

CS 473

Final Exam

Solutions

READ THESE INSTRUCTIONS

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following instructions. There will be a ten point deduction for failure to comply with them:

- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how answers were arrived
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 10:00 to finish the exam. The questions are equally weighted.

1. Floating point arithmetic:

- (a) Add the following pair of IEEE floating point numbers: 0xc0a80000 + 0x40900000

<pre> 1100 0000 1010 1000 - 0 1 10000001 010 1000 - 0 Sign: 1 Exponent: 10000001 Significand: 1.0101 </pre>	<pre> 0100 0000 1001 0-0 0 10000001 001 0-0 Sign: 0 Exponent: 10000001 Significand: 1.001 </pre>
---	--

Since the exponents are the same, there is no need to adjust either operand. Since one is positive and one is negative, we need to subtract:

```

  1.0101
- 1.001
-----
  0.0011
  
```

We have to renormalize; as the binary point has to move three places to the right, we have to subtract three from the exponent giving

```

Sign:          1
Exponent:    01111110
Significand:  1.1
  
```

```

1 01111110 1 0-0
1011 1111 01 0-0
  
```

bf400000

Penalty	Error
-3	Didn't renormalize
-2	Added instead of subtracting
-2	Denormalized operands; got it wrong.
-3	Forgot phantom bit
-2	Subtracted larger from smaller
-3	Subtracted exponents
-2	Upside down table for fraction conversion
-2	Renormalized wrong
-5	Didn't get final result

- (b) Convert your result into human-readable decimal.

Exponent: $01111110 - 01111111 = -1$. The significand takes the value $.11_2$. So, we get $-.75$

Penalty	Error
-7	Wrong, no work
-2	One division too many
-1	Forgot minus sign
-3	Converted to $-2.5 \times 1/8$

2. Branch Prediction

Suppose the following code is executed on a machine using two-bit branch prediction, with the predictors set to “weak taken”. What will the percentage of correct predictions be?

```

        addiu $3, $0, 10
        addiu $1, $0, 0
outer   addiu $2, $0, 0
inner   addiu $2, $2, 1
        bne  $2, $3, inner
        addiu $1, $1, 1
        bne  $1, $3, outer

```

The inner loop will predict correctly on every iteration but the last. On the first iteration the prediction will be changed from “weak taken” to “strong taken”; since the last iteration is only a single mispredict, it will only change the prediction from “strong taken” to “weak taken”, so the first iteration of the inner loop on the next iteration of the outer loop will predict correctly. Each iteration of the outer loop will require ten iterations of the inner loop; nine of these will be correct and one will be incorrect.

The outer loop will be executed ten times; nine of these will have a correct prediction and one will be incorrect.

This is a total of 110 branches; 99 of them are correct and 11 are incorrect. $99/110$, or 90%, of the predictions are correct.

Penalty	Error
-5	Thought all outer loop predictions would be wrong
-3	Forgot outer loop

3. Cache

- (a) A computer with a 32-bit address has a 256K, 8-way set-associative cache with an eight byte block size. What are the tag, index, and byte offset for address $0x87654321$?

The block size is 8 bytes, so the offset field is three bits. It's a 256K cache with an eight byte block, so it has $256K/8 = 32K$ blocks. It's 8-way set-associative, so it has $32K/8 = 4K$ sets; $4K$ sets implies a 12 bit index. The tag is $32 - 3 - 12 = 17$ bits.

17	12	3
10000111011001010	100001100100	001
10eca	864	1
Tag	Index	Offset

Penalty	Error
-3	Calculated associativity as size/sets, not blocks/sets
-5	Field width

- (b) A computer has a 7 bit index and 4 bit byte offset. If it uses a 32 bit address, what is the width of the tag field? If it has a 1MB cache, how set-associative is it?

The tag field will be $32 - 7 - 4 = 21$ bits.

11 bits are being used for index+offset, so the associativity must be $1M/2K = 512$. Alternatively, since the offset field is 4 bits, the block must be 16 bytes. There are $1M/16 = 64K$ blocks. As the index field is 7 bits there must be 128 sets; $64K/128 = 512$. A common mistake was to try to combine these two methods, calculating the associativity as $1M/128$. No....

Penalty	Error
-3	Calculated associativity as size/sets, not blocks/sets
-5	Associativity looks like a guess
-2	Forgot to raise associativity to power of 2

4. (20 points) Virtual Memory

Following are the PDBR and some memory addresses from an Intel. For each of the following attempts to access memory, state whether the result is a page fault, a protection violation, or a success. For a successful write, tell what physical address is written; for a successful read, tell what physical address is read and tell the data that is found there. All numbers are in hexadecimal.

<i>Penalty</i>	<i>Error</i>
-3	<i>Wrong operation</i>
-5	<i>Translation error</i>
-3	<i>Misread a bit</i>

- (a) Kernel mode read from 25cfa3e0

The directory entry is at 3a5c625c. and contains 5f238000. Since bit 0 is 0, we have a page fault (Present is checked before access rights, so we don't have a protection violation).

- (b) Kernel mode write to 1902af4c

The directory entry is at 3a5c6190 and contains 2b937003. Bit 0 is 1 so we have no page fault, and bit 1 is 1 so we have no protection violation (it's not user accessible, but we're in kernel mode). So we go on to the page table; the page table entry is at 2b9370a8 and contains 35b75001. Bit 0 is 1 again so we have no page fault, but this time it's not writeable (bit 1 is 0) so we get a protection violation.

- (c) User mode read from 235a851c

The directory entry is at 3a5c6234 and contains 72356007. No fault or protection violation, so we go on to the PTE at 723566a0. This contains 51bdd005; it is present and user-accessible so we succeed (if we were writing we'd have a protection failure, but we aren't writing). We read from 51bdd51c, and the data we get is 06afe284.

PDBR: 3a5c6000

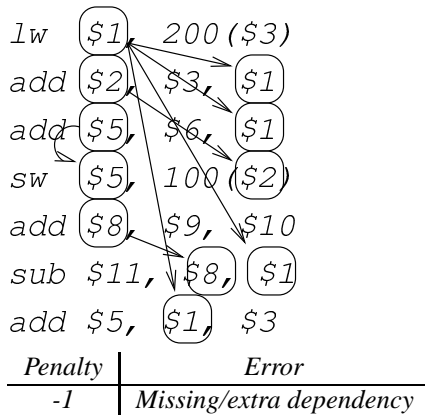
Address	Contents
2b9370a8	35b75001
35b75f4c	615ce4c6
3a5c6190	2b937003
3a5c6234	72356007
3a5c625c	5f238000
4949a3e0	5cced33f
51bdd51c	06afe284
5f2383e8	4949a000
723566a0	51bdd005

5. Superscalar Pipelining

Consider the following sequence of MIPS instructions:

```
lw    $1, 200($3)
add   $2, $3, $1
add   $5, $6, $1
sw    $5, 100($2)
add   $8, $9, $10
sub   $11, $8, $1
add   $5, $1, $3
```

- (a) Draw arrows showing all of the dependencies between the registers in instructions in this code. Note that if a register is set by an instruction, and then read by two different instructions, there are two dependencies.



I didn't ask for antidependencies, so I didn't worry about the antidependency between instructions 4 and 7.

- (b) Draw a Gantt chart showing how this code is executed in the superscalar pipeline from the web page. If you wish, you may use the following table for your response. Alternate lines of the table are labelled "Memory" and "ALU" respectively; instructions that take the Memory pipe should be in lines labelled "Memory", while instructions that take the ALU pipe should be in lines labeled "ALU". You won't need all the lines in the table.

Instruction	Pipe	
lw \$1, 200(\$3)	Mem	IF Q ID EX M WB
add \$2, \$3, \$1	ALU	IF Q ID EX WB
add \$5, \$6, \$1	Mem	IF Q ID EX M WB
sw \$5, 100(\$2)	Mem	IF Q ID EX M WB
add \$8, \$9, \$10	ALU	IF Q ID EX WB
sub \$11, \$8, \$1	Mem	IF Q ID EX M WB
add \$5, \$1, \$3	ALU	IF Q ID EX WB

<i>Penalty</i>	<i>Error</i>
-1	<i>Error in stall, instruction down wrong pipeline, etc.</i>
-2	<i>No Q pipeline stage at all</i>
-5	<i>Not superscalar</i>

While completely missing a pipeline stage probably warranted more points off, everybody who made that mistake also made enough other mistakes that they pegged out at five points off on the problem part (I didn't want to take off more than that).

- (c) Reorder the code to execute as quickly as possible and draw another Gantt chart showing how the modified code is executed. As before, you can use the following table for your response.

Instruction	Pipe	
<i>lw \$1, 200(\$3)</i>	<i>Mem</i>	<i>IF Q ID EX M WB</i>
<i>add \$8, \$9, \$10</i>	<i>ALU</i>	<i>IF Q ID EX WB</i>
<i>add \$2, \$3, \$1</i>	<i>Mem</i>	<i>IF Q ID EX M WB</i>
<i>add \$5, \$6, \$1</i>	<i>ALU</i>	<i>IF Q ID EX WB</i>
<i>sw \$5, 100(\$2)</i>	<i>Mem</i>	<i>IF Q ID EX M WB</i>
<i>sub \$11, \$8, \$1</i>	<i>ALU</i>	<i>IF Q ID EX WB</i>
<i>add \$5, \$1, \$3</i>	<i>Mem</i>	<i>IF Q ID EX M WB</i>

<i>Penalty</i>	<i>Error</i>
-1	<i>Error in stall, instruction down wrong pipeline, etc.</i>
-2	<i>Error in reordering breaking dependencies</i>