National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member: Dr.Nasir Mahmood **Dated: 13/04/2018**

Course/Section: BSCS-6-A **Semester: 4th Semester**

Computer Organization and

Achitecture (EE321)

Lab # IO Interrupts

Name	Reg. no.	Report Marks / 10	Viva Marks / 5	Total/15
Abdul Razzaque Jawad	191408			
Awais Latif Khatti	199189			
Abdullah Rafaqat	146905			

Lab 11: IO Interrupts and Pipelining

Objectives

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

Lab Instructions

- ✓ This lab activity comprises two parts, namely Lab tasks, and Post-Lab Viva session.
- ✓ Each group to upload completed lab report on LMS for grading.
- ✓ The students will start lab task and demonstrate each lab task separately for step-wise evaluation
- ✓ There are related questions in this activity give complete answers. Also provide complete code and results.

Investigating IO Interrupts

Introduction

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	0000
IntVect2	0020
IntVect5	0040

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.

Interrupt	
INT1	0
INT2	20
INT5	40

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

The values in the first table are starting addresses of the user defined vector handler programs as defined by the user, whereas the values in the second table are the starting addresses of the vector handler in the vector table, the addresses in that table serves as an index to start program whenever a particular interrupt is triggered.

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:

During the program execution, if any interrupt is triggered the registers values and the instruction address is pushed to stack and the program is jumped to interrupt service routine program of that interrupt, after execution of interrupt service routine program the pushed values of the program from stack is popped back to registers and the program flow is jumped to previously executing program. If multiple interrupts are triggered simultaneously then it handles interrupts sequentially one after the other.

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:

In this program polled interrupt is used, firstly the integer named 'v' is assigned 0 and the program runs infinitely as the while loop is set to true. In the while loop, it reads user input from keyboard with no special waiting for it. During the input, if user enters any number greater than zero then it breaks the while loop and the program ends. The purpose of this program is end program whenever positive number is entered. Input is checked after every 100 iterations of for loop.

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)

In this program vectored interrupt is used, firstly the integer named 'v' is assigned 0 and the program runs infinitely as the while loop is set to true. In the interrupt service routine of int 1, it reads user input from keyboard with no special waiting for it. When input is given, interrupt 1 is triggered and its interrupt service routine is executed and the input is given then if user enters any number greater than zero then it breaks the while loop and the program ends. The purpose of this program is end program whenever positive number is entered

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.

PolledInt checks for input value from keyboard on every while loop iteration, while as in the VectoredInt program, whenever any input is given the input interrupt int1 is triggered and reads the entered value. PolledInt is slower than the VectoredInt, because input in PolledInt input is read on each iteration after 100 iterations of for loop and takes time for the 'for' loop, whereas in VectoredInt, input is read only when input is entered by the user.

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):

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?

Polled interrupt is used when the piece of code has more importance than the input from the device. When we don't want to disturb particular program execution, and once the program's execution is completed then it checks for the input and so on. For this case, program will run smoothly and checks for input after its completion.

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

When input from device has more importance than piece of code then for faster input Vectored interrupt is used. When we want to handle interrupt as soon as input is given and stops execution of program for a while.