Investigating IO Interrupts

Introduction

At the end of this lab you should be able to:

- Describe what interrupt vectors are and explain how they are used
- Describe two main methods of IO interrupt handling
- Explain the difference between the two main methods of IO interrupt handling
- Compare the merits of the two main methods of IO interrupt handling

Basic Theory

Computer systems use the interrupt mechanism as a means of responding to external events such as input and output operations. The CPU is momentarily interrupted just before executing the next instruction and is forced to execute the instructions of an interrupt handler. Once the interrupt handling is completed the CPU is returned back to executing the instruction it was about to execute before it was interrupted. The stack is used to store the CPU state such as the contents of registers and the return address when interrupted. These are then restored once the interrupt handler is exited.

Lab Exercises - Investigate and Explore

The lab investigations are a series of exercises that are designed to demonstrate the various aspects of IO interrupt handling.

Exercise 1 – Describe what interrupt vectors are and explain how they are used

In the compiler window, check only the boxes **Generate code**, **Enable optimizer** and **Redundant Code**. Enter the following source code and compile it:

```
program Vectors
   sub IntVect1 intr 1
        writeln("This is intr 1")
   end sub

sub IntVect2 intr 2
        writeln("This is intr 2")
   end sub

sub IntVect5 intr 5
        writeln("This is intr 5")
   end sub

while true
   wend
end
```

In the compiled code window locate the subroutines *IntVect1*, *IntVect2* and *IntVect5*. Make a note of the starting addresses of these subroutines below:

Subroutine	Starting address
IntVect1	
IntVect2	
IntVect5	

Next, do the following:

- 1. Load the code generated in CPU memory.
- 2. Click on the INTERRUPTS... button to view the INTERRUPT VECTORS window.
- 3. Make a note of the numbers displayed in text boxes next to INT1, INT2 and INT5.

<u>Note</u>: The **INTERRUPT VECTORS** window in the simulator represents that part of the CPU hardware that stores the various interrupt routine addresses.

Interrupt	
INT1	
INT2	
INT5	

Compare the tw the space below	ve and enter	a brief comme	ent on your ob	servation in	

Now, follow the instructions below:

- 1. Click on the **INPUT OUTPUT...** button to view the console window.
- 2. Select **Stay on top** boxes both in the console and the interrupt vectors windows.
- 3. Reset the **Vectors** program and run it at the fastest speed.
- 4. While the program is running, click **TRIGGER** buttons in the interrupts window against **INT1**, **INT2** and **INT5** one after the other.
- 5. Observe the messages displayed on the console. Comment on your observations:

<u>Tip</u>: If you run the program at a slow pace (speed slider down), you should be able to see the effects of clicking on the **TRIGGER** buttons.

Comment on your observations in the space below:		
Exercise 2 – Describe two main methods of interrupt handling		
Enter the following source code in a new source editor and compile it.		
program PolledInt		
var v integer		
v = 0		
<pre>writeln("Program Starting") while true</pre>		
read(nowait, v)		
for i = 1 to 100		
if $v > 0$ then		
break *		
end if		
write(".")		
next		
wend writeln("Program Ending")		
end		
Notoga		
Notes: ■ The nowait keyword in the read statement makes sure the program is not		
suspended while waiting for an input.		
■ If there is no input, the value of the variable v will remain unchanged.		
■ The break * statement takes the program out of the outermost loop which in this		
case is the while loop.		
So, now, briefly explain what the above program is doing:		

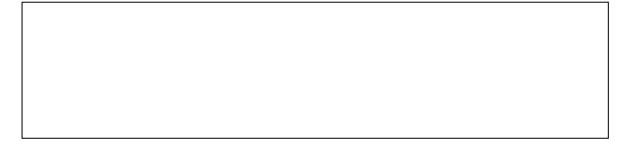
Next, follow the instructions below:

- 1. Load the code generated in CPU memory.
- 2. Set the speed of simulation to maximum.
- 3. Bring the console window up (use the **INPUT OUTPUT...** button).
- 4. Check the **Stay on top** check box on the Console.
- 5. Click in the **INPUT** box on the Console.
- 6. Start the simulation by clicking the CPU Simulator's **RUN** button. As soon as the **Program Starting** message is displayed on the Console, type any single character in the **INPUT** box of the Console. Wait until the program terminates.

Next, enter the following source code in a new source editor and compile it.

```
program VectoredInt
     var v integer
     sub InputInt intr 1
        read(nowait, v)
     end sub
     v = 0
     writeln("Program Starting")
     while true
          for i = 1 to 100
               if v > 0 then
                    break *
               end if
               write(".")
          next
     wend
     writeln("Program Ending")
end
```

Briefly explain what the above program is doing (note where the read statement is in this case)



Load the code generated in CPU memory. Reset and run this code at the fastest speed. As soon as the **Program Starting** message is displayed on the Console, type any single character in the **INPUT** box of the Console. Wait until the program terminates.

Exercise 3 – Explain the difference between polled and vectored interrupts

Based on your observation in the previous exercise, briefly explain the difference in the behaviours of the two programs, $PolledInt$ and $VectoredInt$, with respect to the speed of response to input. Explain why this difference.				
			looking at the table below, which s more efficient (put an X against it):	
	-	efficient one		
	Polled Interrupt			
	Vectored Interrupt			
1. Ba	ased on your observati	ions above, suggest le Polled Interrupt r	main methods of interrupt handling and briefly describe a reason method in preference to the	
	ery briefly describe wh preference to the Poll		the Vectored Interrupt method	