# Re-create yourself

QUEST
INNOVATIVE SOLUTIONS

# PIC MICROCONTROLLER

# PIC

# Peripheral Interface Controller

## Microcontrollers from Microchip Technology Inc.

Microchip Technology Inc. is a leading provider of microcontroller and analog semiconductors, providing low-risk product development, lower total system cost and faster time to market for thousands of diverse customer applications worldwide. Headquartered in Chandler, Arizona, Microchip offers outstanding technical support along with dependable delivery and quality.

PIC

# Classification

PIC devices are grouped by the size of their instruction word length

Base line: 12 bit instruction  word length.
Mid-range:14 bit instruction word length.
High-end:  16 bit instruction word length.

# Device Structure

Each part of PIC can be placed into one of the three groups

- Core
- Peripherals
- Special Features

# The Core

The core includes the basic features that are required to make the device operate. These include

✓CPU (Central Processing Unit) operation
✓ALU (Arithmetic Logical Unit) operation
✓Device memory map organization
✓Device Oscillator
✓Reset logic
✓Interrupt operation
✓Instruction set

# Peripherals

Peripherals are the features that add a differentiation from a microprocessor. These include

- ✓ General purpose I/O
- ✓ Timer0
- ✓ Timer1
- ✓ Timer2
- ✓ Capture, Compare, and PWM (CCP)
- ✓ Synchronous Serial Port (SSP)
- ✓ USART
- ✓ Analog to Digital (A/D) Converter

# Special Features

Special features are the unique features that help to

- ✓ Decrease system cost
- ✓ Increase system reliability
- ✓ Increase design flexibility

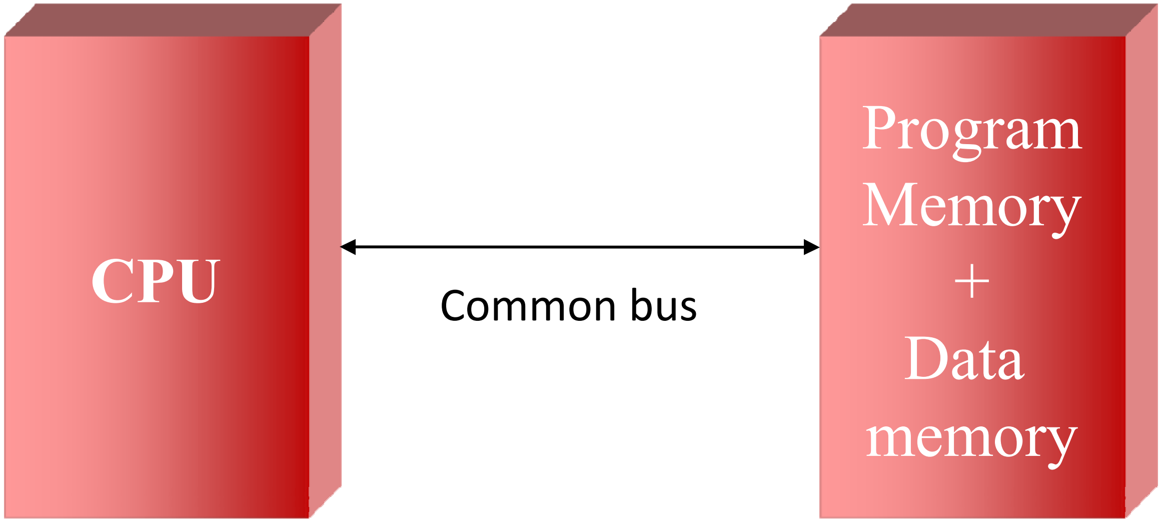
The Mid-Range PIC offers several features such as

- ✓ On-chip Power-on Reset (POR)
- ✓ Brown-out Reset (BOR) logic
- ✓ Watchdog Timer
- ✓ Low power mode (Sleep)

# CPU Architectures

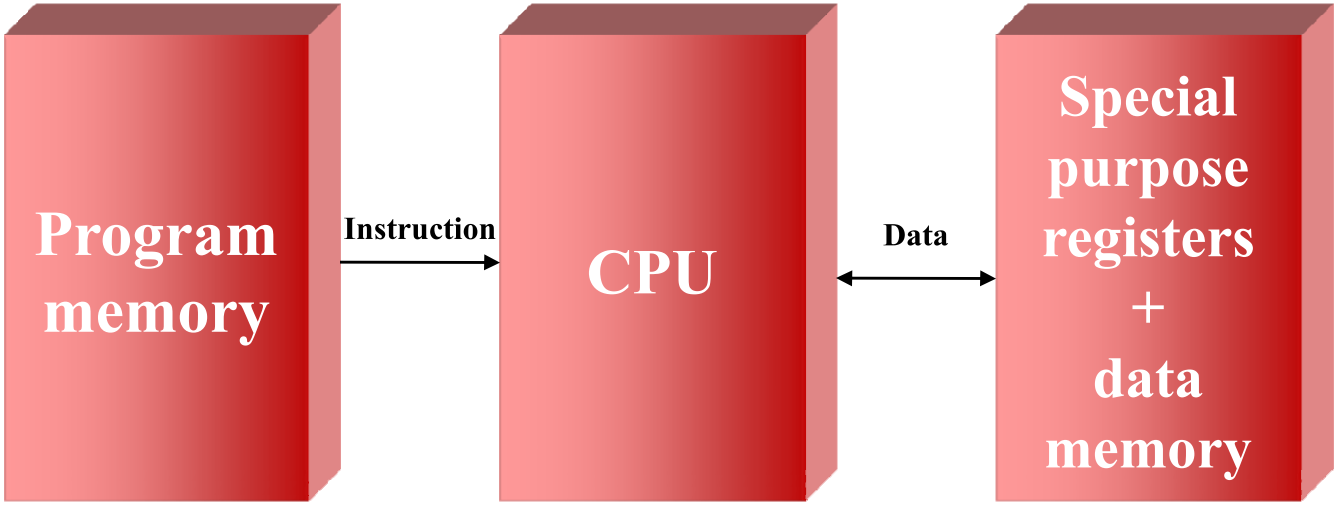
- Von-Neumann’s architecture
- Harvard architecture

# Von-Neumann's architecture



- Common data and address bus.
- More overhead.

# Harvard architecture



- Data bus and address bus are separate.
- So greater speed of work.

# Architectural Features

The high performance of the PIC micro™ devices can be attributed to a number of architectural features .These include:

- Harvard architecture
- Long Word Instructions
- Single Word Instructions
- Single Cycle Instructions
- Instruction Pipelining
- Reduced Instruction Set
- Register File Architecture
- Orthogonal (Symmetric) Instructions

# Memory Organization

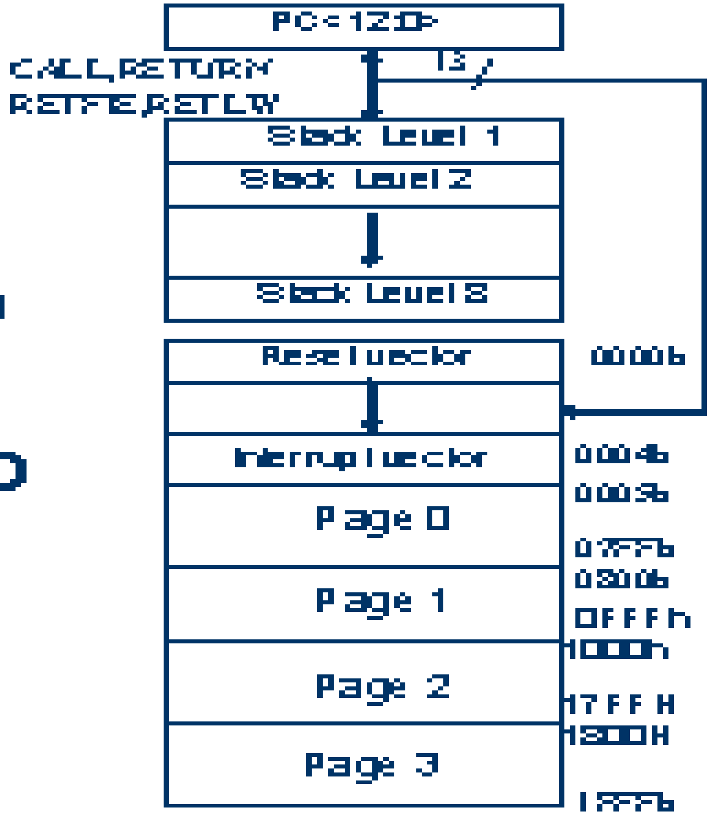
Three memory blocks

- ❑ Program memory – 14 bit
- ❑ Data memory – 8 bit
- ❑ Internal EEPROM data memory -8 bit

# Program Memory Organization

## Program memory map and stack

0000h AND 0004h ARE RESERVED



On chip program memory

PIC

# Data Memory Organization

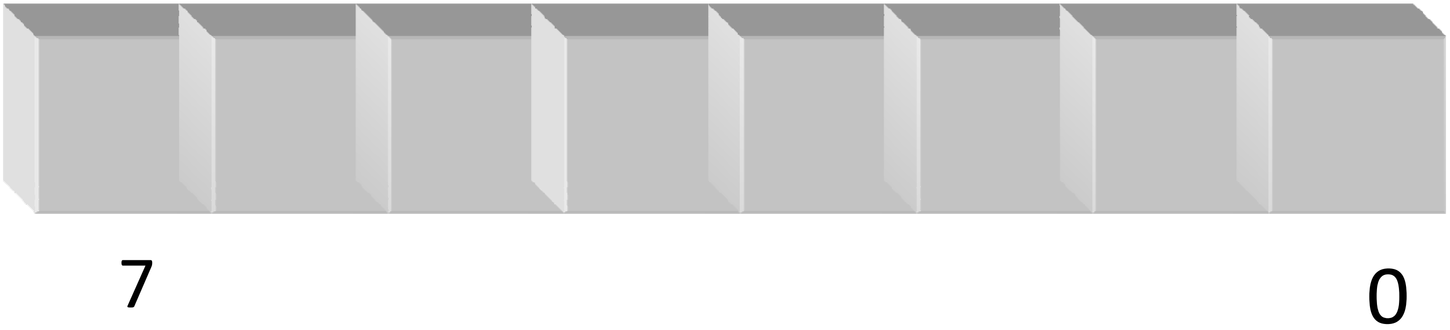
- Data memory is partitioned into multiple banks which contain General purpose registers and Special function registers.
- Each bank extends up to 7Fh (128 bytes).
- The lower locations of each bank are reserved for Special Function Registers.
- Above the SFR s are General Purpose Registers.
- Some ‘high use’ SFR s are mirrored in another bank for code reduction and quicker access.
- Bits RP1 and RP0 of the status register are the bank select bits.

# Register File Map

| | | | |
|---|---|---|---|
| 00h | 80h | 100h | 180h |
| RP1 RP0<br>0 0 | RP1 RP0<br>0 1 | RP1 RP0<br>1 0 | RP1 RP0<br>1 1 |
| 7Fh | FFh | 17Fh | 1FFh |
| BANK 0 | BANK1 | BANK2 | BANK3 |

# CPU Registers

✓Working register



Used as source of operand.
Destination for the results.

# CPU Registers

- ✓ Status register

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C/$\overline{B}$ |
| 7 | | | | | | | 0 |

# Bank Switching….

| RP1 | RP0 | |
|---|---|---|
| 0 | 0 | BANK0 |
| 0 | 1 | BANK1 |
| 1 | 0 | BANK2 |
| 1 | 1 | BANK3 |

MICROCHIP

# PIC16F73

# Naming of PIC

Lets take **PIC16F73**

- **PIC** – Peripheral Interface Controller
- **16** – Midrange series
- **F** – Flash memory
- **73** – 28 pin,8bit adc, with out internal EEPROM

# Naming of PIC

- **12** – Base line
- **16** – Midrange
- **17/18** – High end

- **C** – EPROM
- **CR** – ROM
- **F** – Flash

- **7X** – 28 pin,8bit adc, without internal EEPROM
- **87XX** - 28 pin,10bit adc, with internal EEPROM

# Core features

- 4K -Program memory
- 192 bytes - Data memory
- 8-bit RISC ALU
- Harvard architecture
- 28 pins with 22 I/O pins

# Question for you?

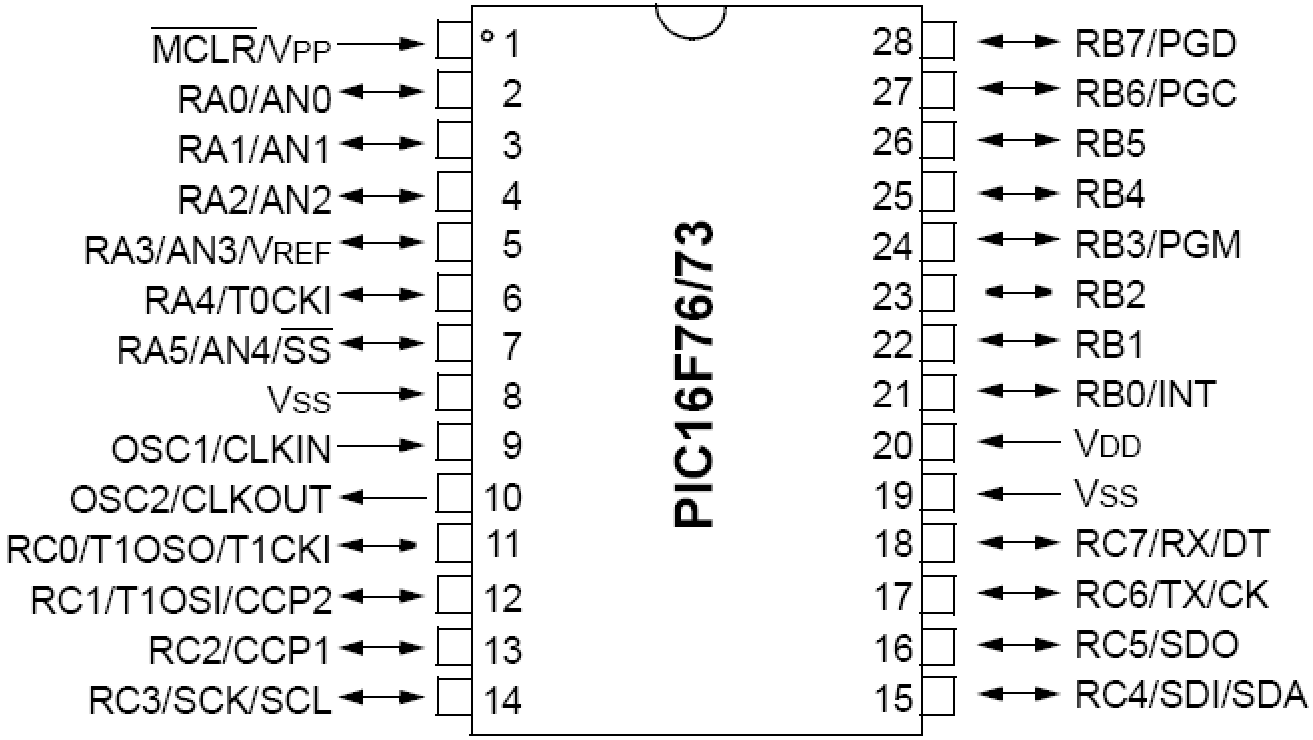
- PIC is an 8-bit microcontroller? Why?

The CPU can work on only 8-bits of data at a time.

# Peripheral features

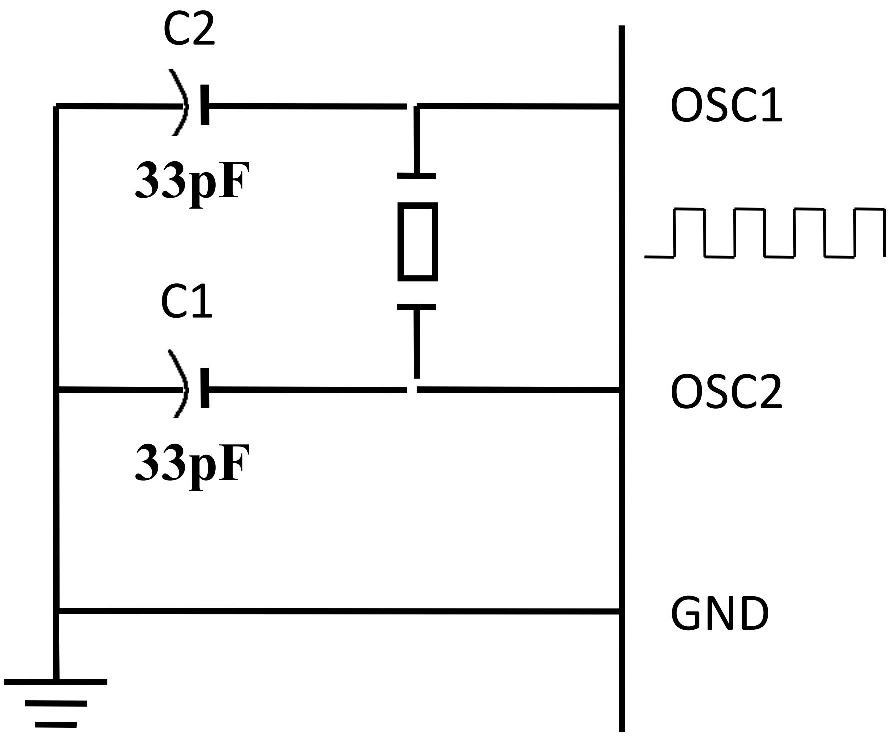
- 3 ports - PORTA - 6 bit
- PORTB – 8 bit
- PORTC - 8 bit
- 3 Programmable timers.
- 5 channel A/D converter.
- 11 interrupt sources.
- One USART module.
- Two Capture,Compare,PWM modules.

# Pin configuration



# XTAL Connection

- 4Mhz quartz crystal oscillator



PIC

# INSTRUCTION SET

# INSTRUCTION SET CLASSIFICATION

- BIT ORIENTED INSTRUCTIONS
- BYTE ORIENTED INSTRUCTIONS
- LITERAL AND CONTROL INSTRUCTIONS
- CALL AND GOTO INSTRUCTIONS

# COMMON SYMBOLS

- f-REGSITER(SFR OR GPR)
- b-BITPOSITION(0-7)
- d-DESINATION (0-working reg. & 1-reg.)

# Single bit manipulations

✓BCF   f,b

Clear bit b of register f, where b = 0 to 7

✓BSF   f,b

Set bit b of register f, where b = 0 to 7

# BCF f, b - Bit Clear Function

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

BCF R0, 4

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

# BSF f, b - Bit Set Function

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

BSF R0, 4

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

# Move

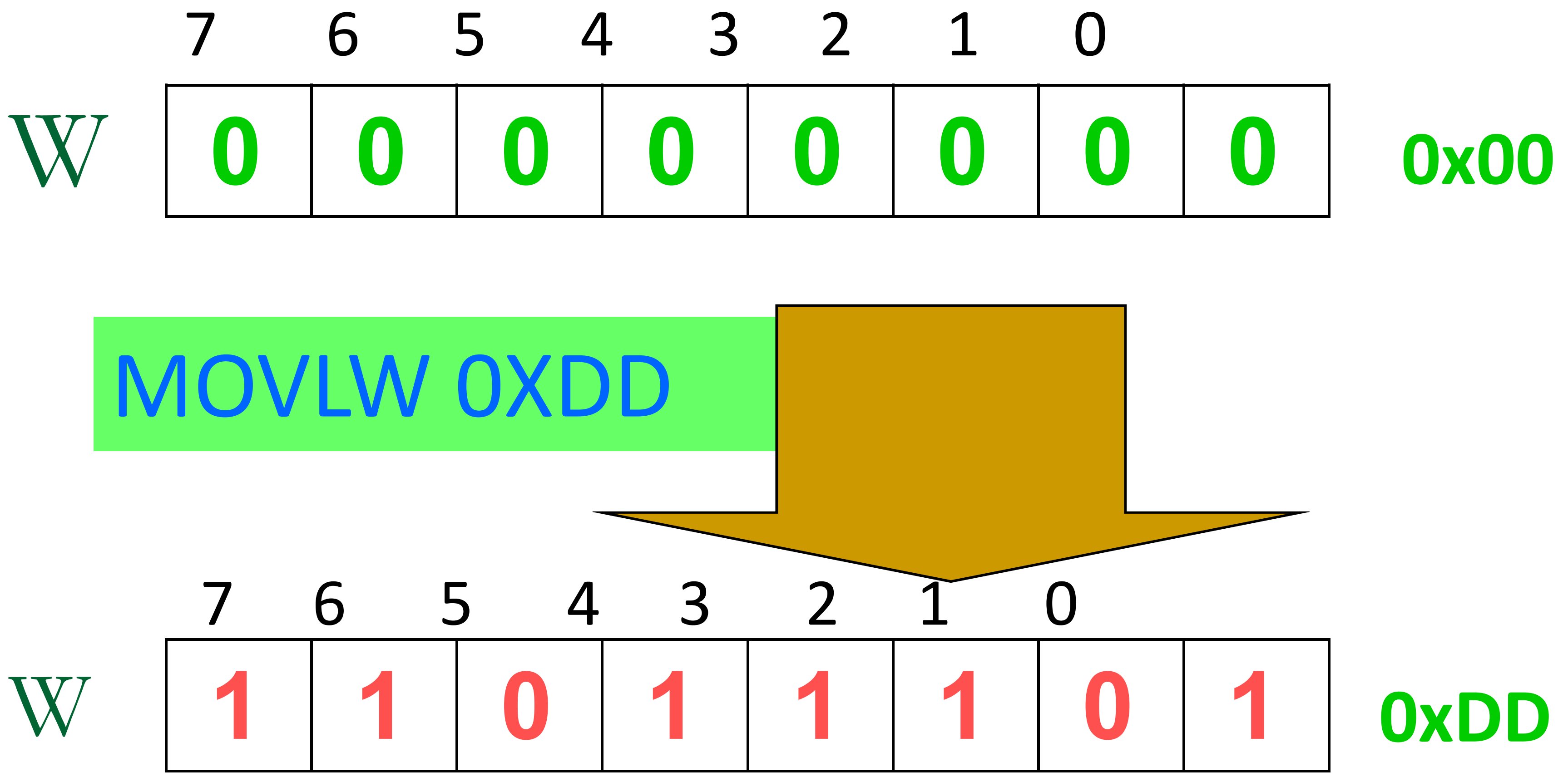
- ✓MOVLW k — w = k (literal value)

  where k=0x00,D’0’,B’0000’

- ✓MOVWF f — f = w (working reg. to reg.)

- ✓MOVF f ,w — w = f (reg. to working reg.)

# MOVLW k - Move Literal to w

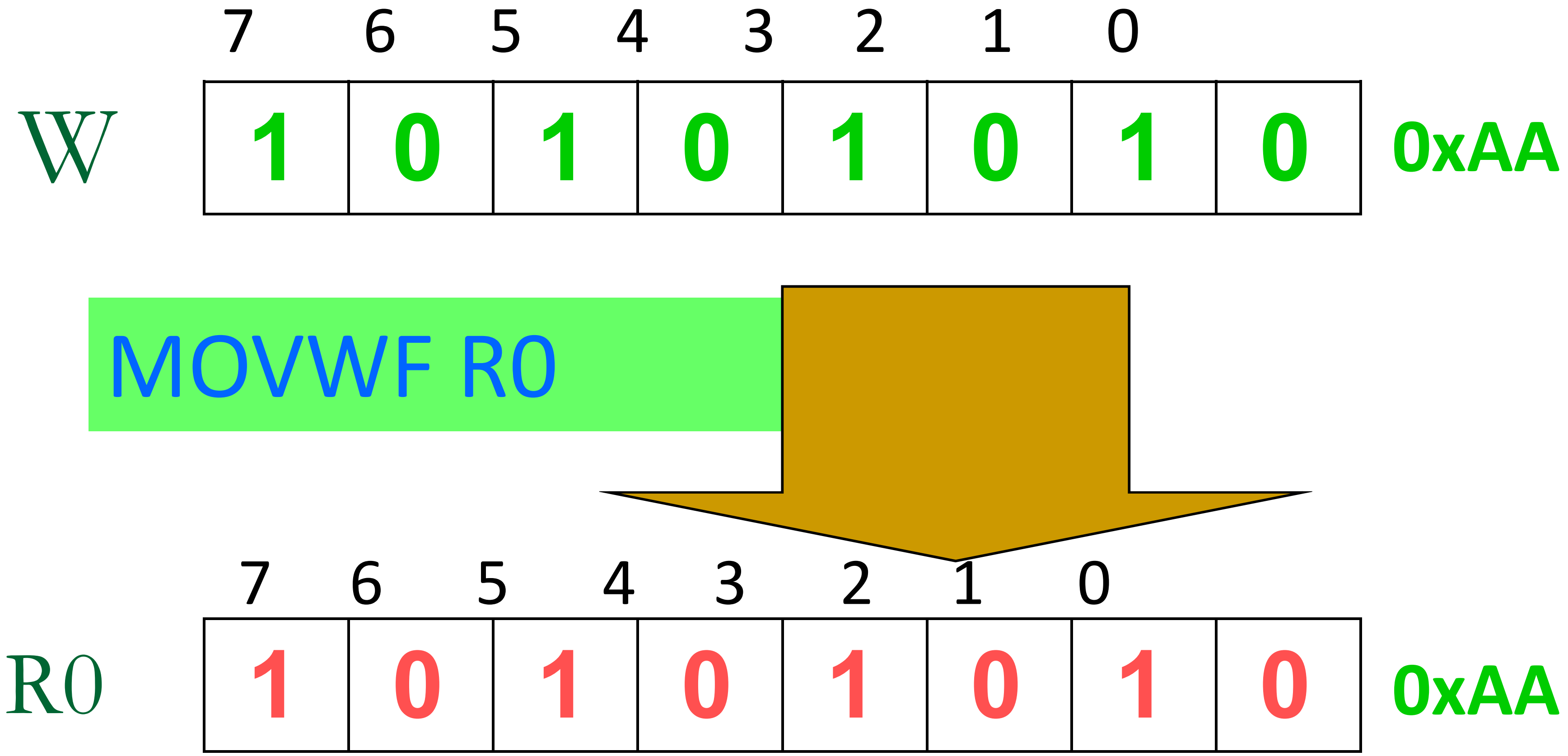


|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| W | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x00 |

MOVLW 0XDD

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0xDD |

# MOVWF f - Move w to f

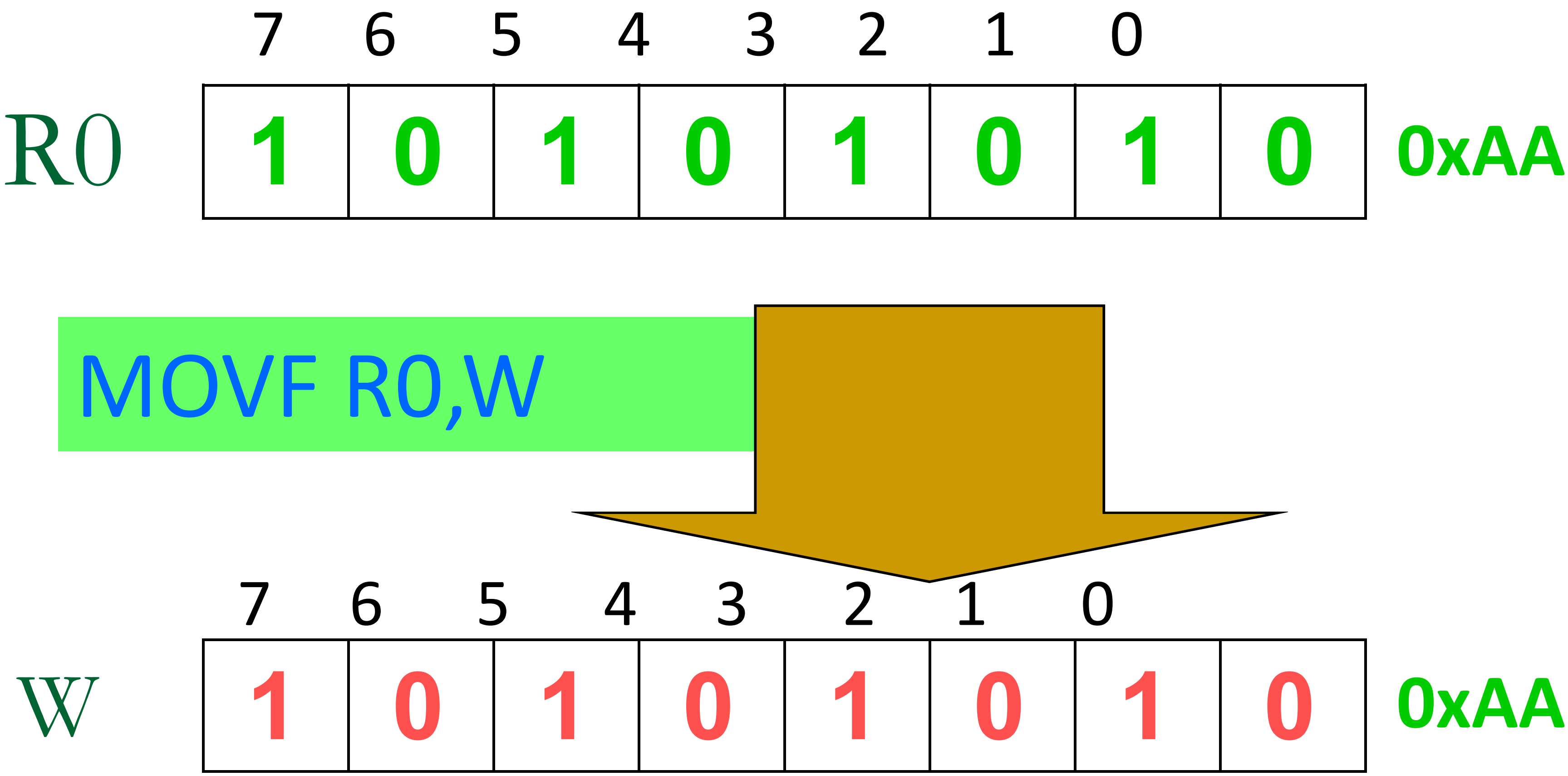


|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0xAA |

MOVWF R0

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0xAA |

# MOVF f, w - Move f to w



|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0xAA |

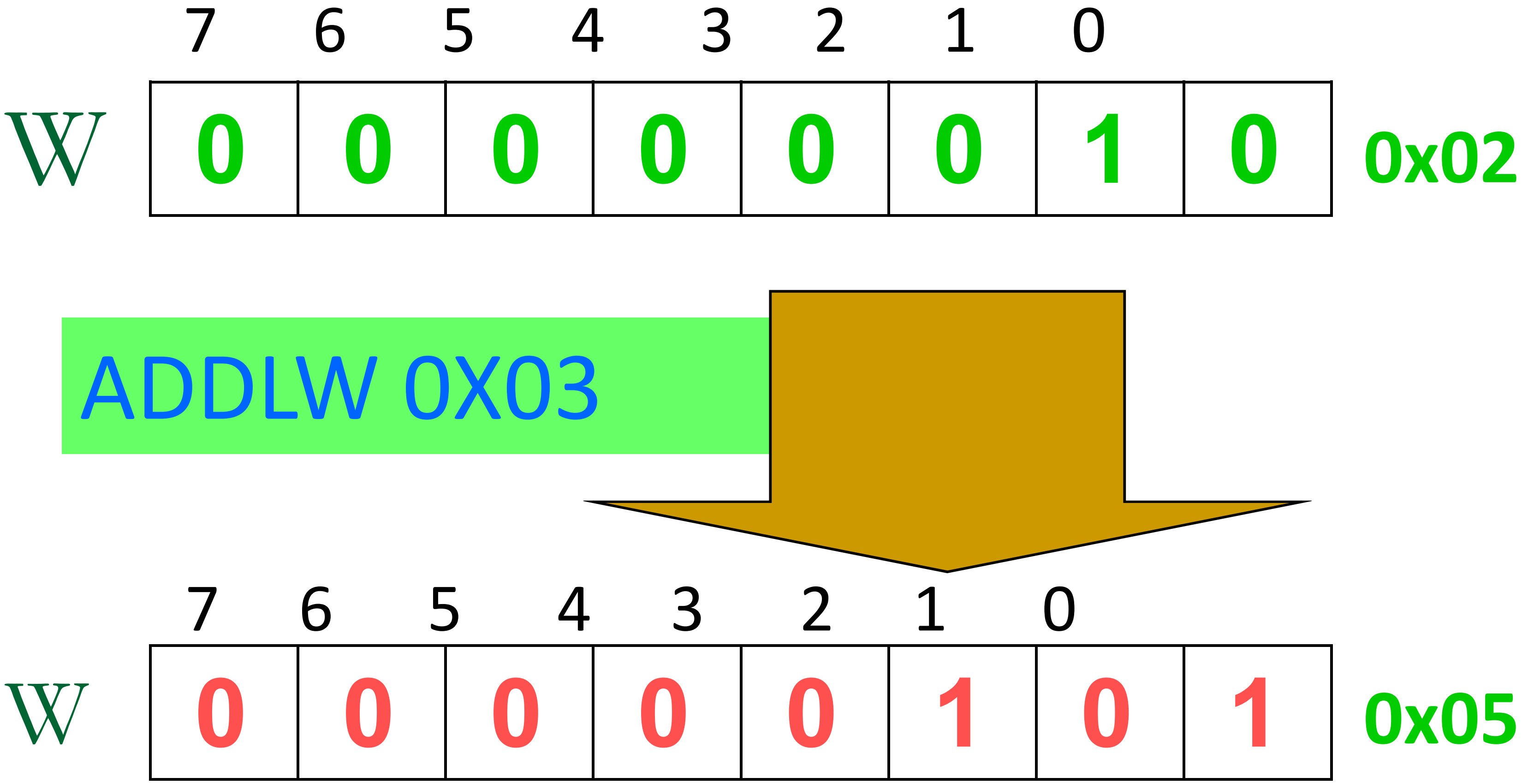
MOVF R0,W

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0xAA |

# Arithmetic Instructions

| | |
|---|---|
| ✓ADDLW k | w = w + k |
| ✓ADDWF f ,d | d = f + w |
| ✓SUBLW k | w = k -w |
| ✓SUBWF f ,d | d = f – w |
| ✓INCF f, d | d = f + 1 |
| ✓DECF f, d | d = f – 1 |

# ADDLW k – Add Literal and w



|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| W | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0x02 |

ADDLW 0X03

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| W | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0x05 |

# ADDWF f, d – Add W and f

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| W | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0X02 |

+

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| R0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0X03 |

ADDWF R0,0

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| W | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0X05 |

# Logical Instructions

| | |
|---|---|
| ✓ANDLW k | w = w AND k |
| ✓ANDWF f ,d | d  = w AND f |
| ✓IORLW k | w = w IOR k |
| ✓IORWF f ,d | d  = w IOR f |
| ✓XORLW k | w = w XOR k |
| ✓XORWF f ,d | d = w XOR f |
| ✓COMF f ,d | d = NOT f |

# XORLW k – XOR Literal with w

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0XAA |

XOR

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0XDD |

XORLW 0XDD

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| W | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0X77 |

# XORWF f ,d – XOR w with f

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0XA1 |

XOR

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0XF2 |

XORWF R0,1

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|---|---|---|---|---|---|---|---|---|---|
| R0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0X52 |

# Rotate Instructions

- ✓RLF f, d rotate F left through the carry bit
- ✓RRF f ,d rotate F right through the carry bit

# RLF f, d - Rotate left through Carry

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

RLF R0, 1

|  | C | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

# RRF f, d - Rotate right through Carry

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

RRF R0, 1

|  | C | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

# Clear ,Swap

| | |
|---|---|
| ✓CLRW | W = 0(Clear working reg.) |
| ✓CLRF f | f = 0(Clear any reg.) |
| ✓SWAPF f ,d | swap nibbles of f |

# SWAPF f, d – Swap the nibbles

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 |  | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|  | 1 |  |  |  |  |  |  |  |

```
SWAPF R0, 1
```

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 |  |  |  |  |  |  |  |  |

# Conditional branch Instructions

| | | |
|---|---|---|
| ✓BTFSC | f,b | Test bit b of register f ,skip if clear |
| ✓BTFSS | f,b | Test bit b of register f ,skip if set |
| ✓DECFSZ | f ,d | Decrement f ,skip if zero. |
| ✓INCFSZ | f ,d | Increment f ,skip if zero. |

# BTFSC f, b test bit b of register f, skip if clear

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

BTFSC R0, 4

The bit is Set

The bit is Clear

SKIPPING

# BTFSC f, b   test bit b of register f ,skip if clear

|    | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|---|---|---|---|---|---|---|---|
| R0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

BTFSC R0, 3

The bit is Set

The bit is Clear

NO SKIPPING

# Branch Instructions

| | |
|---|---|
| ✓ GOTO label | Go to labeled instruction |
| ✓ CALL label | Call labeled subroutine |
| ✓ RETURN | Return from subroutine |
| ✓ RETLW k | Return from subroutine ,putting literal value in W. |
| ✓ RETFIE | Return from interrupt service routine. |

# Other Instructions

| | |
|---|---|
| ✓CLRWDT | clear watchdog timer |
| ✓SLEEP | go into standby mode |
| ✓NOP | no operation |

MICROCHIP
WebSeminars

# An Introduction to MPLAB®<br>Integrated Development Environment

PIC

QUEST
INNOVATIVE SOLUTIONS

## What Is MPLAB® IDE?

- MPLAB® IDE is a software program that runs on your PC to provide a development environment for your embedded system design.

MICROCHIP WebSeminars

# First Project

- Select Device
- Create Project
- Select Language Tools
- Put Files in Project
- Create Code
- Build Project
- Test Code with Simulator

# How To Start

```
LIST P = PIC16F73
#INCLUDE  “P16F73.INC”

CBLOCK  0X20
REGISTERS
ENDC

ORG00H                          ;(RESET VECTOR LOCATION)
PROGRAM
END
```

PIC

# PIC16F73

# I/O PORTS

# PORTS

A port is a group of pins on a microcontroller on which the desired combinations of zeros and ones can be set simultaneously or the present status can be read.

PIC16F73 has 3 ports

- PORTA - 6 bit
- PORTB - 8 bit
- PORTC - 8 bit

# TRIS REGISTERS

- TRIS registers are control registers for ports
- TRIS registers are in BANK1

- TRISA control register for PORTA
- TRISB control register for PORTB
- TRISC control register for PORTC

# Configuration…….

- If a bit of TRIS Register = 1,corresponding PORT pin will be configured as input
- If a bit of TRIS Register = 0,corresponding PORT pin will be configured as output

# TRIS REGISTER

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PORTB | | | | | I/P | | | |
| TRISB | | | | | 1 | | | |
| | 7 | | | | | | | 0 |

- **1** - INPUT
- **0** - OUTPUT

INPUT

# TRIS REGISTER

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PORTB | | | | | O/P | | | |
| TRISB | | | | | 0 | | | |
| | 7 | | | | | | | 0 |

- **1** - INPUT
- **0** - OUTPUT

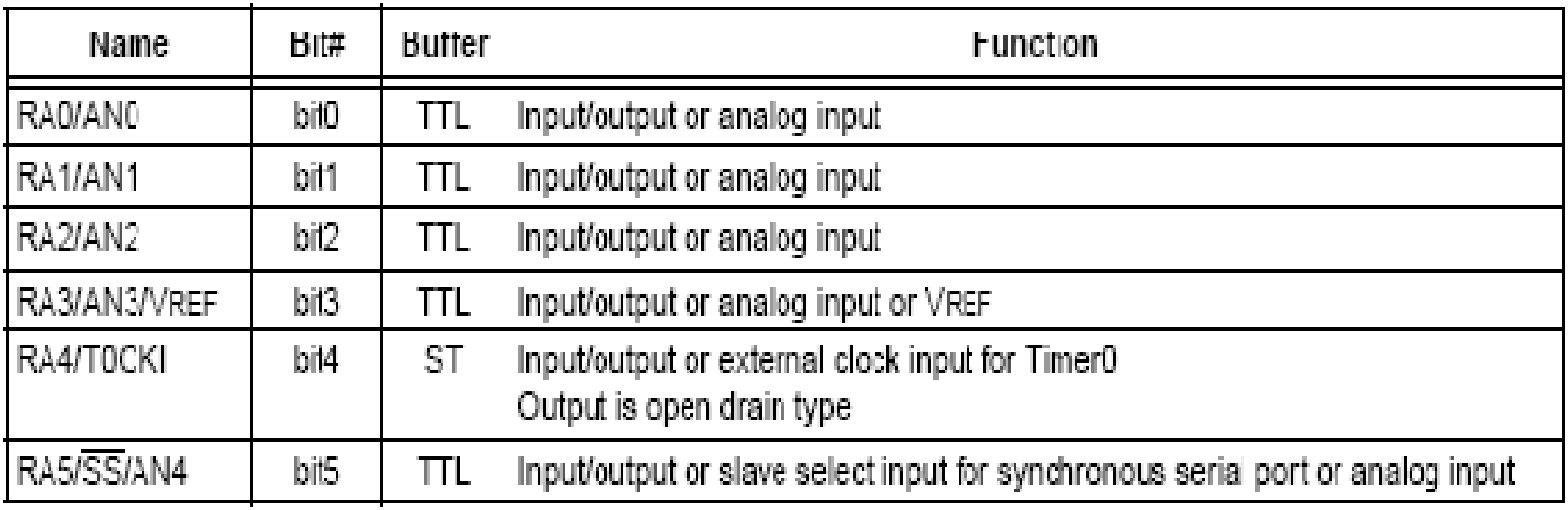
OUTPUT

# PORT A

- PORTA is a 6-bit wide bi-directional port.
- The corresponding data direction register is TRISA.
- Setting a TRISA bit (=1) will make the corresponding PORTA pin an input
- Clearing a TRISA bit (=0) will make the corresponding PORTA pin an output
- Multiplexed with 5-channel 8-bit ADC

  (PA0, PA1, PA2, PA3, PA5)

# PORTA FUNCTIONS



| Name | Bit# | Buffer | Function |
|---|---|---|---|
| RA0/AN0 | bit0 | TTL | Input/output or analog input |
| RA1/AN1 | bit1 | TTL | Input/output or analog input |
| RA2/AN2 | bit2 | TTL | Input/output or analog input |
| RA3/AN3/VREF | bit3 | TTL | Input/output or analog input or VREF |
| RA4/T0CKI | bit4 | ST | Input/output or external clock input for Timer0<br>Output is open drain type |
| RA5/$\overline{SS}$/AN4 | bit5 | TTL | Input/output or slave select input for synchronous serial port or analog input |

# PORT B

- PORTB is an 8-bit wide, bi-directional port.
- The corresponding data direction register is TRISB.
- Setting a TRISB bit (=1) will make the corresponding PORTB pin an input
- Clearing a TRISB bit (=0) will make the corresponding PORTB pin an output
- Three pins of PORTB are multiplexed with the Low Voltage Programming function; RB3/PGM, RB6/PGC and RB7/PGD.

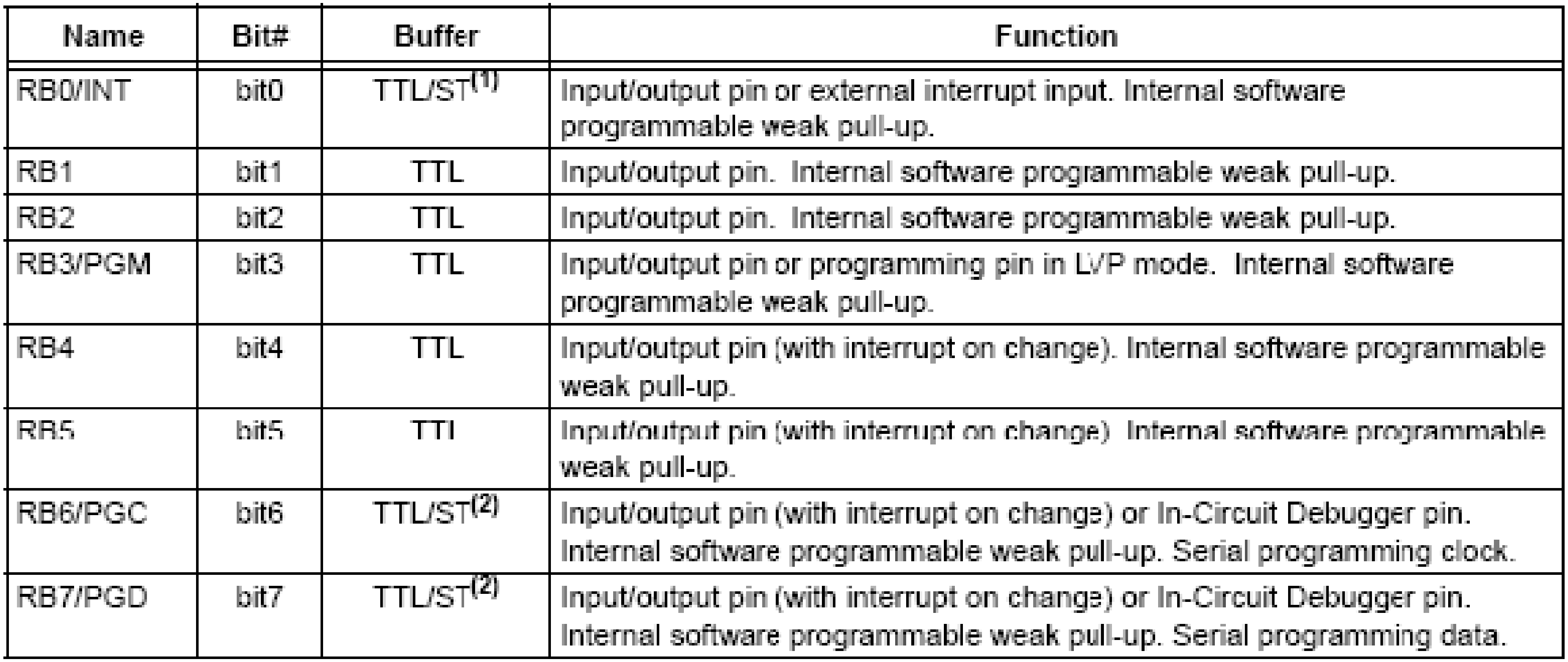
# PORT B

- Four of PORTB's pins, RB7:RB4, have an interrupt on change feature.

- RB0/INT is an external interrupt input pin

- Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups.

# PORTB FUNCTIONS

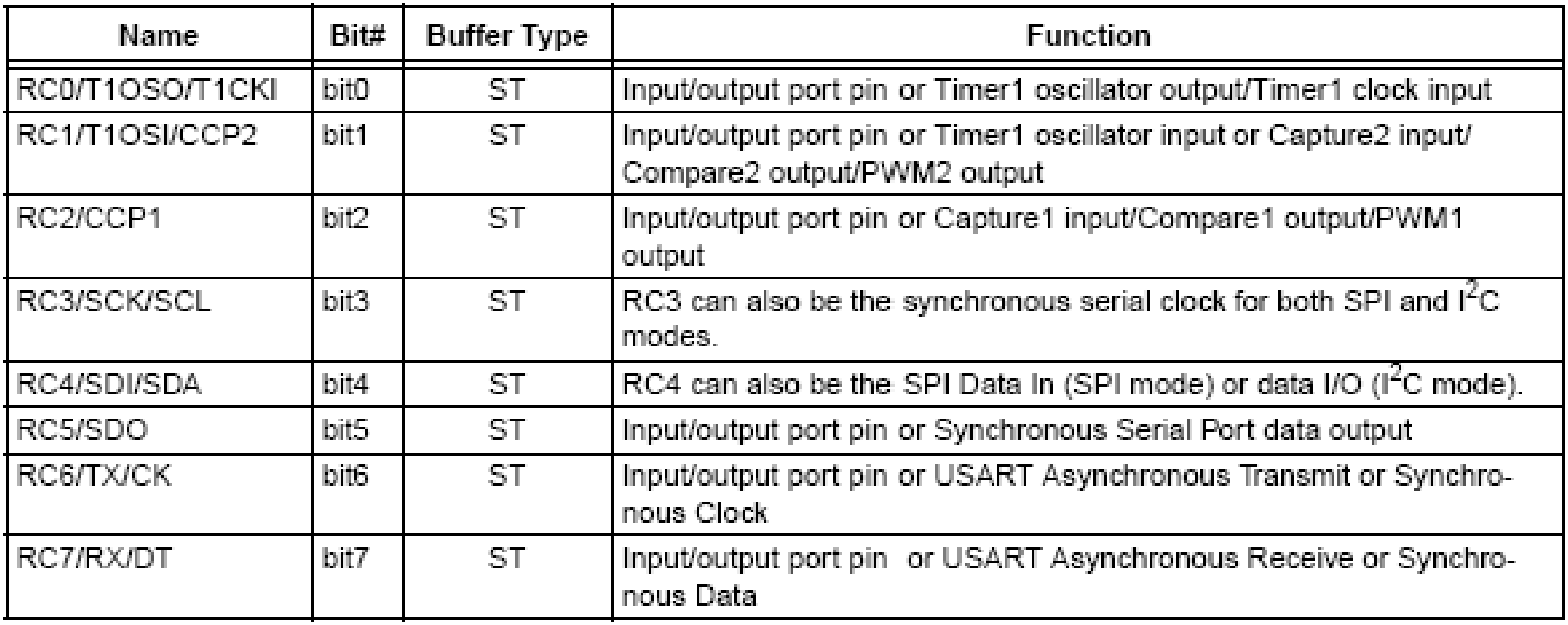


| Name | Bit# | Buffer | Function |
|---|---|---|---|
| RB0/INT | bit0 | TTL/ST(1) | Input/output pin or external interrupt input. Internal software programmable weak pull-up. |
| RB1 | bit1 | TTL | Input/output pin. Internal software programmable weak pull-up. |
| RB2 | bit2 | TTL | Input/output pin. Internal software programmable weak pull-up. |
| RB3/PGM | bit3 | TTL | Input/output pin or programming pin in LVP mode. Internal software programmable weak pull-up. |
| RB4 | bit4 | TTL | Input/output pin (with interrupt on change). Internal software programmable weak pull-up. |
| RB5 | bit5 | TTL | Input/output pin (with interrupt on change). Internal software programmable weak pull-up. |
| RB6/PGC | bit6 | TTL/ST(2) | Input/output pin (with interrupt on change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming clock. |
| RB7/PGD | bit7 | TTL/ST(2) | Input/output pin (with interrupt on change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming data. |

# PORT C

- PORTC is an 8-bit wide, bi-directional port.
- The corresponding data direction register is TRISC.
- Setting a TRISC bit (=1) will make the corresponding PORTC pin an input
- Clearing a TRISC bit (=0) will make the corresponding PORTC pin an output

# PORTC FUNCTIONS



| Name | Bit# | Buffer Type | Function |
|---|---|---|---|
| RC0/T1OSO/T1CKI | bit0 | ST | Input/output port pin or Timer1 oscillator output/Timer1 clock input |
| RC1/T1OSI/CCP2 | bit1 | ST | Input/output port pin or Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output |
| RC2/CCP1 | bit2 | ST | Input/output port pin or Capture1 input/Compare1 output/PWM1 output |
| RC3/SCK/SCL | bit3 | ST | RC3 can also be the synchronous serial clock for both SPI and $I^2C$ modes. |
| RC4/SDI/SDA | bit4 | ST | RC4 can also be the SPI Data In (SPI mode) or data I/O ($I^2C$ mode). |
| RC5/SDO | bit5 | ST | Input/output port pin or Synchronous Serial Port data output |
| RC6/TX/CK | bit6 | ST | Input/output port pin or USART Asynchronous Transmit or Synchronous Clock |
| RC7/RX/DT | bit7 | ST | Input/output port pin  or USART Asynchronous Receive or Synchronous Data |

# PIC16F73

## INTERRUPTS

# INTERRUPTS

- Interrupts are mechanisms of a microcontroller which makes it possible to respond to some events at the moment when they occur, regardless of what microcontroller is doing at the time.

Each interrupt changes the flow of program execution, after executing an interrupt subprogram (interrupt service routine) it continues from the same point .

# INTCON

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF |

# INTCON

**RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7: RB4 pins changed state
0 = None of the RB7: RB4 pins have changed state

**INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred
0 = The RB0/INT external interrupt did not occur

**T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 has overflowed
0 = TMR0 did not overflow

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

# INTCON

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

**T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

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

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

# PIC16F73

## TIMERS

# TIMERS

PIC16F73 has 3 Timers

- TIMER0 - 8 bit
- TIMER1 - 16 bit
- TIMER2 - 8 bit

# TIMER 0

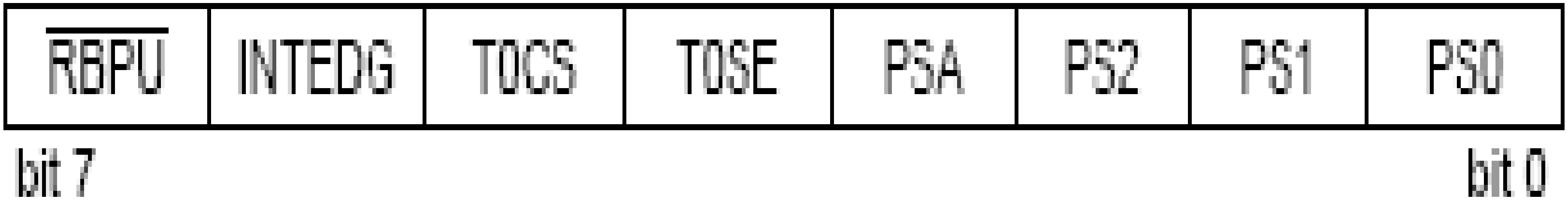
Features of Timer0 module:

- 8-bit timer/counter
- Readable and writable
- 8-bit software programmable prescaler
- Interrupt on overflow from FFh to 00h

# TIMER 0 CONFIGURATION

- Configure the OPTION REGISTER
- Load the count value
- Wait for overflow flag to set
- Clear the overflow flag

# OPTION REGISTER

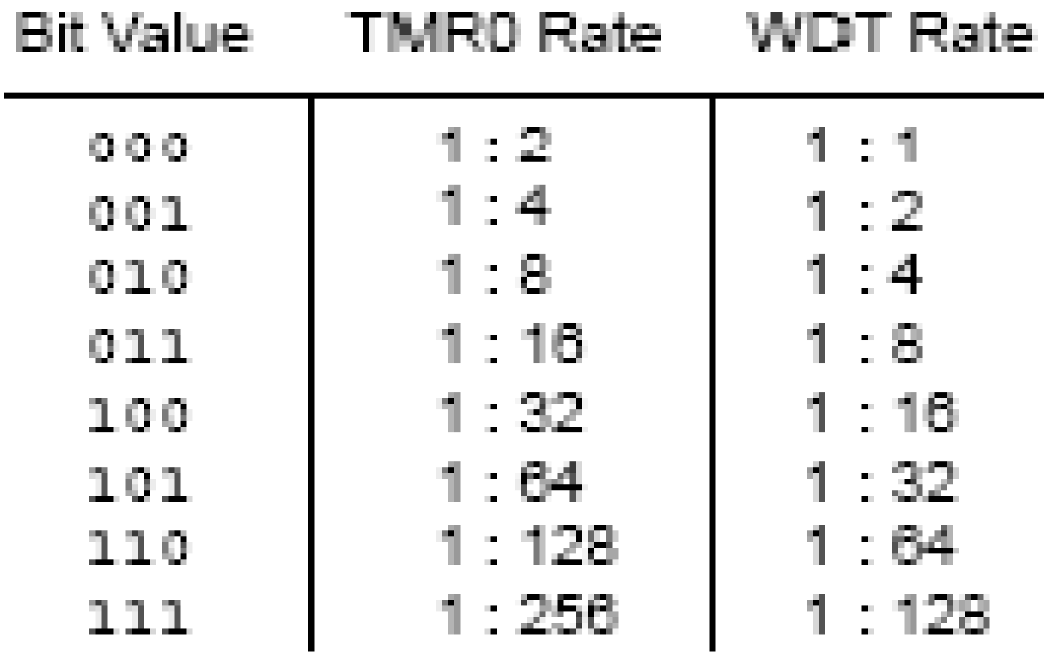


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\overline{\text{RBPU}}$ | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 |

bit 7 bit 0

# OPTION REGISTER

**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 |

# OPTION REGISTER

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

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

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

**INTEDG**

**RBPU**

# Count Calculation

$$\text{Delay} = \frac{4}{\text{F Oscillator}} * \text{Prescalar}\ (\ 255 - \text{COUNT}\ )$$

# Registers

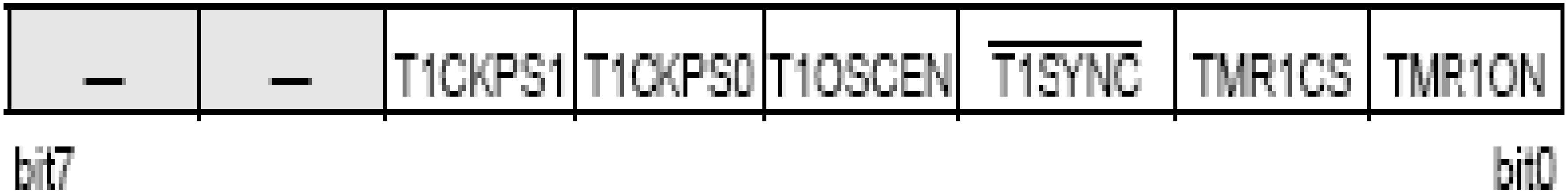
| | | |
|---|---|---|
| **OPTION_REG** | Configuration Register | BANK1 |
| **TMR0** | Count Register | BANK0 |
| **INTCON*(2nd bit)*** | Overflow Flag | BANK0 |
| **INTCON*(5th bit)*** | Interrupt Enable | BANK0 |

# TIMER 1

- The Timer1 module is a 16-bit timer consisting of two 8-bit registers (TMR1H and TMR1L)

- The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh

# T1CON REGISTER



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| — | — | T1CKPS1 | T1CKPS0 | T1OSCEN | $\overline{\text{T1SYNC}}$ | TMR1CS | TMR1ON |
| bit7 | | | | | | | bit0 |

# T1CON REGISTER

**TMR1ON**: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

**TMR1CS**: Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (FOSC/4)

**T1SYNC**: Timer1 External Clock Input Synchronization Control bit
TMR1CS = 1
1 = Do not synchronize external clock input
0 = Synchronize external clock input
TMR1CS = 0
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

# T1CON REGISTER

**T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut off

**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

**Unimplemented:** Read as ’0’

# Registers

| | | |
|---|---|---|
| **T1CON** | Configuration Register | BANK0 |
| **TMR1L & TMR1H** | Count Register | BANK0 |
| **PIR1 *(0th BIT)*** | Overflow Flag | BANK0 |
| **PIE1 *(0th BIT)*** | Interrupt Enable | BANK1 |
| **INTCON** | Interrupt Control | BANK0 |

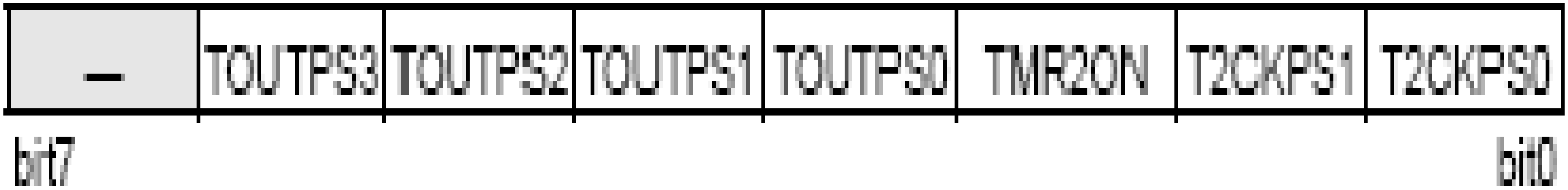
# TIMER 2

- Timer2 is an 8-bit timer with a prescaler and a postscaler.

- The Timer2 module has an 8-bit period register PR2.

Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle.

# T2CON REGISTER



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| – | TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | TMR2ON | T2CKPS1 | T2CKPS0 |
| bit7 | | | | | | | bit0 |

# T2CON REGISTER

**T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

**TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

**TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
0010 = 1:3 Postscale
•
1111 = 1:16 Postscale

**Unimplemented:** Read as '0'

# Registers

| | | |
|---|---|---|
| **T2CON** | Configuration Register | BANK0 |
| **TMR2** | Count Register | BANK0 |
| **PR2** | Period Register | BANK1 |
| **PIR1** ***(1st BIT)*** | Overflow Flag | BANK0 |
| **PIE1** ***(1st BIT)*** | Interrupt Enable | BANK1 |
| **INTCON** | Interrupt Control | BANK0 |

PIC

# PIC16F73

# USART

QUEST
INNOVATIVE SOLUTIONS

# USART

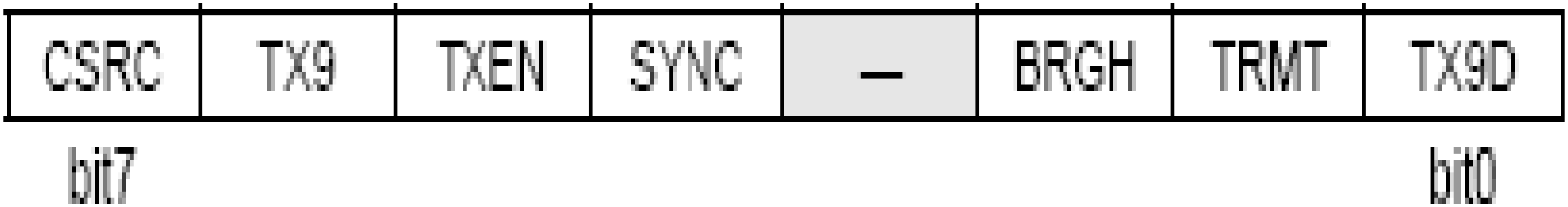
- The Universal Synchronous Asynchronous Receiver

Transmitter (USART) module is one of the two serial I/O modules. (USART is also known as a Serial Communications Interface or SCI).

- The USART can be configured as a full duplex asynchronous system that can communicate with peripheral devices such as CRT terminals and personal computers.

# TXSTA

## TRANSMIT STATUS AND CONTROL REGISTER



| CSRC | TX9 | TXEN | SYNC | — | BRGH | TRMT | TX9D |
|---|---|---|---|---|---|---|---|
| bit7 | | | | | | | bit0 |

# TXSTA

**TX9D:** 9th bit of transmit data. Can be parity bit.

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

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

**Unimplemented:** Read as '0‘

# TXSTA

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

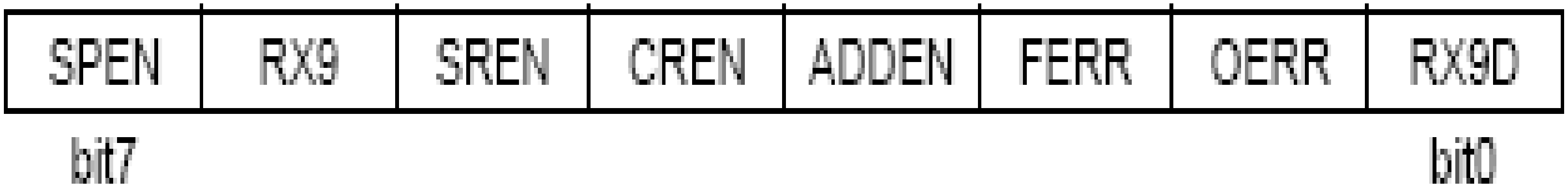
**TXEN**: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled

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

**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)

# RCSTA

## RECEIVE STATUS AND CONTROL REGISTER



|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| SPEN | RX9 | SREN | CREN | ADDEN | FERR | OERR | RX9D |
| bit7 |  |  |  |  |  |  | bit0 |

# RCSTA

**RX9D:** 9th bit of received data (Can be parity bit)

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

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

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

**SPEN:** Serial Port Enable bit
1 = Serial port enabled
0 = Serial port disabled

# RCSTA

| | |
|---|---|
| SPBRG: | Baud Rate Generator Register |
| TXREG: | Transmit Buffer |
| RCREG: | Receive Buffer |

# Registers

| | | |
|---|---|---|
| **TXSTA** | Configuration Register | BANK1 |
| **RCSTA** | Configuration Register | BANK0 |
| **TXREG** | Transmit Buffer | BANK0 |
| **RCREG** | Receive Buffer | BANK0 |
| **SPBRG** | Baud Rate | BANK1 |
| **TXSTA *(1st BIT)*** | **Transmit Flag** | BANK1 |
| **PIR1 *(5th BIT)*** | Receive Flag | BANK0 |
| **PIE1 *(4th 5th BIT)*** | Interrupt Enable | BANK1 |
| **INTCON** | Interrupt Control | BANK0 |

# Steps for Transmission

STEP1: Configure TXSTA and RCSTA

STEP2: Set Baud rate

STEP3: Load TXREG with the data to be transmitted

STEP4: Wait for TRMT

# Steps for Reception

STEP1: Configure TXSTA and RCSTA

STEP2: Set Baud rate

STEP3: Wait for receive flag RCIF in PIR1

STEP4: Take the value from receive buffer RCREG

# PIC16F73

## ADC

# ADC

- The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the other devices.

- The analog input charges a sample and hold capacitor. The output of the sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. The A/D conversion of the analog input signal results in a corresponding 10-bit digital number.

# REGISTERS

The A/D module has four registers.

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

# ADCON0 REGISTER

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ADCS1 | ADCS0 | CHS2 | CHS1 | CHS0 | GO/$\overline{\text{DONE}}$ | — | ADON |
| bit7 | | | | | | | bit0 |

# ADCON0 REGISTER

**ADON**: A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shutoff and consumes no operating current

**Unimplemented**: Read as '0‘

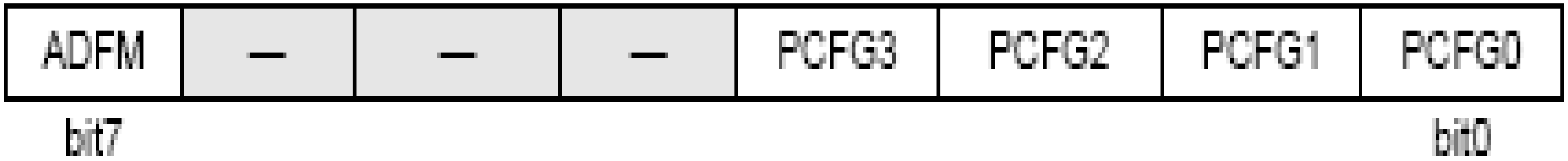
**GO/DONE:** A/D Conversion Status bit
1 = A/D conversion in progress
0 = A/D conversion completed

# ADCON0 REGISTER

**CHS2:CHS0**: Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)

**ADCS1:ADCS0: A/D Conversion Clock Select bits**
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (clock derived from an RC oscillation)

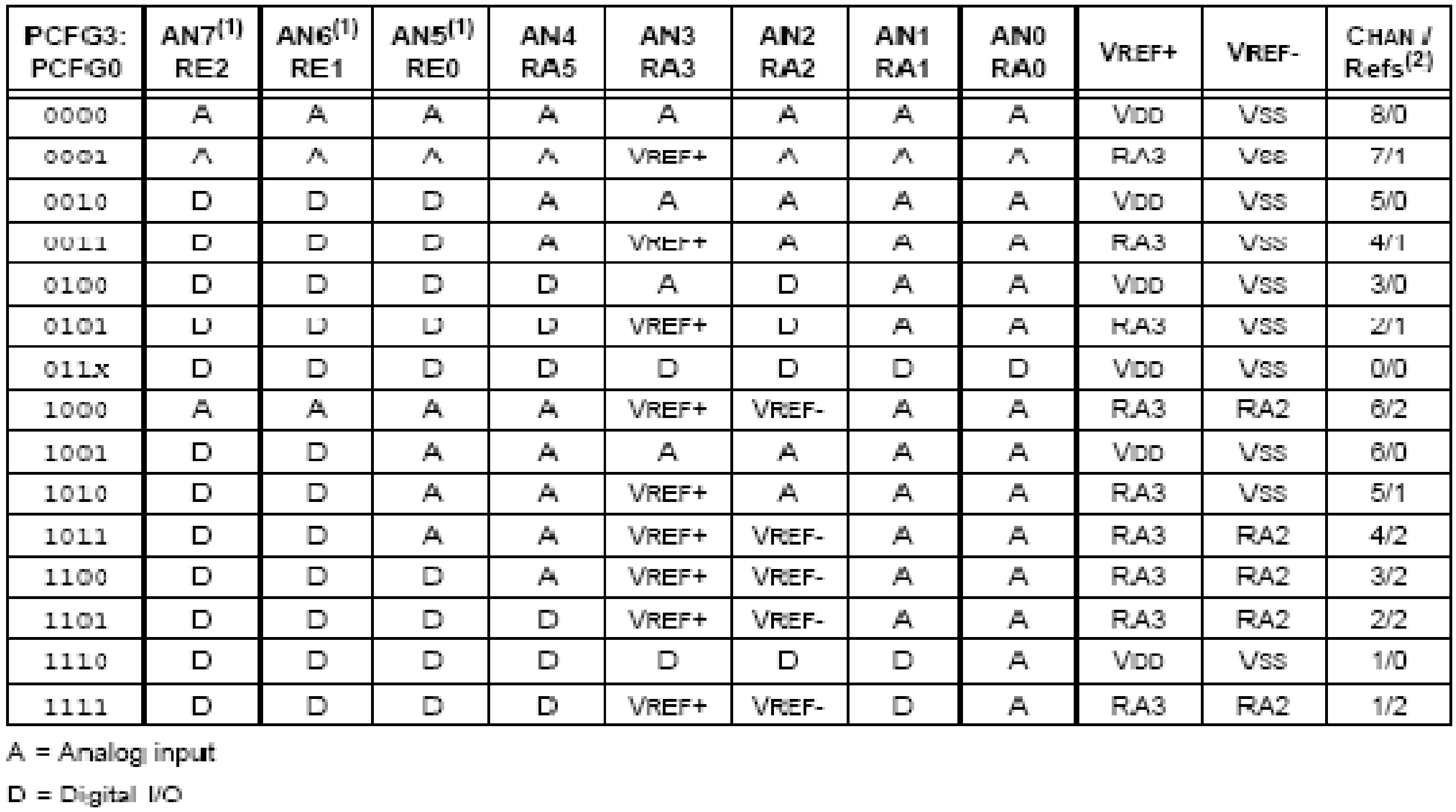
# ADCON1 REGISTER



| ADFM | — | — | — | PCFG3 | PCFG2 | PCFG1 | PCFG0 |
|---|---|---|---|---|---|---|---|
| bit7 |  |  |  |  |  |  | bit0 |

# ADCON1 REGISTER

**PCFG3:PCFG0**: A/D Port Configuration Control bits



| PCFG3:<br>PCFG0 | AN7[1]<br>RE2 | AN6[1]<br>RE1 | AN5[1]<br>RE0 | AN4<br>RA5 | AN3<br>RA3 | AN2<br>RA2 | AN1<br>RA1 | AN0<br>RA0 | VREF+ | VREF- | CHAN /<br>Refs[2] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | A | A | A | A | A | A | A | A | VDD | VSS | 8/0 |
| 0001 | A | A | A | A | VREF+ | A | A | A | RA3 | 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 | RA3 | 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 | RA3 | VSS | 2/1 |
| 011x | D | D | D | D | D | D | D | D | VDD | VSS | 0/0 |
| 1000 | A | A | A | A | VREF+ | VREF- | A | A | RA3 | RA2 | 6/2 |
| 1001 | D | D | A | A | A | A | A | A | VDD | VSS | 6/0 |
| 1010 | D | D | A | A | VREF+ | A | A | A | RA3 | VSS | 5/1 |
| 1011 | D | D | A | A | VREF+ | VREF- | A | A | RA3 | RA2 | 4/2 |
| 1100 | D | D | D | A | VREF+ | VREF- | A | A | RA3 | RA2 | 3/2 |
| 1101 | D | D | D | D | VREF+ | VREF- | A | A | RA3 | RA2 | 2/2 |
| 1110 | D | D | D | D | D | D | D | A | VDD | VSS | 1/0 |
| 1111 | D | D | D | D | VREF+ | VREF- | D | A | RA3 | RA2 | 1/2 |

A = Analog input

D = Digital I/O

# ADCON1 REGISTER

**Unimplemented: Read as ’0’**

**ADFM: A/D Result format select**
1 = Right Justified. 6 most significant bits of ADRESH are read as ‘0’.
0 = Left Justified. 6 least significant bits of ADRESL are read as ‘0’.

# Registers

| | | |
|---|---|---|
| **ADCON0** | Configuration Register | BANK0 |
| **ADCON1** | Configuration Register | BANK1 |
| **ADRESH** | Output Register | BANK0 |
| **ADRESL** | Output Register | BANK0 |

# Steps for A to D Conversion

STEP1: Configure Port A as input according to the number of channels required.

STEP2: Configure ADCON0 & ADCON1

STEP3: Transition delay of 20us

STEP4: Start conversion

STEP5:  Wait for DONE

STEP6:  Take the output from ADRESH

# PIC16F73

# SPI

# What is SPI ?

- SPI is frequently used when few I/O lines are available, but communication between two or more devices must be fast and easy to implement.

# SPI - Overview

- SPI stands for Serial Peripheral Interface
- Used for moving data simply and quickly from one device to another
- Serial Interface
- Synchronous
- Master-Slave
- Data Exchange

# SPI - Overview

- SPI is a Synchronous protocol
  - ❑ The data is clocked along with a clock signal
  - ❑ The clock signal controls when data is changed and when it should be read
  - ❑ Since SPI is synchronous, the clock rate can vary, unlike RS-232 style communications

# SPI - Overview

- SPI is a Master-Slave protocol
  - The Master device controls the clock (SCK)
  - No data is transferred unless a clock signal is present
  - All slaves are controlled by the master clock
  - The slave devices may not manipulate the clock

# SPI - Overview

- SPI is a Data Exchange protocol
  - As data is being clocked out, data is clocked in
  - Data is ***exchanged*** - *no device can just be a transmitter only or receiver only*
  - The master controls the exchange by manipulating the clock line (SCK)

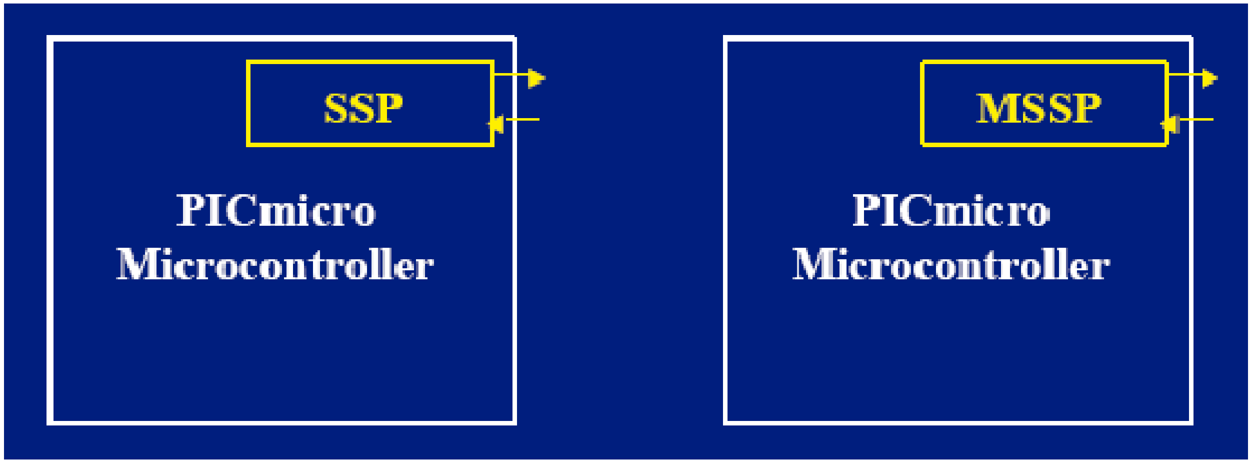
# SPI - Overview

- SPI is a Data Exchange protocol

  - ❑ Often a signal controls when a device is accessed - this is the CS or SS signal
  - ❑ CS or SS signal is known as “Chip Select” or “Slave Select” and is frequently an active-low signal.

# SPI - Overview

- The SSP (or MSSP) module in the PICmicro device allows SPI and other synchronous serial protocols



# SPI – Overview Cont..

- SPI is a Serial Interface using these signals:

- SS Chip Select or Slave Select

  When this signal goes low, the slave will listen for SPI clock and data signals

- SCK Serial ClocK

  This controls when data is sent and when it is read

# SPI – Overview Cont..

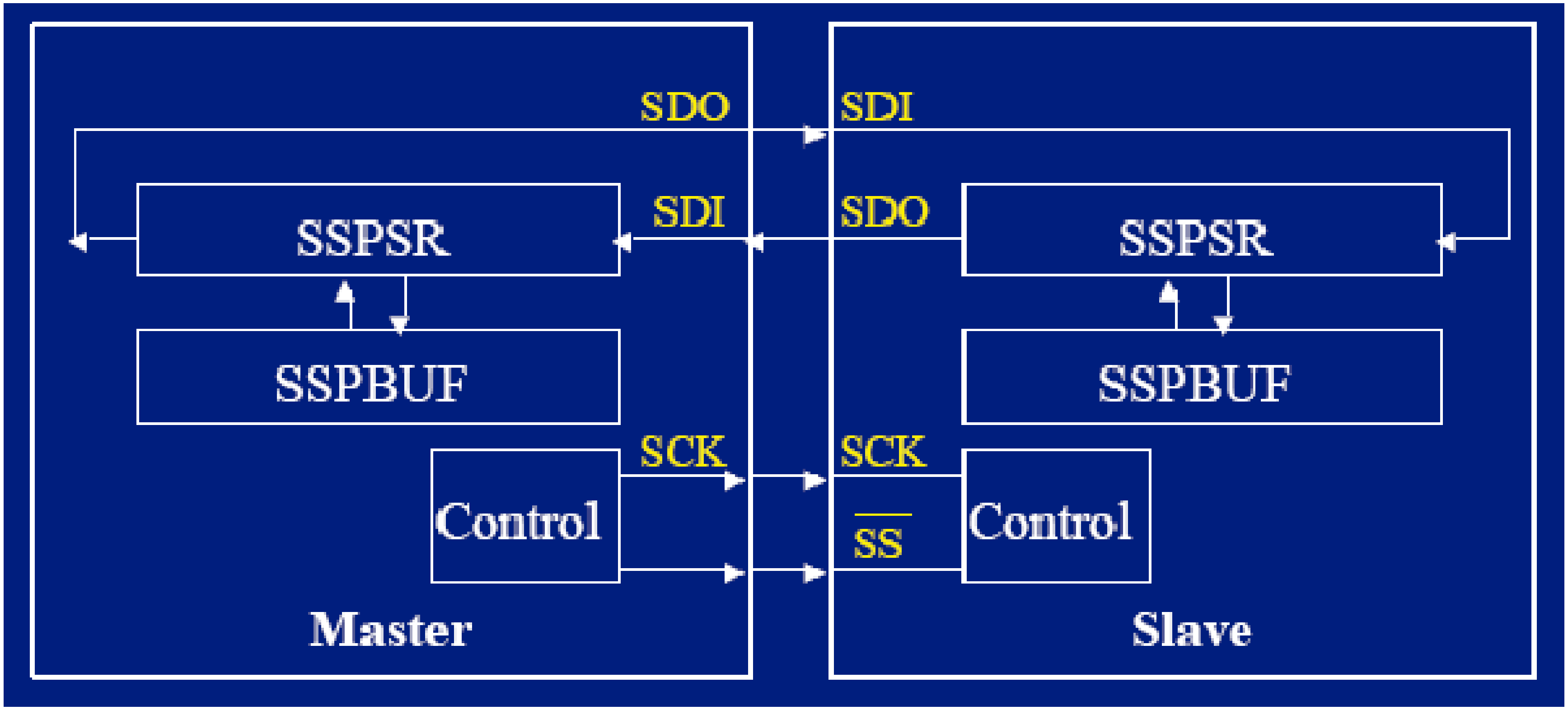
- SDO Serial Data Output

  This signal carries the data sent *out of* the device

- SDI Serial Data Input

  This signal carries the data sent *into* the device

# SPI Data Transfer



PIC

# PIC16F73

# I2C

# I2C - Overview

- Used for moving data simply and quickly from one device to another
- Serial Interface
- Synchronous
- Bidirectional

# I2C - Overview

- I2C is a Synchronous protocol
  - The data is clocked along with a clock signal
  - The clock signal controls when data is changed and when it should be read
  - Since I2C is synchronous, the clock rate can vary, unlike asynchronous (RS-232 style) communications

# I2C - Overview

- I2C is a Master-Slave protocol
  - ❑ The Master device controls the clock (SCL)
  - ❑ The slave devices may hold the clock low to prevent data transfer
  - ❑ No data is transferred unless a clock signal is present
  - ❑ All slaves are controlled by the master clock

# I2C - Overview

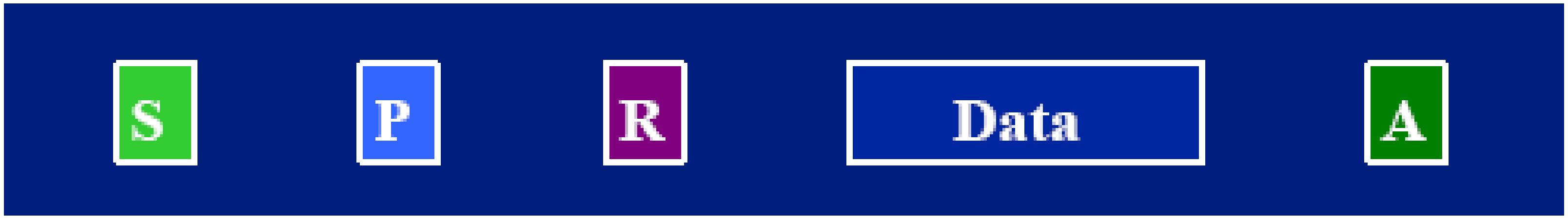
- I2C is a Bidirectional protocol
  - Data is sent in either direction on the serial data line (SDA) by the master or slave.

# I2C - Overview

- I2C is a Serial Interface of only two signals:
  - ❑ SDA Serial DAta
    - This line transfers data to or from the master.
  - ❑ SCL Serial CLock
    - This controls when data is sent and when it is read. The master controls SCK.

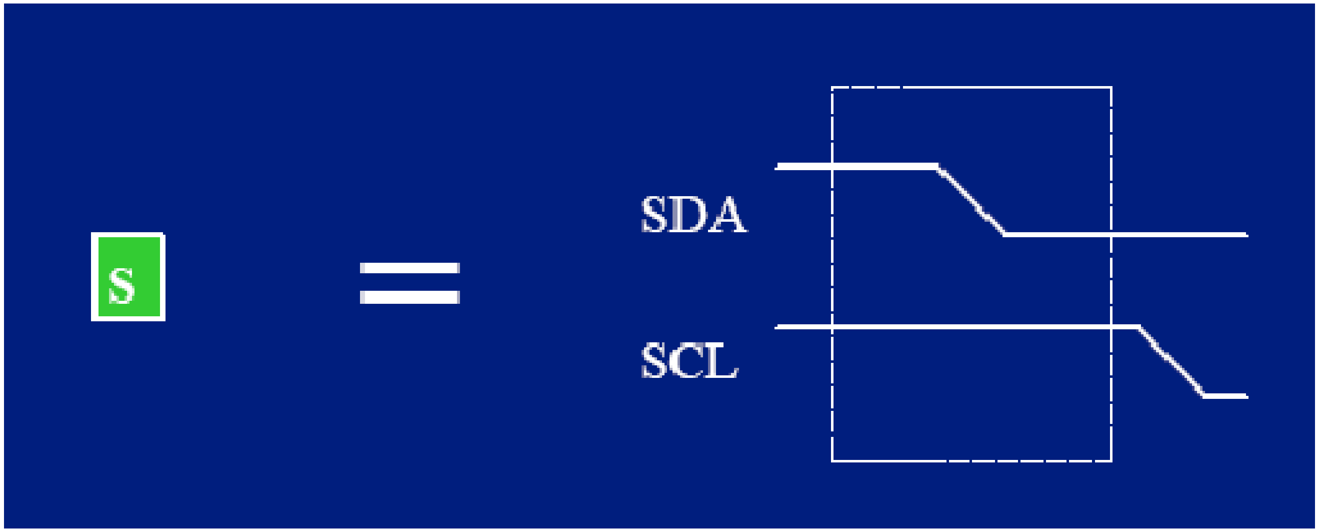
# I2C - Elements

- Building Blocks of I2C
  - I2C consists of many “conditions” which to simplify this presentation will be represented as “elements”.



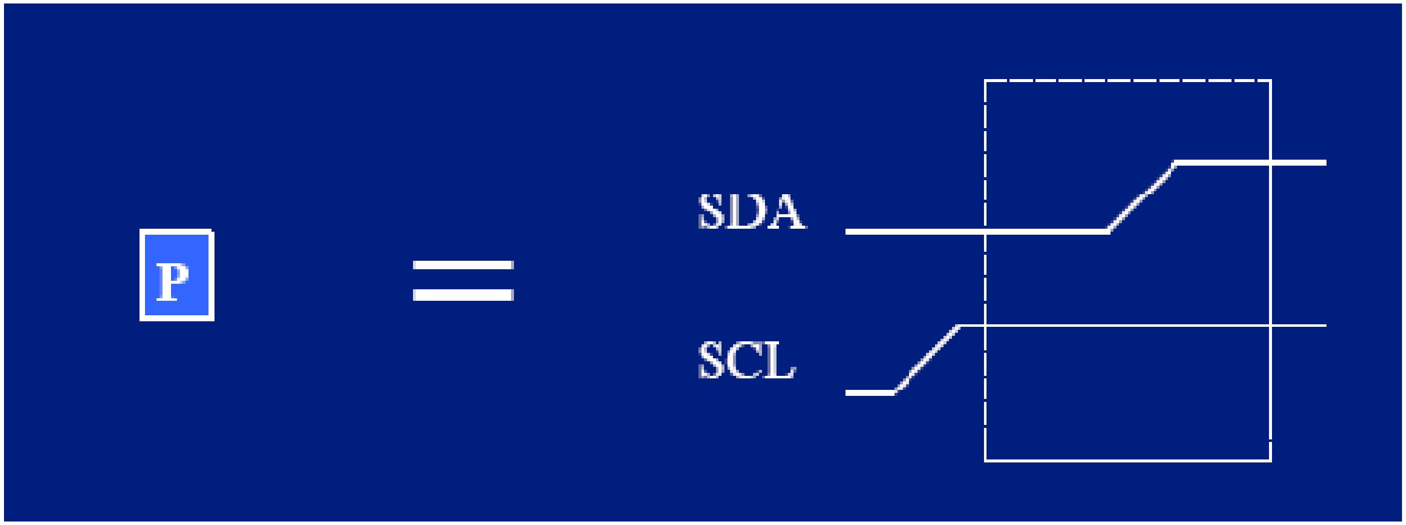
# I2C - Elements

- Start Condition
  - ❑ Initializes I2C Bus
  - ❑ SDA is pulled low while SCL is high



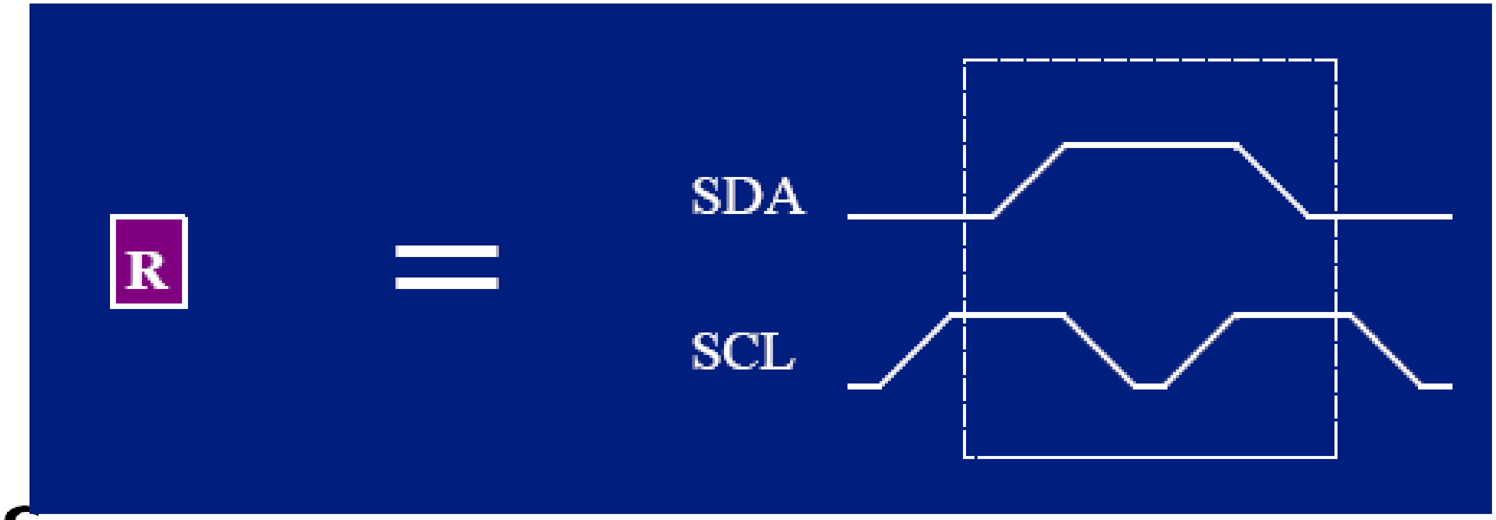
# I2C - Elements

- Stop Condition
  - Releases I2C Bus
  - SDA is released while SCL is high



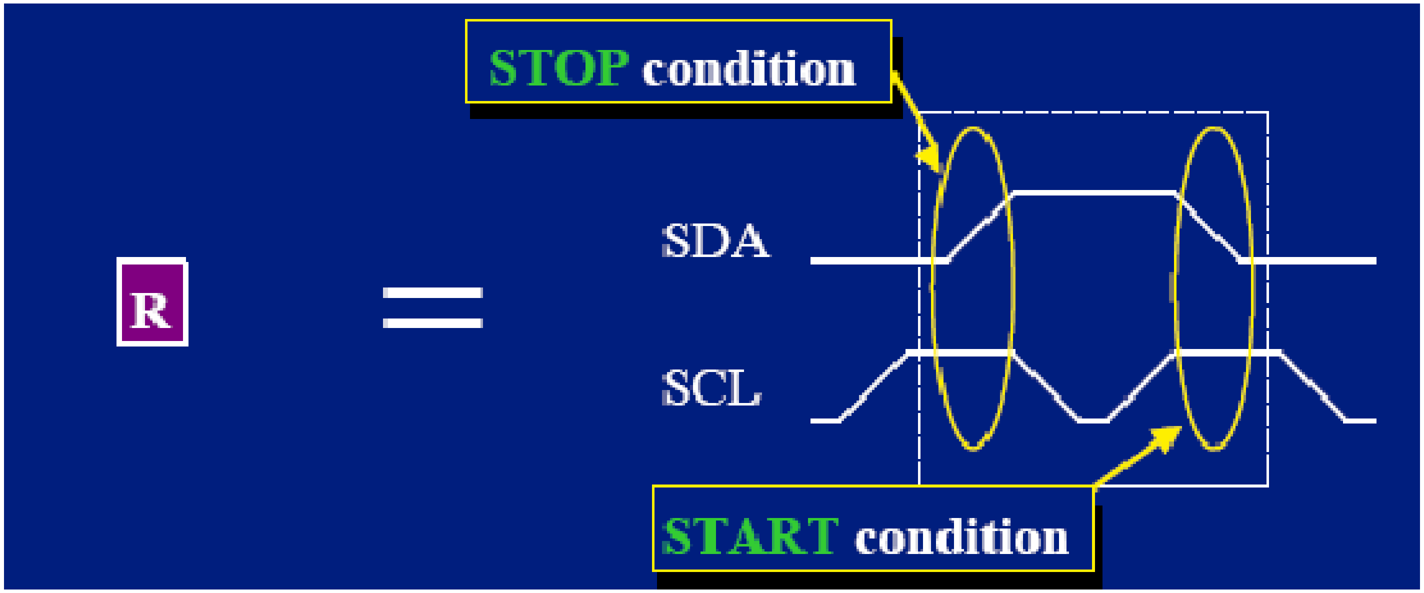
# I2C - Elements

- Restart Condition
  - Reinitializes I2C Bus
  - Used when START does NOT follow STOP



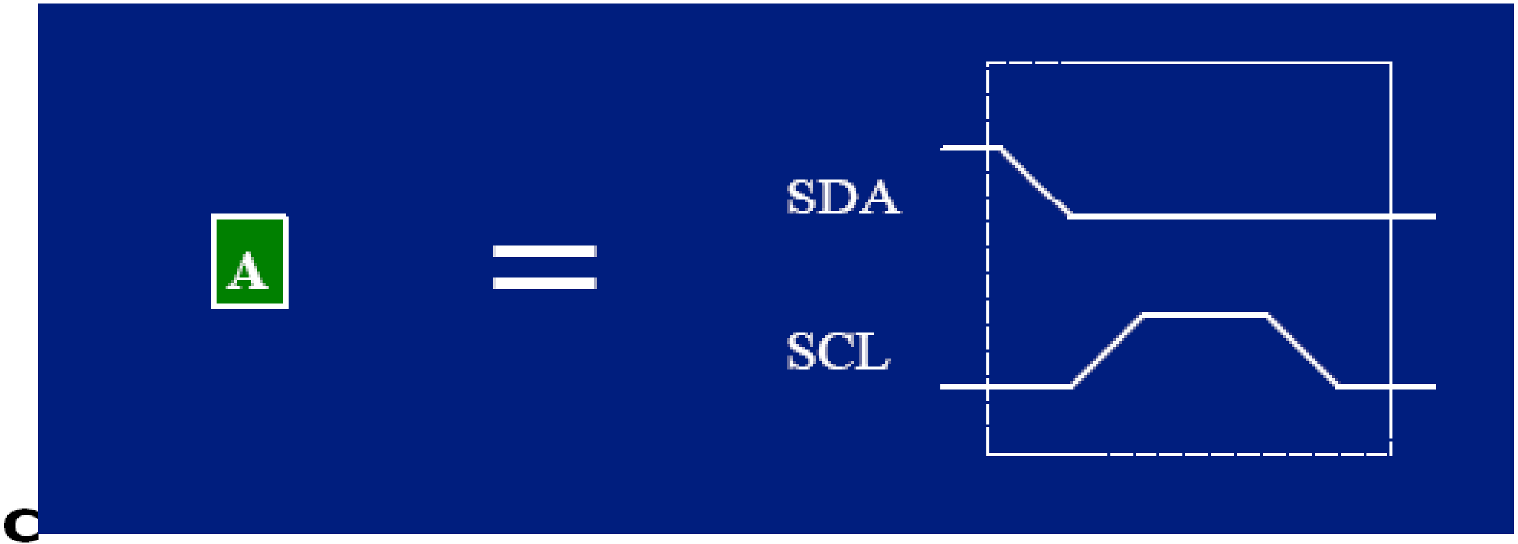
# I2C - Elements

- Restart Condition
  - Reinitializes I2C Bus
  - Used when START does NOT follow STOP



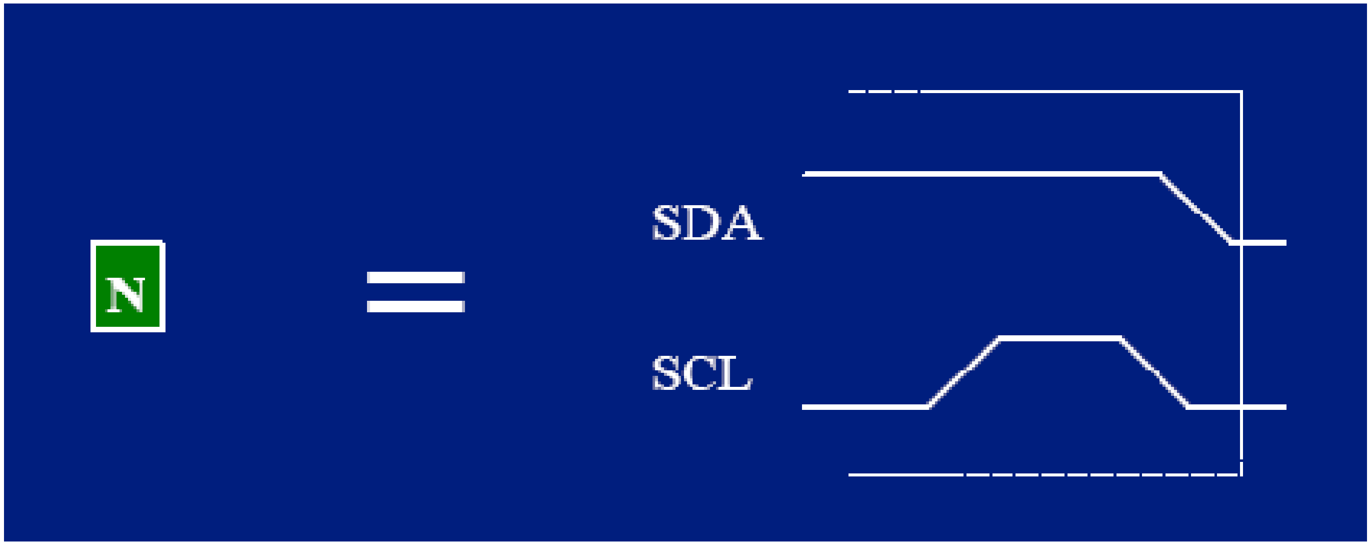
# I2C - Elements

- ACK Condition
  - Acknowledges a data transfer
  - ACK is when the recipient drives SDA low

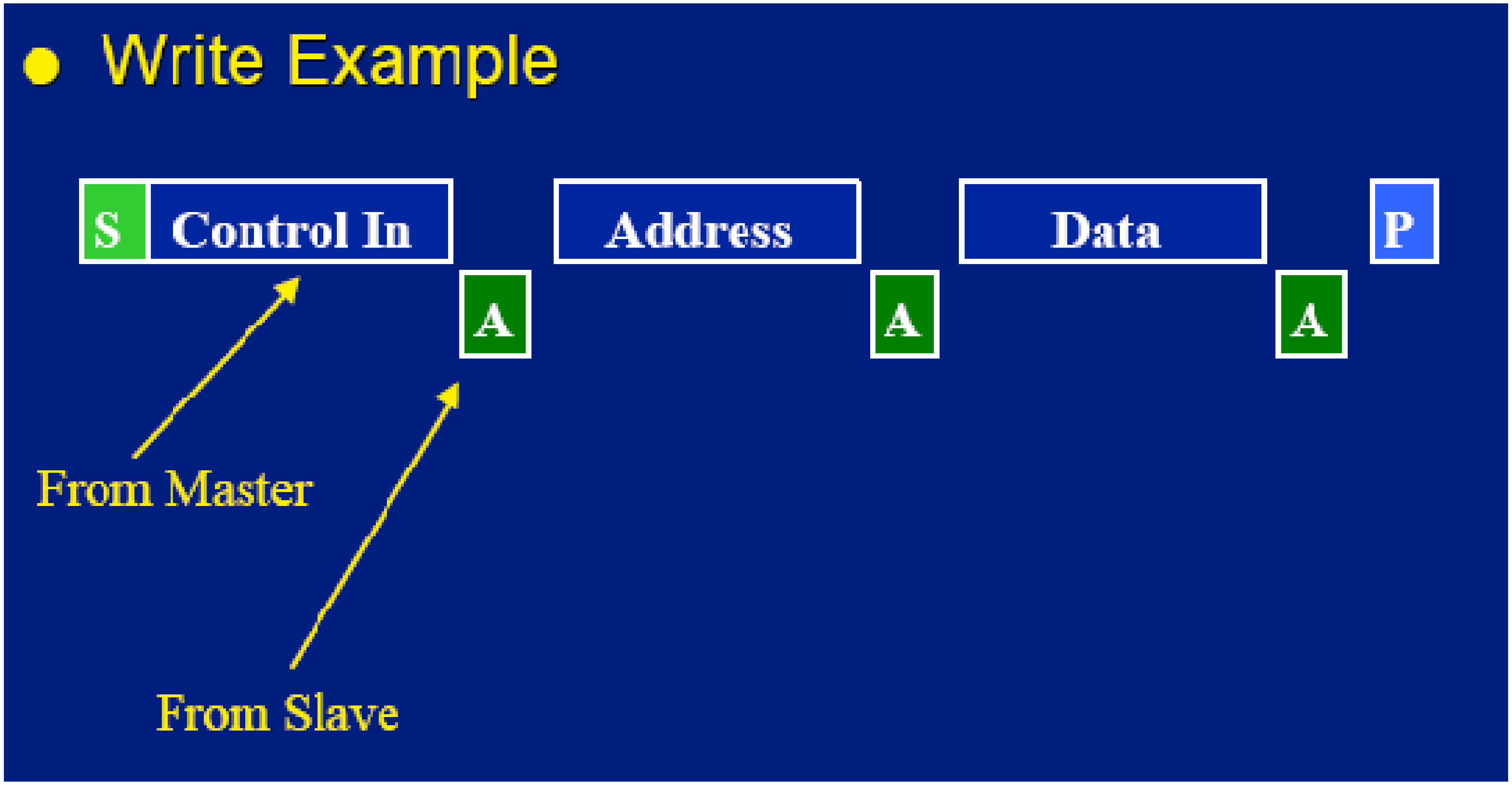


# I2C - Elements

- NACK Condition
  - Negatively acknowledges a data transfer
  - NACK - the recipient does NOT drive SDA low



# Writing to a I2C EEPROM



- Write Example

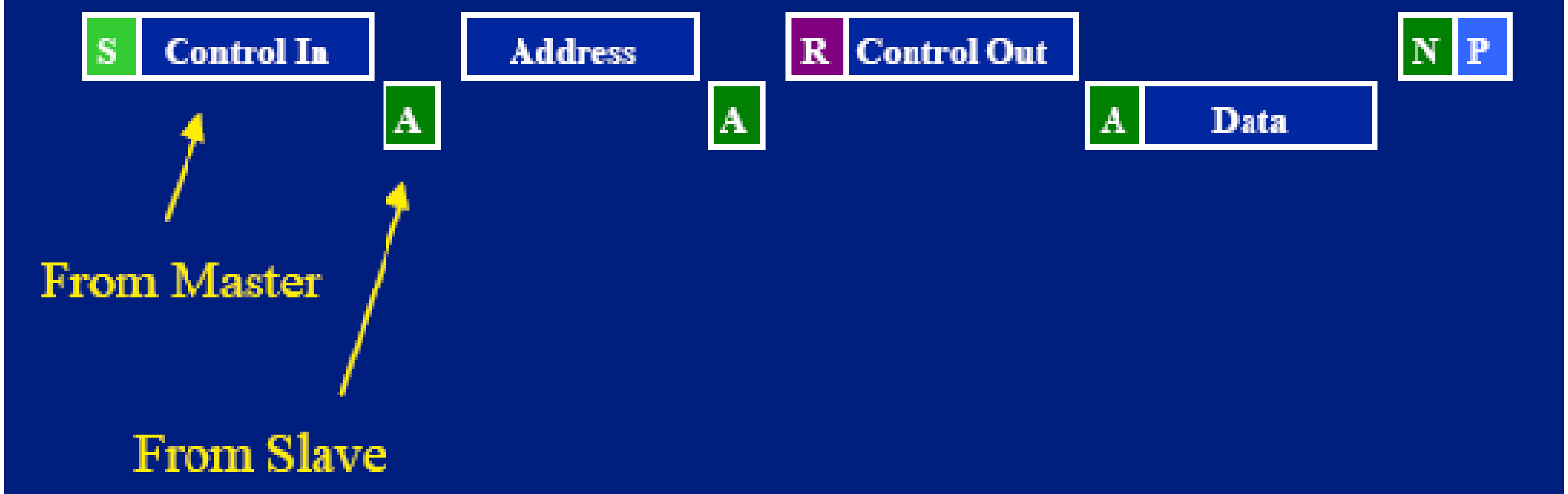
S | Control In | A | Address | A | Data | A | P

From Master

From Slave

# Reading from an I2C EEPROM

- Read Example



S Control In

A

Address

A

R Control Out

A Data

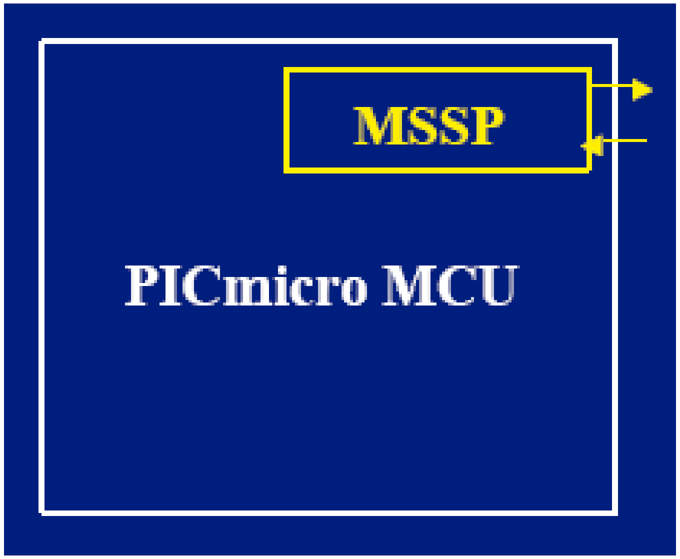
N P

From Master

From Slave

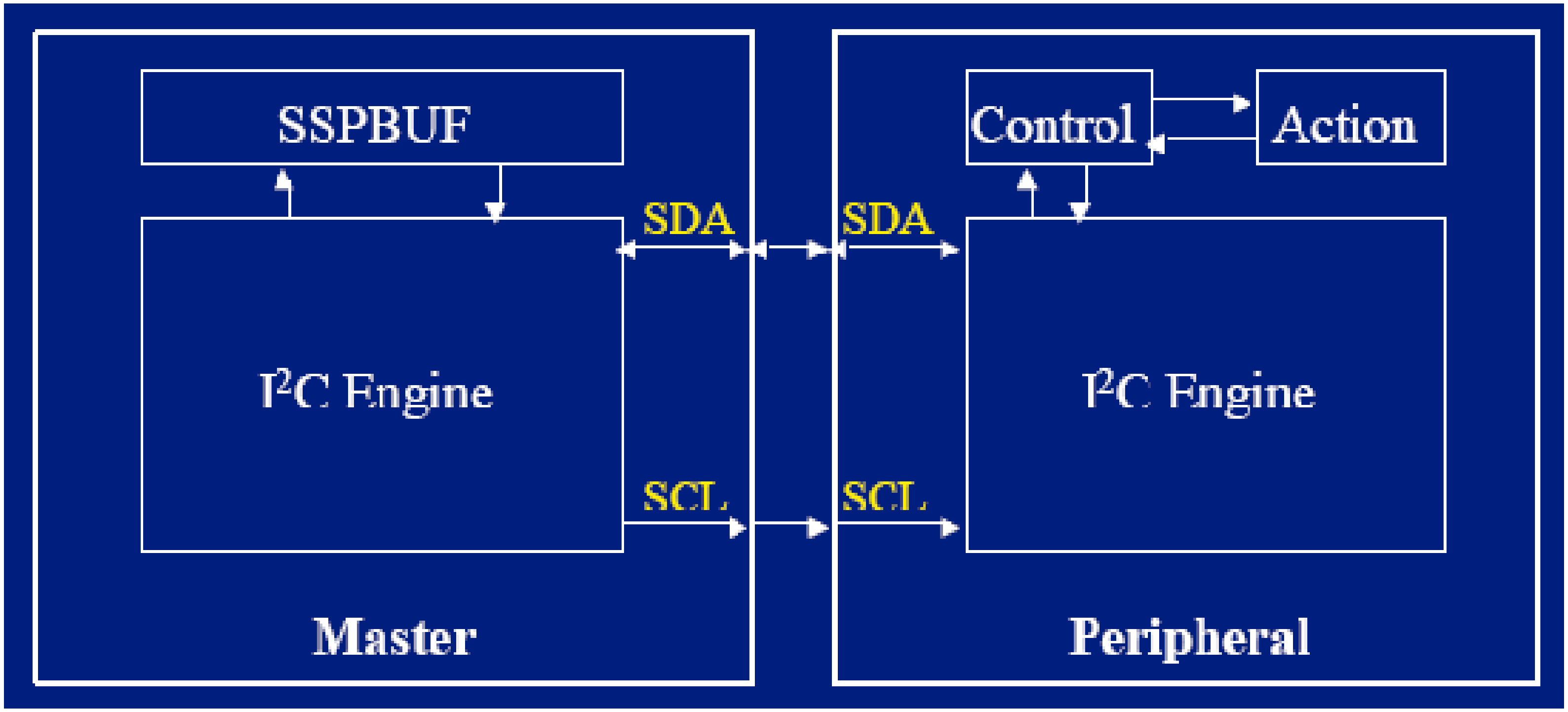
# I2C in the PICmicro MCU

- The MSSP module in the PICmicro microcontroller (MCU) allows I2C and other synchronous serial protocols



# I2C in the PICmicro MCU

- I2C Data Transfer



# THANK YOU…..

Quest innovative Solutions Pvt. Ltd.
1st Floor, MKS Towers, S. A. Road
Kadavnathra, Cochin-20
Phone: +91 484 2204108/9 email: training@qis.co.in
www.qis.co.in