COMP 273 Study guide

Francis Piche

April 19, 2018

Contents

Part I
Preliminaries

1   Introduction

This guide is based off of the lectures and slides by Professor Joseph Vybihal at McGill University, Winter 2018. Images are taken from Prof. Vybihal's lecture slides.

This guide is my best attempt to express the ideas of the course in a clear and concise way, given that there is less than a week until the final. That being said, I'll work backwards, starting from the latest material (and in my opinion the hardest) to ensure that the most difficult material will get covered in time for the exam.

2   System Board

3   Number Formats

Part II
Basic Circuits

4   Intro to Circuits

5   RAM & Registers

6   Arithmetic Logic Unit

7   Control Unit

8   Classical & MIPS Pipeline CPU

Part III
Assembler Programming

9   Basic Assembler

10   Complex Data

11   Recursion, Stack & Heap

Part IV
Advanced Circuitry

12   Polling & Interrupts

12.1   Peripheral Devices

Peripheral are external devices such as keyboards, mice, screens, and network adapters.

The devices each have a **controller** (simple CPU).
There are two main types of controllers:

- On-Board: controlling registers are integrated into the system board

- External: controlling registers are part of the card plugged into slots on the system board. These slots are connected to the bus.

  A controller chip is made up of:

- Status register (ready, on/off, error-codes)

- Data register (information to be processed)

- Command register (in binary)

- ROM (to hold basic information)

Often the registers are combined into one.

Note that integrated, on-board controllers are faster, since they skip the slot. Their registers are directly connected to the bus and have addresses.

Also note that if we don't look at registers before the next key press, the data is lost.

12.2   IO & Communication

There are two main techniques for communication:

- Interrupt Driven: Device signals the CPU when state changes.

- Polling Driven: CPU looks at the device's status register

Within each of these techniques, there are two ways to exchange data:

- Synchronous I/O: The CPU monitors the device, sending and reading byte by byte

- Asynchronous I/O: CPU signals when to start, device signals back when finished.

Asynchronous is accomplished by this process: first the CPU loads into registers the start address, limit, and command. The CPU is then free to do anything else until the device sends an interrupt to signal that the task is complete.

The registers are accessed either using a general data path (for example the RAM zero page via the bus), or by using a specialized path such as a DMA or interrupt wire.

12.3   Memory Mapping

There is a special portion of RAM directly allocated to peripheral registers, called the zero page. This mean these special addresses are actually wired to go to the peripheral registers.

12.4   Polling

This is basically accomplished using a busy loop.

```
while(status != 0); // assume 0 means it's ready
```

Now in assembler:

```
LOOP:
    lw $t0, STATUS
    bne $t0, 0, LOOP
#else check the flags and handle
```

Problem is that it uses 100% CPU capacity.

12.5   MIPS I/O Communication

MARS is only able to do simulation for the keyboard and screen (text).

The keyboard is commonly referred to as the **receiver**, and the screen the **transmitter**. The receiver control register's address is at 0xffff0000, followed by the receiver data at 0xffff0004, transmitter control at 0xffff0008, and finally the transmitter data at 0xffff000c.

The first bit of the control registers is the "isReady" bit, 1 meaning ready, 0 meaning not ready.

The second bit of the control registers is to control whether the device is allowed to send an interrupt or not. 1 for yes, 0 for the default (depends on the machine).

12.5.1   GETCHAR and PUTCHAR functions

```
GETCHAR:
   lui $a3, 0xffff       #load address of control register

ISREADY:
   lw $t1, 0($a3)     #read from control register
   andi $t1, 0($t1), 1  #check if first bit is zero
   beqz $1, ISREADY   #if yes then check again
   lw $v0, 4($a3)      #if 1, then load the contents of data bit into v0
   jr $ra             #return

PUTCHAR:
   lui $a3, 0xffff
CHECK:
   lw $t1, 8($a3)     #check the transmitter this time
   andi $t1, $t1, 1   #check if ready
```

```
        beqz $t1, CHECK         #if 0 then try again
        sw $a0, 12($a3)         #if 1 then send character to data register of
            device
        jr $ra
```

Notice the similarities and differences between the two programs. In getChar, we load FROM the data register to GET the character, whereas in putChar we need to load TO the data register to PUT the character. Notice the offsets when accessing the base address.

There was also the process of using logical AND to extract a bit. This is called **bit masking**. When an AND is performed between an unknown sequence of bits and all 1's, the result will be whatever the unknown sequence was!
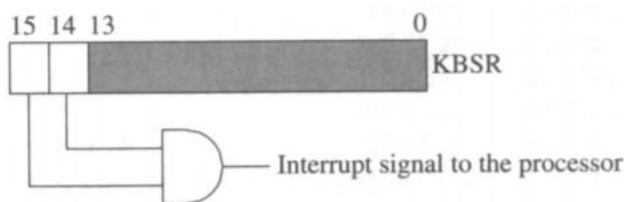
12.6   Interrupts

First, some definitions:

- Exception: Any event that stops the normal execution of the CPU. For example,stopping the program due to divide by zero, stack overflow etc.

- Interrupt: There are two kinds of interrupt: Signal, an event purposely triggered by a program to re-route the CPU flow to another process (think `throw` in Java), and trap, an event purposely triggered by a device.

So just remember: signal interrupt = program throw, trap = device throw. The difference between exceptions and interrupts is that the former is to handle instruction faults (division by zero, undefined op-code, etc) while the latter is to hand external events.

This is a picture of what an interrupt would look like on the register's end:

So basically there's an AND gate on the last two bits. One of the bits is the "enable interrupts bit" and the other is the "interrupt bit". The former is turned to 1 by the programmer (or by default), and then when an interrupt is to occur, the other bit will turn to 1, allowing the interrupt signal to flow out the AND gate.

12.7

13   Cache & Performance

14   Virtual Memory & Performance