Faculty of Science
FINAL EXAMINATION

COMPUTER SCIENCE COMP 273
INTRODUCTION TO COMPUTER SYSTEMS

Examiner:	Prof. Michael Langer	April 18, 2012
Associate Examiner:	Mr. Joseph Vybihal	2 P.M. – 5 P.M.

STUDENT NAME: _____ **ID:** _____

Instructions:

The exam is 16 pages, including this cover page.

There are 11 questions, worth a total of 50 points.

Answer all questions on the exam question sheet.

The 16 pages include two extra pages at the end, which you may use as you wish.

No calculators, notes, or books are allowed.

Question	Points	Grade
1	12	
2	3	
3	4	
4	5	
5	4	
6	4	
7	3	
8	3	
9	4	
10	4	
11	4	
Total	50	

1. **(12 points)**

   For each of the three MIPS instructions below:

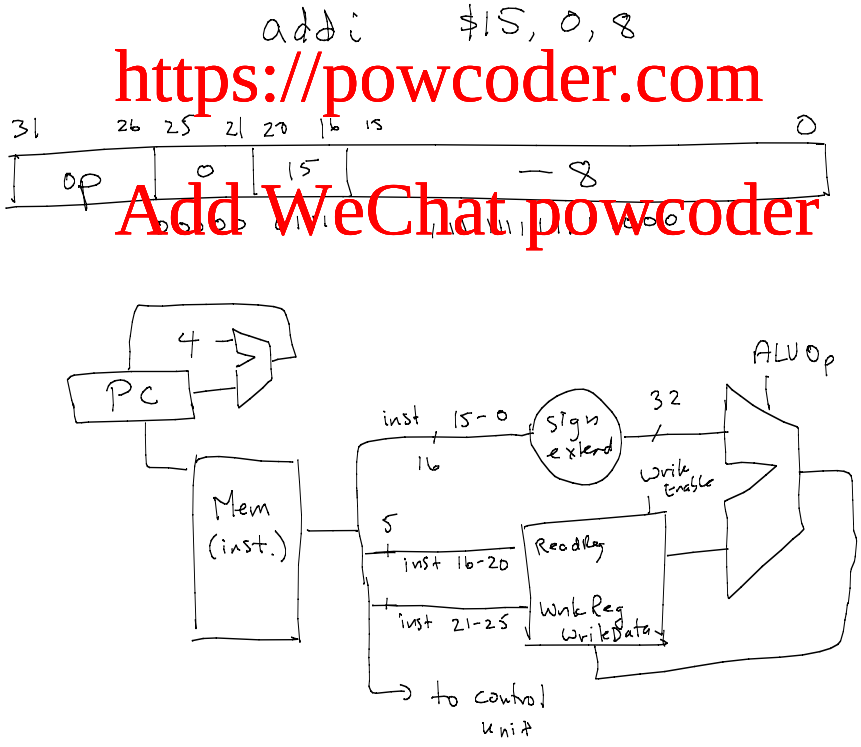
   - specify as much of the machine code of the instruction as you can, showing how many bits are in each field and leaving unknown fields blank;
   - draw the datapath and controls for a *single cycle* implementation of the instruction; only include parts of the datapath that are used in the instruction; specify the bit width of any lines you draw in the datapath, and write any known values on the lines.

   (a)
   ```
   addi   $15,  $0, -8      # add immediate
   ```
   (b)
   ```
   sltu   $16, $15,  $0     # set less than (unsigned)
   ```
   (c)
   ```
   jalr   $16, $15          # special version of jump and link
   ```

   The special version of the jump-and-link instruction, `jalr rs rd`, jumps to the address in register `rs` and puts the return address in register `rd`.

SOLUTION:
Question 1a (7 points) 1 point for each of the following:

- I-format partition with the appropriately labelled fields of each.
- The values 15 and 0 should be given in binary and some indication given of which fields they go into.
- fetch stage: PC register to Memory (instructions) and instruction read out and decomposed into fields.
- $PC \leftarrow PC + 4$
- lines into and out of registers should be labelled: indicate you know which instruction fields correspond to which lines.
- sign extend the immediate argument from 16 to 32 bits
- the value read from the register is fed to the ALU, along with the sign extended value; result is written into a register



SOLUTION:
Question 1b (2 points)

The datapath here is similar to 1a so we only give a few more points here.

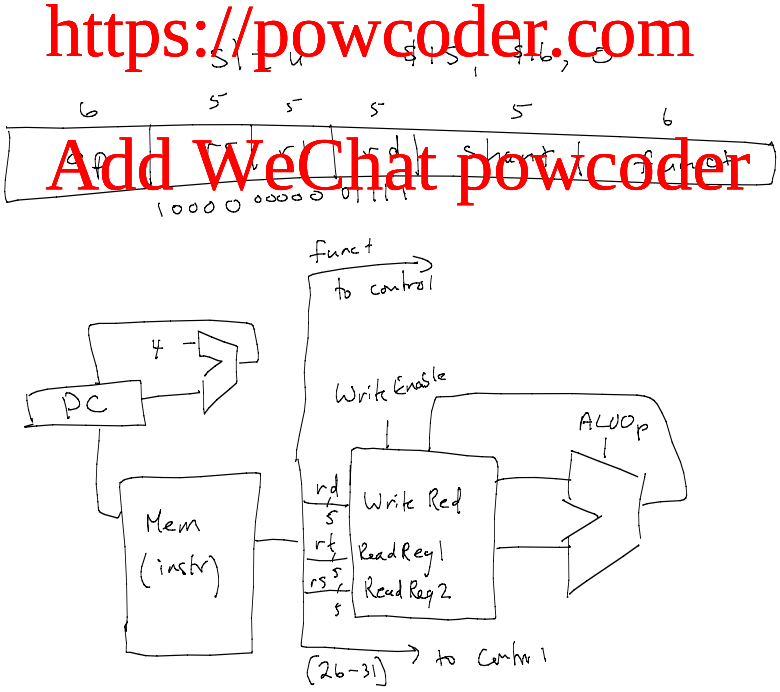
- `sltu` is R format so the partition of the instruction into fields is different. You need to know how many bits are in each field. (You should know the opcode is 000000 for R format, but we didn't require this.)

  The rs, rt, rd fields are 15 (011111), 0 (00000), 16 (10000). The bit codes should be written here and put in correct position.

- Otherwise, it is the same datapath as the add instruction from the lecture notes. It reads two registers, sends the two 32 bit values into the ALU. The result is then written back to the registers.

  I am expecting some labelling of the wires in the registers to indicate that the student knows that the rd register is the one we write to and the rs and rt are the ones we read from.

Some students tried to write up some special circuitry to compute the value from the ALU, but this is not necessary. The `slt` is computed *within* the ALU.



SOLUTION:
Question 1c (3 points)

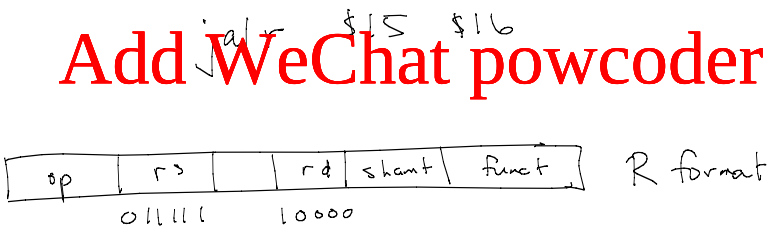
The datapath for the `jalr` instruction is a combination of `jr` and `jal` datapaths which I sketched out in class.

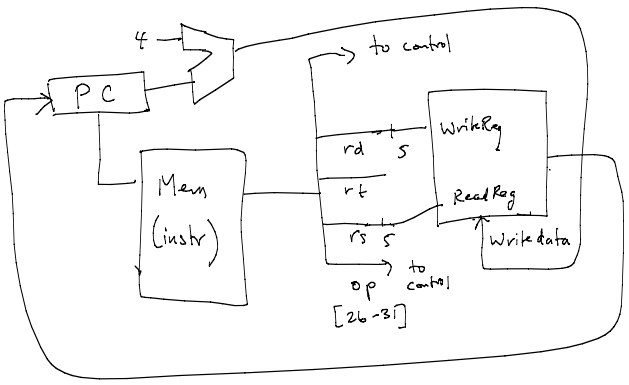
- You might use an I format or an R format for this. (The correct answer is R format but I don't expect you to know that.)

  If you assumed I format, then the rs register (source) register 15 (01111) is read out and fed into the PC, and the destination register would be rt and would have the value 16 (10000). This is where PC + 4 is written.

  If you assumed R format, then the rs register (source) register 15 (01111) is read out and fed into the PC, and the destination register would be rd and would have the value 16 (10000). This is where PC + 4 is written.

- the PC is updated with the 32 bit value from the source register. Some students might think that only the low order 28 bits are used as in the jump instruction (j) but this is incorrect.
- the PC+4 value is written into a register so there should be a line from the PC to the register. (They may use a selector which would be fine.) Some students might put PC in there, and would only get 0.5 points.





2. **(3 points)**

Convert each of the MIPS pseudoinstructions into real MIPS instructions that perform the equivalent operation.

Hint: For (b), you are allowed to use up to three instructions. If you need a temporary register, use `$1`.

(a) `move $s0, $s1`

(b) `seq $s0, $s1, $s2 # set if equal (similar to slt)`

SOLUTION:

(a) **(1 point)**

I was expecting a *one instruction* solutions such as:

```
add  $s0, $s1, $0
addi $s0, $s1, 0
```


Grading scheme. Some students make syntax errors, e.g.

```
addi $s0, $s1 $0
```


Give only 0.5 for that.

(b) **(2 points)**

```
subu $s0, $s1, $s2   #  treats $s1 and $s2 as unsigned.  result $s0
                     #  is either 0 or not zero
addi $1,  $0,  1     #  put the value 1 in $1
sltu $s0, $s0, $1    #  treating $s0 as unsigned, check if it is 0.
```

Another clever solution used `xor` for the first line, instead:

```
xor  $s0, $s1, $s2   #  gives $s0 = 0 if and only if $s1=$s2
```

Grading scheme: Gave 1 point if the student performed a subtraction in the first step but otherwise the solution didn't make sense.

3. **(4 points)**

For a processor that uses a microinstruction implementation:

(a) What is the microprogram counter, and how does it differ from the program counter ?

(b) In general, what are the bits in a microinstruction? How do they differ from the bits in a (regular machine code) instruction ?

(c) How does the processor decide which instruction to execute next?

(d) How does the processor decide which microinstruction to execute next?

SOLUTION:

(a) Each machine code instruction is broken into a sequence of microinstructions. A microprogram counter keeps track of which microinstruction we are executing (within the machine code instruction).

(b) The bits in a microinstruction are the control signals that are set during that microinstruction e.g. ALUOp, WriteReg, ReadMem, etc. This is different from the bits in an actual instruction e.g. in MIPS, these are the opcode, register numbers, funct and shamt fileds (for R format), and immediate fields (see Question 1).

(c) The input line to the PC register is the output of a multiplexor, which selects from various possibilities. In the lecture notes, the selector was PCSrc. The default next instruction address is PC+4, but other possibilities exist e.g. for conditional and unconditional branch instructions, and for exceptions.

(d) The default for the next microinstruction is microPC+1. For the scheme I developed in class, there two cases in which microPC+1 is *not* the next microinstruction to be executed: first, when a microinstruction sequence finishes, and second, after the fetch stage. Which of these three cases is occuring is coded by control signals, and so the next microinstruction is selected using these controls. For the example in class, the control signals were Next, Fetch.

Grading scheme: 1 point for each of the above. For (b), if you just talked about fetch and next as if they were the only bits in the microinstruction, you got only 0.5. For (c) if you only mentioned PC+4, you got 0.5. For (d), if you just wrote "fetch and next" and hardly explained, you got 0.5.

4. **(4 points – was originally 5)**

Identify the hazards in the the following if-then-else sequence. Describe how to resolve these hazards using as few `nop` instructions as possible.

```
if:     slt  $t0, $s0, $s1
        bne  $t0, $0,  else
        lw   $s3, 0($sp)
        add  $s3, $s3, $s4
        j     endif
else:   sub  $s3, $s3, $s4
endif:  and  $t0, $0,  $0
        or   $t1, $0,  $0
```

SOLUTION:

IF	ID	ALU	MEM	WB	
	IF	ID	**ALU**	MEM	WB

There are four (pipeline) hazards to be identified here.

(a) (data hazard) The result of `slt` is not known until the WB stage, but it is needed by the `bne` instruction. The value is known after the ALU stage of `slt` and must forwarded to the ALU stage of the `bne` instruction.

(b) (control hazard) The `lw` and `add` instructions will be fetched while the `bne` is decoded and goes through the ALU, respectively. The `lw` and `add` instructions should only be executed if `bne` condition is *not* met. If the branch condition *is* met, then the WriteReg control of the `lw` and `add` instructions should be set to 0. They will still go through the pipeline but they won't write back.

(c) (data hazard) The `lw` instruction gets a word from Memory but this only happens at the end of the MEM stage. This value is needed at the ALU stage of the `add` instruction. Thus, a `nop` needs to be inserted between `lw` and `add`.

Some students may be tempted to move the `or` instruction in there, instead of `nop`. But this is incorrect since `add` is conditioned on the branch, but `or` is unconditional.

(d) (control hazard) The `sub` enters the pipeline after `j` but it shouldn't be executed. So we (re)set its RegWrite control so it doesn't write in the WB stage.

Grading scheme:

Give 1 point for identifying each hazard and showing how to resolve each. (0.5 for identification and 0.5 for solution)

5. **(4 points)**

Assume a computer has a main memory with 64 MB and a data cache with 256 entries of 4 words each. Assume the cache is accessed using direct mapping.

(a) How many KB (kilobytes) of data are stored in the cache ?

(b) How many bits are in the tag field of each line of the cache?

(c) Suppose a program makes a sequence of five memory accesses in the order shown below. The numbers are physical byte addresses in main memory.

```
0x1a24380
0x211f0d0
0x211f388
0x212f12c
0x1c8fad0
```

What are the indices of the cache entries that are used in these memory accesses? Give you answer in hexadecimal.

(d) What are the contents of the tag fields of these entries after all five memory accesses have been completed ? (Don't show intermediate steps.)

SOLUTION:

(a) $256 \times 8 = 4096$ bytes, or 4 KB

(b) 64 MB requires 26 bits of address space, which are broken into a 2 bit "byte offset", a 2 bit "word offset", a tag, and an 8 bit cache index (256 entries in the cache). The tag is thus 14 bits (26 - 2 - 2 - 8 = 14).

(c) The byte offset and word offset use bits 0,1 and 2,3 respectively. Thus, the cache index is determined by the next 8 bits (4-11), or equivalently, by the hex digits 1 and 2 (where least significant is hex digit 0). The cache indices are 0x38, 0x0d, 0x38, 0x12, 0xad. Note that two of these (0x38) are the same and so there are only four different cache lines that are accessed.

(d) The tag field is upper part of the address, beyond the cache index. (It is 4 hex digits.) The contents of the tag fields after all five accesses are:

```
index   tag
38      211f  (note that the first tag, 1a24, was overwritten)
0d      211f
12      212f
ad      1c8f
```

Grading scheme: if you gave the byte index in (c), we gave you the point. But if you did that they you gave five tags for (d) which was incorrect so we only gave 0.5 points for (d) in that case.

6. **(4 points)**

Consider a cache/TLB design in which the cache is accessed after the virtual page number has been translated to a physical page number (as discussed in class). With this design, the CPU can only fetch an instruction if the address translation is in the TLB and the instruction is in the cache. Thus, the best case scenario for fetching an instruction is a TLB hit followed by a cache hit. Outline the steps that must be taken in the *worst case* scenario, namely the instruction is on the hard disk. (You may assume that the entire page table is in main memory.)

SOLUTION:
Given a virtual address,

```
- check the TLB for translation (TLB miss)
- check page table entry (not valid -> page fault!)
- copy page from hard disk to main memory
- update page table
- copy page table entry to TLB
- translate virtual to physical using new TLB entry
- check cache (miss!)
- copy block from main memory to instruction cache
- fetch instruction from cache
```


Grading scheme: I tried to give 0.5 points for each of the nine steps, above up to a max of 4 points. However, this grading scheme often didn't work since many students mixed up the order and used the wrong terminology. A typical mistake was to put the page fault after the cache miss, or to have two page faults, or to say things like "move the page into the cache".

If there were lots of mistakes, then I fell back on the following: 1/4 for mostly nonsense, 2/4 for missing several important steps but otherwise correct, 3/4 for logic correct but details and a few steps missing.

7. **(3 points)**

   (a) Briefly describe the difference between *isolated I/O* and *memory mapped I/O*. Which does MIPS use ? Explain.

   (b) What are the advantages of memory mapped I/O?

SOLUTION:

(a) Isolated I/O has separate assembly language instructions for different I/O devices. Memory mapped I/O treats addresses in I/O devices as if they are part of the virtual address space.

MIPS uses memory mapped I/O. It uses the upper 64 KB of virtual address space i.e. addresses `0xffff____`.

(b) The advantage of memory mapped I/O is that we can reduce the number of assembly language instructions and "keep it simple". Rather that having specialized instructions for each I/O register specified in the assembly language, the assembly language program can just load and store words to (virtual) memory. The hardware and OS will have to translate the memory mapped I/O address to physical addresses. But exactly how this is implemented is *not* specified at the assembly language level. Having this flexibilty is part of the advantage of memory mapped I/O.

Grading scheme: 2 points for (a) 1 for (b).

8. **(3 points)**

   (a) Describe the *polling* method for I/O, and how it uses the system bus.

   (b) Argue why polling is more suitable than interrupts for mouse input.

   (c) How often do you think the mouse should be polled ?

SOLUTION:
1 point for each of the following:

(a) In polling, the CPU repeatedly and regularly asks an I/O device whether it is ready to input/output data. The CPU puts the number of the I/O device on the address bus and values on the data and control lines which specify relevant information. The I/O device then uses the bus to respond.

(b) Polling is simpler since it can be done between processes and so there is no extra overhead. (Interrupts require saving the state of a process.)
Moving a mouse creates many (x,y) position changes which would generate many interrupts.

(c) It is sufficient to poll at the rate that the screen refreshes, which is say 60 frames per second.

Grading scheme: 1 point for each.

9. **(3 points – originally was 4)**

   (a) What is direct memory access (DMA) ?

   (b) Describe how DMA could be used to record a video signal that is measured with a laptop's camera. Give sufficient detail to show that you understand DMA.

SOLUTION:

(a) DMA is used for transfering data between an I/O device and main memory, without the CPU being used.

With DMA, an I/O device is given control of the system bus, so that the CPU can continue processing instructions (of another process, by using the cache only. With DMA, the CPU tells the I/O device once that it wants to read or write to be done. The I/O device then administers the transfer of data to or from main memory on its own.

(b) Application program must tell the video controller to start recording. The CPU would then gives control of the system bus to the video capture I/O device.

The video capture I/O (DMA) controller would contain a local memory which would buffer the sequence of RGB pixel values in each frame coming from the camera. (This would be hundreds of megabytes per second.) When system bus is granted, DMA would use then copy these buffered frames into main memory (or maybe to hard disk).

Grading scheme:

10. **(4 points)**

(a) The first step that occurs in an interrupt is that the CPU receives a IRQ=1 signal. Describe the steps that the CPU takes after receiving this signal, assuming it allows the interrupt. (In this part of the question, you may assume that there is a separate pair of IRQ/IACK lines for each I/O device.)

(b) Describe the *daisy chaining* method for interrupts.

SOLUTION:

(a) After receiving the interrupt request, IRQ = 1, the CPU decides whether or not to grant it. If yes, then the interrupt handler:

- saves the state of the current process (all the registers).
- sends an IACK signal and gets whatever info it needs to process the interrupt.

(b)
- Daisy chaining is a method for assigning priorities to I/O devices. It allows the devices themselves to decide who gets to interrupt the CPU, rather than having the CPU decide. This is done by passing the IACK signals down a chain. See figure from class, which many students reproduced and explained.
- To make the interrupt request, there is a single pair of control wires IRQ and IACK going to/from CPU. IRQ is ON if and only if at least one of the devices requests a interrupt. This is done with a big OR gate.

11. **(3 points – originally was 4)**

(a) The transfer of data from the computer to a printer does not use a clock. Why not? Describe a source initiated *handshaking* method that is used when the printer is connected by a *parallel port*. Draw a timing diagram to illustrate.

(b) Suppose the one-byte ASCII values, 128 and 142, are sent as part of a message on a *serial bus*. Give a timing diagram of the binary values that would be measured when reading the bus. Explain values for all bits in your timing diagram.

SOLUTION:

(a)
- The reason a computer and printer do not communicate using a clock is that the physical distance between them is too large to make a fast clock feasible. With a clock-based bus, the signal has to stabilize over the entire bus before the end of the clock pulse. For the case of a printer, the cable is of indeterminate length (1 m, 3 m, 5 m) and of inductance. Moreover, the amount of data that is sent per second is much less than with, say, an internet connection off a network, and so we don't *need* a fast bus connecting computer and printer.
- I was just looking for the handshaking timing diagram given in class. Either the source initiated or destination initiated was fine.

(b) The ASCII values 128 and 142 are 10000000 and 10001110, respectively. To transmit these, you need to have a *start bit* and a *stop bit* for each byte.

I gave a bonus point if you mentioned how the devices have a clock which is faster than the pulse width of the bits that are being sent and that the receiving device samples from the *center* of each bit. i.e. each has length T seconds and it samples at T/2 from the start of each bit.