

**CS 473**  
**Final Exam**  
**May 6, 2005**

**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:  $0xc0a80000 + 0x40900000$
- (b) Convert your result into human-readable decimal.

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
```

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  $0x87654321$ ?
- (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?

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.

- (a) Kernel mode read from 25cfa3e0
- (b) Kernel mode write to 1902af4c
- (c) User mode read from 235a851c

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.
- (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 labelled “ALU”. You won’t need all the lines in the table.

Instruction	Pipeline	Cycles																			
	Memory																				
	ALU																				
	Memory																				
	ALU																				
	Memory																				
	ALU																				
	Memory																				
	ALU																				
	Memory																				
	ALU																				
	Memory																				
	ALU																				
	Memory																				
	ALU																				

- (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	Pipeline	Cycles																									
	Memory																										
	ALU																										
	Memory																										
	ALU																										
	Memory																										
	ALU																										
	Memory																										
	ALU																										
	Memory																										
	ALU																										
	Memory																										
	ALU																										