EECS 470 *Final Exam*

Winter 2018

Name: _____ unique name: _____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

Scores:

NOTES:

- Open book and Open notes
- Calculators are allowed, but no PDAs, Portables, Cell phones, etc.
- Don't spend too much time on any one problem.
- You have about 120 minutes for the exam.
- There are ***10*** pages including this one.
- Do not write on the back of any pages.
- **Be sure to show work and explain what you've done when asked to do so.**
- **The last page has two "answer areas" for the last question. Clearly mark which one you want graded or we will grade the first one.**

1. Multiple choice questions

Fill-in-the-blank or circle the best answer. *Circle the best answer.*
[12 points, -2 per wrong/blank, minimum 0]

a. For a high degree of sequential data accesses, what would you choose for a higher hit rate, given a fixed size cache?

Small block size **Large block size** **It makes no difference**

b. For random data accesses, what would you choose for a higher hit rate, given a fixed size cache?

Small block size **Large block size** **It makes no difference**

c. You have a program that runs for 10 seconds. The processor uses 100 Watts during that time. If we drop program's run time to 8 seconds, about what would you expect the power used to become? Assume a cubic relationship between power and performance.

50 Watts **70 Watts** **90 Watts** **130 Watts** **160 Watts** **200 Watts**

d. In the MESI protocol as taught in class, which of the following transitions might happen for a given address without the address of the block in question being on the bus as a result or cause of this transition?

S→ M **M→ I** **S → E** **E→ M** **E→S**

e. You would expect a 1024 byte fully-associative cache with a block size of 32 bytes to get about what hit rate on a memory access with a stack distance of 10?

0% **12%** **40%** **74%** **92%** **100%**

f. In the R10K scheme, if you have a ROB size of 48, RS size of 8, and ARF size of 32, what is the maximum number of PRF entries you would expect to have?

40 **48** **50** **56** **70** **80**

g. Assume a memory access to main memory on a cache miss takes 20 ns and a memory access to the cache on a cache hit takes 2 ns. If 75% of the processor's memory requests result in a cache hit, about what is the average memory access time?

22ns **18ns** **16ns** **10ns** **6ns** **2ns**

h. If you have a 2 KB, two-way associative cache with 32-byte lines on a computer with a 20-bit address space, you would need about how many bits to store all the tags in the cache?

640 **1024** **2048** **6400** **20,480** **21,480**

2. Predicated instructions

The ARMv7 and IA64 instruction set architectures both use predicated instructions. That is, instructions such as:

```
(p1) DADD R1,R2,R3 ; if (p1) then R1=R2+R3
```

Answer the following questions about predicated instructions (*continued on the next page*). **[16 points]**

a) What is the primary advantage of a predicated instruction set? Give an assembly example of when it can be useful compared to a non-predicated instruction and explain why it's useful. **[5]**

b) What is the primary *disadvantage* of a predicted instruction set? Use an example to illustrate your point. **[4]**

c) While most ISAs don't have predicated instructions, most do support some type of CMOV instruction. Explain how CMOV gets some of the benefits of predicated instructions while greatly reducing the disadvantages. **[4]**

d) Give an example of where a predicated instruction could be easily used, but where replacing it with a CMOV would be a poor idea. *Briefly* explain why it would be hard to replace. **[3]**

3. Stack it up.

Write a Verilog module that implements a 4 entry **stack**. A stack is a Last-In First-Out buffer that outputs the latest value written (pushed) into it. The stack should also support a pop function, such that upon a pop the last entry written will be cleared in the next cycle. A pop will not affect the output of the stack in the same cycle (the stack will always output the top of stack). You may assume that a pop will never happen when the stack is empty, a push will never happen when the stack is full, and a push and a pop will never happen in the same cycle. For full credit, your code must be synthesizable and reasonably efficient. *Put your answer on the next page.* **[16 points]**

```
module Stack (
      input                clock, reset,
      input [31:0]         data_i,         // data to push if any
      input                valid_i,        // we are pushing data
      input                pop,            // remove top of stack
      output logic         data_valid_o,   // the stack isn't empty
      output logic [31:0]  data_o          // data at the top of stack if any
);
      logic [3:0][31:0]    stack, next_stack;
      logic [3:0]          valid, next_valid;
      logic [1:0]          stack_ptr, next_stack_ptr;
////////////////ANSWER BEGIN

////////////////ANSWER END
      always_ff @(posedge clock) begin
            if (reset) begin
                  stack       <=    0;
                  valid       <=    0;
                  stack_ptr   <=    0;
            end else begin
                  stack       <=    next_stack;
                  stack_ptr   <=    next_stack_ptr;
                  valid       <=    next_valid;
            end
      end
endmodule
```

```
module Stack (
      input                clock, reset,
      input [31:0]         data_i,            // data to push if any
      input                valid_i,           // we are pushing data
      input                pop,               // remove top of stack
      output logic         data_valid_o,      // the stack isn't empty
      output logic [31:0]  data_o             // data at the top of stack if any
);
      logic [3:0][31:0]    stack, next_stack;
      logic [3:0]          valid, next_valid;
      logic [1:0]          stack_ptr, next_stack_ptr;
////////////////ANSWER BEGIN
```

```
////////////////ANSWER END
      always_ff @(posedge clock) begin
            if (reset) begin
                  stack        <=     0;
                  valid        <=     0;
                  stack_ptr    <=     0;
            end else begin
                  stack        <=     next_stack;
                  stack_ptr    <=     next_stack_ptr;
                  valid        <=     next_valid;
            end
      end
endmodule
```

4. Cache bandwidth

You are working on a processor capable of using a write-back or write-through scheme. It is always allocate-on-write. The block size is 64 bytes. All loads and stores are to 8 byte locations. The bus supports both 8 and 64 byte transactions.

[12 points]

a. If your processor is generating one billion stores a second and the cache has a hit rate of X on stores, what is the write bandwidth (in bytes/second) you would expect to associated with each of the two write schemes in steady-state? **[6]**

   Write-through:

   Write back:

b. Assuming you are only worried about minimizing write *bandwidth* (bytes of data written per second), for what values of X would you be best off using a write-through cache rather than write-back? Be sure to specify a range. **[3]**

c. Assuming you are only worried about minimizing the *number of writes*, for what values of X would you be best off using a write-through cache rather than write-back? Be sure to specify a range. Explain your answer. **[3]**

Aside: sometimes you will be limited by data bandwidth, sometimes by address bandwidth (which is basically the number of transactions). Depends on your bus and a few other things…

5. A fairly clean MESI problem

Consider a case of having 2 processors using a snoopy MESI protocol where the memories can snarf data. Both have a 2-line direct-mapped cache with ***each line consisting of 16 bytes***. The caches begin with all lines marked as invalid. Fill in the following tables indicating:

- If the processor gets a hit or a miss in its cache.
- What bus transaction(s) (if any) the processor performs (BRL, BWL, BRIL, BIL)
- If a HIT or HITM (or nothing) occurs on the bus during snoop.
- For misses only, indicate if the miss is compulsory, capacity, conflict, or coherence. A coherence miss is one where there would have been a hit, had some other processor not interfered.

Finally, indicate the state of the processor after all of these memory operations have completed. The operations occur in the order shown. **[15 points, -0.5 per wrong or blank, minimum of 0]**

Processor	Address	Read/Write	Cache Hit/Miss	Bus transaction(s)	HIT/ HITM	"4C" miss type (if any)
1	0x001	Write				
1	0x100	Read				
1	0x108	Write				
1	0x111	Read				
1	0x00F	Write				
2	0x100	Read				
2	0x11A	Read				
2	0x00F	Write				
1	0x11C	Write				
1	0x00F	Read				

Final state:

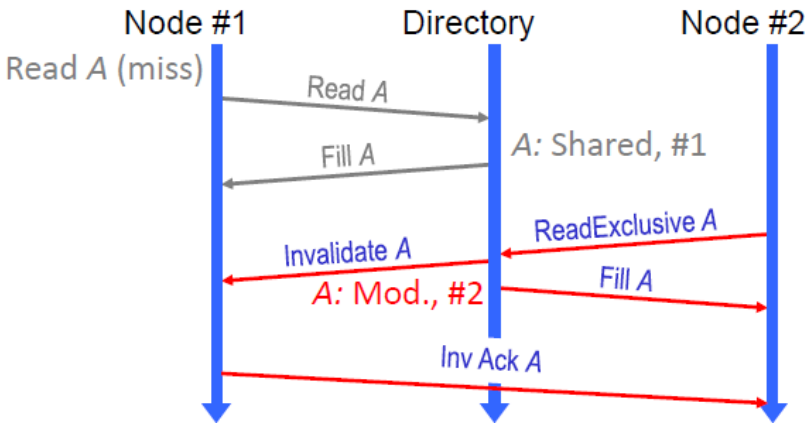
	Proc 1	
	**Tag . **	**State**
Set 0		
Set 1		

	Proc 2	
	**Tag . **	**State**
Set 0		
Set 1		

6. Misc. questions on multi-processor systems

Answer the following questions **[14 points]**

a. The diagram to the right describes an example in the context of a directory-based multi-processor system. Circle the correct answers for the following questions. **[7]**



- Node 1 is initiating a ***BRL / BRIL / BWL / BIL*** bus transaction
- Node 2 is initiating a ***BRL / BRIL / BWL / BIL*** bus transaction
- Why does one bus request from Node #2 result in both a "Fill A" and an "Inv Ack A"? Explain, in your own words, what each of those two arrows are describing.

b. Let's say that you find that occasionally cosmic rays strike the MESI state storage in your bus-based coherence modules, causing a state to instantaneously change to another.

Fill in a cell in the table below with a tick (☑) if, for a starting MESI state on the top, instantaneously changing the state to the state on the left affects neither correctness nor performance. Fill in a cell in the table with a circle (○) if correctness is not affected but performance could be affected by the state change. If correctness may be affected, fill in a cross (☒). **[7]**

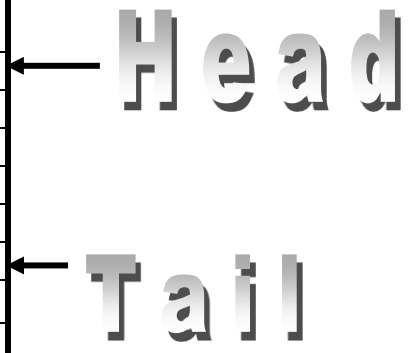
Ending State (Current Status)	Starting State (Real Status): M	E	S	I
M				
E				
S				
I				

7. Little boxes

Consider the following tables that represent the state of a processor that implements what we have called the P6 scheme:

RAT	
Arch Reg. #	**ROB# (-- if in ARF)**
0	--
1	2
2	4
3	--
4	1
5	--

ROB				
Buffer Number	**PC**	**Done with EX?**	**Dest. Arch Reg #**	**Value**
0	20	N	1	--
1	24	N	4	--
2	28	Y	1	6
3	32	Y	--	--
4	36	Y	2	8
5				
6				
7				
8				



RS						
RS#	**Op type**	**Op1 ready?**	**Op1 RoB/value**	**Op2 ready?**	**Op2 RoB/value**	**Dest ROB**
0	Add	Y	4	Y	5	0
1	Mult	N	0	Y	3	1
2						
3						
4						

ARF	Reg#	0	1	2	3	4	5
	Value	1	5	4	3	2	1

The instruction at PC 32 is a branch that has been predicted not-taken, but it is actually taken. The destination of the branch is PC 200, where the following code resides:

```
R3=R3+R1          // A
R1=R1+R3          // B
R5=R5+R0          // C
R1=R2*R5          // D
```

Show the state of the above tables if instruction A has retired, instruction B has been issued but has not finished execution, while C and D have progressed as far along as possible. ***Be sure to label the head and tail of the ROB***. Please place instruction A in slot 5 of the ROB. When other arbitrary decisions need to be made, you are to just make them. **[15 points]**

(A second copy is available on the following page, ***please cross out the one you don't want graded!***)

(This is a copy of the state shown on the previous page. ***Please cross out the one you don't want graded!***)

RAT	
Arch Reg. #	**ROB# (-- if in ARF)**
0	--
1	2
2	4
3	--
4	1
5	--

ROB					
	Buffer Number	**PC**	**Done with EX?**	**Dest. Arch Reg #**	**Value**
← Head	0	20	N	1	--
	1	24	N	4	--
	2	28	Y	1	6
	3	32	Y	--	--
	4	36	Y	2	8
← Tail	5				
	6				
	7				
	8				

RS						
RS#	**Op type**	**Op1 ready?**	**Op1 RoB/value**	**Op2 ready?**	**Op2 RoB/value**	**Dest ROB**
0	Add	Y	4	Y	5	0
1	Mult	N	0	Y	3	1
2						
3						
4						

ARF							
	Reg#	**0**	**1**	**2**	**3**	**4**	**5**
	Value	1	5	4	3	2	1

The instruction at PC 32 is a branch that has been predicted not-taken, but it is actually taken. The destination of the branch is PC 200, where the following code resides:

```
R3=R3+R1              // A
R1=R1+R3              // B
R5=R5+R0              // C
R1=R2*R5              // D
```

Show the state of the above tables if instruction A has retired, instruction B has been issued but has not finished execution, while C and D have progressed as far along as possible. ***Be sure to label the head and tail of the ROB***. Please place instruction A in slot 5 of the ROB. When other arbitrary decisions need to be made, you are to just make them.