EECS 470 W18 Final Exam Answer Key

Answers in red. Notes/Explanations in blue.

1. Multiple Choice Questions [12 points, -2 per wrong/blank, minimum 0]

Circle the best answer.

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**

See F21 Q3 for a demonstration of this

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**

With a random-access pattern, every address is equally likely to be accessed regardless of prior accesses. Therefore, the only factor that can change the hit rate is the cache size.

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**

The cubic relationship between power and performance can be modeled as $P \propto (1/t)^3$, where P is power, and t is time. Let P1 and t1 be the original time and power and P2 and t2 be the new time and power. Using the proportionality:
$(P2 / P1) = (t1 / t2)^3$
$(P2 / 100\ \text{Watts}) = (10\ \text{s} / 8\ \text{s})^3$

Solving for P2, we get that P2 = 195 Watts, about 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**

A cache line can silently upgrade from the exclusive to modified state.

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%**

There are (1024 bytes) / (32 bytes per cache block) = 32 cache blocks.

Stack distance refers to the number of unique memory references made between two successive accesses to the same memory location. A stack distance of 10 means that there are 10 unique blocks accessed between accesses to the same block.

Given the fully associative cache can hold 32 blocks, the 10 unique accesses between subsequent accesses all fit within the cache's capacity. Since the memory block being accessed won't be evicted after the 10 unique accesses, the hit rate will be about 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**

Due to our ROB size, we can have a maximum of 48 instructions in-flight, each requiring a destination physical register. Due to the ARF size, we need 32 physical registers to maintain the current architected state. So, we need a total of 48 + 32 = 80 physical registers.

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**

Simultaneous cache and memory access:
AMAT = 20ns * (25% cache miss) + 2ns * (75% cache hit) = 6.5ns

Cache and then memory access:
AMAT = 2ns * (25% cache miss * 20ns) = 7ns

The closest answer under both interpretations is 6ns.

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**

Set size = 2 blocks per set * 32 bytes per block = 64 bytes

Number of sets = cache size / set size = 2 KB / 64 bytes = 32 sets

Set index bits = log2(number of sets) = log2(32 sets) = 5 bits

Block offset bits = log2(block size) = log2(32 bytes) = 5 bits

Tag bits = Address bits – set index bits – block offset bits = 20 – 5 – 5 = 10 bits

Total tag store = number of cache blocks * tag bits = 2 KB / (32 bytes per block) * 10 bits = 640 bits

2. Predicated Instructions [16 points]

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.

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

A predicated instruction set allows for conditional execution without explicit branching, avoiding the performance penalties of frequent branch misprediction.

```
if (R4 == 0) goto SKIP
     ADD  R1,  R2,  R3
SKIP:
```

The above assembly can be rewritten with predicated instructions to avoid branches

```
P1 = R4 == 0
DADD R1,  R2,  R3
```

b. What is the primary *disadvantage* of a predicated instruction set? Use an example to illustrate your point.

A predicated instruction set requires additional register encoding overhead, reducing the bit space available for other instruction fields.

For example, consider an ISA with fix-length 32-bit instructions, 32 general-purpose registers, and 32 predicate registers. A predicated add instruction requires 5 bits (destination register) + 10 bits (source registers) + 5 bits (predicate register) = 20 bits, whereas a non-predicated add instruction only requires 15 bits for register encoding. The predicated instruction set leaves only 12 bits for other instruction fields.

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.

CMOV still allows a processor to effectively achieve conditional execution without explicit branching. However, unlike predicated instructions, CMOV does not require the additional bits in the instruction encoding.

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.

CMOV can increase total instruction count and size of dependency chains. This is especially detrimental in loops. Additionally, CMOV often requires using temporary registers, increasing register pressure.

Small Predicated example:

```
P1 = R4 == 0
DADD R1,  R2,  R3
```

Small CMOV alternative:

```
Set flags for R4 == 0
ADD  R5,  R2,  R3
CMOV R1,  R5
```

3. Stack It Up [16 points]

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.

(One of many solutions)

```
module Stack (
      input                   clock, reset,
      input [31:0]            data_i,
      input                   valid_i,
      input                   pop,
      output logic            data_valid_o,
      output logic [31:0]     data_o
);
      logic [3:0][31:0] stack, next_stack;
      logic [3:0]       valid, next_valid;
      logic [1:0]       stack_ptr, next_stack_ptr;

      assign data_o = stack[stack_ptr];
      assign data_valid_o = valid;
      assign next_valid = next_stack_ptr != '0;

      always_comb begin
            next_stack_ptr = stack_ptr;
            next_stack = stack;

            if (valid_i) begin
                  next_stack_ptr = stack_ptr + 1;
                  next_stack[stack_ptr] = data_i;
            end else if (pop) begin
                  next_stack_ptr = stack_ptr - 1;
            end
      end

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

4. Cache Bandwidth [12 points]

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.

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?

Write-through: 8 billion bytes/second
All stores are written to memory under write-through. 1 billion stores/second * 8 bytes/store = 8 billion stores/second

Write back: 64 * (1-X) * 1 billion bytes/second
With a X hit rate, we have a (1-X) miss rate. For every miss, there must be an eviction to free a cache line. Therefore, there are (1-X) * 1 billion evictions/second.
Since the processor is only generating stores, we know that all evictions are dirty evictions. A dirty eviction results in a full cache block (64 bytes) being written back to memory. So, the write bandwidth is: 64 bytes/eviction * (1-X) * 1 billion evictions/second = 64 * (1-X) * 1 billion bytes/second

b. Assuming you are only worried about minimizing the 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 a write-back? Be sure to specify a range.

   For X < 0.875, a write-through cache is better at minimizing write bandwidth.

   Solving for the value of X such that the write-back bandwidth equals the write-through bandwidth:
   64 * (1-X) * 1 billion bytes/second = 8 billion bytes/second
   1-X = 0.125
   X = 0.875

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.

   Write-through caching always results in the same number or more total writes compared to write-back caching. In write-through, every store operation immediately writes data to memory. On the other hand, with write-back caching, store operations that hit in the cache are grouped together into a single memory write, leading to fewer memory writes overall.

5. A fairly clean MESI problem [15 points, -0.5 per wrong or blank, minimum 0]

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 cache begins with all lines marked as invalid. Fill in the following tables indicating:

- If the processor gets a hit or 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.

Processor	Address	Read/Write	Cache Hit/Miss	Bus Transaction(s)	HIT/HITM	"4C" miss type (if any)
1	0x001	Write	Miss	BRIL	-	Compulsory
1	0x100	Read	Miss	BRL/BWL	-	Compulsory
1	0x108	Write	Hit	-	-	-
1	0x111	Read	Miss	BRL	-	Compulsory
1	0x00F	Write	Miss	BRIL/BWL	-	Capacity
2	0x100	Read	Miss	BRL	-	Compulsory
2	0x11A	Read	Miss	BRL	HIT	Compulsory
2	0x00F	Write	Miss	BRIL	HITM	Compulsory
1	0x11C	Write	Miss	BIL	HIT	Coherence
1	0x00F	Read	Miss	BRL	HITM	Coherence

Final state:

Proc 1:

	Tag	State
Set 0	0x00	S
Set 1	0x08	M

Proc 2:

	Tag	State
Set 0	0x00	S
Set 1		I

If you included the set index in the "tag" field of the table, you should have got the following:

Final state:

Proc 1:

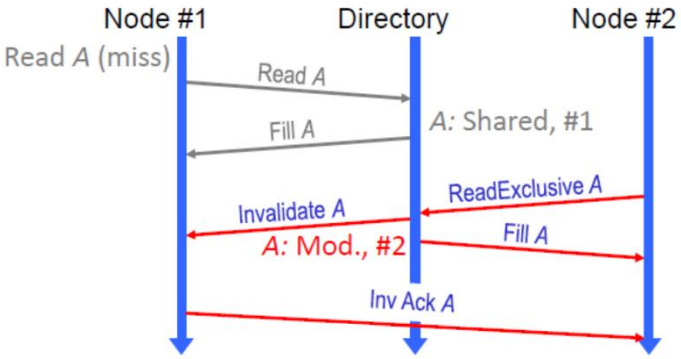
	Tag	State
Set 0	0x00	S
Set 1	0x11	M

Proc 2:

	Tag	State
Set 0	0x00	S
Set 1		I

6. Misc. questions on multi-processor systems [14 points]

Answer the following questions.



a. The diagram describes an example in the context of a directory-based multi-processor system. Circle the correct answers for the following questions.
   - 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.

     The BRIL request results in a data fulfillment response ("Fill A") and an acknowledgement from other nodes in the system that they have invalidated their copies of A. The "Inv Ack" from Node 1 is necessary to ensure that a reader of A and a writer to A do not exist simultaneously, potentially causing incoherence.

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 transition 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.

	M	**E**	**S**	**I**
M	✓	O	X	X
E	X	✓	X	X
S	X	O	✓	X
I	X	O	O	✓

M → (E, S): Dirty cache line will not be written back to memory on eviction
M → I: Dirty cache line is lost

E → M: Performance decrease due to unnecessary write of clean data on eviction
E → S: Performance decrease due to the now required BIL for upgrade to modified
E → I: Performance decrease since data must read from memory again

S → M: Processor can modify cache line without first invalidating potential sharers
S → E: Processor can silently upgrade to modified without first invalidating potential sharers
S → I: Performance decrease since data must be read from memory/other caches again

I → M: Random data in cache line will get written back to memory
I → (E, S): Random data in cache line will be read

7. Little Boxes [15 points]

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	8
2	--
3	--
4	--
5	7

ROB					
Buffer Number	**PC**	**Done with EX?**	**Dest Arch Reg #**	**Value**	**Head/Tail**
0	~~20~~	~~Y~~	~~1~~	~~9~~	T
1	~~24~~	~~Y~~	~~4~~	~~27~~	
2	~~28~~	~~Y~~	~~1~~	~~6~~	
3	~~32~~	~~Y~~	~~-~~	~~--~~	
4	~~36~~	~~Y~~	~~2~~	~~8~~	
5	~~200~~	~~Y~~	~~3~~	~~9~~	
6	204	N	1	--	H
7	208	Y	5	2	
8	212	Y	1	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	6	4	9	27	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.