CS250 Spring 2007 Final Exam Solutions, May 5, 2007
Prof. Chris Clifton

Time will be tight. If you spend more than the recommended time on any question, **go on to the next one**. If you can't answer it in the recommended time, you are either going in to too much detail or the question is material you don't know well. You can skip one or two parts and still demonstrate what I believe to be an A-level understanding of the material.

Keep in mind that I am concerned about your knowledge of concepts. If you show that you understand the material, but don't quite get the right answer, I can give partial credit. So describe (briefly) your reasoning if you aren't sure of your answer.

Note: It is okay to use abbreviations in your answers, as long as the abbreviations are unambiguous and reasonably obvious.

Solutions in italics, followed by (anticipated, and subject to change) scoring of each question. Although my expectations will probably change as I grade the exams, at this point I would expect to see an A student get at least 31 of 36, a B student 25. Someone getting less than 20 would need to have demonstrated knowledge elsewhere to convince me they have C-level knowledge.

1 Boolean Algebra/Logic (15 minutes, 6 points)

You are given the following truth table:

A	B	C	O
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	?

1.1 Boolean Logic (5 minutes, 3 points)

Give a boolean logic formula for the above truth table. Note that for "don't care" (?) values in the truth table, your formula can give either a 1 or a 0; it will be correct either way. You will get 2 points for any correct formula; for the full three points, your formula should be as compact as possible.

Letting the "don't care" be 1, we get $O = \overline{A}B + C$. *This is easily discovered with a Karnaugh map.* Scoring: 1 point for showing evidence that you know how to construct a formula, 1 for getting it right, 1 for compact.

1.2 Circuit Diagrams (10 minutes, 3 points)

Draw a circuit diagram to implement the formula (2 points if correct). For full credit, use only one type of gate (remember that logic chips often contain multiple gates of a single type, so using a single type of gate may result in lower cost even if it increases the number of gates.)

Applying DeMorgan's Theorem $\overline{A}B + C = \overline{\overline{\overline{A}B\overline{C}}}$ *This is implemented using four nand gates:*

- *one with both inputs tied to A to get $\overline{A}$,*
- *one with both inputs tied to C to get $\overline{C}$,*
- *one to get $\overline{\overline{A}B}$, and*
- *the final taking this and $\overline{C}$ to get the output.*

Scoring: 1 for getting close to your formula or the truth table, 1 for exact match of either, 1 for using only one type of gate.

2 Memory Mapped I/O vs. System Calls (35 minutes, 9 points)

You are asked to compare two approaches to support a (very fast) output device.

1. Memory-mapped I/O: The device has a control/status register and a data register that are mapped to consecutive words in memory. When a program needs to use the I/O device, it issues a system call that maps the physical address of the device into the program's virtual address space. It then reads/writes to the control register and writes to the data register at the corresponding locations in memory.

2. Kernel-based I/O: A program sets up a system call. One argument to the system call determine if the control register is to be read or written, or the data register is to be written, the other argument is the data to be written (if appropriate); both arguments (as well as the result) are passed in registers (as with the MIPS `syscall`). Assume the syscall is implemented using a software interrupt and is executed as a kernel function (i.e., the code handling the interrupt has direct address to physical memory and can read/write the device registers directly.)

2.1 Performance (10 minutes, 3 points)

Which would you expect to be faster and why?

Memory-mapped, as this saves the switch to kernel space, flushing TLB, switching page tables, possibly flushing a level-1 cache, etc. Scoring: 1 for an answer and explanation showing some understanding, 1 for the explanation matching the answer, 1 for getting the correct answer (memory-mapped).

2.2 Other tradeoffs (15 minutes, 4 points)

Other than performance, give and explain at least one advantage (or explain why there are no advantages) to using:

1. The memory-mapped I/O approach:

   In addition to being more efficient, the memory-mapped approach may give somewhat simpler code, although this is largely a matter of opinion (number of instructions are very similar.) It is also acceptable to say "no advantage other than performance, since any action that could be done using the memory-mapped approach could be performed through appropriate arguments to the syscall." (Scoring: 1 for an advantage, 1 for explanation.)

2. The kernel / system call approach:

   Sharing. With the memory-mapped approach, it is possible for two processes to both use the device, but there is no control over the ordering/interleaving of their use of the device. The kernel could control this with system calls. Other reasonable answers are protection, error checking, wasted virtual space (since a whole page would have to be mapped to get two words), etc. Scoring: 1 for an advantage, 1 for explanation.

2.3 Implementation (10 minutes, 2 points)

Do you need to use different hardware for these approaches, or can the difference be accomplished at the software level? Explain.

Assuming a virtual memory system, the difference is at the software level. The key is if the kernel allows the memory addresses to be mapped into a user virtual address space, or prevents it. The one issue would be caching - the cache would need to recognize that it couldn't cache this address. Scoring: 1 point for answering "can be done in software", 2 points for a good explanation why (or saying "needs special hardware" and explaining that virtual memory hardware or some special cache hardware is needed.

3 MIPS Assembly (20 minutes, 12 points)

For this question, you'll be working with the following piece of MIPS assembly code. (This IS syntactically correct code, and has been run on SPIM.)

```
        .text
Final:  # Purpose: Challenge Students
        # Input, Output, function:  You'll have to figure it out.

        # Prologue
        addiu     $sp,$sp,-8
        sw        $a0,0($sp)
        addiu     $sp,$sp,-8
        sw        $ra,0($sp)

        # Initialize locals
        move      $s1,$0
        li        $s0,10

Loop:   blez      $s0,Done          # Loop termination check
        move      $t0,$sp
        addiu     $t0,8

        lw        $a0,0($t0)        # Set up arguments for subroutine call
        sub       $a0,$a0,$s0
        jal       unknown_subroutine

        bge       $v0,$s1,Next      # If ...
```

```
        move    $s1,$v0
Next:   addi    $s0,$s0,-1
        j       Loop

Done:   move    $v0,$s1
        # Epilogue
        lw      $ra,0($sp)
        addiu   $sp,$sp,16
        jr      $ra             # Return to calling function.
```

Each of the following questions is worth 2 points.

1. What is value of register `s0` the first time `Loop:` is reached?

   10. Scoring: 2 points for right, 1 point if you manage to misinterpret the question but clearly get it right based on your interpretation.

2. What is value of `s0` the first time `Done:` is reached?

   0. Scoring: 2 points for right, not sure how you could get it partially right and still have any clue what is happening. Might give 1 point for "less than or equal to 0."

3. This procedure has a flaw in it. While it does not affect the value produced by the function, it is a mistake that could result in problems. Describe the flaw and give code to fix it. (You can write your answer in the program above if you wish.)

   Procedures fails to save registers s0 and s1 before use. One solution is to add `sw $s0,4($s0)` and `sw $s1,12($sp)` immediately before "Initialized locals", and add `lw $s1,12($sp)` and `lw $s0,4($sp)` at the beginning of the epilogue. Note quite appropriate use of stack convention, but works. Scoring: 1 point for noting the failure to save registers, 1 for wasted stack space, 1 for a reasonable shot at a fix (doesn't need to be syntactically correct.)

4. Give a change to the code that will accomplish the same thing with fewer instructions.

   What I had in mind was replacing the use of register t0 with offset from the stack pointer, e.g., replacing the 3 lines after the blez with `lw $a0,8($sp)`. Might be other valid answers. Scoring: 1 point for recognizing redundancy, 1 for a semantically valid fix.

5. Describe what the result of the function will be. You might not be able to give the exact value, but you should be able to give a description in terms of the arguments and function(s) called.

   Computes the maximum of the result of the function unknown_subroutine applied to the argument to Final -10, argument to Final -9, ..., argument to Final -1. Scoring: 1 point for showing knowledge that result is in register `v0`, 1 for maximum over function called, 1 for proper dependence on argument.

6. Give a change to the code that might make it run faster without changing the number of instructions. Explain. (Credit possible for explaining even if the change you suggest wouldn't actually make a difference on the MIPS.)

   The line immediately before `Loop:` could cause a pipeline stall; the branch depends on the immediately preceding instruction. It would be easy to switch the `move $s1,$0` and `li $s0,10`

to prevent a possible stall. This is one example (and probably wouldn't make a difference on the latest versions of the MIPS processor); other examples may exist as well. What I was looking for was "pipeline stall", although other reasonable answers are valid as well. Scoring: 1 point for noting something other than instruction count that affects time taken, 1 if your example / change could conceivably fix the problem you found.

4 Virtual Memory (15 minutes, 7 points)

Assume a virtual memory system that has a page table and 16 byte pages. The total physical memory in the system is 4096 bytes (i.e., a 12 bit physical address), however each process only has access to 256 bytes (i.e., an 8 bit virtual address). A memory dump of a portion of physical memory is given below, along with a page table.

Physical Address	*Data (bytes)*															
080	2A	1B	0C	3D	5A	02	13	12	22	13	4A	0B	10	21	21	12
090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C0	02	1B	A1	2C	11	31	22	33	11	12	14	2B	11	2B	15	13
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0E0	02	01	03	04	11	01	01	0B	11	10	12	13	00	03	11	0B
0F0	00	00	01	10	00	00	00	00	00	00	00	00	00	00	00	00
100	1A	2B	3C	4D	5E	01	10	02	20	03	40	0A	11	22	01	10
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
130	01	1A	A0	2B	10	21	12	13	01	02	04	2A	01	1B	02	03
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
160	01	01	02	03	10	01	02	0A	10	11	02	03	00	02	10	0A
170	01	02	02	01	01	02	03	04	00	00	00	00	00	00	00	00

Page Table:

0	08
	09
	0A
	0B
	0C
	0D
	0E
	0F
	10
	11
	12
	13
	14
	15
	16
P	17

1. (2 points) Give the physical address corresponding to the virtual address 70.

   70 = page 7 = 0F, offset 0, so the answer is 0F0. Scoring: 1 point for getting page correct, one for offset correct.

2. (2 points) What is the byte at the virtual address C1?

   00. Scoring: 2 for correct answer, 1 for 1B (Physical address C1).

3. (2 points) What is the decimal value of the 32 bit integer at the virtual address EC? What assumptions did you make in arriving at your answer?

   135178. I assumed Big-endian notation, as it made my calculation easier. Scoring: 1 for an answer based on 32-bit word beginning at Physical address 16C, 1 for noting you need to know representation, 1 for noting you need to know byte order, 1 for correct calculation.

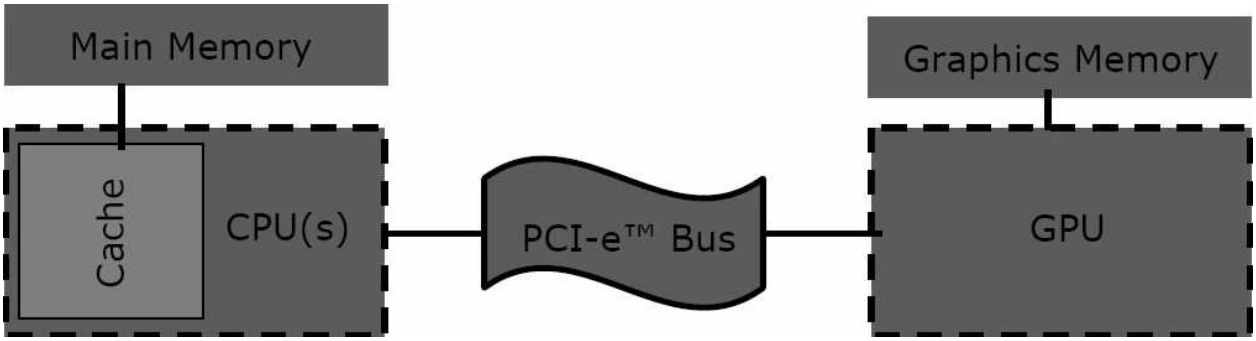
4. (1 point) Describe a simpler method of implementing virtual memory that would give exactly the same mapping as the above page table.

   Using a base register with a value of (page) 08 and a bound register with a value of 16 (pages) would give the same result. Scoring: 1 for saying base-bound (or anything else reasonable.)

5 Novel Architectures (15 minutes, 2 points)

You are given the task of speeding up an N-body (or particle) simulation. The idea is that movement of a particle is governed by forces of nearby particles (think electromagnetic or gravitational.) After the movement of a particle is determined, a nearby particle (that might be affected by the movement of the previous particle) is chosen, and the operation is repeated.

You discover that the most expensive operation is computing the new position/direction of a particle; this is a floating point equation involving several neighboring particles. You read that the Graphics Processing Unit (GPU) in your computer is capable of very fast parallel floating point operations, and have the bright idea to use the GPU to compute the new position/direction. The GPU is a separate processor, as shown in the following diagram.



You write (brilliant) code to enable the GPU to compute the new position/direction of a particle given the particle and its neighbors, and discover that this operation takes 1/10th of the time it takes to compute on the CPU. Based on this, you rewrite the program to send each such computation to the GPU, with the result returned to the CPU to choose the next computation to send.

1. You find that while the program works, the overall simulation runs *slower* than the original. Give a good reason why (based on what you have learned in the course.)

   The problem is that sending the "next point" to the GPU over the PCI-e bus takes longer than computing the position on the CPU. Scoring: 1 point for recognizing this problem (or some other reasonable suggestion.)

2. You still like the idea of using the GPU to do the calculations, since it is much faster - but it doesn't really have the capability to run the complete program. Can you think of a way to make this work? (This might require hardware or software changes – it isn't necessarily something you will be able to do with the computer you have.)

   A heterogeneous multicore architecture with the GPU on the same chip as the CPU could overcome this problem. Scoring: 1 point for recognizing that it is possible to move the GPU closer to the CPU, or some other reasonable solution to the problem you identify.