DIGITAL CONTROLLER

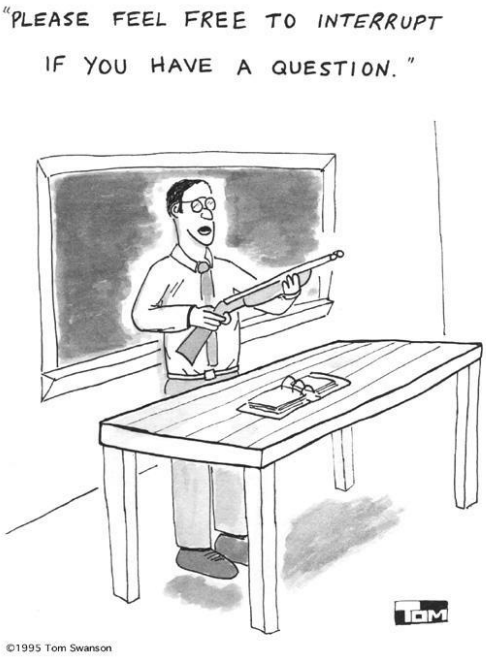
3rd STAGE

CHAPTER : 5
INTERRUPTS

Lecturer: Ali Salman Kurji

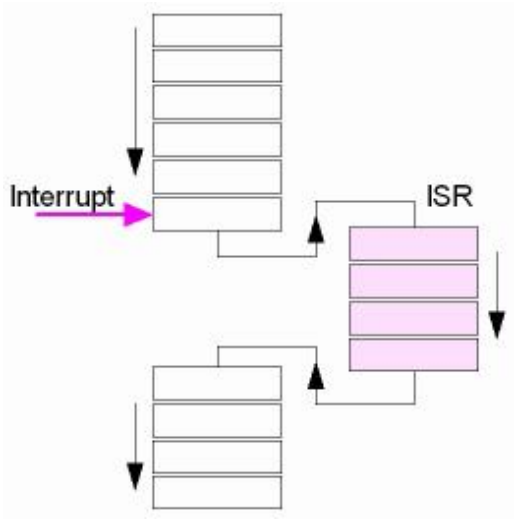
Chapter 5
Interrupts

The subject of interrupts is probably going to be the longest and most difficult to go through. There is no easy way of explaining interrupts, but hopefully by the end of this section you will be able to implement interrupts into your own programs. We have split the section into two parts. This is to help break the subject up, and to give you, a break. So what is an interrupt? Well, as the name suggests, an interrupt is a process or a signal that stops a micro-processor/microcontroller from what it is doing so that some-thing else can happen. Let me give you an everyday exam-ple. Suppose you are sitting at home, chatting to some-one. Suddenly the telephone rings. You stop chatting, and pick up the telephone to speak to the caller. When you have finished your telephone conversation, you go back to chatting to the person before the telephone rang. You can think of the main routine as you chatting to someone, the telephone ringing causes you to interrupt your chatting, and the interrupt routine is the process of talking on the telephone. When the telephone conversation has ended, you then go back to your main routine of chatting. This example is exactly how an interrupt causes a processor to act. The main program is running, performing some function in a circuit, but when an interrupt occurs the main program halts while another routine is carried out. When this routine finishes, the processor goes back to the main rou-tine again.



The PIC has 4 sources of interrupt. They can be split into two groups. Two are sources of interrupts that can be applied externally to the PIC, while the other two are in-ternal processes. We are going to explain the two external ones here. The other two will be explained in timers and storing data.

If you look at the pin-out of the PIC, you will see that pin 6 shows it is RB0/INT. Now, RB0 is obviously Port B bit 0. The INT symbolizes that it can also be configures as an external in-terrupt pin. Also, Port B bits 4 to 7 can also be used for inter-rupts. Before we can use the INT or other Port B pins, we need to do two things. First we need to tell the PIC that we are going to use interrupts. Secondly, we need to specify which port B pin we will be using as an interrupt and not as an I/O pin.



Inside the PIC there is a register called INTCON, and is at ad-dress 0Bh. Within this register there are 8 bits that can be en-abled or disabled. Bit 7 of INTCON is called GIE. This is the Global Interrupt Enable. Setting this to 1 tells the PIC that we are going to use an interrupt. Bit 4 of INTCON is called INTE, which means INTerrupt Enable. Setting this bit to 1 tells the PIC that RB0 will be an interrupt pin. Setting bit 3, called RBIE, tells the PIc that we will be using Port B bits 4 to 7. Now the PIC knows when this pin goes high or low, it will need to stop what it's doing and get on with an interrupt routine. Now, we need to tell the PIC whether the interrupt is going to be on the rising edge (0V to +5V) or the falling edge (+5V to 0V) transi-tion of the signal. In other words, do we want the PIC to interrupt when the signal goes from low to high, or

from high to low. By default, this is set up to be on the rising edge. The edge 'triggering' is set up in another register called the OPTION register, at address 81h. The bit we are interested in is bit 6, which is called INTEDG. Setting this to 1 will cause the PIC to interrupt on the rising edge (default state) and setting it to 0 will cause the PIC to interrupt on the falling edge. If you want the PIC to trigger on the rising edge, then you don't need to do anything to this bit.

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Features
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Features
OPTION	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Ok, so now we have told the PIC which pin is going to be the interrupt, and on which edge to trigger, what happens in the program and the PIC when the interrupt occurs? Two things happen. First, a 'flag' is set. This tells the internal processor of the PIC that an interrupt has occurred. Secondly, the program counter which points to a particular address within the PIC. Let's quickly look at each of these separately.

Interrupt Flag

In our INTCON register, bit 1 is the interrupt flag, called INTF. Now, when any interrupt occurs, this flag will be set to 1. While there isn't an interrupt, the flag is set to 0. And that is all it does. Now you are probably thinking 'what is the point?' Well, while this flag is set to 1, the PIC cannot, and will not, respond to any other interrupt. So, let's say that we cause an interrupt. The flag will be set to 1, and the PIC will go to our routine for processing the interrupt. If this flag wasn't set to 1, and the PIC was allowed to keep responding to the interrupt, then continually pulsing the pin will keep the PIC going back to the start of our interrupt routine, and never finishing it. Going back to my example of the telephone, it's like picking up the telephone, and just as soon as you start to speak it starts ringing again because someone else want to talk to you. It is far better to finish one conversation, then pick up the phone again to talk to the second per-son.

There is a slight drawback to this flag. Although the PIC automatically sets this flag to 1, it doesn't set it back to 0! That task has to be done by the programmer – i.e. you. This is easily done, as We are sure you can guess, and has to be done after the PIC has executed the interrupt routine.

Memory Location | Interrupt Routine

When you first power up the PIC, or if there is a reset, the Program Counter points to address 0000h, which is right at the start of the program memory. However, when there is an interrupt, the Program Counter will point to address 0004h. So, when we are writing our program that is going to have interrupts, we first of all have to tell the PIC to jump over address 0004h, and keep the interrupt routine which starts at address 0004h separate from the rest of the program. This is very easy to do.

First, we start our program with a command called ORG. This command means Origin, or start. We fol-low it with an address. Because the PIC will start at address 0000h, we type ORG 0000h. Next we need to skip over address 0004h. We do this by placing a GOTO instruction, followed by a label which points to our main program. We then follow this GOTO command with another ORG, this time with the address 0004h. It is after this command that we enter our interrupt routine. Now, we could either type in our inter-rupt routine directly following the second ORG command, or we can place a GOTO statement which points to the interrupt routine. It really is a matter of choice on your part. To tell the PIC that it has come to the end of the interrupt routine we need to place the command RTFIE at the end of the routine. This command means return from the interrupt routine. When the PIC see this, the Program Counter points to the last location the PIC was at before the interrupt happened.

This is how we set an interrupt system in Assembly. However in Proton Basic it is simply a procedure. Since interrupt routines have to be fast and release the processor from interrupt as soon as possible, many programmers prefer to manage the interrupts in assembly.

There are two things you should be aware of when using interrupts. The first is that if you are using the same register in your main program and the interrupt routine, bear in mind that the contents of the register will probably change when the interrupt occurs. For example, let's you are using the w register to send data to Port A in the main program, and you are also using the w register in the interrupt routine to move data from one location to another. If you are not careful, the w register will contain the last value it had when it was in the interrupt routine, and when you come back from the interrupt this data will be sent to Port A instead of the value you had before the interrupt happened. The way round this is to temporarily store the contents of the w register before you use it again in the interrupt routine. The second is that there is a delay between when one interrupt occurs and when the next one can occur. As you know, the PIC has an external clock, which can either be a crystal or it can be a resistor-capacitor combination. Whatever the frequency of this clock, the PIC divides it by 4 and then uses this for it's internal timing. For example if you have a 4MHz crystal connected to your PIC, then the PIC will carry out the instructions at 1MHz. This internal timing is called an Instruction Cycle. Now, the data sheet states (admittedly in very small print) that you must allow 3 to 4 instruction cycles between interrupts. My advice is to allow 4 cycles. The reason for the delay is the PIC needs time to jump to the interrupt address, set the flag, and come back out of the interrupt routine. So, bear this in mind if you are using another circuit to trigger an interrupt for the PIC.

Now, a point to remember is that if you use bits 4 to 7 of Port B as an interrupt. You cannot select individual pins on Port B to serve as an interrupt. So, if you enable these pins, then they are all available. So, for example, you can't just have bits 4 and 5 – bits 6 and 7 will be enabled as well. So what is the point of hav-ing four bits to act as an interrupt? Well, you could have a circuit connected to the PIC, and if any one of four lines go high, then this could be a condition that you need the PIC to act on quickly. One example of this would be a house alarm, where four sensors are connected to Port B bits 4 to 7. Any sensor can trigger the PIC to sound an alarm, and the alarm sounding routine is the interrupt routine. This saves examining the ports all the time and allows the PIC to get on with other things.

We covered quite a bit of ground , and so we think it is time that we wrote our first program. The program we are going to write will count the number of times we turn a switch on, and then display the number. The program will count from 0 to 9, displayed on 4 LEDs in binary form, and the input or interrupt will be on RB0. in PIC Lab-II SW5 is connected to RB0. although it is active low, we are going to use interrupt on rising edge, thus the interrupt will take place when key is released. The processor will be held in an endless loop, from which it can not come out. Under normal circumstances if the processor is busy in some loop, it can not scan the input buttons, however using interrupt it will attend the button press.

```
Device=18F452
XTAL=20
ALL_DIGITAL true
Symbol GIE INTCON.7
Symbol INT0IE INTCON.4
Symbol INT0F INTCON.1
Dim x As Byte
Output PORTC
on_interrupt GoTo Jingle
GoTo Start
Jingle:
x=x+1

PORTC=x
INT0F=0
Context Restore

Start:
x=0
PORTC=0
INT0IE = 1
GIE=1
aa:
GoTo aa
```

Notice the interrupt routine starting at label Jingle, we have defined the symbols, for easy understanding for enabling general interrupt system GIE, enabling RB0 interrupt also called INT0 and IN0 flag INT0F then we have defined variables and direction of PORTC. Next we have issued the on_interrupt command that tells the compiler where to branch whenever an interrupt takes place. Since the code at jingle is to be executed on interrupt we jump over it to start label. Here we have initialized our variables and ports, and enable INT0, and the enable GIE. Then we put the processor into an endless loop. Without interrupt system, this program should not respond to key press. Bust since interrupt is enabled, when SW5 (RB0) is pressed and released (trigger on rising edge) the program will jump to interrupt, here we do something, and on finishing, reset the interrupt flag to 0 and context restore command restores the stack, and other registers used for jump, back to the state in which they were before interrupt.

Now pressing the SW5, will change the LEDs on PORTC.Interrupts can also take place on internal events, like a timer / counter reaching its maximum value, data received on USART port, ADC conversion completed and others. These interrupts are called hardware in-terrupts as they occur on peripheral devices present within the PIC hardware. We shall talk about timers and timer interrupts in chapter