CSE 378 Winter 1998

Machine Organization and Assembly Language Programming

Final

Wednesday March 18

NAME : ______________________________

Do all your work on these pages. Do not add any pages. Use back pages if necessary. Show your work to get partial credit.

This exam is worth 100 points. After each question, you will find the number of points it is worth. You should spend approximately x minutes on a question worth x points (e.g., 12 minutes on question 1 worth 12 points). That will leave you with 20 minutes to look over your work.

1. 12 points ____________
2. 26 points ____________
3. 13 points ____________
4. 15 points ____________
5. 12 points ____________
6. 12 points ____________
7. 10 points ____________

1. (12 points)
Given a computer with the MIPS ISA, a processor implemented with a pipelined ALU, on-chip instruction and data caches, and a TLB, which of the following factors would improve, worsen or not impact the execution time of a program. Justify your answer (a single sentence per question should suffice).

(a) Take advantage of newest technology to increase the clock cycle rate.

(b) Use an optimizing compiler so that fewer instructions are executed.

(c) Disallow forwarding in the pipeline.

(d) Doubling the instruction and data cache capacities without changing the clock cycle time.

(e) Do not have the register 0 hardwired to the value 0.

(f) Increasing the disk capacity.

2. (26 points) (This question continues on the next three pages)
After having studied the MIPS architecture, a group of designers decides to change the structure of the pipeline as follows.

- The pipeline has 6 stages instead of 5 (see table below). Now the Memory stage is replaced by two stages MEM1 and MEM2.
- The branch target computation starts in stage 2 and the result of the branch target computation and of the comparison is known at the end of the third stage (i.e., the value of the PC will be modified at the end of stage 3 if the branch is successful. The fetch of the target instruction can start on the subsequent cycle).

Stage name	Mnemonic	Function
IF	IF	Instruction fetch
ID	ID	Instruction decode and register fetch Compute branch target At this point it is known if the instruction is a branch
EX	EX	ALU result computed Effective address of memory operand available Branch condition tested PC updated with target address if needed
MEM1	M1	First part of memory cycle
MEM2	M2	Memory access complete
WB	WB	Write of the result of either an arithmetic instruction or a Load instruction in the register file

In addition, assume that:

- Instruction and data caches are separate and can be accessed concurrently in the same cycle.
- The register file is multiported, i.e., two registers can be read and one register can be written in the same cycle. In the last stage, the result register can be written in the first part of the cycle and read (from the register file) in the second part of the cycle.

When drawn out the pipeline looks like this:

IF1	ID	EX	M1	M2	WB						
	IF1	ID	EX	M1	M2	WB					
		IF1	ID	EX	M1	M2	WB				
			IF1	ID	EX	M1	M2	WB			
				IF1	ID	EX	M1	M2	WB		
					IF1	ID	EX	M1	M2	WB	
						IF1	ID	EX	M1	M2	WB

Assume no forwarding and no optimization unless explicitly indicated.

Questions [a-d], [e-i], and [j-k] are independent.

(a) (2 points) Give a two-instruction sequence of MIPS assembly-language arithmetic operations with a data dependency (RAW hazard).

(b) (3 points) Assuming no forwarding, how many stall cycles will there be between the 2 instructions that you just wrote?

(c) (2 points) Indicate what kind of forwarding you would advocate to minimize the number of stalls *in this case.* Indicate the stages involved in the forwarding process. How many stalls will remain after the forwarding?

(d) (3 points) Between which stages should you add forwarding hardware if you never wanted to stall between two arithmetic operations?

(e) (2 points) Give a two-instruction sequence of MIPS assembly-language with a load operation followed by an arithmetic operation with a data dependency (RAW hazard).

(f) (2 points) Assuming no forwarding, how many stall cycles will there be between the two instructions that you just wrote?

(g) (3 points) Indicate what kind of forwarding you would advocate to minimize the number of stalls *in this case*. Indicate the stages involved in the forwarding process. How many stalls will remain after the forwarding?

(h) (2 points) Assuming forwarding how many stall cycles will be incurred when the following instruction sequence is executed;

```
lw    $5, 4($4)
sw    $6, 8($4)
```

Indicate what type of forwarding is needed, if any.

(i) (2 points) Assuming forwarding how many stall cycles will be incurred when the following instruction sequence is executed;

```
lw    $5, 4($4)
sw    $5, 8($4)
```

Indicate what type of forwarding is needed, if any.

(j) (3 points) How many cycles will be lost when a branch is encountered (assume that the pipe is stalled as soon as a branch is decoded)?

(k) (2 points) Assume a branch not taken optimization takes place. How many cycles will be lost when a branch is encountered and the branch is not taken? How many cycles will be lost when a branch is encountered and the branch is taken?

3. (13 points) This question has no relation with the preceding one, so don't infer anything about stalls etc.

Consider two implementations of the MIPS ISA. The first one, say M1, uses a machine with a 5 stage pipeline and a cycle time of 10 ns. The second one, say M2, uses a 6 stage pipeline and a cycle time of 8 ns.

(a) (3 points) Define the throughput of a pipeline. Which of the pipelines above has better throughput? Justify your answer quantitatively.

(b) (10 points) Consider now the performance of the two machines when executing the following loop of 6 instructions (this loop has no intended meaning).

```
Loop:     addi     $t0,$t0,-1
          sub      $t2,$t2,$t3
          lw       $t4,0($t1)
          add      $t3,$t4,$t6
          addi     $t1, $t1,4
          bnez     $t0, loop
```

In design M1, data dependencies (RAW) between consecutive instructions induce a stall of 1 cycle. In design M2, data dependencies (RAW) between consecutive instructions induce a stall of 2 cycles and data dependencies between instruction i and instruction $i+2$ induce a stall of 1 cycle.

In both designs the taken branch induces a stall of 2 cycles.

What are the average CPI's of M1 and M2 when executing the loop (consider an iteration in steady state, i.e., neither the first nor the last one)? Which of the 2 machines has better performance when executing the loop (you may assume that the effects of execution of the first and last iterations are negligible)?

(You may do your work on the next page)

4. (15 points) (This question continues on the next page)

Check entries in the following tables that correspond to reasonable parameters and policies for the design and management of the memory hierarchy: cache, translation buffer (abbreviated TB), and paging system (abbreviated PS).

In some cases, you might need to check more than one entry per row in the tables.

(a) (3 points)

Capacity

	512 bytes	8KB	128KB	1 MB	64MB
I-Cache (on-chip)					
D-Cache (on-chip)					
Cache (off-chip)					

(b) (3 points)

Organization

	Direct-mapped	4-way set-associative	Fully associative
Cache			
TB			
PS			

(c) (3 points)

Write policy

	Write-back	Write-through
Cache		
TB		
PS		

(d) (3 points)

Mainly hardware or mainly software action?

	Hardware	Software
Cache miss		
TB miss		
Page fault		

(e) (3 points)

Context-switch or not?

	Context-switch	No context-switch
Cache miss		
TB miss		
Page fault		

5. (12 points) Consider the memory hierarchy (cache + TLB + paging system) of a system with the following components:

- Virtual address: 32 bits
- Physical address: 32 bits
- Page size: 4 KBytes
- TLB: 64 entries 2-way set associative
- On-chip I-cache: 8 KByte direct-mapped, line size 16 bytes
- On-chip D-cache: 16 KB 2-way set-associative, line size 32 bytes

(a) (6 points) Show the format of one TLB entry. Indicate the length and function of each field.

(b) (3 points) Show how the physical address is divided when the I-cache is accessed.

(c) (3 points) Show how the physical address is divided when the D-cache is accessed.

6. (12 points) Consider a pipelined RISC processor with on-chip instruction and data caches. The data cache is write through, write-around. Under ideal conditions, once the pipeline is full, an instruction is executed every cycle. Thus the ideal CPI is 1. In particular, this means that a cache hit takes 1 cycle. These ideal conditions are not met in the following cases:

- Execution of a branch instruction when the branch is taken. In that case, the pipeline is stalled for 2 cycles.
- A miss in the instruction cache. In that case, the pipeline is stalled for 12 cycles (or, in other words the memory access penalty is 12 cycles).
- A miss in the data cache. On a read miss, the pipeline is stalled for 12 cycles (or, in other words the memory access penalty is 12 cycles). On a write miss, there is enough buffering in the system so that the pipeline does not have to stall (or, in other words there is no memory access penalty).

While running benchmarks, the following measurements were reported:

- Branches occur 30% of the time. Two-thirds of the branches are taken.
- Load-store instructions occur 36% of the time with loads being twice as frequent as stores.
- The instruction cache hit rate is 0.96.
- The data cache read hit rate is 0.92. The data cache write hit rate is 0.88.

What is the CPI that can be computed from these benchmarks? What is the average memory access time for bringing instruction and data in the caches ?

(Show all your work. You might get partial credit even if the final answer is wrong.)

7 (10 points) Which hardware and software structures are affected when, on a page fault, a page has to be written back from memory to disk? Indicate the actions that have to be taken (e.g., invalidations etc.). Limit yourself to the components of the memory hierarchy that we have studied and do not dwell in the details of the O.S. (e.g., do not consider the data structures needed to know which page to replace).

How does the O.S. know that the page has been written back?

What happens to the program which had a page fault?