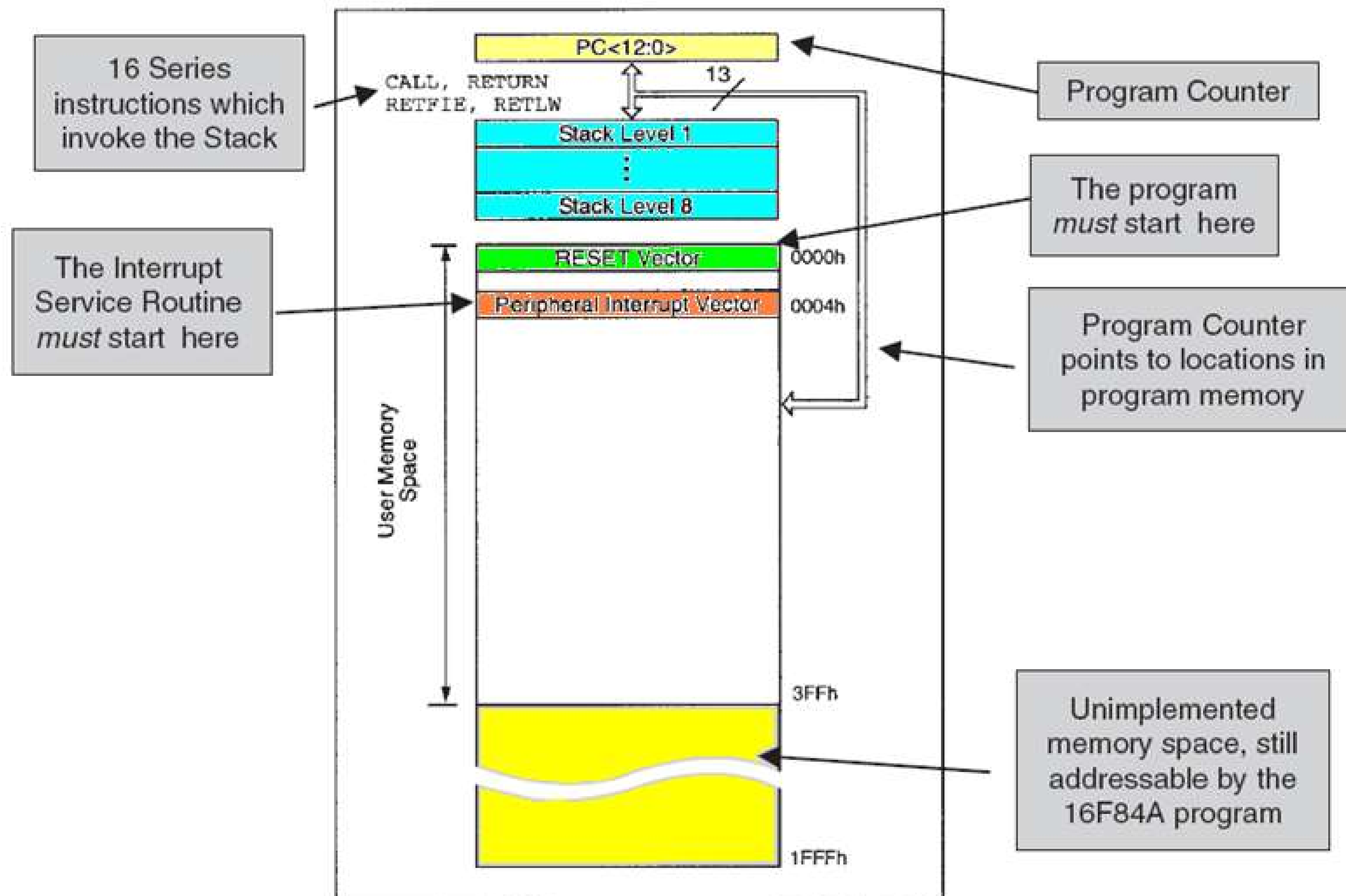
- ***Arithmetic & Logic Unit***
  - 8-bit ALU
  - Supports 35 simple instructions
  - Input operands are
    - The working register
    - Content of some file register or a literal
  - The result is stored in Working register or in a File register



- ***The Working Register***
  - Inside the CPU
  - For many instructions, it can be chosen to hold the result of the last instruction executed by the CPU

The PIC 16F84A Memory Organization

- *Program Memory and Related Units*

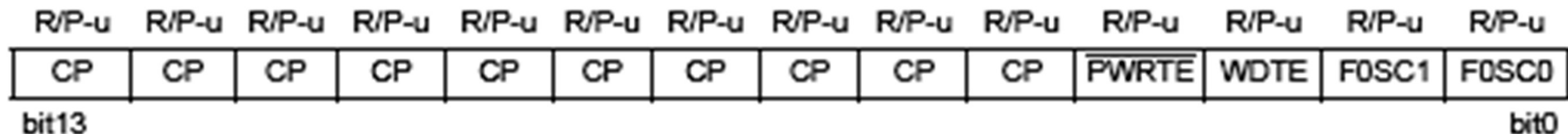


The PIC 16F84A Memory Organization

- ***Program Memory***
  - *1K x 14 Bits*
  - *Address range 0000H – 03FFH*
  - *Flash (nonvolatile)*
  - *10000 erase/write cycles*
  - *Location 0000H is reserved for the reset vector*
  - *Location 0004H is reserved for the Interrupt Vector*
- ***Program Counter***
  - *Holds the address of the instruction to be executed (next instruction)*
- ***Stack***
  - *8 levels (each is 13 bits)*
  - *SRAM (volatile)*
  - *Used to store/load the return address with instruction like CALL, RETURN,RETFIE, and RETLW (interrupts and subroutines)*
- ***Instruction Register***
  - *Holds the instruction being executed*

The PIC 16F84A Memory Organization

- ***The Configuration Word***
  - A special part of the program memory
  - Allows the user to configure different features of the microcontroller at the time of program download and is not accessible within the program or while it is running



R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	$\overline{\text{PWRTE}}$	WDTE	F0SC1	F0SC0
bit13													bit0

bit 13-4 **CP**: Code Protection bit
1 = Code protection disabled
0 = All program memory is code protected

bit 3 **$\overline{\text{PWRTE}}$**: Power-up Timer Enable bit
1 = Power-up Timer is disabled
0 = Power-up Timer is enabled

bit 2 **WDTE**: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

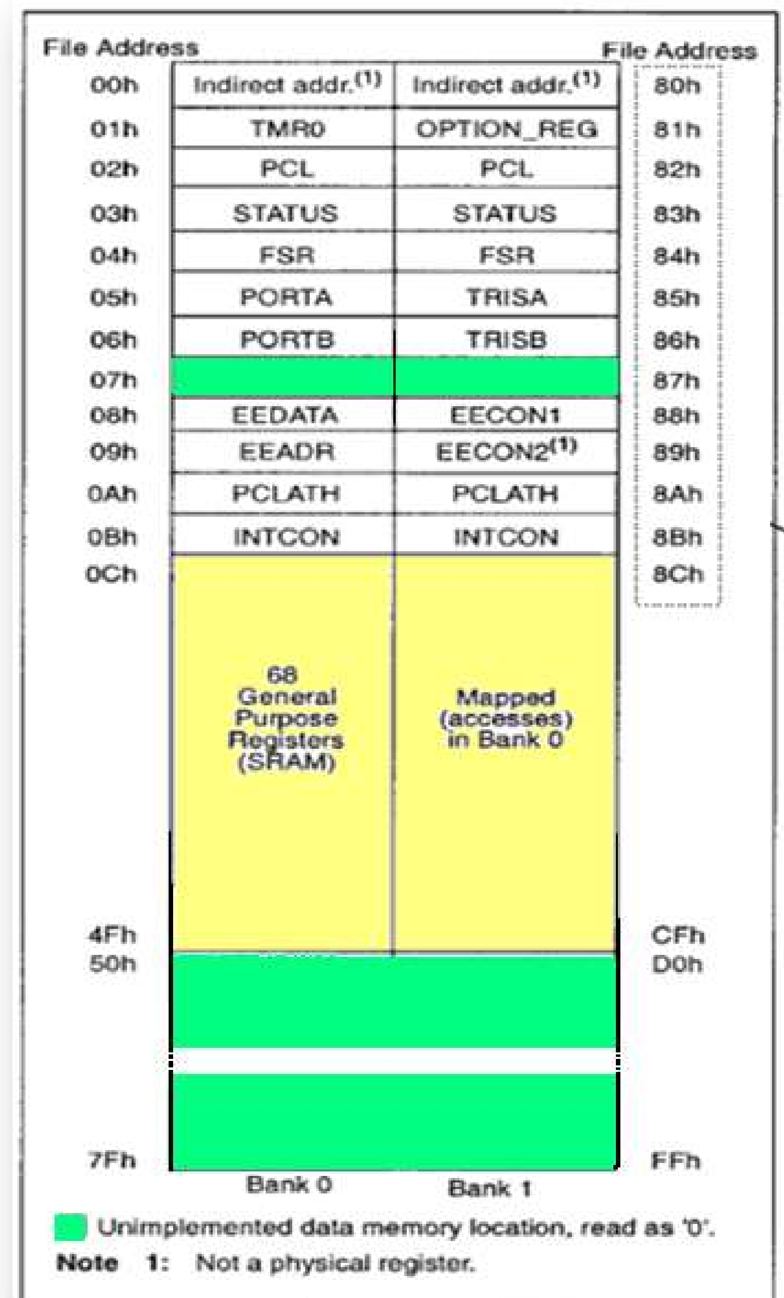
bit 1-0 **FOSC1:FOSC0**: Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

The PIC 16F84A Memory Organization

Data Memory and Special Function Registers (SFRs)

- *SRAM (volatile)*
- *Banked addressing*

- ***Special Function Registers SFRs***
  - *Locations 01H-0BH in bank 0 and 81H-8BH in bank 1*
  - *Used to communicate with I/O and control the microcontroller operation*
  - *Some of them hold I/O data*
- ***General Purpose Registers***
  - *Addresses 0CH – 4FH (68 Bytes)*
  - *Used for storing general data*



The PIC 16F84A Memory Organization

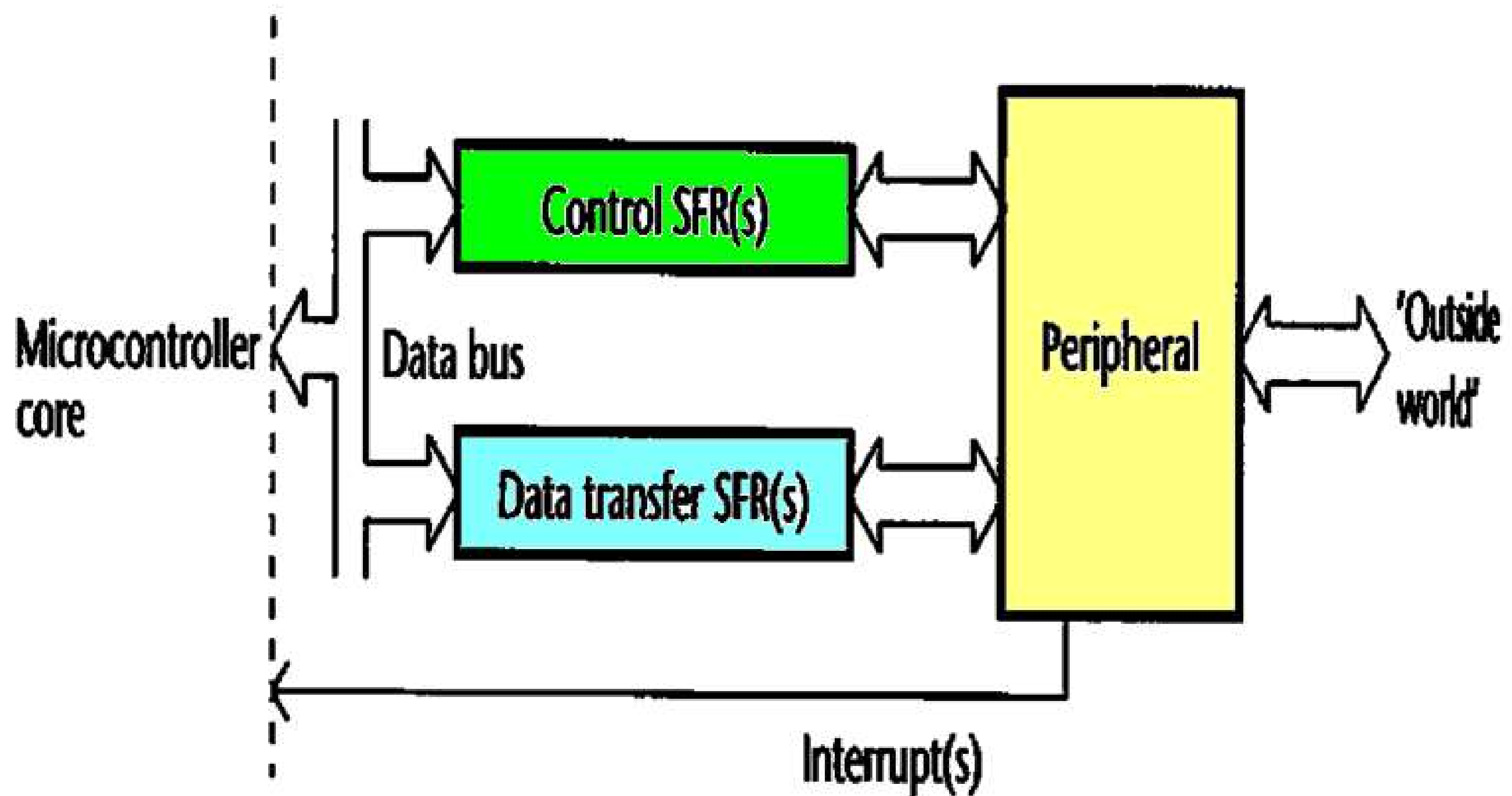
Special Function Registers (SFRs)

Address	Bank 0	Bank 1	Address
00h	INDF	←	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	←	82h
03h	STATUS	←	83h
04h	FSR	←	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	Unimplemented	←	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	89h
0Ah	PCLATH	←	8Ah
0Bh	INTCON	←	8Bh
0Ch - 4Fh	GPR	←	8Ch - CFh

INDF : Data memory contents by indirect addressing
TMR0 : Timer counter
PCL : Low order 8 bits of program counter
STATUS : Flag of calculation result
FSR : Indirect data memory address pointer
PORTA : PORTA DATA I/O
PORTB : PORTB DATA I/O
EEDATA : Dtata for EEPROM
EEADR : Address for EEPROM
PCLATH : Write buffer for upper 5 bits of the program counter
INTCON : Interruption control
OPTIN_REG : Mode set
TRISA : Mode set for PORTA
TRISB : Mode set for PORTB
EECON1 : Control Register for EEPROM
EECON2 : Write protection Register for EEPROM

The PIC 16F84A Memory Organization

- ***Special Function Registers (SFRs) interacting with peripherals***

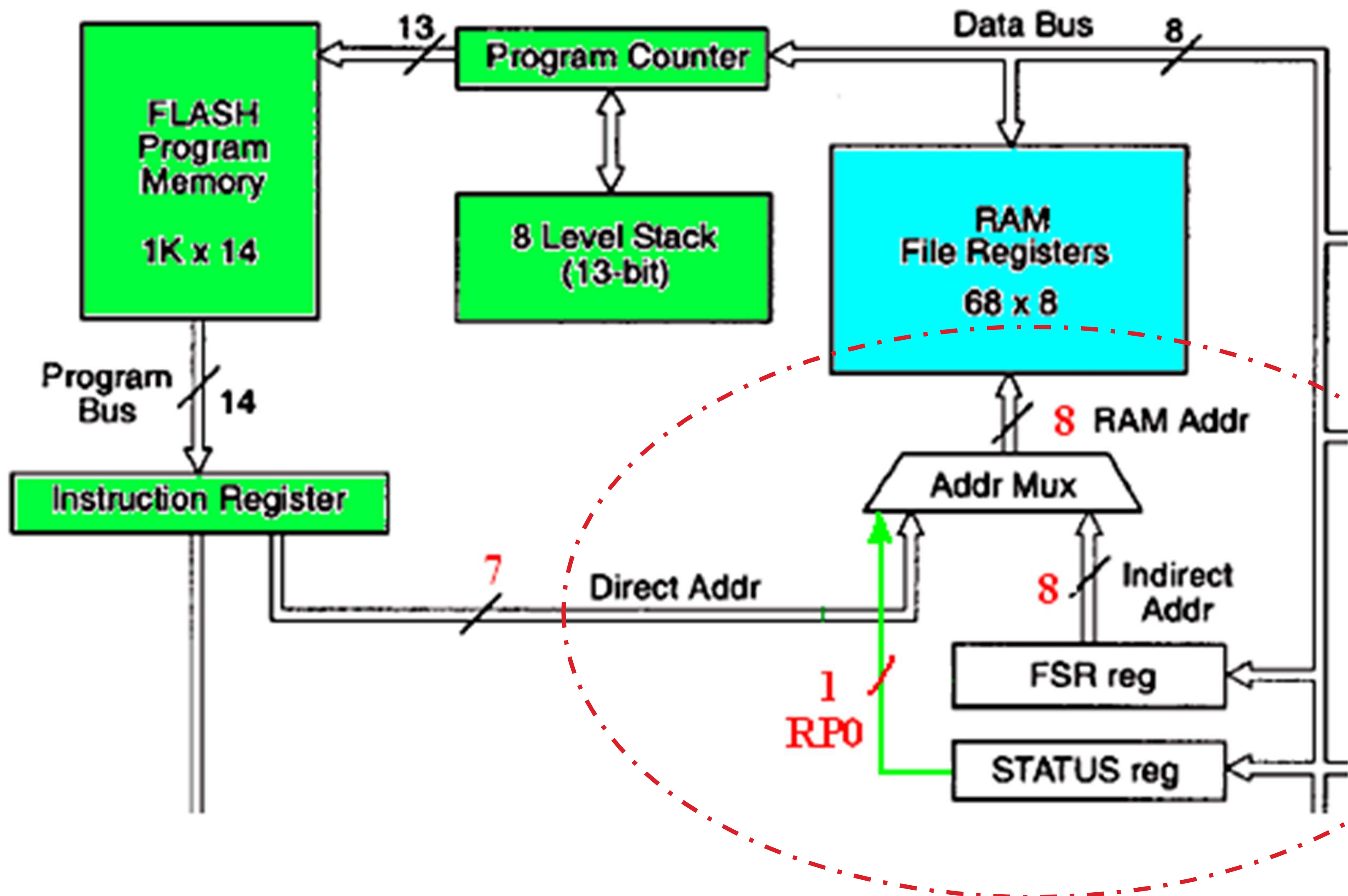


The PIC 16F84A Memory Organization

- ***Data Memory Addressing***
  - For PIC 16F84A, the address of any memory location (File Register is 8 bits
    - One bit is used to select the bank
    - Seven bits to select a location in the bank
  - Bank selection is done through using bits 5 and 6 of the STATUS registers (RP0 and RP1)
  - For the 16F84A, only RP0 is needed since we have two banks
  - In general, two forms to address the RAM (File Registers)
    - **Direct addressing –** the 7-bit address is part of the instruction
    - **Indirect addressing**
      - the 7-bit address is loaded in lower 7 bits of the *File Select Register* (FSR , 04H)
      - Bank selection is done using the most significant bit of FSR and the IRP bit in the STATUS register

The PIC 16F84A Memory Organization

- ***Data Memory Addressing***



The PIC 16F84A Memory Organization

- ***The STATUS Register (03H, 83H)***

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 7-6 **Unimplemented:** Maintain as '0'

bit 5 **RP0**: Register Bank Select bits (used for direct addressing)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)

bit 4 **$\overline{TO}$**: 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

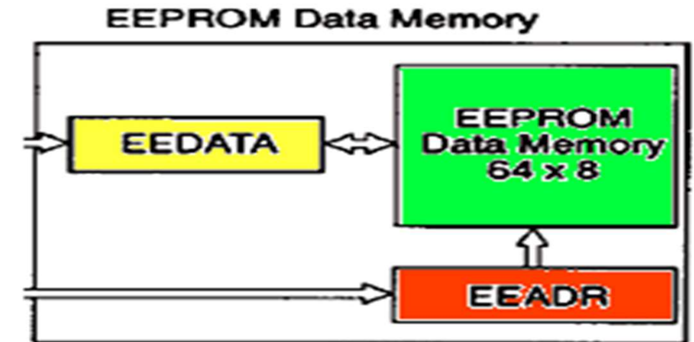
bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

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) (for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: 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.

The PIC 16F84A Memory Organization



- ***Data Related***
  - *EEPROM Data Memory*
    - *64 bytes Non-volatile*
    - *10 000 000 erase/write cycles*
    - *Used to store data that is likely to be needed for long term*
    - *Operation is controlled through EEDATA (08H), EEADR (09H), EECON1 (88H), and EECON2 (89H) SFRs*
    - ***To read a location***
      - *store the address in EEADR and set the RD bit in EECON1*
      - *data is copied to EEDATA register*
    - ***To write to a location***
      - *data and address are placed in EEDATA and EEADR, respectively*
      - *enable writing by setting the WREN bit in EECON1 SFR*
      - *store 55H then AAH in EECON2*
      - *commit writing by enabling the WR bit*
      - *Once the write is done, the EEIF flag is set in EECON1.*

The PIC 16F84A Memory Organization

- ***The EECON1 Register (88H)***

U-0	U-0	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
—	—	—	EEIF	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **EEIF:** EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **WRERR:** EEPROM Error Flag bit
1 = A write operation is prematurely terminated
(any $\overline{MCLR}$ Reset or any WDT Reset during normal operation)
0 = The write operation completed

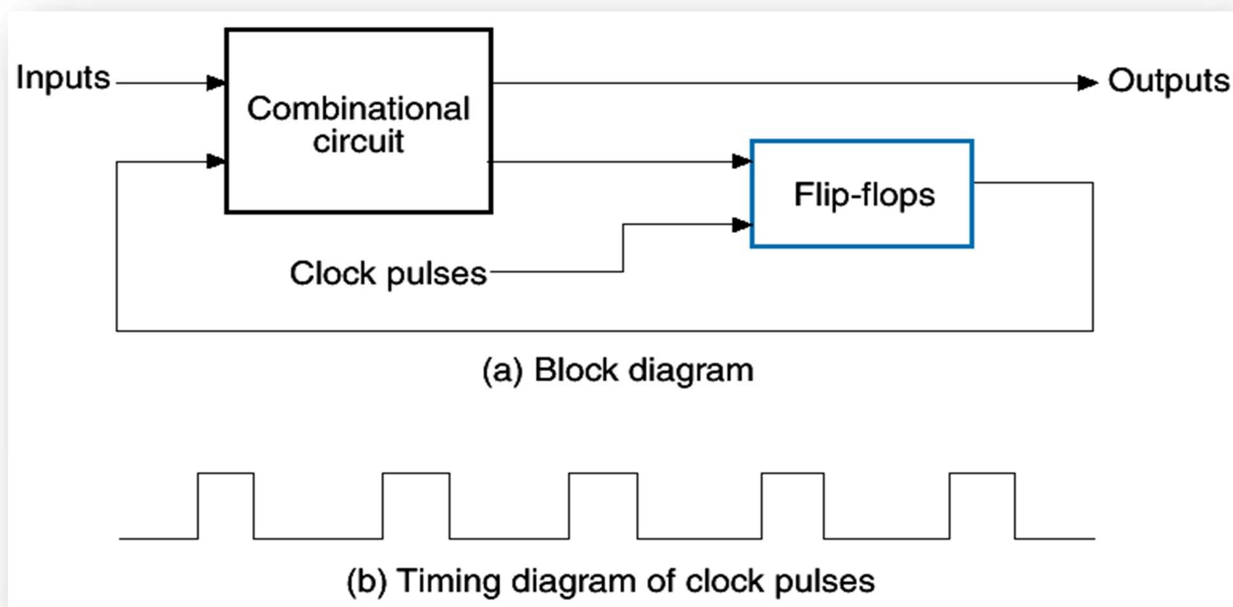
bit 2 **WREN:** EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a write cycle. The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read RD is cleared in hardware. The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

Some Issues of Timing

- **The Clock**
  - The microcontroller is made up of combinational and sequential logic. Thus, it requires a clock !
  - Clock – a continuously running fixed frequency logic square wave
  - Timers, counters, serial communication functions are also dependent on the clock
  - Operating frequency has direct impact on power consumption
  - Every microcontroller has a range for its clock



(a) Block diagram

(b) Timing diagram of clock pulses

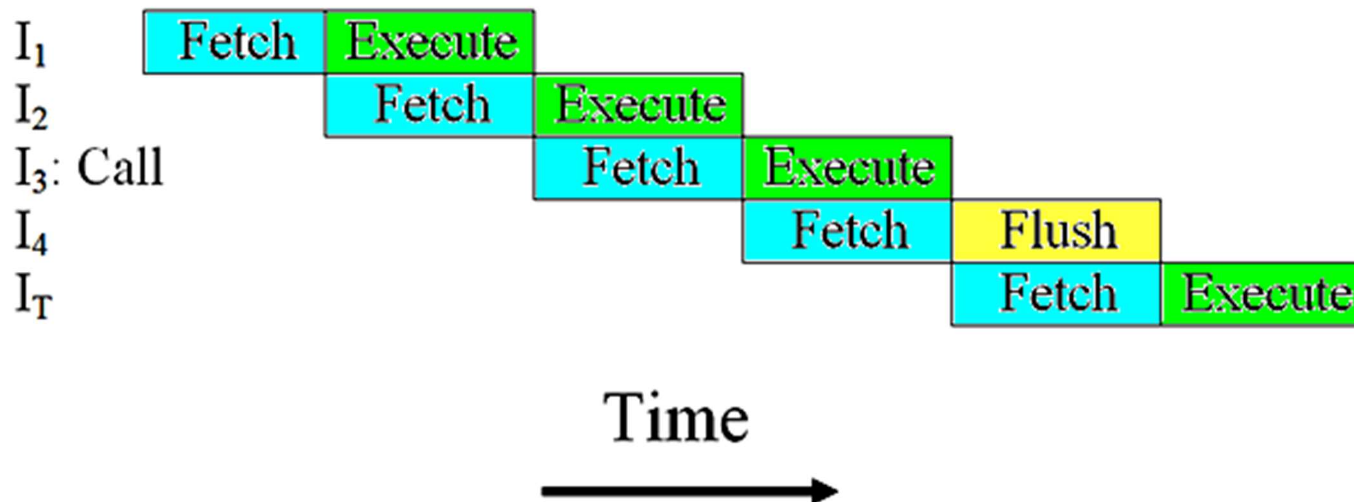
Some Issues of Timing

- **Instruction Cycle**
  - The main clock is divided by a fixed value (4 in the 16 series) into a lower-frequency signal
  - The cycle time of this signal is called the ***instruction cycle***
  - The primary unit of time in the action of processor

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

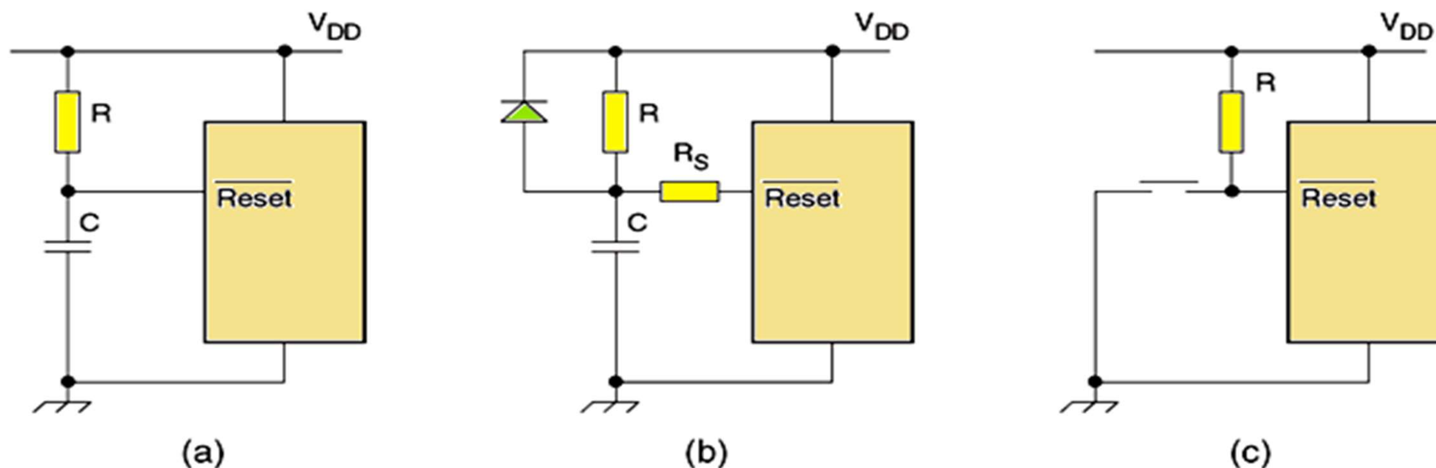
Some Issues of Timing

- **Pipelining**
  - Every instruction in the computer has to be fetched from memory and then executed. These steps are usually performed one after another
  - The CPU can be designed to fetch the next instruction while executing the current instruction. This improves performance significantly!
  - This is called ***Pipelining***
  - All PIC microcontrollers implement pipelining (RISC+Harvard make it easy)

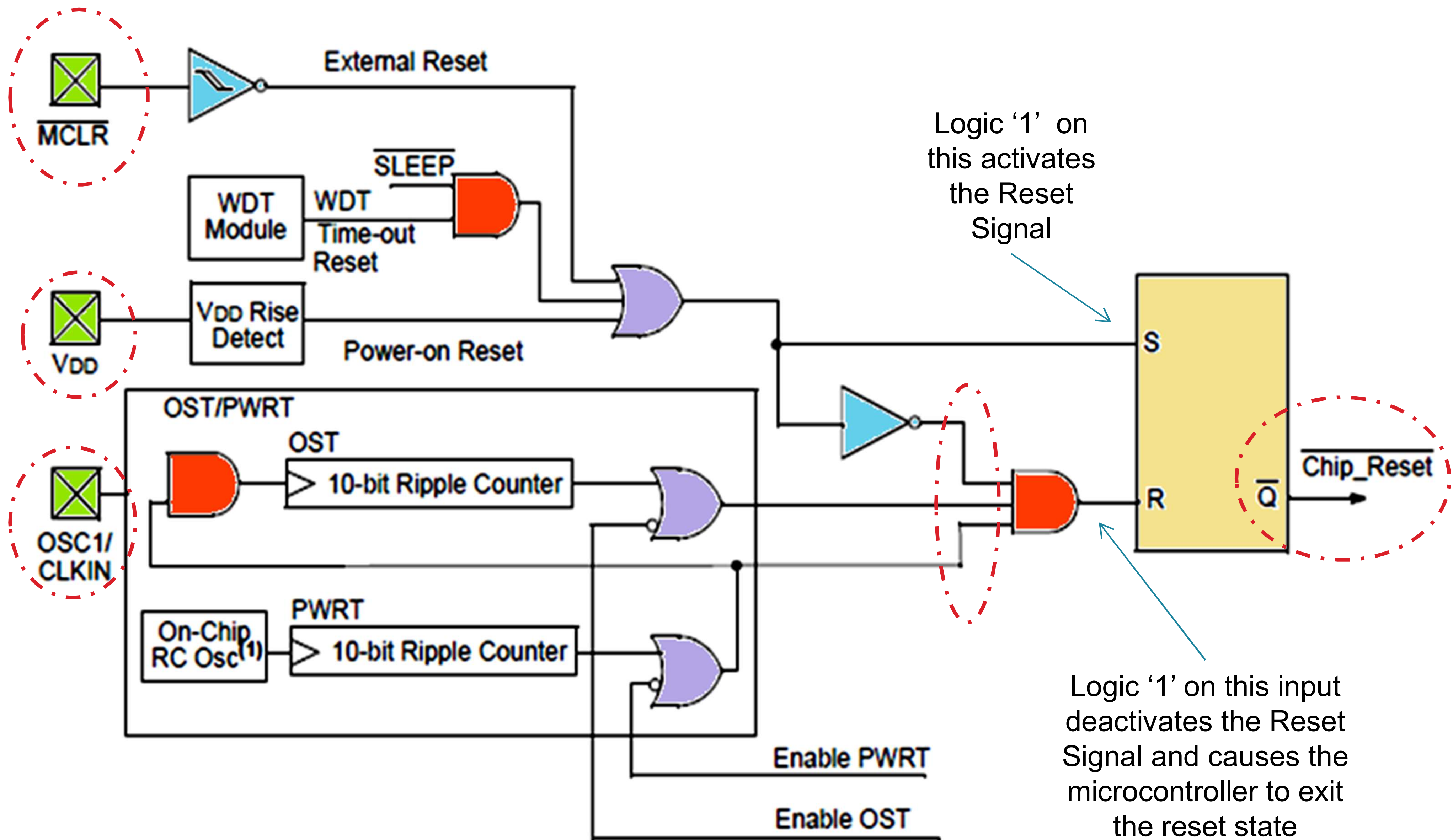


Power-up and Reset

- On power-up, the microcontroller must start to execute the program stored in the program memory from its beginning (address 0000H)
- A specialized circuit inside the microcontroller detects this and is responsible for putting the microcontroller in the ***reset state***:
  - the program counter is set to zero
  - the SFRs are set such that the peripherals are safe and disabled
- Another way to put the microcontroller in the reset state is to apply logic zero to the Master Clear input (MCLR)
- Some reset circuit configurations

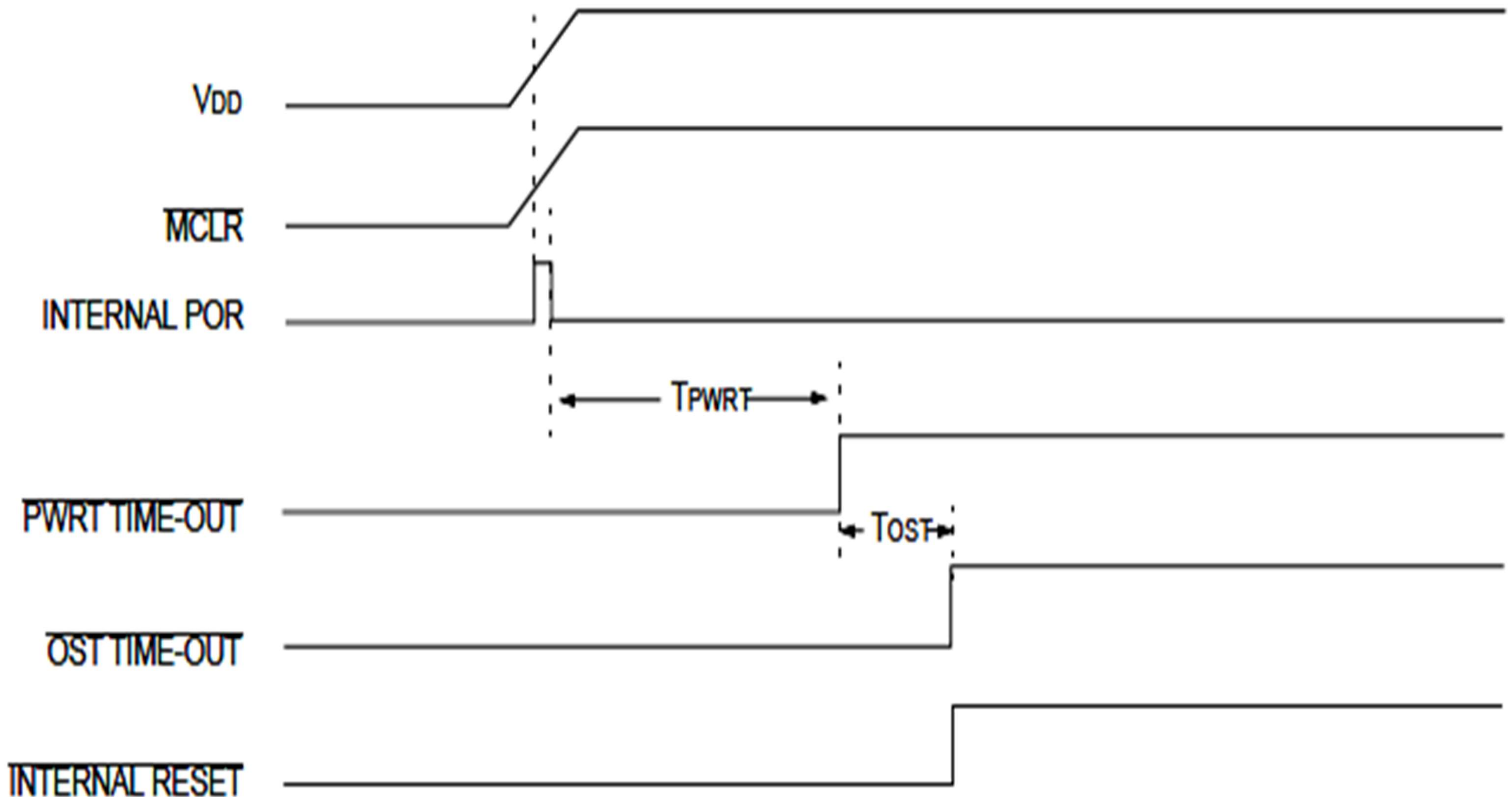


The 16F84A on-Chip Reset Circuit



The 16F84A on-Chip Reset Circuit

Example on reset timing when MCLR is connected to VDD



Summary

- The PIC 16F84A series is a diverse and cost effective family of microcontrollers
- The PIC 16F84A is pipelined RISC processor with Harvard architecture
- The PIC 16F84A has three different memory types
- An important memory area is the Special Function Register area which act as link between the CPU and peripherals
- Reset operation must be understood for proper operation of the microcontroller

Starting to Program

Chapter 4
Sections 1-4 , 10

Dr. Iyad Jafar

Outline

- Introduction
- Program Development Process
- The PIC 16F84A Instruction Set
- Examples
- The PIC 16F84A Instruction Encoding
- Assembler Details
- Sample Programs

Introduction

- Every computer can recognize and execute a group of instructions called the *Instruction Set*
- These instructions are represented in binary (*machine code*)
- *A program* is a sequence of instructions drawn from the instruction set and combined to perform specific operation
- To run the program:
  - It is loaded in binary format in the system memory
  - The computer steps through every instruction and execute it
  - Execution continues unless something stops it like the end of program or an interrupt

How to Write Programs

- **Machine code**
  - The binary representation of instructions
  - Slow, tedious, and error-prone

```
00 0111 0001 0101
```

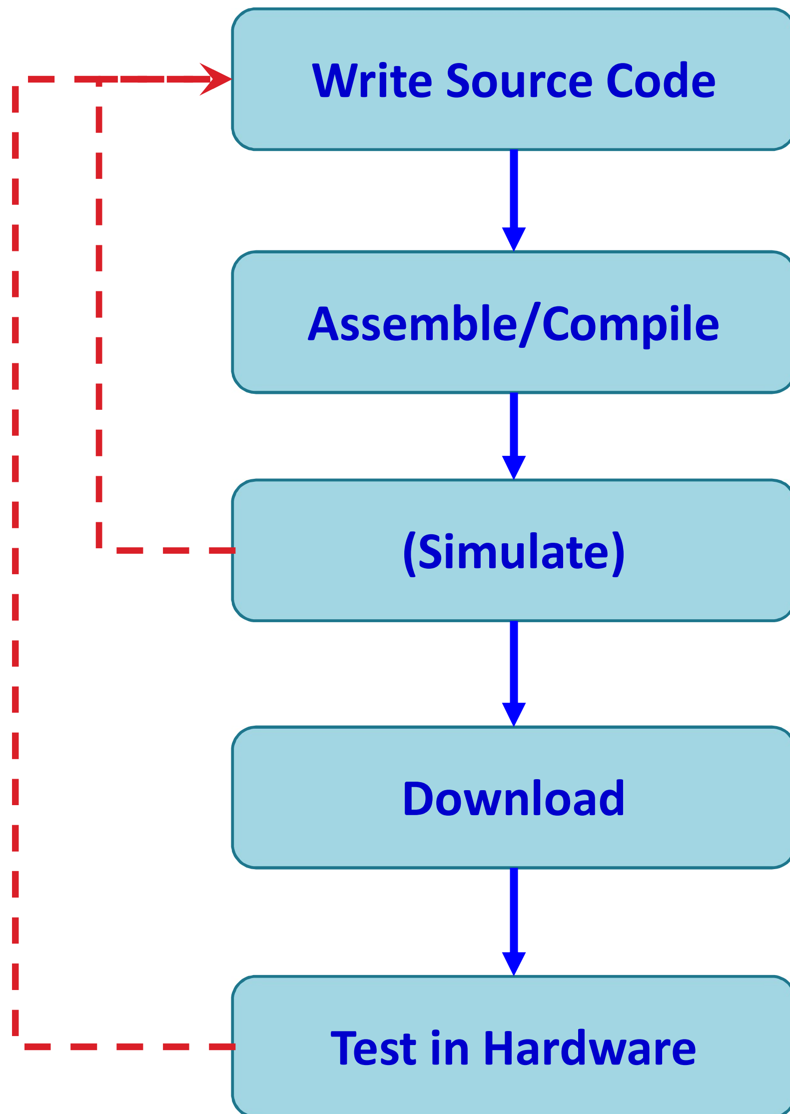
- **Assembly**
  - Each instruction is given a *mnemonic*
  - A program called *Assembler converts to machine code*
  - Rather slow and inefficient for large and complex programs

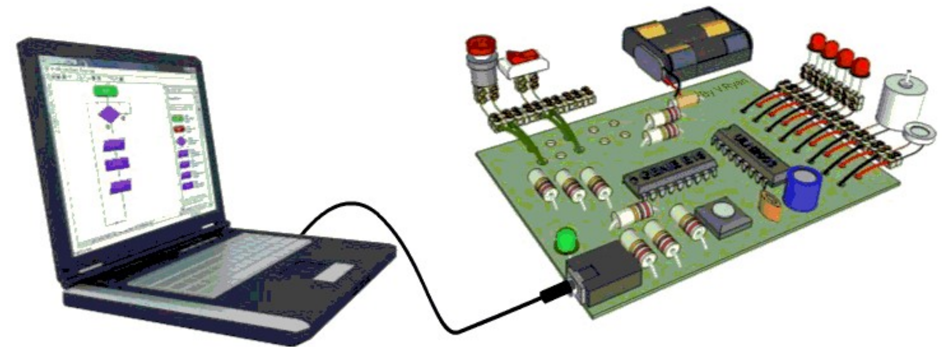
```
addw NUM, w
```

- **High-level language**
  - Use English-like commands to program
  - A program called *Compiler* converts to machine code
  - Easy !! The program could be inefficient !

```
for (i=0; i<10; i++) sum += a[i];
```

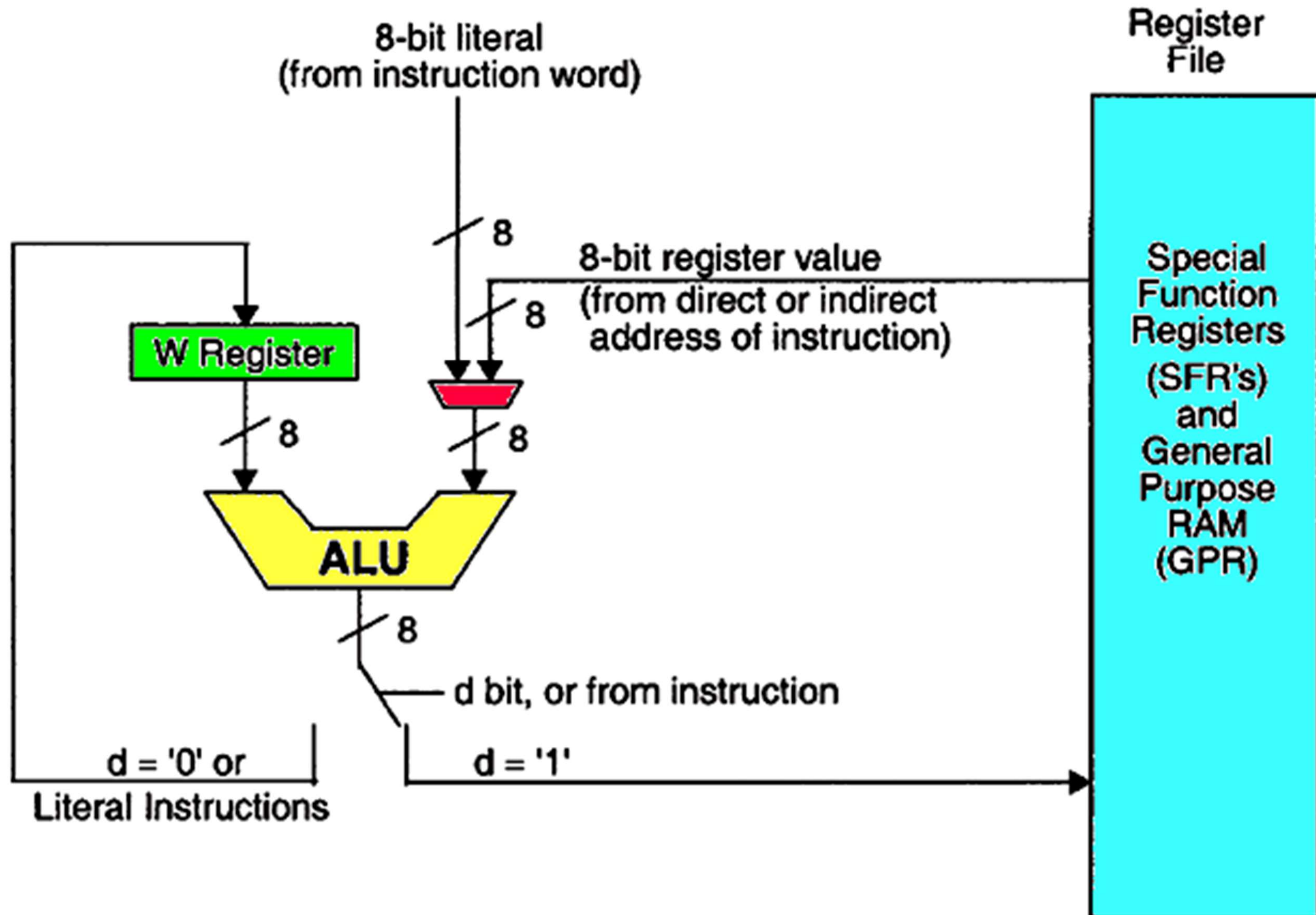
Program Development Process





The PIC 16 Series Instruction Set

- **The PIC 16 Series ALU**



The PIC 16 Series Instruction Set

- 35 instructions represented using 14 bits
- The binary code of the instruction itself is called the *Opcode*
- Most of these instruction operate/use on values called *Operands (ranging from no operands to two)*

- **Three categories of instructions**
  1. Byte-oriented file register operations
  2. Bit-oriented file register operations
  3. Literal and control operations

- **Type of operations**
  - Arithmetic, logical, data movement, control and miscellaneous

The PIC 16 Series Instruction Set

- **Introduction to PIC 16 ISA**
  - Types of operands
    - A 7-bit address for a memory location in RAM (Register File) denoted by **f**
    - A 3-bit to specify a bit location within an the 8-bit data denoted by **b**
    - A 1-bit to determine the destination of the result denoted by **d**
    - A 8-bit number for literal data or 11-bit number for literal address denoted by **k**

The PIC 16 Series Instruction Set

- Examples
  - **clrw**
    - Clears the working register W
  - **clrf f**
    - Clears the memory location specified by the 7-bit address f
  - **addwf f , d**
    - Adds the contents of the working register W to the memory location with 7-bit address in f. the result is saved in *W if d =0, or in f if d = 1*
  - **bcf f , b**
    - Clears the bit in position specified by b in memory location specified by 7-bit address f
  - **addlw k**
    - Adds the content of W to the 8-bit value specified by k. The result is stored back in W

The PIC 16 Series Instruction Set

Byte-oriented File Register Operations

- **Format:** `op f, d`
  - **`op`**: operation
  - **`f`**: address of file or register
  - **`d`**: destination (0: working register, 1: file register)

- **Example:**

```
addwf   PORTA, 0
```

Adds the contents of the working register and register `PORTA` then puts the result in the working register.

The PIC 16 Series Instruction Set

Bit-oriented File Register Operations

- **Format:** **`op f, b`**
  - **`op`**: operation
  - **`f`**: address of file or register
  - **`b`**: bit number, 0 through 7

- **Example:**

```
bsf STATUS, 5
```

Sets to 1 Bit 5 of register `STATUS`.

The PIC 16 Series Instruction Set

Literal and Control Operations

- **Format:** **`op k`**
  - **`op`**: operation
  - **`k`**: literal, an 8-bit if data or 11-bit if address

- **Examples:**

```
addlw   5
```

Adds to the working register the value 5.

```
call    9
```

Calls the subroutine at address 9.

The PIC 16 Series Instruction Set

Arithmetic Instructions

Mnemonic	Operands	Description	Cycles	Status Affected
ADDWF	f, d	Add W and f	1	C,DC,Z
COMF	f, d	Complement f	1	Z
DECF	f, d	Decrement f	1	Z
INCF	f, d	Increment f	1	Z
SUBWF	f, d	Subtract W from f	1	C,DC,Z
ADDLW	k	Add literal and W	1	C,DC,Z
SUBLW	k	Subtract W from literal	1	C,DC,Z

d = 0 , result is stored in W
d = 1 , result is stored in F

The PIC 16 Series Instruction Set

Logic Instructions

Mnemonic	Operands	Description	Cycles	Status Affected
ANDWF	f, d	AND W with f	1	Z
IORWF	f, d	Inclusive OR W with f	1	Z
XORWF	f, d	Exclusive OR W with f	1	Z
ANDLW	k	AND literal with W	1	Z
IORLW	k	Inclusive OR literal with W	1	Z
XORLW	k	Exclusive OR literal with W	1	Z

d = 0 , result is stored in W
d = 1 , result is stored in F

The PIC 16 Series Instruction Set

Data Movement Instructions

Mnemonic	Operands	Description	Cycles	Status Affected
MOVF	f, d	Move f	1	Z
MOVWF	f	Move W to f	1	
SWAPF	f, d	Swap nibbles in f	1	
MOVLW	k	Move literal to W	1	

d = 0 , result is stored in W
d = 1 , result is stored in F

The PIC 16 Series Instruction Set

Control Instructions

Mnemonic	Operands	Description	Cycles	Status Affected
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	
CALL	k	Call subroutine	2	
GOTO	k	Go to address	2	
RETFIE	-	Return from interrupt	2	
RETLW	k	Return with literal in W	2	
RETURN	-	Return from Subroutine	2	

d = 0 , result is stored in W , d = 1 , result is stored in F

The PIC 16 Series Instruction Set

Miscellaneous Instructions

Mnemonic	Operands	Description	Cycles	Status Affected
CLRF	f	Clear f	1	Z
CLRW	-	Clear W	1	Z
NOP	-	No Operation	1	
RLF	f, d	Rotate Left f through Carry	1	C
RRF	f, d	Rotate Right f through Carry	1	C
BCF	f, b	Bit Clear f	1	
BSF	f, b	Bit Set f	1	
CLRWDT	-	Clear Watchdog Timer	1	TO',PD'
SLEEP	-	Go into standby mode	1	TO',PD'

d = 0 , result is stored in W , d = 1 , result is stored in F

The PIC 16 Series Instruction Set

Examples

Instruction	Operation	Flags Affected
bcf 0x31, 3	clear bit 3 in location 0x31	None
bsf 0x04 , 0	set bit 0 location 0x04	None
bsf STATUS , 5	set bit 5 in STATUS register to select bank 1 in memory	None
bcf STATUS , C	clear the carry bit in the status register	None
addlw 4	Adds 4 to working register W and store the result in back in W	C, DC, Z
addwf 0x0C, 1	Add the content of location 0x0C to W and store the result in 0CH (d =1)	C, DC , Z
sublw 10	Subtract W from 10 and put the result in W	C, DC, Z
subwf 0x3C, 0	Subtract W from contents of location 0x3C and store the result in W	C, DC, Z

The PIC 16 Series Instruction Set

Examples

Instruction	Operation	Flags Affected
incf 0x06, 0	Increment location 0x06 by 1 and store result in W	Z
decf TEMP, 1	Decrement location TEMP by 1 and store in TEMP	Z
compf 0x10, 1	Complement the value in location 10H and store in 0x10	Z
andlw B’11110110’	AND literal value 11110110 with W and store result in W	Z
andwf 0x33, 1	AND location 0x33 with W and store result in 0x33	Z
iorlw B’00001111’	Inclusive-or W with 00001111	Z
iorwf X1 , 0	Inclusive-or W with location X1 and store result in W	Z
xorlw B’01010101’	Exclusive-or W with 01010101	Z
xorwf 0x2A , 0	Exclusive-or W with location 0x2A and store result in W	

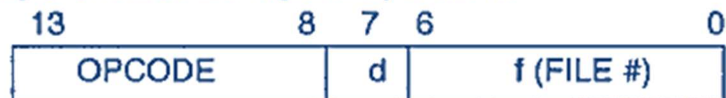
The PIC 16 Series Instruction Set

Examples

Instruction	Operation	Flags Affected
clrw	Clear W	**Z**
clrf 0x01	Clear location 0x01	**Z**
movlw 18	Move literal value 18 into W	**NONE**
movwf 0x40	Move contents of W to location 0x40	**NONE**
movf 0x21 , 0	Move contents of location 0x21 to W	**Z**
movf 0x21 , 0x33	Incorrect syntax	**--**
movwf 0x1B , 1	Incorrect syntax	**--**
swapf T1 , 1	Swap 4-bit nibbles of location T1	**NONE**
swapf DATA , 0	Move DATA to W, swap nibbles, no change on DATA	**NONE**
rlf TEMP , 1	Rotate contents of location TEMP to left by one bit position through the C flag	**C**
rlf 0x25 , 0	Copy contents of 0x25 to W and rotate to left by one bit position through the C flag	**C**

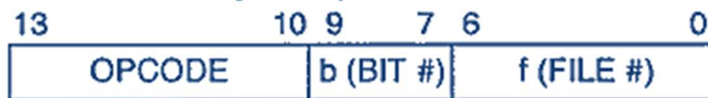
The PIC 16 Series Instruction Set Encoding

Byte-oriented file register operations



d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

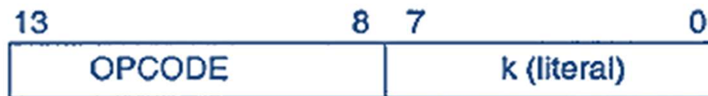
Bit-oriented file register operations



b = 3-bit bit address
f = 7-bit file register address

Literal and control operations

General



k = 8-bit immediate value

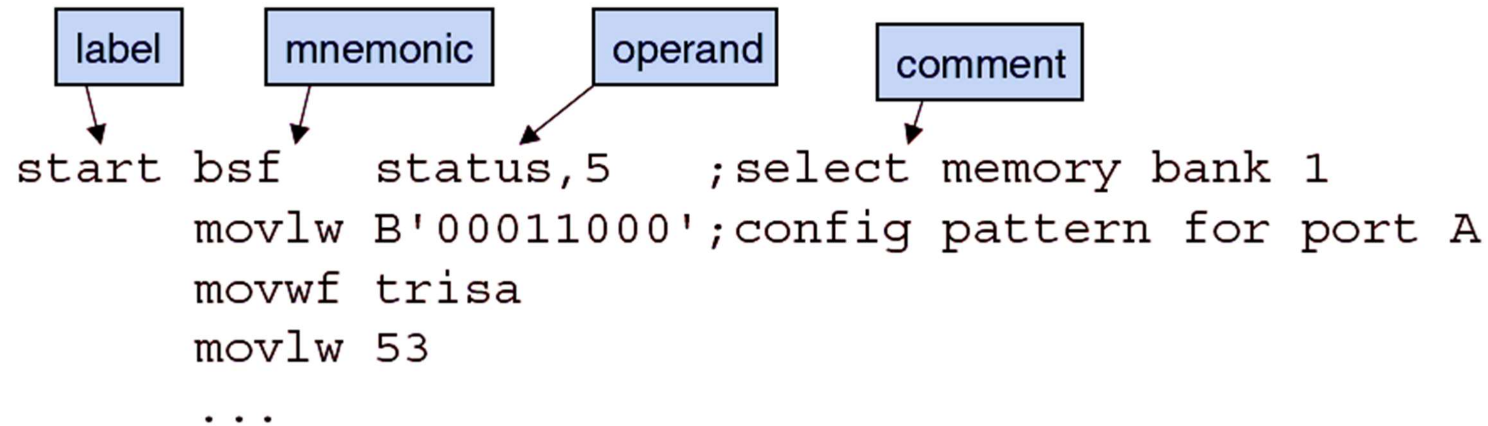
CALL and GOTO instructions only

k = 11-bit immediate value

Check Appendix A for opcode binary codes

Assembler Details

- Any assembler line may have up to four different elements



- We can specify values in different bases in assembler programs

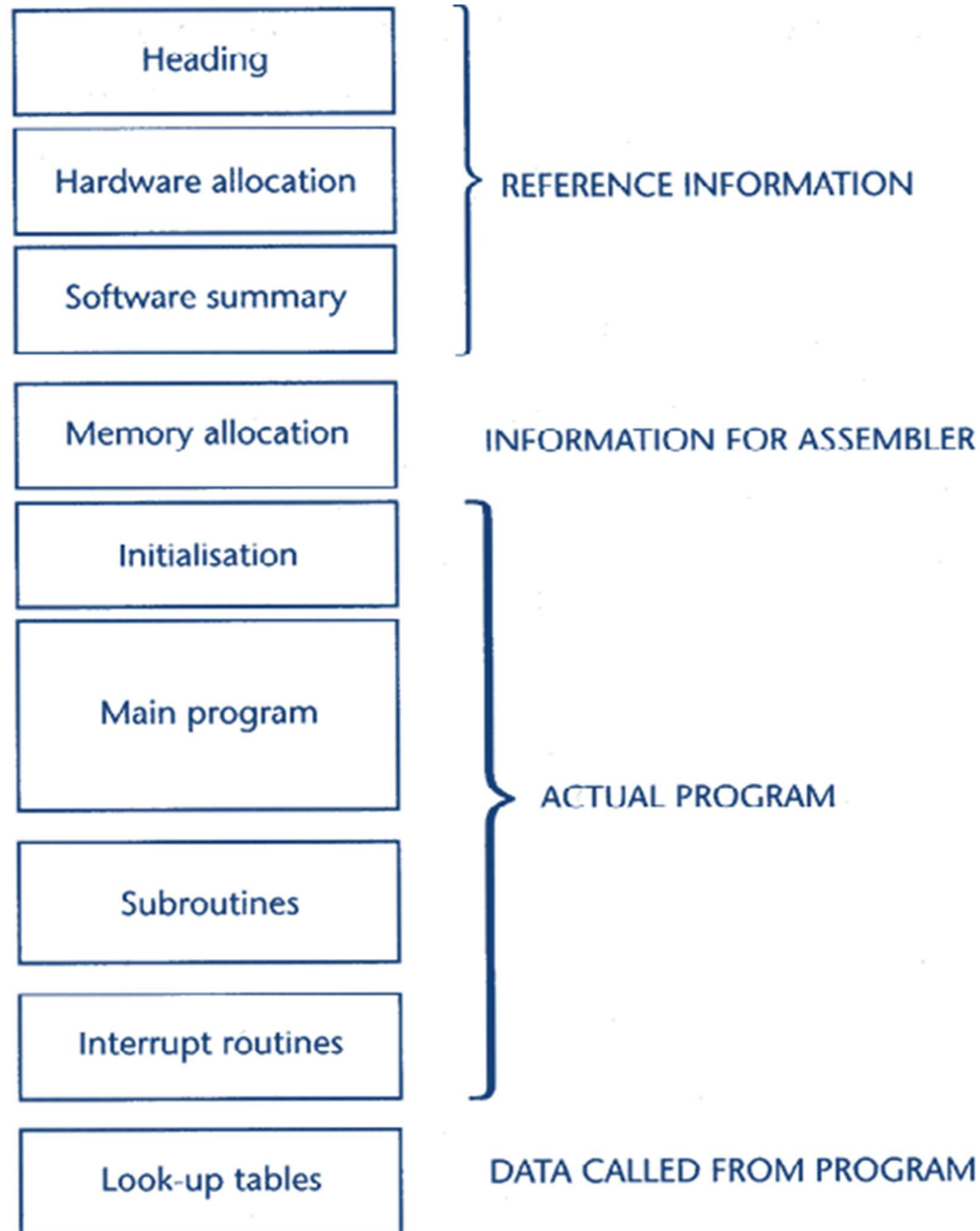
Radix	Example
Decimal	D’255’
Hexadecimal	H’8d’ or 0x8d
Octal	O’574’
Binary	B’01011100’
ASCII	‘G’ or A’G’

Assembler Details

- **Assembler directives**
  - These are assembler-specific commands to aid the processing of assembly programs

Directive	Command to Assembler
org	Set program origin
equ	Define an assembly constant; this allows us to assign a value to a label
end	End program block
#include	Include additional source file

Program Structure



Sample Program 1

- Write a program to add the numbers stored in locations 31H, 45H, and 47H and store the result in location 22H

Sample Program 1

```
; ************************* EQUATES ********************************
STATUS          equu    0x03            ; define SFRs
RP0             equ     5
;************************* VECTORS ********************************
                org     0x0000          ; reset vector
                goto    START
                org     0x0004
INVEC           goto    INVEC           ; interrupt vector
;************************  MAIN PROGRAM ************************
START           bcf     STATUS , RP0     ; select bank 0
                movf    0x31 , 0        ; put first number in W
                addwf   0x45 , 0        ; add second number
                addwf   0x47 ,  0       ; add third number
                movwf   0x22            ; save result in 0x22
DONE            goto    DONE            ; endless loop
                end
```

Sample Program 2

- Write a program to swap the contents of location 0x33 with location 0x11

Sample Program 2

```
; ************************* EQUATES *********************************
STATUS      equ     0x03            ; define SFRs
RP0         equ     5
;************************** VECTORS *********************************
            org     0x0000          ; reset vector
            goto    START
            org     0x0004
INVEC       goto    INVEC           ; interrupt vector
;************************* MAIN PROGRAM *************************
START       bcf     STATUS , RP0    ; select bank 0
            movf    0x33 , 0        ; put first number in W
            movwf   0x22            ; store the 1st number temporarily
            movf    0x11 ,  0       ; get 2nd number
            movwf   0x33            ; store 2nd in place of 1st
            movf    0x22 , 0        ; get 1st number from 0x22
            movwf   0x11            ; store 1st  in place of 2nd
DONE        goto    DONE            ; endless loop
            end
```

Summary

- The PIC 16F84A has 35 instructions to perform different computational and control operations
- Programs can be written using different levels of abstraction
- Using assemblers simplifies the program development process
- There exist many IDE to aid writing programs and simulate their behavior before putting them into hardware

Building Assembler Programs

Chapter 5
Sections 1-6

Dr. Iyad Jafar

Outline

- Building Structured Programs
- Conditional Branching
- Subroutines
- Generating Time Delays
- Dealing with Data
- Example Programs

Building Structured Programs

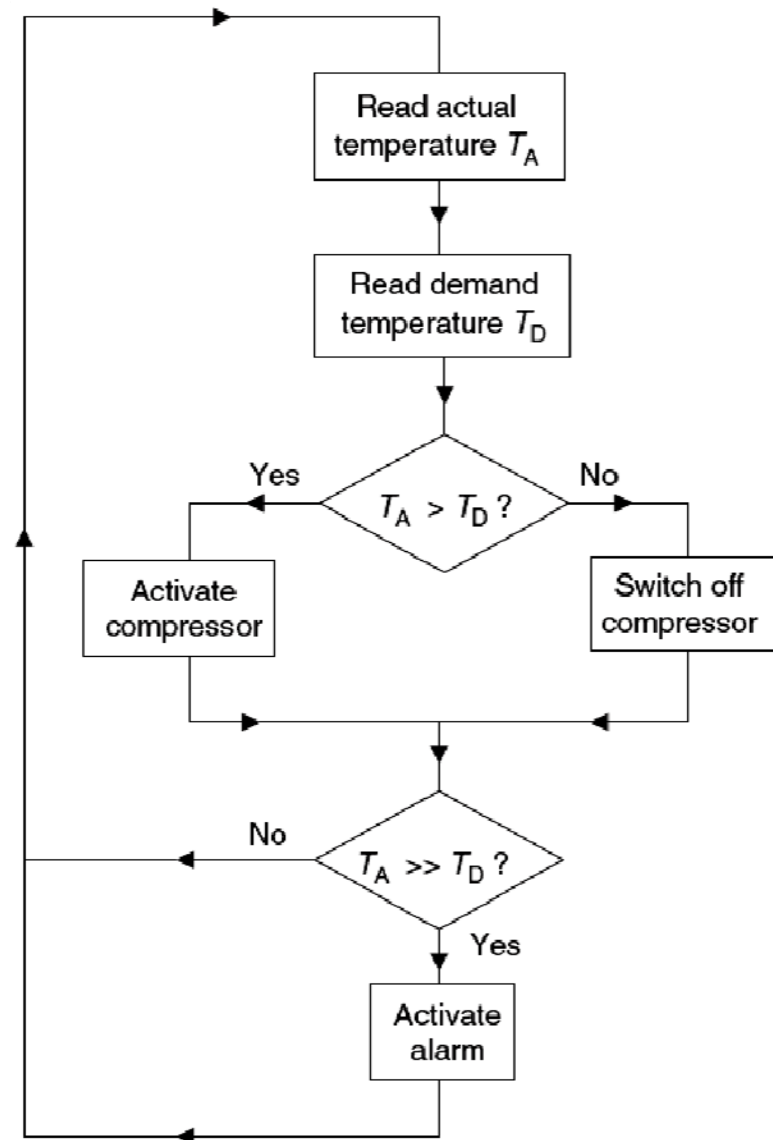
- Writing programs is not an easy task; especially with large and complex programs

- It is essential to put down a design for the program before writing the first line of code

- This involves documenting the programs flow charts and state diagrams

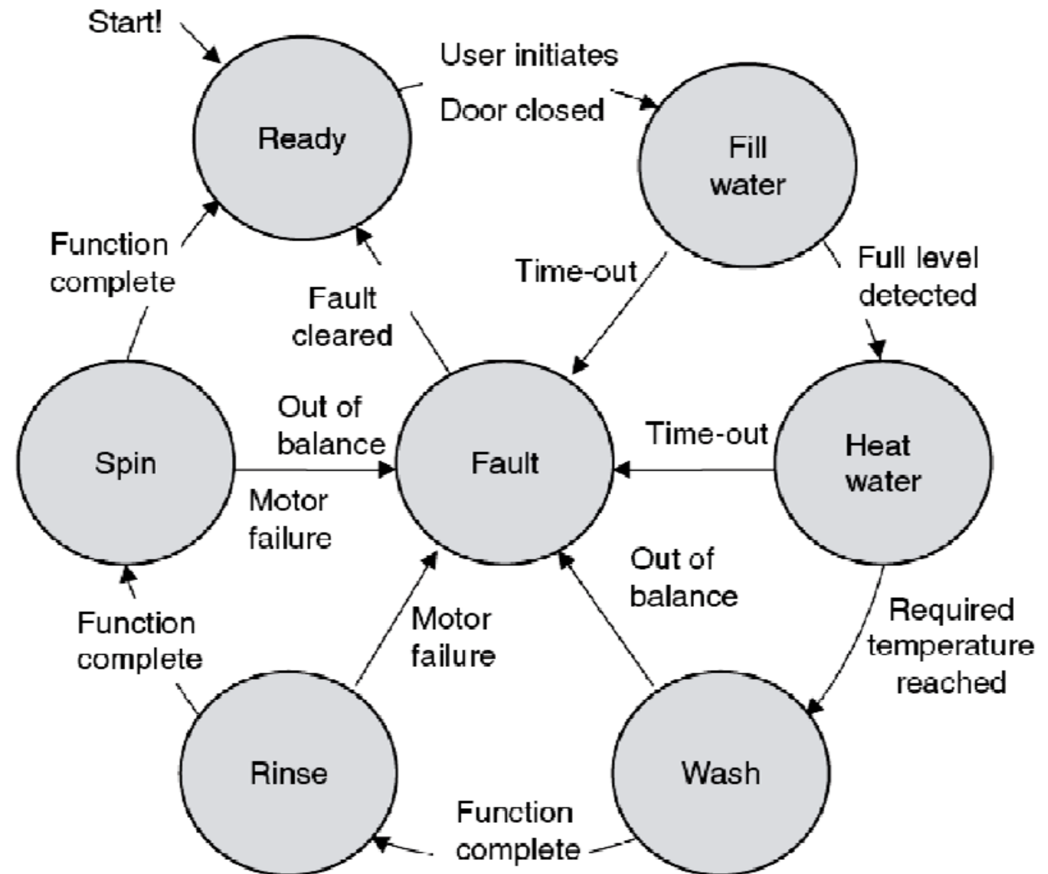
Building Structured Programs

- **Flowcharts**
  - Rectangle for process
  - Diamond for decision



Building Structured Programs

- **State Diagrams**
  - Circle for state
  - Arrow for state transition labeled with condition(s) that causes the transition



Conditional Branching

- Microprocessors and microcontroller should be able to make **decisions**
- This enables them to behave according to the state of logical variables
- The PIC 16 series is not an exception ! They have ***four conditional skip*** instructions
- These instructions *test for a certain condition and skip the following instruction* if the tested condition is true !

Conditional Branching

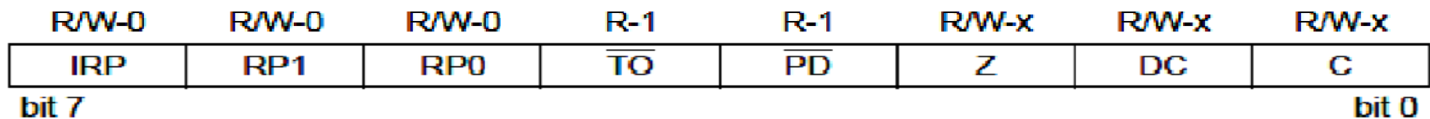
Instruction	Operation	Example
btfsc f , b	Test bit at position b in register f. skip next instruction if the bit is clear '0'	**btfsc STATUS , 5**
btfss f , b	Test bit at position b in register f. skip next instruction if the bit is set '1'	**btfss 0x21 , 1**

Instruction	Operation	Example
decfsz f , d	Decrement the contents of register f by 1 and place the result in W if d = 0 or in f if d = 1. Skip next instruction if the decremented result is zero	**decfsz 0x44, 0**
incfsz f , d	Increment the contents of register f by 1 and place the result in W if d = 0 or in f if d = 1. Skip next instruction if the incremented result is zero	**incfsz 0xd1 , 1**

Conditional Branching

- **Example1:** a program to add two numbers that are stored in locations 0x11 and 0x22. If the addition results in ***no*** carry, the result is stored in location 0x33. otherwise, the result is stored in location 0x44

- The 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

Conditional Branching Example 1

```
STATUS      equ     0x03          ; define SFRs
            org     0x0000        ; reset vector
            goto    START
            org     0x0006
START       movf    0x11  , 0     ; get first number to W
            addwf    0x22 , 0     ; add second number
            btfsc   STATUS , 0    ; check if carry is clear
            goto    C_SET         ;  go to label C_Set if C==1
            movwf   0x33          ; store result in 0x33
            goto    DONE
C_SET       movwf   0x44
DONE        goto    DONE          ; endless loop
            end
```

Conditional Branching

- **Example 2:** Write a program that multiplies the content of location 0x30 by 10.

Conditional Branching

- **Example 3:** The upper and lower bytes of a 16-bit counter are stored in locations COUNTH and COUNTL, respectively. Write a program to decrement the counter until it is zero. Decrementing the counter is allowed if the counter is initially non zero.

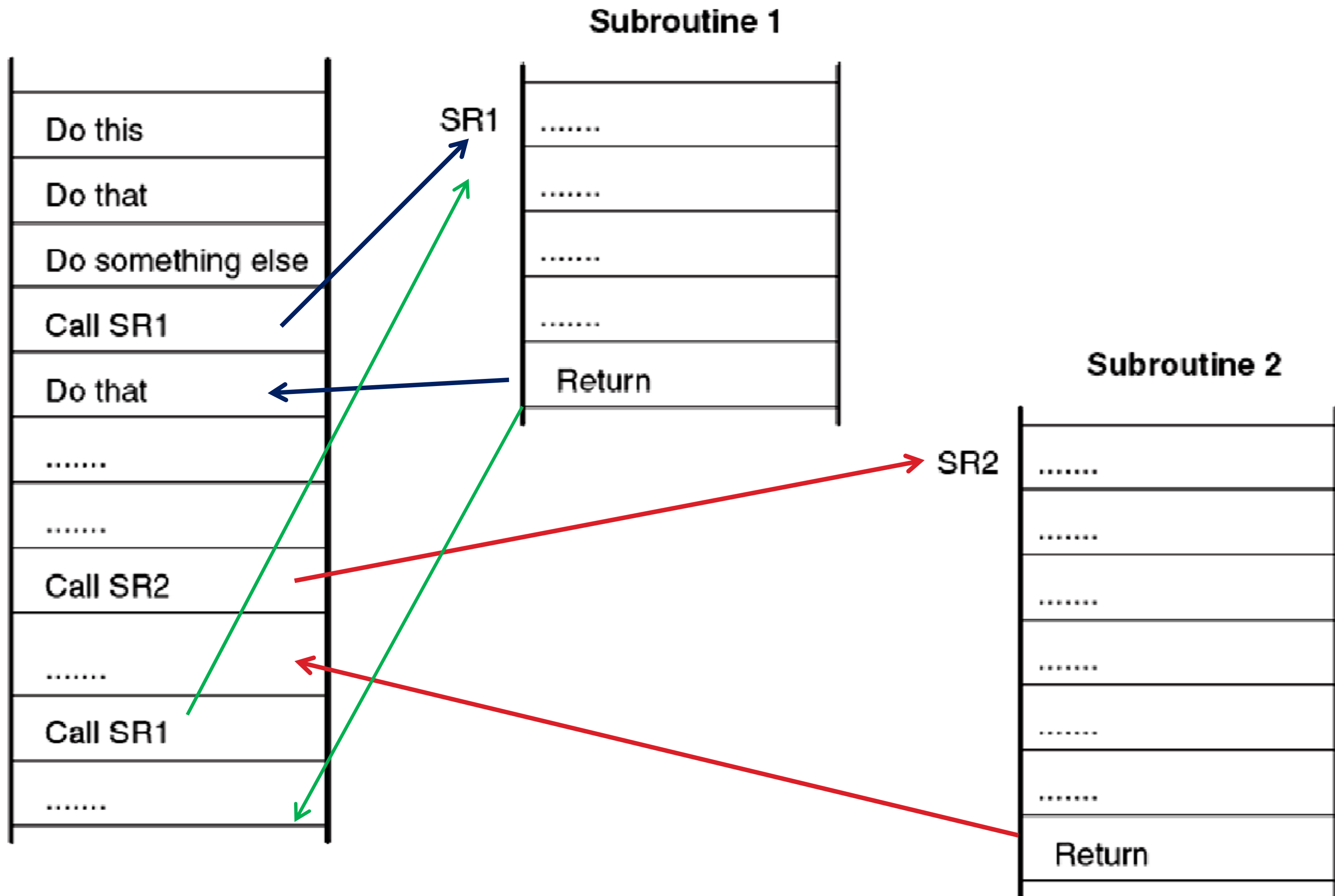
Conditional Branching Example 2

```
COUNTL        equ     0x10              ; lower byte of counter in 0x10
COUNTH        equ     0x11              ; upper byte of counter in 0x11
              #include "P16F84A.INC"
              org     0x0000
START         movf    COUNTL , F        ; check if the both locations are zeros
              btfss   STATUS ,Z         ; if so, then finish
              goto    DEC_COUNTL        ; if COUNTL is not zero, decrement it
              movf    COUNTH , F        ; if it is zero check COUNTH
              btfsc   STATUS ,Z
              goto    DONE              ; if both are zeros, then DONE
              decf    COUNTH, F
DEC_COUNTL    decf    COUNTL,F
              goto    START
DONE          goto    DONE              ; program gets here if both are zeros
              end
```

Subroutines

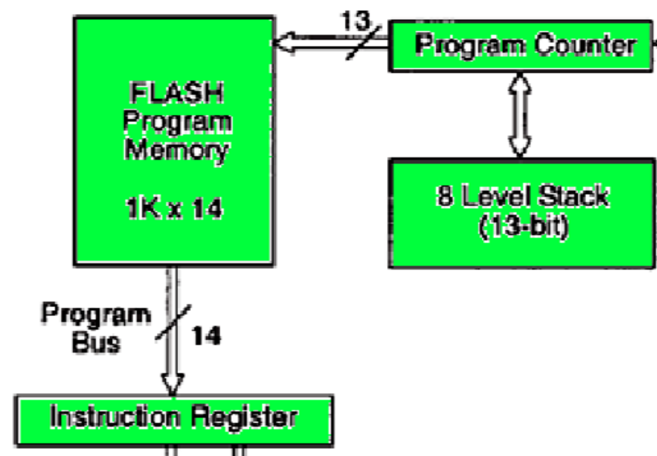
- In many cases, we need to use a block of code in a program in different places
- Instead of writing the code wherever it is needed, we can use *subroutines/functions/procedures*
  - Block of code saved in memory and can be called/used from anywhere in the program
  - When a subroutine is called, execution moves to place where the subroutine is stored
  - Once the subroutine is executed, execution resumes from where it was before calling the subroutine

Subroutines



Subroutines

- The ***program counter*** holds the address of the instruction to be executed
- In order to call a subroutine, the program counter has to be loaded with the address of the subroutine
- Before that, the current value of the PC is saved in ***stack*** to assure that the main program can continue execution from the following instruction once the execution of the subroutine is over



Subroutines

- In PIC, to invoke a subroutine we use the ***CALL*** instruction followed by the address of the subroutine
- The address is usually *specified by a symbolic label in the program*
- To exit a subroutine and return to the main program, we use the ***RETURN*** or ***RETLW*** instructions

Subroutines

- **Example 4:** Write a program that uses a subroutine to multiply the contents of locations 0x30 and 0x31 and then return the result in the working register.

Subroutines - Example

```
STATUS      equ     0x03        ; define SFRs
            org     0x0000      ; reset vector
            goto    START
            org     0x0005
START       ........
            movlw   0x15        ; pass the first number
            movwf   0x30
            movlw   0x09        ; pass the second number
            movwf   0x31
            call    multiply    ; call the subroutine
            ........
            movlw   0x05        ; pass the first number
            movwf   0x30
            movlw   0x04        ; pass the second number
            movwf   0x31
            call    multiply    ; call the subroutine
            ........
DONE        goto    DONE        ; endless loop
```

Example - Continued

```
multiply    clrw
Repeat      addwf  0x30, 0    ; repeated  addition
            decfsz 0x31, 1    ; counter
            goto   repeat
            return

            end
```

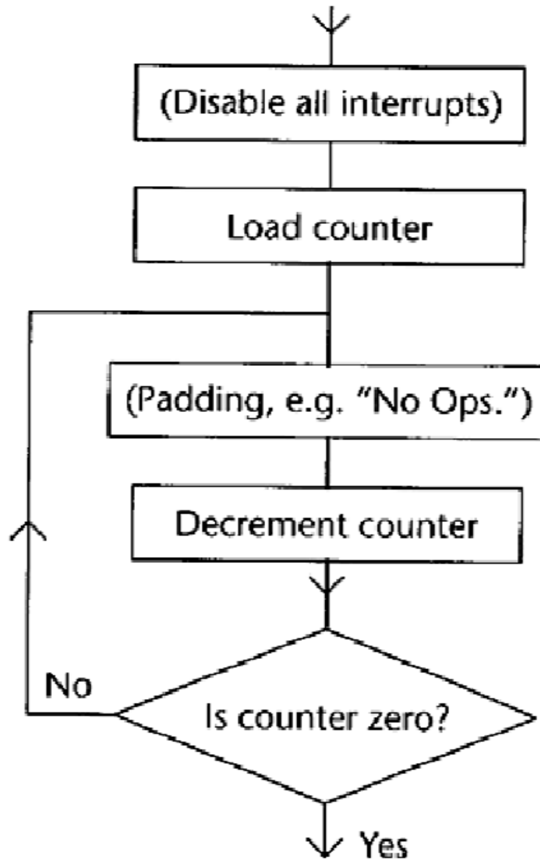
Generating Time Delays

- In many applications, it is required to delay the execution of some block of code; i.e. a time delay!
- In most microcontrollers this can be done by
  - Software
  - Hardware (Timers)
- To generate time delay using software, *let the microcontroller execute non useful instructions* for certain number of times!
- If we know the clock frequency and the cycles to execute each instruction we can generate different delays

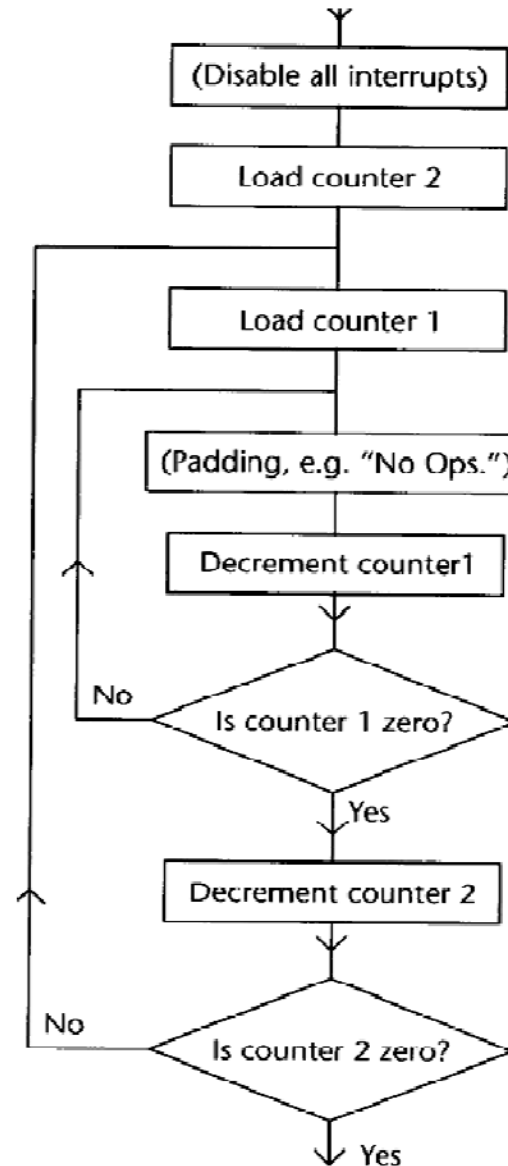
$$Delay = \#cycles \text{ x } clock\ cycle\ time$$
$$= \#\text{cycles x } 4 / F_{osc}$$

Generating Time Delays

- Structure of Delay Loops



One loop for small delays



Nested loops for large delays

Generating Time Delays

- **Example 5:** Determine the time required to execute the following code. Assume the clock frequency is 800KHz.

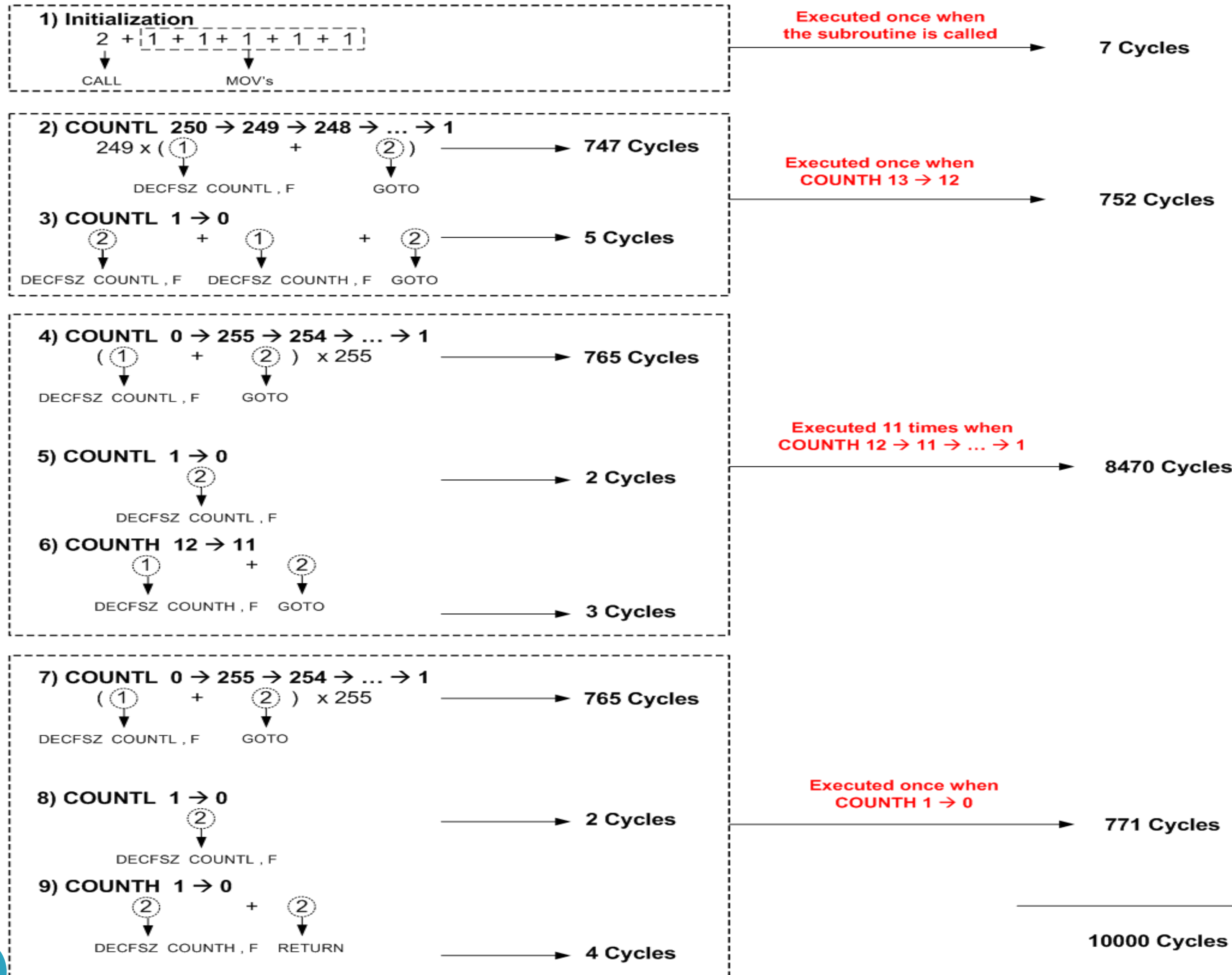
```
        movlw    D'200'  ; initialize counter
        movwf    COUNTER
del     nop              ; main loop for delay
        nop
        decfsz   COUNTER, F
        goto     del
```

- What if this code to be used as a subroutine??!!

Generating Time Delays

- **Example 6:** Analyze the following subroutine and show how it can be used to generate a delay of 10 ms exactly including the call instruction. Assume 4 MHz clock frequency

```
TenMs   nop                         ; beginning of subroutine
        movlw   D'13'
        movwf   COUNTH
        movlw   D'250'
        movwf   COUNTL
Ten1    decfsz  COUNTL , F          ; inner loop
        goto    Ten1
        decfsz  COUNTH , F          ; outer loop
        goto    Ten1
        return
```

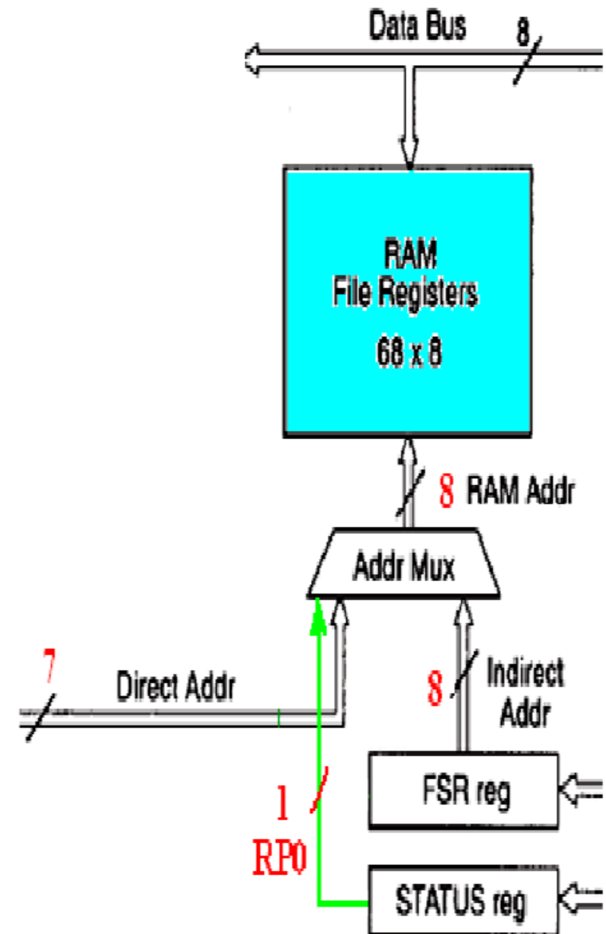


1) Initialization
2 + 1 + 1 + 1 + 1 + 1
CALL
MOV's
Executed once when the subroutine is called
7 Cycles
2) COUNTL 250 → 249 → 248 → ... → 1
249 x (1 + 2)
DECFSZ COUNTL , F
GOTO
747 Cycles
3) COUNTL 1 → 0
2 + 1 + 2
DECFSZ COUNTL , F
DECFSZ COUNTH , F
GOTO
5 Cycles
Executed once when COUNTH 13 → 12
752 Cycles
4) COUNTL 0 → 255 → 254 → ... → 1
(1 + 2) x 255
DECFSZ COUNTL , F
GOTO
765 Cycles
5) COUNTL 1 → 0
2
DECFSZ COUNTL , F
2 Cycles
6) COUNTH 12 → 11
1 + 2
DECFSZ COUNTH , F
GOTO
3 Cycles
Executed 11 times when COUNTH 12 → 11 → ... → 1
8470 Cycles
7) COUNTL 0 → 255 → 254 → ... → 1
(1 + 2) x 255
DECFSZ COUNTL , F
GOTO
765 Cycles
8) COUNTL 1 → 0
2
DECFSZ COUNTL , F
2 Cycles
9) COUNTH 1 → 0
2 + 2
DECFSZ COUNTH , F
RETURN
4 Cycles
Executed once when COUNTH 1 → 0
771 Cycles
10000 Cycles

Working with Data

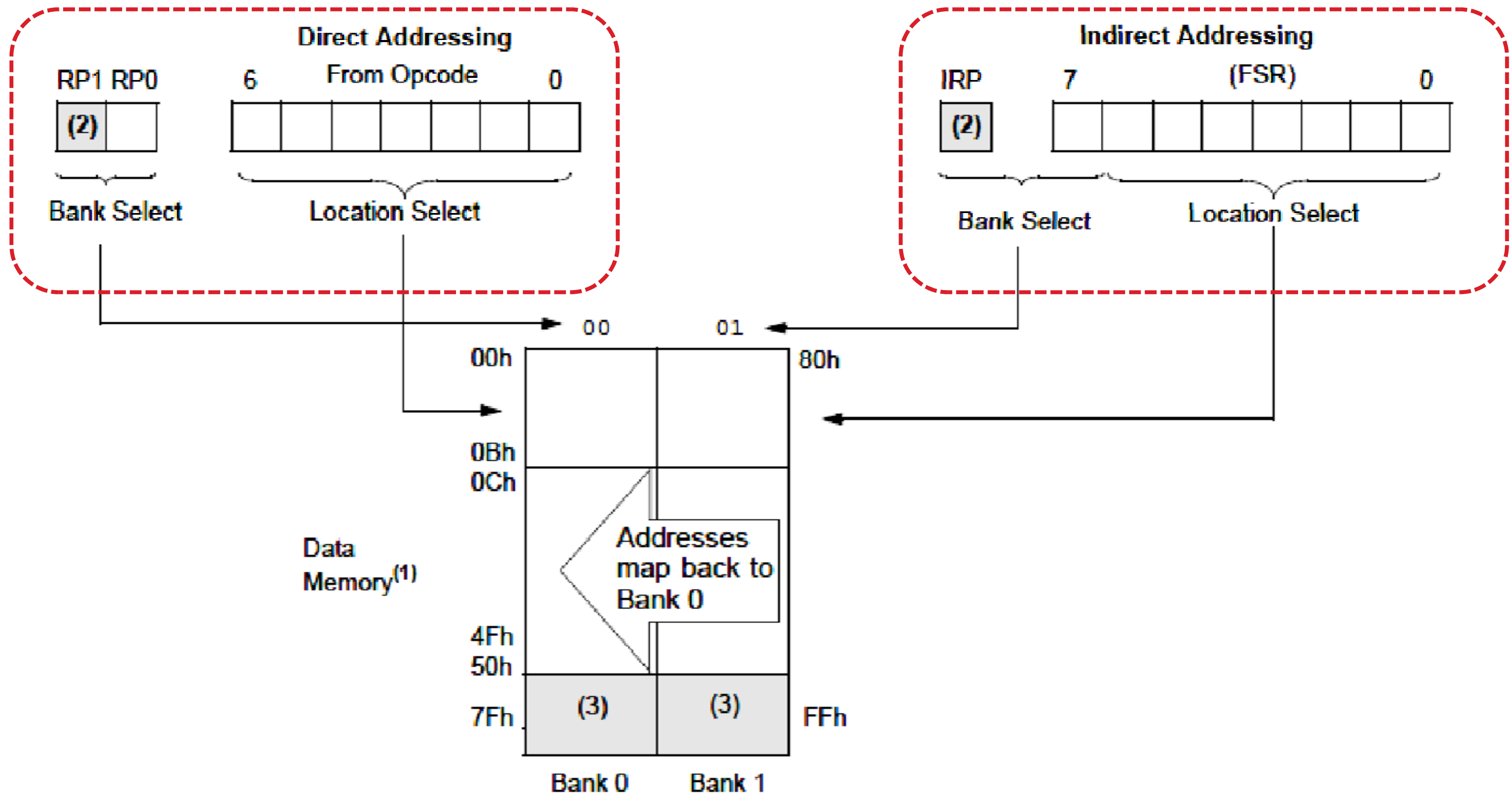
Indirect Addressing

- Direct addressing is capable of accessing single bytes of data
- Working with list of values using direct addressing is inconvenient since the address is part of the instruction
- Instead, we can use indirect addressing where
  - The File Select Register FSR register acts as a pointer to data location.
  - The FSR can be incremented or decremented to change the address
- The value stored in FSR is used to address the memory whenever the INDF (0x00) register is accessed in an instruction
- <u>This forces the CPU to use the FSR register to address memory</u>



Working with Data

Direct/Indirect Addressing in 16F84A



Note 1: For memory map detail, see Figure 2-2.

2: Maintain as clear for upward compatibility with future products.

3: Not implemented.

Working with Data

- **Example 7: A program to add the values found locations 0x10 through 0x1F and stores the result in 0x20**

```
STATUS      equu    0x03            ; define SFRs
FSR         equ     0x04
INDF        equ     0x00
RESULT      equ     0x20
N           equ     D'15'
COUNTER     equ     0x21
            org     0x0000          ; reset vector
            goto    START
            org     0x0005
START       movlw   N               ; initialize counter
            movwf   COUNTER
            movlw   0x11            ; initialize FSR as a pointer
            movwf   FSR
            movf    0x10 , W        ; get 1st number in W
LOOP        addwf   INDF , W        ;  add using indirect addressing
            incf    FSR , F         ;  point to next location
            decfsz  COUNTER , F     ; decrement counter
            goto    LOOP
            movwf   RESULT
DONE        goto    DONE
            end
```

Working with Data

Look-up Tables

- A look-up table is a block of data held in the program memory that the program accesses and uses
- The **movlw** instruction allows us to embed one byte within the instruction and use ! How about a look-up table ?
- In PIC, look-up tables are defined as a subroutine inside which is a group of **retlw** instructions
- The **retlw** instruction is similar to the return instruction; however, it has one operand which is an 8-bit literal that is placed in **W** after the subroutine returns
- In order to choose one of the **retlw** instructions in the look-up table, the program counter is modified to point to the desired instruction by changing the value in the **PCL** register **(0x02)**
- The **PCL** register holds the lower 8 bits of the program counter

Working with Data

- **Example 8: A subroutine to implement a look-up table for the squares of numbers 0 through 5. To compute the square, place the number is stored in W before calling the subroutine SQR_TABLE.**

```
SQR_TABLE    addwf   PCL , 1  ; modify the PCL to point the
                              ; required instruction
             retlw   D'0'     ;  square value of 0
             retlw   D'1'     ;  square value of 1
             retlw   D'4'     ;  square value of 2
             retlw   D'9'     ;  square value of 3
             retlw   D'16'    ;  square value of 4
             retlw   D'25'    ;  square value of 5
```

; Remember that the PC always points to the instruction to be executed

Summary

- Building complex programs requires putting down it requirements and design
- Programs tends to execute instructions sequentially unless branching or subroutines are used
- A subroutine is piece of code that can be called from anywhere inside the program
- A simple way to generate time delays is to use delay loops

Working with Time: Interrupts, Counters, and Timers

Chapter 6

Dr. Iyad Jafar

Outline

- Introduction
- Interrupts
- Timer/Counter
- Watchdog Timer
- Sleep Mode
- Summary

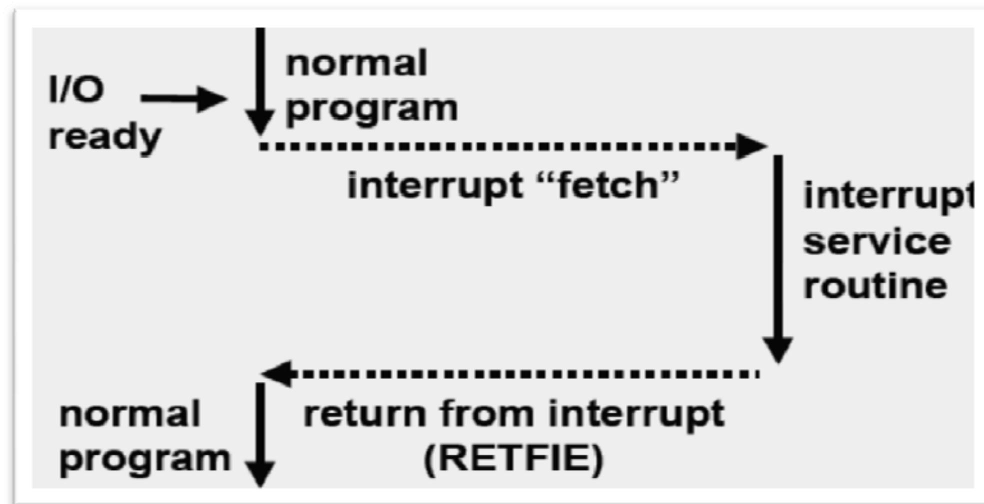
Introduction

- Microcontroller should be able to deal with time
  - Respond in a timely manner to external events
  - Measure time between events
  - Generate time-based activity

- For this purpose, microcontrollers are usually provided with timers and support interrupts

Interrupts

- An interrupt is an event that causes the microcontroller to halt the normal flow of the program and execute another program called the interrupt service routine (ISR)



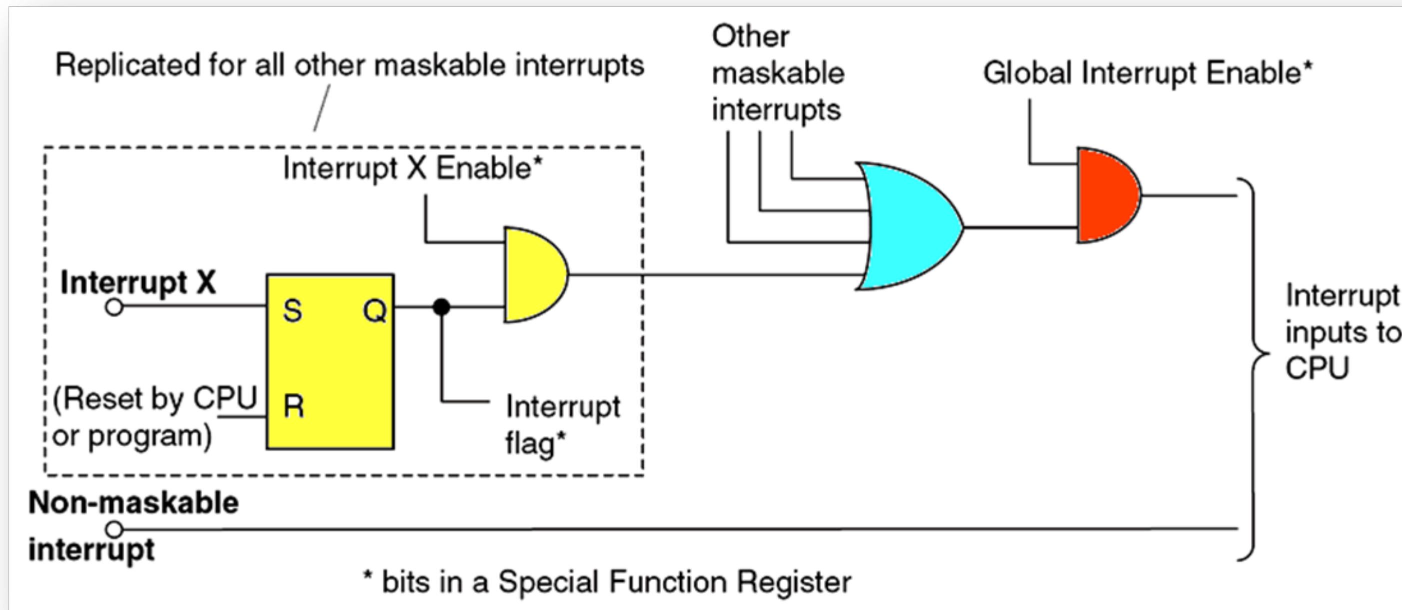
- Interrupts can be thought of as *hardware-initiated subroutine calls*
- Usually, interrupts are generated by I/O devices such as timers or external devices

Interrupts vs. Polling

- **Advantages**
  - Immediate response to I/O service request
  - Normal execution continues until it is known that I/O service is needed
- **Disadvantages**
  - Coding complexity for interrupt service routines
  - Extra hardware needed
  - Processor’s interrupt system I/O device must generate an interrupt request

General Hardware Structure for Interrupts

- Interrupts sources can be *external and internal*
- Two types of interrupts *: maskable* and *non-maskable*
  - Maskable can be enabled/disabled by setting/clearing some bits
  - Non-maskable interrupts can not be disabled and they always interrupt the CPU
- Usually, each interrupt has a flag (a bit) that is set whenever the interrupt occurs

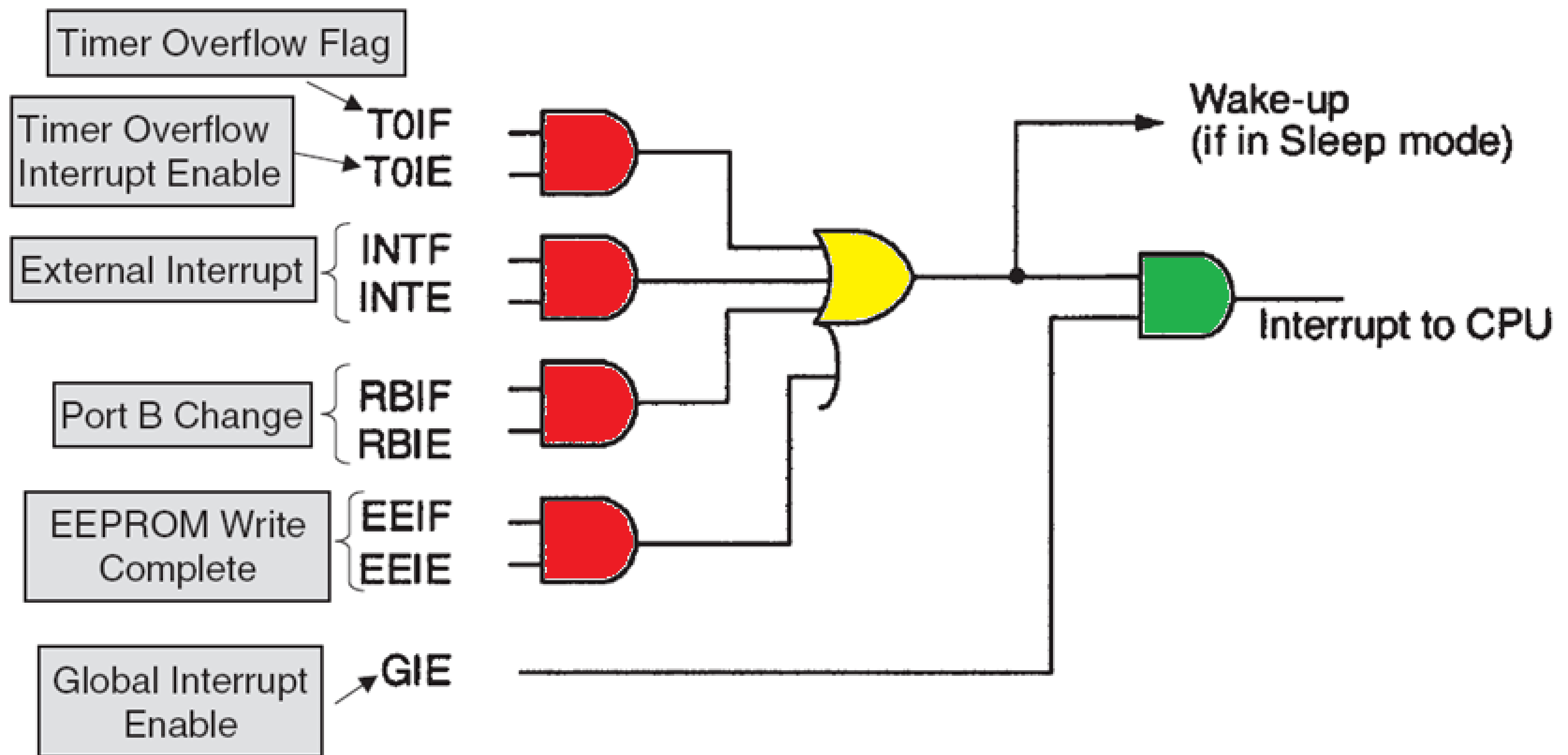


The 16F84A Interrupt Structure

- **Sources of interrupts**
  - ***External interrupt***
    - The only external interrupt input
    - The input is multiplexed with RB0 pin of port B
    - It is edge triggered
  - ***Timer overflow interrupt***
    - It is an internal interrupt that occurs when the 8-bit timer overflows
  - ***Port B on change interrupt***
    - An interrupt occurs when a change is detected on any of the upper 4 bits of port B
  - ***EEPROM write complete interrupt***

The 16F84A Interrupt Structure

- **Interrupt Hardware Structure**



No non-maskable interrupts in 16F84A

The 16F84A Interrupt Structure

- **The INTCON Register**

INTCON REGISTER (ADDRESS 0Bh, 8Bh)

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	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

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

bit 6 **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE Write Complete interrupts
0 = Disables the EE Write Complete interrupt

bit 5 **T0IE**: 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 **T0IF**: 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 (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

The 16F84A Interrupt Structure

- **The Option Register (81H) – interrupt related bit**

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{RBPU}$**: 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

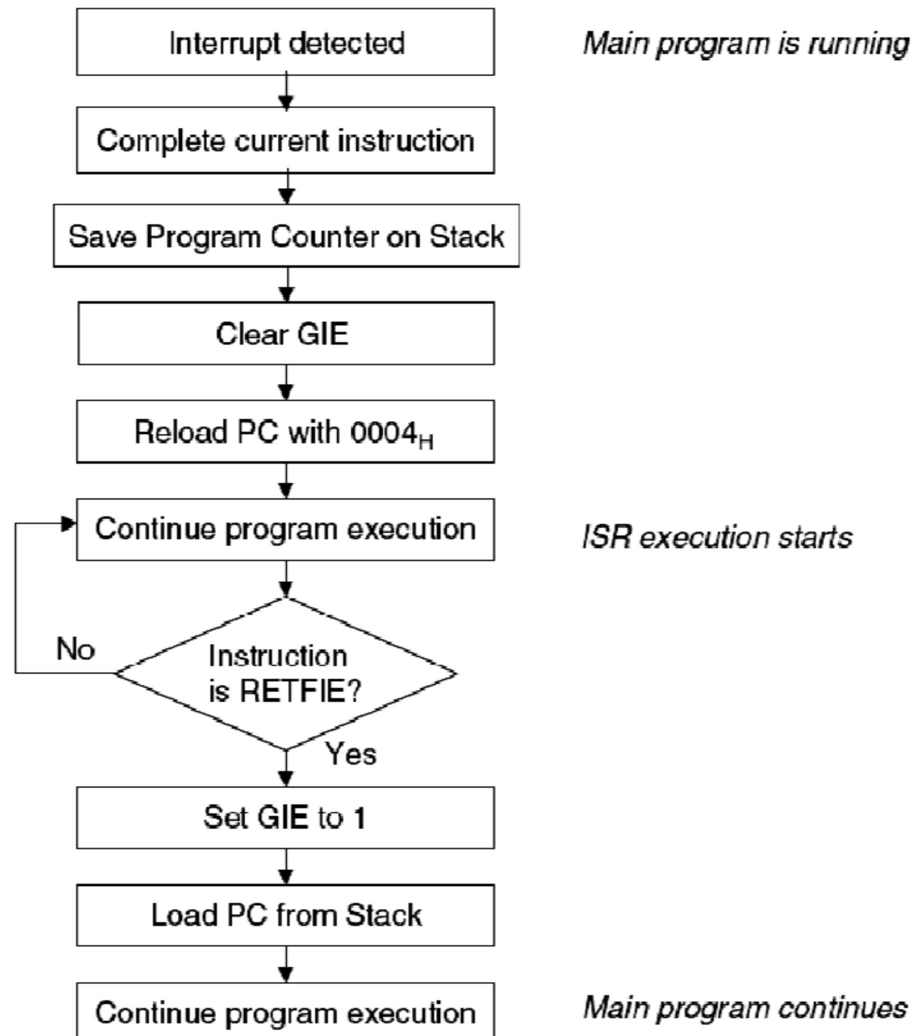
bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

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

Select the transition type on input RB0/INT that will cause an interrupt

The 16F84A Interrupt Structure

- **Interrupt Operation**



The 16F84A Interrupt Structure

- **How to use interrupts ?**

1. Start the interrupt service routine at 0x0004
2. Clear the flag of the used interrupt in the INTCON register (if it is not cleared on reset, e.g. RBIF)
3. Enable the corresponding interrupt by setting its bit in INTCON register
4. Enable global interrupts by setting the GIE bit
5. End the interrupt subroutine with **RETFIE** instruction to resume program execution

The 16F84A Interrupt Structure

- **Example 1**

  Write a PIC16F84 program that continuously adds the content of memory location 0x0A until an external interrupt is observed on RB0. In this case the result is stored in location 0x10 and the working register is cleared. The interrupt should be configured on the arrival of rising edge.

The 16F84A Interrupt Structure

```
        #include  p16F84A.inc          ; include the definition file for 16F84A
        org       0x0000               ; reset vector
        goto      START
        org       0x0004               ; define the ISR
        goto      ISR
        org       0x0006               ; Program starts here
START   bsf       STATUS , RP0         ; select bank 1
        bsf       OPTION_REG , INTEDG  ; select to interrupt on rising edge
        bsf       INTCON , INTE        ; enable external interrupt on RB0/INT
        bsf       INTCON , GIE         ; enable global interrupts
        bcf       STATUS , RP0         ; select bank 0
        molw      0x00                 ; clear W
ADD     addwf     0x0A , W             ; add the contents of 0x0A to W
        goto      ADD                  ; keep adding until an interrupt occurs

        org       0x00BC               ; location of ISR
ISR     movwf     0x10                 ; on interrupt store the accumulated result
        clrw                           ; clear working register
        bcf       INTCON , INTF        ; clear the interrupt flag
        retfie                         ; return from the ISR
        end
```

Context Saving

- **What if the main program is to preserve the W register and the interrupt service routine uses it?**
  - Save it temporarily in memory at the beginning of the ISR

    ```
    MOVWF    TEMP  ; push
    ```

  - Restore the value at the end of ISR

    ```
    MOVF     TEMP, W   ; pop
    ```

- **What if we want to preserve some memory location such as the STATUS register on interrupt?**
  - Save it temporarily in memory at the beginning of the ISR

    ```
    SWAPF    STATUS,0        ; push
    MOVWF    TEMP
    ```

  - Restore the value at the end of ISR

    ```
    SWAP     TEMP, 0         ;  pop
    MOVWF    STATUS
    ```

The 16F84A Interrupt Structure

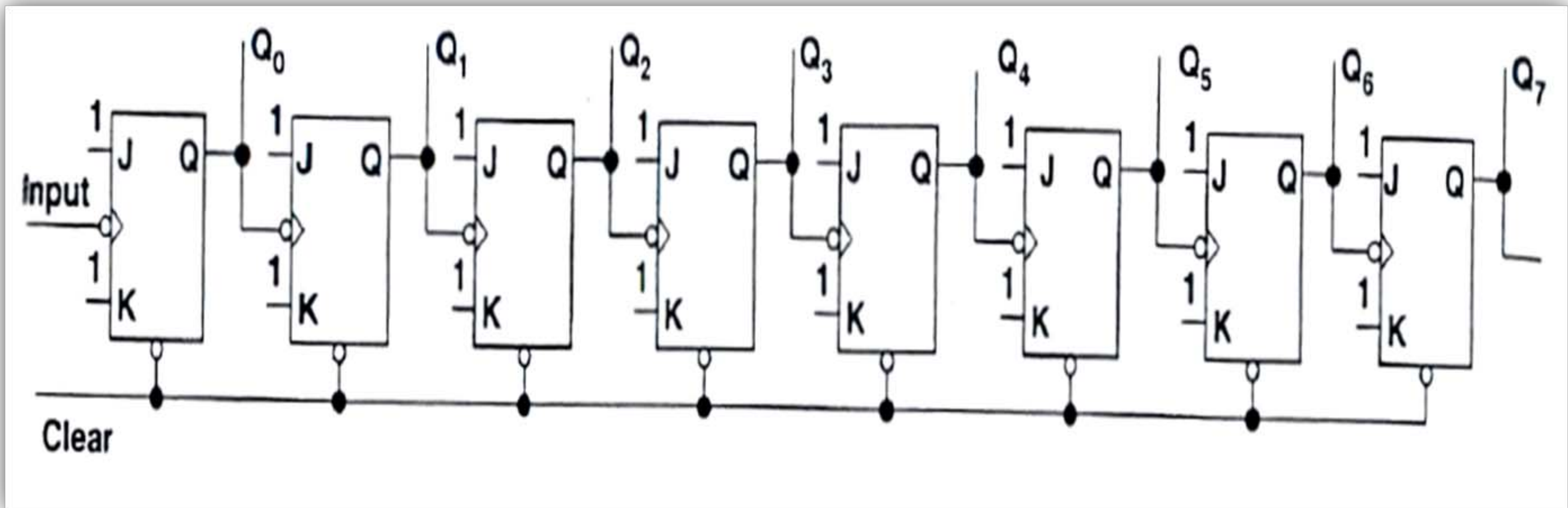
- **Multiple Interrupts**
  - Note that there is only one interrupt vector for all types of interrupts
  - In other words, regardless of the interrupt type, the microcontroller will start executing from location 0x0004 on any interrupt
  - How to determine the source of interrupt ?
  - Check the interrupt flag bits in the INTCON register at the beginning of the interrupt service routine to determine what is the source of the interrupt !

```
Interrupt_SR    btfsc intcon,0    ;test RBIF
                goto  portb_int   ;Port B Change routine
                btfsc intcon,1    ;test INTF
                goto  ext_int     ;external interrupt routine
                btfsc intcon,2    ;test T0IF
                goto  timer_int   ;timer overflow routine
                btfsc eecon1,4    ;test EEPROM write complete flag
                goto  eeprom_int  ;EEPROM write complete routine
```

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	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

Counters and Timers

- Digital counters can be built with flip-flops. They can count up or down, reset, or loaded with initial value
- When the most significant bit changes from 1 to 0 , this indicates an overflow. This signal can be used to interrupt the microcontroller
- *If the counter operates using a clock with known frequency we can use it as a timer*



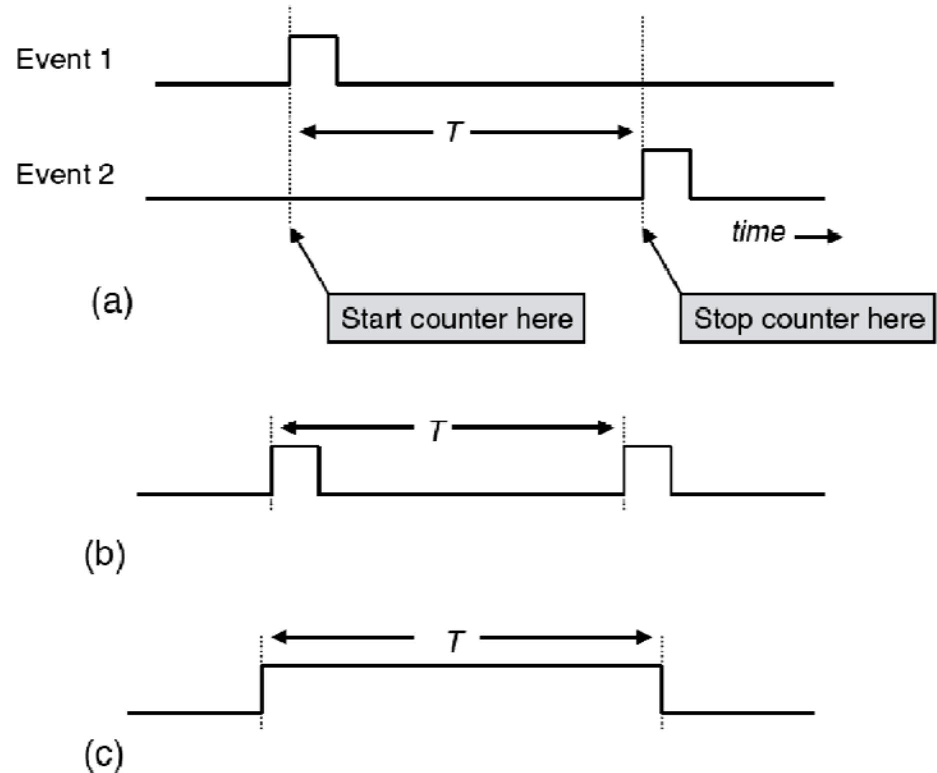
Counters and Timers

- **Timer applications**

(a) Measure the time between two events

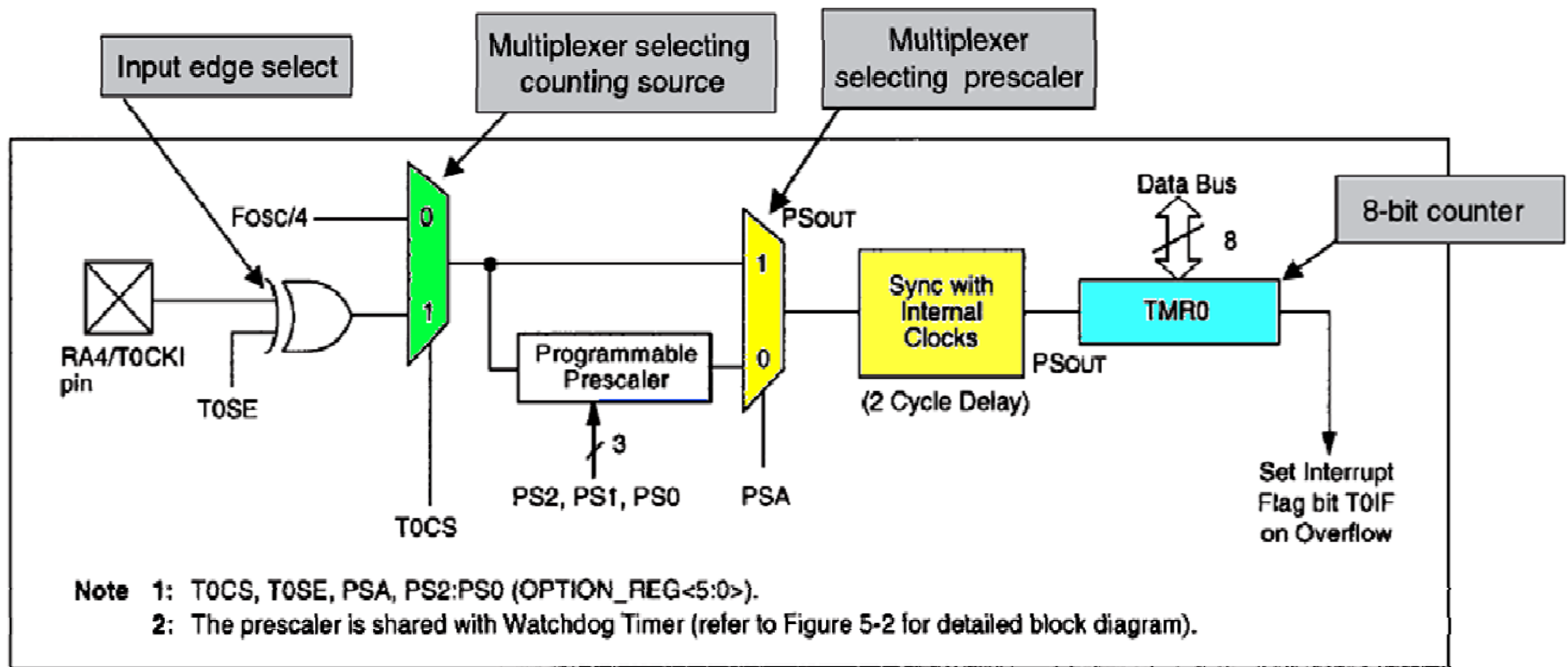
(b) Measure the time between two pulses

(c) Measure a pulse duration



Use **polling** or **interrupts**

The 16F84A Timer 0 Module



- 8-bit counter , memory address 0x01
- Configurable counter using the OPTION register (0x81)
- Two sources for the timer clock : instruction cycle clock (Fosc/4) or RA4/T0CKI
- The programmable prescaler is shared with the Watchdog Timer WDT
- The value of frequency division is determined by PS2, PS1, and PS0 bits in the OPTION register

The 16F84A Timer 0 Module

- **The Option Register – Timer related bits**

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 **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

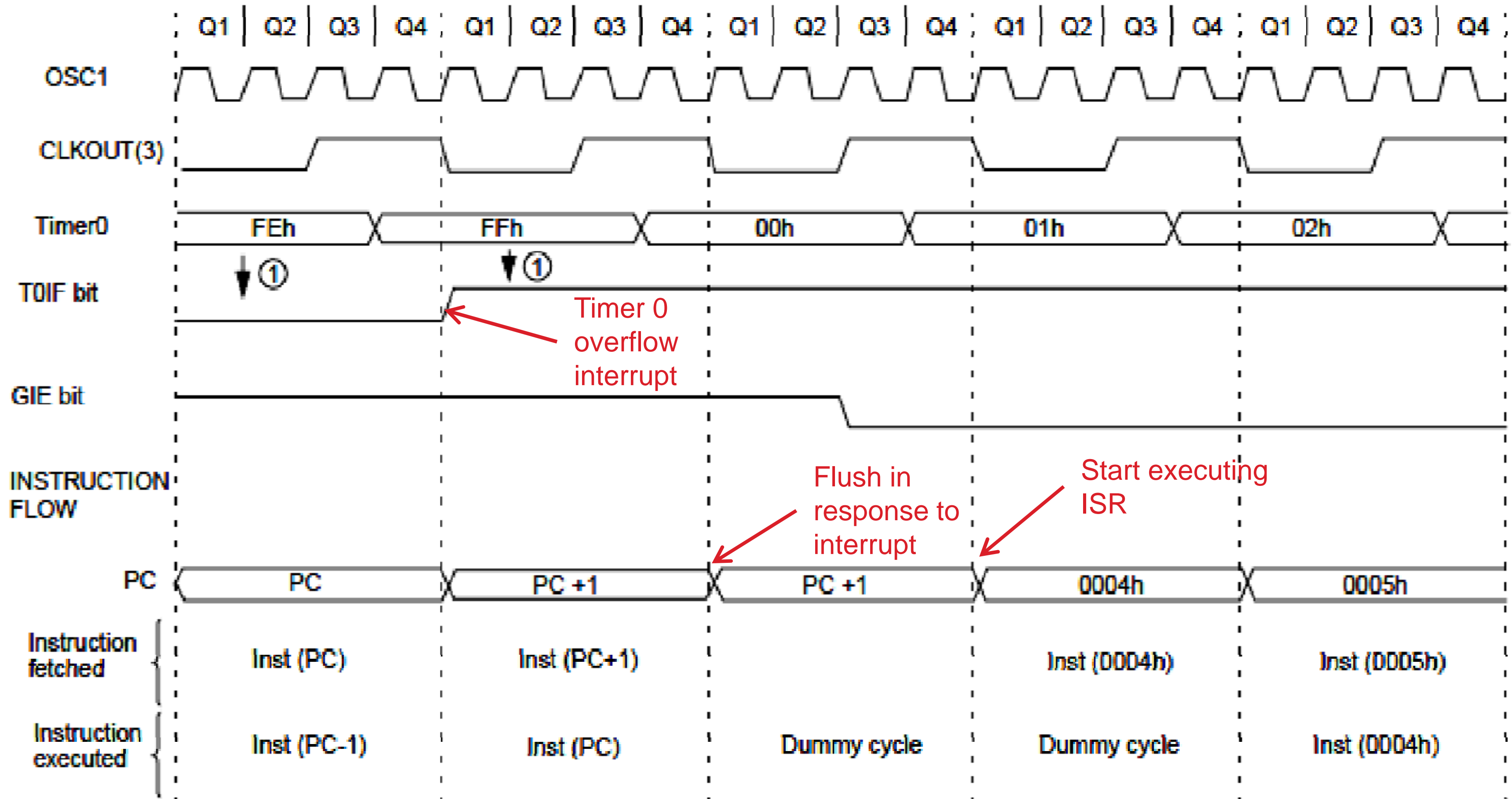
bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

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

The 16F84A Timer 0 Module

- **Timer Timing**



The 16F84A Timer 0 Module

- **Example 2: Write a program that generates a 5 ms delay using the TMR0 module without using interrupts. Assume the clock frequency is 800 KHz.**
  - Fosc = 800 KHz → the timer internal clock = Fosc/4 = 200 KHz → instruction cycle = 5 us → timer increment every 5 us
  - For these settings, the timer generates an interrupt after 256 * 5 us = 1280 us only ?!
  - How about changing the prescale factor ?
    - 256 x prescale x 5 us = 5 ms → prescale = 3.9 ~= 4
    - This will generate a delay of 4 x 256 x 5 us = 5.12 ms
  - What if we need more accurate delay !! We can play around with the count value (we don't have to start from 0 always)
    - N x prescale x 5 us = 5 ms --> N x prescale = 1000 → we can select the prescale 8 and the count N to be 125
    - We have to load TMR0 with 256 – 125 = 131 as initial value

The 16F84A Timer 0 Module

- **Example – cont'd**

```
        #include p16f84A.inc
        org     0x0000
        goto    start
        org     0x0010
start   . . . .
        call    delay5
        . . . .
delay5  movlw   D'131' ;      preload T0, it overflows after 125 counts
        movwf   TMR0
        bsf     STATUS, RP0   ;select memory bank 1
        movlw   B'00000010'   ;set up T0 for internal input, prescale by 8
        movwf   OPTION_REG
        bcf     STATUS, RP0   ;select bank 0
del1    btfss   intcon,T0IF   ;test for Timer Overflow flag
        goto    del1          ;loop if not set (no timer overflow)
        bcf     intcon,T0IF   ;clear Timer Overflow flag
        return
```

Watchdog Timer

- Special timer internal to the microcontroller that is continually counting up.
- If enabled and it overflows, the microcontroller is reset
- Can be used to reset the Microcontroller if a program fails or gets stuck
- Properties
  - The WDT timer is enabled/disabled by the *WDTE* bit in the configuration word
  - It has its own internal *RC oscillator*
  - The nominal time-out period is *18 ms*
  - It can be extended through the prescaler bits in the *OPTION* register (up to 128x18 ms= 2.3 sec)
  - The WDT timer can be cleared by software using the *CLRWDT* instruction
- **How does the watchdog timer know if the program is stuck ???!!! It does not!**

Sleep Mode

- An important way to save power!
- The microcontroller can be put in sleep mode by using the ***SLEEP*** instruction
- **Once in sleep mode, the microcontroller operation is almost suspended**
  - The oscillator is switched off
  - The WDT is cleared. If the WDT is enabled, it continues running
  - Program execution is suspended
  - All ports retain their current settings
  - $\overline{PD}$ and $\overline{TO}$ bits are cleared and set respectively
  - Power consumption falls to a negligible amount
- **To exit the sleep mode**
  - Interrupt occurs (even if GIE = 0)
  - WDT wake-up

  } Program continues execution from PC+1

  - External reset the MCLR pin

  } MC is reset !

Summary

- Microcontrollers can deal with time by using timers and interrupts
- Interrupts saves the microcontrollers computational power as they require its attention when they occur only
- Most interrupts are configurable
- Hardware timer can be used as a counter or a timer and it is very useful in measuring time

Parallel Ports, Power Supply, and the Clock Oscillator

Chapter 3

Dr. Iyad Jafar

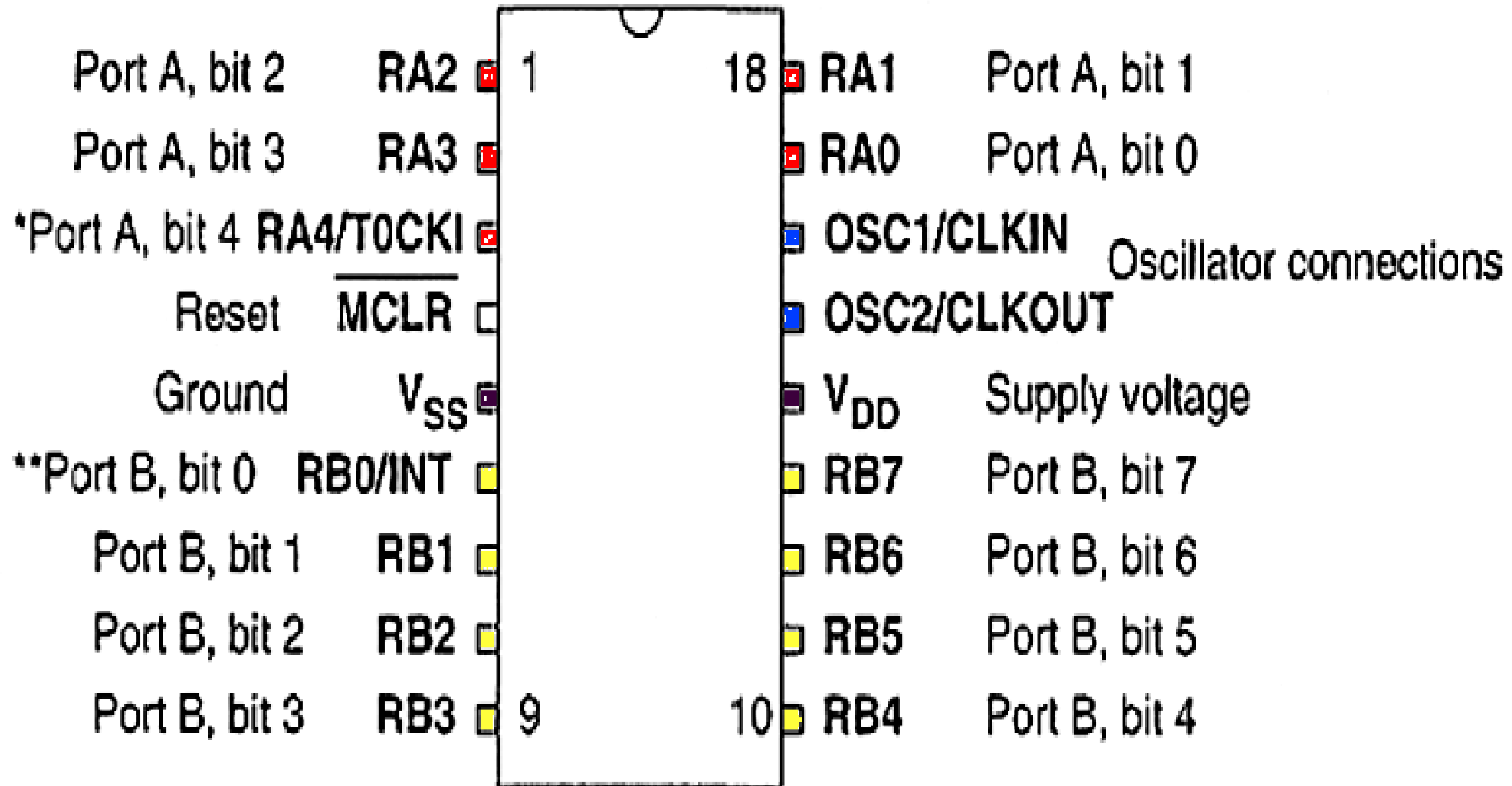
Outline

- Why Do We Need Parallel Ports?
- Hardware Realization of Parallel Ports
- Interfacing to Parallel Ports
- The PIC 16F84A Parallel Ports
- The Power Supply
- The Clock Oscillator

Why Do We Need Parallel Ports?

- Almost any microcontroller needs to transfer digital data from/to external devices and for different purposes
  - Direct user interface – switches, LEDs, keypads, displays
  - Input measurement - from sensors, possibly through ADC
  - Output control information – control motors and actuators
  - Bulk data transfer – to other systems/subsystems
- Transfer could be serial or parallel ! Analog or digital !

The PIC 16F84 Parallel Ports



The PIC 16F84 Parallel Ports

PORT A

- *5-bit general-purpose* bidirectional digital port

- **Related registers**
  - Data from/to this port is stored in *PORTA* register *(0x05)*
  - Pins can be configured for input or output by setting or clearing corresponding bits in the *TRISA* register *(0x85)*

- Pin *RA4* is multiplexed and can be used as the clock for the TIMER0 module

The PIC 16F84 Parallel Ports

PORT B

- *8-bit general-purpose* bidirectional digital port
- **Related registers**
  - Data from/to this port is stored in *PORTB* register *(0x06)*
  - Pins can be configured for input or output by setting or clearing, corresponding bits in the *TRISB* register *(0x86),* respectively
- **Other features**
  - Pin *RB0* is multiplexed with the external interrupt INT and has Schmitt trigger interface
  - Pins *RB4 – RB7* have a useful 'interrupt on change' facility

The PIC 16F84 Parallel Ports

- **Example 1 –** *configuring port B such that pins 0 to 2 are inputs, pins 3 to 4 outputs, and pins 5 to 7 are inputs*

```
bsf      STATUS , RP0     ; select bank1
movlw    0xE7
movwf    TRISB            ; PORTB<7:5> input,
                          ; PORTB<4:3> output
                          ; PORTB<2:0> input
```

The PIC 16F84 Parallel Ports

- **Example 2 –** *configuring PORTB as output and output value 0xAA*

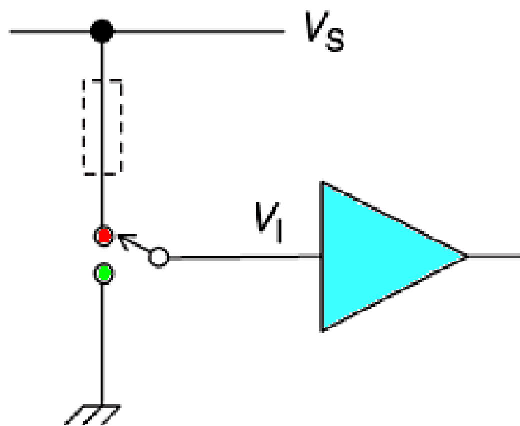
```
bsf      STATUS , RP0    ; select bank1
clrf     TRISB           ; PORTB is output
movlw    0xAA
 bcf     STATUS , RP0    ; select bank0
movwf    PORTB           ; output data
```

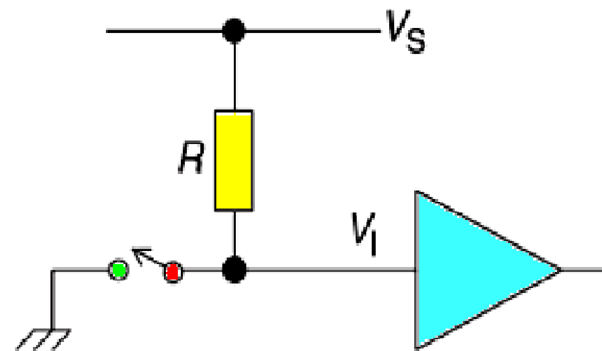
- **Example 3 –** *configuring PORTA as input, read it and store the value in 0x0D*

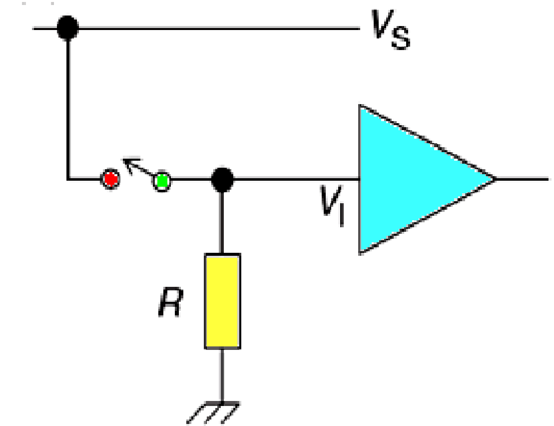
```
bsf      STATUS , RP0    ; select bank1
movlw    0xFF
movwf    TRISA           ; PORTA is input
 bcf     STATUS , RP0    ; select bank0
movf     PORTA, W        ; read data
movwf    0x0D             ; save data
```

Interfacing to Parallel Ports

Switches







Interfacing to SPDT switch. A current limiting resistor might be needed

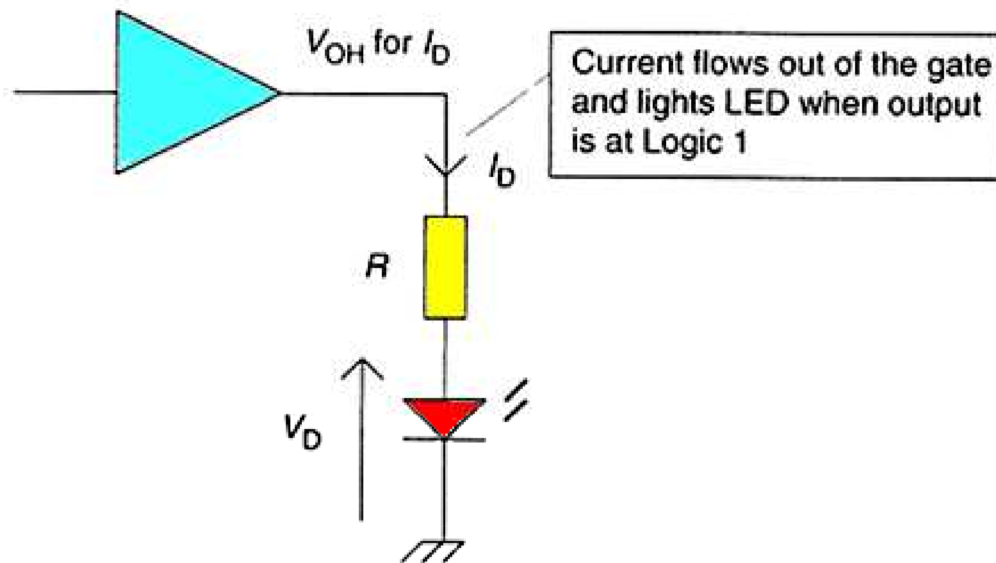
Interfacing to SPST switch. To reduce wasted current, the pull-up resistor R should be high (10-100KOhms)

Interfacing to SPST switch using a pull-down resistor

Interfacing to Parallel Ports

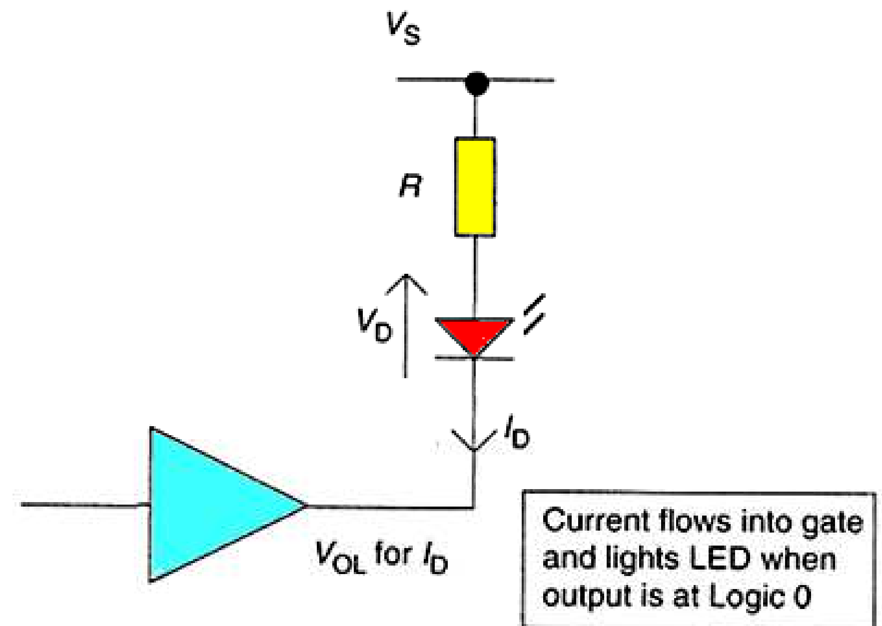
Light Emitting Diodes (LEDs)

- LEDs can be driven from a logic output as long as the current requirements are met. Interfacing of LEDs depending on the logic type and their capability to source and sink current



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

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



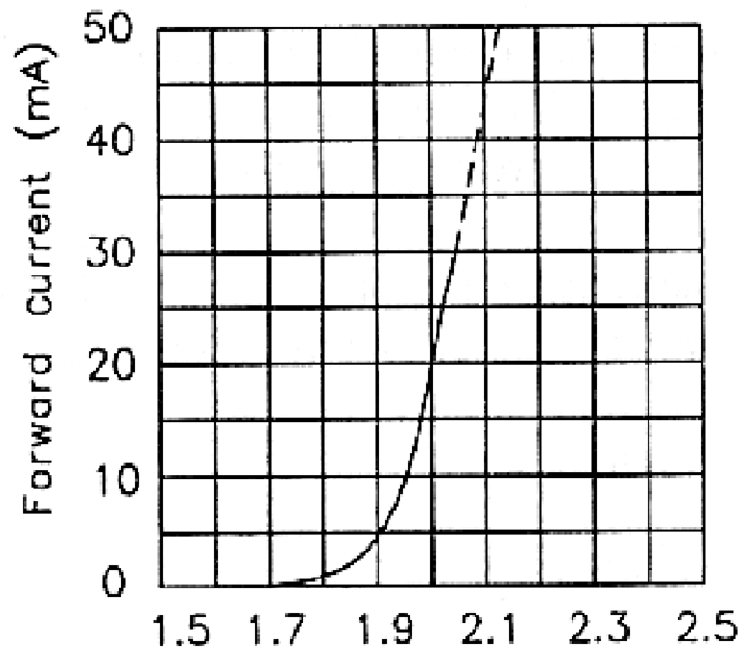
For current sink: $V_S = V_{OL} + RI_D + V_D$

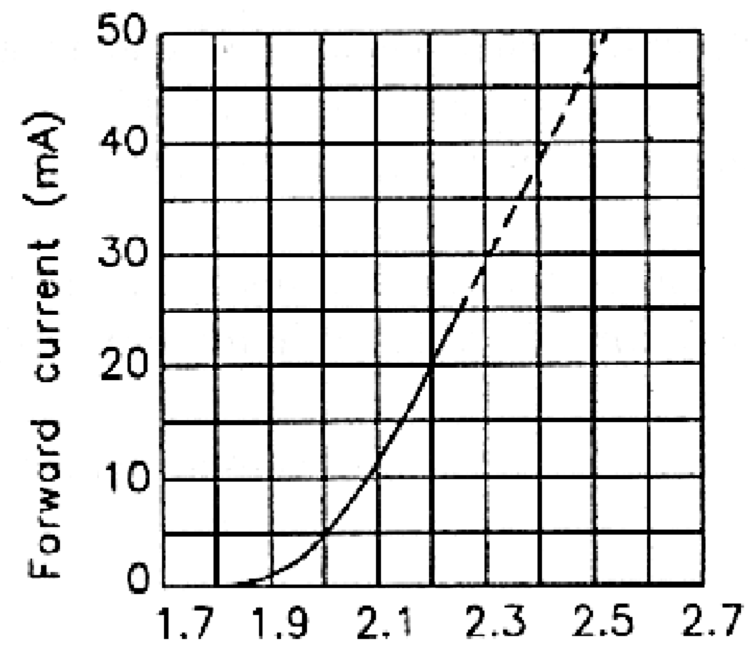
$$R = \frac{V_S - V_D - V_{OL}}{I_D}$$

Interfacing to Parallel Ports

Light Emitting Diodes (LEDs)

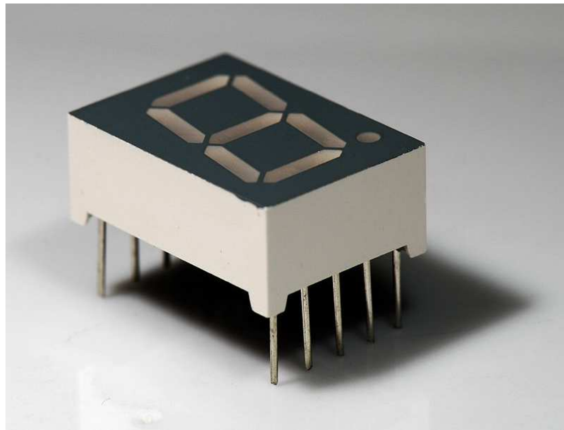
- A special type of diodes made of semiconductor material that can emit light when forward biased

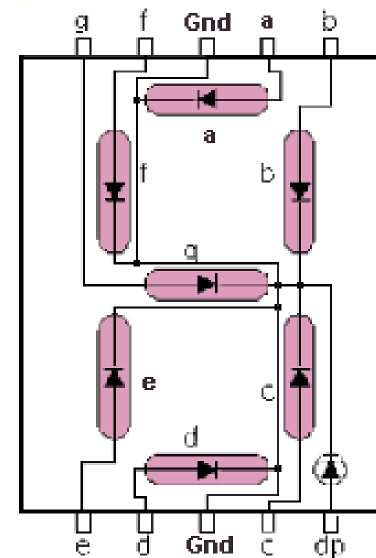


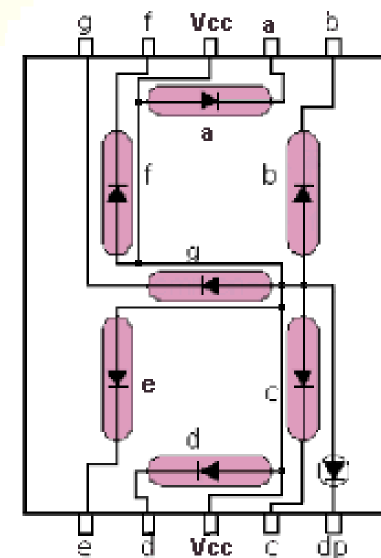


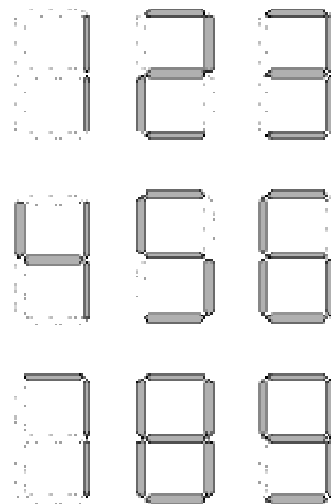
Interfacing to Parallel Ports

7-Segment Display







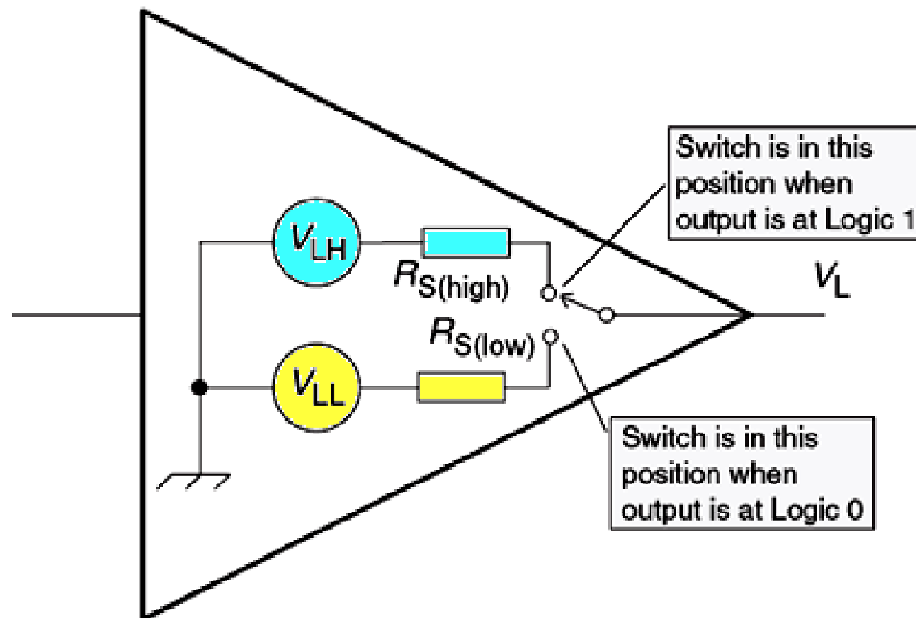


Digit Shown	Illuminated Segment (1 = illumination)						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	0	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

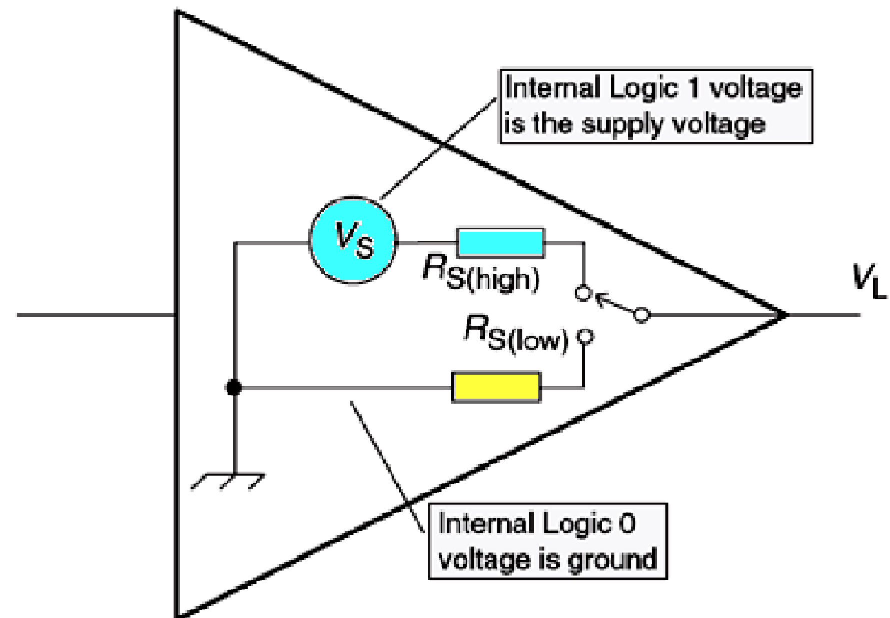
Interfacing to Parallel Ports

Port Electrical Characteristics

- Logic gates are designed to interface easily with each other, especially when connecting gates from the same family
- The concern arises when connecting logic gates to non-logic devices such as switches and LEDs



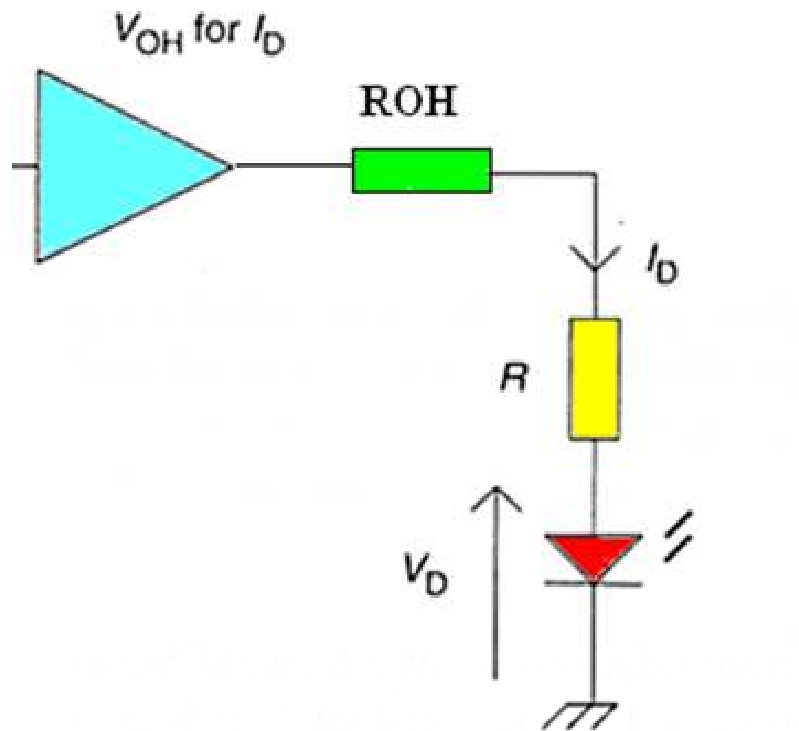
Generalized model

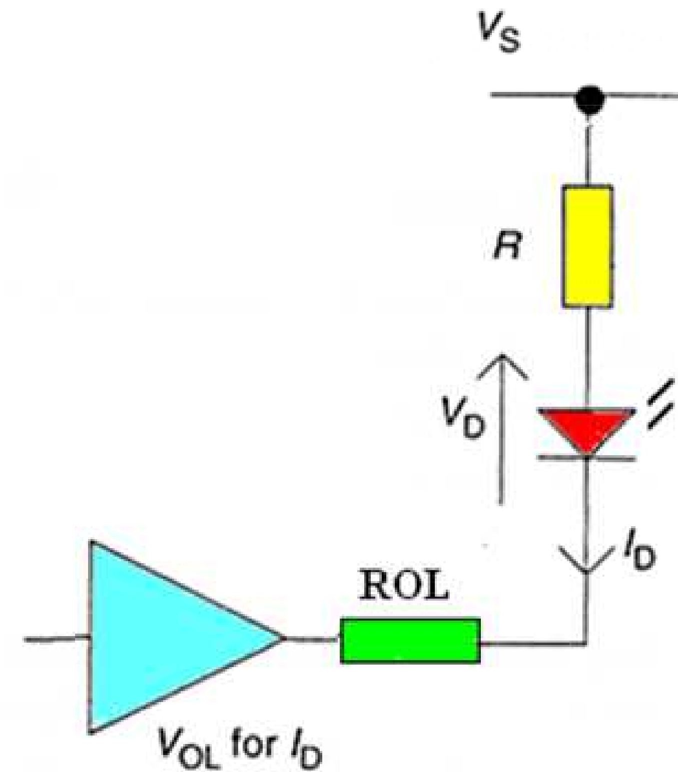


CMOS model

Interfacing to Parallel Ports

Light Emitting Diodes (LED)



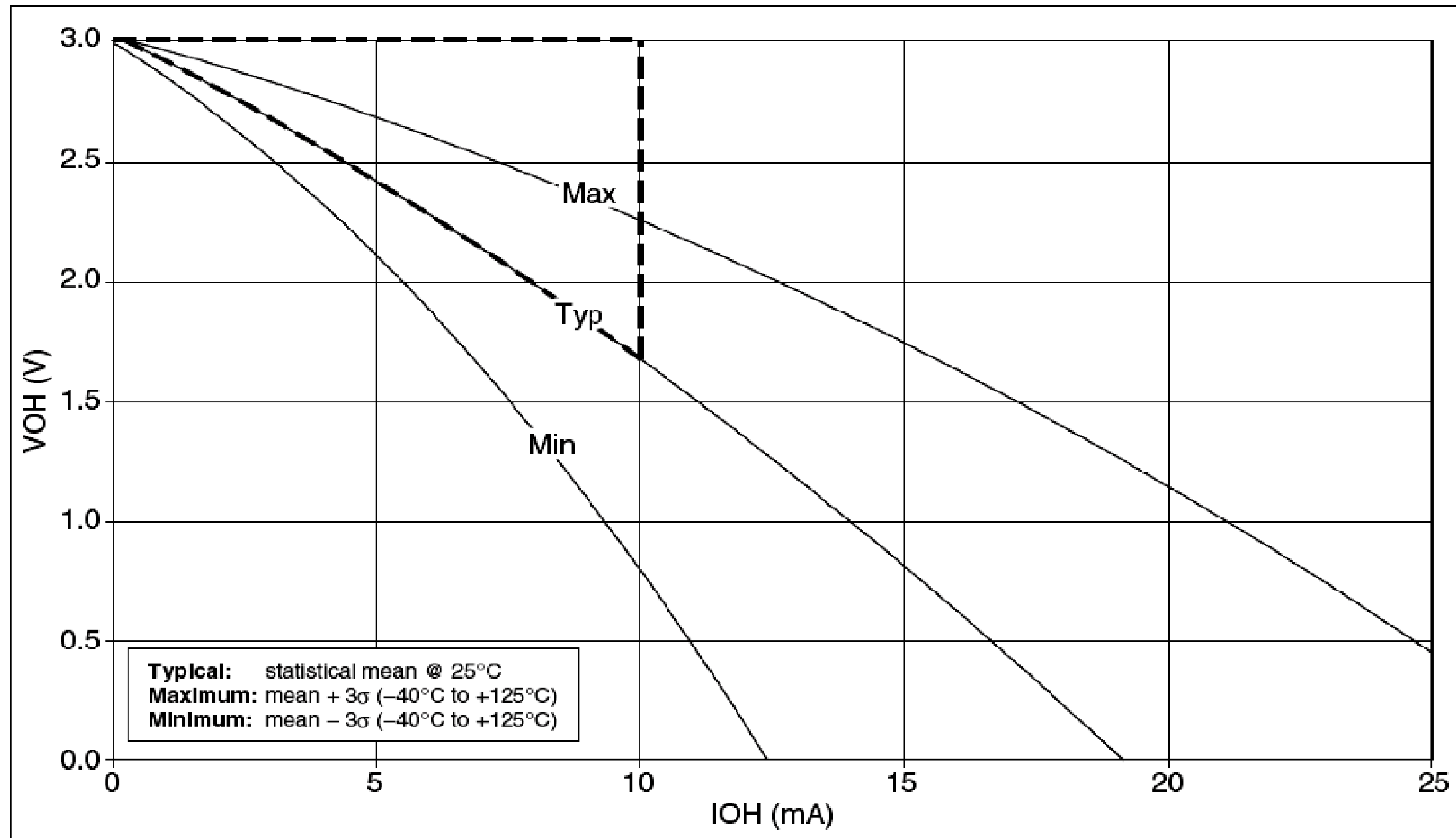


Computation of limiting resistors when internal resistance of the port pin is considered

The PIC 16F84 Parallel Ports

Port Output Characteristics

V_{OH} vs. I_{OH} (VDD = 3V, –40 to 125◦C)

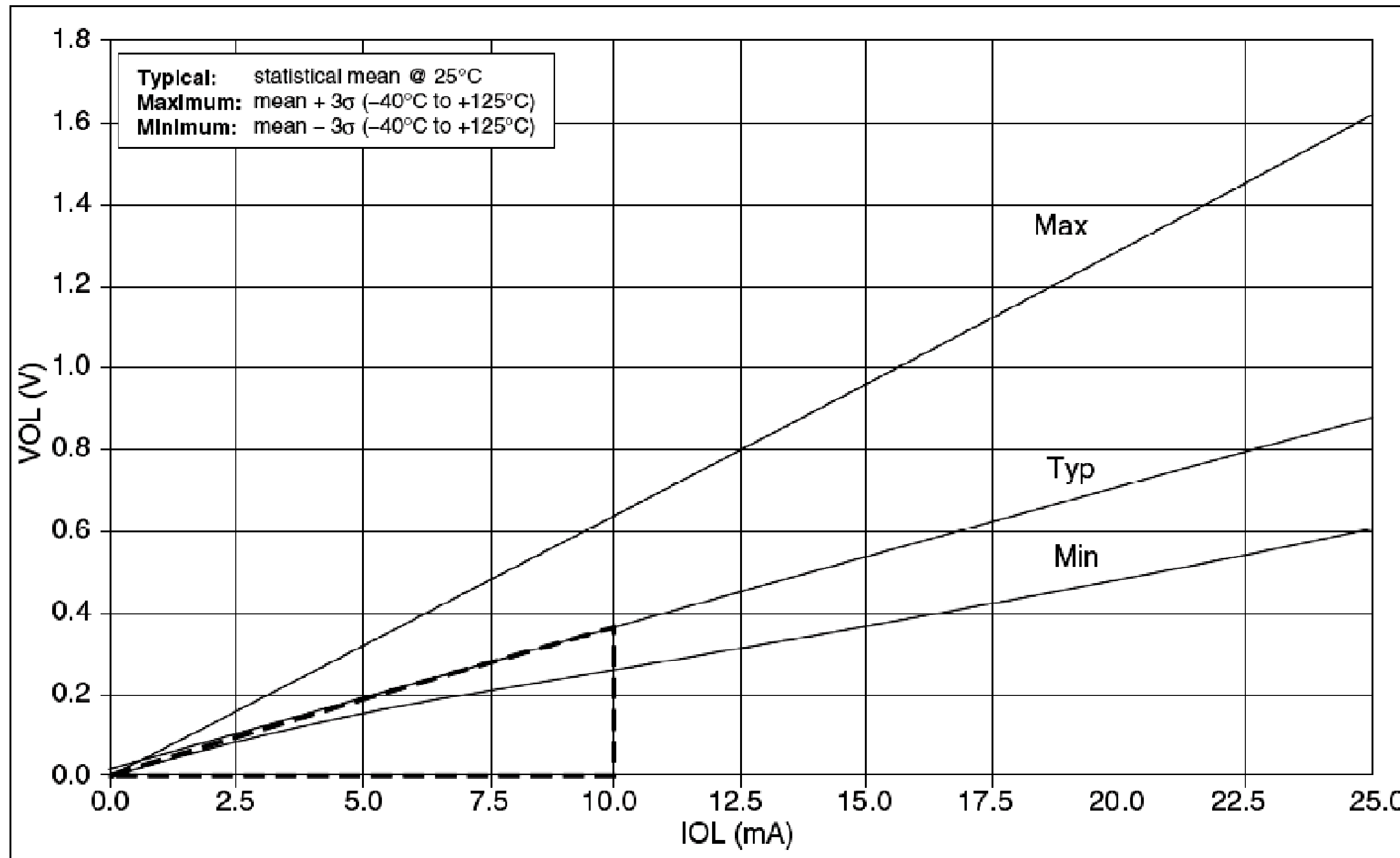


R_{OH} = 130 Ω

The PIC 16F84 Parallel Ports

Port Output Characteristics

V_{OL} vs. I_{OL}(VDD = 3V, –40 to 125◦C)



R_{OL} = 36 Ω

Example 3.1

- **Example –** Write a program that continuously reads an input value from 4 switches connected to PORTA (RA3-RA0) and display the value on 4 LEDs connected to PORTB (RB7-RB4). Make sure to draw the circuit and configure the ports properly.

- ***Requirements:***

1) *Connect four switches to RA3-RA0. Configure these pins as input.*
2) *Connect four LEDs to RB7-RB4. Configure these pins as outputs.*

Example 3.1

```
            #include   "P16F84A.INC"
TEMP        EQU        0X20
            ORG        0X0000
            ; --------------------- MAIN PROGRAM ------------------------------------
MAIN        BSF        STATUS,RP0            ; SELECT BANK 1
            MOVLW      B'00001111'
            MOVWF      TRISA                 ; CONFIGURE RA3-RA0 AS INPUT
            MOVLW      B'00000000'
            MOVWF      TRISB                 ; CONFIGURE RB7-RB4 AS OUTPUT
            BCF        STATUS, RP0

REPEAT      MOVF       PORTA, W              ; READ FROM PORT A
            ANDLW      0X0F                  ; MASK THE LOWER 4 BITS IN PORTA
            MOVWF      TEMP
            SWAPF      TEMP, F               ; MOVE BITS TO RB7-RB4
            MOVWF      PORTB

            GOTO       REPEAT
            END
```

Example 3.2

- **Example –** Modify the program and the circuit in Example 3.1 such that the switches are read and displayed when an external interrupt occurs (falling edge) only.

- ***Requirements:***

1) *Connect four switches to RA3-RA0. Configure these pins as input.*
2) *Connect four LEDs to RB7-RB4. Configure these pins as outputs.*
3) *Connect a switch to RB0 and configure it as input*

Example 3.2

```
            #include    "P16F84A.INC"
TEMP        EQU         0X20
            ORG         0X0000
            GOTO        MAIN
            ORG         0X0004
            GOTO        ISR
            ; -------------------- MAIN PROGRAM ----------------------
MAIN        BSF         STATUS,RP0                ; SELECT BANK 1
            MOVLW       B'00001111'
            MOVWF       TRISA                     ; CONFIGURE RA3-RA0 AS INPUT
            MOVLW       B'00000001'               ; CONFIGURE RB0 AS INPUT
            MOVWF       TRISB                     ; CONFIGURE RB7-RB4 AS OUTPUT
            BCF         OPTION_REG, INTEDG        ; INTERRUPT ON FALLING EDGE
            BCF         STATUS, RP0
            BSF         INTCON, INTE              ; ENABLE INTERRUPT
            BSF         INTCON, GIE
WAIT        GOTO        WAIT                      ; WAIT FOR INTERRUPT
            ; --------------------- ISR ----------------------------------
ISR         MOVF        PORTA, W                  ; READ FROM PORT A
            ANDLW       0X0F                      ; MASK THE LOWER 4 BITS IN PORTA
            MOVWF       TEMP
            SWAPF       TEMP, F                   ; MOVE BITS TO RB7-RB4
            MOVWF       PORTB
            BCF         INTCON, INTF
            RETFIE
            END
```

Example 3.3

- **Example –** Write a program to control the flashing of a LED that is connected to RB1 using a pushbutton that is connected to RB0. The LED starts flashing upon the arrival of the first rising edge on RB0. Afterwards, successive edges toggle the state of flashing (On, off, on, …). When the LED is flashing, this implies that it is 0.5 second ON and 0.5 second OFF. *Assume 4MHz clock.*
- ***Requirements:***

1) *Configure RB0 as input and RB1 as output*
2) *Enable external interrupt (INTE) and global interrupts (GIE)*
3) *Write a 0.5 second delay routine*
4) *Keep track of the current status of flashing (on/off)*

Example 3.3

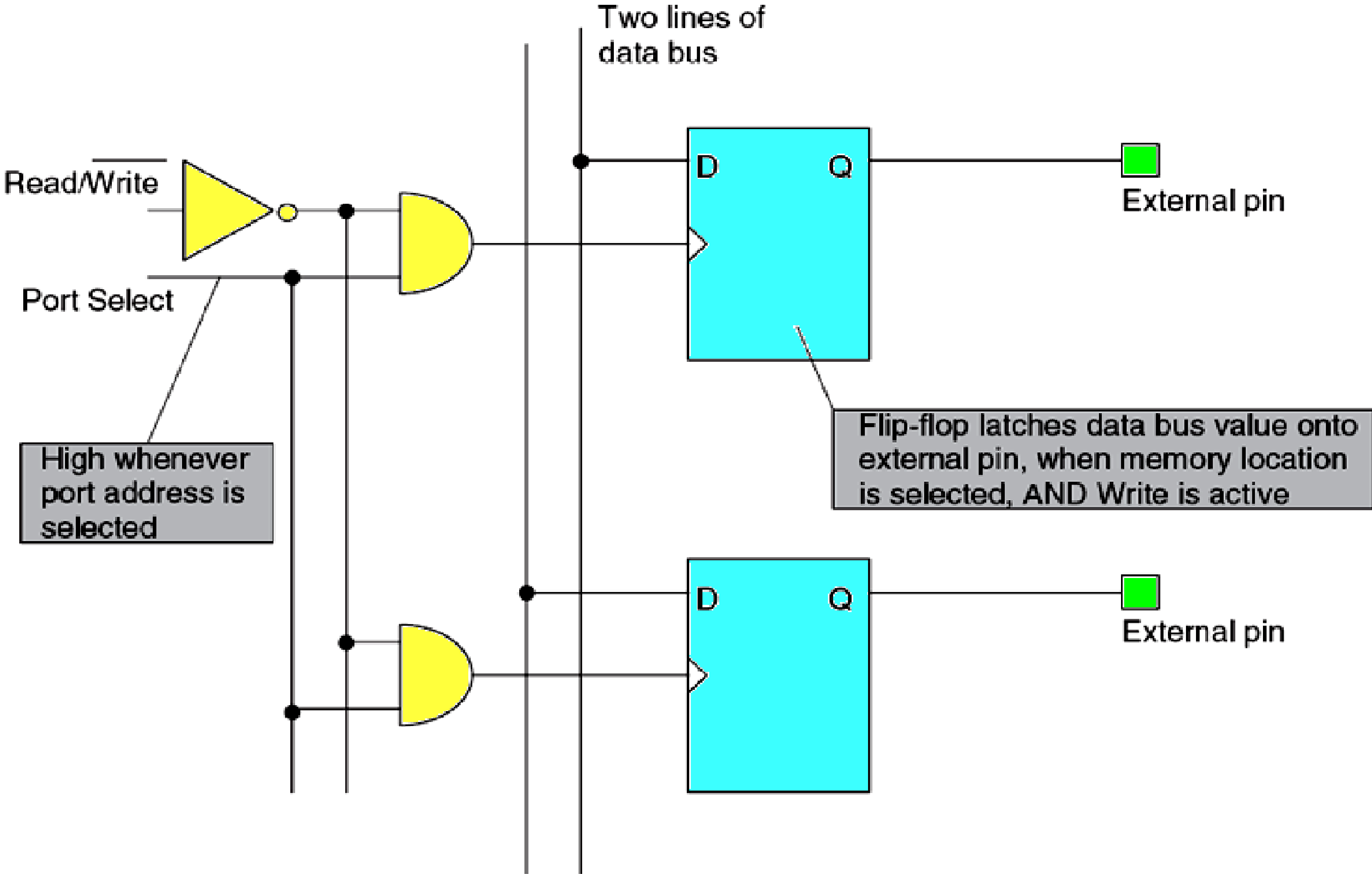
```
        #include  "P16F84A.INC"
FLASH   EQU       0X20              ; STORE THE STATE OF FLASHING
COUNT1  EQU       0X21              ; COUNTER FOR DELAY LOOP
COUNT2  EQU       0X22              ; COUNTER FOR DELAY LOOP
        ORG       0X0000
        GOTO      START
        ORG       0X0004
        GOTO      ISR
; --------------------------------- MAIN PROGRAM ---------------------------------------------
START   CLRF      FLASH             ; CLEAR FLASHING STATUS
        BSF       STATUS,RP0        ; SELECT BANK 1
        MOVLW     B'00000001'       ; CONFIGURE RB0 AS INPUT AND RB1 AS OUPUT
        MOVWF     TRISB
        BSF       OPTION_REG, INTEDG ; SELECT RISING EDGE FOR EXTERNAL INTERRUPT
        BSF       INTCON , INTE     ; ENABLE EXTERNAL INTERRUPT
        BSF       INTCON , GIE      ; ENABLE GLOBAL INTERRUPT
        BCF       STATUS,RP0        ; SELECT BANK 0
        CLRF      PORTB             ; CLEAR PORTB; TURN OFF LED
WAIT    BTFSS     FLASH , 0         ; IF BIT 0 OF FLASH IS CLEAR THEN NO FLASHING
        GOTO      WAIT              ; WAIT UNTIL BIT 0 IS SET
        MOVLW     B'00000010'
        XORWF     PORTB , 1         ; COMPLEMENT RB1 TO FLASH
        CALL      DEL_p5sec
        GOTO      WAIT
```

Example 3.3

```
---------------- INTERRUPT SERVICE ROUTINE ----------------
ISR         MOVLW    0x01
            XORWF    FLASH , F          ; COMPLEMENT THE STATUS
            BCF      INTCON , INTF      ; CLEAR THE INTF FLAG
            RETFIE

; ---------------- DELAY ROUTINE ------------------------
DEL_p5sec
            MOVLW    D'0'
            MOVWF    COUNT1
            MOVLW    D'244'
            MOVWF    COUNT2
LOOP        NOP
            NOP
            NOP
            NOP
            NOP
            DECFSZ   COUNT1 , F
            GOTO     LOOP
            DECFSZ   COUNT2 , F
            GOTO     LOOP          ; delay 0.500207 seconds
            RETURN

            END
```

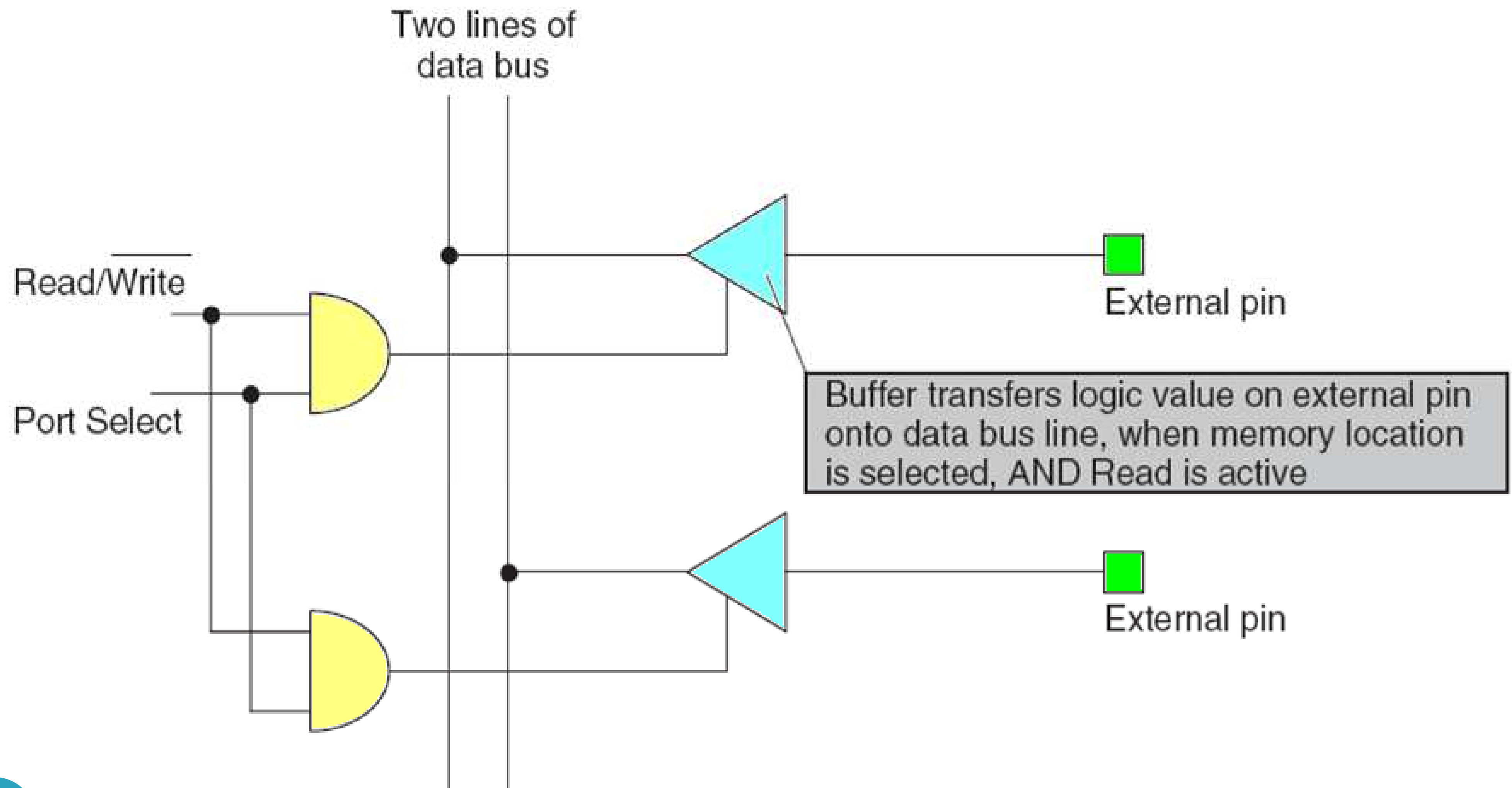
Hardware Realization of Parallel Ports

Output Parallel Port



Hardware Realization of Parallel Ports

Input Parallel Port



Hardware Realization of Parallel Ports

Bidirectional Parallel Port

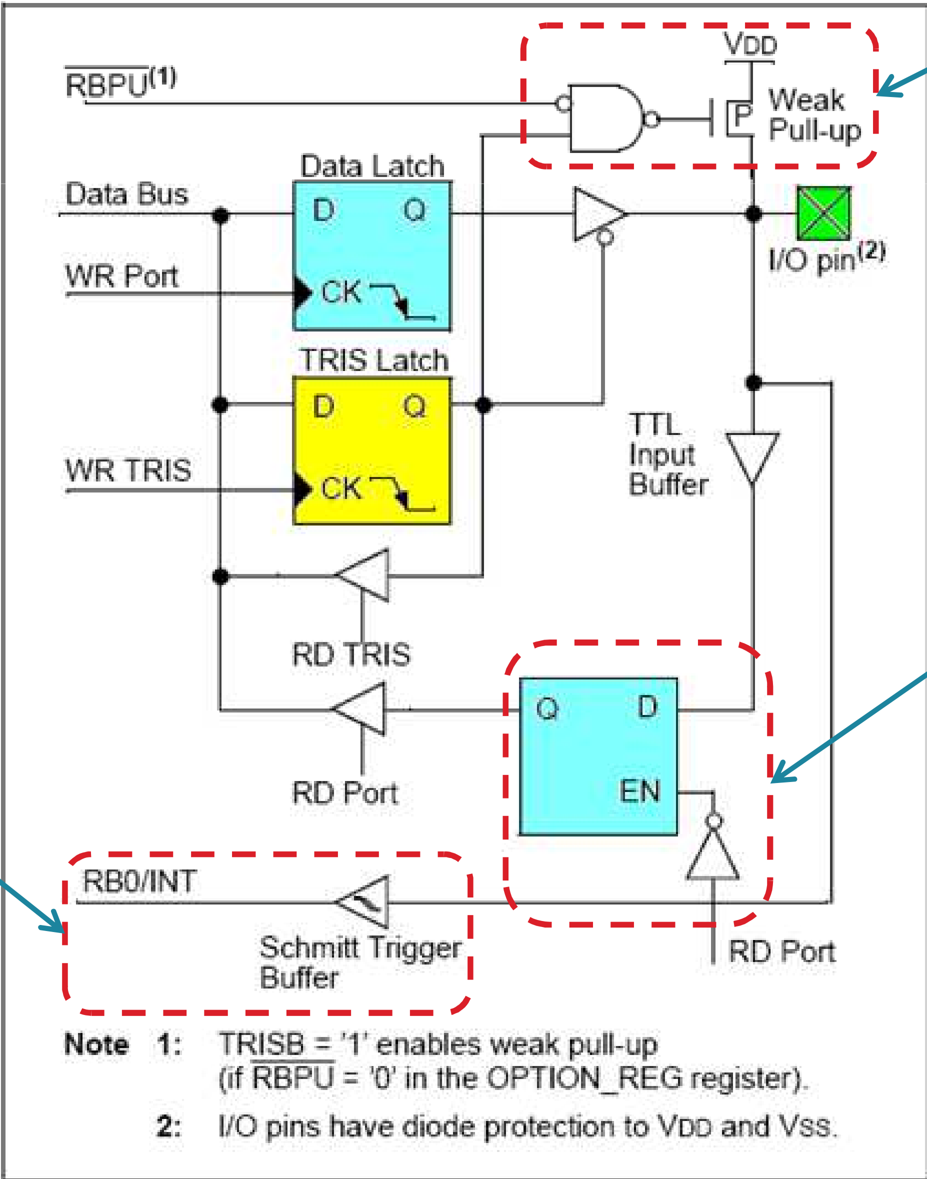
Read/$\overline{Write}$
Read port
Data bus (bit *n*)
Input buffer
I/O pin (bit *n* of an 8-bit port)
'Data'
Output buffer
Write port
D
Q
Holds bit output value
Port Select
8 of these flip-flops form the 'Data' SFR
Buffer, enabled when pin is output
'Direction'
Write DDR
D
Q
Determines whether port bit is input or output
Direction Select
Alternate Input Function
'Data' SFR
8 of these flip-flops form the 'Data Direction' SFR
'Direction' SFR

Hardware Realization of Parallel Ports

Configurable pull-up resistors using RBPU bit in the OPTION register

Latches input data whenever the port is read

Multiplexed input



Hardware Realization of Parallel Ports

PORT B

PINS RB7:RB4

RBPU(1)
Data Bus
WR Port
Data Latch
D Q
CK
TRIS Latch
D Q
WR TRIS
CK
VDD
Weak Pull-up
P
I/O pin(2)
TTL Input Buffer
RD TRIS
Latch
Q D
EN
RD Port
Set RBIF
From other RB7:RB4 pins
Q D
EN
RD Port

Note 1: TRISB = '1' enables weak pull-up (if RBPU = '0' in the OPTION_REG register).

2: I/O pins have diode protection to VDD and VSS.

Lathes data on port read

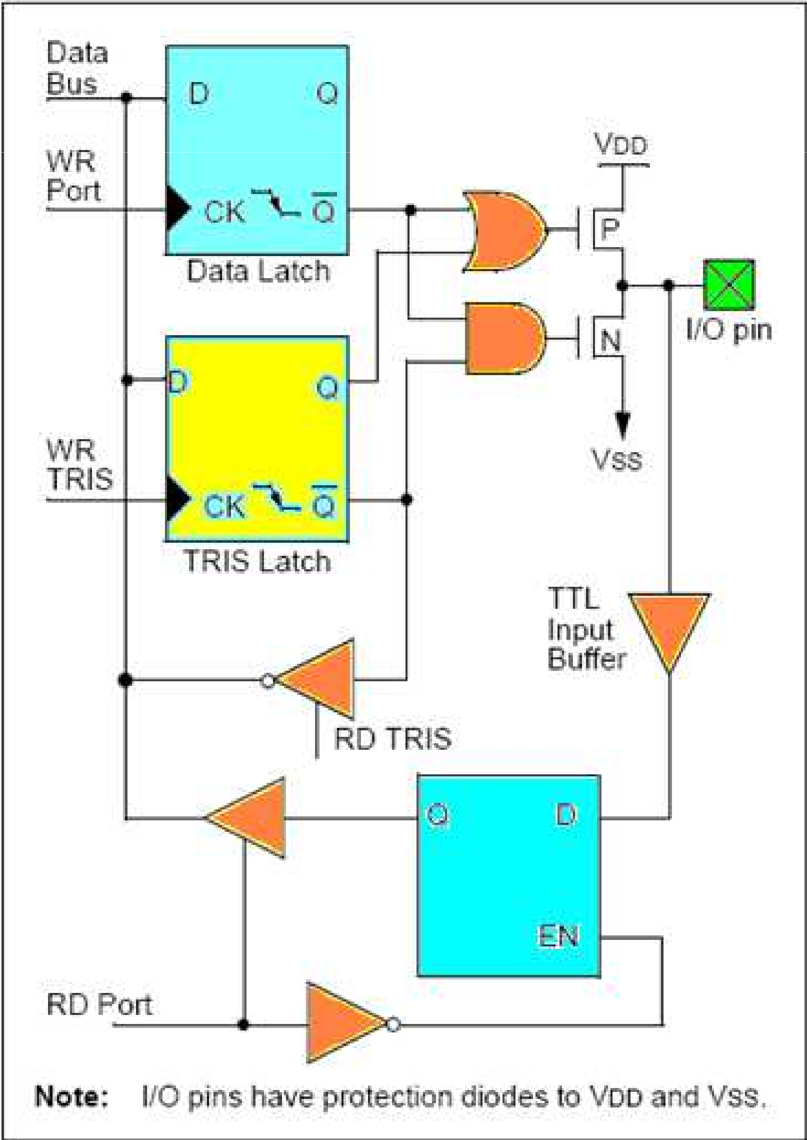
Holds previous latched data

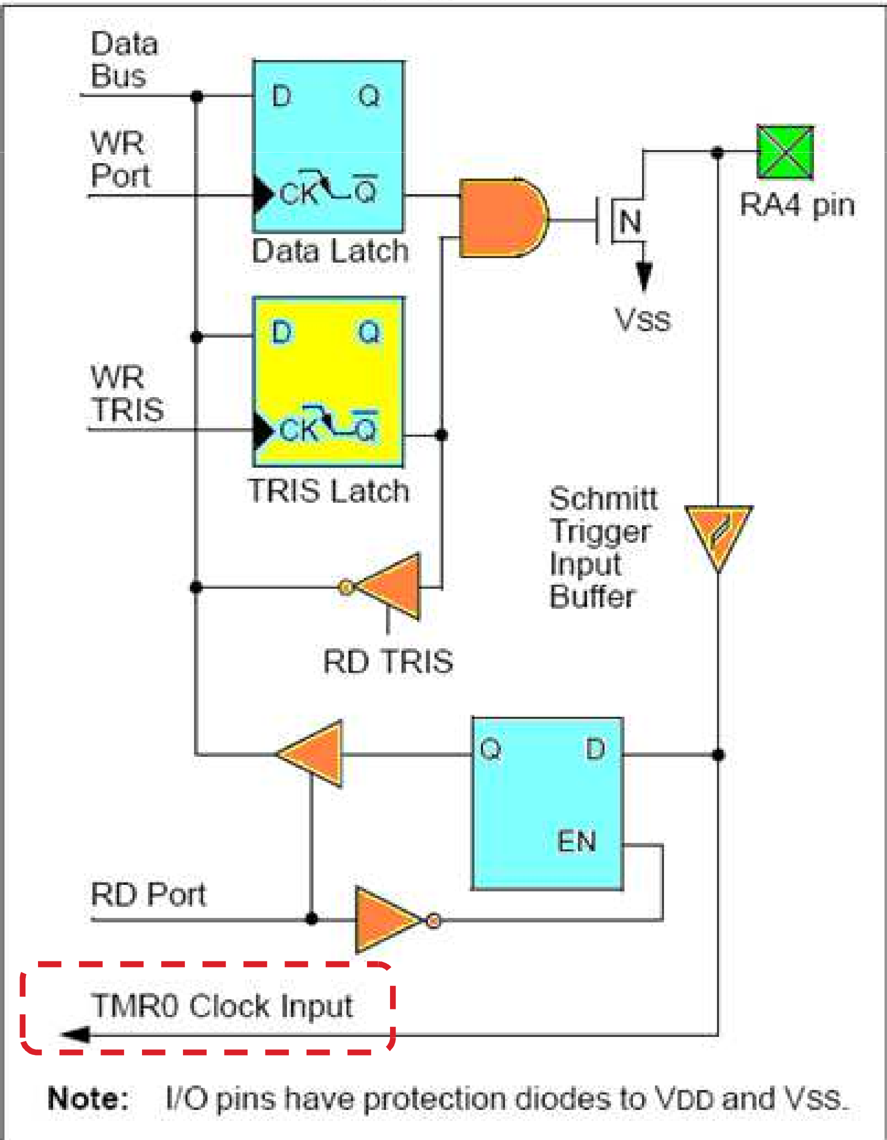
Clearing the RBIF bit ?

Compares previous and present port input values

Hardware Realization of Parallel Ports

PORT A

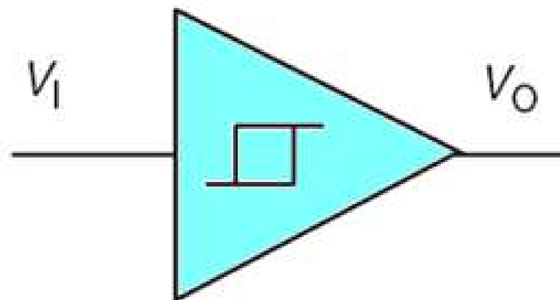


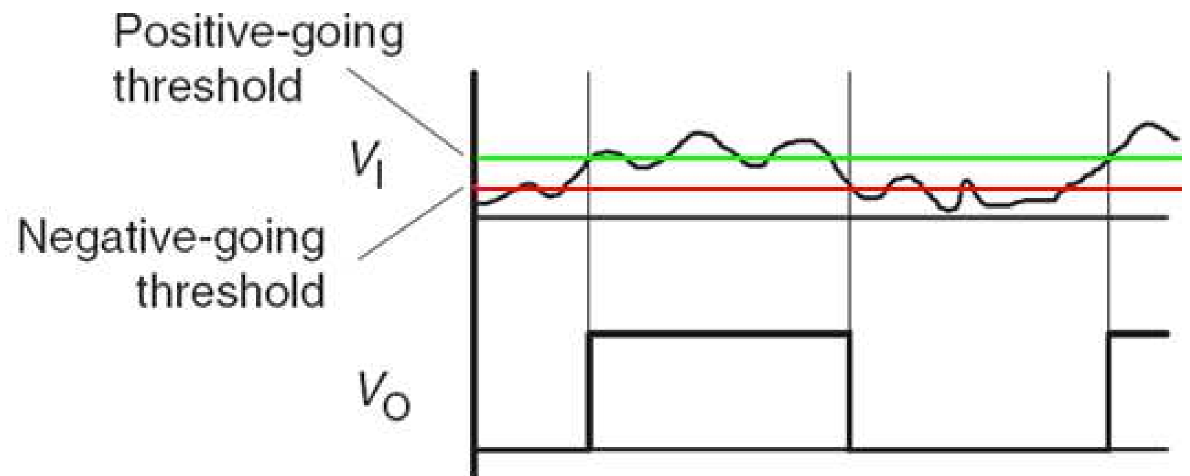


Hardware Realization of Parallel Ports

Electrical Characteristics

- **Schmitt Trigger Input**
  - A special type of gate with two thresholds
  - Remove fluctuations and corruptions in the input signal

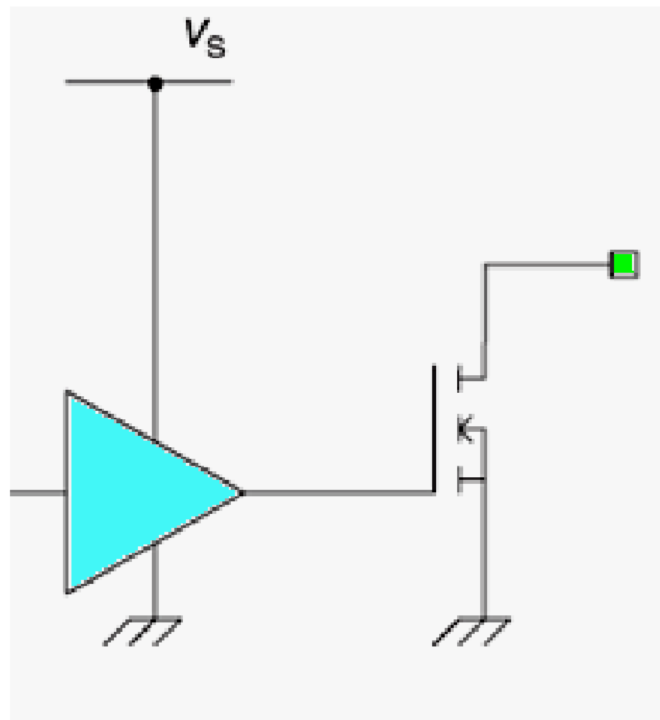




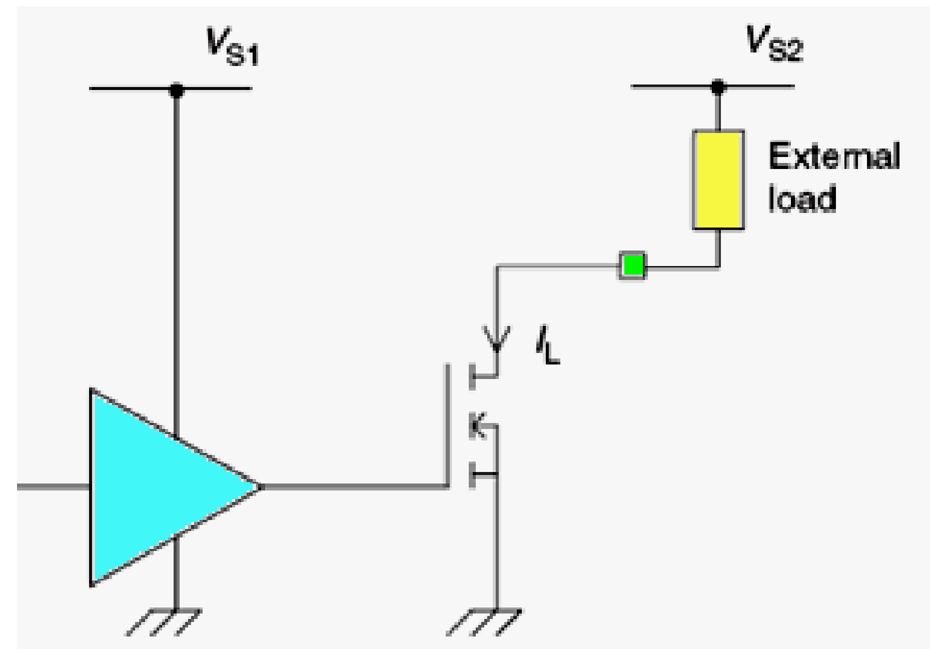
Hardware Realization of Parallel Ports

Electrical Characteristics

- **Open Drain Output**
  - Flexible style of output that can be adapted as a standard logic output or a direct drive for small loads



Open Drain Output

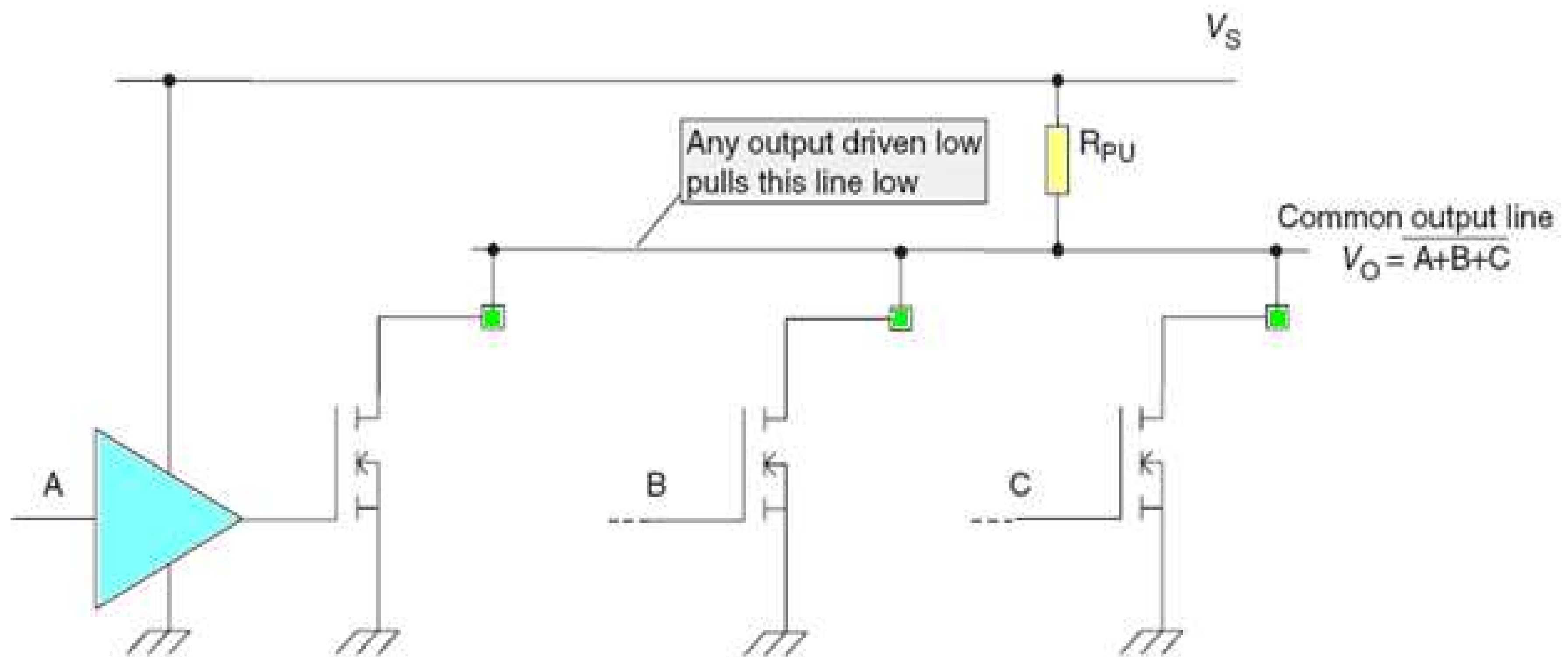


Open Drain Output Driving A Small Load

Hardware Realization of Parallel Ports

Electrical Characteristics

- **Open Drain Output**
  - Can be used as a wired-OR

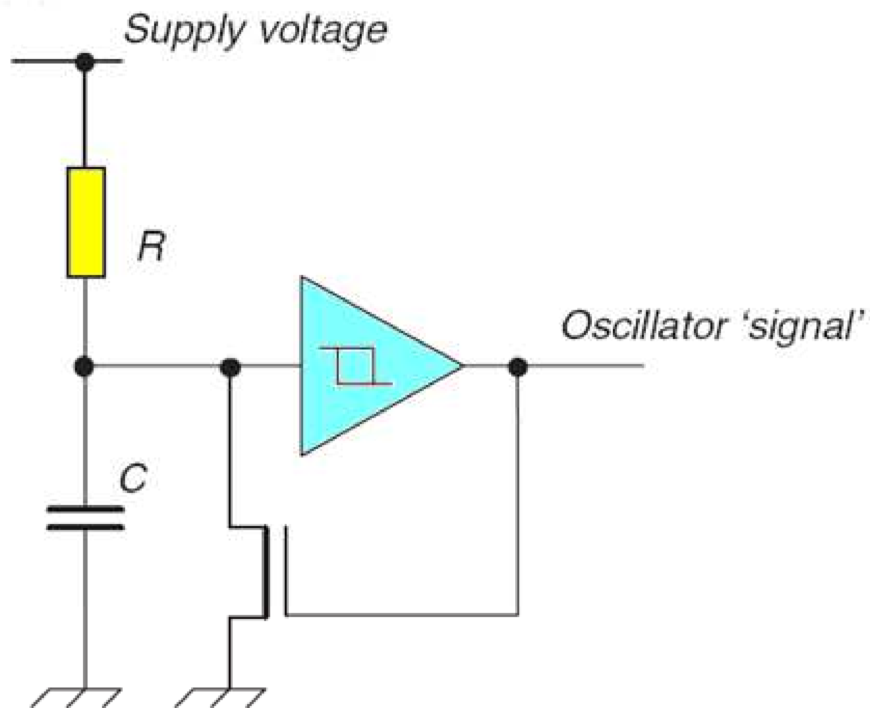


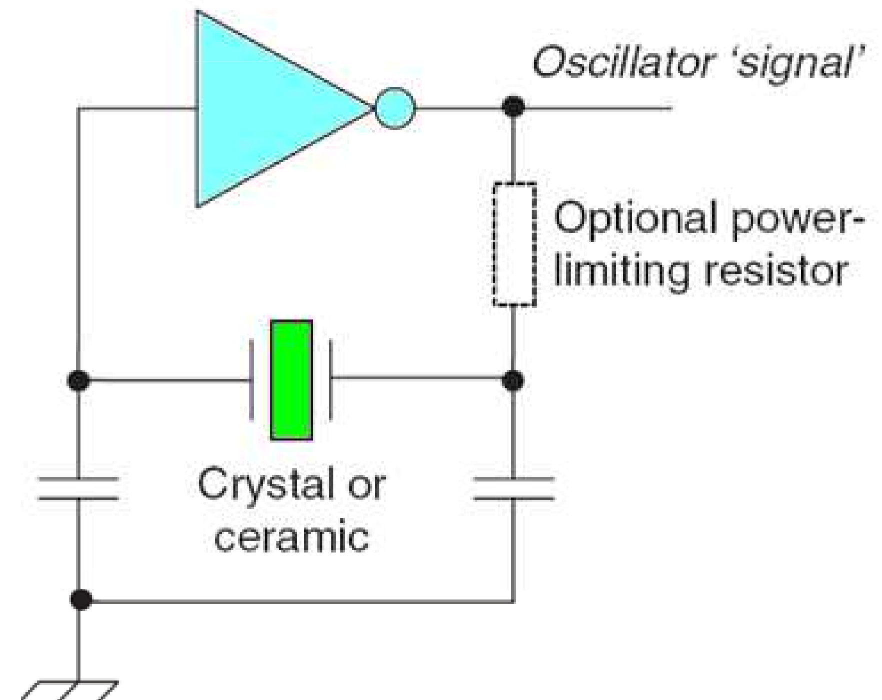
The Oscillator

- The choice of clock determines the operating characteristics for the microcontroller
- *Faster clock gives* faster execution, *but more power consumption*
- Accurate and stable operation of the microcontroller requires accurate and stable clock

The Oscillator

Oscillator types





Resistor–capacitor (RC).

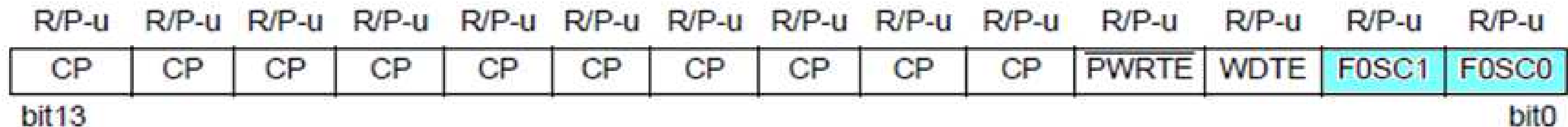
- *low cost*
- *not precise*

Crystal or ceramic

- *expensive*
- *stable and precise*
- *mechanically fragile*

The PIC 16F84A Oscillator

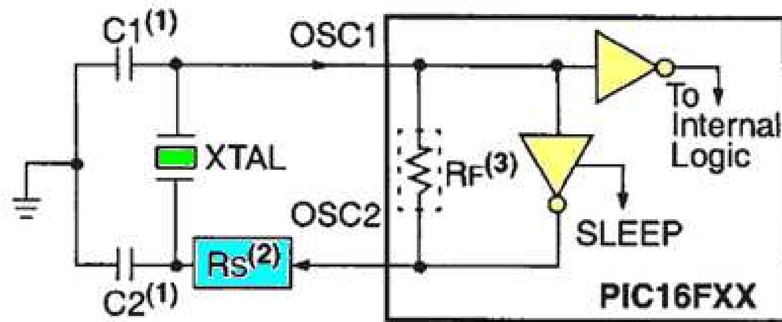
- The 16F84A can be configured to operate in four different oscillator modes using the F0SC1 and F0SC0 in the configuration word



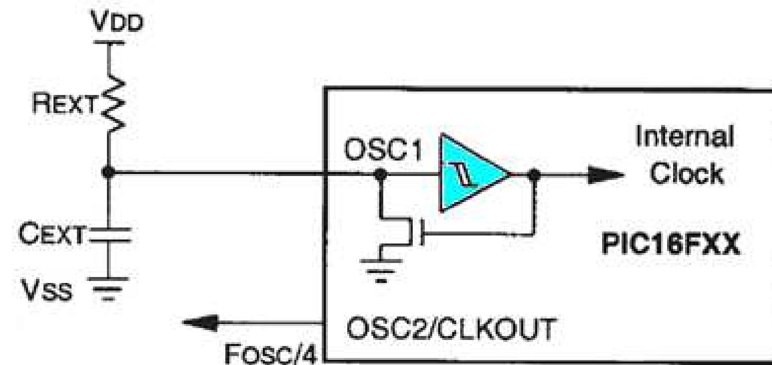
F0SC1	F0SC0	Mode
0	0	LP oscillator – *intended for low frequency (<200 KHz) crystal application to reduce power consumption*
0	1	XT oscillator – *standard crystal configuration (1-4 MHz)*
1	0	HS oscillator – *high speed (>= 4MHz)*
1	1	RC oscillator - *requires external resistor an capacitor*

The PIC 16F84A Oscillator

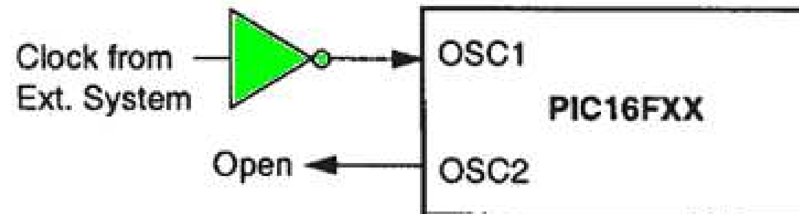
- The 16F84A has two oscillator pins ; OSC1 and OSC2.



XT configuration



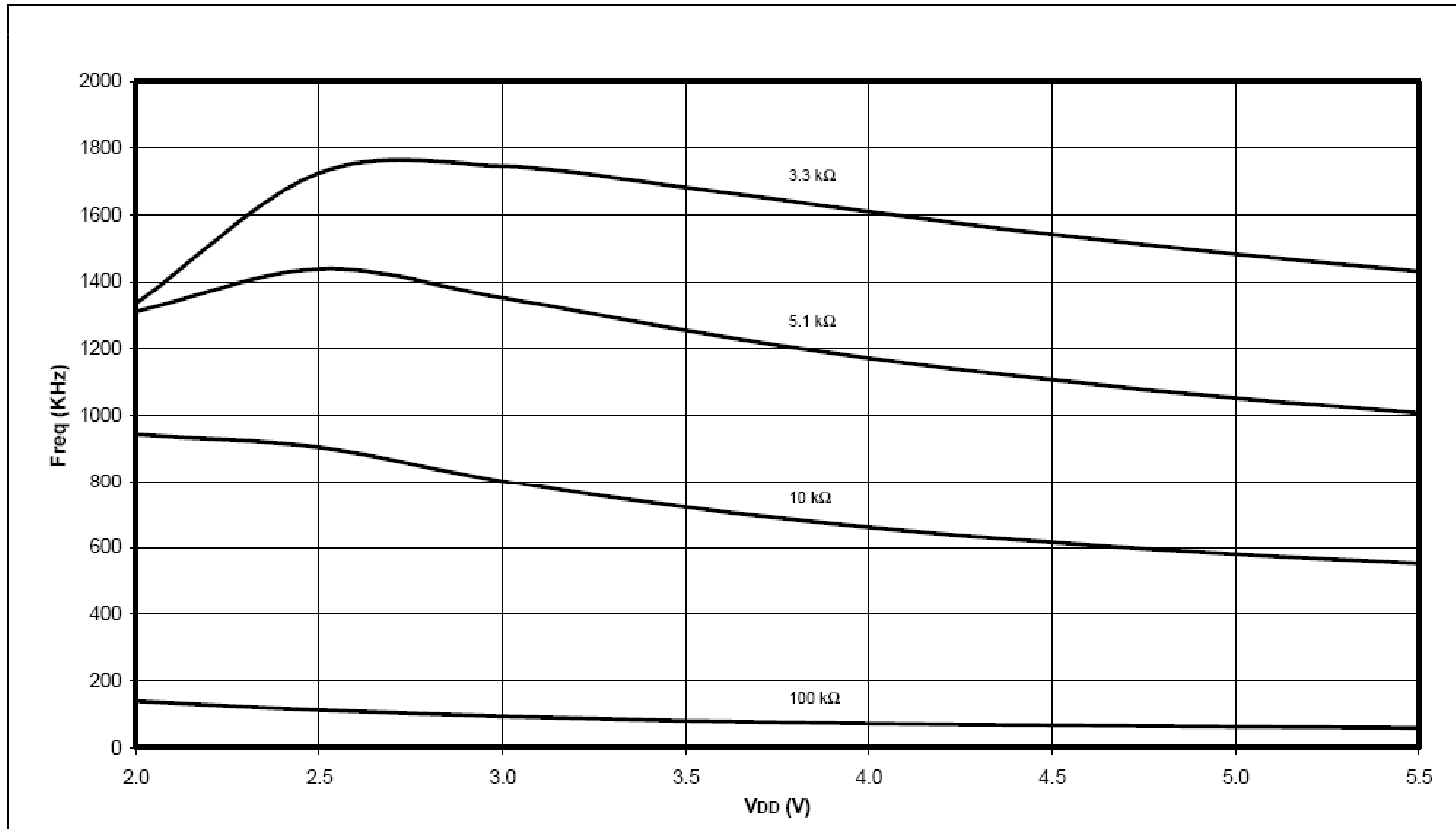
RC configuration



External Clock

The PIC 16F84A Oscillator

- RC oscillator frequency dependence on power supply



The Power Supply

PIC16F84A-04
(Commercial, Industrial, Extended)
PIC16F84A-20
(Commercial, Industrial, Extended)

Standard Operating Conditions (unless otherwise stated)
Operating temperature 0°C ≤ T_A ≤ +70°C (commercial)
-40°C ≤ T_A ≤ +85°C (industrial)
-40°C ≤ T_A ≤ +125°C (extended)

Param No.	Symbol	Characteristic	Min	Typ†	Max	Units	Conditions
	V_{DD}	**Supply Voltage**					
D001		16LF84A	2.0	—	5.5	V	XT, RC, and LP osc configuration
D001 D001A		16F84A	4.0 4.5	— —	5.5 5.5	V V	XT, RC and LP osc configuration HS osc configuration
D002	V_{DR}	**RAM Data Retention Voltage (Note 1)**	1.5	—	—	V	Device in SLEEP mode
D003	V_{POR}	**V_{DD} Start Voltage** to ensure internal Power-on Reset signal	—	V_{SS}	—	V	See section on Power-on Reset for details
D004	S_{VDD}	**V_{DD} Rise Rate** to ensure internal Power-on Reset signal	0.05	—	—	V/ms	
	I_{DD}	**Supply Current (Note 2)**					
D010		16LF84A	—	1	4	mA	RC and XT osc configuration **(Note 4)** F_{OSC} = 2.0 MHz, V_{DD} = 5.5V
D010 D010A D013		16F84A	— — —	1.8 3 10	4.5 10 20	mA mA mA	RC and XT osc configuration **(Note 4)** F_{OSC} = 4.0 MHz, V_{DD} = 5.5V RC and XT osc configuration **(Note 4)** F_{OSC} = 4.0 MHz, V_{DD} = 5.5V (During FLASH programming) HS osc configuration (PIC16F84A-20) F_{OSC} = 20 MHz, V_{DD} = 5.5V
D014		16LF84A	—	15	45	µA	LP osc configuration F_{OSC} = 32 kHz, V_{DD} = 2.0V, WDT disabled

The Power Supply

100 nF decoupling capacitor

Summary

- Parallel ports allow the exchange of data between the outside world and the CPU
- It is essential to understand the electrical characteristics and internal circuitry of ports
- All microcontrollers need a clock. The clock speed determine the power consumption
- Active elements of the oscillator are usually built inside the microcontroller and the designer selects the type and configure it
- It is a must to understand the power requirements of the microcontroller

Starting with Serial

Chapter 10
Sections 1,2,9,10

Dr. Iyad Jafar

Outline

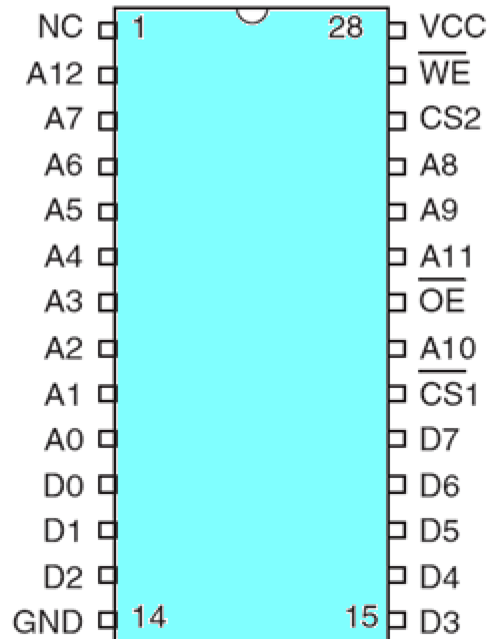
- Introduction
- Synchronous Serial Communication
- Asynchronous Serial Communication
- Physical Limitations
- Overview of PIC 16 Series
- The 16F87xA USART
- Summary

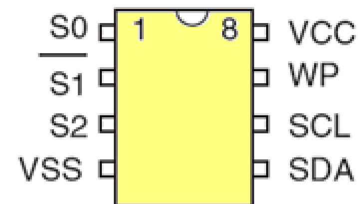
Introduction

- Microcontrollers need to move data to and from external devices
- In general, two approaches
  - ***Parallel***
    - Data word bits are transferred at the same time
    - A wire is dedicated for each bit
    - Simple and fast but expensive
    - Short distances
  - ***Serial***
    - Bits are transferred one after another over the same link/wire
    - Requires complex hardware to transmit and receive
    - Slow but cheap
    - Short and long distances

Introduction

- Two memories of the same size. However, one uses parallel transfer while the other uses serial





Serial Communication

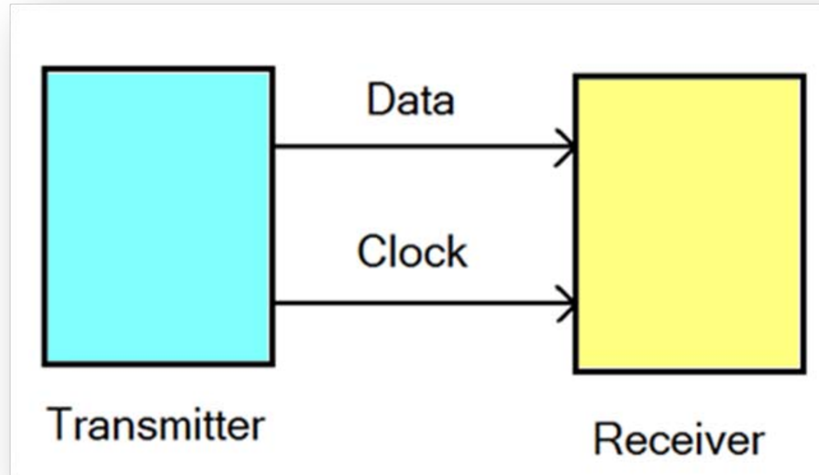
- Bits are transferred one after another on the same wire !!!

- **Challenges**
  - How to distinguish the start and end of the bit ?
  - How to determine the start and end of a word ?

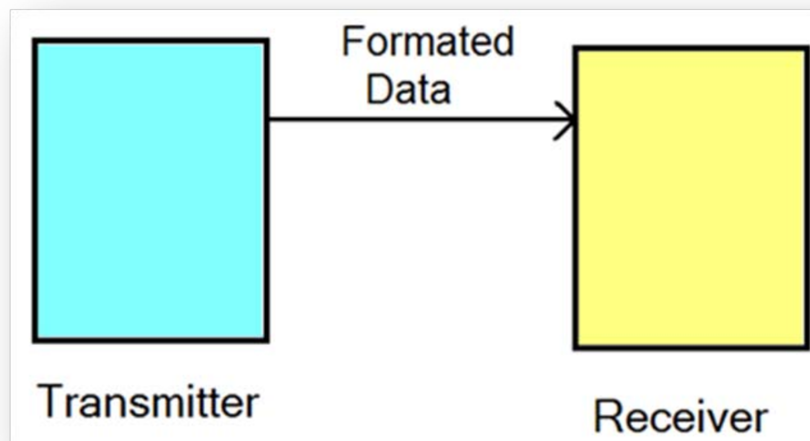
- **Two approaches**
  - Synchronous serial communication
    - A separate clock signal is sent in parallel with the data
    - Each clock cycle represents one bit duration
  - Asynchronous serial communication
    - No clock signal !
    - Timing is derived from the data itself

Serial Communication

Synchronous

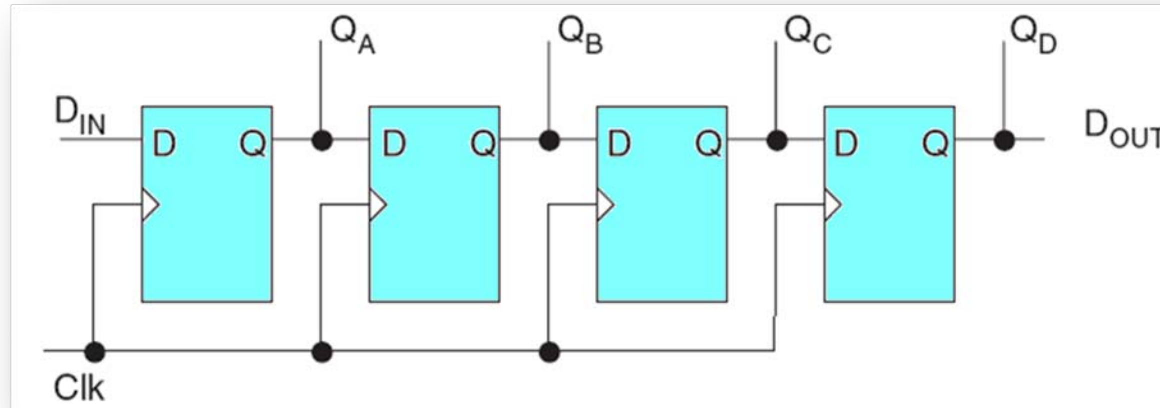


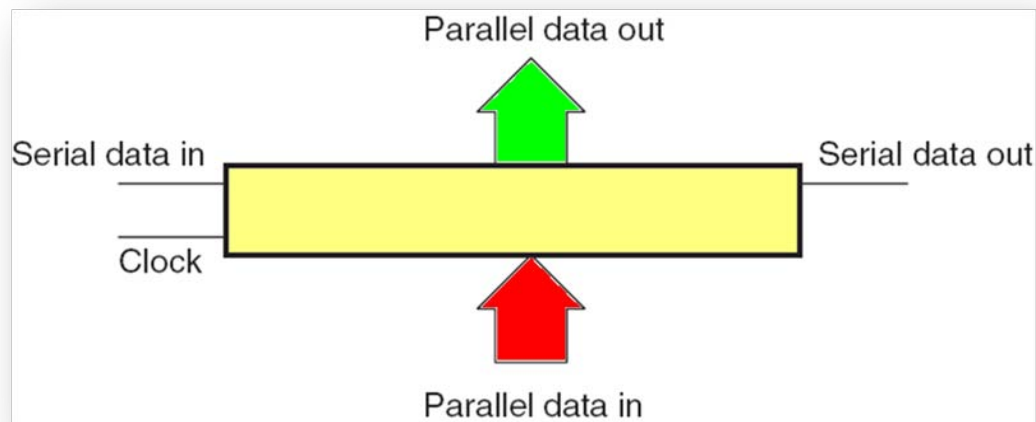
Asynchronous



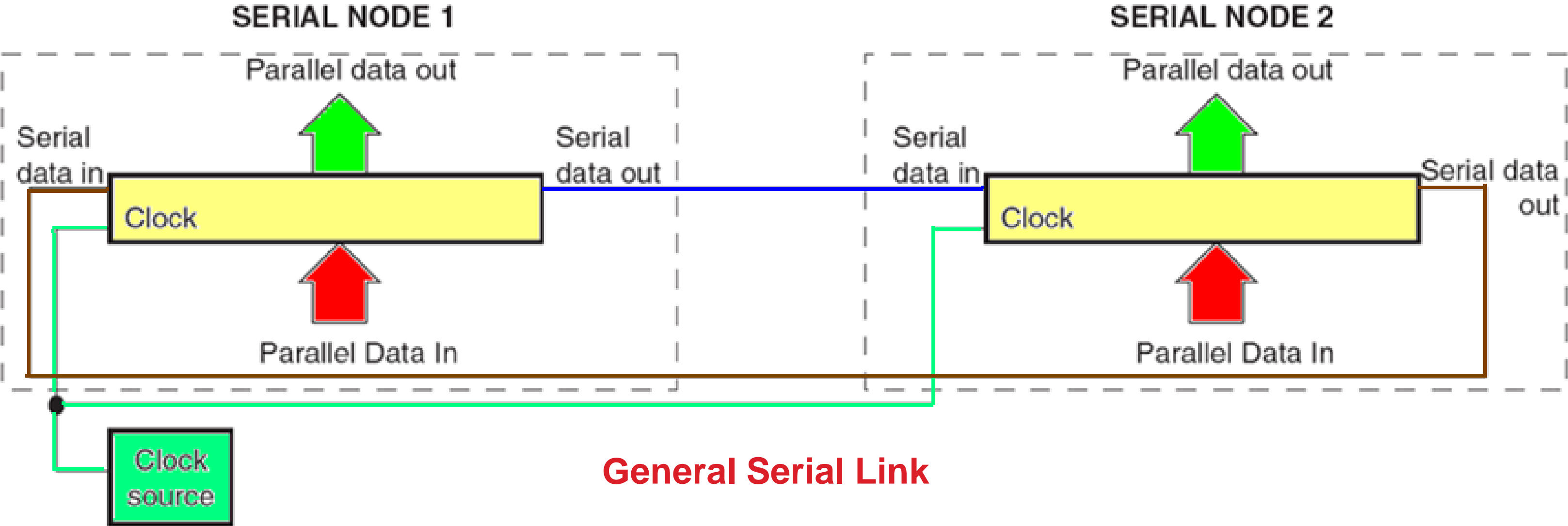
Serial Communication

- Data inside the memory and microprocessor is formatted in parallel. How to transmit it serially?
- Shift registers

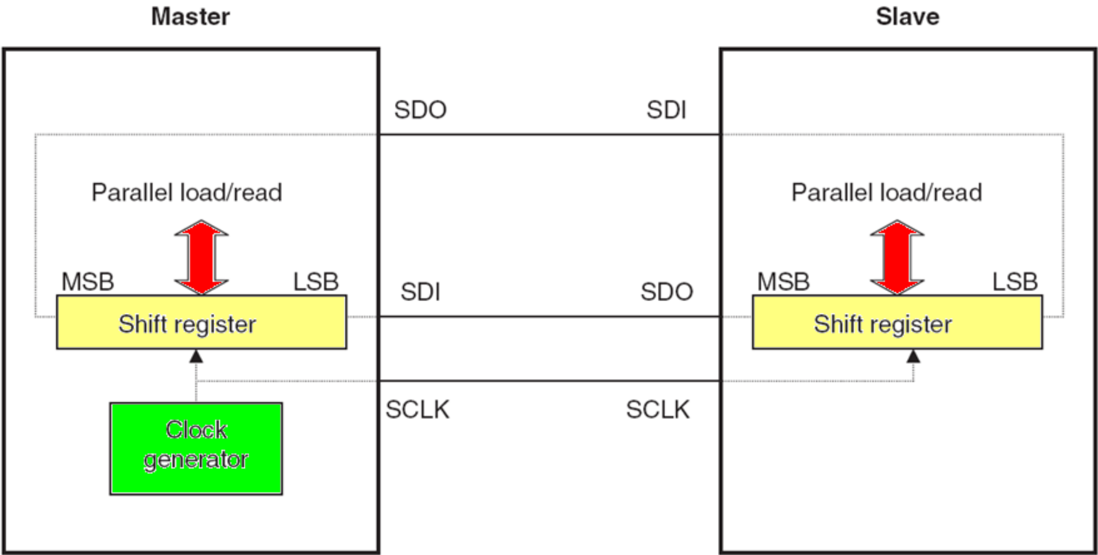




Synchronous Serial Communication

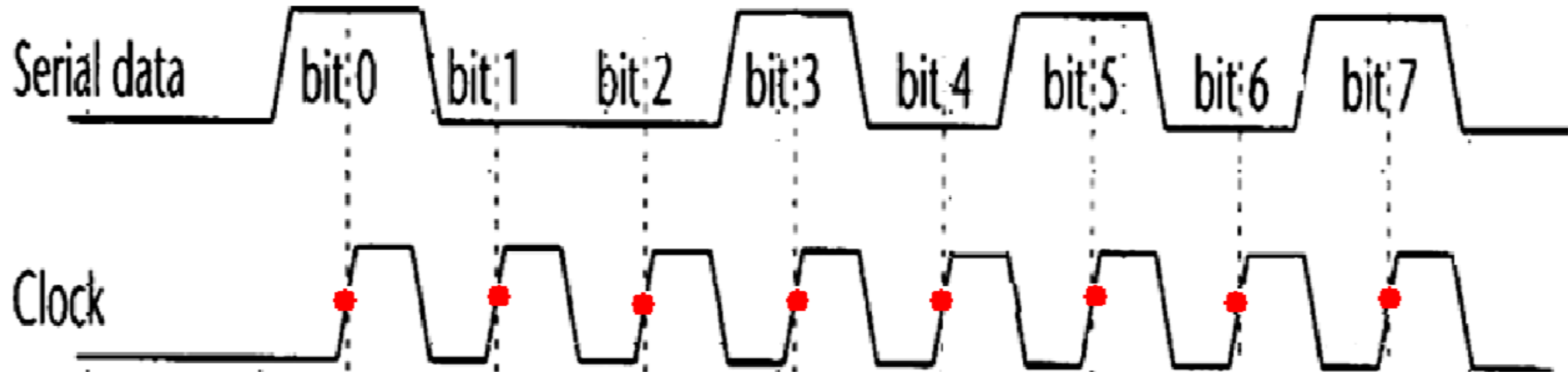


General Serial Link



Synchronous link implemented using a microcontroller

Synchronous Serial Communication



Advantages

- Simple hardware
- Efficient
- High speed

Disadvantages

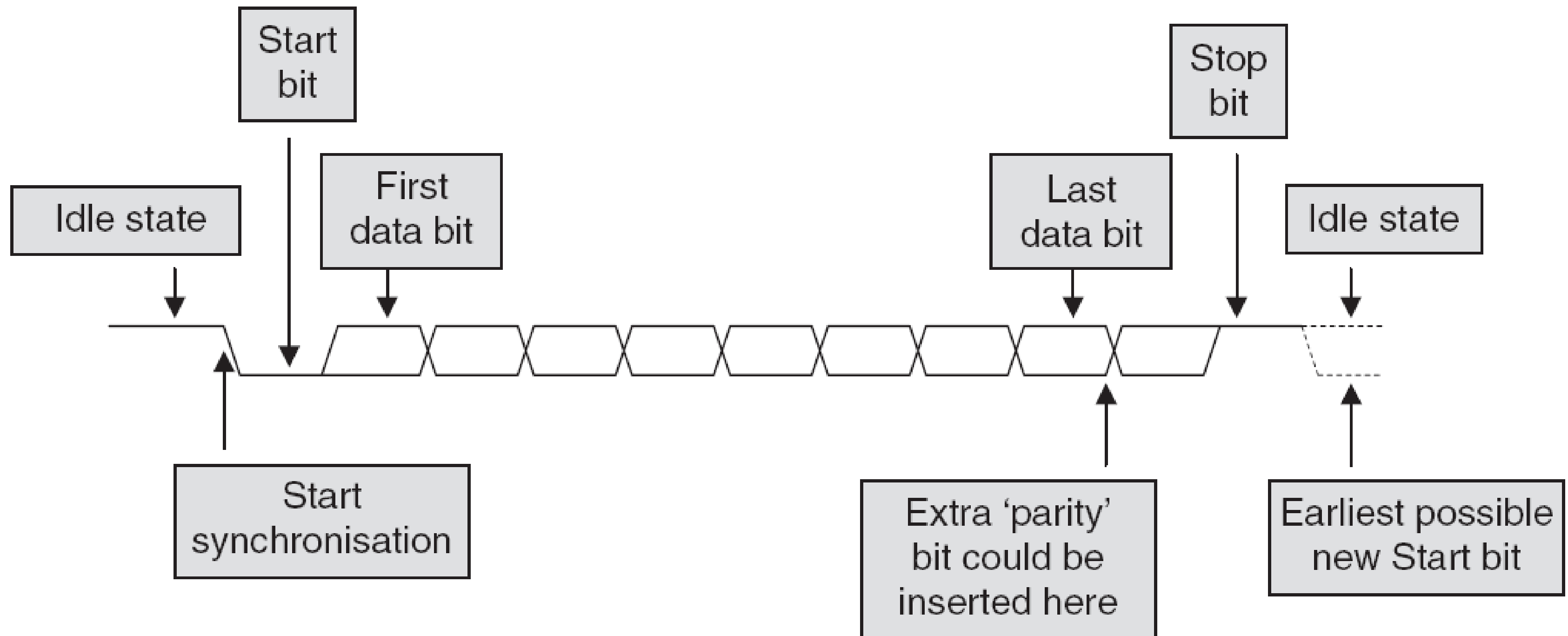
- Extra line for the clock
- The bandwidth needed for the clock is twice the data bandwidth
- Data and clock may lose synchronization over long distance

Asynchronous Serial Communication

- No clock signal !
- The transmitter and receiver should operate a clock at the same rate
- To synchronize the clocks of the transmitter and receiver, data is framed with a start and stop bits

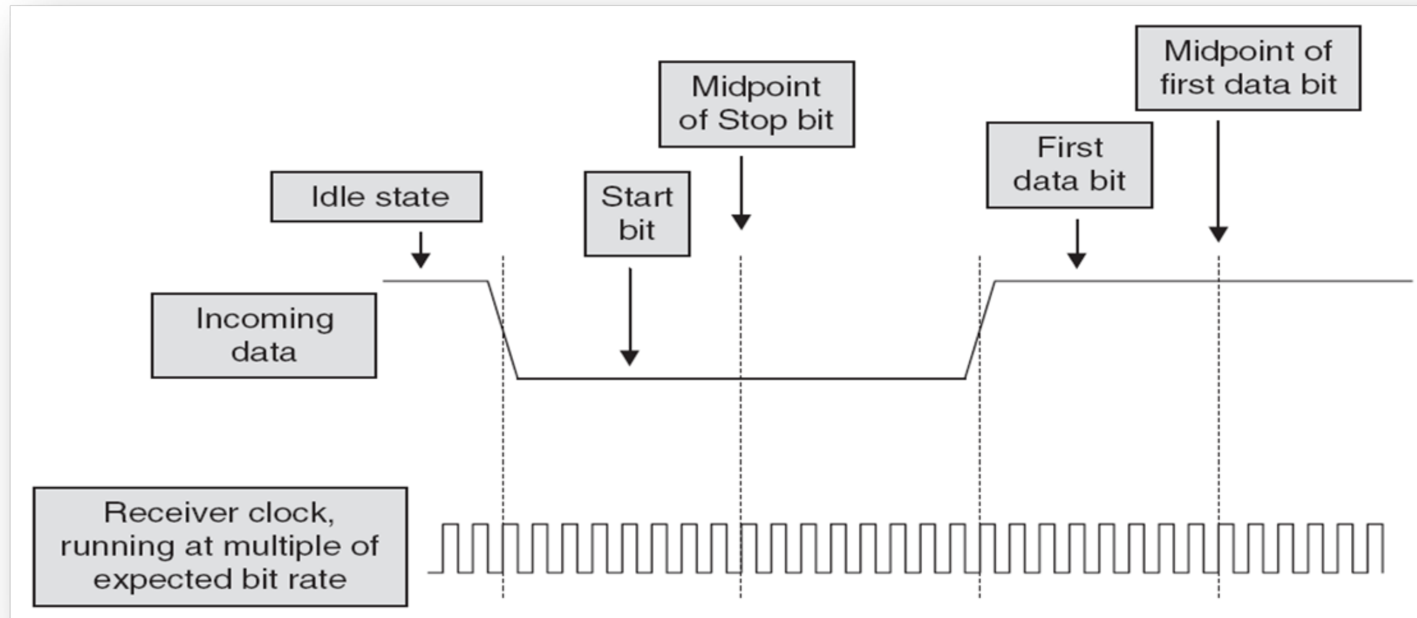
Asynchronous Serial Communication

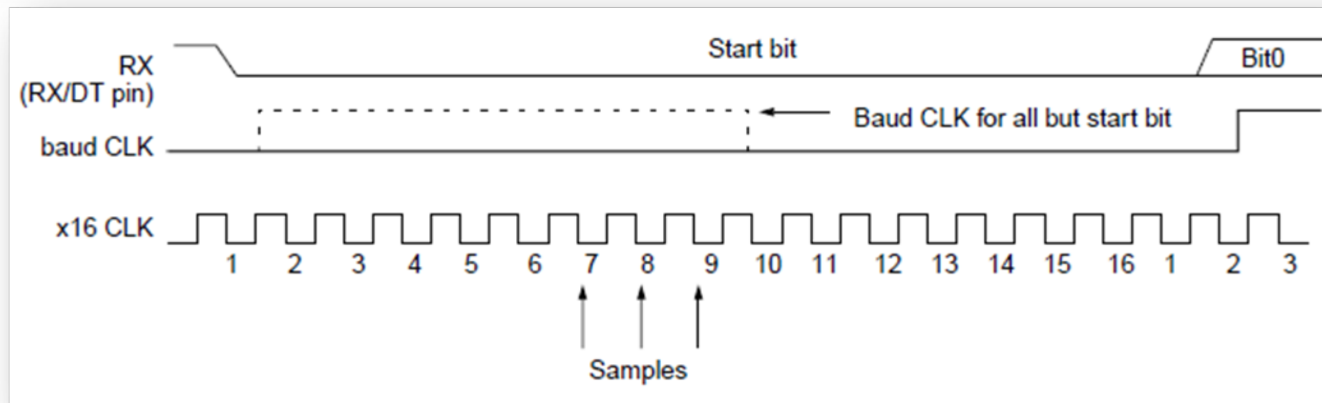
- **Framing**



Asynchronous Serial Communication

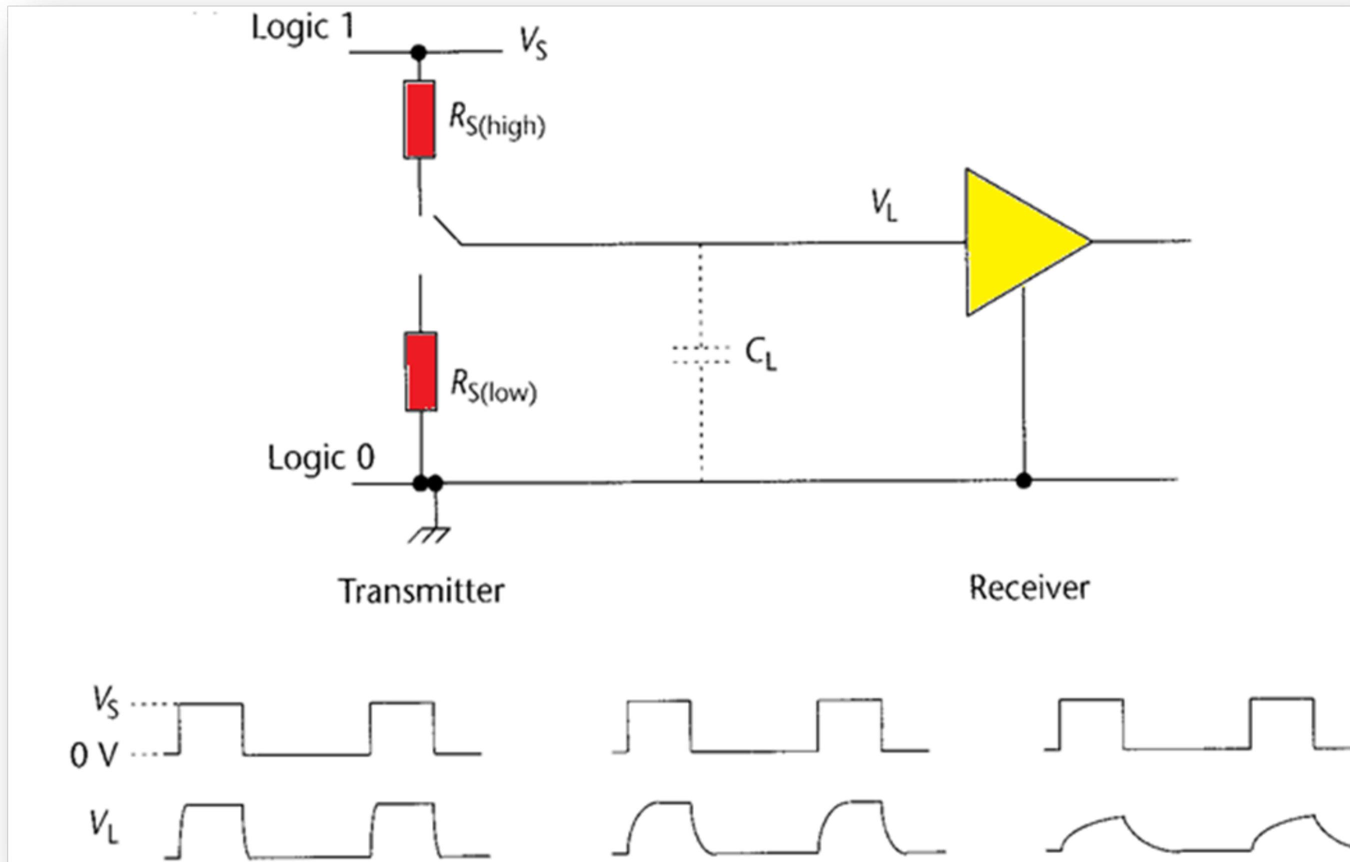
- **Synchronization**





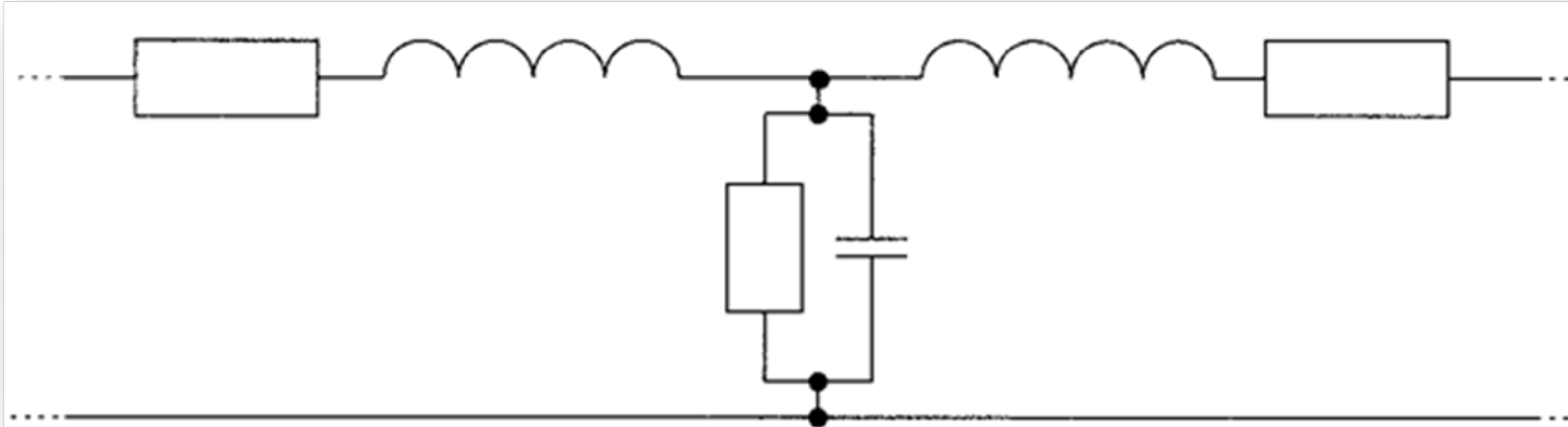
Physical Limitations

- Time Constant effect



Physical Limitations

- Transmission Line Effects
  - Characteristic impedance and reflections
  - Lines should be terminated properly

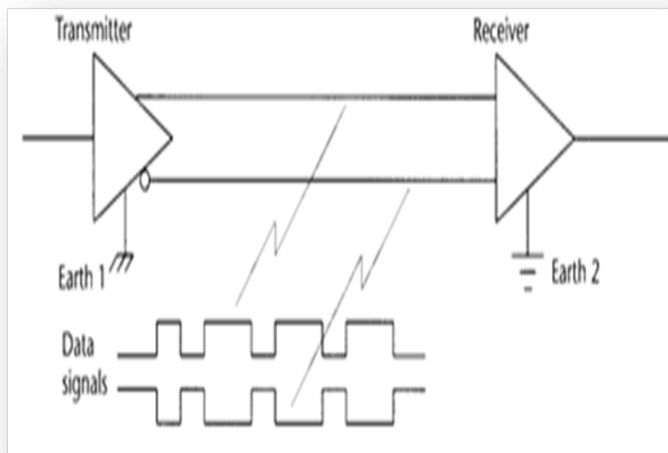


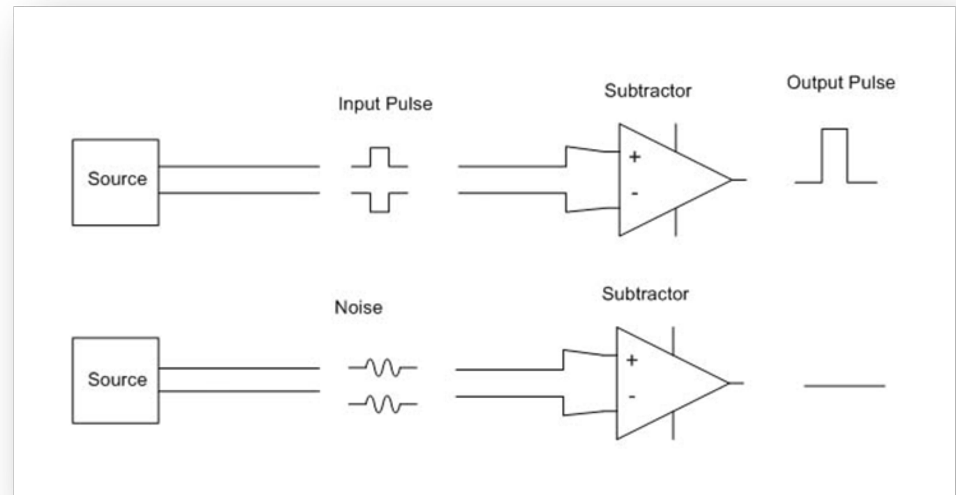
Physical Limitations

- **Electromagnetic Interference**
  - Generated due to high voltage rates of change.
  - How to minimize:
    - At source:
      - reduce voltage rate of change.
    - In communication link:
      - large separation from source of interference.
      - Increase data voltage.
      - Screening
      - Use optical links
    - At receiver:
      - Use filtering techniques

Physical Limitations

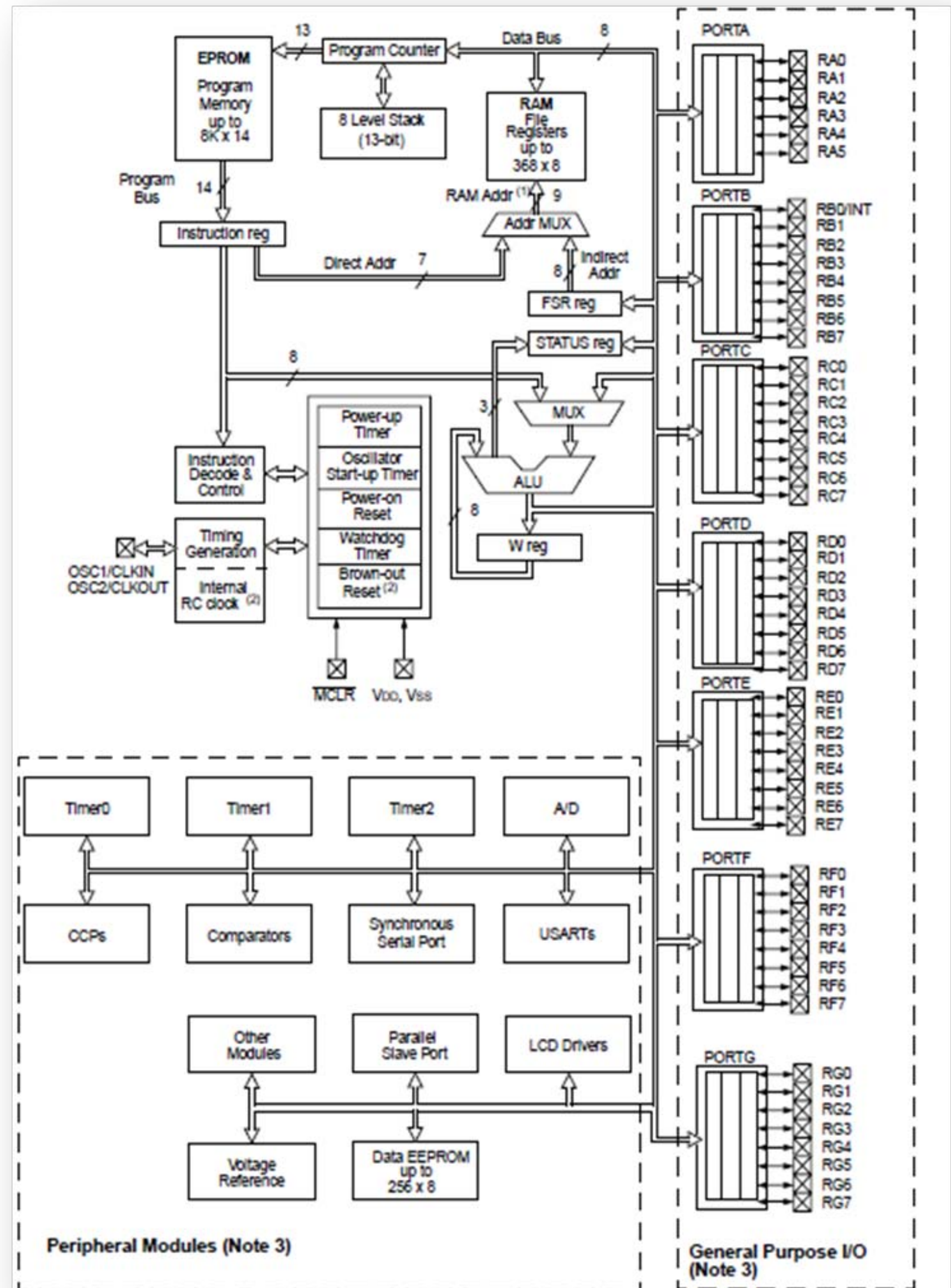
- **Ground Differentials**
  - With longer wires, ground potential at one point might not be the same at another point.
  - Solutions:
    - Differential transmission.
    - Electrical isolation
    - Use optical communication links





Overview of the PIC 16 Series

- We have already seen the PIC 16F84A
- Other members in the series have more features:
  - Additional I/O ports
  - More HW timers
  - A/D converters
  - LCD Drivers
  - USARTs
  - Synchronous Serial
  - Comparators
  - ….



Overview of the PIC 16 Series

PCLATH

PC<12:0> | PC<12:8> | PCL

CALL, RETURN
RETFIE, RETLW

13

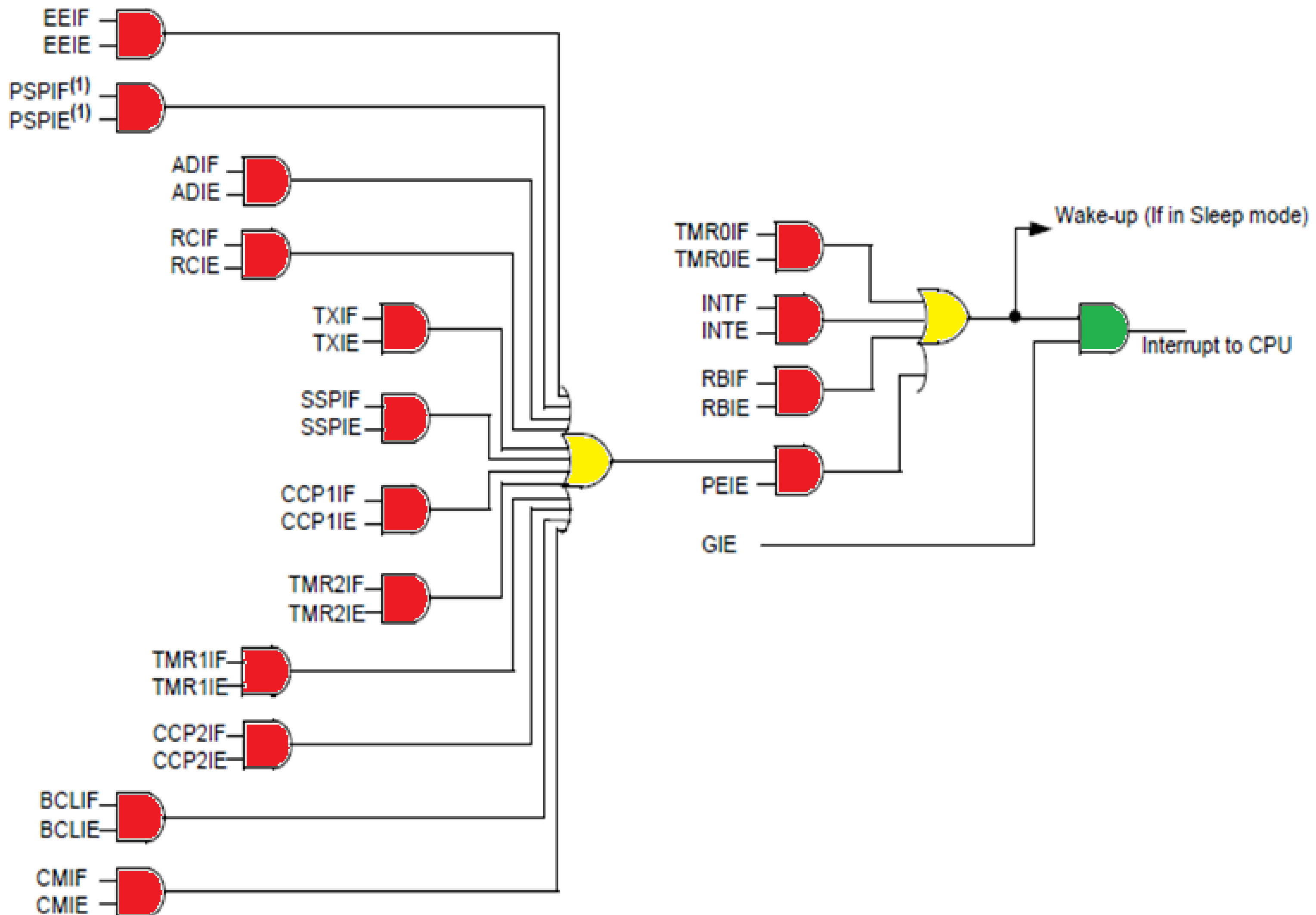
Stack Level 1
⋮
Stack Level 8

	Program Memory	Address
2K, 4K, 6K, 8K	Reset Vector	0000h
	⋮	
	Interrupt Vector	0004h
	On-chip Program Memory (Page 0)	0005h – 07FFh
	On-chip Program Memory (Page 1)	0800h – 0FFFh
	On-chip Program Memory (Page 2)	1000h – 17FFh
	On-chip Program Memory (Page 3)	1800h – 1FFFh

Bank0	File Address	Bank1	File Address	Bank2	File Address	Bank3	File Address
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	PORTF	107h	TRISF	187h
PORTD	08h	TRISD	88h	PORTG	108h	TRISG	188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch		10Ch		18Ch
PIR2	0Dh	PIE2	8Dh		10Dh		18Dh
TMR1L	0Eh	PCON	8Eh		10Eh		18Eh
TMR1H	0Fh	OSCCAL	8Fh		10Fh		18Fh
T1CON	10h		90h		110h		190h
TMR2	11h		91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPATAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRES	1Eh		9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Registers (2)	20h – 7Fh	General Purpose Registers (3)	A0h – EFh	General Purpose Registers (3)	120h – 16Fh	General Purpose Registers (3)	1A0h – 1EFh
		Mapped in Bank0 70h - 7Fh (4)	F0h – FFh	Mapped in Bank0 70h - 7Fh (4)	170h – 17Fh	Mapped in Bank0 70h - 7Fh (4)	1F0h – 1FFh
Bank0		Bank1		Bank2 (5)		Bank3 (5)	

Overview of the PIC 16 Series

Interrupt Logic for 16F874A/16F877A



Overview of the PIC 16 Series

Device	Pins	Features
16F873A 16F876A	28	3 parallel ports, 3 counter/timers, 2 capture/compare/PWM, **2 serial,** 5 10-bit ADC, 2 comparators
16F874A 16F877A	**40**	**5** parallel ports, 3 counter/timers, 2 capture/compare/PWM, **2 serial,** **8** 10-bit ADC, 2 comparators

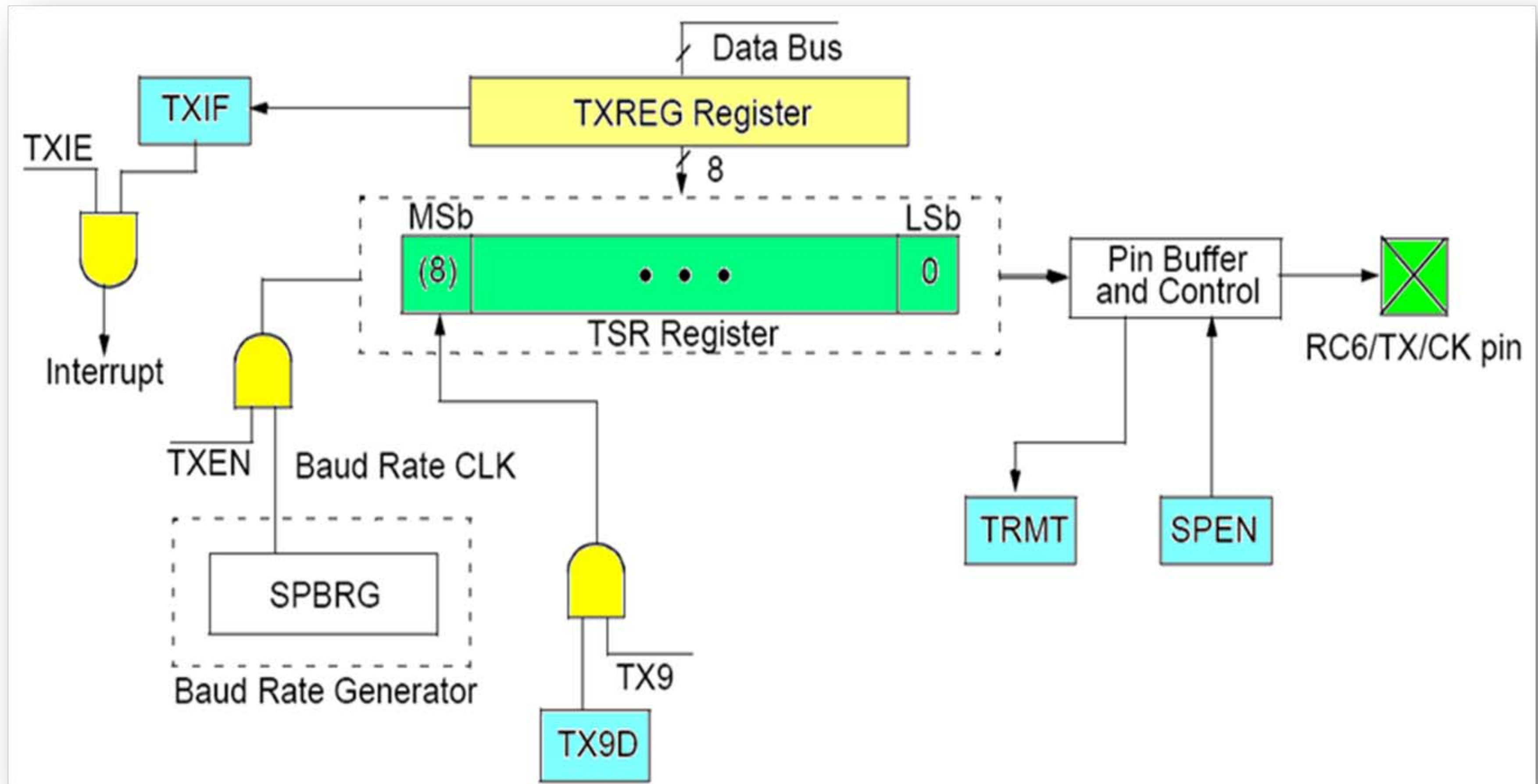
The 16F87xA USART

- The 16F87XA family has a Universal Synchronous Asynchronous Receiver Transmitter (USART)
  - Configurable
  - Half duplex synchronous master or slave
  - Full-duplex asynchronous transmitter and receiver
- The USART shares pins with PORTC
  - pin 7 being the receive line
  - pin 6 being the transmit line
- Operation involves the following registers

  TXSTA (0x98) TXREG (0x19) RCSTA (0x18)
  RCREG (0x1A) SPBRG (0x99) PIE1 (0x8C)
  PIR1 (0x0C) INTCON (0x0B, 0x8B,0x10B,0x18B)
  TRISC (0x87)

The 16F87xA USART

- **Asynchronous USART Transmitter Block Diagram**



The 16F87xA USART

- **Asynchronous USART Transmitter Operation Notes**
  - Data is transmitted LSB first on RC6 pin
  - The shift register TSR is buffered by the TXREG (19H) and is not accessible as a memory location
  - Transmission is controlled by the TXEN bit which enables the clock to start the transmission
  - To enable serial transmission on RC6, bit SPEN in RCSTA register has to be set
  - To transmit data, it must be loaded in the TXREG. It is transferred to TSR immediately if no transmission or after the stop bit from previous transmission is sent out
  - Transmission status is provided by two bits:
    - TXIF flag in PIR1 register indicates the status of TXREG. It is set when data is transferred to TSR. It is cleared on writing to TXREG. (TXIF is cleared by hardware and it is read-only).
    - TRMT flag in TXSTA it is set when the shift register is empty
  - Parity bit can be sent out by using TXD9 bit and TX9 in TXSTA

The 16F87xA USART

TXSTA (98H)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit
Asynchronous mode:
Don't care.
Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled

Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC**: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode.

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be Parity bit

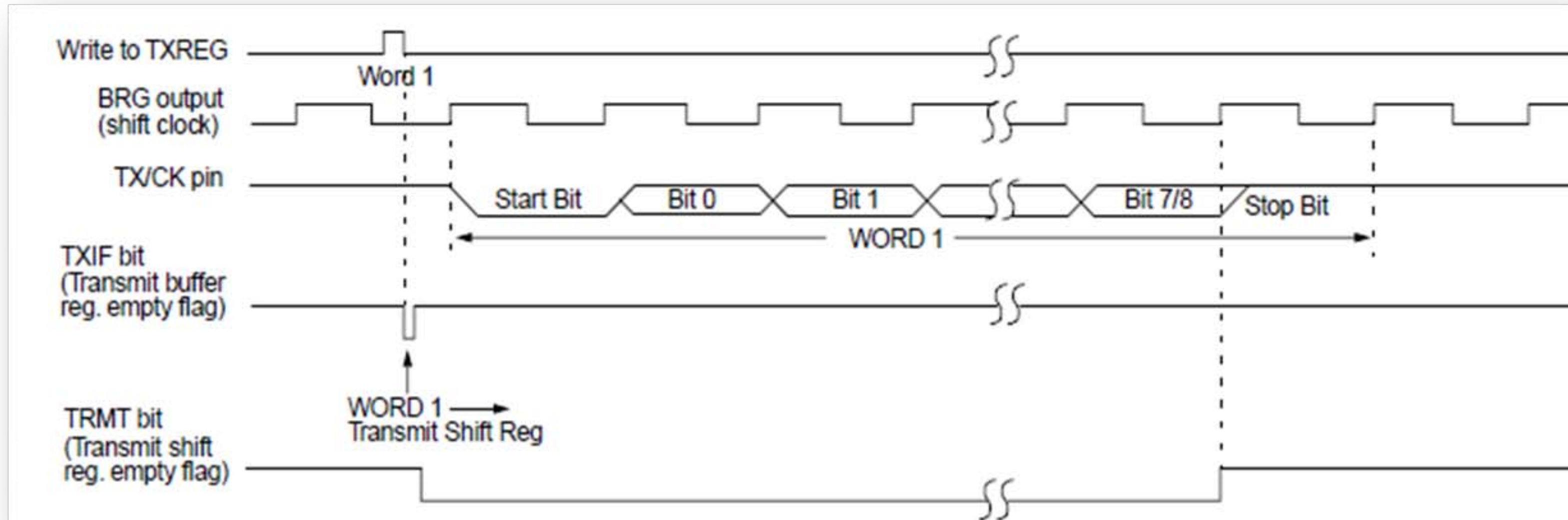
The 16F87xA USART

- **Steps for Using the asynchronous transmitter**

1. Clear TRISC<6> bit to configure RC6 as output
2. Set the SPBRG (0x99) register and BRGH (TXSTA<2>) bit to choose the appropriate baud rate (more on this later)
3. Enable asynchronous serial port by clearing the SYNC (TXSTA<4>) bit and setting the SPEN bit (RCTSA<7>)
4. If interrupts are desired, set the TXIE (PIE1<4>), GIE (INTCON<7>), and PEIE (INTCON<6>) bits
5. If 9-bit transmission is desired, set the TX9 (TXSTA<6>) bit
6. Enable transmission by setting the TXEN (TXSTA<5>), which will set the TXIF (PIR1<4>) bit
7. If 9-bit transmission is selected, then the ninth bit should be loaded in TX9D (TXSTA<0>)
8. Load data in TXREG (0x19) to start the transmission

The 16F87xA USART

- **Timing of asynchronous transmission**

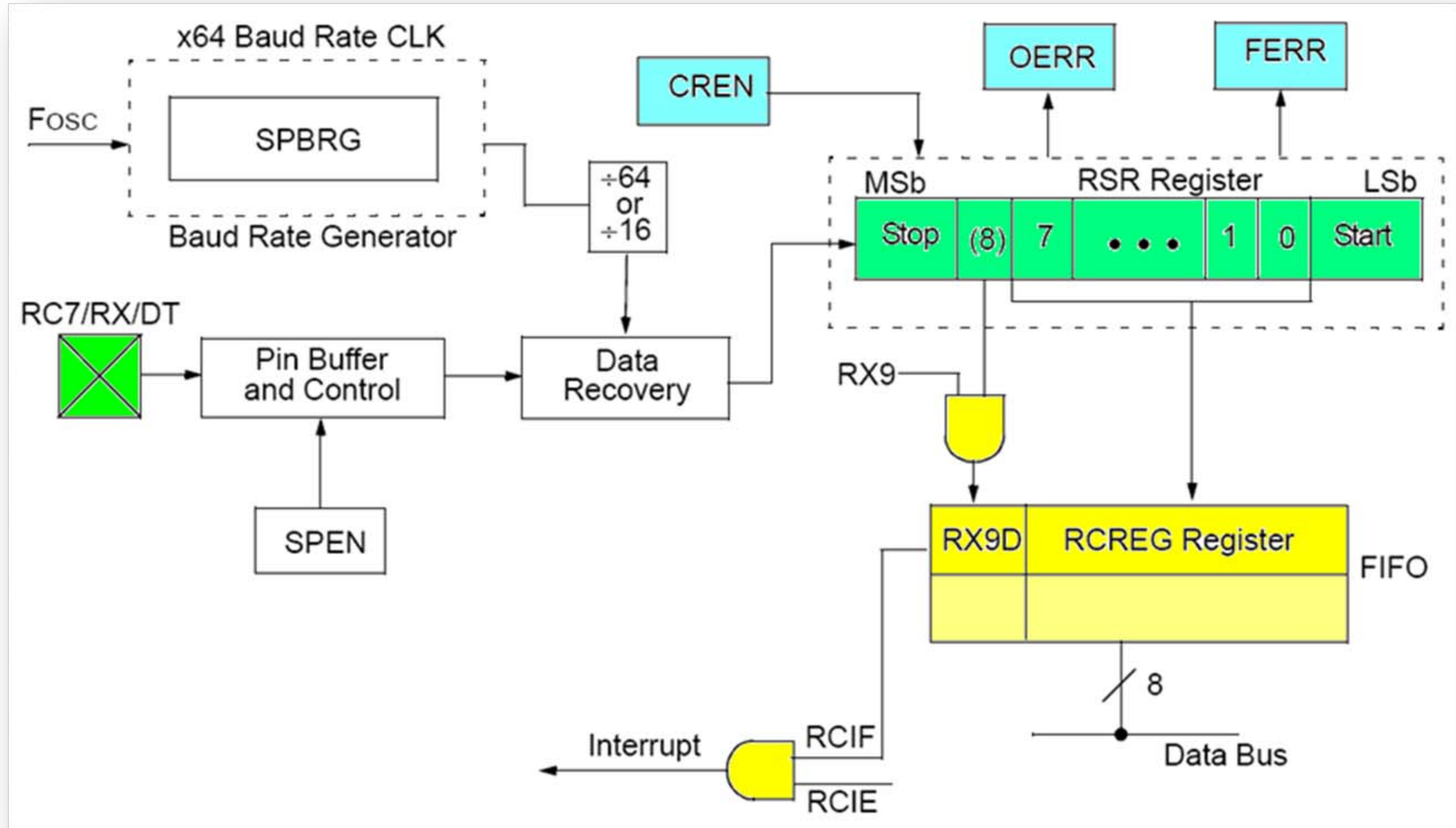


- **Registers involved in asynchronous transmission**

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

The 16F87xA USART

- **Asynchronous Receiver**



The 16F87xA USART

- **Asynchronous USART Receiver Operation Notes**
  - Data is received LSB first on RC7 pin
  - Reception is enabled by the CREN bit
  - At the heart of the block is the RSR register. Once a stop bit is detected, data is transferred to RCREG register, if it is empty, and the RCIF flag is set. (RCIF is cleared by hardware and it is read-only). On-receive interrupt can be enabled by RCIE bit
  - The RCREG is FIFO double buffered register
    - Can be used to receive bytes while reception continues in RSR
    - It can be read twice to read the received two bytes
    - If a stop bit is detected in RSR and the RCREG is still full, an overrun error occurs and it is indicated in OERR bit (The word is RSR is lost)
    - If OERR bit is set, shifting stops in RSR and transfers to the RCREG is inhibited !
    - To clear the overrun error, clear the CREN bit.
  - If the stop bit is received as clear in RSR a framing error occurs and it is indicated by the FERR bit.
  - The 9th bit of data RX9D and FERR are also double buffered. It is essential to read the RCSTA register before the RCREG to avoid losing the corresponding values of RX9D and FERR

The 16F87xA USART

RCSTA (18H)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R-0	R-0	R-0
SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN: Serial Port Enable bit**
1 = Serial port enabled (Configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit
Asynchronous mode
Don't care

Synchronous mode - master
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.

Synchronous mode - slave
Unused in this mode

bit 4 **CREN**: Continuous Receive Enable bit
Asynchronous mode
1 = Enables continuous receive
0 = Disables continuous receive

Synchronous mode
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **Unimplemented:** Read as '0'

bit 2 **FERR**: Framing Error bit
1 = Framing error (Can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit
1 = Overrun error (Can be cleared by clearing bit CREN)
0 = No overrun error

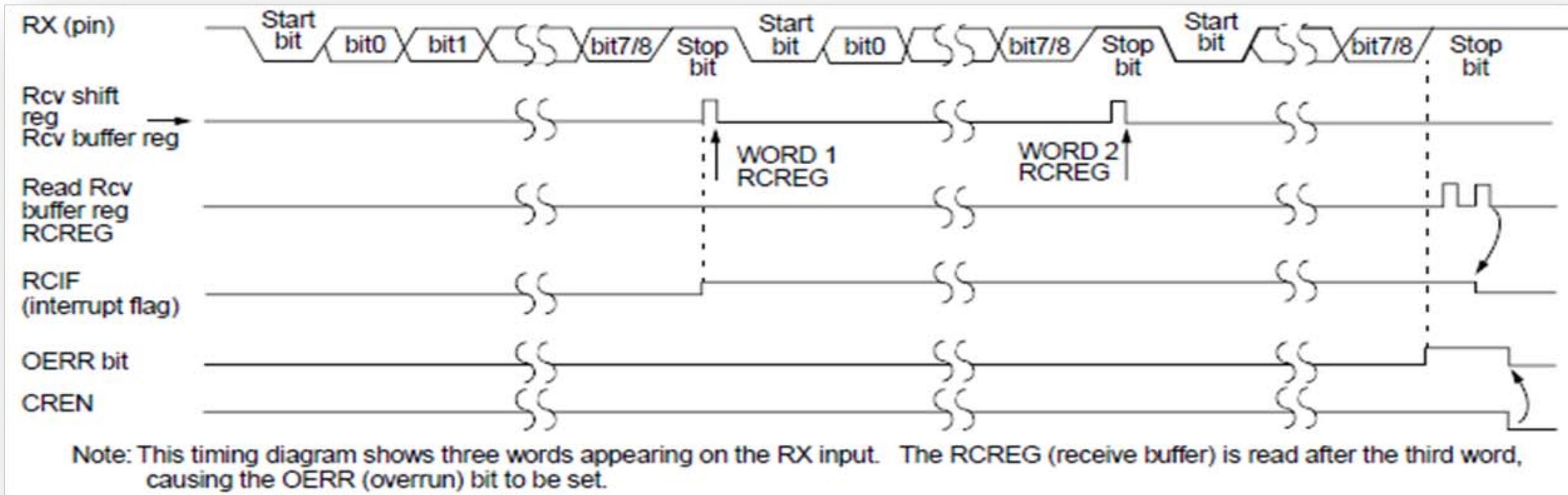
bit 0 **RX9D:** 9th bit of received data, can be parity bit.

The 16F87xA USART

- **Steps for Using the asynchronous receiver**
  1. Set the SPBRG (0x99) register and BRGH (TXSTA<2>) bit to choose the appropriate baud rate
  2. Enable asynchronous serial port by clearing the SYNC (TXSTA<4>) bit and setting the SPEN bit (RCTSA<7>)
  3. If interrupts are desired, set the RCIE (PIE1<5>), GIE (INTCON<7>), and PEIE (INTCON<6>) bits
  4. If 9-bit reception is desired, set the RX9 (RCSTA<6>) bit
  5. Enable the reception by setting bit CREN (RCSTA<4>)
  6. The RCIF (PIR1<5>) will be set when reception of one word is complete and an interrupt will be generated if RCIE is set
  7. Read the RCSTA (0x18) to get the 9th bit and determine if any error occurred (OERR, FERR)
  8. Read the 8-bit received data by reading RCREG (0x1A)
  9. If any error occurred, clear the error by clearing the CREN

The 16F87xA USART

- **Timing of asynchronous reception**



- **Registers involved in asynchronous reception**

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

The 16F87xA USART

- **The BAUD Rate Generator**
  - The BAUD rate for USART is controlled by the value in the SPREG (99H), the SYNC and the BRGH bits in the TXSTA (19H)

SYNC	BRGH = 0	BRGH = 1
0 (asynchronous)	$\frac{F_{osc}}{64(SPBRG+1)}$	$\frac{F_{osc}}{16(SPBRG+1)}$
1 (synchronous)	$\frac{F_{osc}}{4(SPBRG+1)}$	

Example 1

A program to transmit 3 bytes stored in locations 0x40, 0x41, and 0x42 serially with no parity at a rate of 9.6 Kbps. Assume PIC 16F877A with oscillator frequency of 20 MHz

Requirements

1. setup the serial port for transmission
2. choose the appropriate value of SPBRG and BRGH to produce the required rate

Example

```
          #include  p16F877A.inc     ; include the definition file for 16F77A
          org       0x0000           ; reset vector
          goto      START
          org       0x0004           ; define the ISR
ISR       goto      ISR
          org       0x0006           ; Program starts here
START     bsf       STATUS , RP0
          bcf       STATUS , RP1     ; select bank 1
          bcf       TRISC, 6         ; set RC6 as output
          movlw     D'31'
          movwf     SPBRG            ; set the SPBRG value
          bsf       TXSTA, TXEN
          bcf       STATUS, RP0      ; select bank0
          bsf       RCSTA, SPEN      ; enable serial transmission
          movlw     0x40
          mowf      FSR              ; FSR has the address of the first element
```

Example

```
TX      movf    INDF, W       ; read byte to transmit
        movwf   TXREG         ; store in the transmission register
        incf    FSR, F        ; increment FSR to point to next address
WAIT    btfss   PIR1, TXIF    ; check if the TXREG is empty
        goto    WAIT
        movf    FSR,W
        sublw   0x43
        btfss   STATUS, Z     ; check if all values were transmitted
        goto    TX

DONE    goto    DONE
        end
```

Summary

- Serial communication transmits bits one after another in two modes: synchronous and asynchronous
- Stable and accurate clocking plays an important role in serial communication
- It is cheaper to use serial communication over long distances
- Some members of the 16 series are equipped with synchronous and asynchronous communication ports
- These ports can be configured to operated in different modes and rates

Data Acquisition and Manipulation

Chapter 11
Sections 1 - 3

Dr. Iyad Jafar

Outline

- Analog and Digital Quantities
- The Analog to Digital Converter
- Features of Analog to Digital Converter
- The Data Acquisition System
- The 16F873 ADC
- Summary

Analog and Digital Quantities

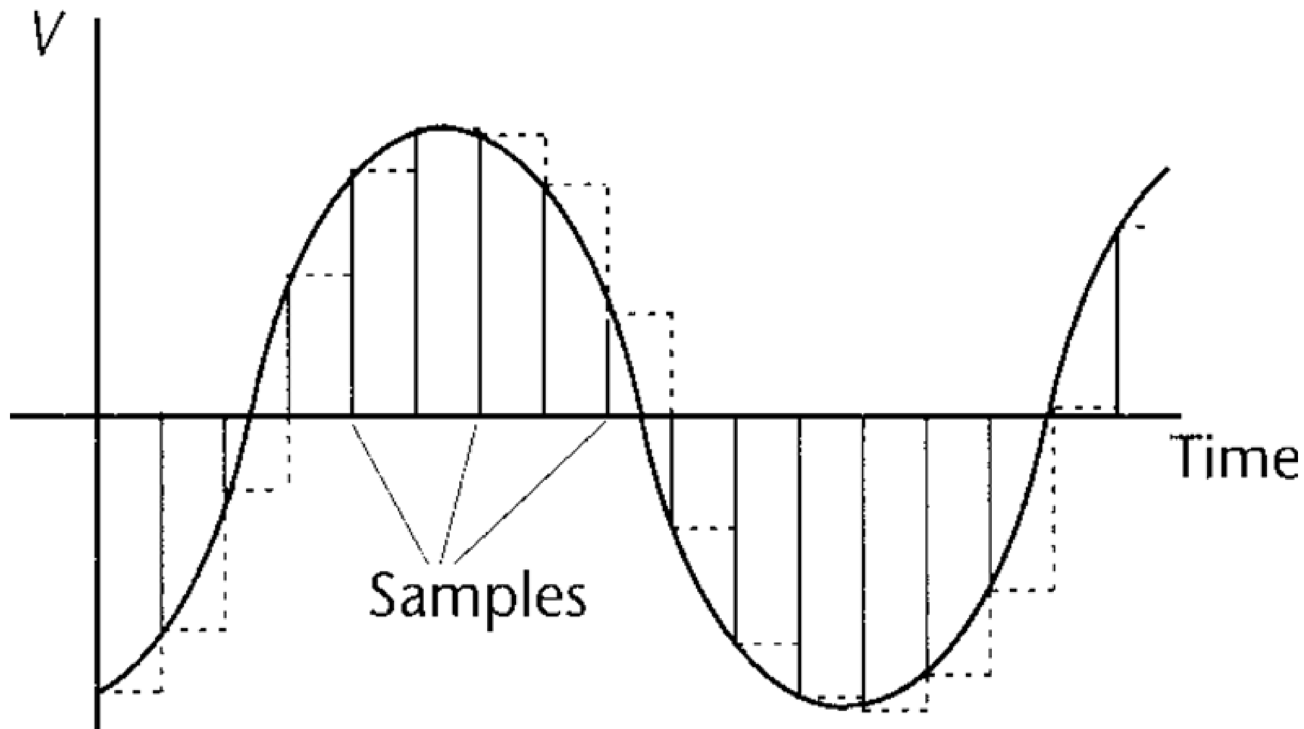
- Most signals that are produced by transducers are analog; continuously variable in time and can take infinite range of values
- Digital signals are *discrete representation* for the analog signals *in time and value*
- Digital signals perform better and are easier to work with
- Analog signals have to be converted into digital form in order to be processed by the microcontroller
- The device that performs this conversion is called Analog to Digital Converter (ADC)

Analog and Digital Quantities

Property	Analog	Digital
Representation	Continuous voltage or current	Binary Number
Precision	Infinite range of values	Only fixed number of digits combination are available
Resistance to Degradation	Suffers from drift, attenuation, distortion, interference. Recovery is hard	Tolerant to most forms of signal degradation. Error checking can be included for complete recovery
Processing	Processing using op amps and other sophisticated circuits. Limited, complex, and suffers from distortion	Powerful computer-based techniques
Storage	Analog storage for any length of time is almost impossible	All semiconductor memory techniques are digital

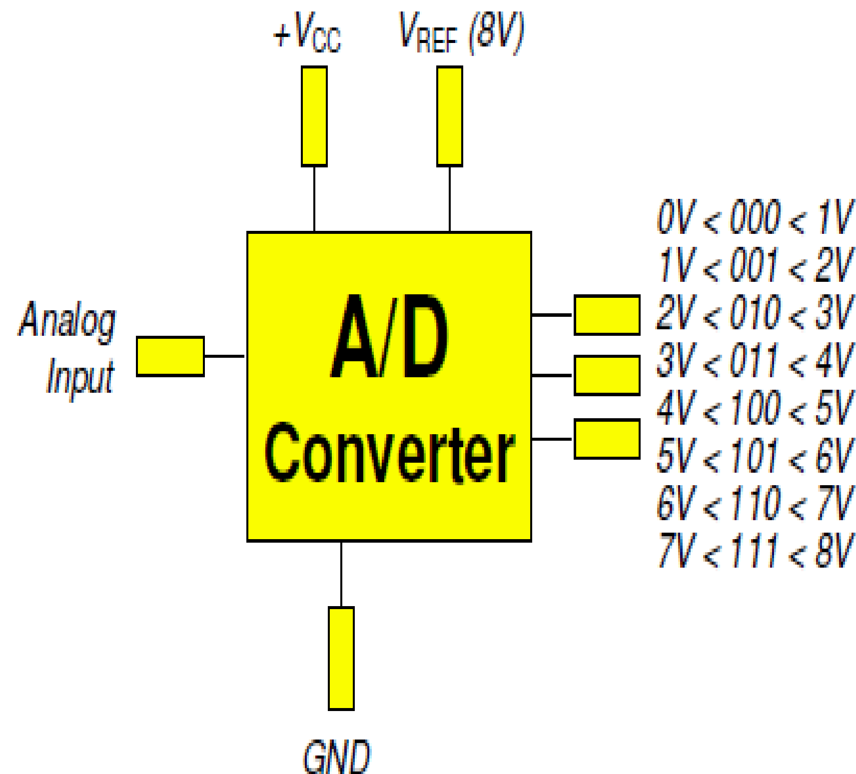
The Analog to Digital Converter

- Conversion to digital form requires two steps
  - Sampling
  - Quantization



Features of Analog to Digital Converter

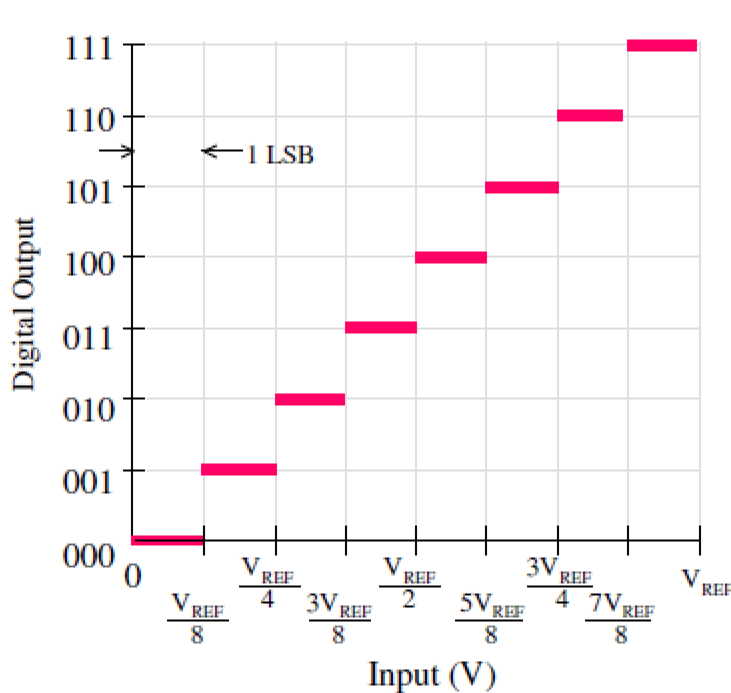
- **Conversion Characteristics**
  - The ADC accepts a voltage that is infinitely variable and converts it to one of a fixed number of output values

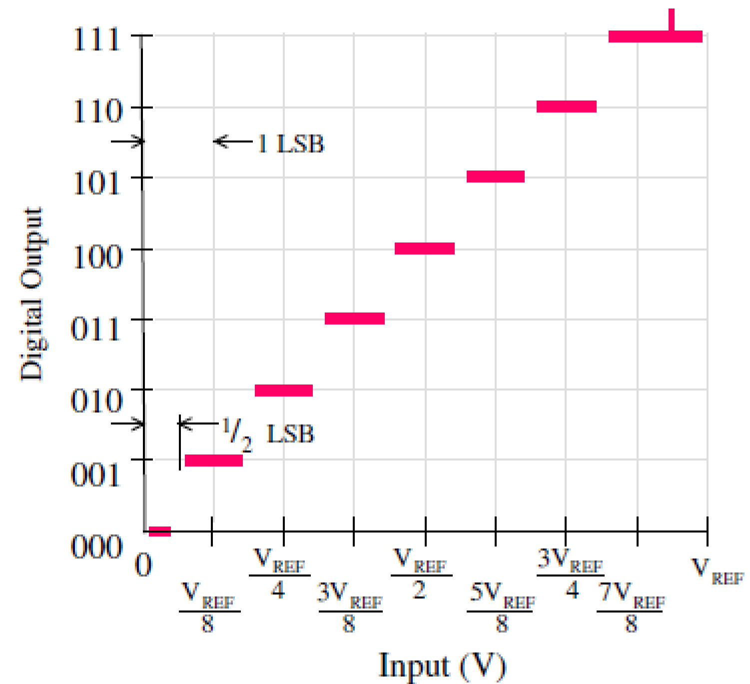


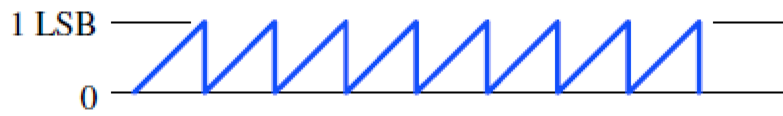
Features of Analog to Digital Converter

- **Conversion Characteristics**

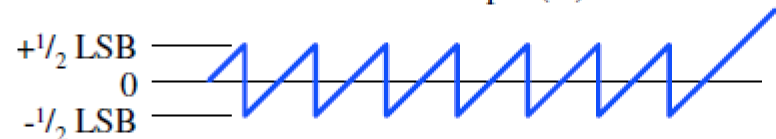
Quantization Error







The Magnitude of the Error Ranges from Zero to 1 LSB



Features of Analog to Digital Converter

- **Reference voltages [V_{min},V_{max}]**
  - Determine the acceptable range of input analog voltage
  - Out of range input values are clipped
  - *Unipolar or bipolar*
  - Should be stable and accurate for proper operation
  - ***Input range $V_r = V_{max} - V_{min}$***

- **Resolution**
  - The amount by which the input voltage has to change to go from one output value to another
  - The more the output bits the more the output steps and finer is the conversion
  - ***Resolution = $V_r / 2^n$***
  - ***Quantization error Q = resolution / 2***

Features of Analog to Digital Converter

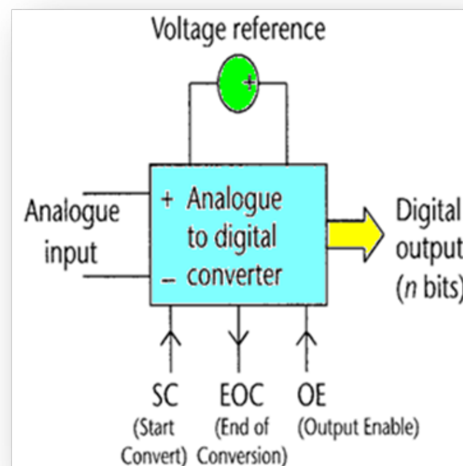
- **Conversion Characteristic**

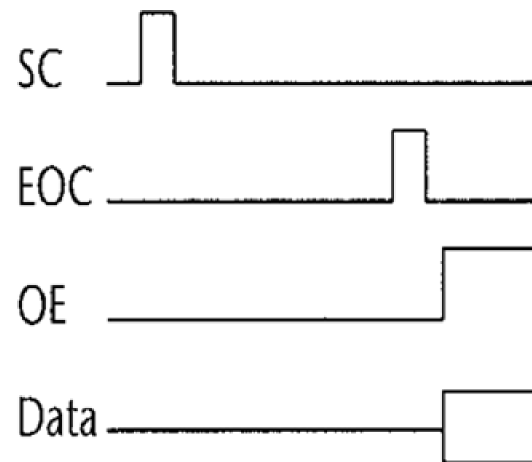
Quantization error as a function of ADC bits

n	No. of quantisation levels	Max. quantisation error as % of range	Quantisation error for range of 5 V
3	8	6.25	312.50 mV
4	16	3.13	156.25 mV
5	32	1.56	78.13 mV
6	64	0.781	39.06 mV
8	256	0.195	9.77 mV
10	1 024	0.0488	2.44 mV
12	4 096	0.0122	0.61 mV
16	65 536	0.00076	38.1 µV

Features of Analog to Digital Converter

- **Conversion Speed**
  - Time for the ADC to do the conversion
  - Slow ADCs are used with low frequency signals
  - High accuracy ADCs take longer to complete conversion
- **Digital Interface**
  - Made up of control signals and data outputs
  - Data outputs – serial or parallel



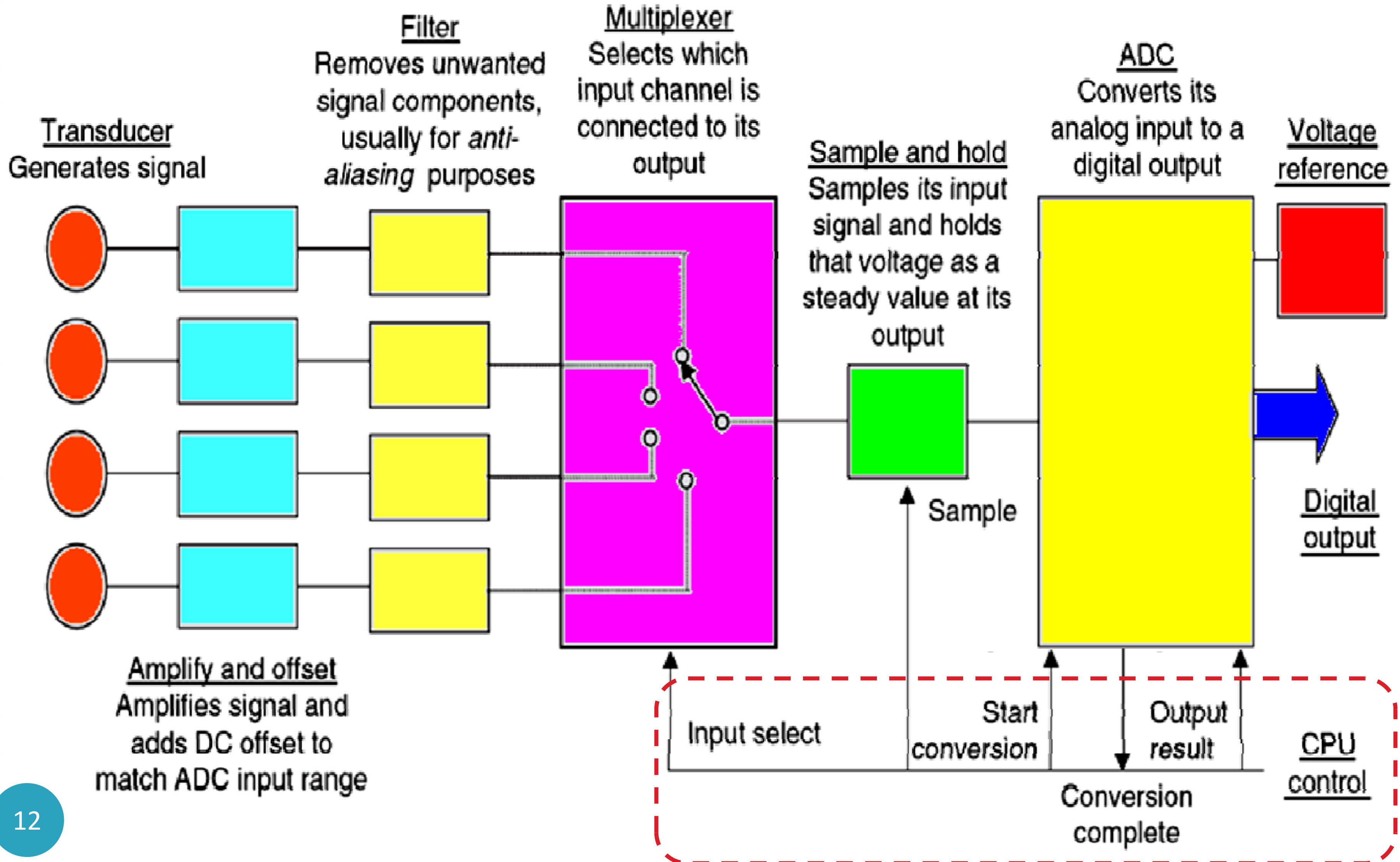


The Analog to Digital Converter

- **ADC Types**
  - *Dual Ramp ADC*
    - Slow but with high accuracy
  - *Flash Converter ADC*
    - Fast but less accuracy
    - Used with high speed signals such as video and radar
  - *Successive Approximation ADC*
    - Medium speed and accuracy
    - Used in general-purpose industrial applications
    - *Commonly found in embedded systems*

The Data Acquisition System

Elements of data acquisition system



The Data Acquisition System

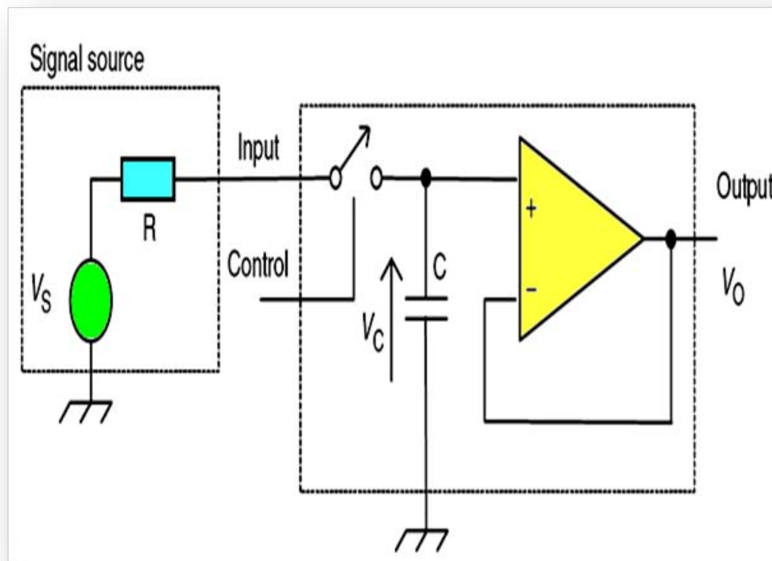
Elements of data acquisition system

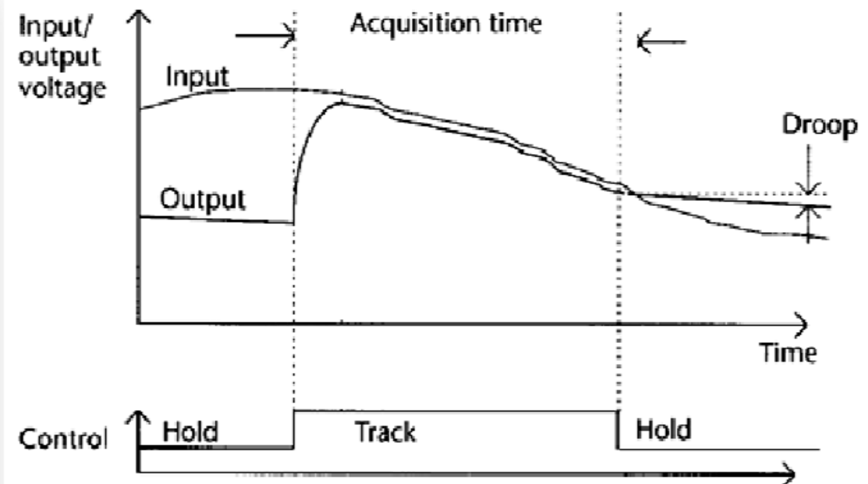
- **Amplification**
  - Most sensors produce low voltages
  - Need to amplify to exploit the input range of the ADC
  - Voltage level shifting might be needed for bipolar signals
- **Filtering**
  - Pick the actual signal and restrict its frequency content to the sampling rate of the ADC to avoid aliasing
  - Remove unwanted signals
- **Analog multiplexer**
  - Used when working with multiple inputs instead of using multiple ADCs
  - Semiconductor switches

The Data Acquisition System

Elements of data acquisition system

- **Sample and Hold**
  - ADCs are unable to convert accurately a changing signal
  - We need to capture the sample value and hold it for the duration of the conversion process
  - Acquisition time !

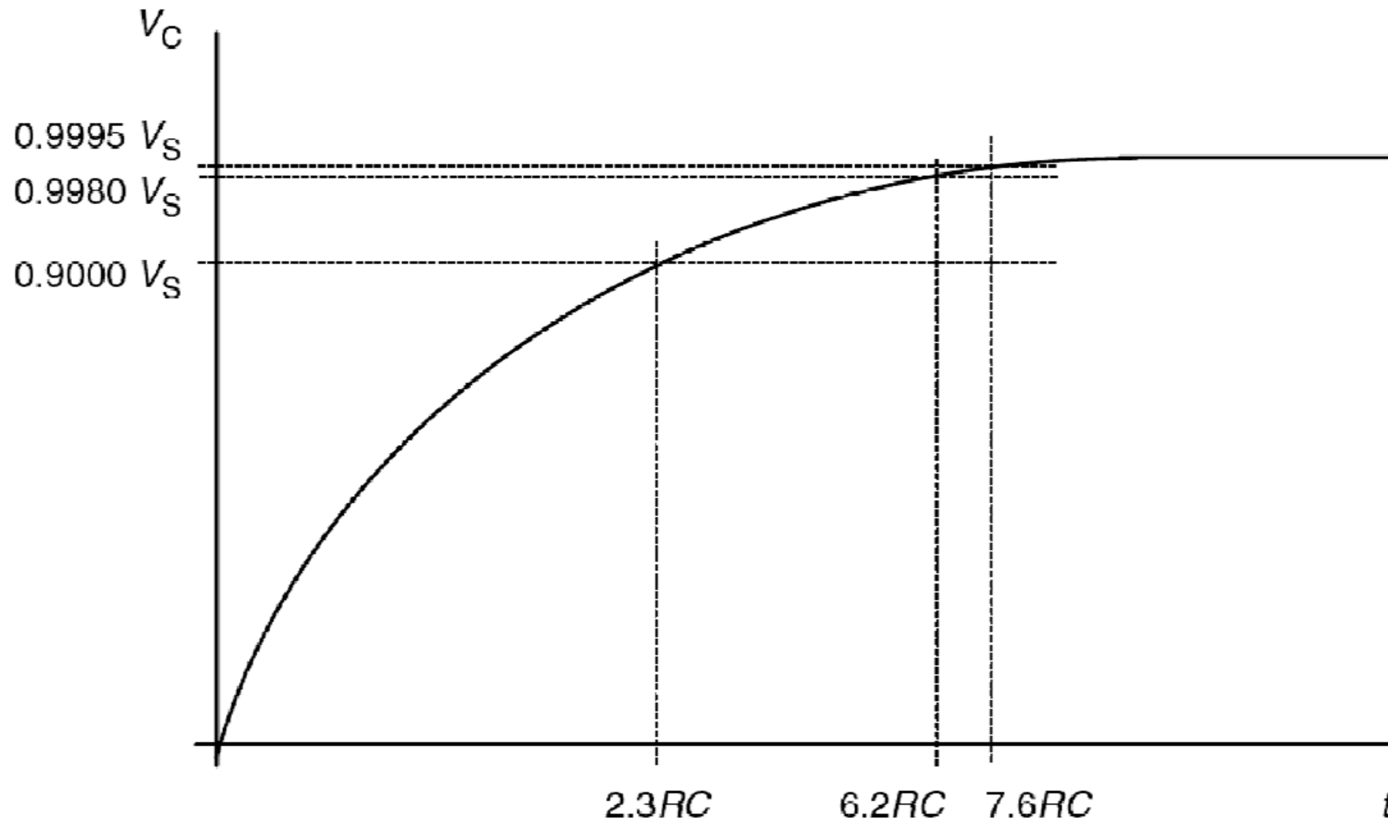




The Data Acquisition System

Elements of data acquisition system

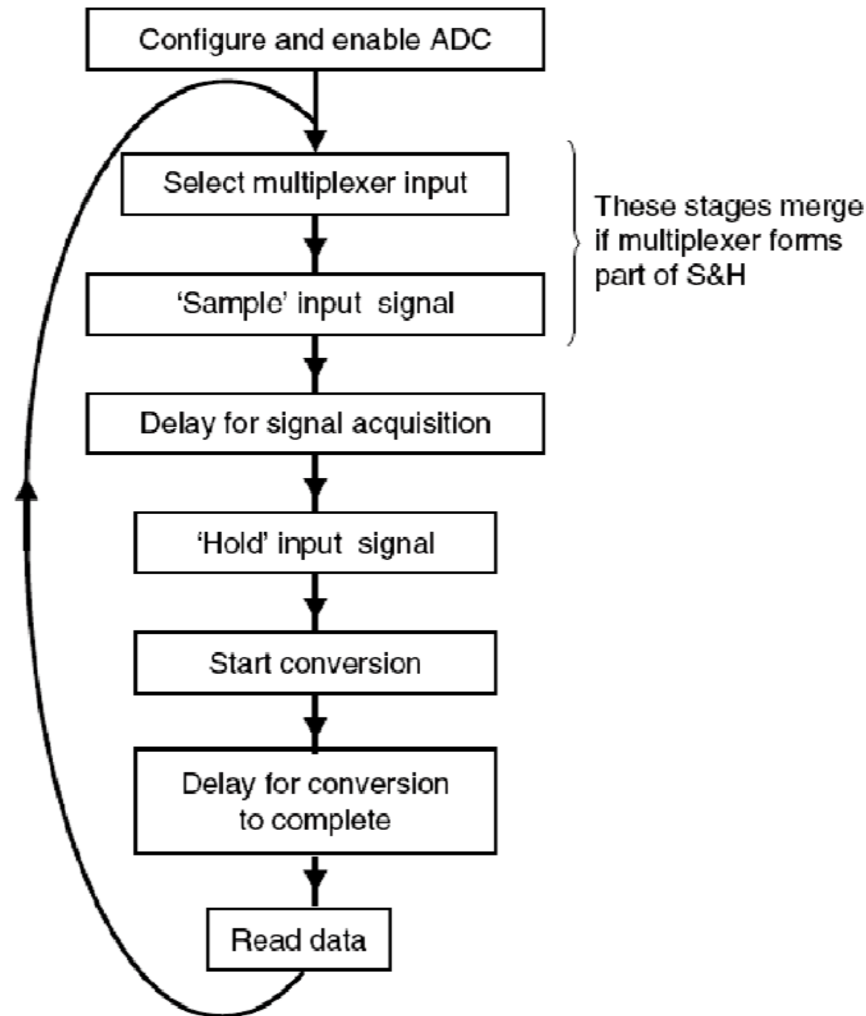
- **Sample and Hold**



Acquisition time increase as we increase the resolution of the ADC

The Data Acquisition System

Typical Timing Requirements for Analog to Digital Conversion



Data Acquisition in Microcontroller Environment

- Embedded systems need ADCs ; usually they are integrated within the MC as 8 or 10 bit ADCs

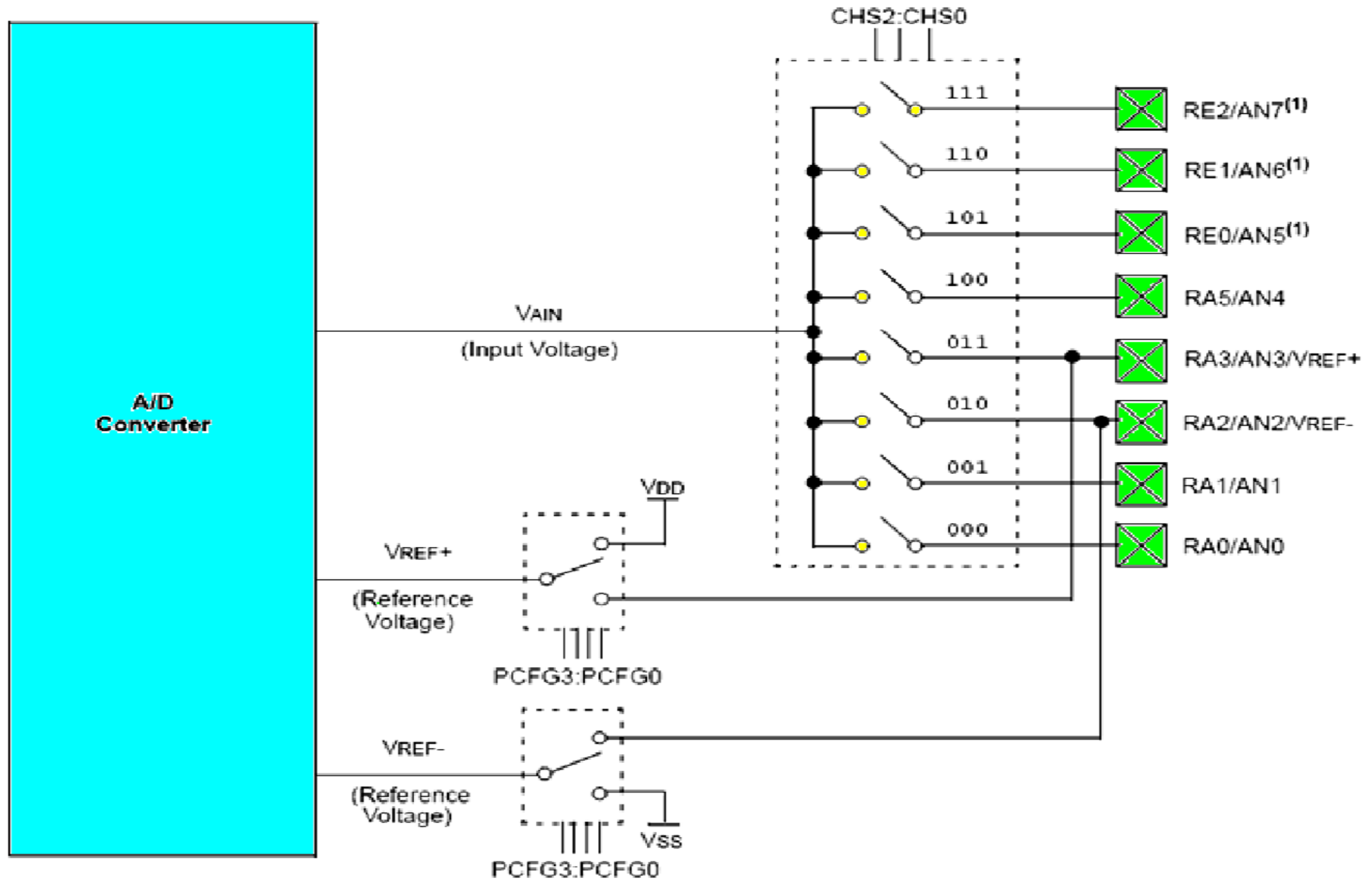
- Integration is not easy !
  - Proper operation of ADCs demands clean power supply and ground and freedom of interference
  - This is not easily available in digital devices

- Compromise accuracy of integrated ADCs !

The PIC 16F87xA ADC Module

Device	Pins	Features
16F873A 16F876A	28	3 parallel ports, 3 counter/timers, 2 capture/compare/PWM, 2 serial, **5 10-bit ADC,** 2 comparators
16F874A 16F877A	40	5 parallel ports, 3 counter/timers, 2 capture/compare/PWM, 2 serial, **8 10-bit ADC,** 2 comparators

The PIC 16F87xA ADC Module



The PIC 16F87xA ADC Module

Related Registers

- Operation is controlled by two SFRs
  - **ADCON0 0x1F**
  - **ADCON1 0x9F**
- Conversion result (10-bit) is placed in two SFRs
  - ADRESL 0x9E
  - ADRESH 0x1E
- ADC interrupt enable and flag are available in
  - PIE1 0x8C
  - PIR1 0x0C
- Related registers
  - TRISA 0x85
  - TRISE 0x89 (in 40-pin devices)

The PIC 16F87xA ADC Module

Controlling the ADC

(1) Switching on

- The ADC is switched on/off by setting/clearing ADON bit (ADCON0<0>)
- It is preferred to turn the ADC off when it is not needed as it offers some power saving

(2) Setting Conversion Speed

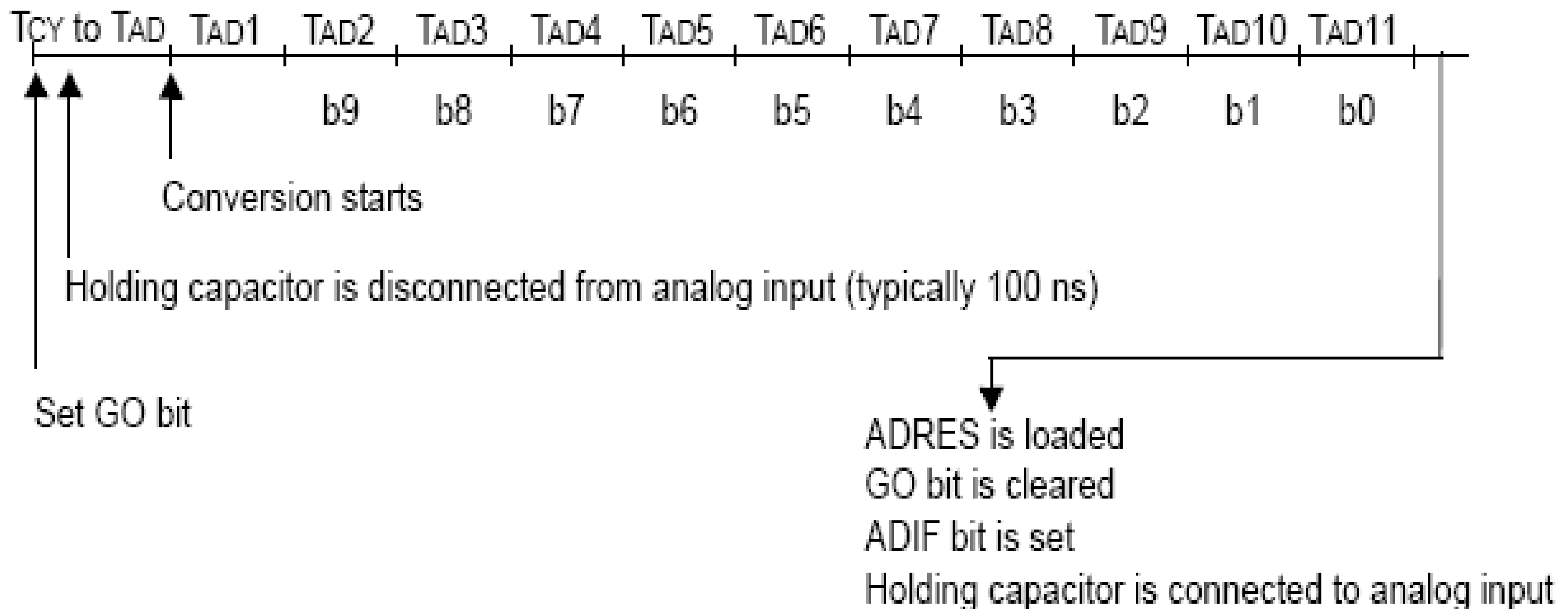
- Operation of the ADC is governed by a clock with period T_{AD}
- For correct conversions, T_{AD} must be 1.6 us at least
- The ADC clock can be selected by software ($2T_{OSC}$, $4\ T_{OSC}$, $8\ T_{OSC}$, $16\ T_{OSC}$, $32\ T_{OSC}$, $64\ T_{OSC}$, or internal RC 2-4 us)
- Selection of ADC clock source is through ADCS2 (ADCON1<6>), ADCS1:ADCS0 (ADCON0<7:6>)
- If the system clock is fast (>500KHz), use it to derive the ADC clock. Otherwise, use the internal RC.

The PIC 16F87xA ADC Module

Controlling the ADC

Setting Conversion Speed

- A full 10-bit conversion requires 12 T_{AD}



The PIC 16F87xA ADC Module

Controlling the ADC

(3) Configuring Inputs and Voltage Reference

- The ADCON1 and TRIS registers control the operation of the A/D port pins
- Inputs AN7 to AN0 can be configured as analog inputs or digital inputs.
- AN3 **(RA3)** and AN2 **(RA2)** can be used as the inputs for the external reference voltages separately
- Configuration is made through PCFG3:PCFG0 (ADCON1<3:0>)

(4) Channel Selection

- We can select one out of five (or eight channels) as the analog input using the bits CHS2:CHS0 (ADCON0<5:3>)
- Selection of the input channel closes the sampling switch.

The PIC 16F87xA ADC Module

Controlling the ADC

(5) Starting Conversion and Flagging its End

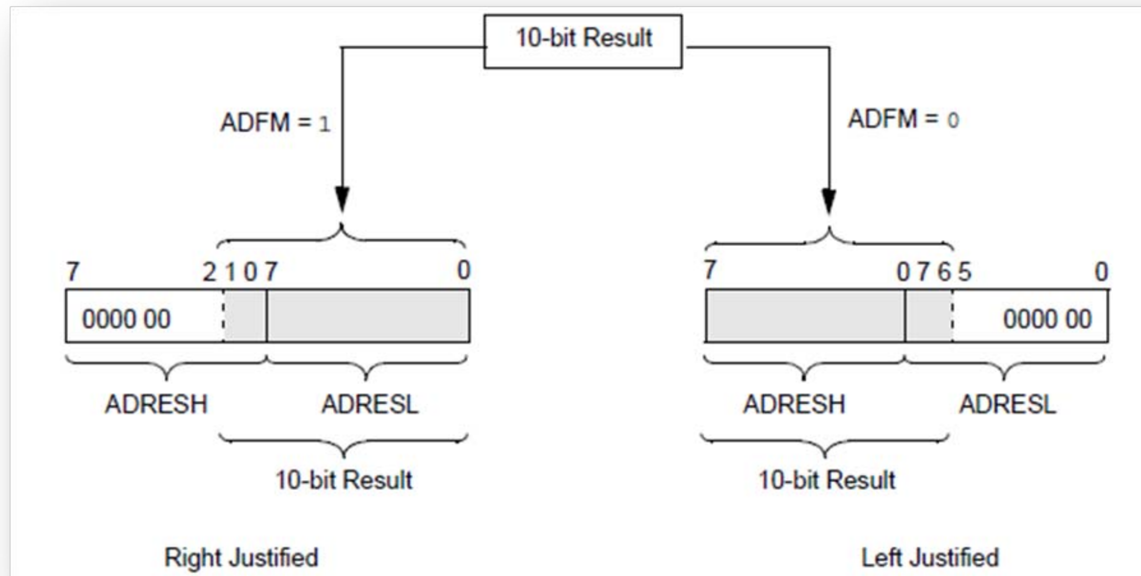
- Conversion can be started by setting the GO/DONE' (ADCON0<2>) bit. **This opens the sampling switch**.
- Once the conversion is complete, this bit is cleared to indicate the end of conversion
- *The GO/DONE' bit should not be set using the same instruction that turns on the A/D.*

The PIC 16F87xA ADC Module

Controlling the ADC

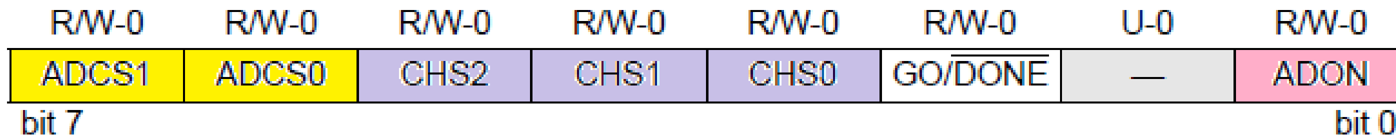
(6) Formatting the result

- The ADC result is 10-bit data that is placed in ADRESH and ADCRESL (0x 1E and 0x9E respectively)
- The result can be left justified or right justified
- Selection of desired format is through the ADFM (ADCON1<7>) bit



The PIC 16F87xA ADC Module

ADCON0 Register 0x1F



bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	FOSC/4
1	01	FOSC/16
1	10	FOSC/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits
000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

bit 2 **GO/$\overline{DONE}$:** A/D Conversion Status bit
When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit
1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

The PIC 16F87xA ADC Module

ADCON1 Register 0x9F

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit

1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.

0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O

C/R = # of analog input channels/# of A/D voltage references

The PIC 16F87xA ADC Module

Related Registers

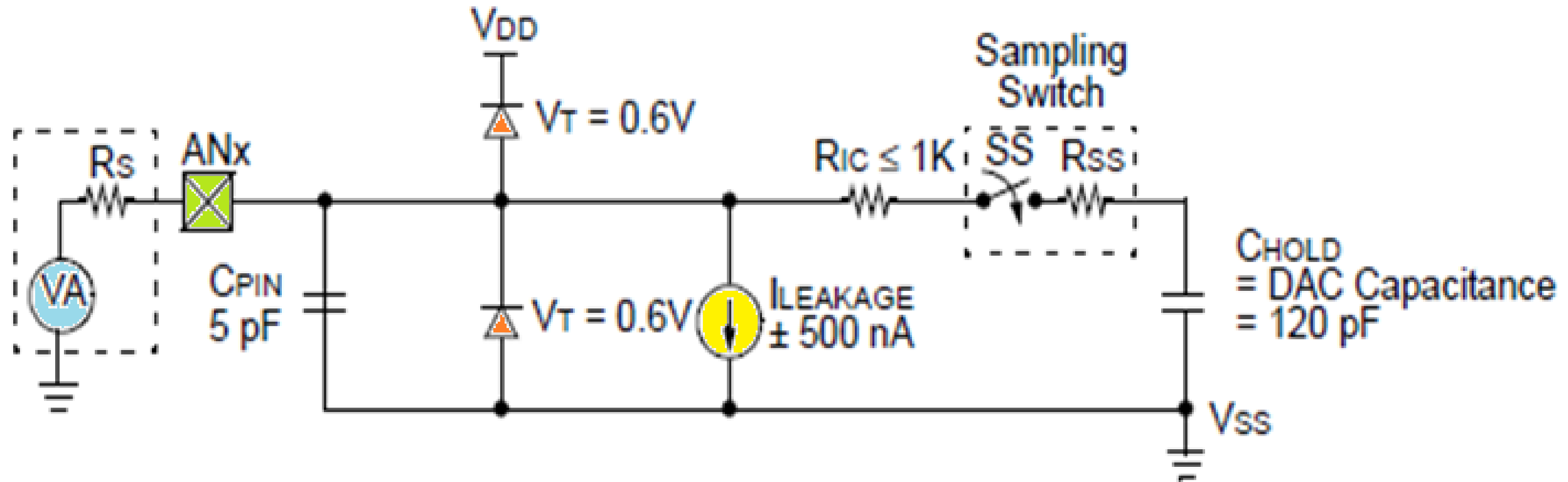
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on $\overline{MCLR}$, WDT
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	uuuu uuuu
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0	0000 00-0
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	--0u 0000
89h[1]	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
09h[1]	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu

The PIC 16F87xA ADC Module

- **Steps for using the A/D module**
  1. Configure the A/D module
     a. Select analog pins/voltage reference and digital I/O (ADCON1)
     b. Select the A/D channel (ADCON0)
     c. Select the conversion clock (ADCON0)
     d. Turn the A/D module on (ADCON0)
  2. Configure interrupts (if desired)
     1. Clear ADIF (PIR1<6>) and set ADIE (PIE1<6>)
     2. Set PEIE (INTCON<6>) then set GIE (INTCON<7>)
  3. Wait the required acquisition time
  4. Start conversion by setting the GO/DONE’ bit
  5. Wait for conversion complete
  6. Read the A/D result register pair ADRESH:ADRESL

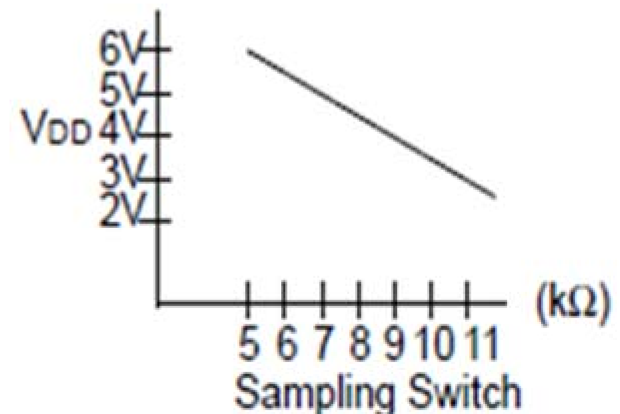
The PIC 16F87xA ADC Module

- **The analog input model**



Legend:

CPIN	= input capacitance
VT	= threshold voltage
ILEAKAGE	= leakage current at the pin due to various junctions
RIC	= interconnect resistance
SS	= sampling switch
CHOLD	= sample/hold capacitance (from DAC)



The PIC 16F87xA ADC Module

- **Calculating conversion speed (Qerror is ½ LSB)**

A/D Total Time = Acquisition Time + A/D Conversion time

= T_{ACQ} + 12 * T_{AD}

TACQ = Amplifier settling time

+ Hold capacitor charging time

+ Temperature coefficient

TACQ = $T_{AMP} + T_{HOLD} + T_{COFF}$

THOLD = $-(R_{IC}+R_{SS}+R_S) * C_{HOLD} * \ln(1/2^{(n+1)})$

= $-(R_{IC}+R_{SS}+R_S) * 120\text{ pF} * \ln(1/2048)$

= 7.6*R*C us

A/D Total Time = 2 µs + 7.6RC + (Temperature − 25°C)(0.05 µs/°C) + 12 T_{AD}

The PIC 16F87xA ADC Module

- **Calculating conversion speed example**

R_{SS} = 7kΩ (V_{DD} = 5V), R_{IC} = 1kΩ, R_S = 0,
Temp = 35 °C, T_{AD} = 1.6 μs

t_{ac} = 2 μs
+ 7.6(7kΩ + 1kΩ + 0)(120pF)
+ (35 – 25)(0.05 μs/°C)
= 2 + 7.3 + 0.5 = 9.8 μs

Total time = t_{ac} + $12T_{AD}$ = 9.8 + 19.2 μs = 29 μs
Maximum sampling rate ~= 34.5 KHz

The PIC 16F87xA ADC Module

- **Repeated Conversions**
  - When a conversion is complete, the converter waits a period of 2*TAD before it is available to start a new conversion
  - This time has to be added to the conversion time !

- **Trading off conversion speed and resolution**
  - If resolution is not an issue, then we can start the conversion with correct clock then we switch it to higher clock
  - Consider only bits produced before switching the clock

The PIC 16F87xA ADC Module

- **Example: use the ADC in PIC 16F877A to obtain one sample of an analog signal that is connected RA0. Assume the ADC clock to be Fosc/8 and reference voltage to be internal. The PIC is operating with Fosc = 4 MHz, VDD = 5 v, and temperature 25 C. The result should be right justified.**

  Setup:

  1) set RA0 as analog input

  2) select the clock

  3) generate appropriate delays (T_{acq} = 2+ 7.6 * (1K + 7K) * 120 pF = 9.3 us ~= 10 us)

Example

```
        #include  p16F877A.inc   ; include the definition file for 16F77A
        org       0x0000         ; reset vector
        goto      START
        org       0x0004         ; define the ISR
ISR     goto      ISR
        org       0x0006         ; Program starts here
START   bsf       STATUS, RP0    ; select bank 1
        movlw     B'00000001'
        movwf     TRISA          ; set RA0 as input
        movlw     B'10001110'    ; select RA0 as analog input, result right
                                 ; justified, and internal reference voltage
        movwf     ADCON1
        bcf       STATUS, RP0    ; select bank 0
        movlw     B'01000001'    ; turn on ADC, clock Fosc/8, select
                                 ; channel 0
        movwf     ADCON0
```

Example

```
            ; start the conversion
            call    delay10us           ; acquisition time delay
            bsf     ADCON0, GO          ; start conversion
            btfsc   ADCON0, GO_DONE ; wait for conversion to complete
            goto    $-1

DONE        goto    DONE

delay10us   movlw   D'2'
            movwf   0x20        ; counter for delay loop
more        nop
            decfsz  0x20,1
            goto    more
            return

            end
```

Summary

- Most signals produced by transducers are analog in nature, while all processing done by a microcontroller is digital.

- Analog signals can be converted to digital form using an analog-to-digital converter (ADC).

- The 16F873A has a 10-bit configurable ADC module

- Data values, once acquired, are likely to need further processing, including offsetting, scaling and code conversion.

The Human and Physical Interface

Chapter 8
Sections 1 - 9

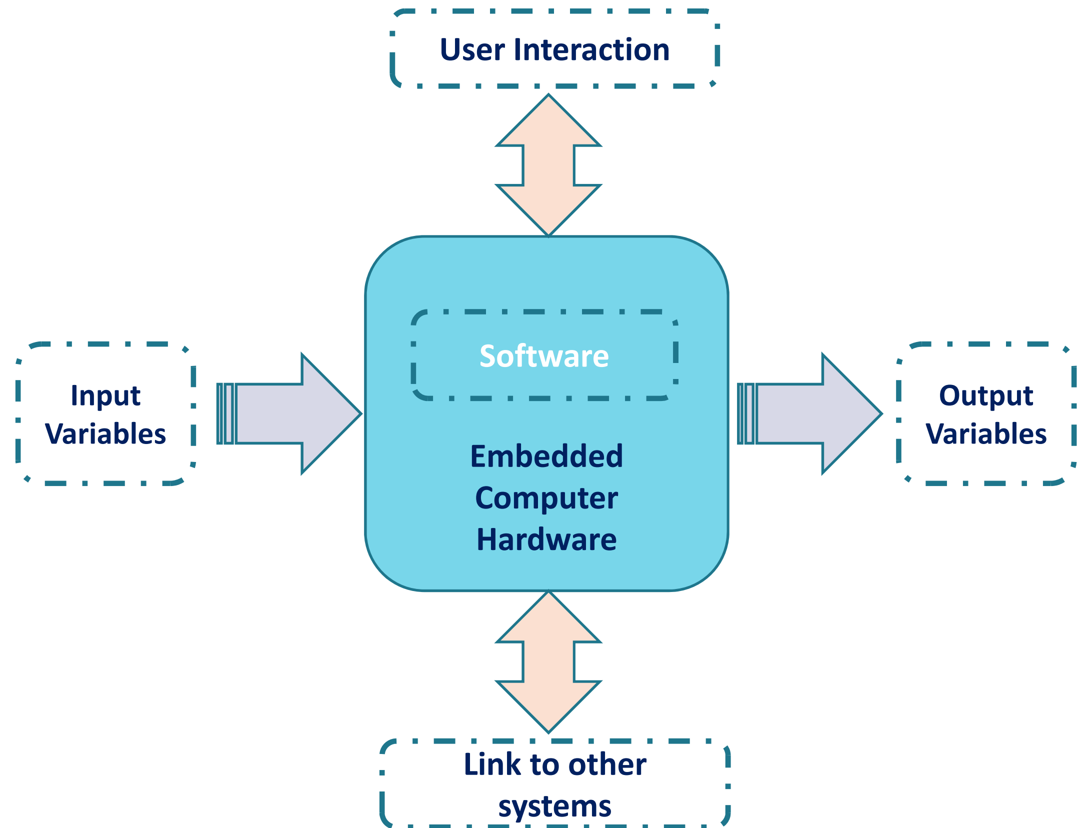
Dr. Iyad Jafar

Outline

- Introduction
- From Switches to Keypads
- LED Displays
- Simple Sensors
- Actuators
- Summary

Introduction

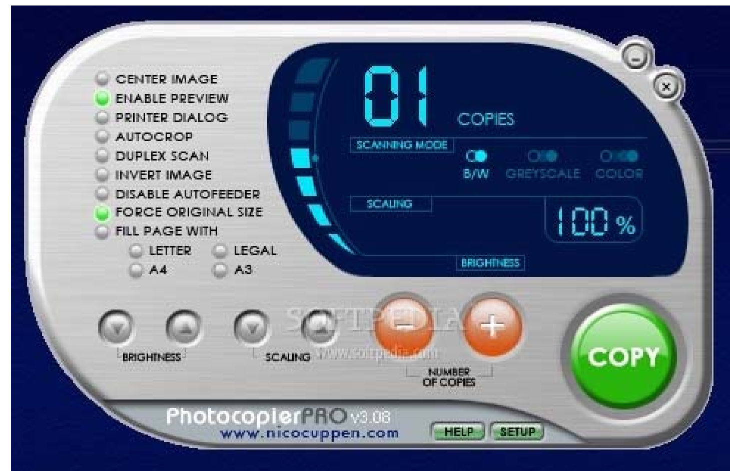
- Humans need to interface with embedded systems ; input data and see response
- Input devices: switches, pushbuttons, keypads, sensors
- Output devices: LEDs, seven-segment displays, liquid crystal displays, motors, actuators



Introduction

- **Examples**

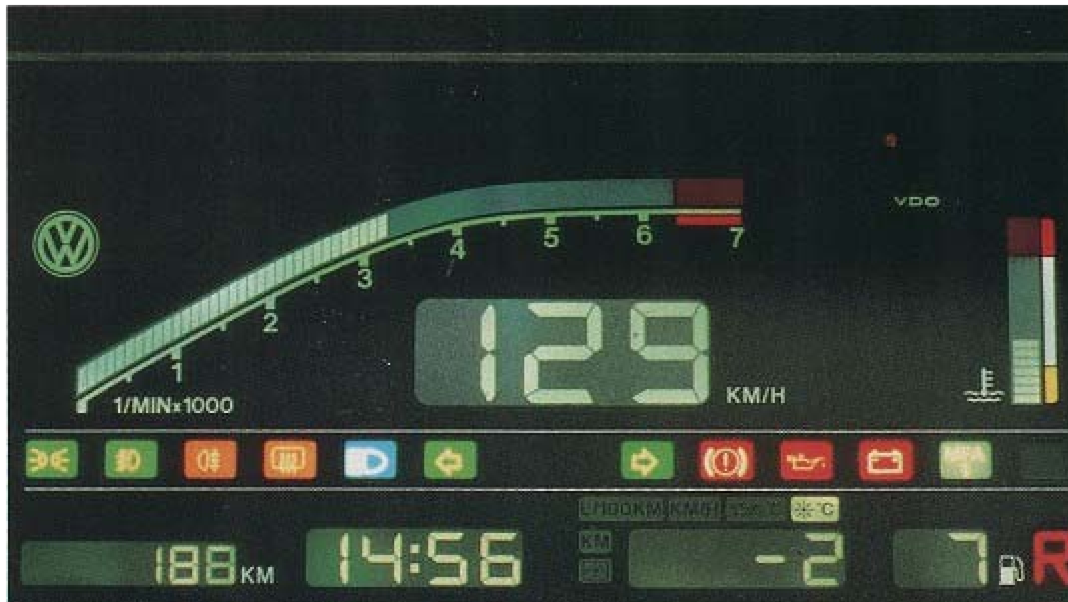
Fridge Control Panel



Photocopier Control Panel

Introduction

- **Examples**



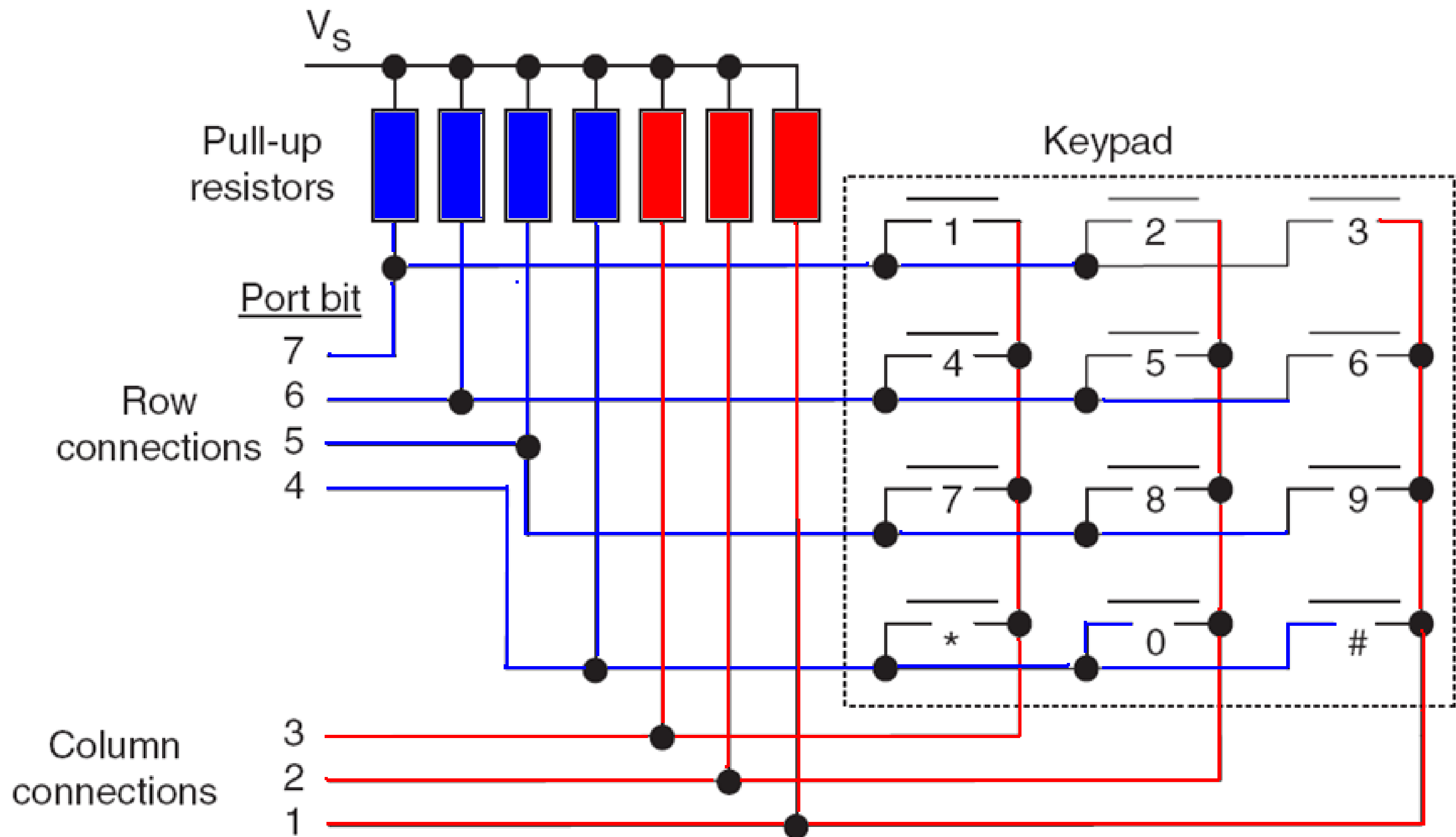
Car Dashboard

Moving From Switches to Keypads

- Switches are good for conveying information of digital nature
- They can be used in multiples; each connected to one port pin
- In complex systems, it might not be feasible to keep adding switches ?!
- Use keypads !
  - Can be used to convey alphanumeric values
  - A group of switches arranged in matrix form

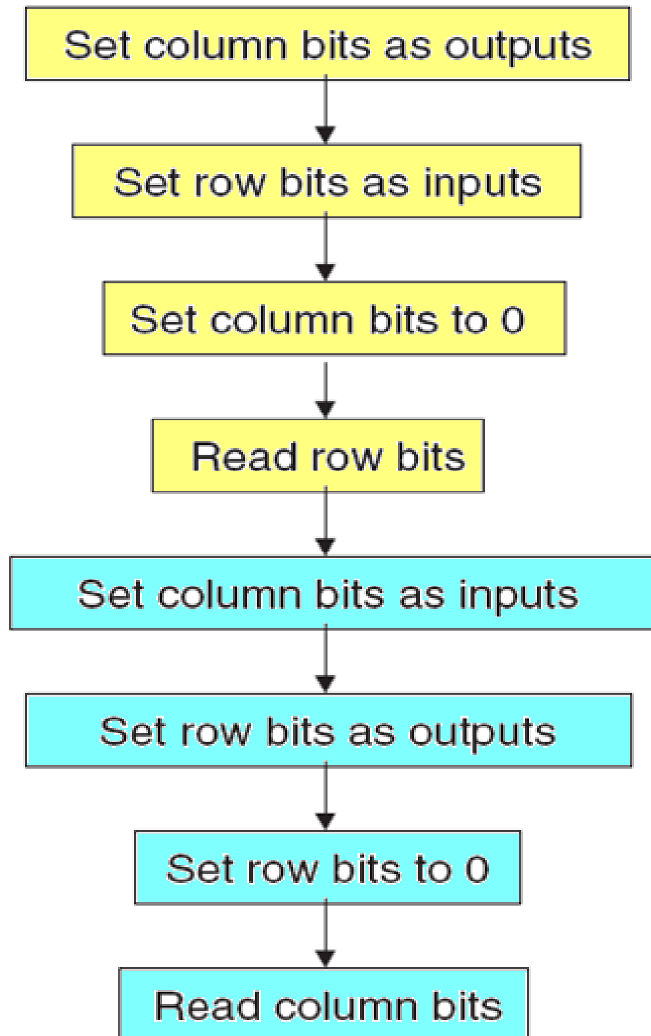
Moving From Switches to Keypads

Internal Structure of Keypad



Moving From Switches to Keypads

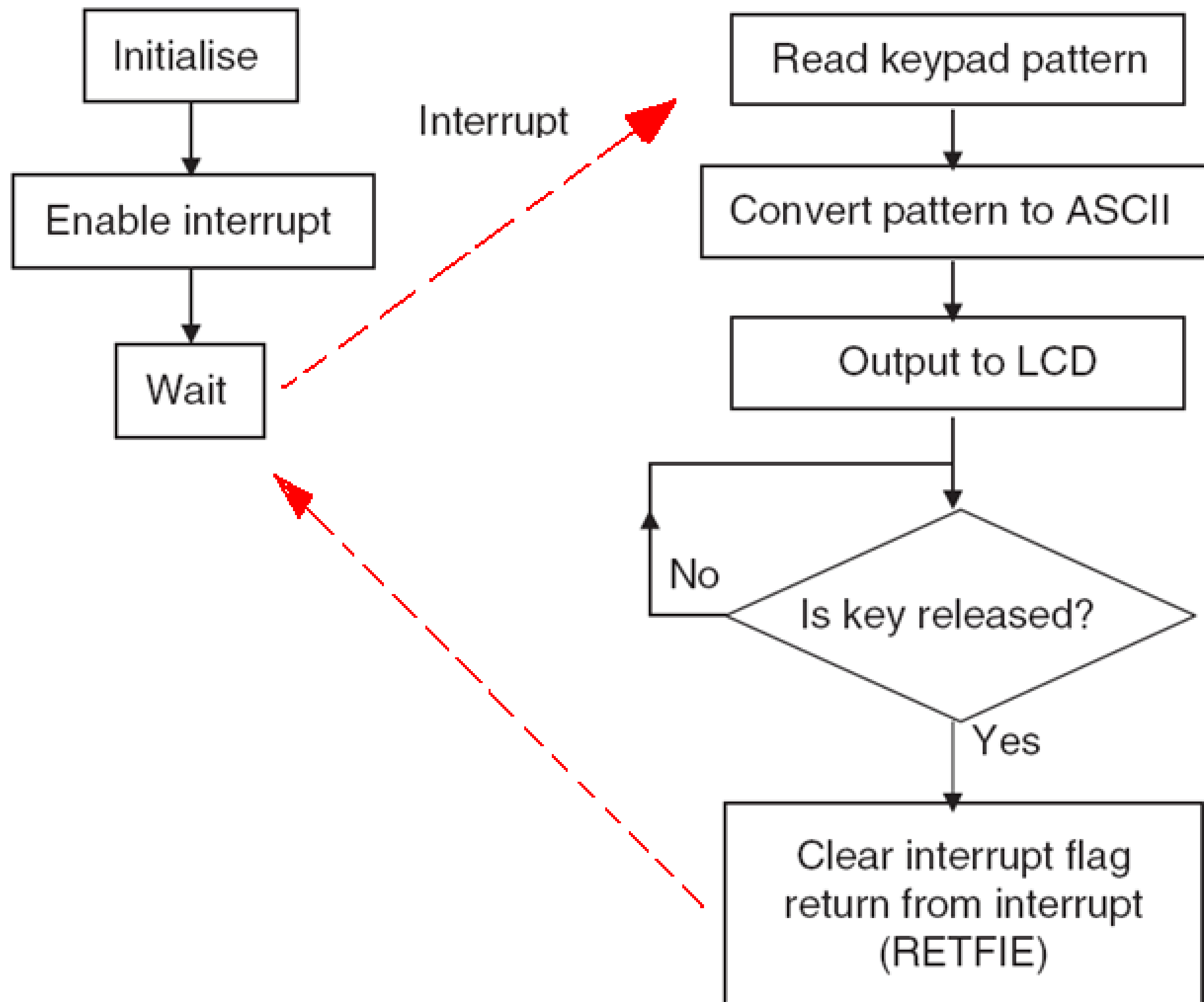
How to Determine the Pressed Key



Key	Value Read
1	0111 011X
2	0111 101X
3	0111 110X
4	1011 011X
5	1011 101X
6	1011 110X
7	1101 011X
8	1101 101X
9	1101 110X
*	1110 011X
0	1110 101X
#	1110 110X

Moving From Switches to Keypads

Using Keypad in a Microcontroller



Moving From Switches to Keypads

Example 1

A program to read an input from a 4x3 keypad and display the equivalent decimal number on 4 LEDs. If the pressed key is not a number, then all LEDs are turned on.

- The keypad will be connected to MC as follows
  - Rows 0 to 3 connected to RB7 to RB4, respectively.
  - Columns 0 to 2 connected to RB3 to RB1, respectively.
- Use **PORTB on-change** interrupt
- Connect the LEDs to RA0-RA3
- Based on the pressed key, convert the row and column values to binary using a lookup table

Keypad Interfacing Example

```
            #include    P16F84A.INC
ROW_INDEX   EQU         0X20
COL_INDEX   EQU         0X21
            ORG         0X0000
            GOTO        START
            ORG         0X0004
            GOTO        ISR
START       BSF         STATUS, RP0
            MOVLW       B'11110000'
            MOVWF       TRISB       ; SET RB1-RB3 AS OUTPUT AND
                                    ; RB4-RB7 AS INPUT
            MOVLW       B'00000000'
            MOVWF       TRISA       ; SET RA0-RA3 AS OUTPUT
            BCF         STATUS, RP0
            CLRF        PORTB       ; INITIALIZE PORTB TO ZERO
            MOVF        PORTB,W     ; CLEAR RBIF FLAG
            BCF         INTCON, RBIF
            BSF         INTCON, RBIE
            BSF         INTCON, GIE ; ENABLE PORT b CHANGE INTERRUPT
LOOP        GOTO        LOOP        ; WAIT FOR PRESSED KEY
```

Keypad Interfacing Example

```
ISR           MOVF      PORTB, W      ; READ ROW NUMBER
              MOVWF     ROW_INDEX
              BSF       STATUS, RP0   ; READ COLUMN NUMBER
              MOVLW     B'00001110'
              MOVWF     TRISB
              BCF       STATUS, RP0
              CLRF      PORTB
              MOVF      PORTB, W
              MOVWF     COL_INDEX
              CALL      CONVERT       ; CONVER THE ROW AND COLUMN
RST_PB_DIRC   BSF       STATUS, RP0   ; PUT THE PORT BACK TO INITIAL SETTINGS
              MOVLW     B'11110000'
              MOVWF     TRISB         ; SET RB1-RB3 AS OUTPUT AND
              MOVLW     B'00000000'   ; RB4-RB7 AS INPUT
              MOVWF     TRISA         ; SET RA0-RA3 AS OUTPUT
              BCF       STATUS, RP0
              CLRF      PORTB
              MOVF      PORTB, W      ; REQUIRED TO CLEAR RBIF FLAG
              BCF       INTCON, RBIF
              RETFIE
```

Keypad Interfacing Example

```
CONVERT     BTFSS    COL_INDEX,3 ;  IF  1ST  COLUMN, COL_INDEX=0
            MOVLW    0
            BTFSS    COL_INDEX,2 ;  IF 2ND  COLUMN, COL_INDEX=1
            MOVLW    1
            BTFSS    COL_INDEX,1 ;  IF 3RD  COLUMN, COL_INDEX=2
            MOVLW    2
            MOVWF    COL_INDEX  ;  STORE THE COLUMN INDEX

FIND_ROW    BTFSS    ROW_INDEX,7 ;  IF  1ST  ROW, ROW_INDEX=0
            MOVLW    0
            BTFSS    ROW_INDEX,6 ;  IF  2ND  ROW, ROW_INDEX=1
            MOVLW    1
            BTFSS    ROW_INDEX,5 ;  IF  3RD ROW, ROW_INDEX=2
            MOVLW    2
            BTFSS    ROW_INDEX,4 ;  IF  4TH ROW, ROW_INDEX=3
            MOVLW    3
            MOVWF    ROW_INDEX
```

; CONTINUED ON NEXT PAGE

Keypad Interfacing Example

```
COMPUTE_VALUE   MOVF    ROW_INDEX, W   ; KEY # = ROW_INDEX*3 + COL_INDEX
                ADDWF   ROW_INDEX, W
                ADDWF   ROW_INDEX, W
                ADDWF   COL_INDEX, W   ; THE VALUE IS IN W
    ; CHECK IF VALUE IS GREATER THAN 11. THIS HAPPENS WHEN THE BUTTON IS RELEASED
    ; LATER, AN INTERRUPT OCCURS WITH ALL SWITCHES OPEN, SO THE  MAPPED  VALUE IS ;
    ; ABOVE 11
                MOVWF   0X30           ; COPY THE BUTTON NUMBER
                MOVLW   0X0C
                SUBWF   0X30,W
                BTFSC   STATUS, C      ; WILL NOT WORK CORRECTLY, OVERFLOW OCCURS
                GOTO    LL
                MOVF    0X30, W
                CALL    TABLE
                MOVWF   PORTA          ; DISPLAY THE NUMBER ON PORTA
LL              RETURN
```

Keypad Interfacing Example

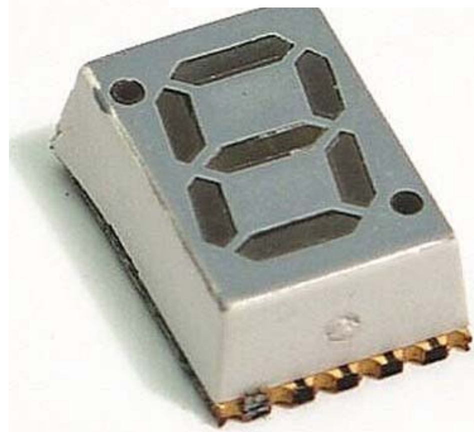
```
TABLE     ADDWF     PCL, F
          RETLW     0X01
          RETLW     0X02
          RETLW     0X03
          RETLW     0X04
          RETLW     0X05
          RETLW     0X06
          RETLW     0X07
          RETLW     0X08
          RETLW     0X09
          RETLW     0X0F      ; ERROR CODE
          RETLW     0X00
          RETLW     0X0F      ; ERROR CODE

          END
```

LED Displays

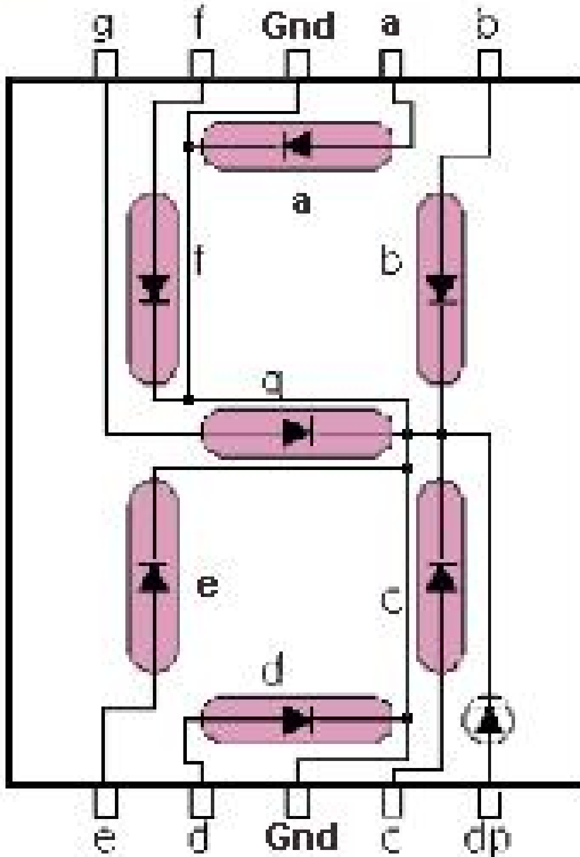
- Light emitting diodes are simple and effective in conveying information

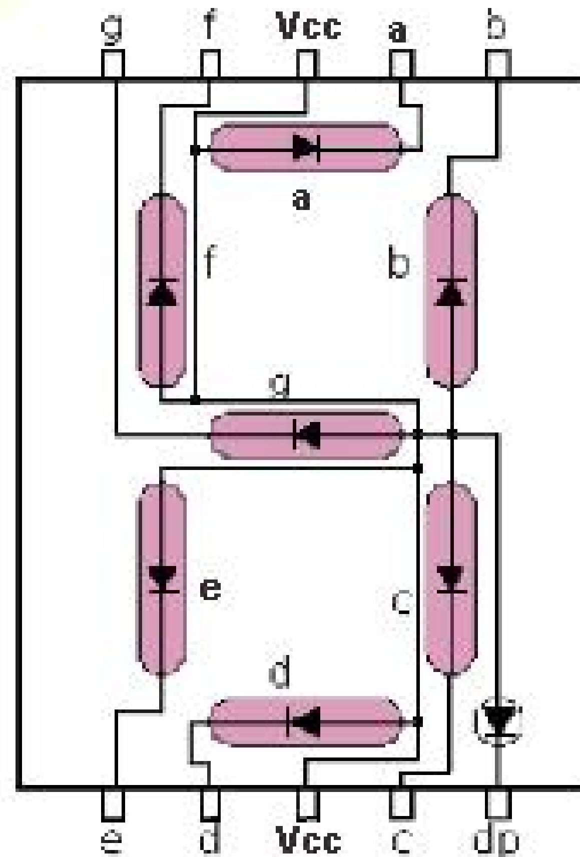
- However, in complex systems it becomes hard to deal with individual LEDs

- Alternatives
  - Seven segment displays
  - Bargraph
  - Dot matrix
  - Star-burst



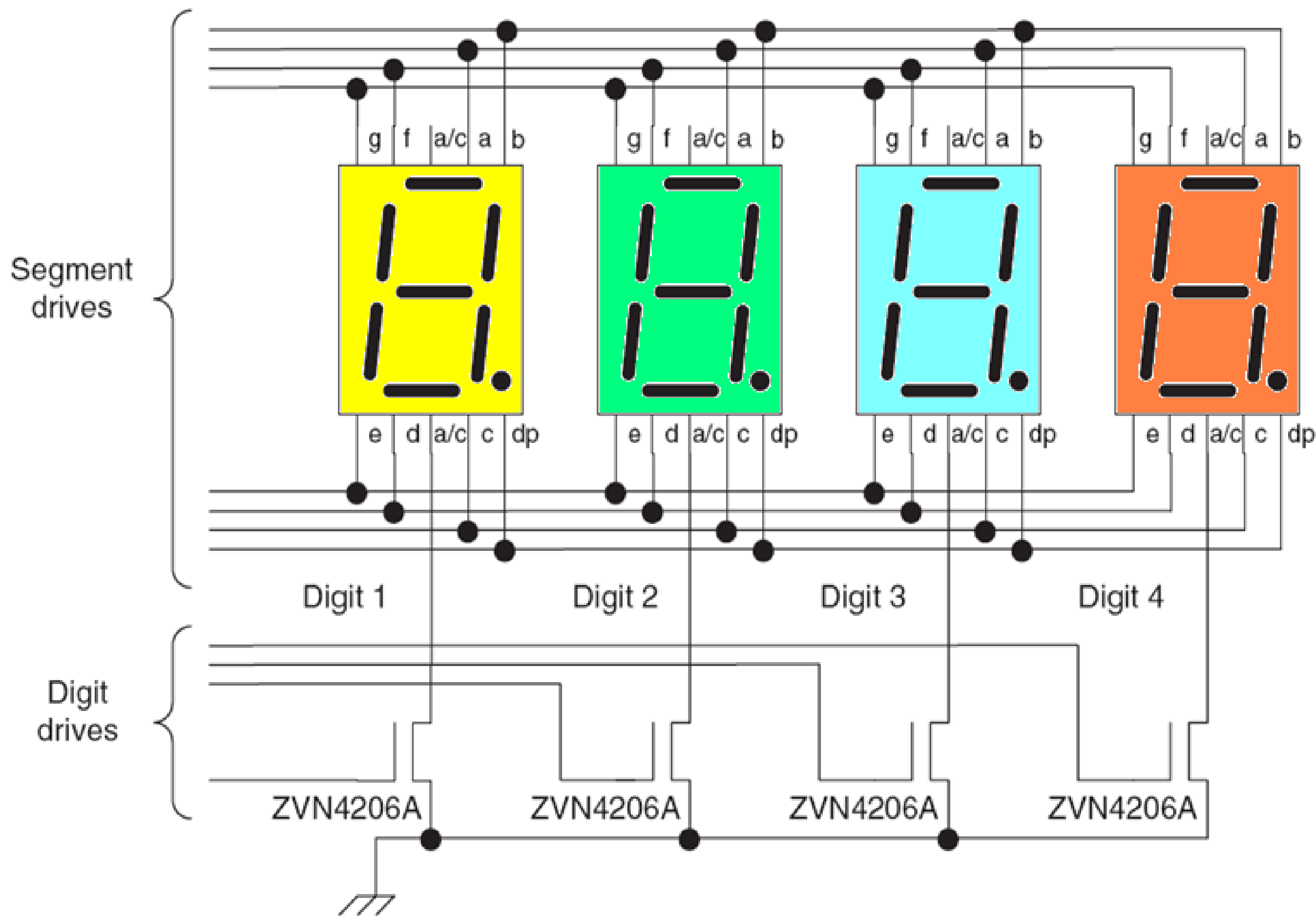
Seven Segment Display





Seven Segment Display

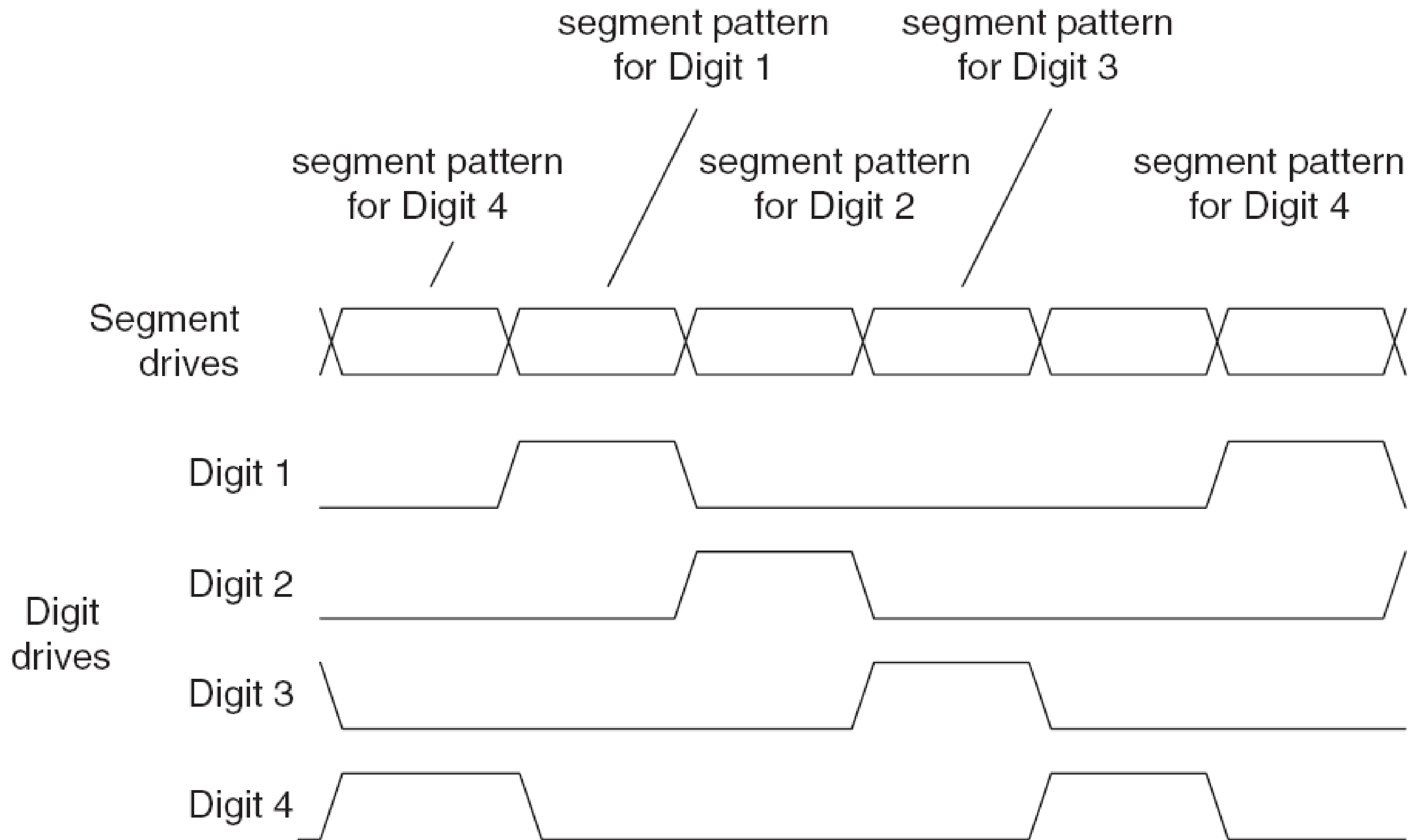
Multiplexing of seven segment digits



Connection	Port Bit
Segment a	RB7
Segment b	RB6
Segment c	RB5
Segment d	RB4
Segment e	RB3
Segment f	RB2
Segment g	RB1
Segment dp	RB0
Digit 1 drive	RA0
Digit 2 drive	RA1
Digit 3 drive	RA2
Digit 4 drive	RA3

Seven Segment Display

Multiplexing of seven segment digits



Seven Segment Display

Example 2

A program to count continuously the numbers 0 through 99 and display them on two seven segment displays. The count should be incremented every 1 sec. Oscillator frequency is 3 MHz.

- Connect the seven segment inputs **a** through **g** to RB0 through RB6, respectively
- Connect the gates of the controlling transistors to RA0 (LSD) and RA1 (MSD)
- The main program will be responsible for display and multiplexing every 5 ms

Seven Segment Display Example

```
            #INCLUDE    PICF84A.INC
LOW_DIGIT   EQU         0X20
HIGH_DIGIT  EQU         0X21
COUNT       EQU         0X22
            ORG         0X0000
            GOTO        START
            ORG         0X0004
ISR         GOTO        ISR
START       BSF         STATUS, RP0
            MOVLW       B’00000000’  ; set port B as output
            MOVWF       TRISB
            MOVWF       TRISA        ; SET RA0-RA1 AS OUTPUT
            BCF         STATUS, RP0
            CLRF        PORTB
            CLRF        PORTA
            CLRF        LOW_DIGIT    ; CLEAR THE COUNT VALUE
            CLRF        HIGH_DIGIT
            CLRF        COUNT
```

Seven Segment Display Example

```
DISPLAY     BSF       PORTA , 0
            BCF       PORTA,  1
            MOVF      LOW_DIGIT, W     ; DISPLAY LOWER DIGIT
            CALL      TABLE            ; GET THE SEVEN SEGMENT CODE
            MOVWF     PORTB
            CALL      DELAY_5MS        ; KEEP IT ON FOR 5 MS
            BCF       PORTA, 0
            BSF       PORTA, 1
            MOVF      HIGH_DIGIT, W    ; DISPLAY  HIGH DIGIT
            CALL      TABLE            ; GET THE SEVEN SEGMENT CODE\
            MOVWF     PORTB
            CALL      DELAY_5MS        ; KEEP IT ON FOR 5 MS
            ; CHECK IF  1 SEC ELAPSED
            INCF      COUNT,F          ; INCREMENT THE COUNT VALUE IF TRUE
            MOVF      COUNT, W
            SUBLW     D'100'
            BTFSS     STATUS, Z
            GOTO      DISPLAY          ; DISPLAY THE SAME COUNT
```

Seven Segment Display Example

```
; TIME TO INCREMENT THE COUNT
CLRF        COUNT
INCF        LOW_DIGIT, F   ; INCREMENT LOW DIGIT AND CHECK IF > 9
MOVF        LOW_DIGIT, W
SUBLW       0X0A
BTFSS       STATUS, Z
GOTO        DISPLAY
CLRF        LOW_DIGIT

INCF        HIGH_DIGIT, F ; INCREMENT HIGH DIGIT AND CHECK IF > 9
MOVF        HIGH_DIGIT, W
SUBLW       0X0A
BTFSS       STATUS, Z
GOTO        DISPLAY
CLRF        HIGH_DIGIT
GOTO        DISPLAY
```

Seven Segment Display Example

```
DELAY_5MS    MOVLW     D'250'
             MOVWF     0X40
REPEAT       NOP
             NOP
             NOP
             NOP
             NOP
             NOP
             NOP
             NOP
             NOP
             NOP
             NOP
             NOP
             DECFSZ    0X40,1
             GOTO      REPEAT
             RETURN
```

Seven Segment Display Example

```
TABLE   ADDWF   PCL,    1
        RETLW   B'00111111'     ;'0'
        RETLW   B'00000110'     ;'1'
        RETLW   B'01011011'     ;'2'
        RETLW   B'01001111'     ;'3'
        RETLW   B'01100110'     ;'4'
        RETLW   B'01101101'     ;'5'
        RETLW   B'01111101'     ;'6'
        RETLW   B'00000111'     ;'7'
        RETLW   B'01111111'     ;'8'
        RETLW   B'01101111'     ;'9'


        END
```

Sensors

- Embedded systems need to interface with the physical world and must be able to detect the state of the physical variables and control them

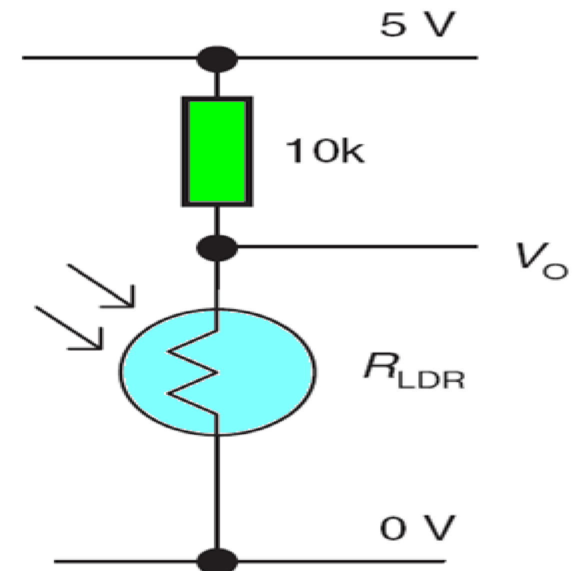
- **Input transducers** or sensors are used to convert physical variables into electrical variables. Examples are the light, temperature and pressure sensors

- **Output transducers** convert electrical variables to physical variables.

Sensors

Light-dependent Resistors

- A light-dependent resistor (LDR) is made from a piece of exposed semiconductor material
- When light falls on it, it creates hole–electron pairs in the material, which improves the conductivity.

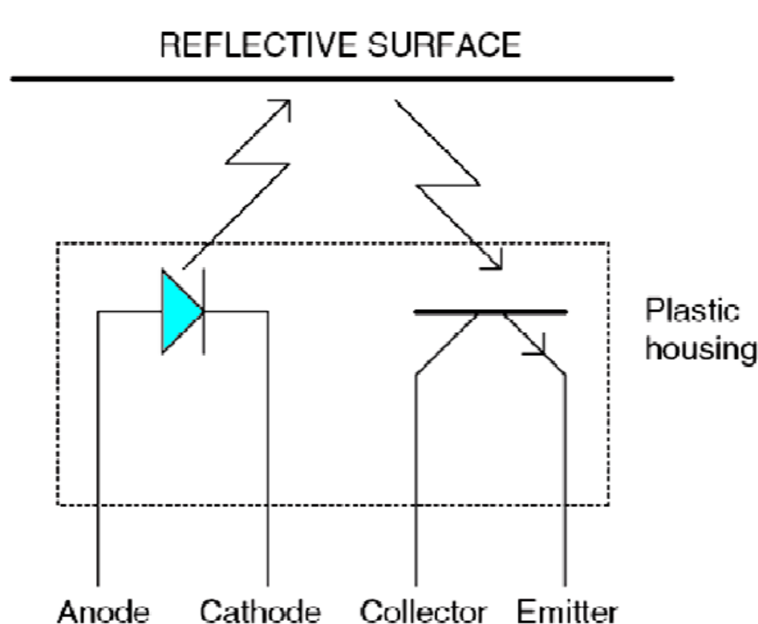


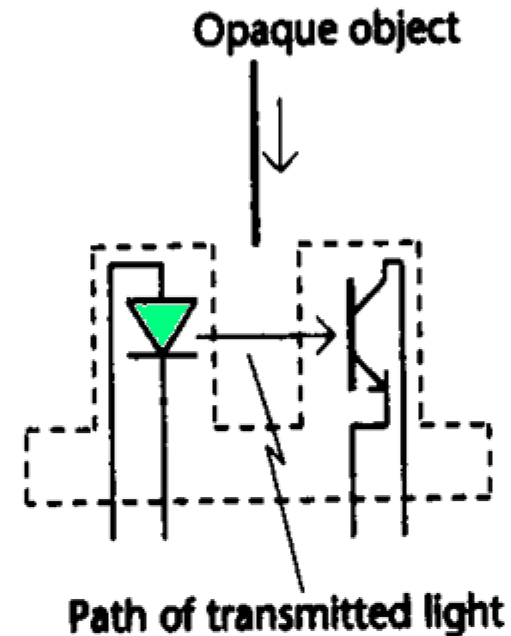
Illumination (lux)	R_{LDR} (Ohms)	Vo
Dark	2M	5
10	9000	2.36
1000	400	0.19

Sensors

Optical Object Sensing

- Useful in sensing the presence or closeness of objects
- The presence of object can be detected
  - If it breaks the light beam
  - If it reflects the light beam





Sensors

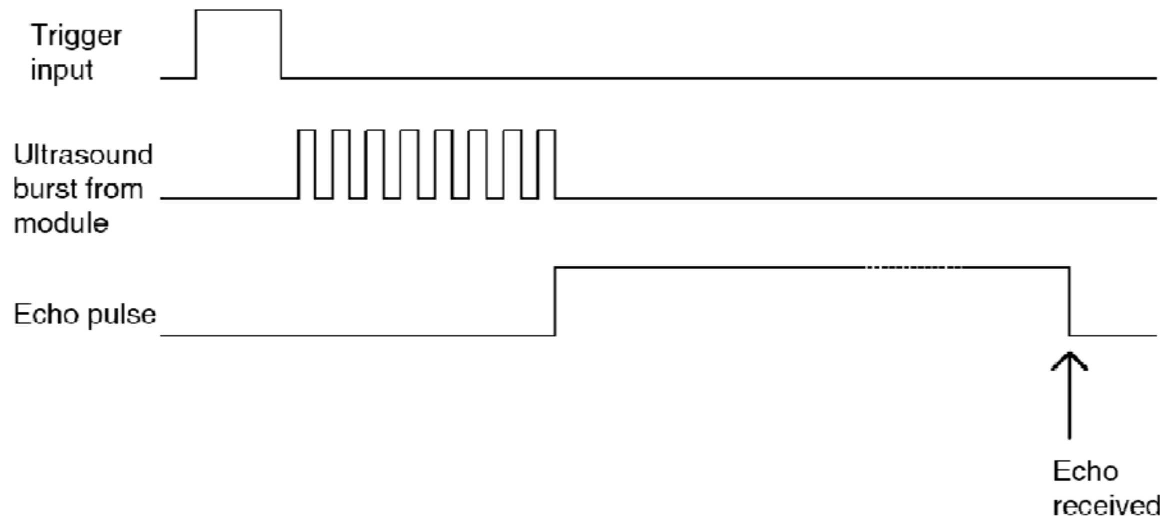
Opto-sensor as a Shaft Encoder

- Useful in measuring distance and speed

Sensors

Ultrasonic Object Sensor

- Based on reflective principle of ultrasonic waves
- An ultrasonic transmitter sends out a burst of ultrasonic pulses and then the receiver detects the echo
- If the time-to-echo is measured, distance can be measured

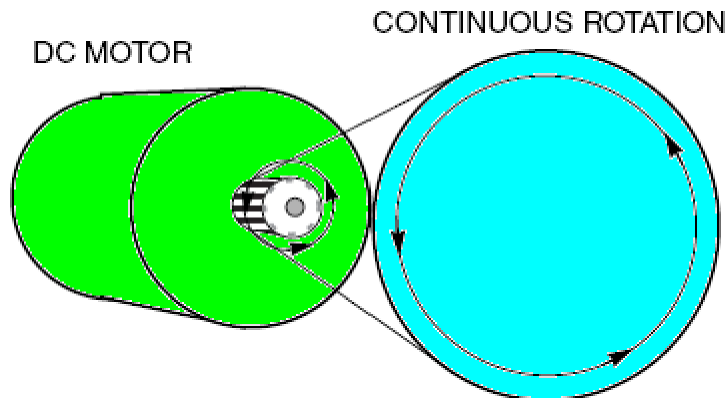


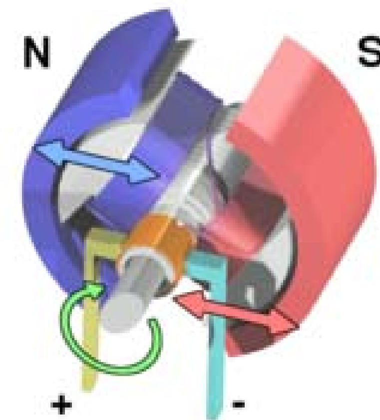
Actuators: motors and servos

- Embedded systems need to cause physical movement
- Linear or rotary motion
- Most actuators are electrical in nature
  - Solenoids (linear motion)
  - DC Motors
  - Stepper motors
  - Servo motors

DC Motors

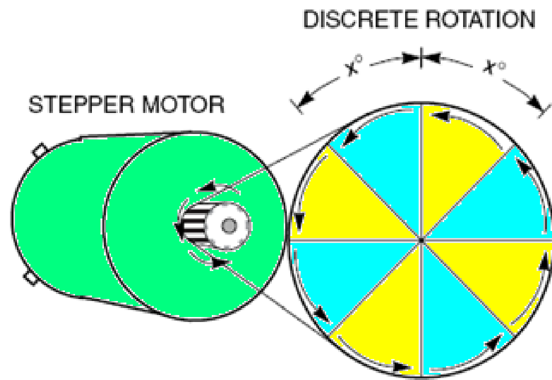
- Range from the extremely powerful to the very small
- Wide speed range
- Controllable speed
- Good efficiency
- Can provide accurate angular positioning with angular shafts
- Only the armature winding needs to be driven

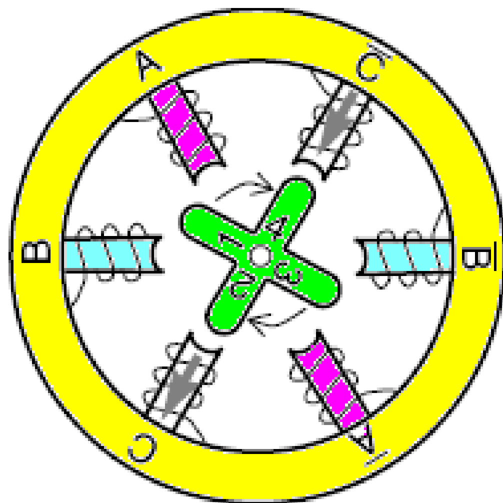


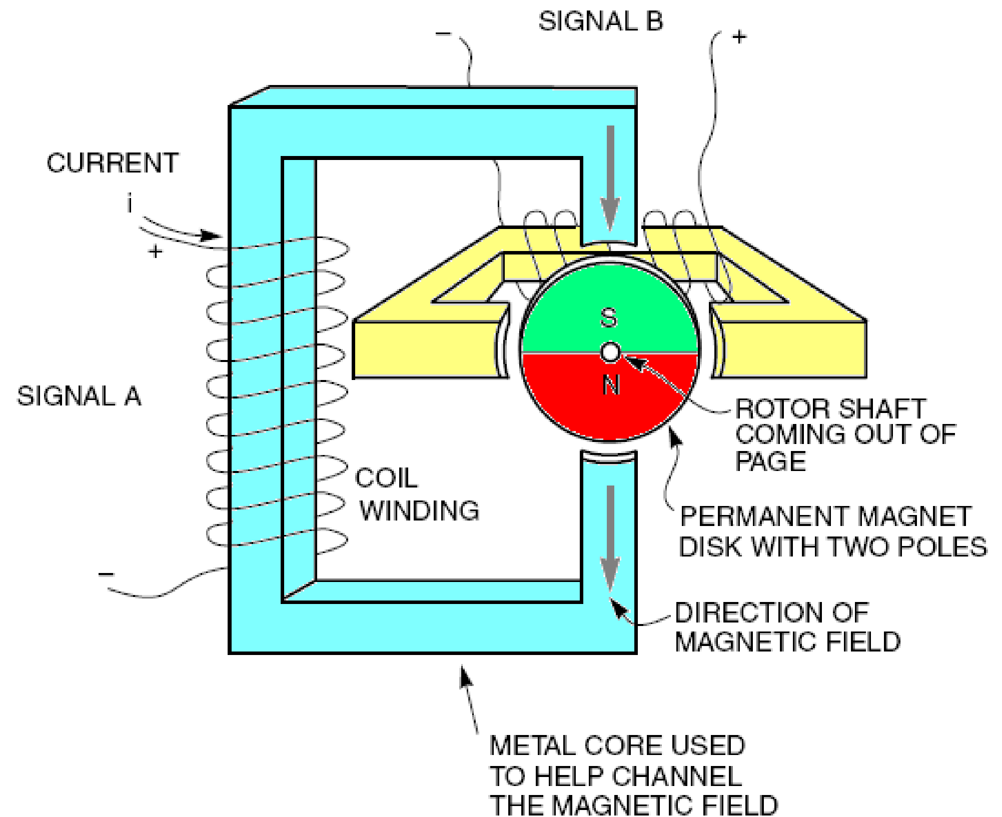


Stepper Motors

- A **stepper motor** (or **step motor**) is a synchronous electric motor that can divide a full rotation into a large number of steps.





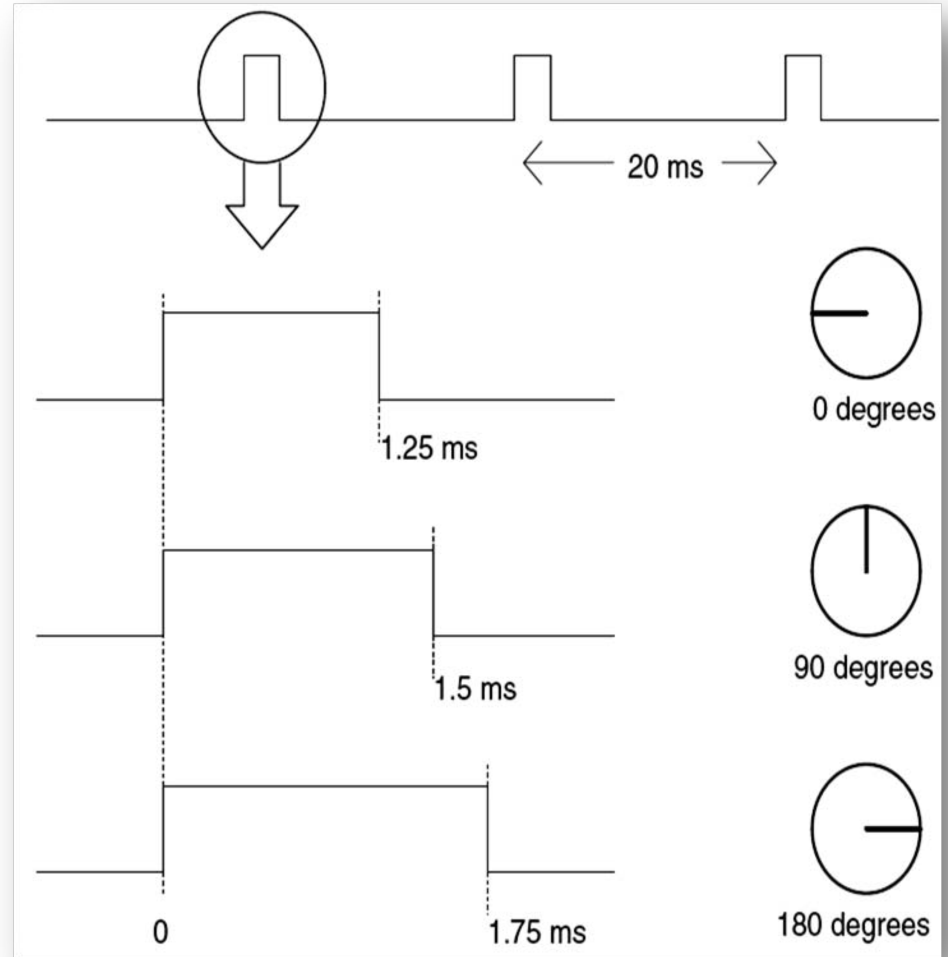


Stepper Motors

- **Features**
  - Simple interface with digital systems
  - Can control speed and position
  - More complex to drive
  - Awkward start-up characteristics
  - Lose torque at high speed
  - Limited top speed
  - Less efficient

Servo Motors

- Allows precise angular motion
- The output is a shaft that can take an angular position over a range of 180°
- The input to the servo is a pulse stream whose width determines the angular position of the shaft



Interfacing to Actuators

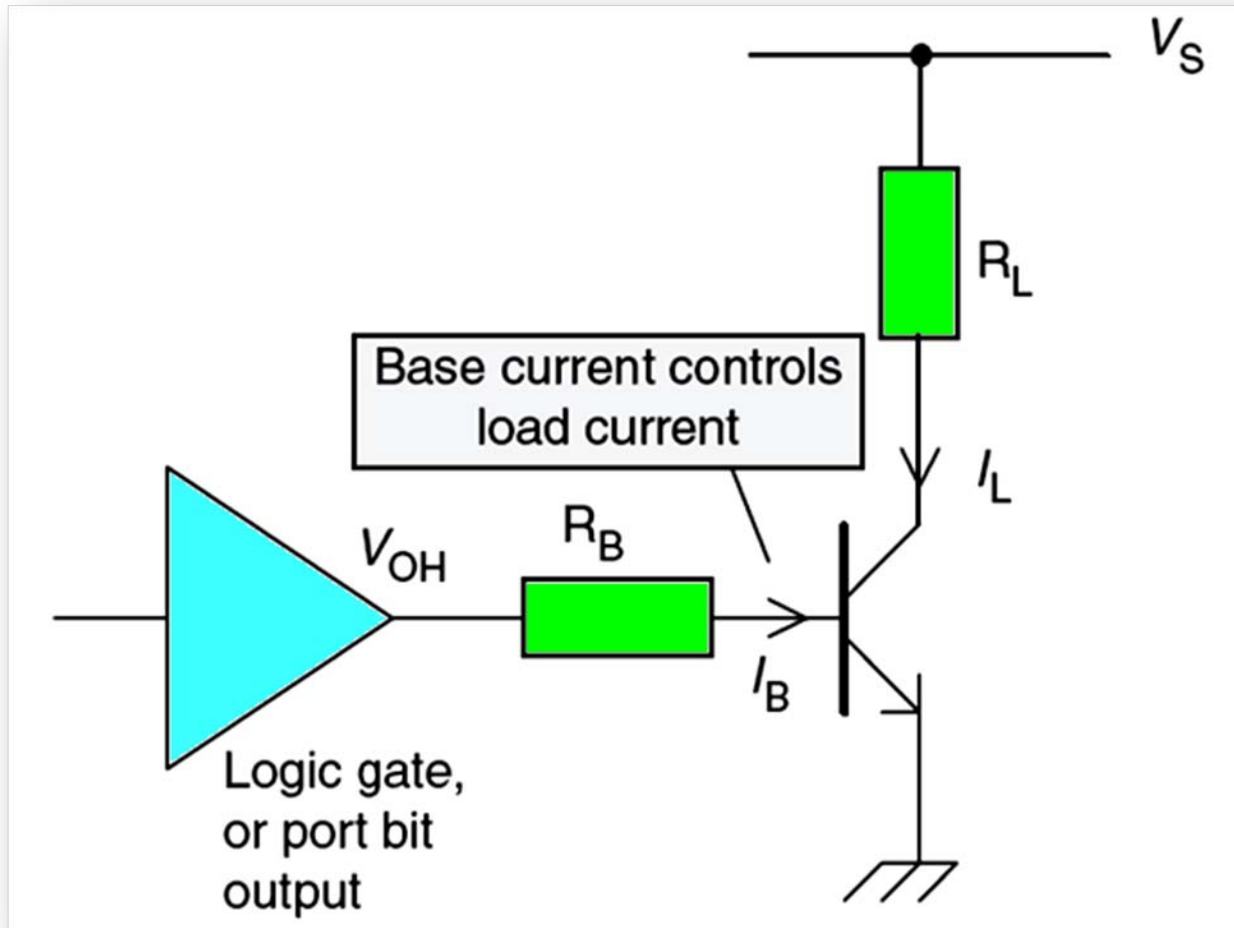
- Microcontrollers can drive loads with small electrical requirements

- Some devices, like actuators, require high currents or supply voltages

- Use switching devices
  - Simple DC switching using BJTs or MOSFETs
  - Reversible DC switching using H-bridge

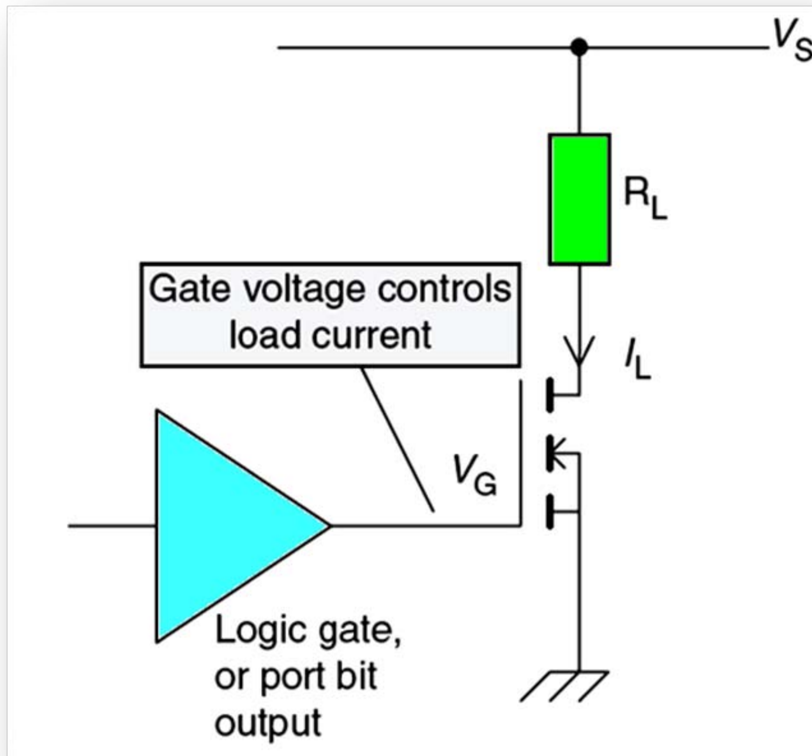
Interfacing to Actuators

Simple DC interfacing

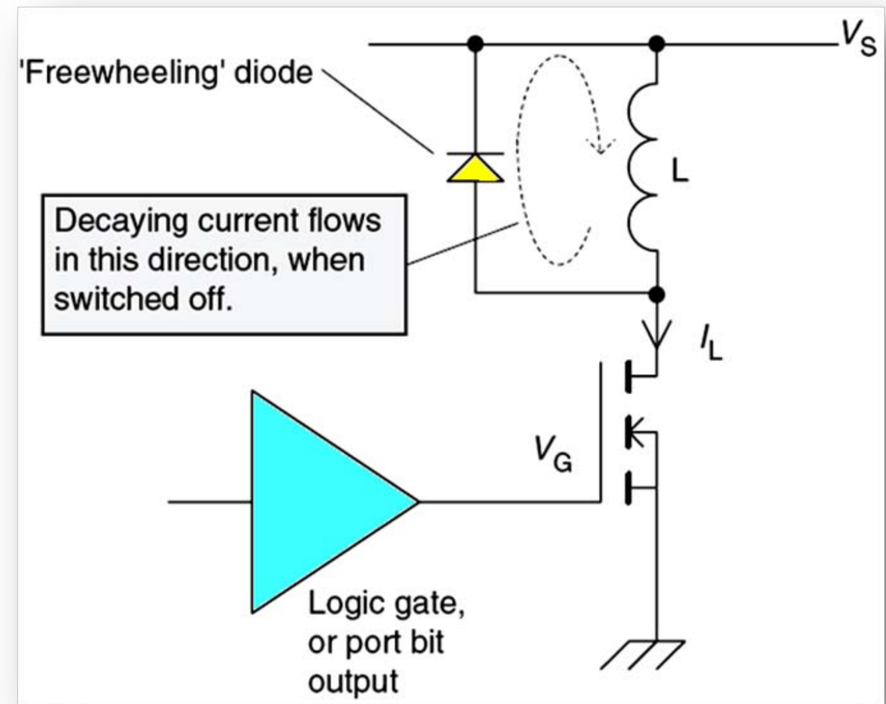


Interfacing to Actuators

Simple DC interfacing



Resistive load



Inductive load

Interfacing to Actuators

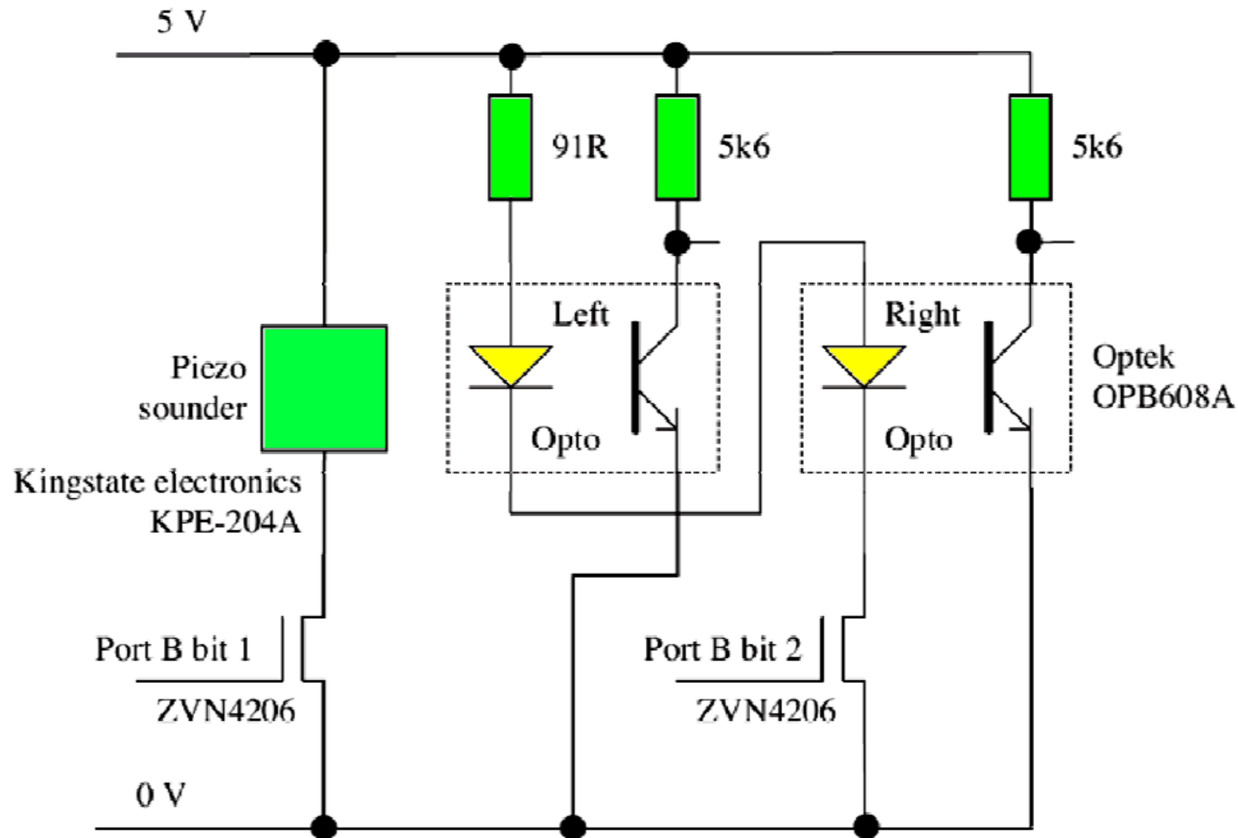
Simple DC interfacing

Characteristics of two popular logic-compatible MOSFETs

Characteristic	ZVN4206A	ZVN4306A
Maximum drain-to-source voltage, V_{DS} (V)	60	60
Maximum gate-to-source threshold, $V_{GS(th)}$ (V)	3	3
Maximum drain-to-source resistance when 'on', $R_{DS(on)}$ (Ω)	1.5	0.33
Maximum continuous drain current, I_D	600 mA	1.1 A
Maximum power dissipation (W)	0.7	1.1
Input capacitance (pF)	100	350

Interfacing to Actuators

Driving Piezo Sounder and Opto-sensors

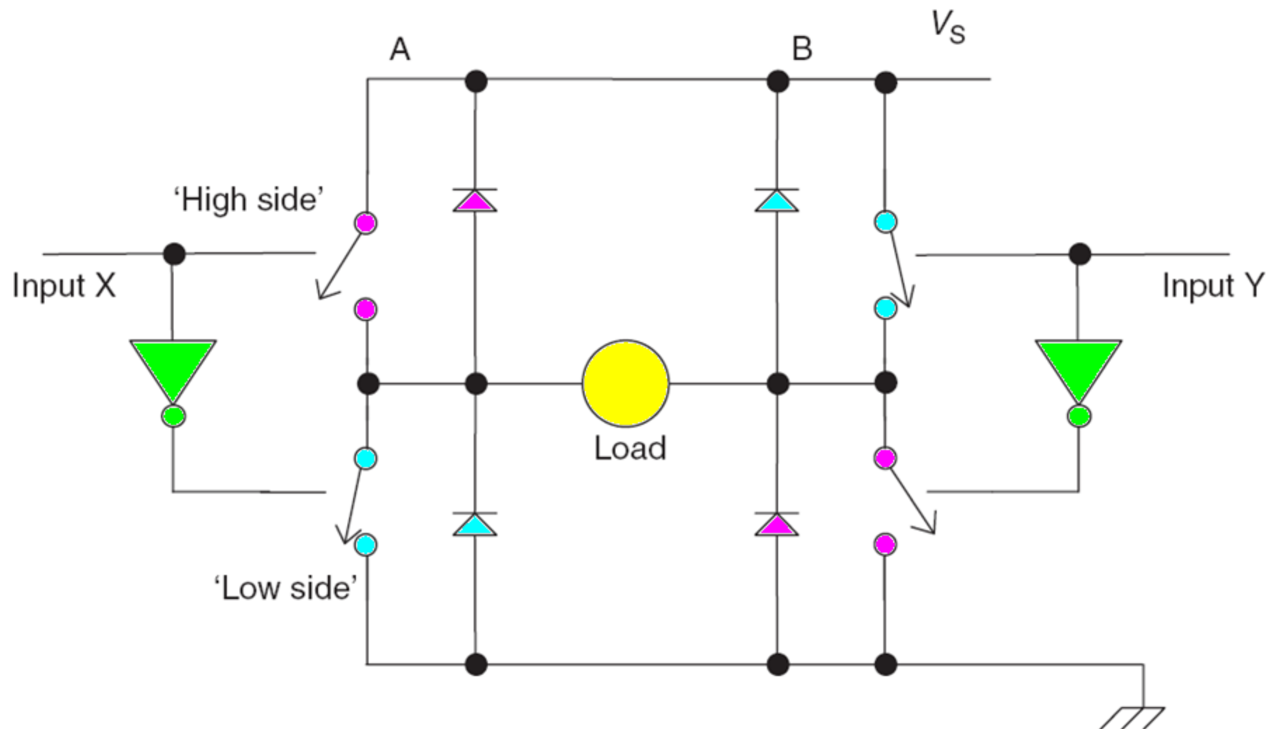


- Piezo sounder ratings: 9mA, 3-20 V
- The opto-sensor found to operate well with 91 Ohm resistor. The diode forward voltage is 1.7V. The required current is about 17.6 mA

Interfacing to Actuators

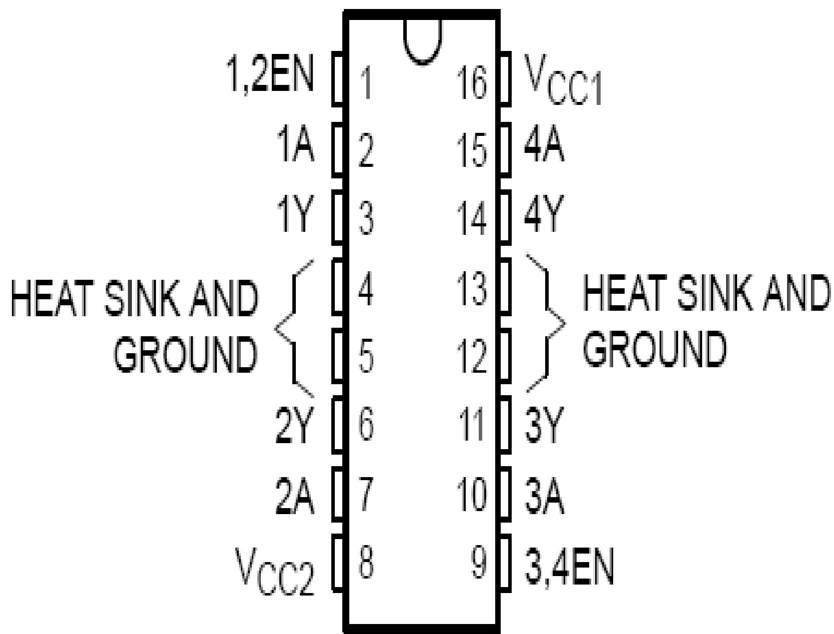
Reversible DC Switching

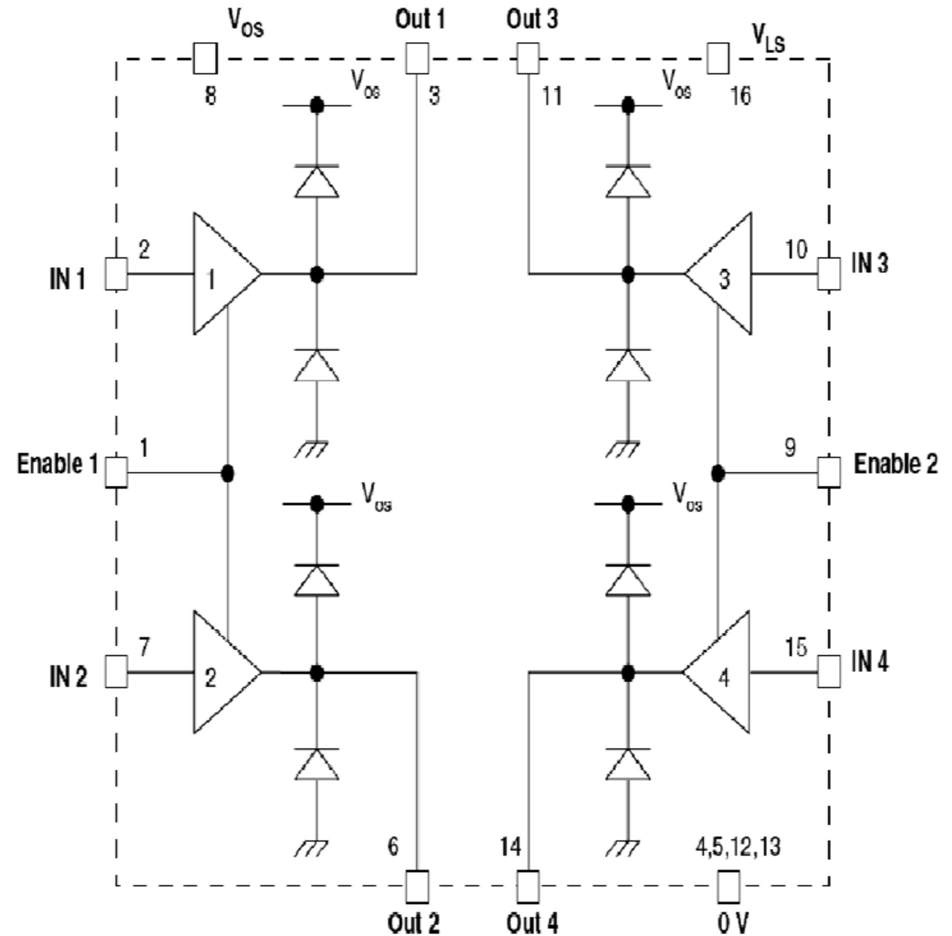
- DC switching allows driving loads with current flowing in one direction
- Some loads requires the applied voltage to be reversible; DC motors rotation depends on direction of current
- Use H-bridge !



Interfacing to Actuators

Reversible DC Switching





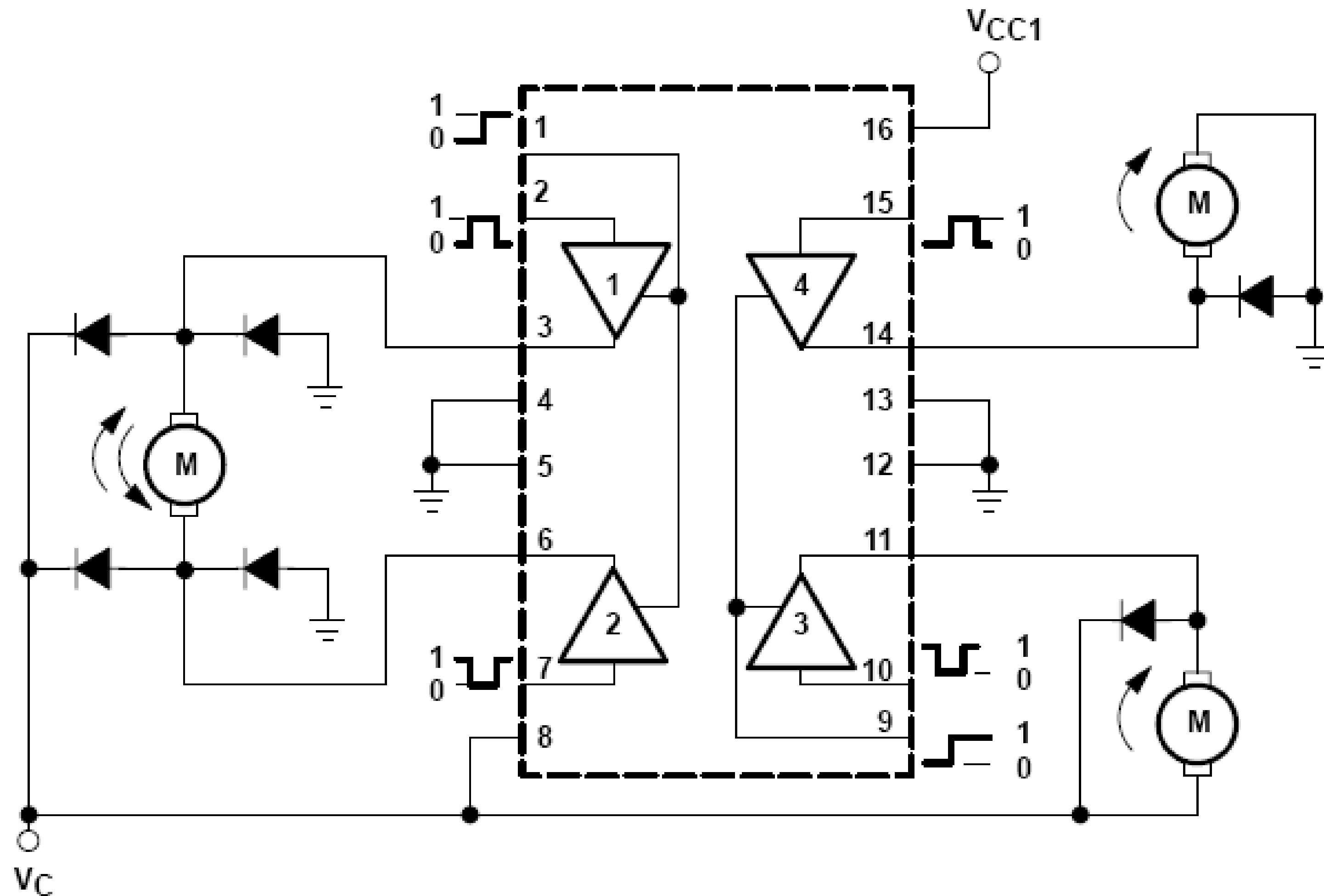
L293D Dual H-bridge

Peak output current 1.2 A per channel

Interfacing to Actuators

Reversible DC Switching

Driving three motors using L293D

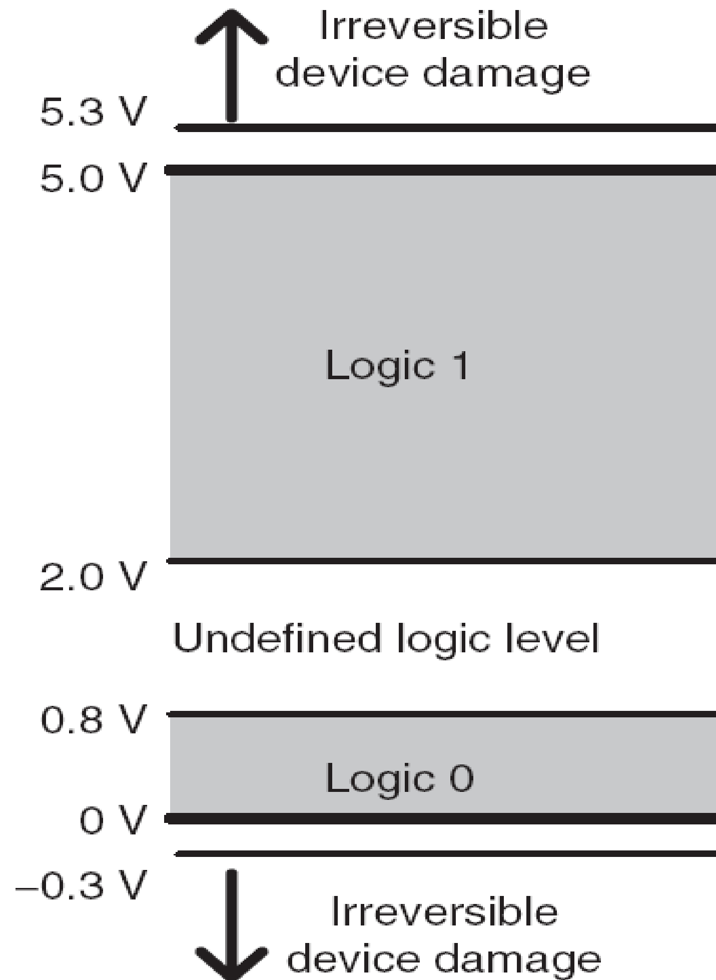


More on Digital Input

- When acquiring digital inputs into the microcontroller, it is essential that the input voltage is within the permissible and recognizable range of the MC
- Voltage range depends on the logic family; TTL, CMOS, …
- Interfacing within the same family is safe
- What for the case
  - Interfacing to digital sensors
  - Signal corruption
  - Interference

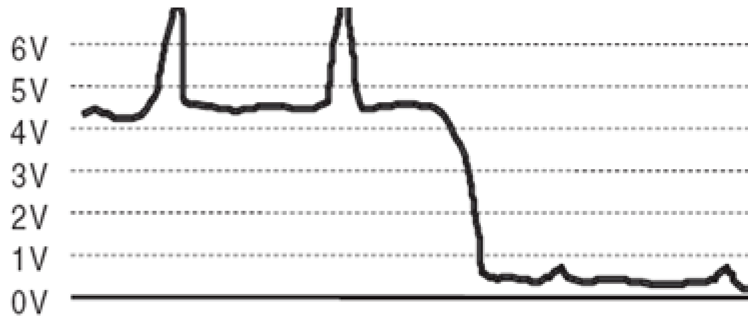
More on Digital Input

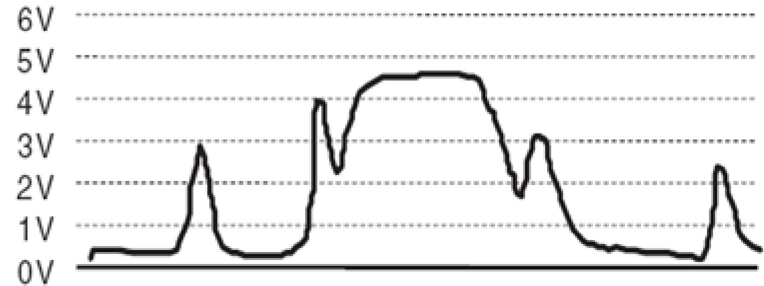
PIC16F873A Port Characteristics



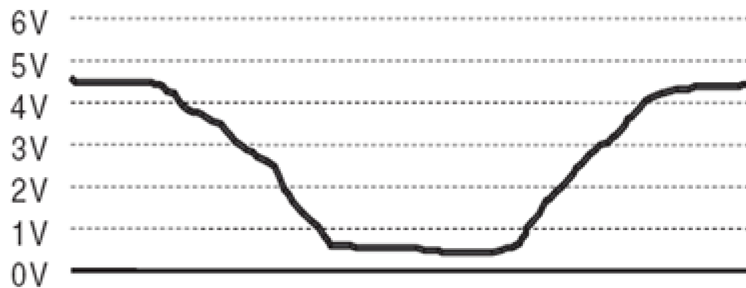
More on Digital Input

Forms of Signal Corruption

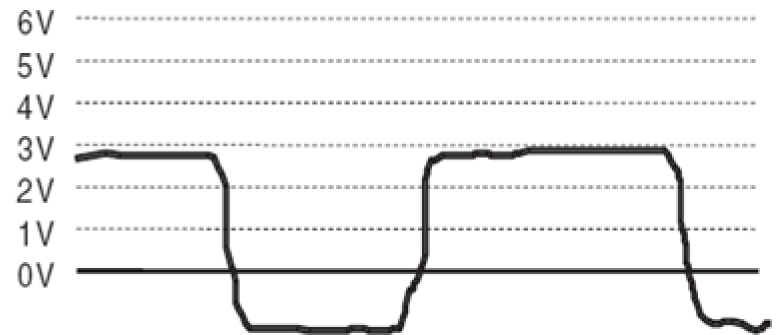




Spikes in the signal



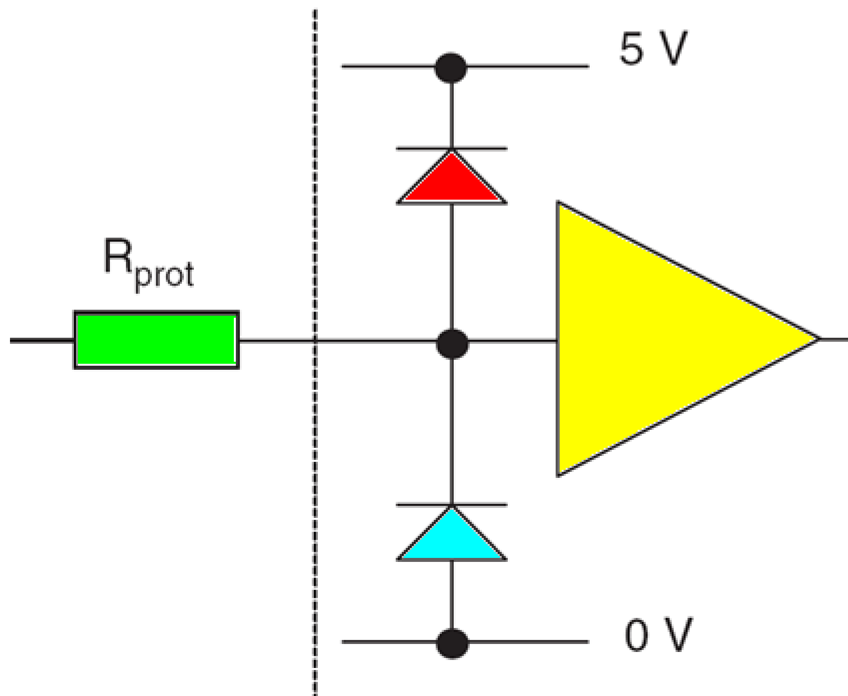
Slow edge



DC Offset in the signal

More on Digital Input

Clamping Voltage Spikes

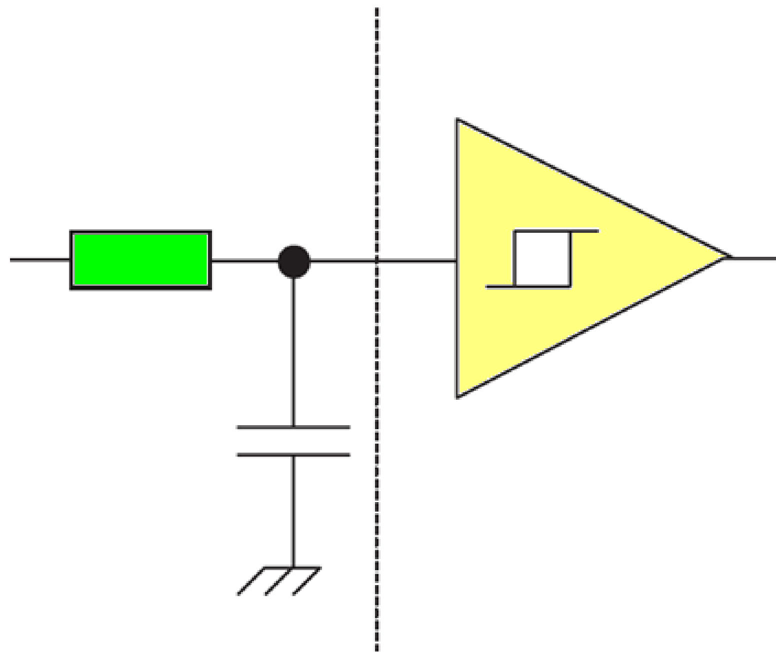


- All ports are usually protected by a pair of diodes
- An optional current limiting resistor can be added if high spikes are expected

- **Question?** Let R_{prot} = 1KΩ and the maximum diode current is 20 mA when V_d = 0.3v, then what is the maximum positive voltage spike that can be suppressed?

More on Digital Input

Analog Input Filtering

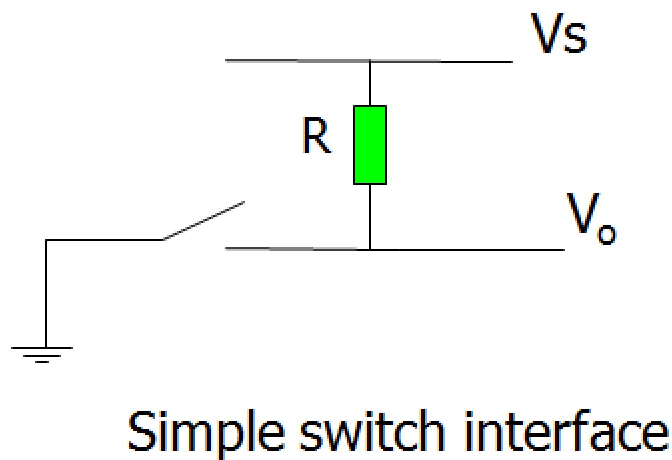


- Can use Schmitt trigger for speeding up slow logic edges.
- Schmitt trigger with RC filter can be used to filter voltage spikes.

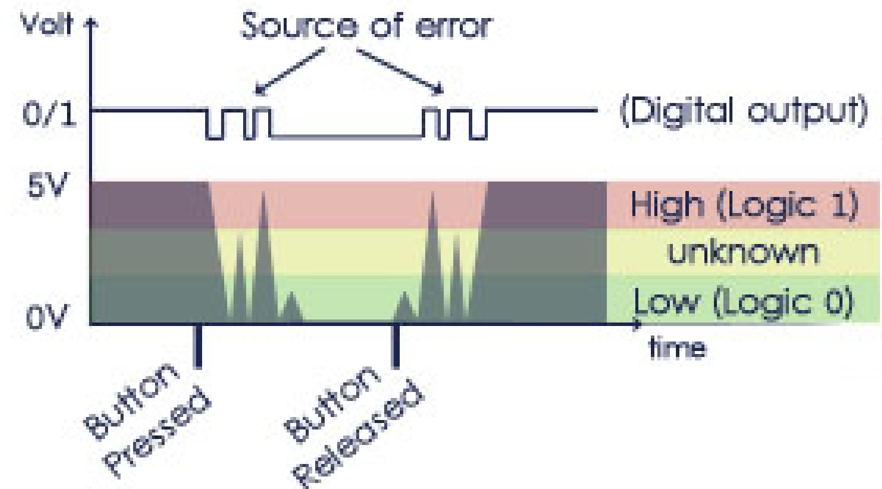
More on Digital Input

Switch Debouncing

- Mechanical switches exhibit bouncing behavior
- The switch contact bounces between open and closed
- A serious problem for digital devices ?!



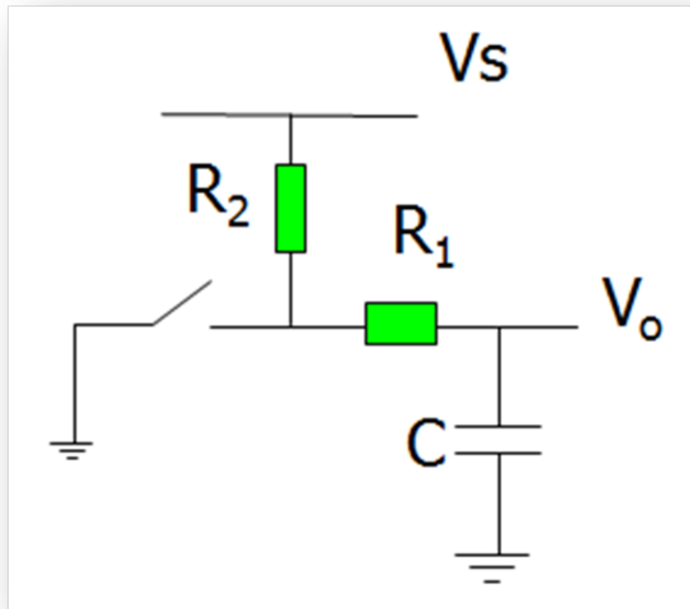
Simple switch interface

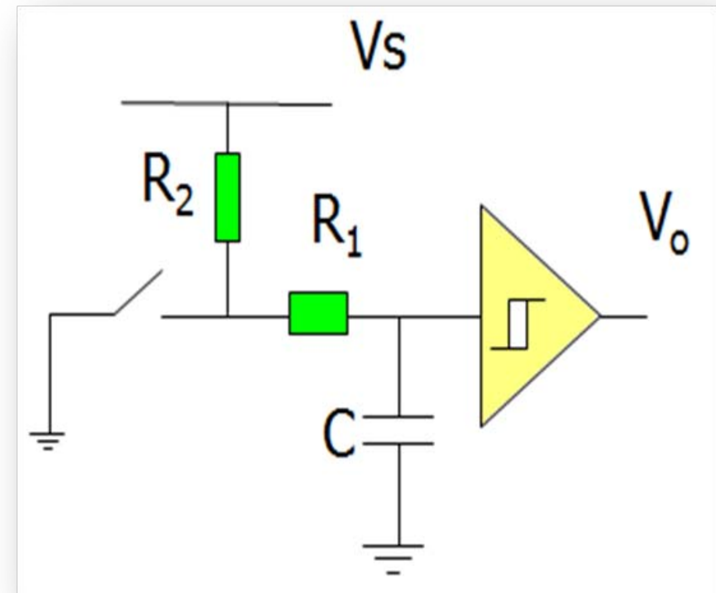


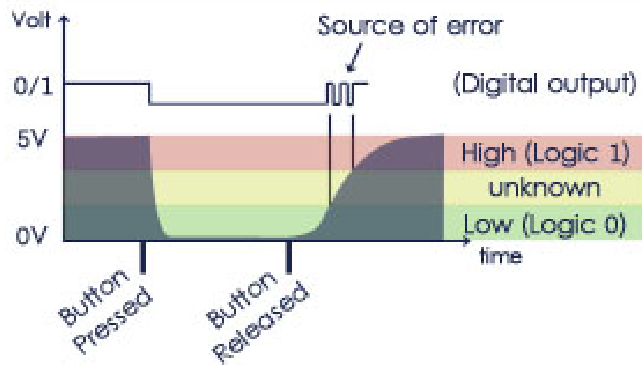
- Switch debouncing!! hardware and/or software techniques

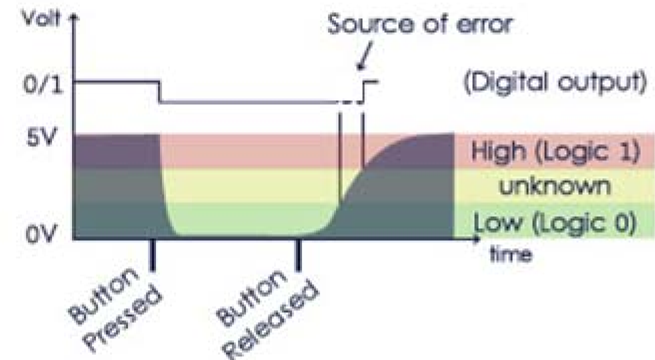
More on Digital Input

Switch Debouncing



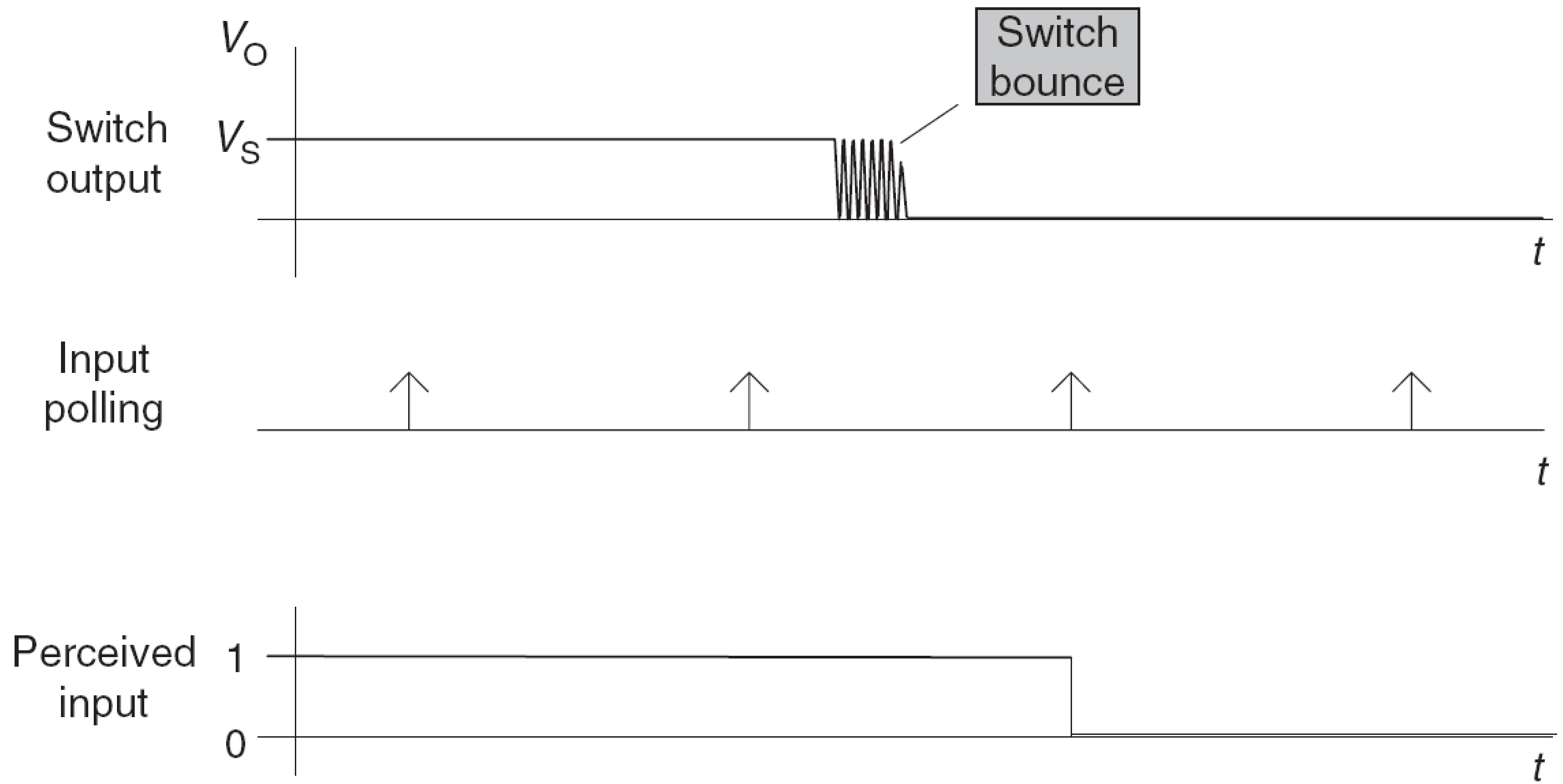






More on Digital Input

Switch Debouncing



Summary

- Microcontrollers must be able to interface with the physical world and possibly the human world
- Switches, keypads and displays represent typical examples for interfacing embedded systems with the humans
- Microcontrollers must be able to interface with a range of input and output transducers.
- Interfacing with sensors requires a reasonable knowledge of signal conditioning techniques
- Interfacing with actuators requires a reasonable knowledge of power switching techniques

Taking Timing Further

Chapter 9

Dr. Iyad Jafar

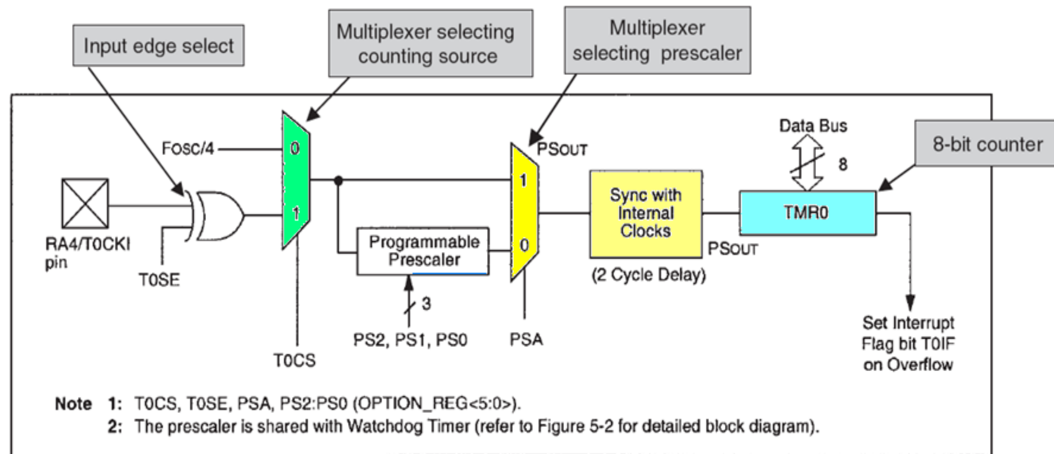
Outline

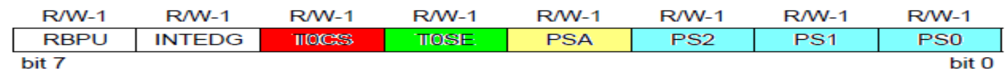
- Introduction
- Review of Timer 0 Module
- Timer 1 Module
- Timer 2 Module
- Capture/Compare/PWM (CCP)
- Digital-to-Analog Conversion
- Frequency Measurement
- Summary

Introduction

- **Why do we need timers ?**
  - Maintaining continuous counting functions
  - Recording ('capturing') in timer hardware the time an event occurs
  - Triggering events at particular times
  - Generating repetitive time-based events
  - Measuring frequency, e.g., motor speed

Review of Timer 0 Module





File Address			File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h

bit 7 **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

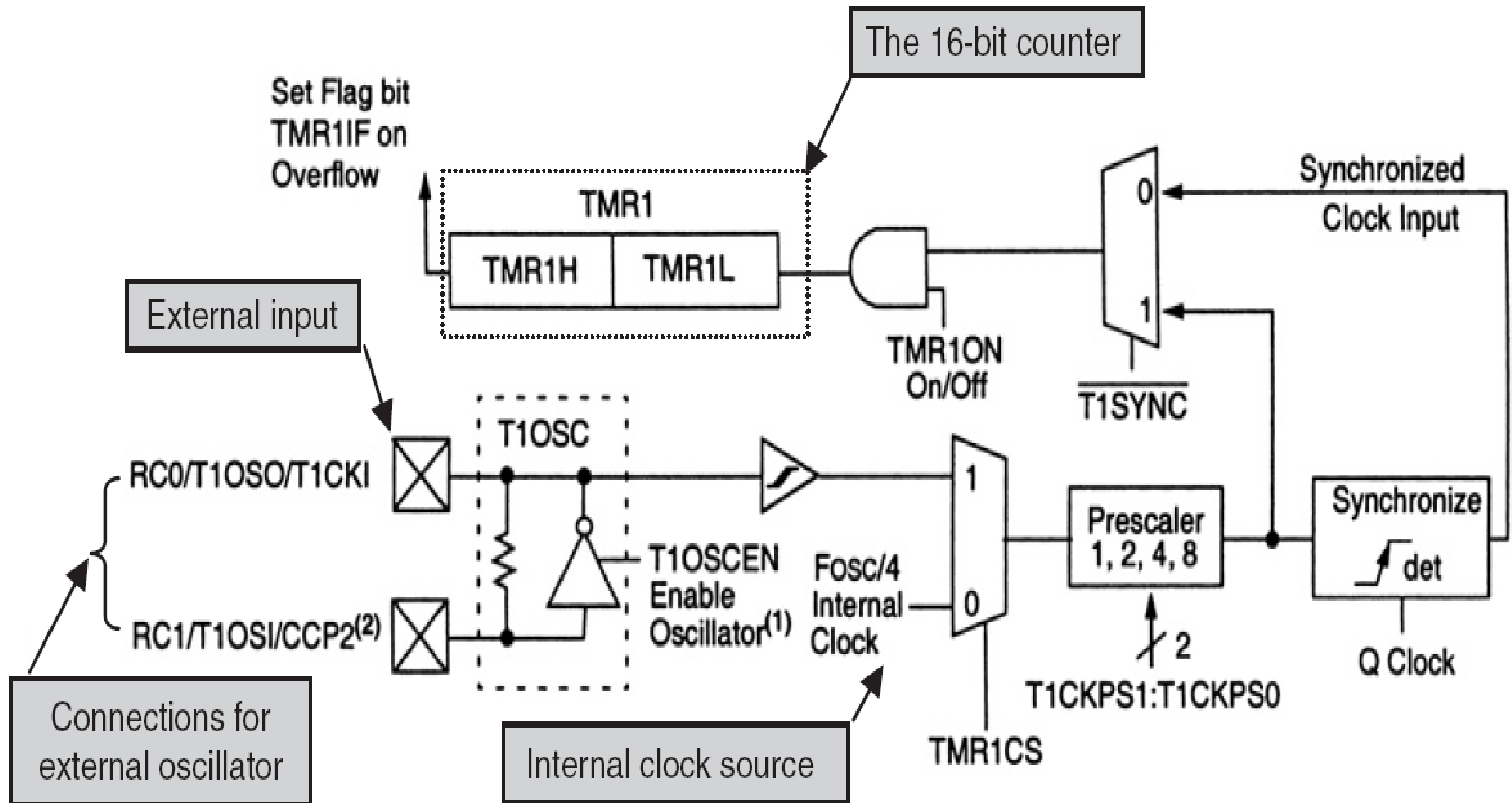
More Timer Modules

Device	Pins	Features
16F84A	18	**1 8-bit timer** 1 5-bit port 1 8-bit port
16F873A 16F876A	28	3 parallel ports, **3 counter/timers,** **2 capture/compare/PWM,** 2 serial, 5 10-bit ADC, 2 comparators
16F874A 16F877A	40	5 parallel ports, **3 counter/timers,** **2 capture/compare/PWM,** 2 serial, 8 10-bit ADC, 2 comparators

Timer 1 Module

- **Features**
  - 16-bit timer/counter (0000H – FFFFH)
  - Count value in TMR1H (0x0F) and TMR1L (0x0E)
  - TMR1 operation controlled by T1CON (0x10)
  - Three clock sources
    - Internal clock Fosc/4
    - External input (RC0/T1OSO/T1CKI) for counting purposes
      - Count on rising edge (after the first falling edge)
    - External oscillator (RC1/T1OSI/CCP2)
      - Removes the dependency on the main oscillator
      - Intended for low frequency oscillation up to 200KHz (typically 32.768 KHz)
      - Counting continue in sleep mode

Timer 1 Module



Timer 1 Module

T1CON Register (0x10)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7:6 **Unimplemented:** Read as '0'

bit 5:4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits

11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable bit

1 = Oscillator is enabled
0 = Oscillator is shut off. The oscillator inverter and feedback resistor are turned off to eliminate power drain

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Select bit

When TMR1CS = 1:

1 = Do not synchronize external clock input
0 = Synchronize external clock input

When TMR1CS = 0:

This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit

1 = External clock from pin T1OSO/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

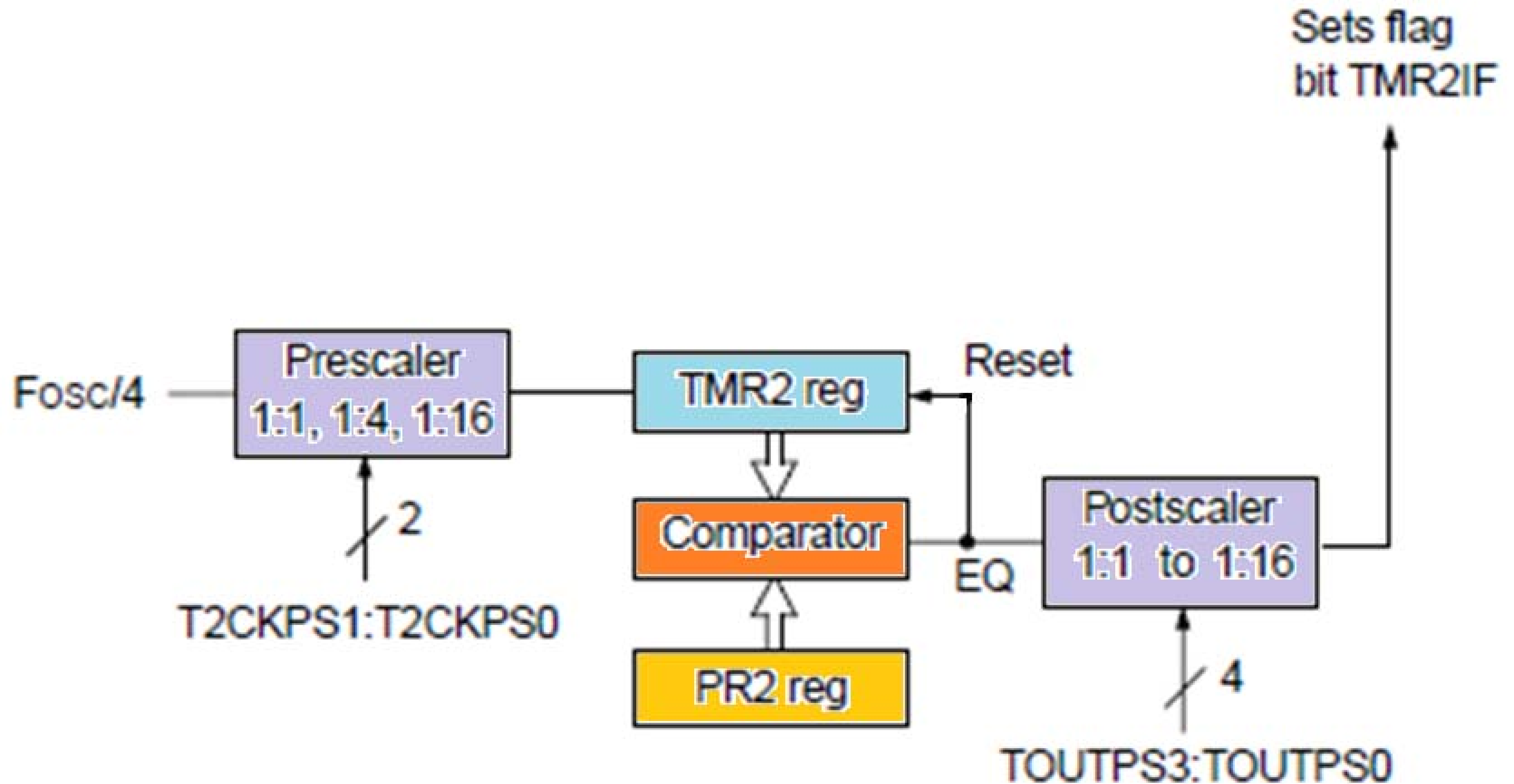
bit 0 **TMR1ON**: Timer1 On bit

1 = Enables Timer1
0 = Stops Timer1

Timer 2 Module

- **Features**
  - 8-bit counter/timer
  - Count value in TMR2 register (0x11)
  - TMR2 operation controlled by T2CON (0x12)
  - No external clock input
  - Has Capture and Compare register PR2 (0x92) and pulse width modulation capability

Timer 2 Module



Timer 2 Module

T2CON Register (0x12)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6:3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits

0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON**: Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

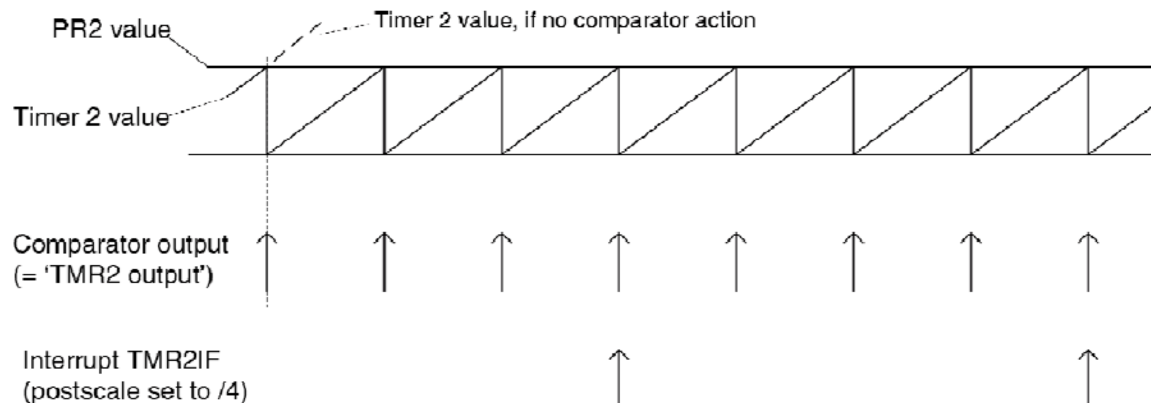
bit 1:0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Timer 2 Module

The PR2 register, comparator and prescaler

- Timer2 has a period register PR2 (0x92) that can be preset by the programmer
- The content of this register is continuously compared with the Timer2 when it is running
- When TMR2 equals PR2,
  - TMR2 is cleared
  - The comparator output (same as TMR2IF in PIR) is high which can be used as interrupt if TMR2IE (PIE) is set
  - The comparator output can be post-scaled by T2OUTPS3:T2OUTPS0 bits (T2CON)



Capture/Compare/PWM Modules

- Embedded systems need to deal with time events such as setting an **alarm** or **recording** the time of an event
- This can be easily achieved by adding one or more registers to the timer/counter registers
  - A register that records the time. It is called the Capture register
  - A register that triggers an alarm. It is called the Compare register
- The PIC 16 series combine these functionalities in the Capture/Compare/PWM (CCP) modules which interact with Timer1 and Timer2 modules

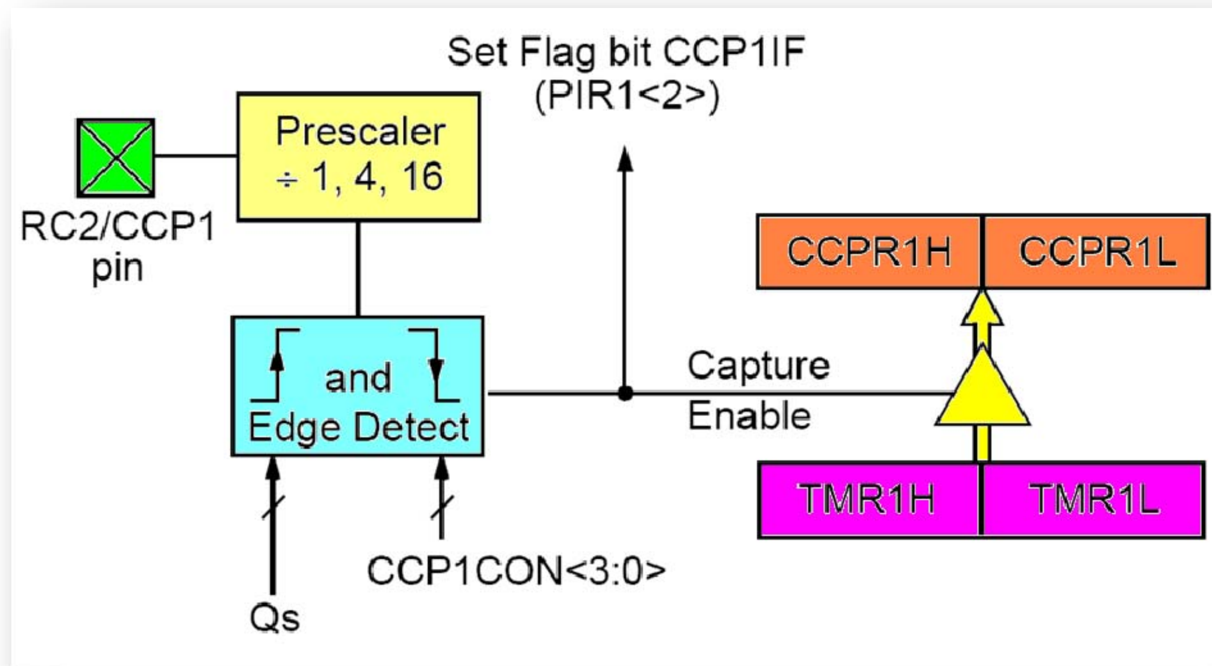
CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

- **The PIC16F873A has two such modules**
  - Each has two 8-bit registers CCP1H (0x16) and CCP1L (0x15) for module CCP1 and CCP2H (0x1C) and CCP2L (0x1B) for module CCP2
  - These registers can be used to capture a value from the timer, store the value to compare with, or store the duty cycle of PWM stream
  - Mode of operation is controlled by CCP1CON (0x17) and CCP2CON (0x1D) registers

Capture/Compare/PWM Modules

Capture Mode

- The compare register operates like a stopwatch!
- Can record the value of the timer when an event occurs

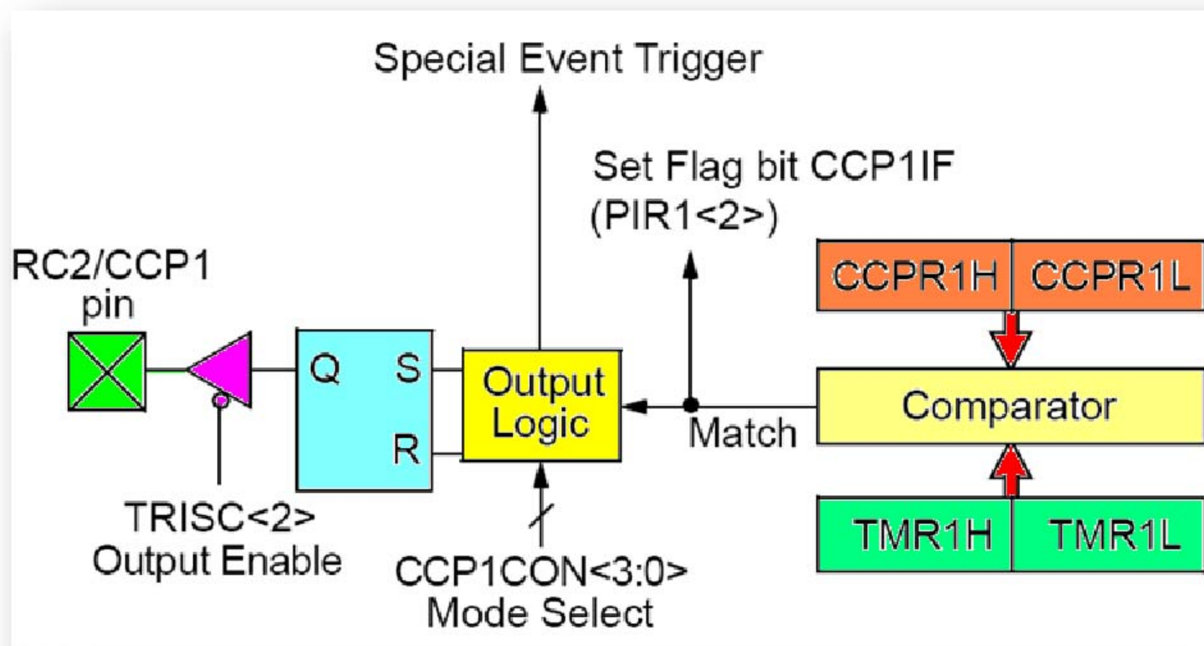


Block diagram of CCP1 module in capture mode

Capture/Compare/PWM Modules

Compare Mode

- The value stored in CCPR1H and CCPR1L is continuously compared to Timer1 registers
- The associated output pin can be set or cleared



Block diagram of CCP1 module in compare mode

Capture/Compare/PWM Modules

CCP Control Registers: CCP1CON and CCP2CON

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7:6 **Unimplemented:** Read as '0'

bit 5:4 **DCxB1:DCxB0**: PWM Duty Cycle bit1 and bit0

<u>Capture Mode:</u>
Unused

<u>Compare Mode:</u>
Unused

<u>PWM Mode:</u>
These bits are the two LSbs (bit1 and bit0) of the 10-bit PWM duty cycle. The upper eight bits (DCx9:DCx2) of the duty cycle are found in CCPRxL.

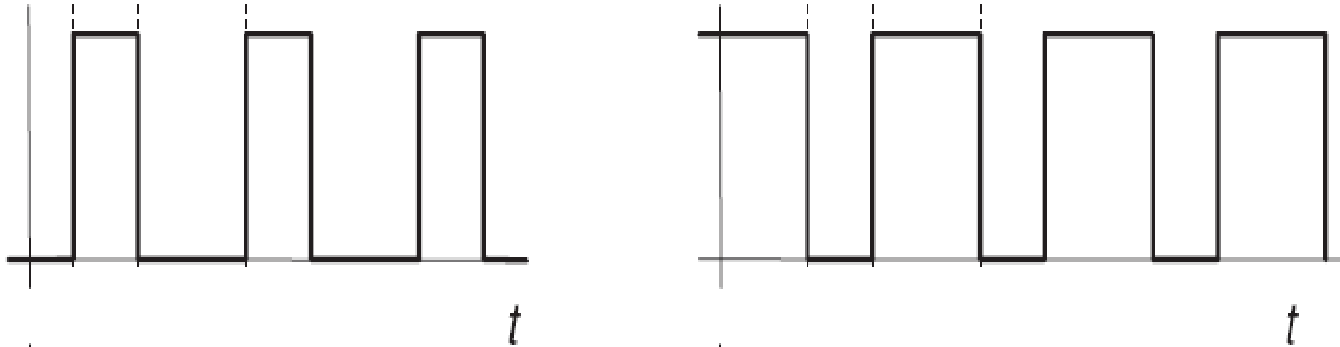
bit 3:0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM off (resets CCPx module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode,
Initialize CCP pin Low, on compare match force CCP pin High (CCPIF bit is set)
1001 = Compare mode,
Initialize CCP pin High, on compare match force CCP pin Low (CCPIF bit is set)
1010 = Compare mode,
Generate software interrupt on compare match
(CCPIF bit is set, CCP pin is unaffected)
1011 = Compare mode,
Trigger special event (CCPIF bit is set)
11xx = PWM mode

Capture/Compare/PWM Modules

Pulse Width Modulation

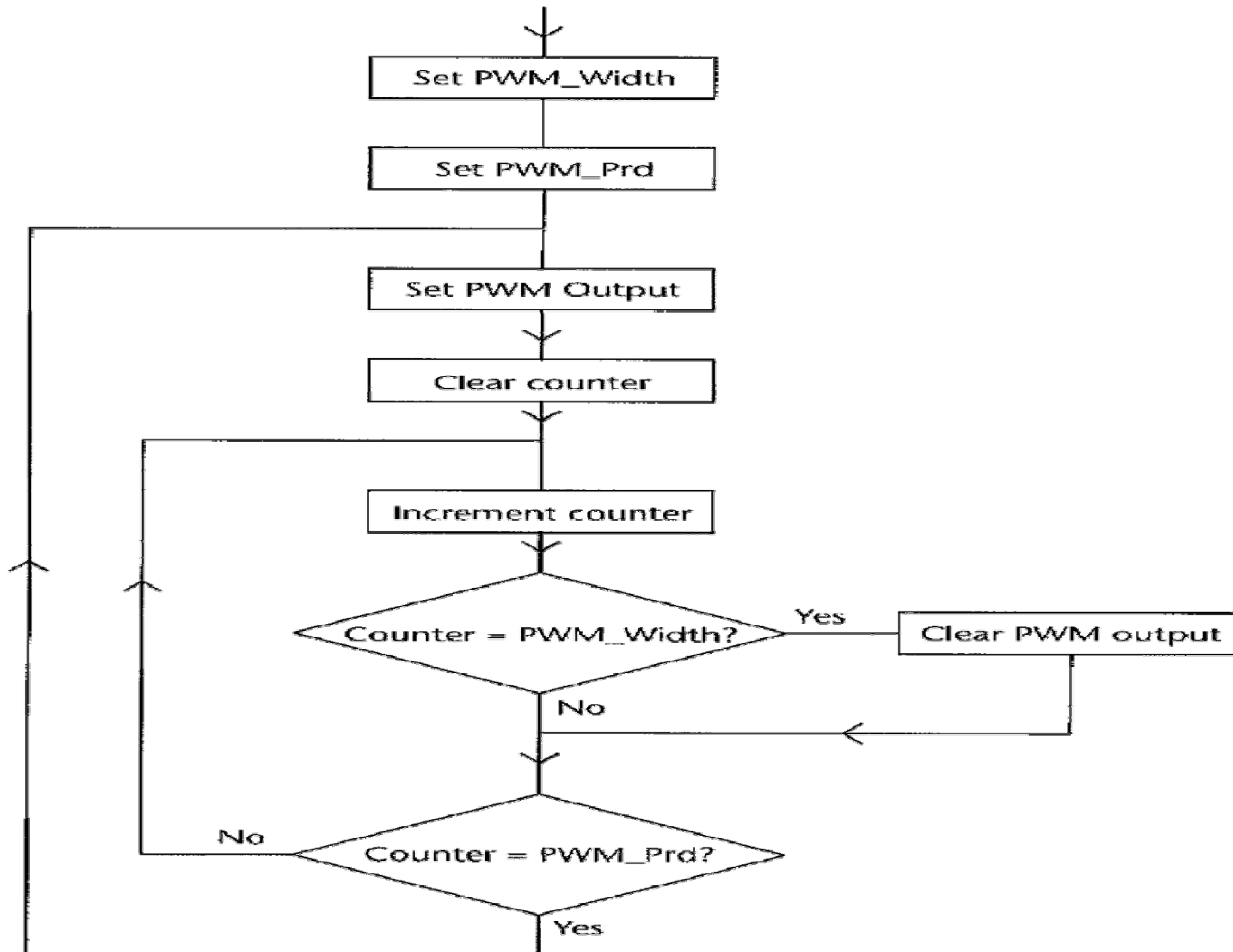
- In many applications, it is required to have a stream of pulses with controllable width/duration



- In embedded systems this can be done in software or hardware

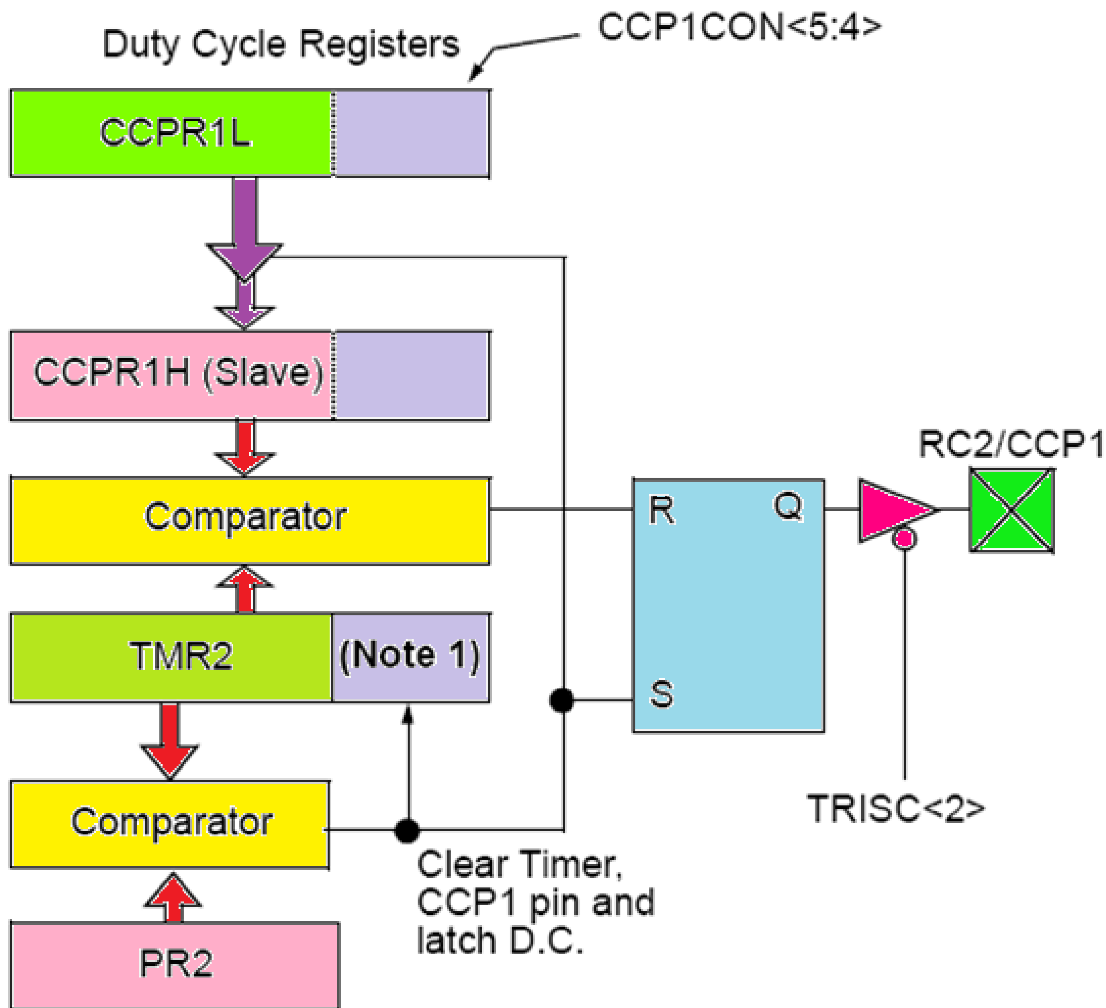
Capture/Compare/PWM Modules

Pulse Width Modulation (software)



Capture/Compare/PWM Modules

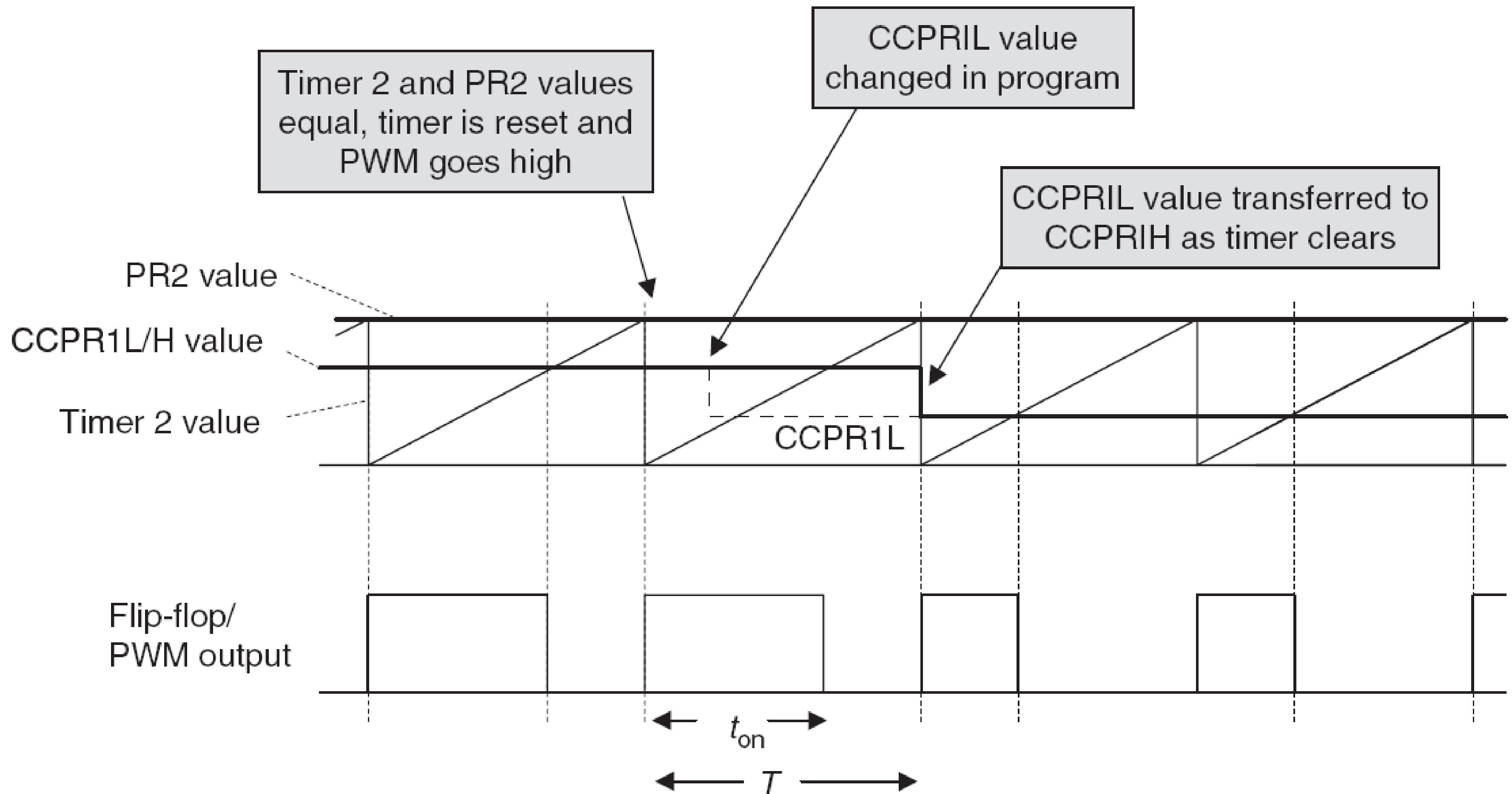
Pulse Width Modulation (using CCP module)



- **Note 1:** The 8-bit timer is concatenated with 2-bit internal Q clock, or 2 bits of the prescaler, to create 10-bit time base.

Capture/Compare/PWM Modules

Pulse Width Modulation (using CCP module)



Capture/Compare/PWM Modules

Pulse Width Modulation (using CCP module)

Calculations

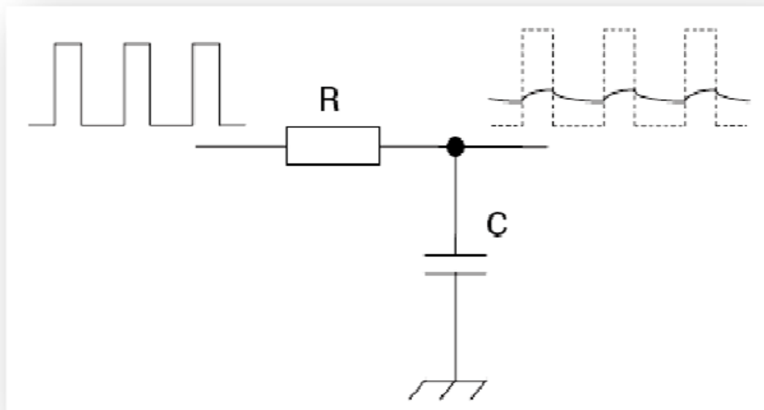
$$T = (\mathbf{PR2} + 1) \times (\text{Timer2 input clock period})$$
$$= (\mathbf{PR2} + 1) \times \{T_{osc} \times 4 \times (\text{Timer2 prescale value})\}$$

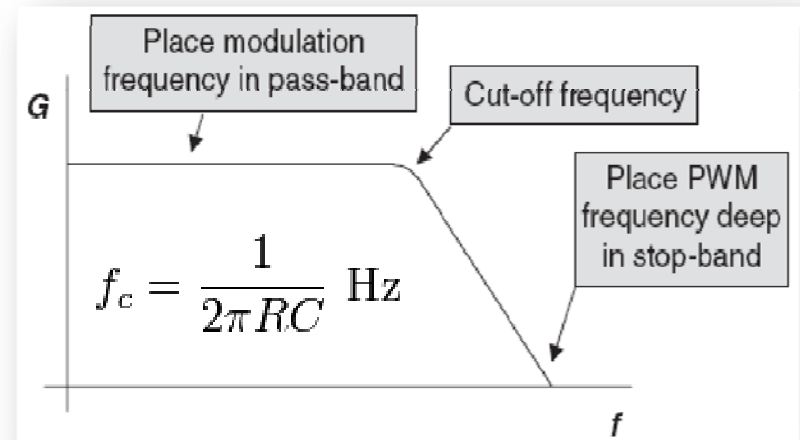
$$t_{on} = (\text{pulse width register}) \times (\text{PWM timer input clock period})$$
$$= (\text{pulse width register}) \times \{T_{osc} \times (\text{Timer2 prescale value})\}$$

pulse width register = ***CCPR1L :: CCP1CON<5:4>***

PWM and Digital To Analog Conversion

- PWM is perhaps primarily used for load control
- Can be used for simple and effective digital-to-analog conversion
  - Space-ratio is fixed
    - Low pass filter the PWM stream to obtain a DC signal with some ripple





  - Space-ratio is modulated
    - Varying output voltage is produced

PWM and Digital To Analog Conversion

- Generating a Sine Wave - change the on-time for the PWM signal so the output of the LPF will be different

```
        clrf   pointer
sin_loop
        movf   pointer,w
        call   sin_table    ;get most significant byte
        movwf  ccpr1l       ;move it to the PWM output
        incf   pointer,f    ;increment the pointer
        movf   pointer,w
        call   sin_table     ;get the MS byte
        andlw  B'11000000'  ;we only use ms 2 bits
```

PWM and Digital To Analog Conversion

- Generating a Sine Wave

```
movwf temp
bcf   status,c      ;adjust for CCP1CON
rrf   temp,f
rrf   temp,w
iorlw B'00001100' ;set some CCP1CON bits
movwf ccp1con
incf  pointer,f
movf  pointer,w
…
call  delay1
goto  sin_loop
```

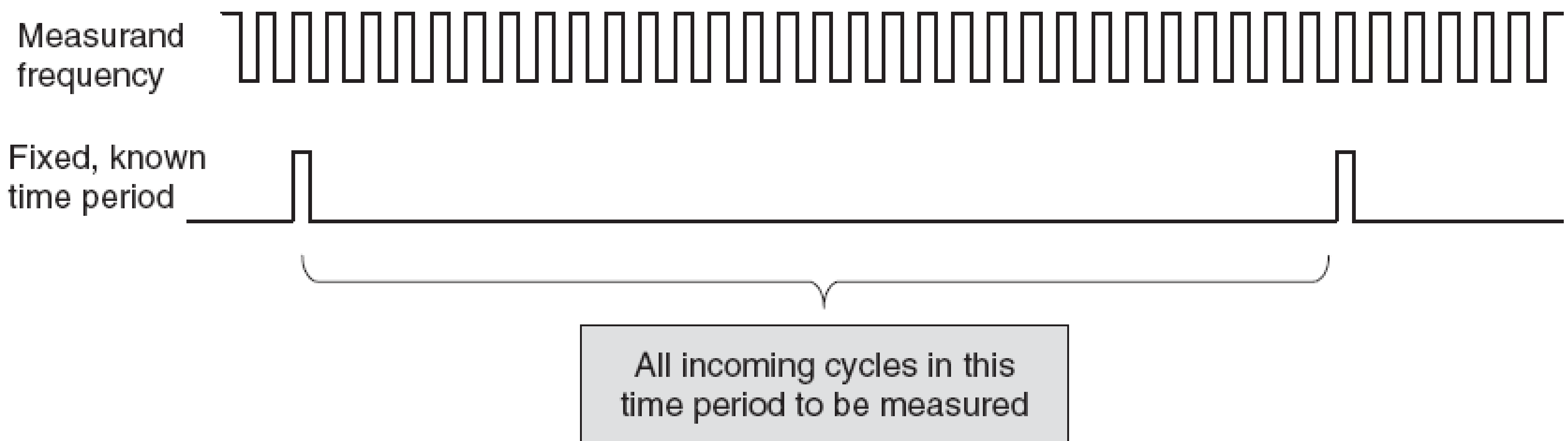
PWM and Digital To Analog Conversion

- Generating a Sine Wave

```
Sin_Table
        addwf pcl,1
        retlw 00     ;0 degrees, higher byte
        retlw 00     ;0 degrees, lower byte
        retlw 03     ;2 degrees, higher byte
        retlw 5A     ;2 degrees, lower byte
        retlw 06     ;4 degrees, higher byte
        retlw 0B2    ;4 degrees, lower byte
        ……
        ……
```

Frequency Measurement

- Frequency measurement is a very important application of both counting and timing
- Both a counter and a timer are needed
    - The timer to measure the reference period of time
    - The counter to count the number of events within that time.



Example 1

Write a program to flash a LED that is connected to RA0 continuously such that it is ON for 3 seconds and OFF for 3 seconds. Use TIMER1 module to generate the delay and assume Fosc = 4MHz.

TABLE 6-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

Example 1

- Maximum time that can be measured by TMR1 is

  Time = 2^16 * 4/Fosc * Prescaler

  = 65536*1usec*8 = 0.5243 s

- How about we configure TMR1 to measure 0.5 sec and use a software counter (post-scaler) to count six times

  0.5 = N * 1 usec * P → N = 62500, P = 8

  TMR1H:TMR1L = 65536 – 62500 = 3036 = 0x0BDC

  → TMR1H = 0x0B , TMR1L = 0xDC

- T1CON = 0x30

--	--	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	T1ON
0	0	1	1	0	0	0	0

Example 1

```
COUNTER     EQU       0x20
            # include "PIC16F877.INC"
            org       0x0000
            goto      START
            org       0x0004
ISR         goto      ISR
START       bcf       STATUS, RP1      ; select bank 1
            bsf       STATUS, RP0
            clrf      TRISA            ; set RA0 as output
            movlw     B'00000110'      ; configure RA0 as digital
            movwf     ADCON1
            bcf       STATUS, RP0
FLASH       movlw     0x06             ;  initialize counter to 6
            movwf     COUNTER
WAIT_3sec   movlw      0x0B
            movwf     TMR1H            ; initialize TMR1H
```

Example 1

```
                movlw    0xDC
                movwf    TMR1L               ; initialize TMR1L
                movlw    0x30
                movwf    T1CON               ; initialize T1CON
                bsf      T1CON, TMR1ON       ; enable timer 1
WAIT_p5sec      btfss    PIR1, TMR1IF        ; wait for overflow
                goto     WAIT_p5sec
                bcf      T1CON, TMR1ON       ; stop timer
                bcf      PIR1, TMR1IF        ; clear interrupt flag
                decfsz   COUNTER, F
                goto     WAIT_3sec
                movlw    0xFF                ; change the state of RA0
                xorwf    PORTA, F
                goto     FLASH
                end
```

Example 2

Consider the contents of the following registers

TMR2 = D'44'

PR2 = D'100'

T2CON = 0x39

If the instruction *bsf T2CON, T2ON* is executed, then how long does it take to set the TMR2IF in the PIR1 register ? Assume Fosc = 8 MHz.

Example 2

T2CON

--	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	T2ON	T2CKPS1	T2CKPS0
0	0	1	1	1	0	0	1

- Initially, the timer is off
- Executing the instruction enables the timer
- The time required to set the TMR2IF is

  Time = (PR2+1) * prescaler * postscaler * 4 /Fosc

  if TMR2 is initialized to zero.
- However, TMR2 = 44. So the time is

  Time = (PR2-TMR2+1) * 4 * 4/8MHz + (PR2+1) * 4 * 4/8MHz * 7 = 1528 usec

Example 3

Write a program that configures and uses the CCP1 module in PIC16F873A to generate a periodic square wave of frequency 50 Hz and 25% duty cycle. Assume that Fosc = 800 KHz.

Requirements

1) Configure RC2 as output
2) Configure TIMER2 module and compute the values to be placed in CCPR1L and PR2 registers which determine the duty cycle and the cycle time, respectively
3) Turn on the timer

Example 3

TABLE 8-5: REGISTERS ASSOCIATED WITH PWM AND TIMER2

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- ---0	---- ---0
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- ---0	---- ---0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
92h	PR2	Timer2 Module's Period Register								1111 1111	1111 1111
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PWM and Timer2.

Example 3

- PWM signal specs
  - T = 1 /50 = 0.02 sec
  - Ton = 0.25 * T = 0.005 sec
- Need to configure the CCP1 and TIMER2
  - PR2 register

    T = (PR2+1) * 4 * Tosc * prescaler

    if we assume prescaler = 16, then PR2 = 249
  - Pulse-width register CCPR1L:CCP1CON<5:4>

    Ton = PWR * Tosc * prescaler

    already the prescaler is chosen to be 16 → PWR = 250 =0xFA

    → CCPR1L = B'00111110' and CCP1CON<5:4> = B'10'
  - T2CON = 0x06
  - CCP1CON = B'00101100'

Example 3

```
                # include    "PIC16F877.INC"
                org        0x0000
                goto       START
                org        0x0004
ISR             goto       ISR
START           bcf        STATUS, RP1       ; select bank 1
                bsf        STATUS, RP0
                bcf        TRISC, 2          ; set RC2 as output
                movlw      D'249'
                movwf      PR2               ; set the cycle time in PR2
                bcf        STATUS, RP0
                movlw      0x3E
                movwf      CCPR1L            ; set the ON time in CCPR1L
                bcf        CCP1CON, 4        ; specify the LSBs  of the  ON time
                bsf        CCP1CON, 5
```

Example 3

```
        bsf     CCP1CON, 3
        bsf     CCP1CON, 2  ; configure  CCP1 in PWM and
        movlw   0x06
        movwf   T2CON       ; configure timer 2 and enable it


DONE    goto    DONE
        end
```

Summary

- Timing is essential element of embedded systems design
- Wide range of timers is available in PIC microcontrollers with clever add-on features such as capture, compare, and pulse width modulation
- It is very occasional to have several timers running simultaneously in an embedded system

Smarter systems and the PIC® 18FXX2

Chapter 12
Sections 1 – 5

Dr. Iyad Jafar

Outline

- Introduction of 18 Series
- 18 Series Architecture
  - Pin Out
  - Block Diagram
  - Status Register
- 18 Series Instruction Set
- 18 Series Memory Organization
  - Data Memory
  - Stack
  - Program Memory
  - Configuration Words
- Summary

PIC Families

PIC Family	Stack Size	Instruction Word Size	No of Instructions	Interrupt Vectors
12CX/12FX	2	12- or 14-bit	33	None
16C5X/16F5X	2	12-bit	33	None
16CX/16FX	8	14-bit	35	1
17CX	16	16-bit	58	4
18CX/18FX	**32**	**16-bit**	**75**	**2**

The PIC 18 Series

Similarities with the PIC 16 Series

- RISC, pipelined, 8-bit CPU, with single Working (W) and Status registers
- Many peripherals identical or very similar
- Similar packages and pinouts
- Many Special Function Register (SFR) and bit names unchanged
- All but one of the 16 Series instructions are part of the 18 Series instruction set
- Instruction cycle made up of four oscillator cycles

The PIC 18 Series

Differences from PIC 16 Series

- The number of instructions more than doubled
- 16-bit instruction word
- Enhanced Status register (overflow and negative bits)
- Hardware 8 × 8 multiply
- More external interrupts
- Two prioritized interrupt vectors
- Radically different approach to memory structures, with increased memory size
- Enhanced address generation for program and data memory
- Bigger Stack, with some user access and control
- Phase-locked loop (PLL) clock generator.

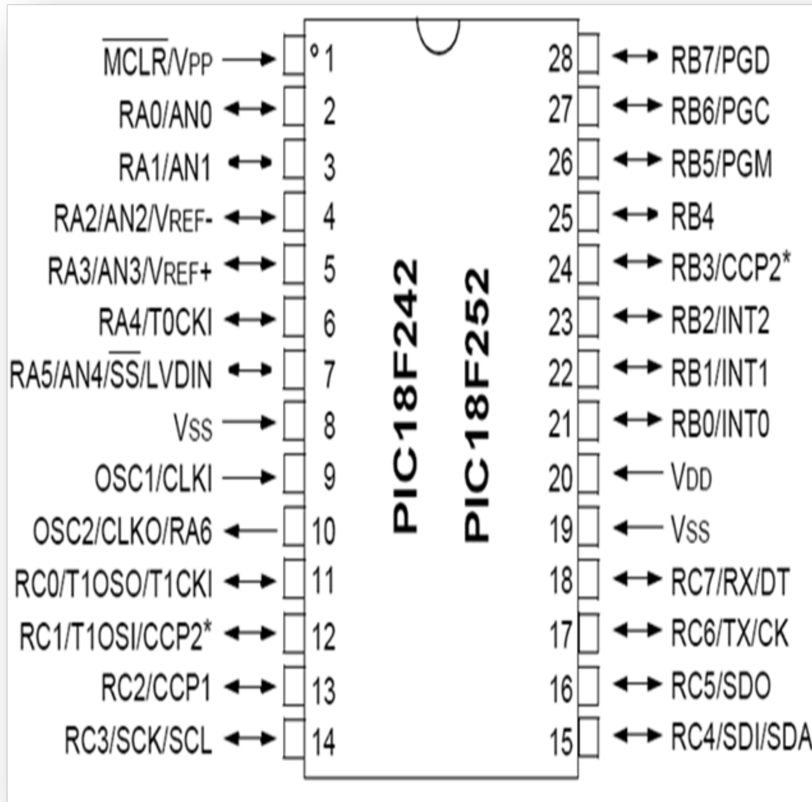
The PIC 18 Series

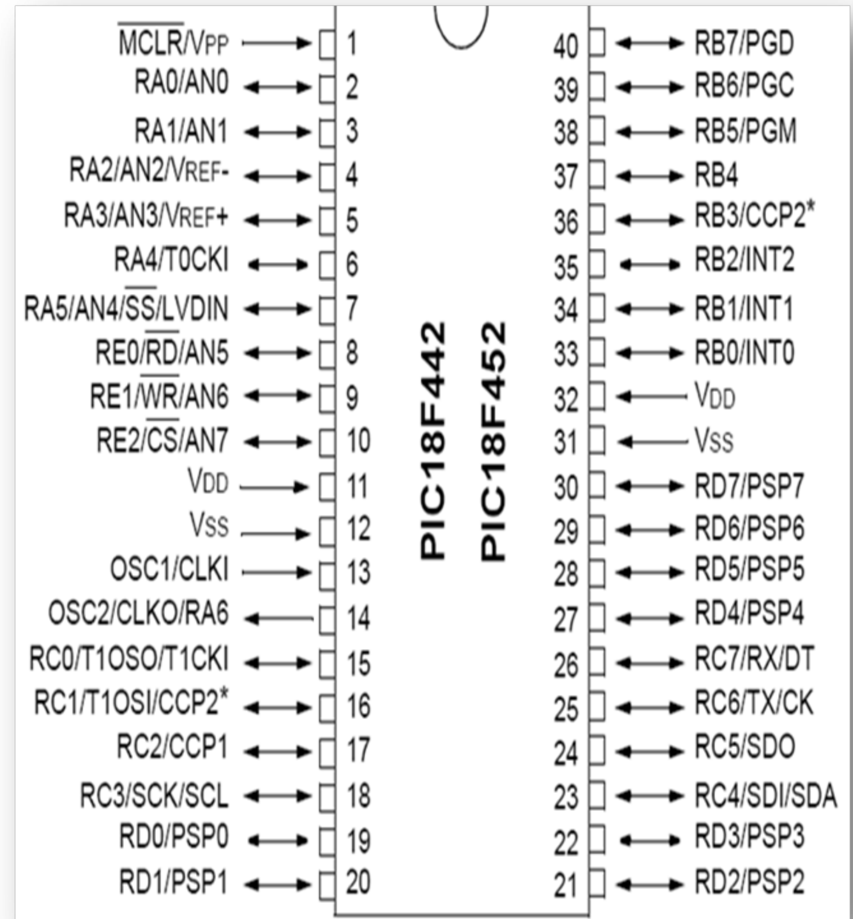
All 18FXX2 Sub-family Devices

Features	PIC18F242	PIC18F252	PIC18F442	PIC18F452
Operating Frequency	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz
Program Memory (Bytes)	16K	32K	16K	32K
Program Memory (Instructions)	8192	16384	8192	16384
Data Memory (Bytes)	768	1536	768	1536
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	17	17	18	18
I/O Ports	Ports A, B, C	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART
Parallel Communications	—	—	PSP	PSP
10-bit Analog-to-Digital Module	5 input channels	5 input channels	8 input channels	8 input channels
RESETS (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions	75 Instructions	75 Instructions	75 Instructions
Packages	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin SOIC	40-pin DIP 44-pin PLCC 44-pin TQFP	40-pin DIP 44-pin PLCC 44-pin TQFP

18FXX2 Sub-Family

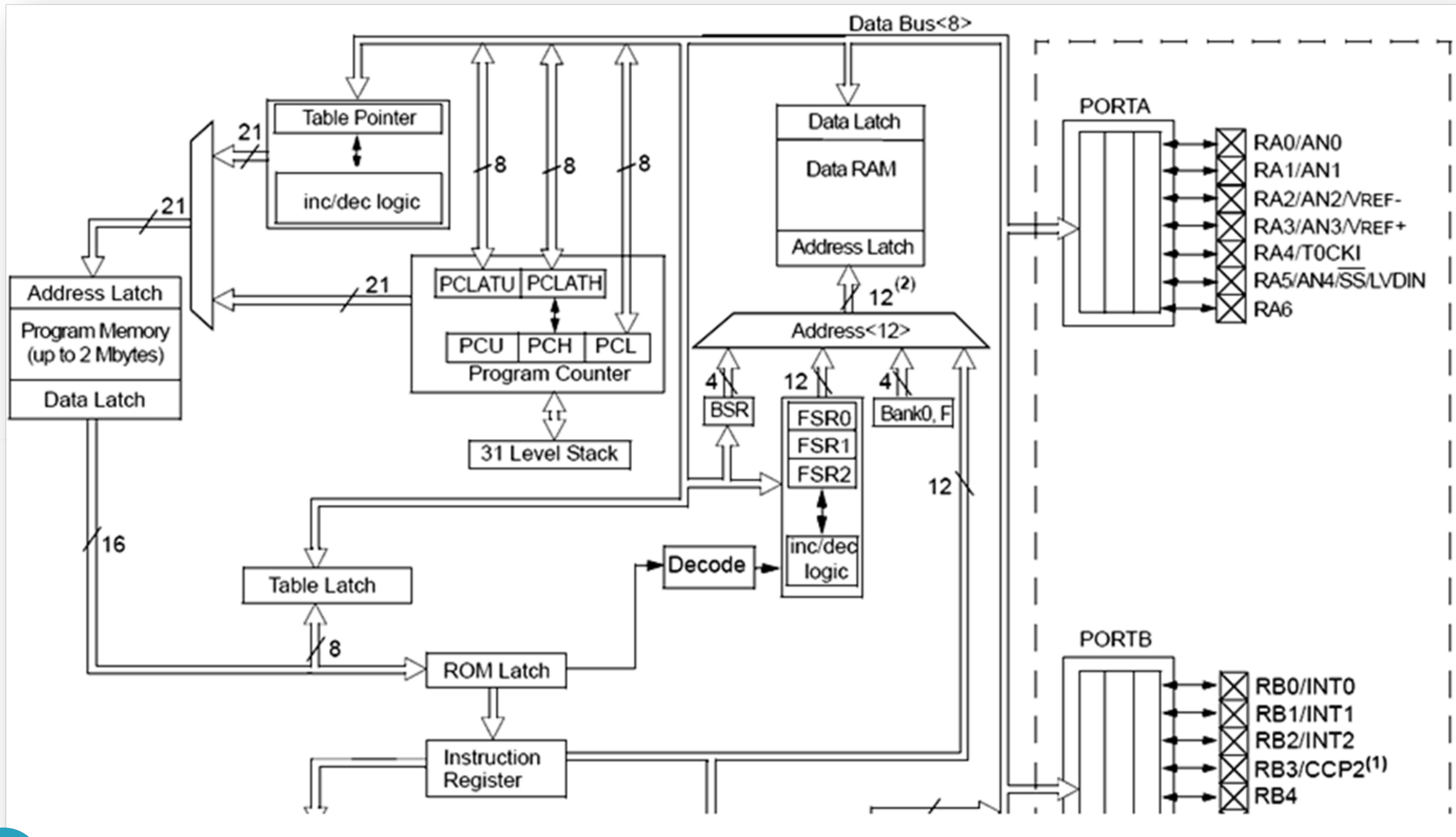
Pinout





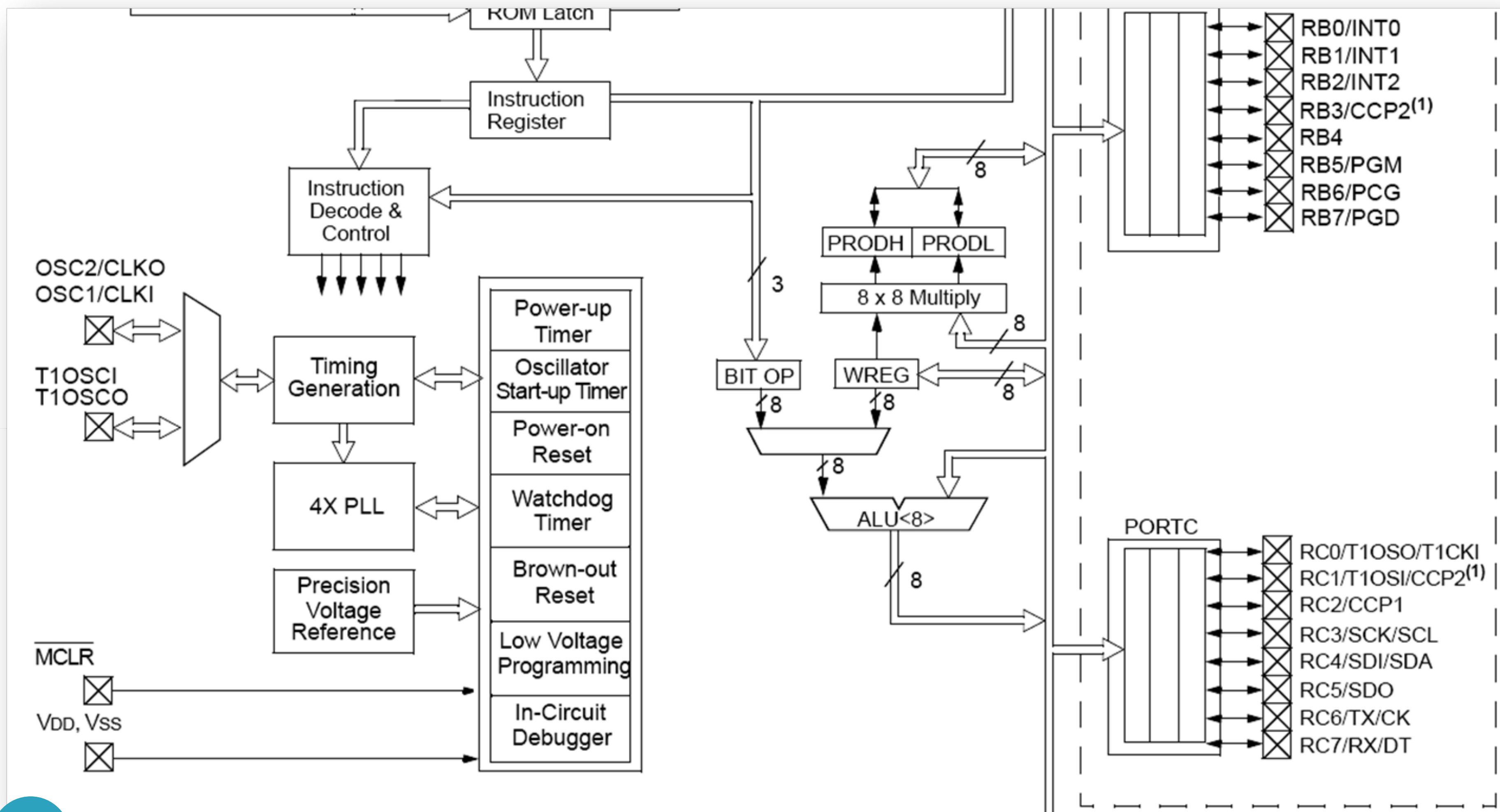
18FXX2 Sub-family

Internal Organization 1/3



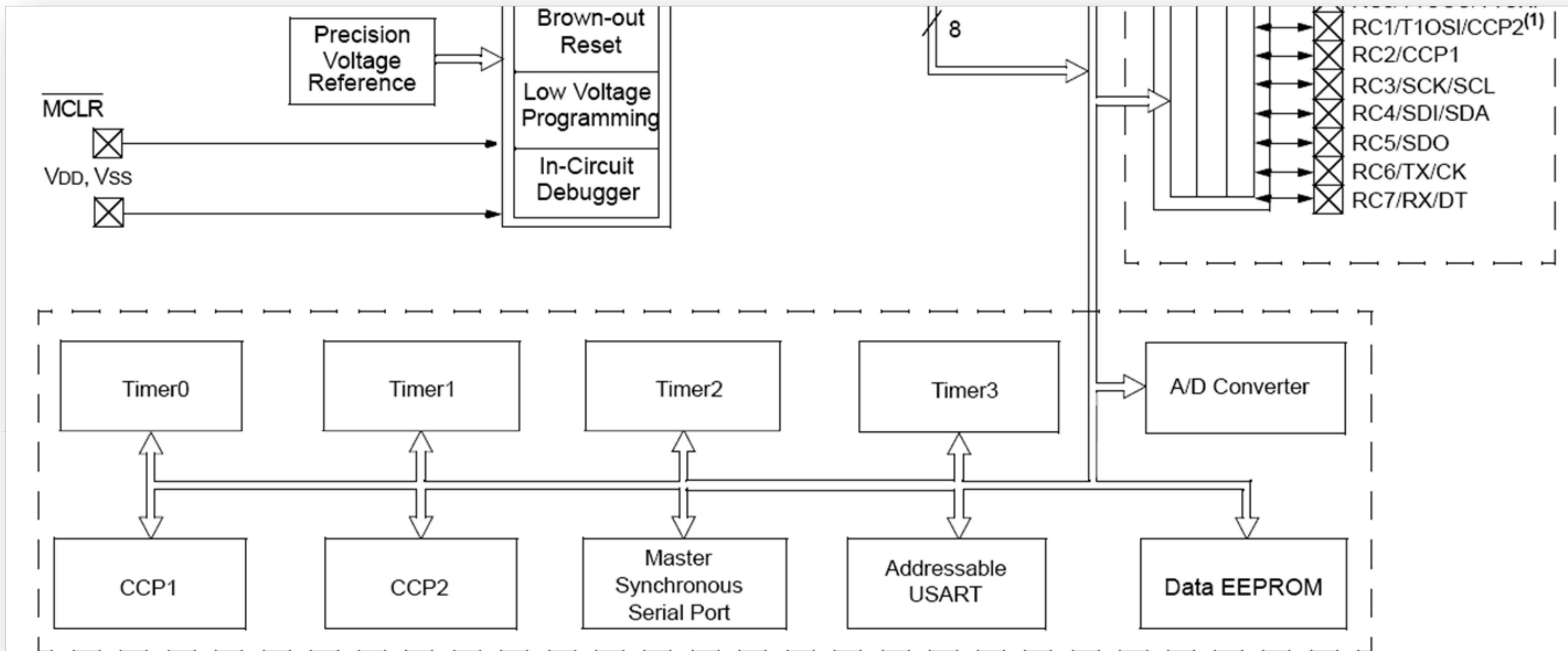
18FXX2 Sub-family

Internal Organization 2/3



18FXX2 Sub-family

Internal Organization 3/3



18FXX2 Sub-family

STATUS Register

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC	C
bit 7							bit 0

- **C**: Carry/Borrow’
- **DC**: Digit Carry/Borrow’
- **Z**: Zero
- **OV**: Overflow
- **N**: Negative

18FXX2 Sub-family

Instruction Set 1/5

16 Series instruction	18 Series equivalents	Description
Byte-oriented file register operations		
addwf f,d	**addwf f,d,a** **addwfc f,d,a**	Add W and f Add W and f with Carry
andwf f,d	**andwf f,d,a**	And W and f
clrf f	**clrf f,a**	Clear f
clrw	–	Clear W
comf f,d	**comf f,d,a**	Complement f
– – –	**cpfseq f,a** **cpfsgt f,a** **cpfslt f,a**	Compare f with W, skip if equal Compare f with W, skip if greater than Compare f with W, skip if less than
decf f,d	**decf f,d,a**	Decrement f
decfsz f,d –	**decfsz f,d,a** **decfsnz f,d,a**	Decrement f, skip if zero Decrement f, skip if not zero
incf f,d	**incf f,d,a**	Increment f
incfsz f,d	**incfsz f,d,a** **incfsnz f,d,a**	Increment f, skip if zero Increment f, skip if not zero

18FXX2 Sub-family

Instruction Set 2/5

iorwf f,d	**iorwf f,d,a**	Inclusive OR f with W
movf f,d –	**movf f,d,a** **movff f_s,f_d**	Move f Move source file f_s to destination file f_d
movwf f	**movwf f,a**	Move W to f
nop	**nop** **nop**	No operation – an intentional instruction The second word of a two-word instruction, which is encoded to execute as a **nop** if it is accidentally interpreted as an instruction
–	**mulwf f,a**	Multipy W and f
–	**negf f,a**	Negate f
rlf f,d	**rlfc f,d,a** **rlnfc f,d,a**	Rotate left through Carry Rotate left, no Carry
rrf f,d	**rrcf f,d,a** **rrncf f,d,a**	Rotate right through Carry Rotate right, no Carry
–	**set f**	Set f
subwf f,d	**subwf f,d,a** **subwfb f,d,a**	Subtract W from f Subtract W from f with borrow
–	**subfwb f,d,a**	Subtract f from W with borrow
swapf f,d	**swapf f,d,a**	Swap nibbles in f
–	**tstfsz f,a**	Test f, skip if zero

18FXX2 Sub-family

Instruction Set 3/5

xorwf f,d	**xorwf f,d,a**	Exclusive OR W with f
Bit-oriented file register operations		
bcf f,b	**bcf f,b,a**	Clear bit b in register f
bsf f,b	**bsf f,b,a**	Set bit b in register f
–	**btg f,d,a**	Toggle bit b in register f
btfsc f,b	**btfsc f,b,a**	Test bit b in f, skip if clear
btfss f,b	**btfss f,b,a**	Test bit b in f, skip if set
Literal operations		
addlw k	**addlw k**	Add literal to W
andlw k	**andlw k**	And literal with W
iorlw k	**iorlw k**	Inclusive OR literal with W
movlw k	**movlw k**	Move literal to W
–	**movlb**	Move literal to BSR
–	**lfsr f,k**	Load FSR **f** with 12-bit literal **k**
–	**mullw**	Multiply literal with W
sublw k	**sublw k**	Subtract W from literal
xorlw k	**xorlw k**	Exclusive OR literal with W

18FXX2 Sub-family

Instruction Set 4/5

Control operations		
call k	**call n,s** **rcall n**	Call subroutine, with (s = 1) or without (s = 0) saving context to Stack Relative call to subroutine
clrwdt	**clrwdt**	Clear Watchdog Timer
–	**daw**	Decimal adjust W
goto k	**goto n**	Go to absolute address, where **k/n** address anywhere in program memory space
–	**pop**	Pop top of return stack (TOS)
–	**push**	Push top of return stack (TOS)
–	**reset**	Software reset
retfie	**retfie s**	Return from interrupt, with (s = 1) or without (s = 0) retrieving context from Stack
retlw k	**retlw k**	Return with literal in W
return	**return s**	Return from subroutine, with (s = 1) or without (s = 0) retrieving context from Stack
sleep	**sleep**	Go into standby mode
–	**bc, bn, bnc, bnn, bnov, bnz, bov, bz**	Eight conditional branch instructions, one for each state of Status register bits **N**, **OV**, **Z**, **C**, all with 8-bit twos complement relative address **n**
–	**bra n**	Branch unconditionally 8-bit twos complement relative address **n**

18FXX2 Sub-family

Instruction Set 5/5

Program memory Table Read/Write operations		
–	**tblrd*, tblrd*+, tblrd*–, tblrd+***	Four Table Read instructions, with pointer change respectively no change, post-increment, post-decrement, pre-increment
–	**tblwt*, tblwt*+, tblwt*–, tblwt+***	Four Table Write instructions, with pointer change respectively: no change, post-increment, post-decrement, pre-increment

18FXX2 Sub-family

Instruction encoding 1/4

Mnemonic,	Operands	Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da0	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	0da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word f_d (destination) 2nd word	2	1100 1111	ffff ffff	ffff ffff	ffff ffff	None	
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	1, 2
RLCF	f, d, a	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	1, 2
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	100a	ffff	ffff	None	
SUBFWB	f, d, a	Subtract f from WREG with borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	
SUBWFB	f, d, a	Subtract WREG from f with borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	1, 2
SWAPF	f, d, a	Swap nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	

18FXX2 Sub-family

Instruction encoding 2/4

BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, d, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

4: Some instructions are 2-word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

18FXX2 Sub-family

Instruction encoding 3/4

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	2	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	1 (2)	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call subroutine1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to address1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	4
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device RESET	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	

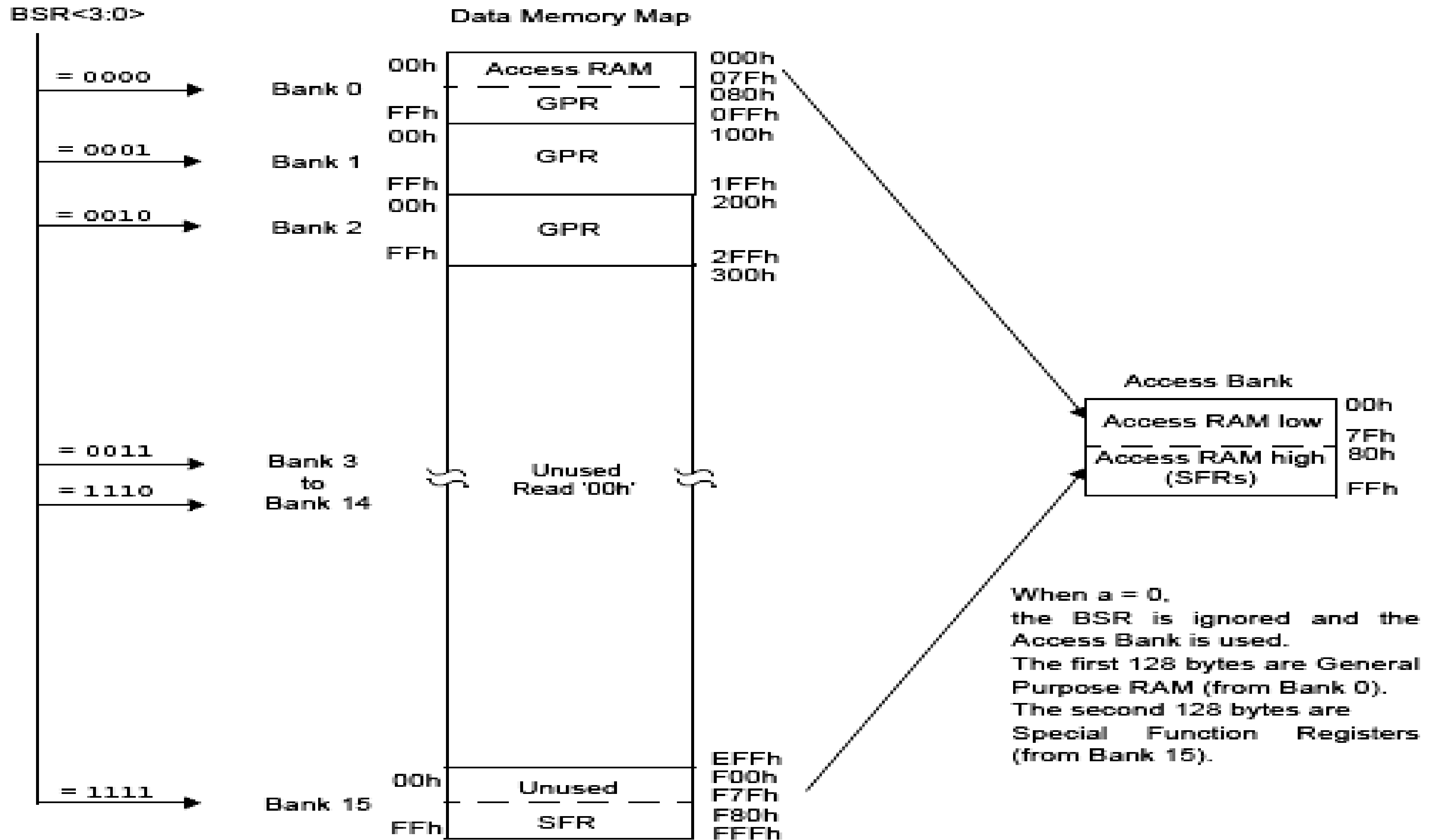
18FXX2 Sub-family

Instruction encoding 4/4

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
LITERAL OPERATIONS									
ADDLW	k	Add literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR	f, k	Move literal (12-bit) 2nd word	2	1110	1110	00ff	kkkk	None	
		to FSRx 1st word		1111	0000	kkkk	kkkk		
MOVLB	k	Move literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOVLW	k	Move literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW	k	Multiply literal with WREG	1	0000	1101	kkkk	kkkk	None	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW	k	Subtract WREG from literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS									
TBLRD*		Table Read	2	0000	0000	0000	1000	None	
TBLRD*+		Table Read with post-increment		0000	0000	0000	1001	None	
TBLRD*-		Table Read with post-decrement		0000	0000	0000	1010	None	
TBLRD+*		Table Read with pre-increment		0000	0000	0000	1011	None	
TBLWT*		Table Write	2 (5)	0000	0000	0000	1100	None	
TBLWT*+		Table Write with post-increment		0000	0000	0000	1101	None	
TBLWT*-		Table Write with post-decrement		0000	0000	0000	1110	None	
TBLWT+*		Table Write with pre-increment		0000	0000	0000	1111	None	

18FXX2 Sub-family

Data Memory Map



18FXX2 Sub-family

Data Memory Map

- **12-bit** address, up to **4096** bytes
- **Banking**
  - Up to 16 banks with each bank having 256 locations
  - Bank selection is done using BSR<3:0>. BSR can be loaded directly using MOVLB instruction
  - SFRs are stored in the upper 128 bytes of Bank15
- **Access RAM**
  - Defined by virtually merging the lower 128 bytes of Bank0 with the upper 128 bytes (SFR) of Bank15
  - Memory access is directed to this bank if the a operand in the instruction is set to 0
  - This arrangement saves software overheads (mapped access of SFRs in Bank15 and GPRs in Bank0)

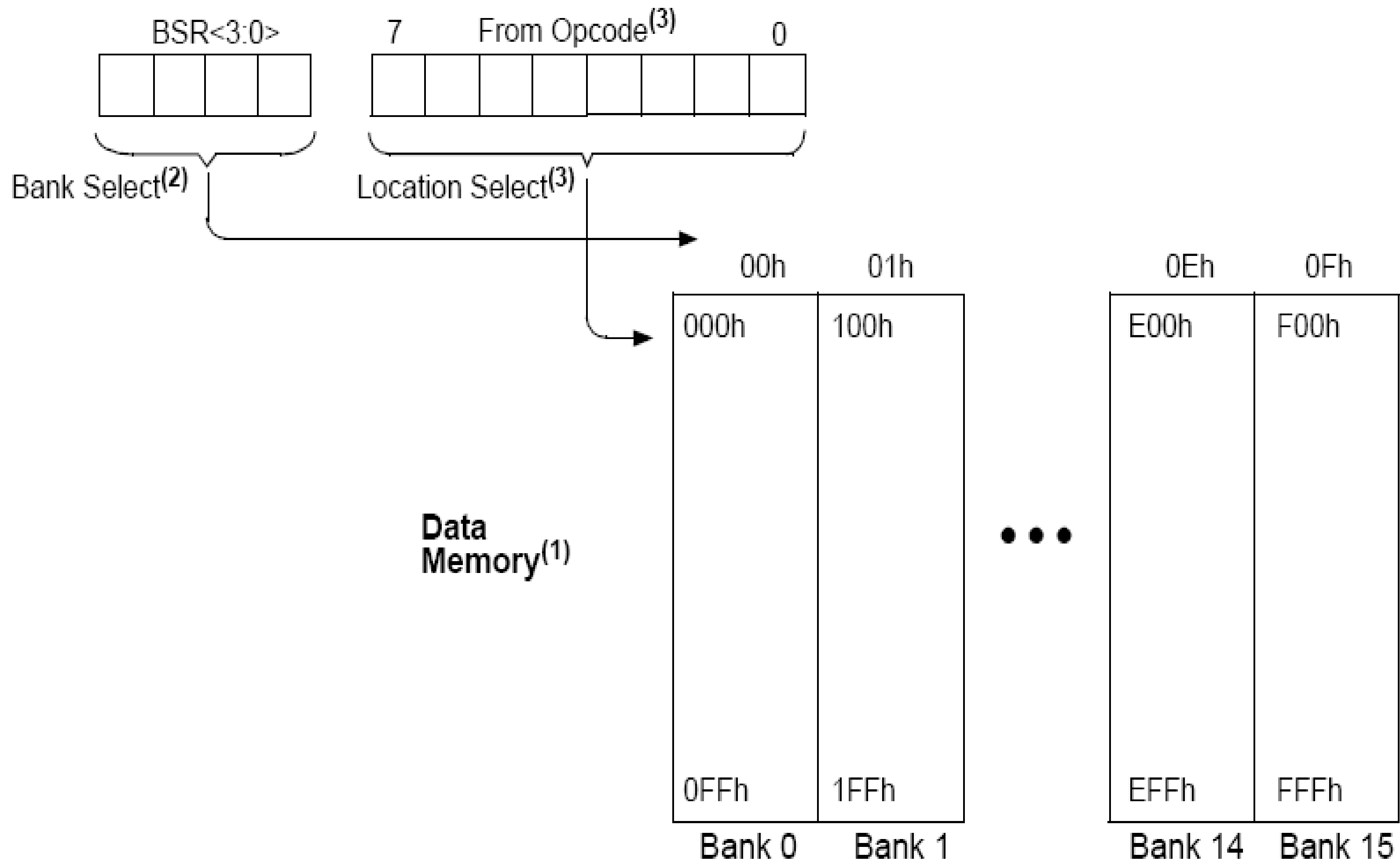
18FXX2 Sub-family

Data Memory Map

Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2[3]	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2[3]	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2[3]	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2[3]	FBCh	CCPR2H	F9Ch	—
FFBh	PCLATU	FDBh	PLUSW2[3]	FBBh	CCPR2L	F9Bh	—
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—
FF9h	PCL	FD9h	FSR2L	FB9h	—	F99h	—
FF8h	TBLPTRU	FD8h	STATUS	FB8h	—	F98h	—
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	—	F97h	—
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	—	F96h	TRISE[2]
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD[2]
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	LVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—
FF0h	INTCON3	FD0h	RCON	FB0h	—	F90h	—
FEFh	INDF0[3]	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—
FEEh	POSTINC0[3]	FCEh	TMR1L	FAEh	RCREG	F8Eh	—
FEDh	POSTDEC0[3]	FCDh	T1CON	FADh	TXREG	F8Dh	LATE[2]
FECh	PREINC0[3]	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD[2]
FEBh	PLUSW0[3]	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	—	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—
FE7h	INDF1[3]	FC7h	SSPSTAT	FA7h	EECON2	F87h	—
FE6h	POSTINC1[3]	FC6h	SSPCON1	FA6h	EECON1	F86h	—
FE5h	POSTDEC1[3]	FC5h	SSPCON2	FA5h	—	F85h	—
FE4h	PREINC1[3]	FC4h	ADRESH	FA4h	—	F84h	PORTE[2]
FE3h	PLUSW1[3]	FC3h	ADRESL	FA3h	—	F83h	PORTD[2]
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	—	FA0h	PIE2	F80h	PORTA

18FXX2 Sub-family

Direct Addressing



18FXX2 Sub-family

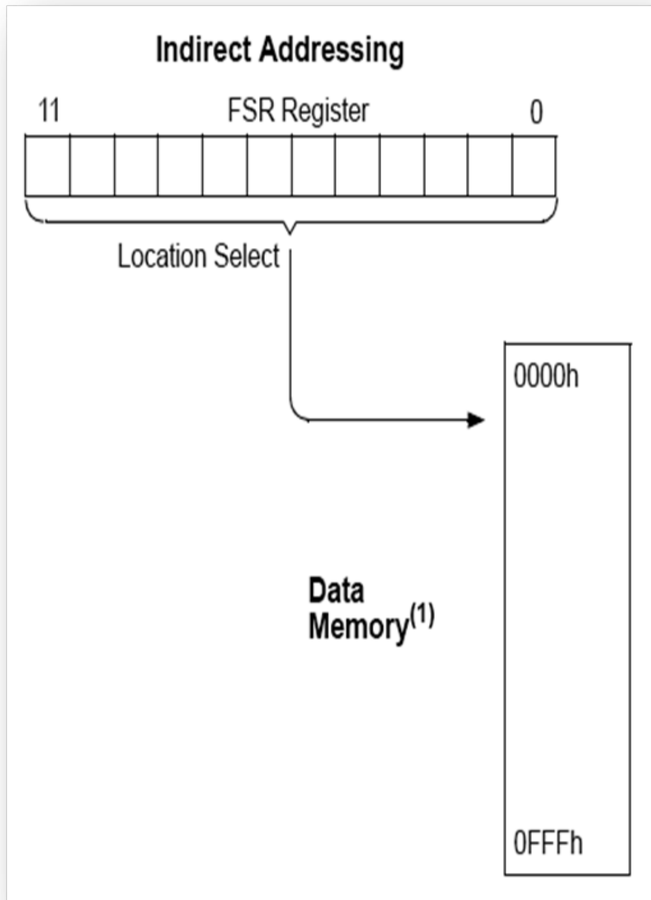
Virtual Registers Used in Indirect Addressing

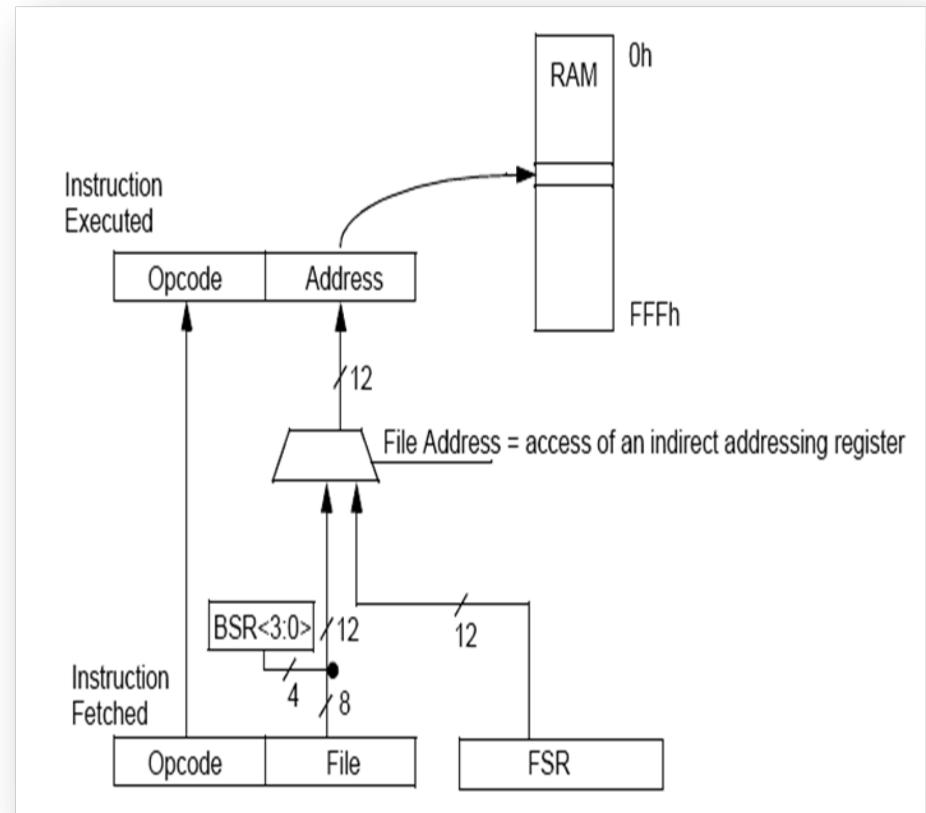
- Registers
  - FSRnH::FSRnL (12 bits)
  - FSRs can be loaded directly using LFSR instruction
  - Accessed by INDFn, POSTINCn, POSTDECn, PREINCn and PLUSWn
  - All these are not physical registers

'Virtual' register addressed $n = 0, 1$ or 2	**Action following instruction invoking FSR**
INDF*n*	No change to FSR*n*
POSTINC*n*	The FSR is automatically incremented following access
POSTDEC*n*	The FSR is automatically decremented following access
PREINC*n*	The FSR is automatically incremented preceding access
PLUSW*n*	The value in WREG is added to FSR*n*, to form indirect address. Neither FSR nor WREG are changed

18FXX2 Sub-family

Virtual Registers Used in Indirect Addressing

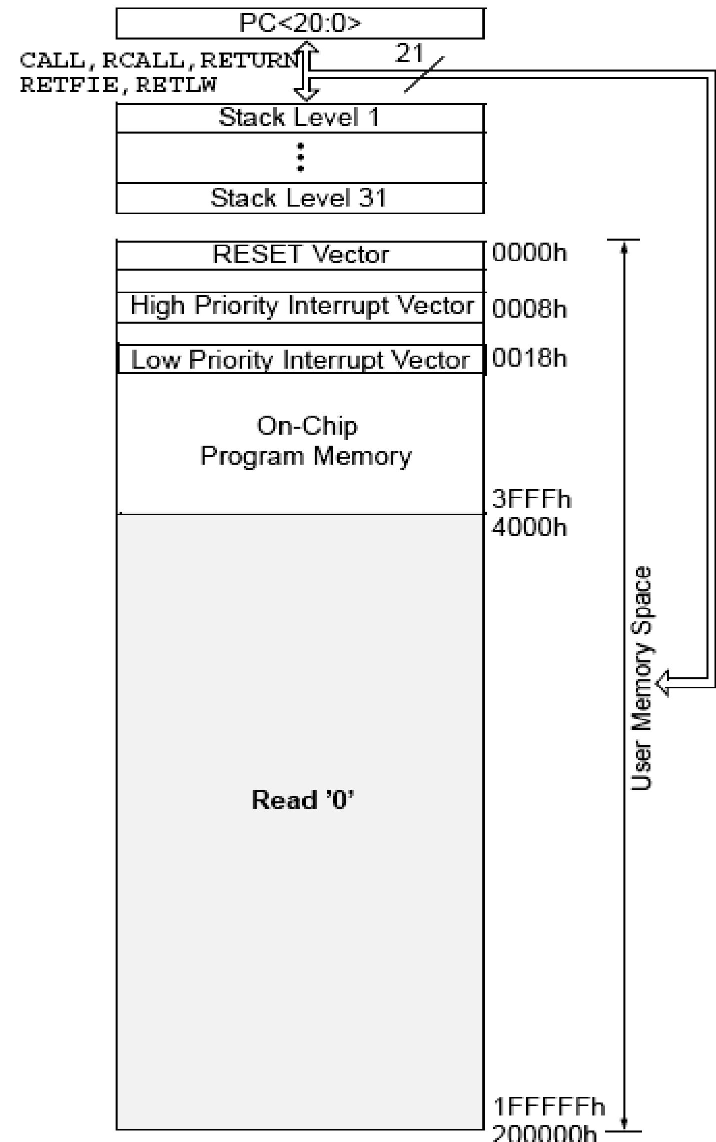




18FXX2 Sub-family

Program Memory

- 21-bit address bus. Up to 2 MB
- Byte-addressable , little endian
- Reset vector 0x0000
- Priority interrupts 0x0008 and 0x0018
- Program counter
  - 21 bits
  - PCU::PCH::PCL
  - Address the bytes
  - LSB is fixed to zero and it increments by two



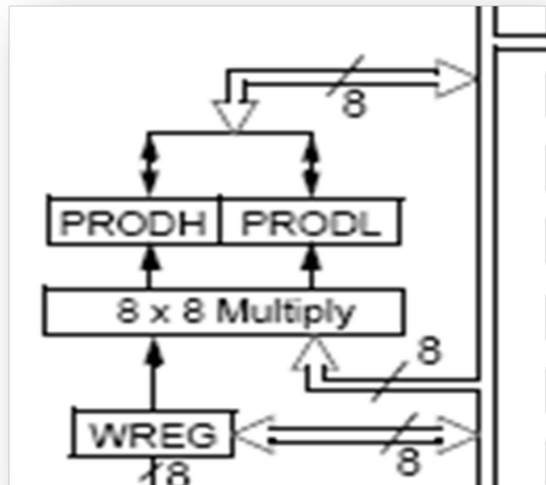
18FXX2 Sub-family

Program Memory

- **Stack**
  - 31-level stack
  - Stack pointer is implemented and it is readable and writable (STKPTR register)
  - Top of stack is readable and writeable TOSU::TOSH::TOSL
  - Allows the implementation of software stack
  - Fast Register Stack
    - Used to store the WREG, STATUS, BSR registers on interrupts
    - Pushed values are reloaded back if the Fast RETURN instruction (RETFIE s) is used at the end of the interrupt routine
    - This feature is available for subroutines if the Fast Call and Return instructions are used (CALL n,s , Return s)

Hardware Multiplier

- An 8 x 8 hardware multiplier is included in the ALU of the PIC18FXX2 devices
- By making the multiply a hardware operation, it completes in a single instruction cycle.
- This is an **unsigned** multiply that gives a 16-bit result stored into the 16-bit product register pair PRODH:PRODL
- Signed and 16-bit multiplication can be programmed in software



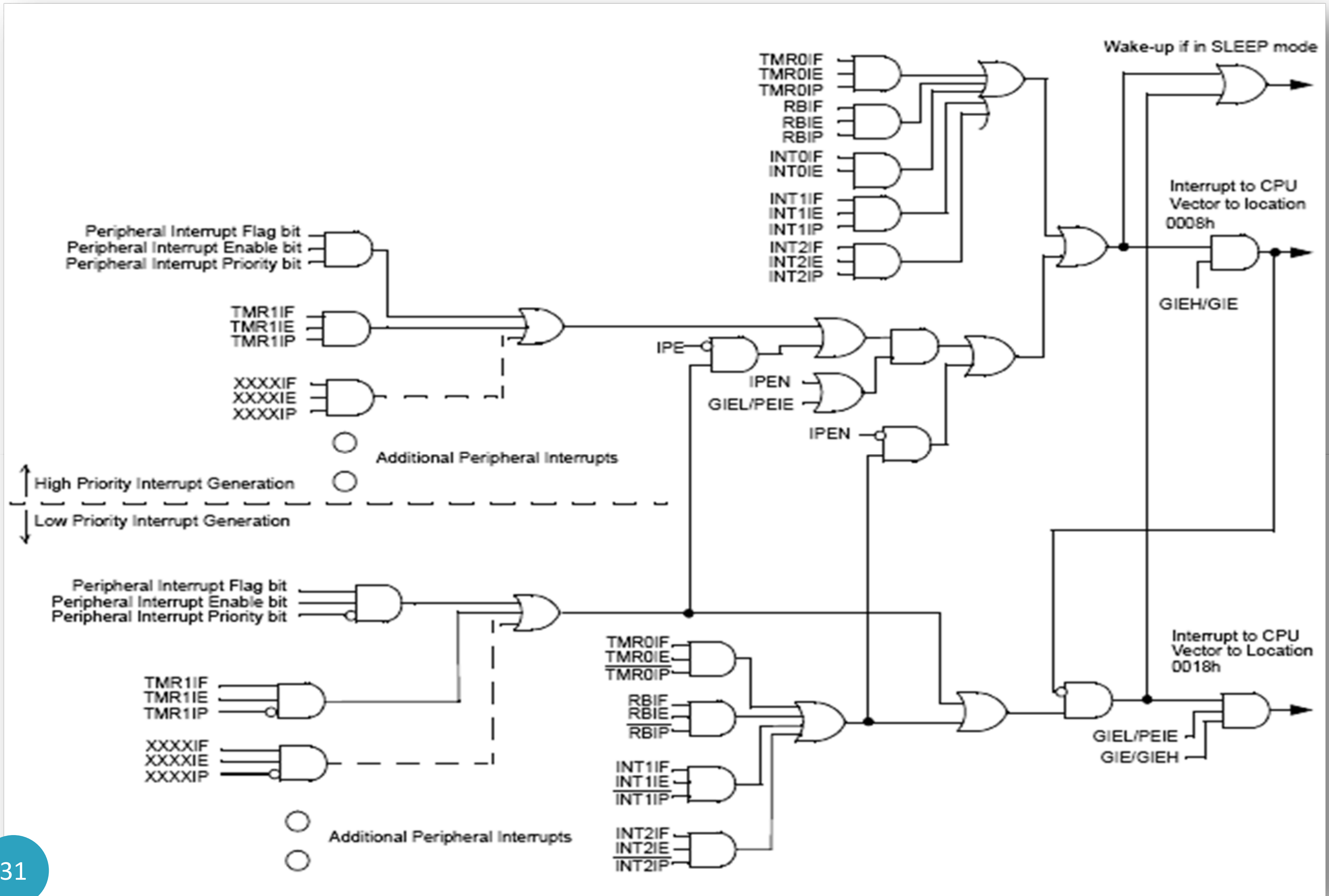

```
MOVF      ARG1, W      ;
MULWF     ARG2         ; ARG1 * ARG2 ->
                       ;   PRODH:PRODL
```

Interrupts

- The PIC18FXX2 devices have multiple interrupt sources
- 10 registers are used to control interrupts

  RCON, INTCON, INTCON2, INTCON3 , PIR1, PIR2, PIE1, PIE2 , IPR1, IPR2
- Interrupt priority feature using the IPEN bit RCON<7>
  - IPEN = 0 → Priority disabled and enabling interrupts is done using GIE and PEIE bits
  - IPEN = 1 → Priority enabled
    - High priority interrupts can interrupt low priority interrupts
    - GIEH bit enable high priority interrupts (interrupt vector 0x0008)
    - GIEL bit enable low priority interrupts (interrupt vector 0x0018)
  - All interrupt sources except INT0 have priority bit (IPR1 and IPR2)

Interrupts



Summary

- The 18 Series microcontrollers represent a very clear step forward in the PIC design strategy. The CPU and memory structure are radically redeveloped, while many peripherals are retained.
- The instruction set is increased to 75 distinct instructions, with big new capability in arithmetic, program branching, table access and memory usage.
- Data memory is structured to give a much greater RAM capacity and a separate grouping of Special Function Registers.
- Program memory has greatly increased capacity, with larger address bus, and the 16-bit instructions are now split into 2 bytes for storage. The Stack is deeper and more flexible.