Program IO and Interrupt

Program-controlled IO (polling)
Interrupt Driven I/O
Handling multiple devices
PIC18 Interrupt example

1

1

Program IO and Interrupt

Program-controlled IO (polling)
Interrupt Driven I/O
Handling multiple devices
PIC18 Interrupt example

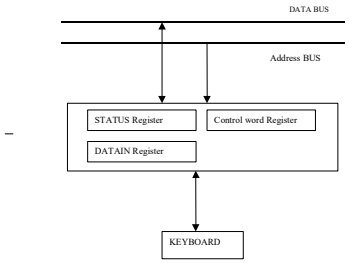
2

2

Program-controlled IO

Consider a computer system

If a key is pressed,
•Datain Reg contains the new key.
•Status Register is set.



3

3

Program-controlled IO

Since the speed of the CPU and the speed of the human operator are different. However, we must ensure that an instruction to read character from the keyboard is executed only when a character is available in the input buffer of the keyboard interface. We must also ensure that an input character is read only once.

Consider the following Pseudocode:

```
WAIT_key    Read STATUS
            Test STATUS
            if no new character, Jump WAIT_key
            Read DATAIN
```

4

4

Program-controlled IO (polling)

```
WAIT_key:    Read STATUS
             Test STATUS
             if no new character, Jump WAIT_key
             Read DATAIN
```

In program controlled IO, the CPU repeatedly checks a status flag to achieve the required synchronization between the CPU and the input device. The programs enters a wait loop in which it repeatedly tests the device status. During this period, the CPU is not performing any useful computation.

5

5

Program IO and Interrupt

Program-controlled IO (polling)

Interrupt Driven I/O

Handling multiple devices

Direct Memory Access

PIC18 Interrupt example

6

6

Interrupt Driven I/O

In many situations, other tasks can be performed while the CPU waits for an IO device to become ready. To do this, we can arrange for the IO device to alert the CPU when it is ready. It does so by sending a hardware signal called an interrupt.

The CPU allows normal program execution to be interrupted by some external signals from I/O devices.

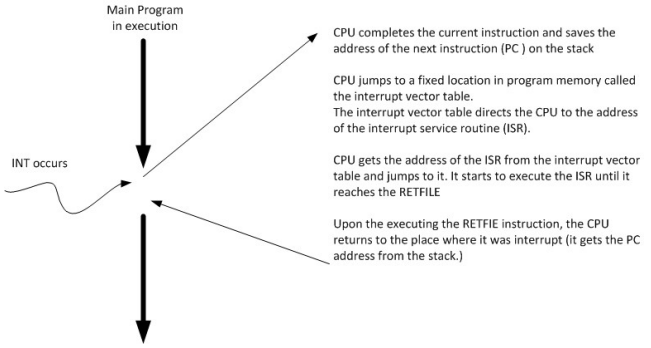
When interrupted, it stops executing its current program and enter an *interrupt sequence*. The status of the current program is saved before entering the *interrupt service routine (ISR)* that services the interrupt.

After servicing the interrupt, the status before the interrupt is restored, execution is then returned to the interrupted program.

7

7

Interrupt Driven I/O

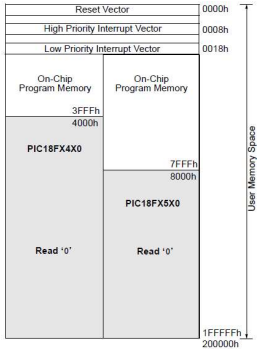


8

8

Interrupt Driven I/O

- When an interrupt is invoked the uC runs the Interrupt Service Routine(ISR)
- Interrupt vector table holds the address of ISRs
  - Power-on Reset 0000h
  - High priority interrupt 0008h
  - Low priority interrupt 0018h



9

9

Steps in executing an interrupt

- Upon activation of interrupt the microcontroller
  - Finishes executing the current instruction
  - Pushes the PC of next instruction in the stack
  - Jumps to the interrupt vector table to get the address of ISR and jumps to it
  - Begin executing the ISR instructions to the last instruction of ISR (RETFIE)
  - Executes RETFIE
    - Pops the PC from the stack
    - Starts to execute from the address of that PC

10

10

Program organization in MPLAB

```
        ORG 0x0000
        goto    Main        ;go to start of main code

;*****************************************************************
;High priority interrupt vector
        ORG 0x0008
        bra HighInt     ;go to high priority interrupt routine

;*****************************************************************
;Low priority interrupt vector and routine

        ORG 0x0018
;   *** low priority interrupt code goes here ***
        retfie

;*****************************************************************
;High priority interrupt routine
HighInt:
;   *** high priority interrupt code goes here ***
        retfie  FAST
;*****************************************************************
;Start of main program
; The main program code is placed here.
Main:
;   *** main code goes here ***

        END
```

11

11

Sources of interrupts in PIC18

- External hardware interrupts
  - Pins RB0(INT0),RB1(INT1),RB2(INT2)
- PORTB change
- Timers
  - Timer0 , Timer1 ,Timer2
- ADC (analog to digital converter)
- CCP (compare capture pulse width modulation, PWM)
- ... etc

12

12

Enabling and disabling an interrupt

- When the PIC is powered on (or resets)
  - All interrupts are masked (disabled)
  - The default ISR address is 0008h
    - No interrupt priorities for interrupts

13

13

Program IO and Interrupt

Program-controlled IO (polling)
Interrupt Driven I/O
Handling multiple devices
Direct Memory Access
PIC18 Interrupt example

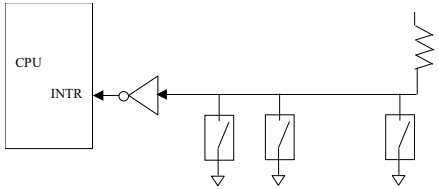
14

14

Handling multiple devices

If there are multiple interrupt lines and each of lines is corresponding to an interrupt routine. So, if INTR i is activated, the CPU will jump to the i-INTR routine. The above scheme is called as multiple interrupt line.

In many cases, several devices capable of initiating interrupts are connected to the CPU, or several interrupts use the same interrupt service routine.



15

15

Handling multiple devices

Questions:
How can the CPU recognize the device requesting an interrupt?
How can the processor obtain the starting address of the appropriate routine?
Should a device be allowed to interrupt the CPU while another interrupt is being serviced?
How should two or more simultaneous interrupt request be handled?

16

16

Device identification by program polling

When a request is received over a common INTR line, additional information is needed to identify the particular device that activated the line. The information is provided in the status in the status registers of the devices.

The interrupt-service routine begins by polling the devices (check interrupt request bit of the status registers of the devices) in some orders. The first device encountered with its IRQ bit set is the device that is serviced, and an appropriate subroutine is called to provide the requested service.

Device identification by program polling

17

17

Device identification by program polling

Polling interrupt service routine

if IRQ of device A set, jump to serve device A routine.
if IRQ of device B set, jump to serve device B routine.
::::::::::::::::::::::
::::::::::::::::::::::

Advantage : simple and easy to implement.
Disadvantage: time spent interrogating the IRQ bits of all the devices.

18

18

Vectored Interrupts

A device requesting an interrupt can identify its self by sending a special code to the CPU over the data bus.
The CPU gets the address of the corresponding interrupt service from a vector table based on the code.
The CPU will jump to the corresponding interrupt service.

NOT ALL CPUs have this mechanism.

8051 does not have
PIC18 does not have
68000 family do have

19

19

Priority of Interruptss

IO devices, or interrupts are organized in a priority structure. An interrupt request with a high priority should be accepted while the CPU is servicing another request from a low-priority device.
A real time clock interrupt should be high priority than a read key interrupt.
Priority of P18 interrupts

INTERRUPT	PRIORITY
High Priority Interrupt	High
Low Priority Interrupt	Low

In PIC18, we can configure some bits in control word registers to set the priority of an interrupt.

20

20

Priority of Interruptss

Each INTR line is assigned a different priority level.
Interrupt requests received over these lines are sent to a priority arbitration circuit in the CPU (or a external circuit).

A request is accepted only if

1. It has a higher priority than other interrupts that are being served or there is not interrupt that are being served.
2. The corresponding interrupt enable pin is enable

21

21

Comparison

QUESTION: Explain the following table

	program IO	**INT**
initialize IO	CPU	IO device
overhead	small	large
speed	slow	slow
Response speed	slow	fast

22

22

Program IO and Interrupt

Program-controlled IO (polling)
Interrupt Driven I/O
Handling multiple devices
Direct Memory Access
PIC18 Interrupt example

23

23

Sources of interrupts in PIC18

- External hardware interrupts
  - Pins RB0(INT0),RB1(INT1),RB2(INT2)
- PORTB change
- Timers
  - Timer0 , Timer1 ,Timer2
- ADC (analog to digital converter)
- CCP (compare capture pulse width modulation, PWM)
- ... etc

24

24

Additional Notes

When INT occurs
•Finishes executing the current instruction
•Pushes the PC of next instruction in the stack
•Jumps to the interrupt vector table to get the address of ISR and jumps to it.
•Disable GIE (automatically)
•Begin executing the ISR instructions to the last instruction of ISR (RETFIE)
•Executes RETFIE
- Pops the PC from the stack
- Set GEIE
- Starts to execute from the address of that PC

25

25

Enabling and disabling an interrupt

- When the PIC is powered on (or resets)
  - All interrupts are masked (disabled)
  - The default ISR address is 0008h
    - No interrupt priorities for interrupts

26

26

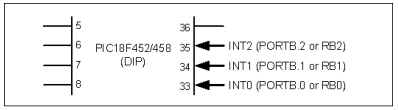
Enabling and disabling an interrupt

- In general, interrupt sources have three bits to control their operation. They are:
- **Flag bit**
  - to indicate that an interrupt event occurred
- **Enable bit**
  - that allows program execution to branch to the interrupt vector address when the flag bit is set
- **Priority bit**
  - to select high priority or low priority

27

27

External interrupts INT0, INT1, INT2



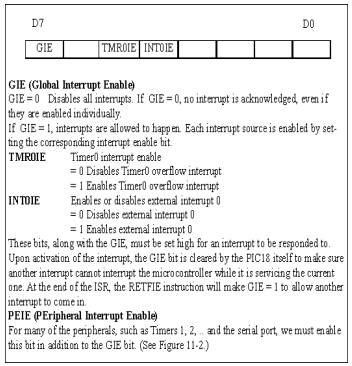
INT (pin)	**Flag bit**	**Register**	**Enable bit**	**Register**	**Pos or Neg edge**	**Register**
INT0 (RB0)	INT0IF	INTCON	INT0IE	INTCON	INTEDGE0	INTCON2
INT1 (RB1)	INT1IF	INTCON3	INT1IE	INTCON3	INTEDGE1	INTCON2
INT2 (RB2)	INT2IF	INTCON3	INT2IE	INTCON3	INTEDGE2	INTCON2
	Set to 1 by the interrupt event.		0 enable 1 disable		0 falling edge 1 rising edge (default power-on	

28

28

Steps in enabling an interrupt

- Set the GIE bit from INTCON REG
- Set the IE bit for that interrupt
- If the interrupt is one of the peripheral (timers 1,2 , serial,etc) set PEIE bit from INTCON reg



GIE (Global Interrupt Enable)
GIE = 0 Disables all interrupts. If GIE = 0, no interrupt is acknowledged, even if they are enabled individually.
If GIE = 1, interrupts are allowed to happen. Each interrupt source is enabled by setting the corresponding interrupt enable bit.

TMR0IE Timer0 interrupt enable
= 0 Disables Timer0 overflow interrupt
= 1 Enables Timer0 overflow interrupt

INT0IE Enables or disables external interrupt 0
= 0 Disables external interrupt 0
= 1 Enables external interrupt 0

These bits, along with the GIE, must be set high for an interrupt to be responded to. Upon activation of the interrupt, the GIE bit is cleared by the PIC18 itself to make sure another interrupt cannot interrupt the micro controller while it is servicing the current one. At the end of the ISR, the RETFIE instruction will make GIE = 1 to allow another interrupt to come in.

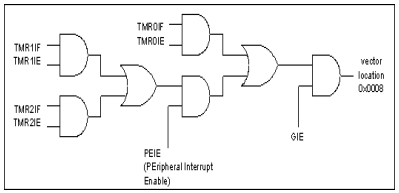
PEIE (PEripheral Interrupt Enable)
For many of the peripherals, such as Timers 1, 2, .. and the serial port, we must enable this bit in addition to the GIE bit. (See Figure 11-2.)

29

29

Example

a)
BSF INTCON,TMR0IE
BSF INTCON,INT0IE
BSF INTCON,GIE
Or
MOVLW B'10110000'
MOVWF INTCON
b)
BCF INTCON,TMR0IE
c) BCF INTCON,GIE

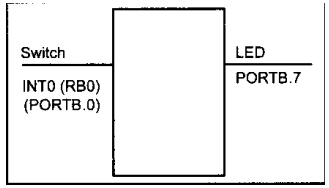


30

30

Example

- Connect a switch to INT0 and an LED to pin RB7
- Every time INT0 is activated, it toggles the LED.
- At the same time, data is being transferred from PORTC to PORTD.



31

31

Program

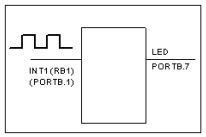
```
        ORG 0000H
        GOTO MAIN

        ORG 0008H
        BTFSS INTCON,INT0IF
        RETFIE
        GOTO INT0_ISR

        ORG 00100H
MAIN
        BCF TRISB,7
        BSF TRISB, 0
        CLRF TRISD
        SETF TRISC
        BCF INTCON,INT0IF
        BSF INTCON,INT0IE
        BSF INTCON,GIE
OVER    MOVFF  PORTC, PORTD
        BRA OVER
INT0_ISR
        ORG 200H
        BTG PORTB,7
        BCF INTCON,INT0IF
        RETFIE
        END
```

32

32

Negative Edge-triggered interrupts

```
        ORG 0000H
        GOTO MAIN

        ORG 0008H
        BTFSS INTCON3,INT1IF
        RETFIE
        GOTO  INT1_ISR

        ORG 00100H
MAIN
        BCF TRISB,7
        BSF TRISB, 1
        BSF INTCON3,INT1IE
        BCF INTCON3,INT1IF
        BCF INTCON2,INTEDGE1
        BSF INTCON,GIE
OVER    MOVFF  PORTC, PORTD
        BRA OVER
INT1_ISR
        ORG 200H
        BTG PORTB,7
        BCF INTCON3,INT1IF
        RETFIE
        END
```



33

33

Sampling the Edge triggered interrupt

- The external source must be held high for at least two instruction cycles

- For XTAL 10Mhz
- Instruction cycle time is 400ns,0.4us
- So minimum pulse duration to detect edge triggered interrupts = 2 instruction cycle =
- 0.8us

34

34

Example

- Connect a switch to INT0 and another switch to INT1.
- Every time INT0 is activated, it increments the content in location 0.
- Every time INT1 is activated, it decrements the content in location 0.
- At the same time, data is being transferred from PORTC to PORTD.

35

35

Example

```
        ORG 0000H
        GOTO MAIN

        ORG 0008H
        BTFSC  INTCON,INT0IF
        RCALL  INT0_ISR
        BTFSC  INTCON3,INT1IF
        RCALL  INT1_ISR
        RETFIE

        ORG 00100H
MAIN
        BSF TRISB, 0
        BSF TRISB, 1
        CLRF TRISD
        SETF TRISC

        BCF INTCON,INT0IF
        BSF INTCON,INT0IE
        BCF INTCON2,INTEDGE0

        BSF INTCON3,INT1IE
        BCF INTCON3,INT1IF
        BCF INTCON2,INTEDGE1

        BSF INTCON,GIE
OVER    MOVFF  PORTC, PORTD
        BRA OVER
INT0_ISR
        INCF 0, F
        BCF INTCON,INT0IF
        RETURN

INT1_ISR
        DECF 0, F
        BCF INTCON3,INT1IF
        RETURN

        END
```

36

36

```
        ORG 0000H
        GOTO MAIN
        ORG 0008H
        GOTO check

        ORG 00100H
MAIN
        BSF TRISB, 0
        BSF TRISB, 1
        CLRF TRISD
        SETF TRISC

        BCF INTCON,INT0IF
        BSF INTCON,INT0IE
        BCF INTCON2,INTEDGE0

        BSF INTCON3,INT1IE
        BCF INTCON3,INT1IF
        BCF INTCON2,INTEDGE1

        BSF INTCON,GIE
OVER    MOVFF  PORTC, PORTD
        BRA OVER
```

```
check
        BTFSC  INTCON,INT0IF
        RCALL  INT0_ISR
        BTFSC  INTCON3,INT1IF
        RCALL  INT1_ISR
        RETFIE

INT0_ISR
        INCF 0, F
        BCF INTCON,INT0IF
        RETURN

INT1_ISR
        DECF 0, F
        BCF INTCON3,INT1IF
        RETURN

        END
```

37

37

Example

- Note that an ISR may change WREG, BSR, and Status register.
- If this happens, when the CPU returns from INT, the original information may lost.
- How to solve this problem?

38

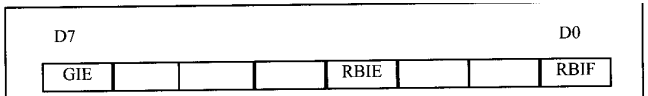
38

PORTB change interrupt

39

39

PORTB change interrupt



GIE (Global Interrupt Enable)
GIE = 0 Disables all interrupts. If GIE = 0, no interrupt is acknowledged, even if they are enabled individually.
If GIE = 1, interrupts are allowed to happen. Each interrupt source is enabled by setting the corresponding interrupt enable bit.

RBIE PORTB-Change Interrupt Enable
= 0 Disables PORTB-Change interrupt
= 1 Enables PORTB-Change interrupt

RBIF PORTB-Change Interrupt Flag.
= 0 None of the RB4–RB7 pins have changed state
= 1 At least one of the RB4–RB7 pins have changed state

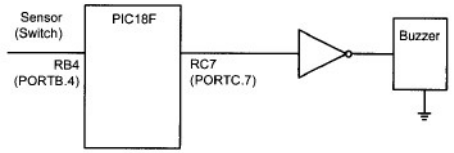
The RBIE bit, along with the GIE, must be set high for any changes on the pins RB4–RB7 to cause an interrupt. The RB4–RB7 pins must also have been configured as input pins for this interrupt to work. In order to clear the RBIF flag we must read the pins of RB4–RB7 **and** use the instruction "BCF INTCON,RBIF".

0

40

PORTB change interrupt

- A door sensor to pin RB4 and a buzzer to pinRC7.
- Every time the door is open, it sounds the buzzer by a sending a square wave for a while.



41

41

Program

```
        MYREG   EQU     0x20
DELRG   EQU 0x80
        ORG     0000H
        GOTO    MAIN

        ORG     0008H
        BTFSS INTCON,RBIF
        RETFIE
        GOTO PB_ISR

        ORG     00100H
MAIN    BCF  TRISC,7
        BSF     TRISB,4
        BSF INTCON,RBIE
        BSF INTCON,GIE
OVER    BRA OVER
```

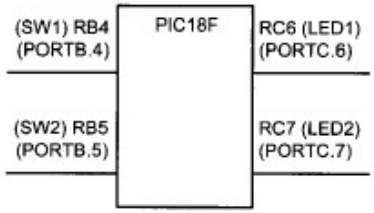
```
PB_ISR
        ORG 200H
        MOVF  PORTB,W
        MOVLW D'250'

        MOVWF MYREG
BUZZ    BTG PORTC,7
        MOVLW D'255'

        MOVWF DELRG
DELAY   DECF DELRG,F
        BNZ DELAY

        DECF MYREG,F
        BNZ BUZZ
        BCF INTCON,RBIF
        RETFIE
        END
```

42

42

Another Example

- SW1 and SW2 to RB4 and R B5 respectively.
- Activation of SW1➔ state change in LED



43

43

Program

```
        ORG     0000H
        GOTO    MAIN
        ORG     0008H
        BTFSS INTCON,RBIF
        RETFIE
        GOTO  PB_ISR
        ORG     0100H
MAIN    BCF TRISC,4
        BCF TRISC,5
        BSF TRISB,4
        BSF TRISB,5
        BSF INTCON,RBIE
        BSF INTCON,GIE
OVER    BRA OVER
```

```
PB_ISR
        ORG 200H
        MOVFF PORTB,W
        ANDLW 0x30
        MOVFF W,PORTC
        BCF INTCON,RBIF
        RETFIE
        END
```

44

44

Outcomes

- Able to describe the concept of Program-controlled IO.
- Able to write a program to control a simple task based on Program-controlled IO.
- Able to describe the concept of Interrupt Driven I/O
- Able to describe the rationale of the steps in nterrupt Driven I/O
- Able to write a program to control a simple task based of Interrupt Driven I/O

45

45