MECE336 Microprocessors I
Input/Output Ports

Dr. Kurtuluş Erinç Akdoğan

kurtuluserinc@cankaya.edu.tr

Course Webpage: http://MECE336.cankaya.edu.tr

ÇANKAYA ÜNİVERSİTESİ
MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ

Why Digital Input/Output?

- Almost any embedded system needs to transfer digital data between its CPU and the outside world. This transfer falls into a number of categories, which can be summarised as:
  - **Direct user interface**, including switches, keypads, light emitting diodes (leds) and displays;
  - **Input measurement** information, from external sensors, possibly being acquired through an analog to digital converter;
  - **Output control** information, for example to motors or other actuators;
  - **Bulk data transfer** to or from other systems or sub-systems, moving in serial or parallel form, for example sending serial data to an external memory.
- How can we provide the required interface between the microcontroller core and the outside world? More precisely, how do we get the data onto or off the data bus at the right moment?

Parallel Port

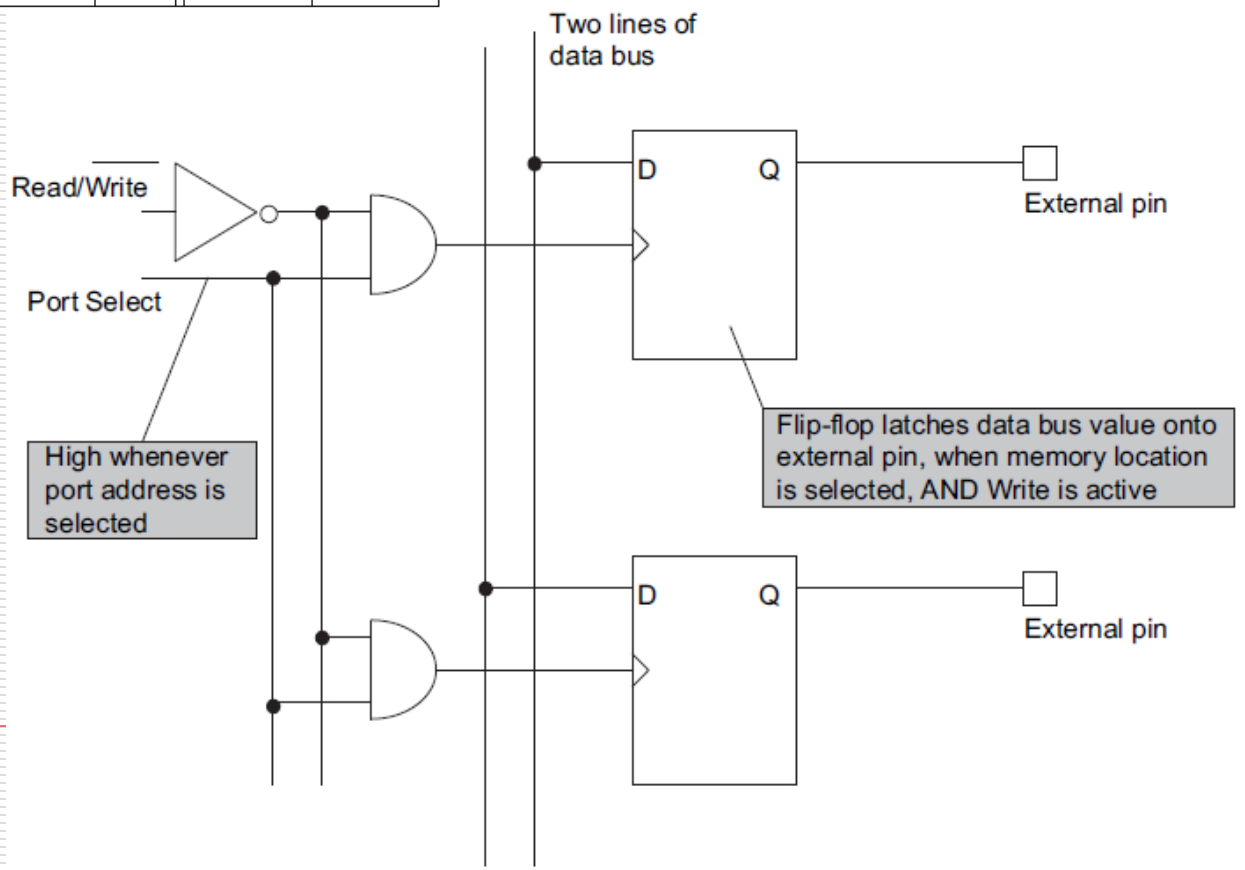
- With this plethora of data coming and going, it is likely we will need to have a variety of digital inputs and/or outputs.
- These are divided broadly into
  - **serial** and
  - **parallel**.
- In serial data transfer, the information is transferred one bit at a time. Only a single interconnection is used to carry the data itself
- In parallel data transfer, a set (for example, eight) of interconnections is used. Each of these can carry 1 bit, and each works in parallel with the others.
- Data can thus be transferred in groups of bits, for example in bytes.
- Parallel input/output (I/O) is the workhorse for all the basic data interchange of a microcontroller, including interfacing with switches, LEDs, displays and so on.
- A group of parallel I/O interconnections, appearing on the pins of the microcontroller, is called a '**parallel port**'.

Building A Parallel Interface Output port

- Let us create an 'output' port.
- Let us assign an address in the memory map to the port.
- Whenever that address is selected by an instruction in the program, it activates a line called 'Port Select'.
-A further line, 'Read/Write', indicates whether the CPU is undertaking a Read (line is high) or Write (line is low) operation.
-This is gated with the Port Select line.
- Each line of the data bus is connected to a bistable, and all of these are clocked by the Port Select line.
-Then the value of the data bus is latched into the bistable whenever the port memory location is addressed, in Write mode.
-The outputs of the bistables are made available for connection to the outside world.

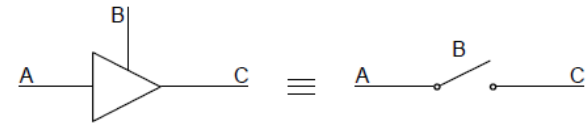
E/C	D	Q	$\overline{Q}$
0	X	Q_{old}	$\overline{Q_{old}}$
1	0	0	1
1	1	1	0

-Flip-Flop is the basic storage element in sequential logic that stores one bit of data
-Two stable states: 0 and 1
-Signal at D is stored in output Q if input E is 1



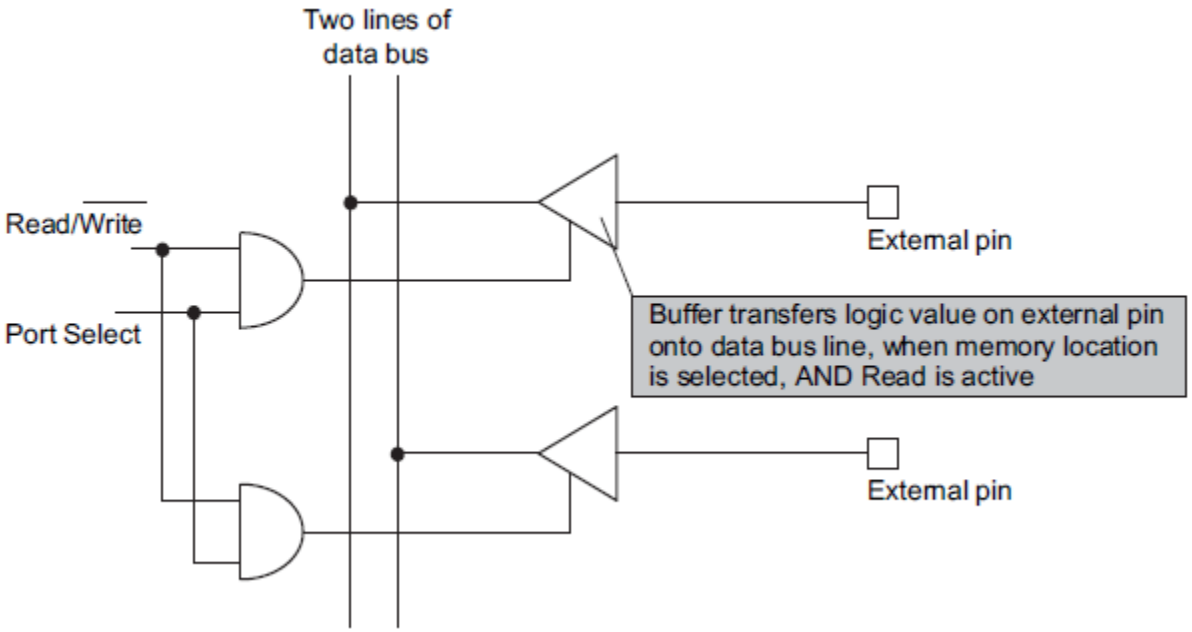
Input Port

-Let us create a set of input pins.
-All that is needed is a tristate buffer gate connected between an external pin and a line of the data bus.
-When the buffer is enabled, again by a logical combination of Port Select line and Read/Write control, the logic value of the external pin is briefly connected to the data bus line, and can be read by the CPU.
-Note that in this design the external data is not latched by the port; it must be held at a stable value by the external source.



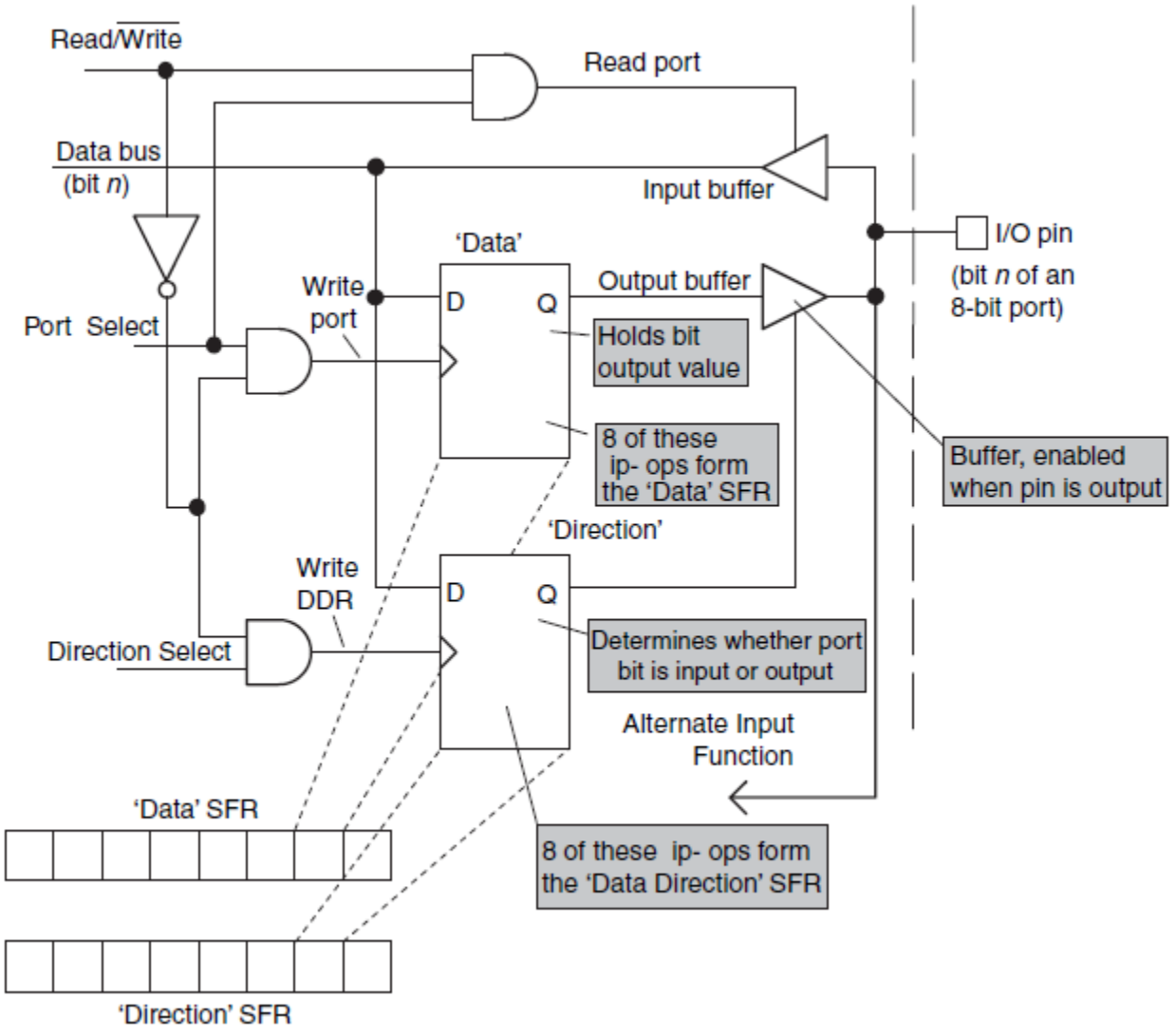
A	B	C
X	0	Z (high impedance)
0	1	0
1	1	1

-Output port C has value of input A if input B is 1
-Output port assumes high impedance if input B is 0
-Multiple circuits can share the same output line



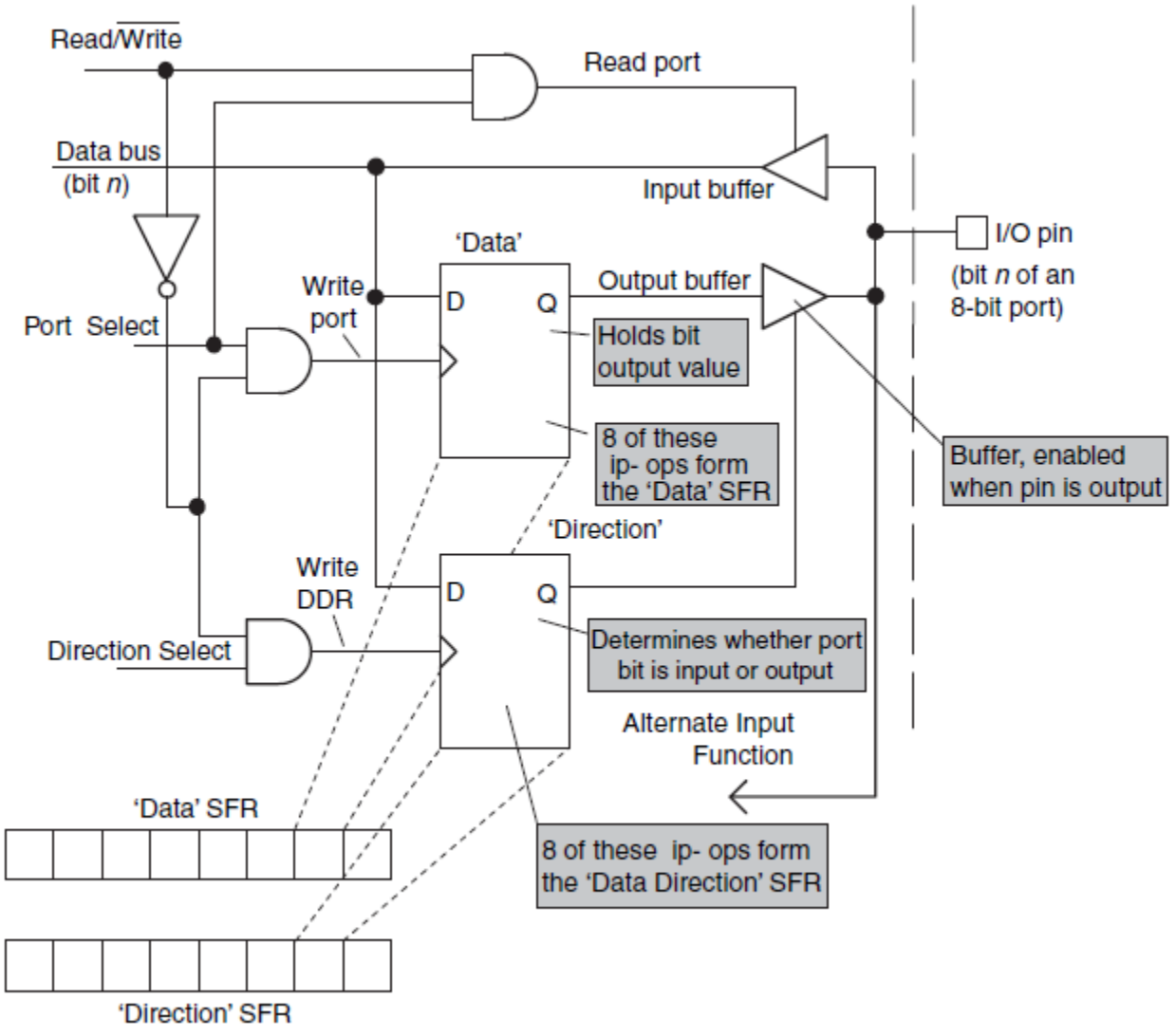
Bi-directional Input/Output Pin Driver

- User should select an external pin of an IC to function as input or output.
-A flip-flop ('Direction'), is added to determine whether this microcontroller pin is to act as an input or output.
- The state of this flip-flop is set by the program.
- It controls the 'Output buffer', which is enabled when the port bit is in output mode.
- This circuit forms the basis for a very useful bi-directional input/output pin driver, and it is easy to find versions of it in many popular microcontrollers.



Bi-directional Input/Output Pin Driver

- Sets of I/O pins are grouped together to form a parallel I/O port.
- Each 'Data' flip-flop then forms one bit of a 'Data' SFR (Special Function register), and
-Each 'Direction' flip-flop forms one bit of a 'Direction' SFR.
-Each SFR is memory mapped, with its own unique address.
-Derived from that address is its select line, which goes high when that location is addressed.
-'Port Select' selects the Data SFR and 'Direction Select' selects the Direction SFR.
- By writing to the Direction SFR the user can determine which bits are to be input and which are to be output.
-By writing to the Data SFR, output is set.
-By reading SFR input is read.



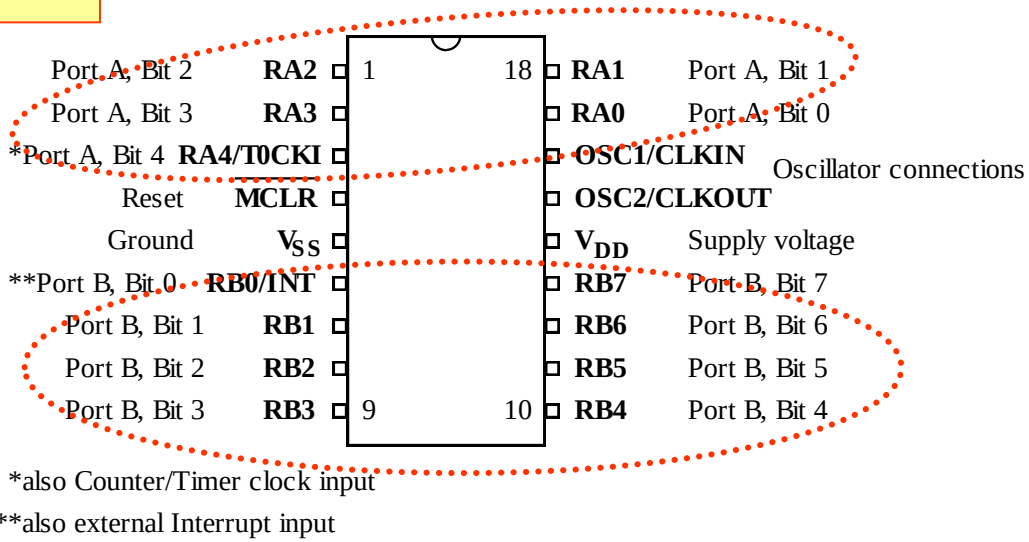
The PIC 16F84A parallel ports

-The Port SFRs can be seen in the memory map, the detail repeated here.
-SFR named PORTX holds the input/output data for the port, ie it holds all the "Data Latch" bits for that port.
-The SFR named TRISX holds all the "TRIS Latch" bits for that port. The bits can be set independently, so one can be input while another output. They cannot be both at the same time.
-'TRIS latch' in replaces the 'Direction' latch of the earlier diagram

05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h

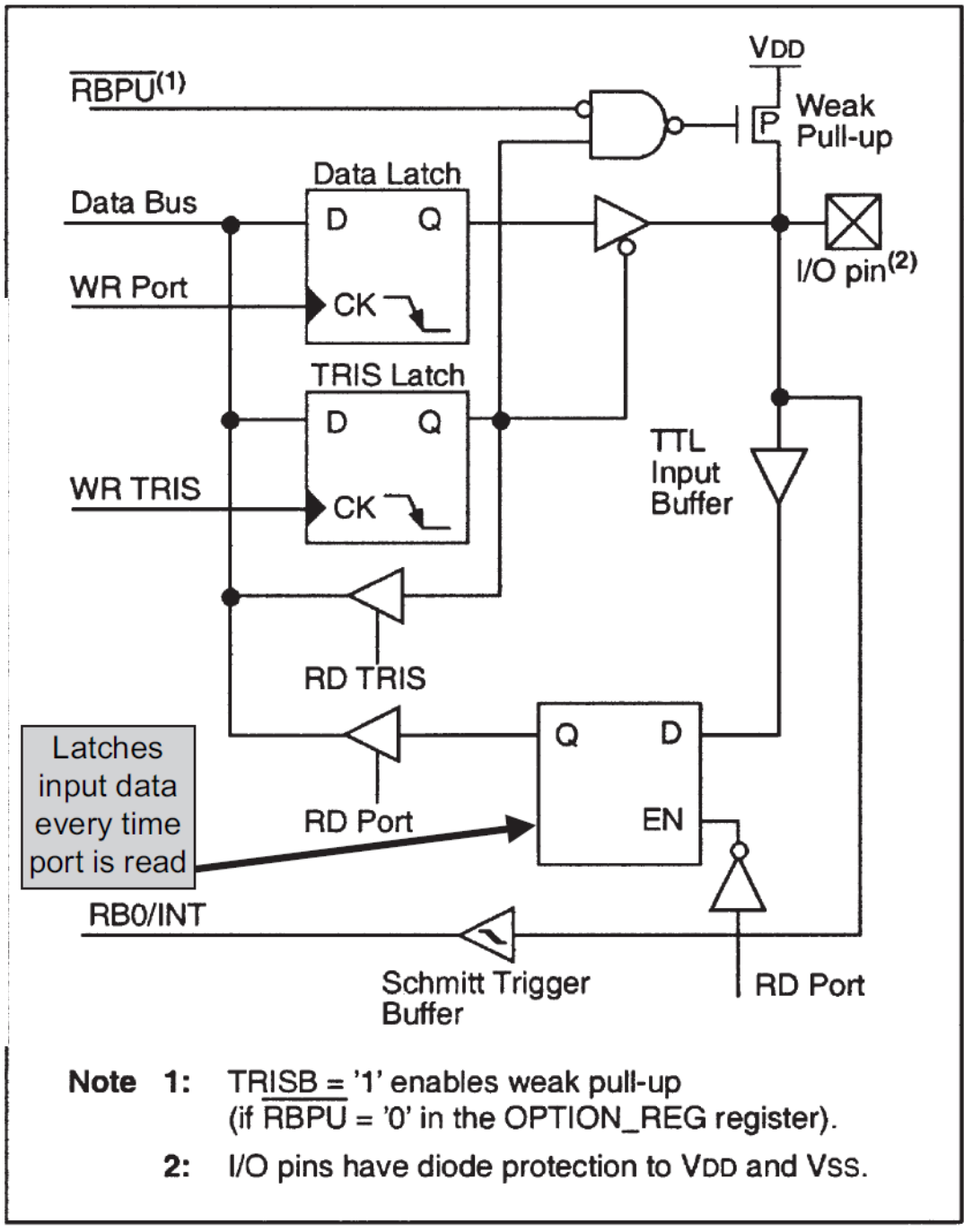
Ports A and B can easily be found on the pin connection diagram. Port A is only 5 bits, while Port B is 8.

Note that some pins have several functions, as indicated on the diagram.



Port B:
Pin RB0 to RB3

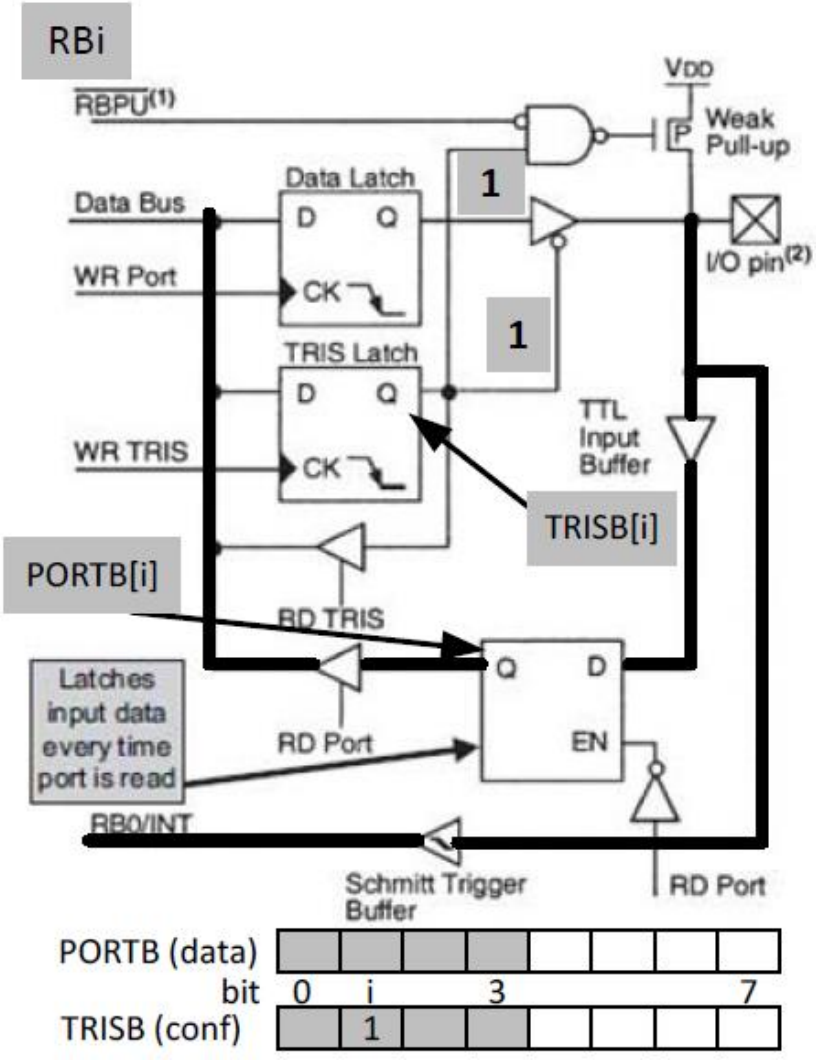
-It can be seen that if the 'TRIS latch' output is set to 0, then the buffer that it drives is enabled and the port bit is in output mode.
-The incoming data is latched, through the lowest latch in the diagram, rather than just its instantaneous value being read.
-Data Latch holds data value
-Write data: trigger WR Port
-Read data: trigger RD Port
-Direction in TRISB register
Value 1: input
Value 0: output
-Write TRIS: trigger WR TRIS
-Read TRIS: trigger RD TRIS
-Schmitt trigger for RB0
-Weak pull-up resistors for input operation



Port B: RB0 to RB3 Input Operation

Description for Pin RBi

- Pin RBi as input
  - TRISB register bit i is 1
  - Output of TRIS latch is 1
    ⇒ Tri-state buffer at output is disabled
- Input value is latched during data read (RD Port is 1)
- Use weak pull-up: clear bit **RBPU** in the **OPTION** register



Port B: Schmitt-Trigger at RB0/INT

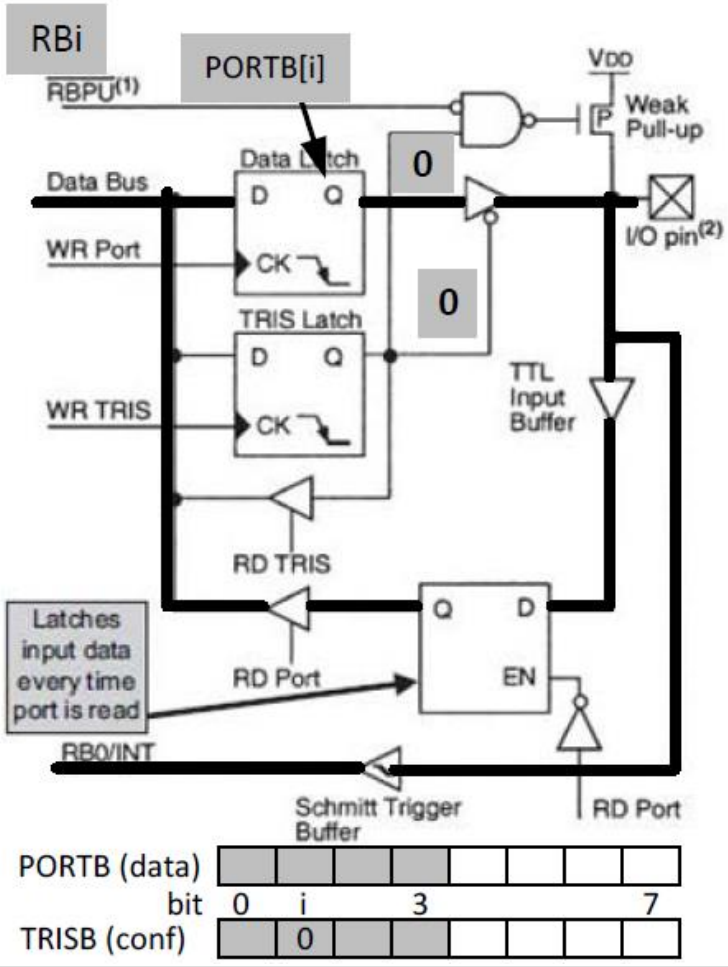
Operation

- Special logic gate input that "cleans up" a corrupted logic signal
- When output signal is logic 0, the input signal has to pass the positive going threshold to obtain logic 1
- When the output signal is logic 1, the input signal has to pass the negative going threshold to obtain logic 0
  - small fluctuations in the input signal do not change the logic level

Port B: RB0 to RB3 Output Operation

Description for Pin RBi

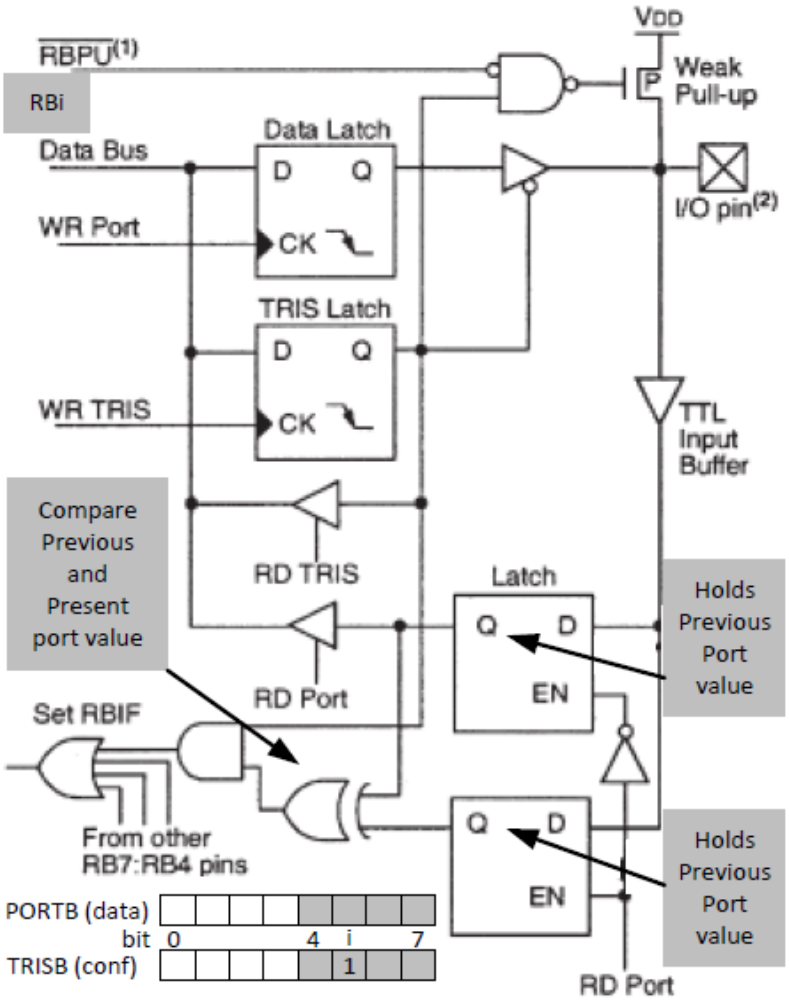
- Pin RBi as output
  - TRISB register bit i is 0
  - Output of TRIS latch is 0
    ⇒ Tri-state buffer at output is enabled
    ⇒ Weak pull-up is disabled
- Direct transfer of data latch value to output pin



Port B: RB4 to RB7

Description for Pin RBi

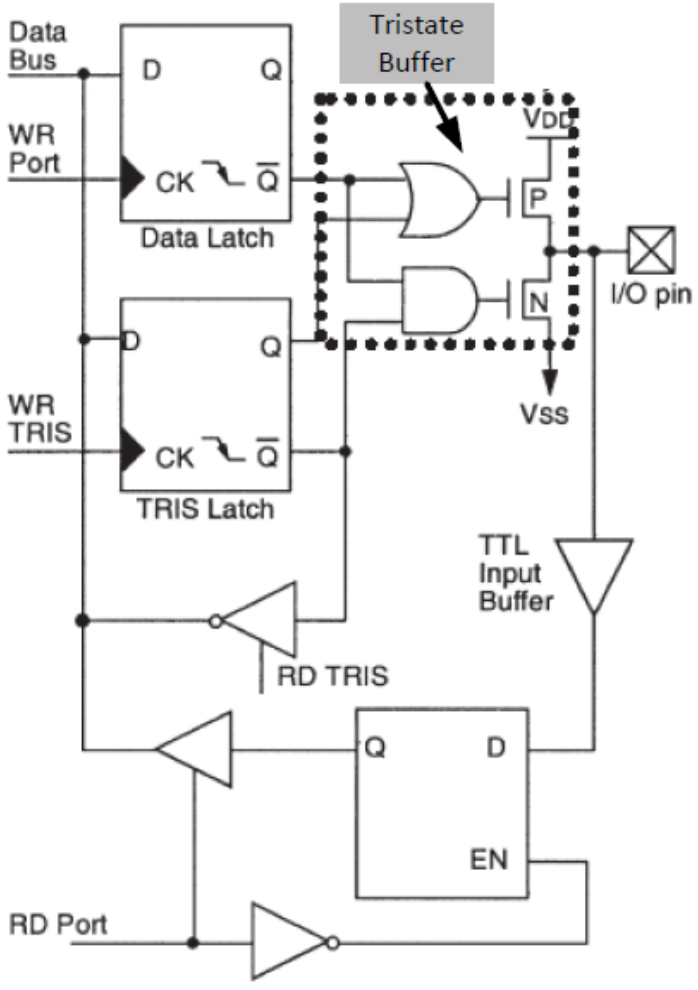
- Same functionality as Port RB0–RB3
- Additional functionality: Data value from previous read operation is stored in second latch
  ⇒ An interrupt can be generated if the data value changes



Port A: RA0 to RA3 General

Description

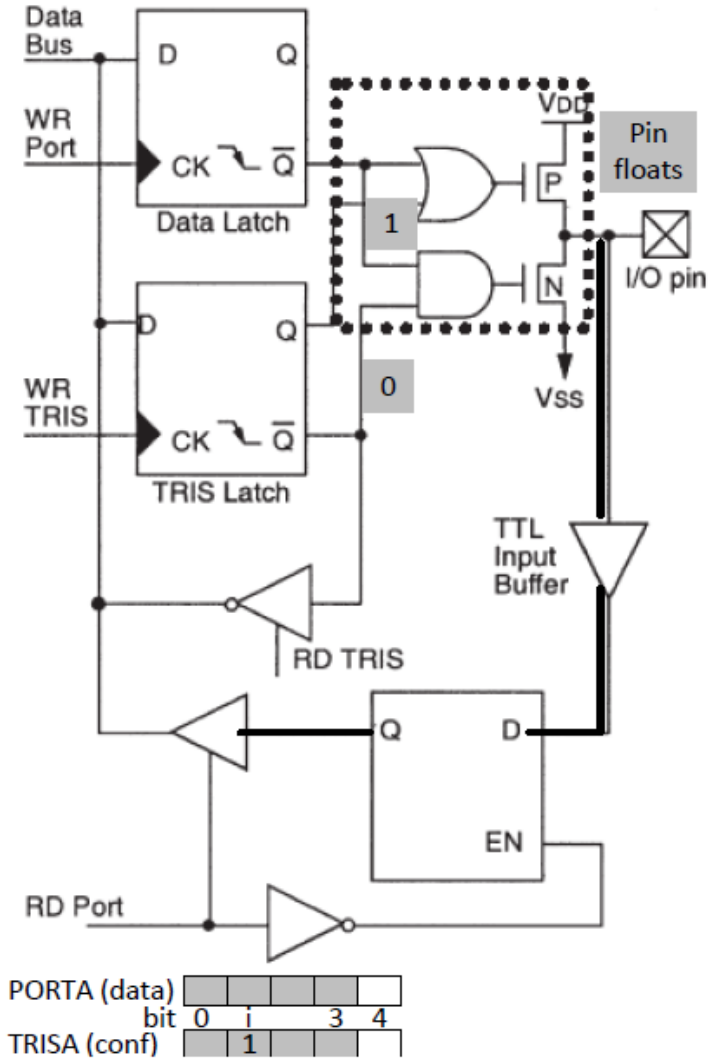
- Data Latch holds data value
  - Write data: trigger WR Port
  - Read data: trigger RD Port
- Direction in TRISA register
  - Value 1: input
  - Value 0: output
- Write TRIS: trigger WR TRIS
- Read TRIS: trigger RD TRIS
- Latch for input data
- Tri-state buffer for "open drain" output



Port A: RA0 to RA3 Input Conguration

Description for Pin RAi

- Pin RAi as input
  - TRISA register bit i is 1
  - Output of TRIS latch is 1
  - Negated TRIS latch is 0
    ⇒ Tri-state buffer is disabled
- Input value is latched during data read (RD Port is 1)

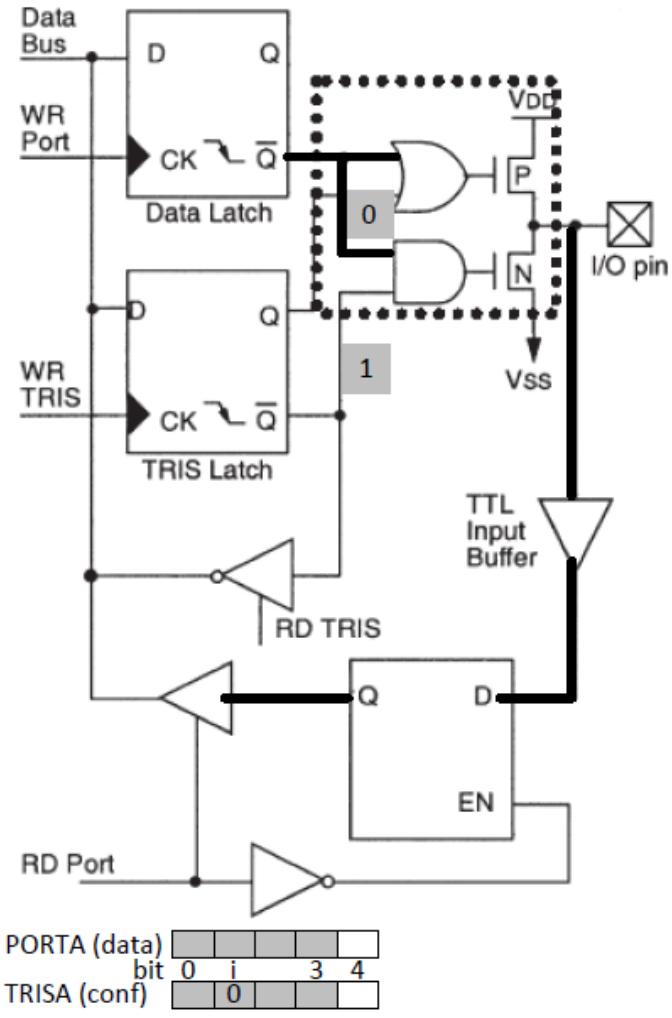


Port A: RA0 to RA3 Output

Description for Pin RAi

- Pin RAi as output
  - TRISA register bit i is 0
  - Output of TRIS latch is 0
  - Negated TRIS latch is 1

$\Rightarrow$ Tri-state buffer provides data value at output



Port B: Programming

Configuration: TRISB register

- TRISB on memory bank 1

```
status  equ   03
bsf     status,5
```

- Write configuration to TRISB

```
movlw B'00000101'
movwf trisb
```

Data Value PORTB

- PORTB on memory bank 0

```
bcf     status,5
```

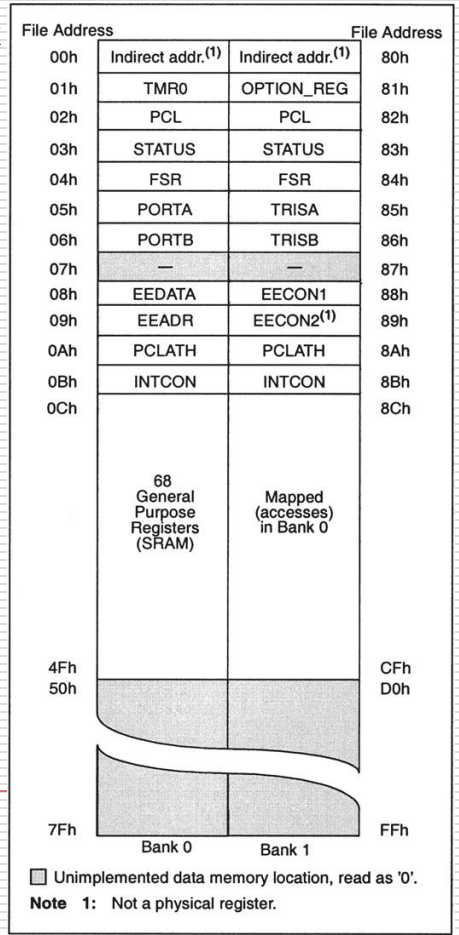
- Write value to Port B

```
movlw B'10100010'
movwf portb
```

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

Bit5: Select RAM memory bank



Port A: Programming

Configuration: TRISA register

- TRISA on memory bank 1

```
status  equ   03
bsf     status,5
```

- Write configuration to TRISA

```
movlw B'00000101'
movwf trisa
```

Data Value PORTA

- PORTA on memory bank 0

```
bcf     status,5
```

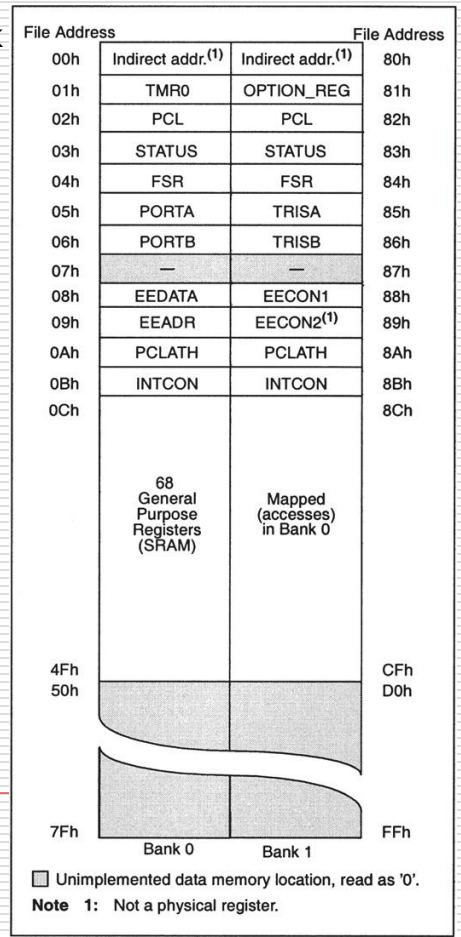
- Write value to Port A

```
movlw B'10100010'
movwf porta
```

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

Bit5: Select RAM memory bank



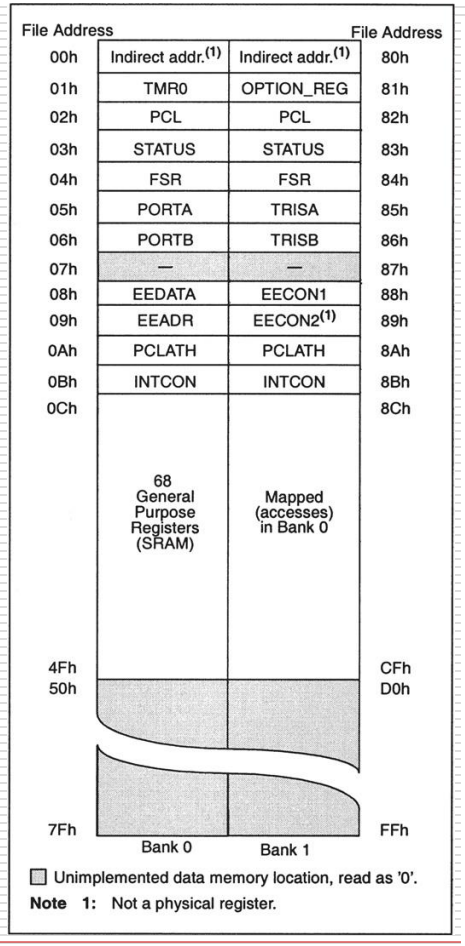
Programming for a target piece of hardware – a simple datatransfer program

-This program just uses the **Status register**, **Ports A and B**, and their control registers **TRISA** and **TRISB**. Labels for these are therefore defined, taking memory addresses directly from the memory map

-The program starts with an initialisation section

-As SFRs are placed in RAM memory bank 1, it is necessary first of all to set bit 5 of the Status register to 1.

-This is done in the first program line, labelled start, using the **bsf** instruction.

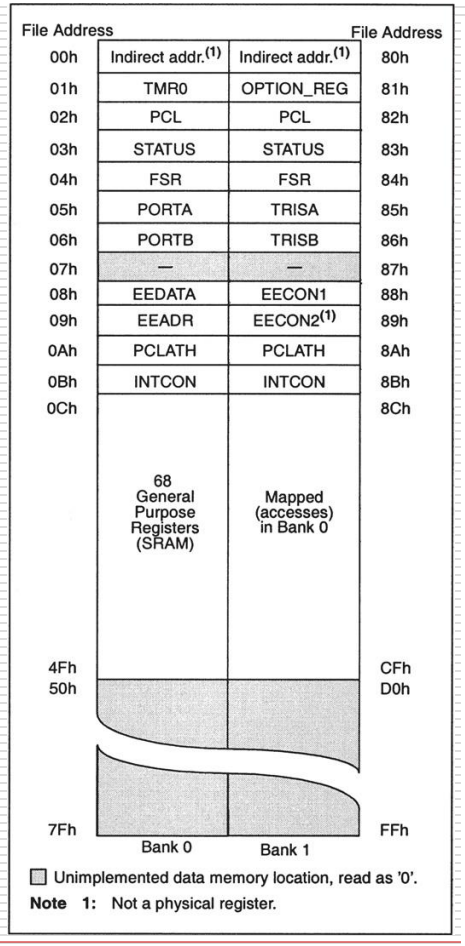



```
;*****************************************************************
;Ping-pong data move
;This program moves push button switch values from Port A to the
;leds on Port B
;TJW 21.2.05                                          Tested 22.2.05
;*****************************************************************
;
;Configuration Word: WDT off, power-up timer on,
;                    code protect off, RC oscillator
;
;specify SFRs
status  equ     03
porta   equ     05
trisa   equ     05
portb   equ     06
trisb   equ     06
;
        org     00
;Initialise
start   bsf     status,5        ;select memory bank 1
        movlw   B'00011000'
        movwf   trisa           ;port A according to above pattern
        movlw   00
        movwf   trisb           ;all port B bits output
        bcf     status,5        ;select bank 0
;
;The "main" program starts here
        clrf    porta           ;clear all bits in ports A
loop    movf    porta,0         ;move port A to W register
        movwf   portb           ;move W register to port B
        goto    loop
        end
```

```
start   bsf   3,5;  select memory bank 1
```

Programming for a target piece of hardware – a simple datatransfer program

-To be an output a **port pin** must have a 0 in its corresponding **TRIS** register bit. It must have a 1 for the bit to be an input. Therefore we must send the word 00011000 to TRISA.
-A similar process is followed for setting up **Port B**.




```
;*****************************************************************
;Ping-pong data move
;This program moves push button switch values from Port A to the
;leds on Port B
;TJW 21.2.05                                          Tested 22.2.05
;*****************************************************************
;
;Configuration Word: WDT off, power-up timer on,
;                    code protect off, RC oscillator
;
;specify SFRs
status  equ     03
porta   equ     05
trisa   equ     05
portb   equ     06
trisb   equ     06
;
        org     00
;Initialise
start   bsf     status,5        ;select memory bank 1
        movlw   B'00011000'
        movwf   trisa           ;port A according to above pattern
        movlw   00
        movwf   trisb           ;all port B bits output
        bcf     status,5        ;select bank 0
;
;The "main" program starts here
        clrf    porta           ;clear all bits in ports A
loop    movf    porta,0         ;move port A to W register
        movwf   portb           ;move W register to port B
        goto    loop
        end
```

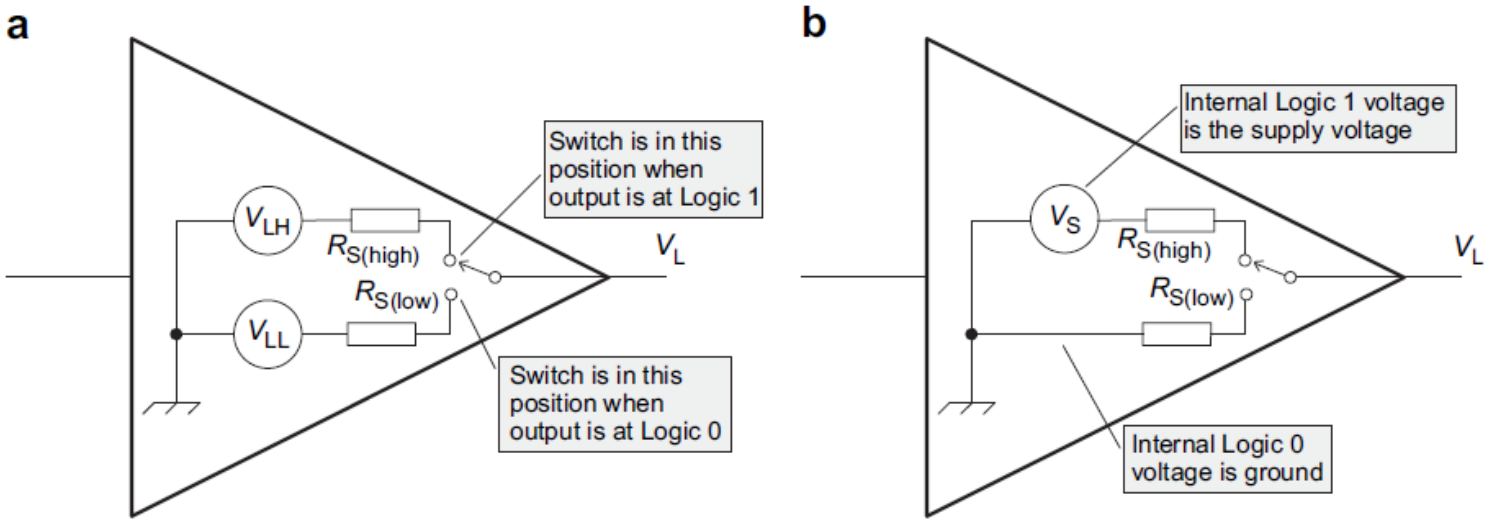
```
start   bsf   3,5;  select memory bank 1
```

Port Electrical Characteristics

- Logic gates are designed to interface easily with each other, and if we connect logic gates from just one family together then we usually don't need to worry about the electrical details of what is going on.
- If, however, we are connecting logic devices (in this case microcontroller port bits) to non-logic elements like
  - **LEDs or**
  - **switches**
- then we do need to understand the electrical characteristics of the logic.
- In particular, we need to understand their input and output characteristics.

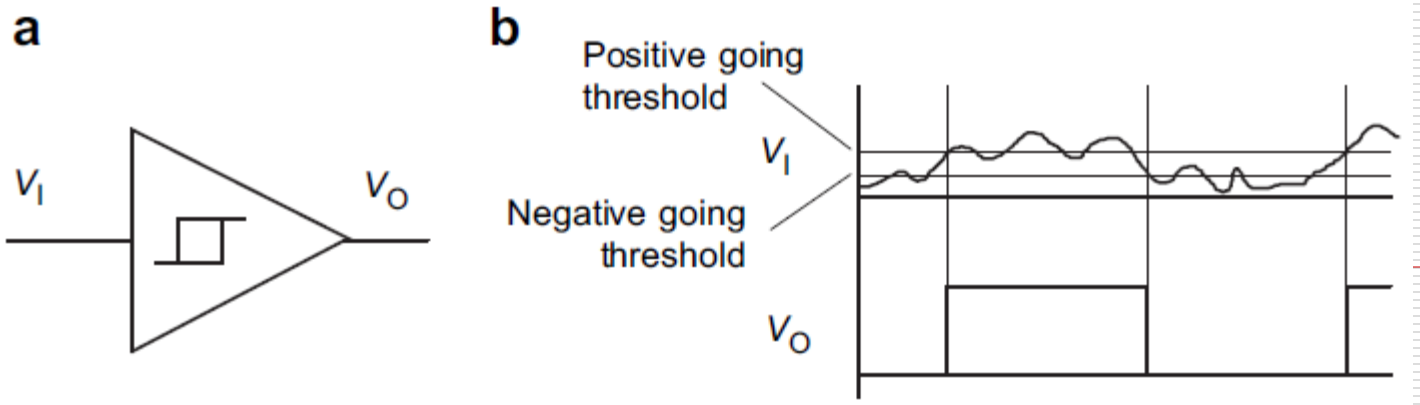
- If the output is at Logic high (or '1'), then the internal switch is in the upper position. It is in the lower position for Logic 0. In either case, the output is modelled as a voltage source in series with a resistor (in circuit theory this is called a 'Thevenin equivalent' circuit).
- VLH is the logic high-output voltage, with an output resistance of RS(high). VLL is the logic low-output voltage, with an output resistance of RS(low).
- VLH is equal to the supply voltage (e.g. 5V), (Logic 1) and VLL is equal to 0 V (Logic 0) if no current is being drawn from the gate output.
- RS(high) and RS(low) are not constant, but depend to some extent on the current being sourced or sunk from the gate output.

Modelling a logic gate output. (a) Generalised model. (b) Model of Complementary Metal Oxide Semiconductor (CMOS) logic gate output



Some special cases: Schmitt trigger inputs

- A Schmitt trigger (Figure 3.5) is a certain type of logic gate input which is designed to 'clean up' a corrupted logic signal.
- It has two input thresholds, with the 'positive-going' higher than the 'negative-going'. A signal starting from a low value has to pass the negative-going threshold (at which point nothing happens) and then cross the 'positive-going' threshold, at which point the output changes state.
- The output will not reverse until the input (now negativegoing) has returned to the negative-going threshold. Thus, small fluctuations which recross a threshold just crossed do not cause any change in output.



The ‘Open Drain’ output

Connecting to the parallel port: *Switches*

- Switches are extensively used in embedded systems.
- Our main initial interest is not to switch directly a voltage or current, but to convert the switch position to a logic level that can be read by a microcontroller port bit.
- Switches are used as a direct user interface in the form of push-buttons, toggle switches, slide switches, or as thumbwheel or rotary switches.
- They are also used, in the form of microswitches, to detect certain types of mechanical movement.

push-button

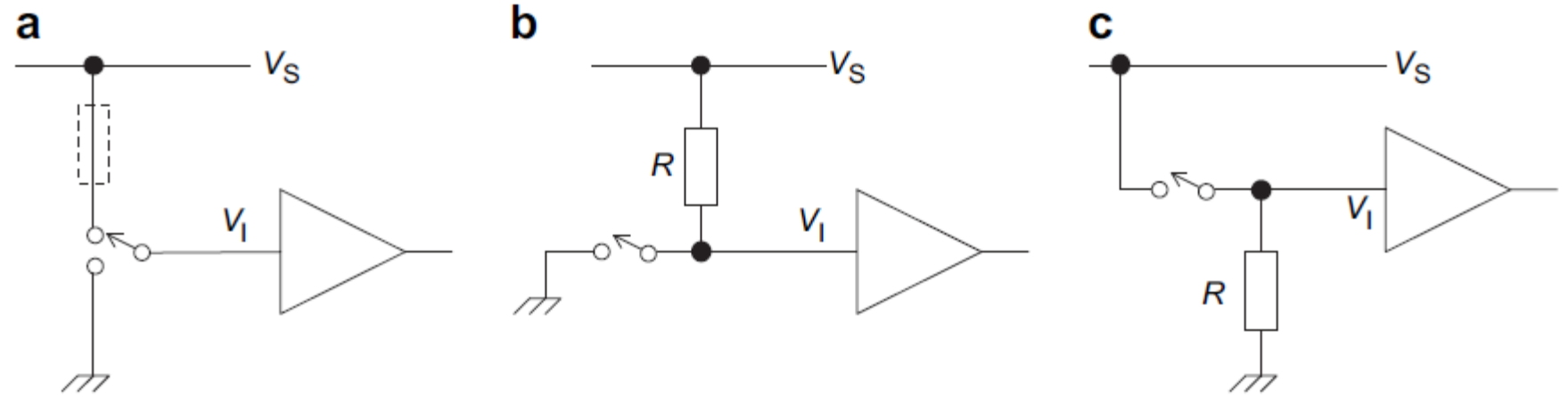
toggle switch

slide switch

microswitch

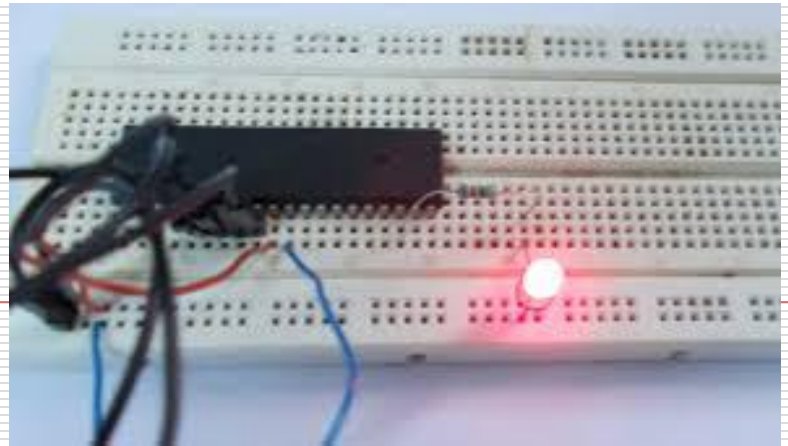
- The simplest way of deriving a logic level from a switch is using a Single-Pole, Double-Throw (SPDT) switch, with one terminal connected to ground and the other to the supply. The switch wiper simply selects one of these two as the logic input.
- Some logic families advise against direct connection of a logic input to the supply voltage, so a series resistor (shown dotted) might be in order.
- Single-Pole, Single-Throw (SPST) switch, for example a pushbutton, has a pull-up resistor is connected to one terminal of the switch, with the other terminal connected to ground.
- If the switch is closed, then the input to the logic gate, VI, is 0V and a current VS/R flows to ground.
- If the switch is open then VO is equal to VS. To reduce wasted current when the switch is closed, the value of R should be high.
- For PIC microcontrollers, pull-up values in the range 10–100 kΩ are usually appropriate.
- The switch circuit of Figure (b) can be reconnected as in Figure (c)

Connecting switches to logic inputs. (a) SPDT connection. (b) SPST with pull-up resistor. (c) SPST with pull-down resistor



Light-Emitting Diodes

- In certain semiconductor materials light is emitted as current flows across a forward-biased p–n junction. LEDs exploit this phenomenon.
- LEDs are widely available in red, green and yellow, as single devices, and as arrays, bargraphs and alphanumeric displays.
- Because they are diodes, LEDs display the normal voltage–current relationship of a forward-- biased diode. This means that, to a reasonable approximation, the voltage across an LED is constant if it is conducting. Note, however, that this forward voltage is considerably higher for
- The red LED changes from 1.90 to 2.00 V if the current increases from 5 to 20 mA. For the green it changes from 1.95 to 2.20 V for the same current range.
- Red is the most efficient, which may account for its greater popularity due less current consumption.
- For a single LED to be comfortably visible, it typically requires around 10mA of current.
- Brighter ones may require up to 20 mA, but special low-power devices (such as the highefficiency red) need as little as 1 or 2mA to be seen.



Light-Emitting Diodes

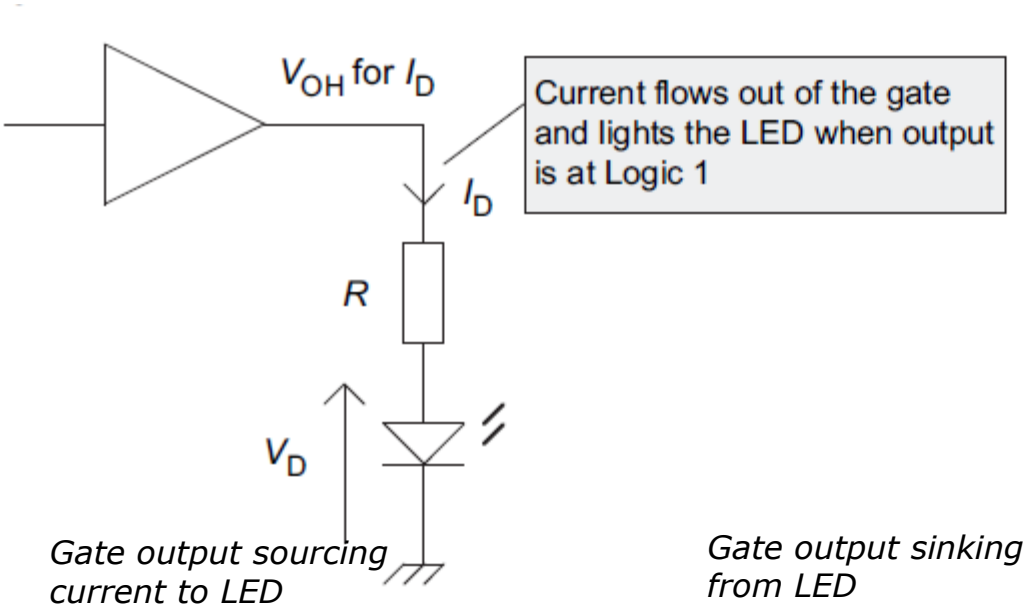
- An LED can be driven from a logic output, for example a microcontroller port, as long as its current requirements can be met. Depending on the capabilities of the port output they can be connected so that the output is sourcing current or sinking current
- Data for the 16F84A (when powered from 5V) shows an output resistances of *approximately* 130Ω when at logic high, and 36Ω when at logic 0.

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

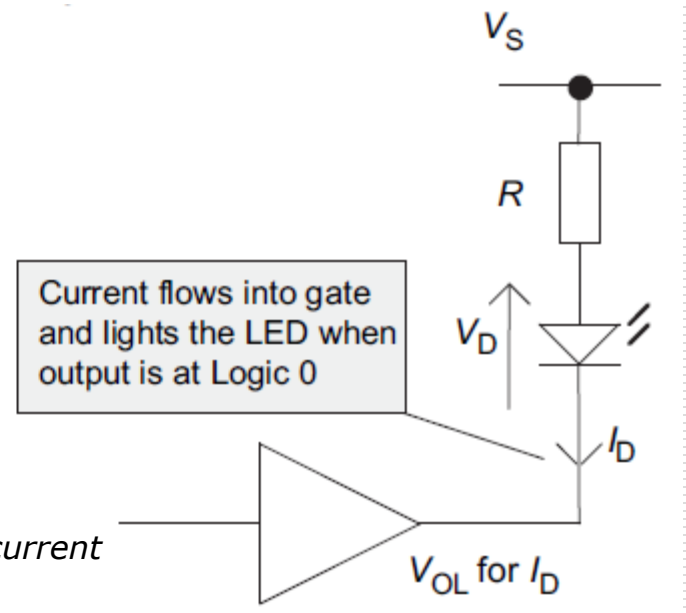
$$R = \frac{V_{OH} \quad V_D}{I_D}$$

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

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



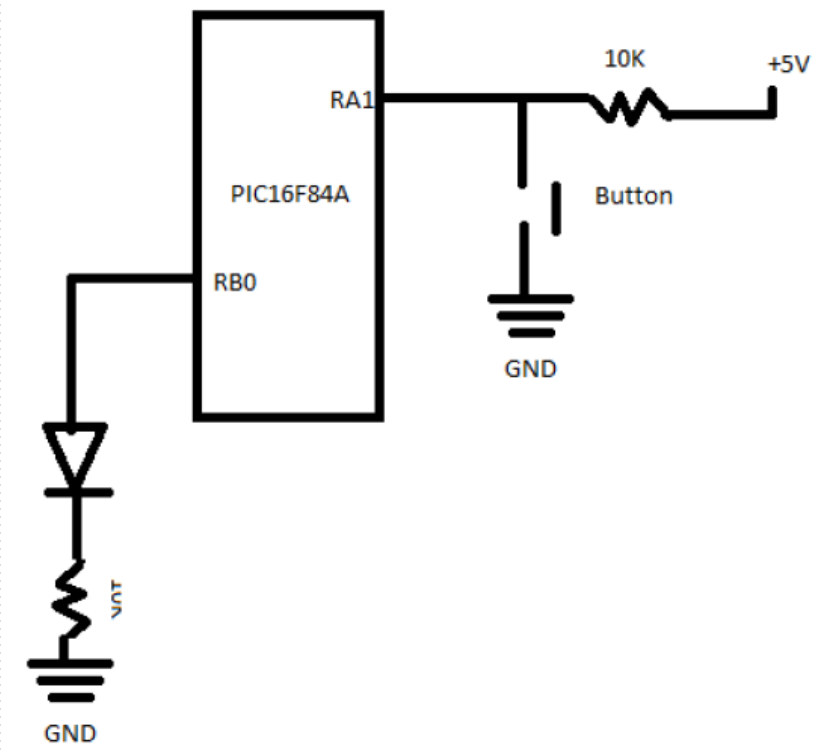
Gate output sourcing current to LED



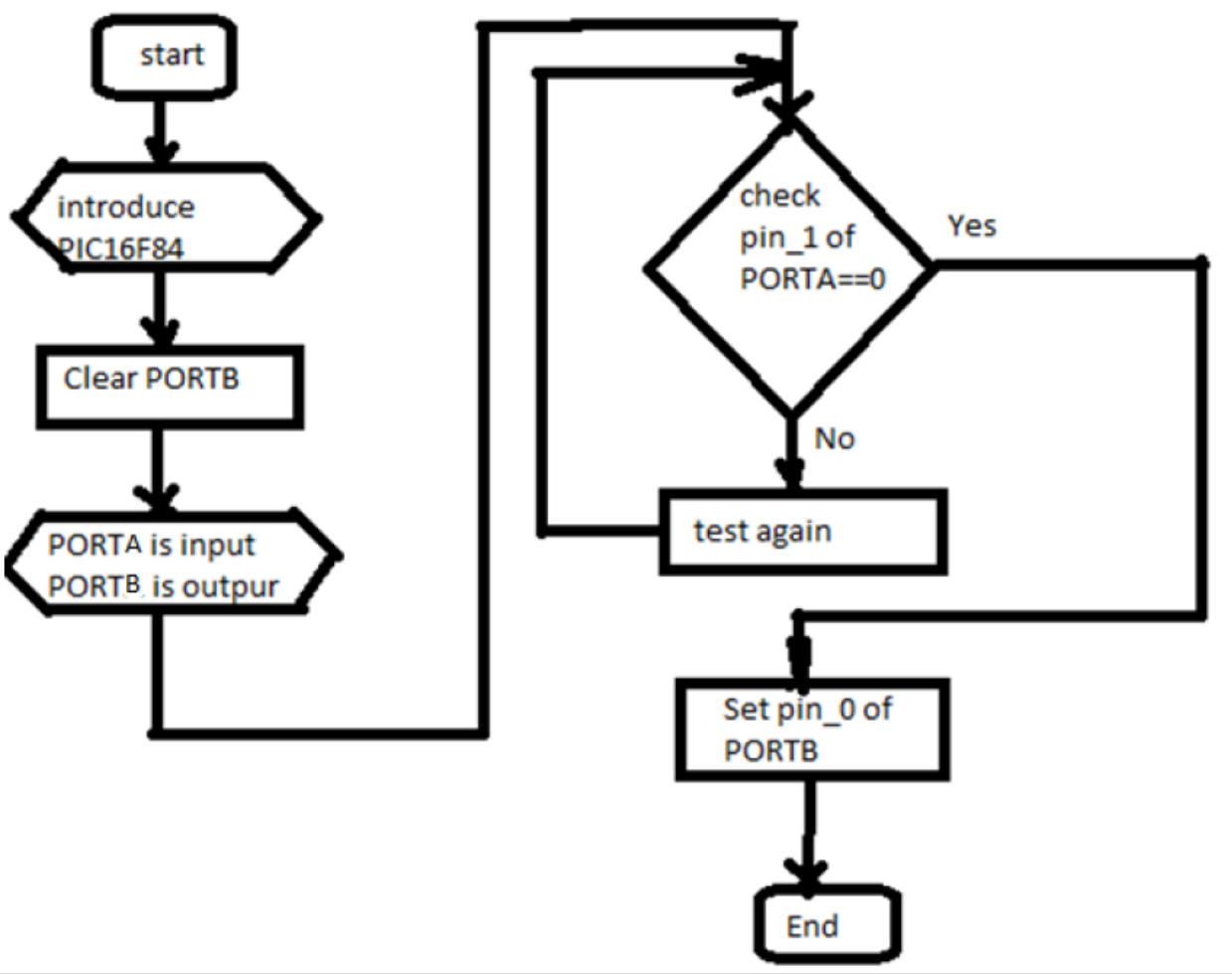
Gate output sinking current from LED

Examples For Bit Test Instructions

- Define bit_1 of PORTA is input and bit_0 of PORTB is output. Write a program that when button of RA1 is pressed led_0 turns on.



Flowchart


start
introduce PIC16F84
Clear PORTB
PORTA is input
PORTB is outpur
check pin_1 of PORTA==0
Yes
No
test again
Set pin_0 of PORTB
End

```
; ===prog_bit_test.asm===
        LIST P=16F84A
PORTA EQU h'05'
PORTB EQU h'06'
STATUS EQU h'03'
TRISA EQU h'85'
TRISB EQU h'86'
        CLRF PORTB; leds of PORTB are off
        BSF STATUS, 5; Change to Bank 1
        CLRF TRISB    ; all pins of PORTB are output
        MOVLW   h'FF';
        MOVWF   TRISA; all pins of PORTA are input
        BCF STATUS, 5; Change to Bank 0
TEST_PORTA
        BTFSC PORTA,1; test bit_1 of PORTA (If we press the button of PORTA ,PORTA_ bit_1=0)
        GOTO TEST_PORTA; if PORTA_bit_1=1 , test again
        BSF PORTB, 0; if PORTA_bit_1=0 , Set to 1 PORTB_bit_0
LOOP
      GOTO LOOP
      END
```