Summer 2016

Practice Final Exam for ECE/CS 250, ~~Fall 2014~~

name: _____

Duke students are bound by an academic integrity standard:

1. I will not lie, cheat, or steal in my academic endeavors, nor will I accept the actions of those who do.

2. I will conduct myself responsibly and honorably in all my activities as a Duke student.

Please sign your name below to acknowledge that you follow this standard:

(Changes from the original Fall 2014 exam are in this blue text)

1) [10 points] **Evolution of Computer Architectures**

[4] (a) List exactly two differences between a ~~current processor chip from Intel and the original Pentium processor chip (1993).~~ *MIPS as we studied it and Intel x86*

[4] (b) Microarchitectures for the same ISA change from generation to generation. List exactly two reasons why we don't just keep the same microarchitecture in every generation of processors that implements the same ISA.

[2] (c) If you learn everything there is to learn in ECE/CS 250 and the subsequent two Duke courses on computer architecture, will your computer architecture training cover all possible processor microarchitectures (or will you need to keep up with new developments in the field)? Circle one:

WILL KNOW EVERYTHING

WILL NEED TO KEEP UP WITH NEW DEVELOPMENTS IN FIELD

2) [10 points] **C Programming and 2s Complement**

[5] (a) You have a struct that contains two floats and one pointer to a char. You want to malloc 10 of these structs. Do not malloc the char itself (just malloc enough space for the pointer). Write the C code to declare the struct and then malloc 10 of them.

[5] (b) (a) Add the following base-10 numbers using **6-bit** 2s complement math: 4, -3. **Show your work!**

3) [15] **Assembly programming**

[8] (a) Fill in the last lines (all except the first three lines) of MIPS code. Follow the comments!

C code	**equivalent MIPS code**	**comment**
int a[30];	*(not shown)*	// assume &a[0] is in $4
int* p= a+5;	*(not shown)*	// assume p is in $5
int** q= &p;	*(not shown)*	// assume q is in $6
int* r = p - 2;		// put r into $7
int b = *p;		// put b into $8
int c = **q;		// put c into $9 // takes 2 instructions
if(c==b), then b=9		// put b into $10 // could be multiple instrs

[5] (b) Let's say procedure foo() calls procedure bar(). Why might bar() need to save $ra on the stack?

[2] (c) If we follow the MIPS calling convention, how does foo() pass an argument to bar() when it calls bar()?

Here's a cache. Fill in all the blanks.

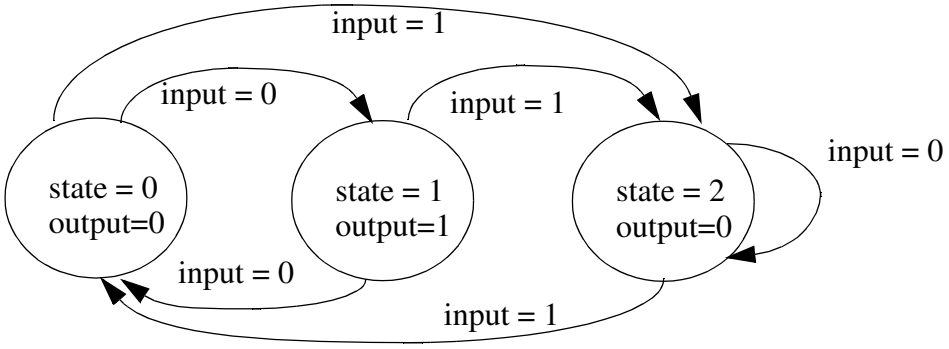
Term	Value
cache size	4096
block size	32
ways	2
frames	
sets	
bits:index	
bits:offset	
bits:tag	

addr-dec	addr-hex	tag	index	offset	result
38	0026				
30	001E				
62	003E				
5	0005				
2049	0801				
2085	0825				
60	003C				
4130	1022				
2085	0825				

^ Hit or miss.
For misses, indicate Cold, Capacity, or Conflict.

5) [15] **Logic and Finite State Machines**

Below is the state transition diagram for a FSM that a single 1-bit input and a single 1-bit output. In this FSM, **the output is strictly a function of the current state** (and NOT a function of the input). Write the truth table for the FSM, and write the boolean equation for the output in sum-of-products form.



6) [15] **Memory Hierarchy Performance**

You are deciding between Cache Hierarchy CA and Cache Hierarchy CB. Your only criteria are performance ~~and power.~~ Which do you choose and why? Show your math for performance ~~and your reasoning for power.~~

(we didn't discuss power impact, but suffice it to say that more capacity/associativity = more power draw)

	CA	**CB**
L1D size, set assoc.	64KB, 1-way set associative	64KB, 4-way set associative
L1 access latency (hit or miss)	1 cycle	2 cycles
L1 miss rate	10%	4%
L2 size, set assoc.	1 MB, 4-way set associative	2 MB, 8-way set associative
L2 access latency	10 cycles	15 cycles
L2 miss rate	10%	5%
main memory	200 cycles	200 cycles

7) [10] **Cache Operation**

[10] (a) Assume a 32-bit machine with a 2-way set-associative 4KB virtual cache that has 32-byte blocks. Assume the following sequence of load addresses, where each load accesses only one byte. Assume LRU replacement and assume all frames are initially invalid.

address	hit or miss?	set index (in base 10)?	block offset (in base 10)?
38			
30			
62			
5			
2049			
2085			
60			
4130			
61			

8) [15 points] **Virtual Memory.** Assume a 64-bit machine with a 16KB page size. Assume 4GB of physical memory. Write your answers in the form of 2^X rather than trying to figure out what that corresponds to. E.g., don't write out 2048 - write 2^{11} instead. Remember 2^{10}=1K, 2^{20}=1M, and 2^{30}=1G.

[2] (a) How many virtual pages are there per process?

[2] (b) How many bits are needed to represent the virtual page number?

[2] (c) How many bits are needed to represent the virtual page offset?

[2] (d) How many page table entries are there per process if we use a flat page table?

[2] (e) How many bits are needed to represent a physical address?

[2] (f) How many bits are needed to represent a physical page number?

[3] (g) We want to design a virtually-indexed, physically-tagged cache, that is 64KB and has 64-byte blocks. What is the minimum set-associativity for this cache that is required to allow us to use virtual indexing with physical tagging? Explain your answer.

9) [10 points] **I/O and GPUs**

[5] Explain howa core communicates with an I/O device using memory-mapped I/O.

Describe the difference between polling and interrupts.

What is DMA and how does it enhance throughput beyond having interrupts alone?

[5] Other than graphics, what kind of software/algorithms does a GPU perform well on? Explain your answer in terms of why the software would run well on a GPU's hardware.

We didn't talk much about GPUs.

Basically, they're extreeeeeeeeme multi-cores (hundreds of cores), where each core is much less powerful than a regular CPU. They're good on really parallel problems, such as "figure out the color of each pixel", as each pixel is independent. They're good on many non-graphics problems too, though, such as "hash a billion different inputs to try to find a match" (done in a brute force attack on hashed passwords as well as the bitcoin mining algorithm) and "compute the velocity and acceleration of all these simulated particles" (the "n-body problem").

10) [15] Data Dependences and Hazards

[10] (a) In the following code, label **all** RAW, WAR, and WAW dependences. Do not label RAR dependences. I've labeled one RAW dependence for you. You label the rest.

add $1, $2, $3

xor $2, $1, $2

lw $1, 4($2)

sub $2, $5, $1

sw $2, 4($5)

[5] (b) Which of the RAW dependences above would be data hazards in the single-cycle processor we learned about in class? Explain your answer.

Just to be clear, this bit is a trick question. A "single-cycle processor" doesn't do pipelining, so there *are* no data hazards. Basically, the key thing here is that dependency doesn't NECESSARILY imply a hazard; it depends on your CPU design with respect to instruction-level parallelism enhancing features such as pipelining.

11) **[10] Control Dependences and Hazards**

[5] (a) Why do control hazards caused by conditional branches hurt the performance of a pipelined processor?

[5] (b) We talked about fast branches, delayed branches, and dynamic branch prediction as three techniques to help reduce the performance loss due to control dependences. Explain **exactly one** of them and its benefits and limitations.

12) [10] **Pipelining**

Assume the 5-stage pipeline we've used in class (F, D, X, M, W). Assume the pipeline forwards/bypasses operands whenever possible and stalls only when needed to satisfy a RAW dependence that can't be forwarded/bypassed. Assume all loads and stores hit in the 1-cycle data cache. Assume the following code:

```
add $1, $2, $3
xor $2, $1, $2
lw $1, 4($2)
sub $2, $5, $1
sw $2, 4($5)
```

Fill in the pipeline diagram for the first 7 cycles, and assume that the add is in the Fetch (F) stage in cycle 1, as drawn.

	1	**2**	**3**	**4**	**5**	**6**	**7**
add $1, $2, $3	F						
xor $2, $1, $2							
lw $1, 4($2)							
sub $2, $5, $1							
sw $2, 4($5)							

13) [10 points] **Multicore and Niagara**

The Niagara chip was designed to exploit thread-level parallelism (TLP) instead of instruction-level parallelism (ILP).

[5] (a) Give an example of a software workload that has a lot of TLP and explain why this workload has a lot of TLP.

[5] (b) Why would Niagara get poor performance on a single-threaded software workload (compared to a high-performance core from Intel or AMD)? Explain your answer.

We aren't going to cover Niagra. Instead, we're hitting Intel x86, which was asked about in Q1.

However, the exam itself gives away the answer: the Niagara optimizes for thread-level parallelism (multithread apps), so if you give it a single sequential app, most of the silicon is going to sit idle... there's fewer features focused on exploiting parallelism inside of a single thread.