ELE 375 / COS471B, COS 471A Final
Fall 2003

Prof: David August
TAs: Neil Vachharajani
David Penry

Question	Score
I	/20
II	/20
III	/20
IV	/20
V	/20
VI	/20
TOTAL	/120

Please write your answers clearly in the space provided. For partial credit, show all work. State all assumptions. You have 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 pledge before turning in the test. *"I pledge my honor that I have not violated the Honor Code during this examination."*

NAME: SOLUTION

COURSE (circle one): COS471B/ELE375 COS471A

HONOR CODE:

QUESTION I:

Mark the following statements as either T (true) or F (false) and explain why.

1. Synchronous timing uses a global clock to coordinate bus transactions.

   T

2. An invalidation cache coherence protocol performs better than an update one when shared data is typically written several times by a processor before another processor reads it.

   T

3. A memory-mapped I/O architecture requires additional opcodes to specify I/O access instructions.

   F, ld/st suf.

4. A processor using vectored interrupts does not require the exception PC (EPC) to be stored when an exception or interrupt occurs.

   F, still need point of event

5. Caching of memory mapped I/O pages is a good way to improve performance.

   F, coherence problems exist when mmio data changes.

6. A 128-byte, fully associative cache with 16 byte cache blocks is the same as a 128-byte, 8-way set associative cache with 16 byte cache blocks.

   T

7. Predicated execution removes the need for jump and branch instructions in an ISA.

   F, consider endless loops
   func. call

8. A virtual cache has shorter access time than a physical cache because it does not require address translation as part of the access path.

   T

QUESTION II:

Give short answers to the following questions:

Suppose that you have an un-pipelined processor which requires 10 clock cycles to perform a memory access, 4 cycles to perform a branch, and 2 cycles to perform all other instructions. Suppose further that the processor clock frequency is 1500 MHz.

(8) 1. Suppose a program runs for 100 billion dynamic instructions. The dynamic instructions consist of 30% memory accesses, 10%, branches, and 60% other instructions. How much time will the program take to run?

$(0.30 \cdot 10) + (0.10 \cdot 4) + (0.60 \cdot 2) =$ 4.6 CPI

$(4.6 \text{ CPI} \cdot 100 \cdot 10^9 \text{ instr.}) = 460 \cdot 10^9$ cycles

$(460 \cdot 10^9 \text{ cycles}) \cdot \frac{1}{1500 \cdot 10^6 \frac{\text{cycles}}{\text{sec}}} = 306.\overline{6}$ seconds

(6) 2. A new design technique allows you to run the processor at 2000 MHz, but memory accesses now take 15 cycles. How much faster would a processor using the new design technique be?

$(0.30 \cdot 15) + (0.10 \cdot 4) + (0.60 \cdot 2) = 6.1$ CPI

only new time: -3

not reporting speedup: -2 (just decrease)

inverted ratios: -2

$6.1 \text{ CPI} \cdot 100 \cdot 10^9 \text{ instr} = 610 \cdot 10^9$ cycles

$\frac{610 \cdot 10^9 \text{ cycles}}{2000 \cdot 10^6 \text{ cycles/sec}} =$ 305 sec.

$\text{speedup} = \frac{t_{slower}}{t_{faster}} = 1.0055 \left(= \frac{184}{183}\right)$

(6) 3. Better compilation techniques might reduce the number of memory accesses. What is the highest speedup (relative to the original processor) possible by reducing the number of memory accesses if all other factors remain constant?

memory is .652 of time $\left(\frac{0.30 \cdot 10}{4.6}\right) = f$

0.3 used as fraction: -2

Amdahl's law: $\frac{1}{1-f} = 2.875 \left(= \frac{23}{8}\right)$

Miss # instr. change per ratio change: -3

QUESTION III:

The MIPS instruction set architecture contains a jump-and-link (`jal`) instruction and a jump-register (`jr`) instruction to support function calls and function returns. The `jal` instruction stores `PC+4` in the register `$ra` and jumps to the specified function address. The function can then return back to the caller by performing a `jr` to `$ra`.

If the `jal` instruction were removed from the ISA, could function calls and returns still be implemented in this reduced MIPS ISA? Explain.

Yes. The assembler could implement jal as a pseudo-instruction by explicitly moving the return address into $ra, and then performing a normal jump to the start of the function.

The return would be handled in the normal way, because the return address would be in $ra, as expected.

The pseudo-assembly follows (I forget the exact syntax, but the idea is correct) It shows both a call to function FUNC, and the returning.

```
loadhi  $ra, <bits 31-16 of RET>
li      $ra, <bits 15-0 of RET>
j       FUNC
RET:
        :
FUNC:
        jr $ra
```

QUESTION IV: Caches

1. A processor has a 32 byte memory and an 8 byte direct-mapped cache. Table 0 shows the current state of the cache. Write hit or miss under the each address in the memory reference sequence below. Show the new state of the cache **for each miss** in a new table, label the table with the address, and circle the change:

Addr	10011	00001	00110	01010	01110	11001	00001	11100	10100
H/M	m	H	H	m	m	m	m	m	m

0. Initial state

Index	V	Tag	Data
000	N		
001	Y	00	Mem(00001)
010	N		
011	Y	11	Mem(11011)
100	Y	10	Mem(10100)
101	Y	01	Mem(01101)
110	Y	00	Mem(00110)
111	N		

1. 10011

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	N		
011	Y	10	m[10011] ←
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	00	m[00110]
111	N		

2. 01010

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	Y	01	m[01010] ←
011	Y	10	m[10011]
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	00	m[00110]
111	N		

3. 01110

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	01	m[01110] ←
111	N		

4. 11001

Index	V	Tag	Data
000	N		
001	Y	11	m[11001] ←
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	01	m[01110]
111	N		

5. 00001

Index	V	Tag	Data
000	N		
001	Y	00	m[00001] ←
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	01	m[01110]
111	N		

6. 11100

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	11	m[11100] ←
101	Y	01	m[01101]
110	Y	01	m[01110]
111	N		

7.

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	10	m[10100] ←
101	Y	01	m[01101]
110	Y	01	m[01110]
111	N		

2. Do the same thing as in part 1, except for a 4-way set associative cache. Assume 00110 and 11011 were the last two addresses to be accessed. Use the Least Recently Used replacement policy.

Addr	10011	00001	00110	01010	01110	11001	00001	11100	10100
H/M	Miss	Hit	Hit	Miss	Miss	Miss	Hit	Miss	Miss

0. initial state

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0			0011	Mem(00110)			1010	Mem(10100) L
1	0000	Mem(00001)	1101	Mem(11011) L	0110	Mem(01101)		

1. 10011

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0			0011	m[00110] L			1010	m[10100] L
1	0000	m[00001] L	1101	m[11011] L	0110	m[01101]	1001	m[10011] L

2. 01010

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010] L	0011	m[00110] L			1010	m[10100] L
1	0000	m[00001] L	1101	m[11011] L	0110	m[01101]	1001	m[10011] L

3. 01110

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010] L	0011	m[00110] L	0111	m[01110] L	1010	m[10100] L
1	0000	m[00001] L	1101	m[11011] L	0110	m[01101]	1001	m[10011] L

4. 11001

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010] L	0011	m[00110] L	0111	m[01110] L	1010	m[10100]
1	0000	m[00001] L	1101	m[11011]	1100	m[11001] L	1001	m[10011] L

5. 11100

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010] L	0011	m[00110]	0111	m[01110] L	1110	m[11100] L
1	0000	m[00001] L	1101	m[11011]	1100	m[11001] L	1001	m[10011] L

6. 10100

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010]	1010	m[10100]	0111	m[01110]	1110	m[11100]
1	0000	m[00001]	1101	m[11011]	1100	m[11001]	1001	m[10011]

7.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

3. What is the hit and miss rate of the direct-mapped cache in part 1?

H: $\frac{2}{9}$

m: $\frac{7}{9}$

2

4. What is the hit and miss rate of the 4-way set associative cache in part 2?

H: $\frac{3}{9}$

2

m: $\frac{6}{9}$

5. Assume a machine with a CPI of 4 and a miss penalty of 10 cycles. Ignoring writes, calculate the ratio of the performance of the 4-way set associative cache to the direct-mapped cache. In other words, what is the speedup when using the machine with the 4-way cache?

Hit: 4

2

Miss: 14

$\frac{\frac{4 \cdot 2}{9} + \frac{7 \cdot 14}{9}}{\frac{4 \cdot 3}{9} + \frac{6 \cdot 14}{9}}$ ← Time DM

$= \frac{8 + 7 \cdot 14}{12 + 6 \cdot 14} = \frac{98+8}{84+12} = \frac{106}{96}$

$= \frac{53}{48}$

QUESTION V: Virtual Memory

Give short answers to the following questions:

1. What is a page table?

page tables translate virtual addresses to physical addresses

they are a list of virtual page numbers (the upper bits of the virtual address) and the physical address (again the upper bits) that they correspond to

same lower number of bits called the page offset are constant b/w the two

2. What is a Translation-Lookaside Buffer (TLB)?

TLB is like a cache of the page table

it contains the most recently used virtual page numbers and their physical translations - saves the processor time since it doesn't have to go all the way to the page table in memory for translation

3. Compare and contrast TLB misses with page faults

a TLB miss just means the virtual page number you wanted was not in the TLB so you have to go to the page table to translate it

a page fault, on the other hand, means that the page you want is on disk and not in memory - this necessitates copying the entire page (often ~8kB) from disk to memory and is very time consuming

4. Give a reason why larger page sizes might be desirable

large page sizes are desirable because they help minimize page faults - fetching a large page on the first fault brings a large block of data into memory capturing a great deal of spatial locality

since disk access time has a great deal of overhead in it (seek time, rotational latency, controller overhead) it is more efficient to fetch one large page of data and hopefully not have any more page faults for a while than it is to fetch many small pag

5. Extra credit: can you page a page table? If your answer is yes, explain how you would do this. If your answer is no, explain why it is impossible.

yes, it is possible and it is sometimes used to minimize the space required for a page table. With modern 32 and 64 bit virtual addresses a naive implementation of a page table can take up a great deal of space in memory - one solution is to use a page table hierarchy - when you need to translate you look at the highest level page table which refers you to lower level page tables to complete translation - since each entry in the top level page table can point to a whole page table full of virtual addresses 1) the top level page table can be fairly small and 2) not all of the page tables that the top level table points to will be needed. This means that the lower level page tables not being used can be paged on disk. If we make the page tables exactly the size of one page then when a new page table is needed (a new subsection of the higher level page table) we can service it like a normal page fault by bringing it into memory and then use it to complete the translation of the virtual address. This kind of system gives huge memory usage savings over a naive implementation.

x3

QUESTION VI: Hazards and Optimization

1. Describe the following types of processor pipeline hazards:

structural hazard – A structural hazard occurs when an instruction cannot immediately proceed through the pipeline because of limited functional units within the processor. For instance, the ALU might be needed by different kinds of instructions at different points in their execution and several of these instructions cannot proceed simultaneously if there is only one ALU but they all need to use it at the same time.

data hazard – Data hazards happen because of dependences in program data accesses. Suppose one operation stores data in a register that is a source operand for the following instruction. Due to pipelining, the second instruction may have already read the data by the time the first instruction writes the new value to the register file. This situation will result in invalid program output and is an example of a data hazard. (unless the processor stalls) Alternately, data may be loaded from memory and not available for several cycles, a load-use data hazard

control hazard – Control hazards are situations when the processor cannot determine which instructions to introduce into the pipeline because the program flow (i.e. branch) depends on the outcome of a branch statement that hasn't cleared the pipeline yet. The processor may need to stop executing new instructions until the branch instruction is complete.

2. Structural, data, and control hazards typically require a processor pipeline to stall. Listed below are a series of optimization techniques implemented in a compiler or ~~processor pipeline~~ designed to reduce or eliminate stalls due to these hazards. For each of the following optimization techniques, state which pipeline hazards it addresses and how it addresses it. *Hint:* Some optimization techniques may address more than one hazard, so be sure to include explanations for all addressed hazards.

branch prediction –

Branch prediction addresses control hazards by guessing the outcome of a branch instruction and then speculatively executing instructions on one "side" of the branch to keep the pipeline moving. If the prediction proves to be correct when the branch completes, everything is fine. If it was incorrect, the processor must squelch the results of the instructions after the branch. Predictions can be made in hardware or by the compiler at run time (dynamically) or in advance.

instruction scheduling –

instruction scheduling can reduce structural hazards by having the compiler only issue sets of instructions that will not cause resource oversubscription during pipelined execution. A form of scheduling, filling delay slots, can help with control and data hazards (see below).

delay slots –

Delay slots address control and data hazards by dictating (in the ISA) that one or more instructions after a branch (control) or after a load (data) will not be affected by the outcome of the branch or load. The compiler can try to fill these slots with non-dependent instructions to keep the pipeline full.

predication –

Predication helps reduce control hazards by removing branches.

increasing available functional units (ALUs, adders, etc.) –

This generally helps avoid structural hazards caused by resource oversubscription (by adding more resources), but it can also be part of the implementation of other hazard remedies, like adding additional comparators and adders to make branch decisions earlier in the pipeline (to reduce control hazards), or do arithmetic earlier to reduce data hazards.

caches –

Caches can keep the pipeline moving by reducing the delay necessary between loading a value from memory and making use of it in another instruction. This lessens the cost of the load-use data hazard.

bypassing –

Bypassing helps avoid stalls due to data hazards by forwarding data that is about to be written to registers directly to another instruction later in the pipeline that is about to make use of that data.