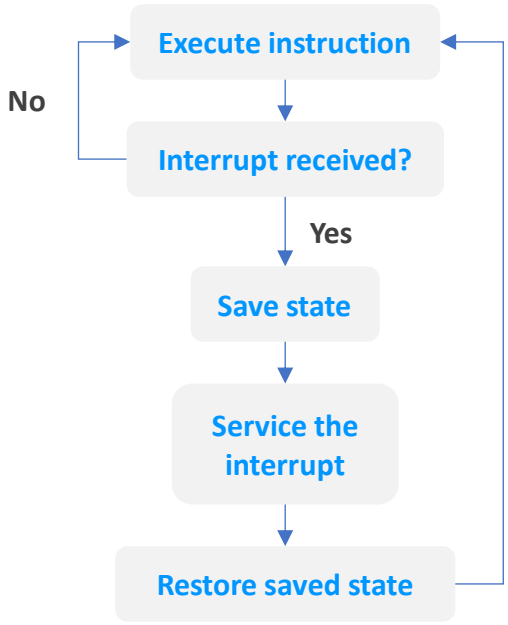
Department of Electronics

ELEC 4601: Laboratory 2
80x86 Interrupts

Version: 09/26/18

Introduction to the Lab (and Interrupts)

Interrupts offer an alternative way of responding to external events in a microprocessor system. The microprocessor can poll for changes in the environment by constantly looking for those changes, or it can wait until a device signals for a change. For real time events (e.g., a keyboard press), an actual hardware signal, referred to as an interrupt request (IRQ), can be generated. IRQs are usually numbered according to their priority, with IRQ0 having the highest priority (the system time of day) and IRQ7 having the lowest (the LPT1 printer port). For simple microprocessor systems, the following flowchart applies.



The software procedure (usually small and fast) that runs in response to an interrupt is referred to as an interrupt service routine (ISR). 80x86 processors operating in real mode (the simple microprocessor mode used in the lab) make use of an interrupt descriptor table (IDT), which consists of 256 ISR addresses, starting at 0000:0000. Most of these entries are for the 255 software interrupts and processor exceptions that the processor supports. Some of the entries in this table have been set aside for hardware interrupts, with IRQ0 to IRQ7 using up the IDT slots at positions 8 to 15. In response to an interrupt, the processor will finish executing the current instruction, immediately jump to the address stored in the IDT and continue executing instructions from there.

Writing software to properly respond to a hardware interrupt is fussy. There is a hardware path from the interrupt signal to the microprocessor that needs to be correctly laid out using specific bits within specific registers in the I/O mapped memory space of the microprocessor. In this lab, the microprocessor is a member of the 80x86 family, **but the process of laying out and enabling the interrupt path is fundamentally the same for all microprocessors.**

There are typically three places where interrupts can be enabled or disabled. (1) The microprocessor itself usually has a global interrupt enable bit(s) as part of a FLAGS register. If there is a subsystem that is responsible for arbitrating incoming interrupts—in this case, (2) the 8259 Programmable Interrupt Controller (PIC)—then it will have registers and bits that need configuring. Finally, (3) the interface subsystem that actually connects to the interrupt source will have some set of registers and bits that need to be configured in order to let the interrupt proceed to the interrupt arbitrator.

Once the hardware path has been properly enabled and interrupts can actually reach the CPU, **the software has to be written to recognize the interrupt, service it, and gracefully restore the system back to its normal (waiting for an interrupt) state.**

The frustrating part of this process arises from the fact that unless everything is working properly, the system will fail. Debugging a hardware interrupt in real time is notoriously difficult, but once you have done it for a system such as this one, you will know what to look for in any microprocessor system. This is something worth knowing.

Part 1: Simple Square Wave Interrupts

Part 1 (of 2) of this lab maps a square wave generator to the IRQ7 hardware interrupt of the 80x86 PC. We have included the full assembly code that continuously services and counts the requested interrupts and prints the total number on an LED printer port. It is your job to run and observe the program, characterize the timing of the ISR, and understand and comment the included code.

Procedure

1. Setup the pin assignments on the logic analyzer, as you did in Lab 1
2. Download the code provided on cuLearn and extract the contents into your **H** drive
3. Login to the DOS machine using the user account given to you in Lab 1
4. Navigate to the downloaded folder on the DOS machine using the **cd** command:
   `cd CODE2`
5. Create an executable file from **LAB2A.PAS:**
   `make LAB2A.PAS`
6. Turn on the function generator at your station and set a square wave for 100 Hz. Note that you must press the **Output** button in order to generate the signal. The physical connection has already been made for you.
7. Run your newly created executable file:
   `LAB2A.EXE`

Some time-saving tips going forward:

- These DOS commands are not case-sensitive
- You can run a .EXE file by just typing the name (e.g., `LAB2A`)
- You can use the up and down arrow keys to cycle through previous commands

8. Observe the LED printer port at the front of the DOS machine. What is happening? You can reduce the frequency of the function generator to get a better look, just make sure to return it to 100 Hz afterwards

9. Take some time to understand the code behind this program. Open **LAB2A.PAS** on the Windows machine (use Notepad++) and read up until the end of the ISR (around line 110)
10. We will now measure the response time of the software. The response time will be defined as the time difference between the rising edge of the **IRQ7** interrupt signal (the square wave) and the final execution of code inside the ISR. The **logic analyzer** will of course be required in order to make this measurement. Performing timing measurements on an interrupt driven system is sometimes tricky, as there may not be a clear piece of code to trigger on. For the ISR in LAB2A.PAS, we inserted a piece of code (a **marker**) that has no purpose in the program other than to help us trigger. Find the **marker** added into the ISR and set up the logic analyzer to trigger on it
11. Run the logic analyzer and verify that you can see the marker. Always make sure to set the **Delay** of the waveform to 0 µs (the time at which the analyzer triggers)
12. There is another instruction inside the ISR that should be clearly visible. **Hint:** it occurs slightly after the marker. Take a screenshot that shows the time from the beginning of the marker to the end of this other instruction
13. Take a screenshot that shows the time between the rising edge of IRQ7 and the beginning of the marker. You will need to zoom out a few times on the logic analyzer
14. Speed up the square wave. At what frequency does the program fail? Think about how a program that uses an ISR to count the rising edges of a square wave would fail if the rising edges came in at a really fast rate. **Hint:** You will need to use the logic analyzer to show the program failing, as the LEDs will be blinking much too fast to observe

Before moving on, make sure you have:

1. A screenshot that shows the time between the two visible instructions of the ISR
2. A screenshot that shows the time between the IRQ7 rising edge and the marker
3. A screenshot that shows the program failing above a certain frequency. Note the frequency

Part 2: Serial Port Interrupts

Part 2 (of 2) of this lab involves generating an interrupt by sending one or more serial characters over a connection between the Windows and DOS computers. For the setup that we are using, the hardware interrupt of interest is IRQ4 (serial port COM1 or COM2 in a PC).

In addition, this lab involves writing software to handle this interrupt and to echo the received characters back to the Windows machine over the serial connection. The software for this part of the lab is almost exactly the same as in Part 1.

Procedure

1. Before you get started with the code, you must setup and test the serial connection. Open **TeraTerm Pro** and select the **serial** connection option upon start up
2. We will use the default serial parameters, so no further configuration is required on the Windows machine. However, we must match the serial parameters on the DOS end with the following command:
   ```
   mode com1: 9600,n,8,1
   ```
3. Enter **Debug** mode on the DOS machine, as you did in Lab 1
4. In TeraTerm, type a character key. This will fill the serial port register (at 0x3F8) with the hexadecimal value of the character (e.g., “a” is 61). Type the following command on the DOS machine to read the data stored in this shared serial port register:
   ```
   i 3F8
   ```
5. Also try writing to the port on the DOS end. A character should show up on TeraTerm. Use the following command:
   ```
   o 3F8 62
   ```
6. Now, open LAB2B.PAS on the Windows machine and begin to finish the skeleton code provided for you. The included comments and the code in LAB2A.PAS should help you out here. **The program must read a character inputted on TeraTerm and echo the same (but uppercased) character back to it. You are also required to update the total amount of received interrupts on LPT1**

Appendix A: x86 Interrupt Vectors

PIC Number	Vector	IRQ Number	Common Uses
	00 – 01	Exception Handlers	–
	02	Non-Maskable IRQs	Parity Errors
	03 – 07	Exception Handlers	–
PIC1	08	Hardware IRQ0	System Timer
0x20	09	Hardware IRQ1	Keyboard
	0A	Hardware IRQ2	Redirected
	0B	Hardware IRQ3	Serial COM2/COM4
	0C	**Hardware IRQ4**	**Serial COM1/COM3**
	0D	Hardware IRQ5	Reserved
	0E	Hardware IRQ6	Floppy Disk Controller
	0F	**Hardware IRQ7**	**Parallel Comms.**
	10 – 6F	Software Interrupts	–
PIC2	70	Hardware IRQ8	Real Time Clock
0xA0	71	Hardware IRQ9	Redirected IRQ2
	72	Hardware IRQ10	Reserved
	73	Hardware IRQ11	Reserved
	74	Hardware IRQ12	PS/2 Mouse
	75	Hardware IRQ13	Math’s Co-Processor
	76	Hardware IRQ14	Hard Disk Drive
	77	Hardware IRQ15	Reserved
	78 – FF	Software Interrupts	–

Appendix B: Useful Register Information

<table>
<tr><th>Device</th><th>Base Address</th><th>Register Offset</th><th>Direction</th><th>Functions</th></tr>
<tr><td rowspan="3">PIC1</td><td rowspan="3">0x20</td><td>0</td><td>Write</td><td>Command</td></tr>
<tr><td rowspan="2">1</td><td rowspan="2">Read/Write</td><td>Interrupt Mask Register (IMR)
For IRQn (n = 0, 1, ..., 7)
0 = Enabled, 1 = Disabled</td></tr>
<tr><td>7 | 6 | 5 | 4 | 3 | 2 | 1 | 0</td></tr>
<tr><td rowspan="4">LPT1</td><td rowspan="4">0x378</td><td>0</td><td>Write</td><td>Data Port</td></tr>
<tr><td>1</td><td>Read</td><td>Status Port</td></tr>
<tr><td rowspan="2">2</td><td rowspan="2">Read/Write</td><td>Control Port
Bits 7, 6: Unused
Bit 5: Enable bi-directional port
Bit 4: Enable IRQ through pin 10
Bit 3: Select printer
Bit 2: Reset printer
Bit 1: Auto linefeed
Bit 0: Strobe</td></tr>
<tr><td>7 | 6 | 5 | 4 | 3 | 2 | 1 | 0</td></tr>
<tr><td rowspan="5">COM1</td><td rowspan="5">0x3F8</td><td rowspan="2">0</td><td rowspan="2">Read/Write</td><td>Data Register</td></tr>
<tr><td>Read receiver data
Write transmitter data</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">Read/Write</td><td>Baud rate divisor low</td></tr>
<tr><td>If DLAB = 1 in LCR
Low byte of 16-bit baud rate</td></tr>
<tr><td>1</td><td>Read/Write</td><td>Interrupt Enable Register (IER)
1 = Enabled, 0 = Disabled
Bits 7 – 4: Reserved
Bit 3: Enable modem status</td></tr>
</table>

<table>
<tr><td rowspan="7"></td><td rowspan="7"></td><td></td><td></td><td>Bit 2: Enable receiver line status
Bit 1: Enable transmitter empty
Bit 0: Enable receiver full
<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>1</td><td>Read/Write</td><td>Baud rate divisor high
If DLAB = 1 in LCR
High byte of 16-bit baud rate</td></tr>
<tr><td>2</td><td>Read</td><td>Interrupt Identification Register (IIR)</td></tr>
<tr><td>3</td><td>Read/Write</td><td>Line Control Register (LCR)
Bit 7: Set DLAB
Bit 6: Set break enable if 1
Bits 5 – 3: Parity select code
Bit 2: Number of stop bits
Bits 1 – 0: Word length
<table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>4</td><td>Read/Write</td><td>Modem Control Register (MCR)</td></tr>
<tr><td>5</td><td>Read.Write</td><td>Line Status Register (LSR)</td></tr>
</table>