Input-Output and 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:

Results:

Subroutine	**Starting address**
IntVect1	0
IntVect2	20
IntVect5	40

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**.

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

Results:

Interrupt	
INT1	0
INT2	20
INT5	40

Compare the two tables above and enter a brief comment on your observation in the space below:

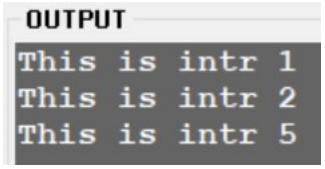
After compare the two table above, the number of starting address of subroutine and the numbers displayed in text boxes next to INT1, INT2 and INT5 is the same number. When click the trigger button, it is going to call the subroutine.

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:

Tip: 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:



The program keeps running and when click trigger buttons, it displays the messages from subroutine.

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

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:

The program is write "." while waiting for an input. If input is greater than 0, the program will stop write "." then write "Program Ending" and finishes.

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
```

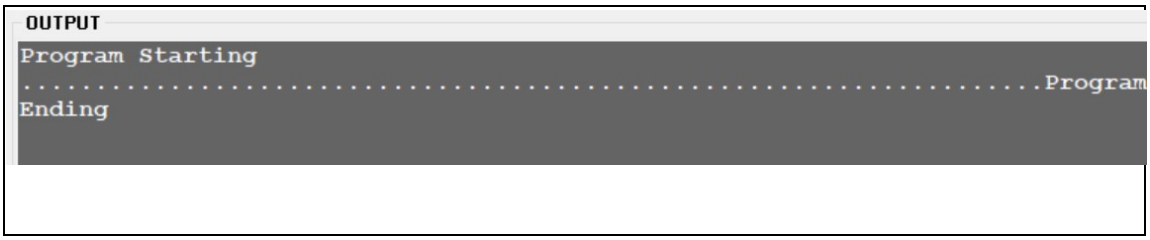
Briefly explain what the above program is doing (note where the read statement is in this case)

The program is write "." while waiting for an input. If input is greater than 0, the program will stop write "." then write "Program Ending" and finishes.

The read statement is in the subroutine InputInt.

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.

Results:



Exercise 3 – Explain the difference between polled and vectored interrupts

Based on your observation in the previous exercise, briefly explain the difference in the behaviors of the two programs, *PolledInt* and *VectoredInt*, with respect to the speed of response to input. Explain why this difference.

PolledInt is slower because it wait for the program to continuously check for new input. While VectoredInt is faster because it immediately interrupt the program when input occurs without a check.

Based on your observations in exercises 2, and looking at the table below, which interrupt handling method listed in the table, is more efficient (put an **X** against it):

Answer:

Interrupt method	**Select the most efficient one**
Polled Interrupt	
Vectored Interrupt	X

Exercise 4 – Compare the merits of the two main methods of interrupt handling

1. Based on your observations above, suggest and briefly describe a reason where you would use the Polled Interrupt method in preference to the Vectored Interrupt method?

When the input frequency is low and the response time isn't critical. So Polled Interrupt is suitable for non-critical, predictable, or infrequent tasks.

2. Very briefly describe where you would use the Vectored Interrupt method in preference to the Polled Interrupt?

When speed, efficiency, and real-time responsiveness are critical. So Vectored Interrupt is suitable for high-priority or real-time tasks requiring immediate attention.

****