ELE375/COS471B, COS471A Final Exam
Fall 2004

Prof: David August

TAs: Jonathan Chang, and Junwen Lai, and Neil Vachharajani

January 12, 2005

Question	Points Possible	Points Earned
1	20	
2	20	
3	20	
4	20	
5	20	
6	20	
Total	120	

Please write your answers clearly in the space provided. For partial credit, neatly show all work. State all assumptions. You have *exactly* 3 hours for this exam. This final is closed book. Only two two-sided, handwritten 8.5x11 sheets are allowed. Put your name on every page. Write out and sign the honor code just before turning in the test. *"I pledge my honor that I have not violated the Honor Code during this examination"*

Name:

Course:

Honor Code:

1 True/False

Are the following statements true or false? Indicate by circling TRUE or FALSE. If the answer is FALSE, *briefly* describe why.

1. **TRUE** **FALSE** The MIPS CPU has a CISC ISA.

2. **TRUE** **FALSE** For a computer to multiply two numbers, the processor must have a multiplication unit.

3. **TRUE** **FALSE** A IEEE denormalized floating point number is a floating point number that is in the form $(-1)^a \times 0.b \times 2^c$, where a is the sign bit, b is the mantissa represented in binary, and c is the exponent represented in binary.

4. **TRUE** **FALSE** A 2-way set associative cache will always have the same number or fewer misses than a direct-mapped cache of the same size.

5. **TRUE** **FALSE** A DMA controller can be used to speed the loading of programs into memory from disk prior to execution.

6. **TRUE** **FALSE** Pipeline registers are a part of MIPS architectural state.

7. **TRUE** **FALSE** Booth's algorithm speeds addition when long runs of 1's and 0's are present.

8. **TRUE** **FALSE** Tomasulo's algorithm eliminates the need for compiler scheduling.

2 Caches

```
int i; int a[1024*1024]; int x=0;

for(i=0;i<1024;i++)
{
    x+=a[i]+a[1024*i];
}
```

Figure 1: Code Snippet

2.1 Misses and Hits

Consider the code snippet in Figure 1. Suppose that it is executed on a system with a 2-way set-associative 16KB data cache with 32-byte blocks, 32-bit words, and an LRU replacement policy. Assume that `int` is word-sized. Also assume that the address of `a` is 0x0, that `i` and `x` are in registers, and that the cache is initially empty. How many data cache misses are there? How many hits are there?

2.2 MIPS

Compile the code snippet in Figure 1 into MIPS assembly. Put `i` into `r1` and `x` into `r2`. (You may find it useful, for the subsequent questions involving execution time, to avoid pseudo-ops here.)

2.3 Execution Time

Assume that

- Instructions execute at a rate of 1 IPC,
- the system has perfect branch prediction,
- the pipeline is five stages deep,
- there are no data hazards except due to cache misses,
- and that data cache misses incur a 10 cycle penalty.

How long will the aforementioned code take to execute (include fill/drain time)? What percentage of the execution time is taken up by data-memory stalls (leave as fraction)?

2.4 Prefetching

The Pentium 4 uses a prefetching scheme to improve performance in certain situations. It works as follows. Suppose a program has 3 successive *cache misses* of the form $a, a+b, a+2b$. Then the system automatically begins fetching $a+3b, a+4b, a+5b, \ldots$ How long does the program take to execute with this prefetching? What percentage of the execution time is taken up by stalls (leave as fraction)? Assume that prefetching ahead does not evict any data which will be accessed.

3 Branch Handling

3.1 Branch Prediction

Consider adding branch prediction to the standard 5-stage MIPS pipeline (originally without branch prediction). Assume that the pipeline does *not* use delayed branching. Further assume that branches are dynamically predicted in the ID stage of the pipeline and that branches are resolved (i.e. the true outcome of the branch is identified) in the MEM stage of the pipeline.

Answer the following questions and provide explanations for each. Provide any diagrams you think will be helpful to explain your answer. Remember that the processor is *not* using delayed branches. You can assume all bubbles are due to the branch instructions.

1. How many bubbles must be inserted if a branch is correctly predicted not taken?

2. How many bubbles must be inserted if a branch is correctly predicted taken?

3. How many bubbles must be inserted if a branch is incorrectly predicted not taken?

4. How many bubbles must be inserted if a branch is incorrectly predicted taken?

3.2 Delayed Branching

Assume the machine from problem 3.1 is extended to use delayed branching. The machine will use 1 delay slot. That is to say, the instruction after any branch is *always* executed regardless of whether the branch is taken or not. For each of the four cases from 3.1, will the number of bubbles be reduced? If so, why is it reduced and how many bubbles remain in the new scenario?

3.3 Dynamic Branch Prediction

Consider the following code sequence.

```
     bne r10, r11, .L1
     add r3, r1, r2
     j   .L2
.L1: sub r3, r1, r2
.L2: bne r10, r11, .L3
     add r4, r5, r6
     j   .L4
.L3: sub r4, r5, r6
.L4: bne r10, r11, .L5
     add r7, r3, r4
     j   .L6
.L5: sub r7, r3, r4
.L6:
```

Assume that this code sequence is part of some larger program and is frequently executed. Further assume that each of the branches in this code sequence is unbiased; that is to say, it is equally likely that they are taken or not taken. Would a GAg branch predictor or a GAp branch predictor better predict the branches in this sequence? Why? Assume that the history register is 1-bit long and there exists no aliasing between any branches in the program.

4 Virtual Memory

4.1 32-bit Virtual Address Spaces

Consider a machine with 32-bit virtual addresses, 32-bit physical addresses, and a 4KB page size. Consider a two-level page table system where each table occupies one full page. Assume each page table entry is 32 bits long. To map the full virtual address space, how much memory will be used by the page tables?

4.2 64-bit Virtual Address Spaces (Part 1)

Consider a machine with a 64-bit virtual addresses, 64-bit physical addresses, and a 4MB page size. Consider a two-level page table system where each table occupies one full page. Assume each page table entry is 64 bits long. To map the full virtual address space, how much memory will be used by the page tables? (*Hint: you will need more than 1 top-level page table. For this question this is okay.*)

4.3 64-bit Virtual Address Spaces (Part 2)

Rather than a two-level page table, what other page table architecture could be used to reduce the memory foot print of page tables for the 64-bit address space from the last question? Assume that you do not need to map the full address space, but some small fraction (people typically do not have 2^{64} bytes of physical memory). However, you should assume that the virtual pages that are mapped are uniformly distributed across the virtual address space (i.e. it is not only the low addresses or high addresses that are mapped, but rather a few pages from all ranges of memory). Explain how your solution works and how it conserves memory.

4.4 Caching and Virtual Memory

Consider an architecture that uses virtual memory, a two-level page table for address translation, as well as a TLB to speed up address translations. Further assume that this machine uses caches to speed up memory accesses. Recall that all addresses used by a program are virtual addresses. Further recall that main memory in the microarchitecture is indexed using physical addresses.

The virtual memory subsystem and cache memories could interact in several ways. In particular, the cache memories could be accessed using virtual addresses. We will refer to this scheme as a virtually indexed, virtually tagged cache. The cache could be indexed using virtual addresses, but the tag compare could happen with physical addresses (virtually indexed, physically tagged). Finally, the cache could be accessed using only the physical address. Describe the virtues and drawbacks for each of these systems. Be sure to consider the case where two virtual addresses map to the same physical address.

5 Disk Technology

Suppose we have a magnetic disk (resembling an IBM Microdrive) with the following parameters. (You may leave any answer as a fraction.)

Average seek time	12 ms
Rotation rate	3600 RPM
Transfer rate	3.5 MB/second
# sectors per track	64
Sector size	512 bytes
Controller overhead	5.5 ms

1. What is the average time to read a single sector?

2. What is the average time to read 8 KB in 16 consecutive sectors in the same cylinder?

3. Now suppose we have an array of 4 of these disks. They are all synchronized such that the arms on all the disks are always on the same sector within the track. The data is striped across the 4 disks so that 4 logically consecutive sectors can be read in parallel. What is the average time to read 32 consecutive KB from the disk array?

6 Instruction-Level Parallelism

Consider the following program:

```
      r2 = 8 * N
LOOP: r1 = MEM[r2]
      r3 = r1 + 1
      MEM[r2] = r3
      r2 = r2 - 4
      br r2 > 0, LOOP
```

Note: the loop always executes an even number of times

6.1 Pipeline Hazards

How many cycles does each iteration of the above loop take to execute (ignoring pipeline fill and drain time) on a pipelined machine where each instruction takes one cycle to execute, except for loads which require 3 cycles? Note: Since the machine is pipelined, 3 loads can execute in successive cycles, but the load-use latency is 3 cycles. This means that if a load begins execution in cycle 0, then the earliest a consumer can begin execution is cycle 3.

6.2 ILP Optimizations

Apply ILP optimizations to the above program to improve the performance of the program on the same machine as problem 6.1. Your optimized program should be faster than the original. You may use any MIPS-like instruction.

6.3 VLIW

Consider a pipelined VLIW machine that can execute two instructions each cycle. The machine is statically scheduled so the compiler (that's you!) must tell it which instructions to execute in parallel. The machine can execute any pair of instructions, but only one branch may execute per cycle, and the branch must be the second instruction in the pair (the first instruction will execute regardless of the branch outcome). Registers for both instructions in the pair are read before the results for either instructions are written. Two instructions in the same pair cannot write to the same register. Just like the previous machine, all instructions take 1 cycle to execute except loads. Once again, loads take 3 cycles and can be pipelined, but the load-use latency is 3 cycles.

Apply ILP optimizations to the original program to maximize its performance on this machine. Make sure you provide the execution schedule of your optimized program (*i.e.* show which instructions run in parallel, and which cycles are empty due to stalls). Your optimized program should run at least twice as fast as the original, unmodified program on the basic pipelined machine.

6.4 Exceptions

If the loop could execute an even or odd number of times (rather than just an even number of times), would the code you provided for the VLIW machine be correct? Why or why not? (Hint: remember that these memory instructions might generate exceptions!)