EECS 470 *Final Exam*

Fall 2006

Name: ___________________________________ unique name: _____________

Sign the honor code:

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

____

Scores:

Page #	Points
2	/8
3	/12
4/5	/26
6	/12
7	/12
8	/10
9	/20
Total	**/100**

NOTES:

1. Open book and Open notes
2. There are 10 pages total. Count them. The last page is an extra copy of the last question.
3. Calculators are allowed, but no PDAs, Portables, Cell phones, etc.
4. Don't spend too much time on any one problem.
5. You have about 120 minutes for the exam.
6. Some questions may be more difficult than others. You may want to skip around.
7. **Be sure to show work and explain what you've done when asked to do so.**

Section I – Short answer/fill-in-the-blank/multiple choice.

1. Use the CMOV instruction to remove one of the three branches (leaving only two). You may rewrite the code as needed but it needs to do the same thing it was doing initially and your code should not fault on a data load when this code wouldn't fault. Of the 32-registers, R0-R25 are all live after this code segment and R20-R25 all contain addresses of data found in the data cache. Your CMOVs and compares should be of the following form: **[8]**

- `Rx=(Ry>C)  ; Rx=1 if Ry > C, otherwise Rx=0.  C is an`
  `            ; immediate and the > can be replaced by == or <.`
- `CMOV(Rx, Ry←Rz) ; If Rx!=0, Ry=Rz, else noop.`
- `CMOV(!Rx, Ry←Rz) ; If Rx==0, Ry=Rz, else noop.`

```
top:
     R3=MEM[R2]
     R4=MEM[R2+0]
     R2=R2+4
     R5=R5+R4
     If(R5<0) GOTO done
     R5=R5+R6
     If(R3==0) GOTO top
     R2=MEM[R3+0]
     GOTO top
done:
     R5=R6+1
     ...
```

2. Consider a standard 5-stage pipeline where:
    - Branches are resolved in the EX stage
    - All branches are predicted not taken
    - All data hazards which can be dealt with by forwarding to the EX stage are implemented (and no others)
    - The CPI would be 1.0 if it were not for control or data hazards

    The program running on the machine has the following characteristics:
    - 10% of all instructions are branches.
    - 30% of all branches are taken
    - 25% of all instructions are loads
    - 20% of all instructions are dependent on the instruction immediately before them.
    - 15% of all instructions are stores
    - The program is quite long (millions of instructions)

    What is the CPI that would be achieved by the machine on the program described? Show your work. **[6]**

3. Consider a processor which has the following characteristics: 80W, 1 BIPS, 1.5V, 1GHz Now say someone has proposed an architectural optimization that will reduce the total processor power to 70W but will reduce the performance to 950 MIPS. How well will this optimization compete with the power savings that is likely available if we just reduce the voltage and frequency to get the same power reduction? Is the architectural optimization better or worse than voltage/frequency scaling? You need to *clearly* show your work. **[6]**

4. Say you have a processor with no cache and you then add a no-write allocate, write-back cache. Explain why adding the cache might *increase* the number of bytes of data *written* to memory compared to the no-cache case. ***Provide a detailed example of when this might happen***. **[5]**

5. Circle the correct answer or fill-in-the-blank. **[21 points, -2 for each wrong or blank answer, minimum of 0]**

a) A victim cache is more helpful when associated with a **direct-mapped cache / fully-associative** cache than a two-way associative cache because the two-way associative cache is less prone to misses of data with **high / low stack** distances.

b) Given a 64-bit address space, a 32 KB cache with 16 byte lines that is two-way set-associative will have _________ index bits and the total size of the tag store (not including things like valid bits, MESI bits, etc.) will be ___________ bytes.

c) Page coloring can hurt over-all system performance because you expect to get (a few) additional **page faults / cache misses / segmentation faults / BTB misses**.

d) In the IA-64 assembly language, instructions are placed into **bundles / groups/ skebers** in units of three, while **bundle/group/skeber** sizes are based on the **ISA / instruction dependencies / branch mis-prediction rates**.

(Continued on the next page)

e) As wires get narrower their resistance goes **up / down / stays the same**. As transistors get smaller, their switching time generally goes **up/down/stays the same**.

f) As you increase size of the RoB, you expect the IPC to go **up / down**, while you expect the clock period to go **up / down**.

g) A return address stack is a type of branch predictor that will generally help the prediction of **forward / backward / direct / indirect** branches in the Alpha ISA.

h) For a given frequency, dynamic power consumption is generally proportional to the **voltage / switching rate / circuit capacitance squared**, while static power consumption is \ generally proportional to the **voltage / switching rate / circuit capacitance squared**.

i) Test-and-set is a(n) ______________ operation, meaning that the test (read) and set (write) happen back-to-back without any operation occurring between the read and write.

j) In IA-64, when you hoist a load above a branch you generally will need to use a(n) **speculative / loop-carry / advanced** load.

k) IA-64 takes advantage of predication and rotating registers to often negate the need for **a prolog / unrolling / an advanced load** when doing software pipelining

l) Static reordering generally allows for a **larger / smaller** window of instructions to be considered for reordering.

m) Say you have a virtually indexed, physically tagged cache of size 64KB. The cache has 32-byte lines and is 4-way set associative. The largest the page size could be is **4KB / 16KB / 64KB / 256KB**

n) In a system with a 64-bit address space, a 3 MB, 6-way set associative write-back cache will have ________ bits for its tag.

Section II: Longer answer

1. Consider a case of having 3 processors using a snoopy MESI protocol where the memory can snarf data. All three 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 table indicating
   - If the processor gets a hit or a miss in its cache
     If a HIT or HITM (or nothing) occurs on the bus during snoop.
   - What bus transaction(s) (if any) the processor performs (BRL, BWL, BRIL, BIL)

   In the event more than one bus transaction occurs due to a given memory read/write indicate the response for each bus transaction. Finally, indicate the state of the processor after all of these memory operations have completed. The operations occur in the order shown. **[12 points, -1 per wrong or blank, minimum of zero]**

Processor	**Address**	**Read/ Write**	**Bus transaction(s)**	**Hit/Miss**	**HIT/ HITM**
1	0x100	Read			
1	0x100	Write			
2	0x100	Write			
3	0x110	Write			
1	0x110	Read			
1	0x200	Write			
1	0x200	Read			
2	0x100	Read			
3	0x100	Write			

	Proc 1	
	Address	**State**
Set 0		
Set 1		

	Proc 2	
	Address	**State**
Set 0		
Set 1		

	Proc 3	
	Address	**State**
Set 0		
Set 1		

2. Consider two different caches:
   - **Cache “A”** has 8 lines in its cache, each line is 16 bytes, and the cache is two-way *set* associative.
   - **Cache “B”** has 8 lines in its cache, each line is 16 bytes, and the cache is two-way *skew* associative.

   Both caches start out with all blocks invalid. Both caches use a true LRU replacement policy. Both caches will place data into way 0 if both options are invalid. Both are write-allocate and write-back. The skew cache uses the lowest-order available address bits (not used for offset) as its index for way 0. The next lowest-order available bits (not used for offset or the way 0 index) are used for the index for way 1.

   The following address stream occurs. Assume all memory requests are for a single byte of data.

   0x123, 0x100, 0x007, 0x102, 0x025, 0x004, 0x127, 0x081

   For the address field, list the *lowest* address of the line. So if the line consisting of the data 0x010-0x01F where in a given block, the address listed should be “0x010”. Blank entries will be assumed to be invalid. **[10]**

Cache “A”				
Way 0			**Way 1**	
Index	Address		Index	Address
0			0	
1			1	
2			2	
3			3	

Cache “B”				
Way 0			**Way 1**	
Index	Address		Index	Address
0			0	
1			1	
2			2	
3			3	

What is the hit rate of each cache? **[2]**

3. The following Verilog code as a problem. It works under simulation, but it doesn't appear to work after synthesis. You think the problem is that amount of logic in yor always @(posedge clock) block. Minimize the amount of logic you have in that block to a single conditional of no more than one variable. In addition make it so the right-hand side of all code in that block is a single variable (wire or reg). Be careful to declare regs/wires needed. **[10]**

```
wire inc_a, dec_a;
wire reset, squash, die;
wire [2:0] op;
reg [31:0] a;
wire [31:0] b;
wire [31:0] c;
reg [31:0] r;


always @(posedge clock)
begin
    if (reset | squash | die)
    begin
       a <= `SD 0;
       r <= `SD 0;
    end
    else
    begin
        if (inc_a)
           a <= `SD a + 1;
        else if (dec_a)
           a <= `SD a - 2;

        case (op)
          `ADD: r <= `SD b + c;
          `OR:  r <= `SD b | c;
           default:
                r <= `SD r;
        endcase
   end
end
```

Section III – Ye Olde Tomasulo's Question

Consider the following tables that represent the state of a processor that implements what we have called Tomasulo's second algorithm:

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

ROB					
Buffer Number	**PC**	**Done with EX?**	**Dest. Arch Reg #**	**Value**	
0	40	N	0	---	← Head
1	44	Y	2	-5	
2	48	Y	--	---	
3	52	Y	4	4	
4	56	N	0	---	← Tail
5					
6					
7					

RS						
RS#	**Op type**	**Op1 ready?**	**Op1 RoB/value**	**Op2 ready?**	**Op2 RoB/value**	**Dest ROB**
0	Mult	Y	21	Y	3	0
1	Add	N	0	N	0	4
2						
3						
4						

ARF	Reg#	0	1	2	3	4	5
	Value	1	3	6	10	15	21

The instruction at PC 48 is a branch that has been predicted not-taken, but it is actually taken. The destination of the branch is PC 100, where the following code resides:

```
R0=R0+R1            // A
R1=R3*R0            // B
R2=R4+R0            // C
R3=R2-R3            // D
R2=R2*R4            // E
```

Show the state of the above tables if instruction A has retired, inst B has not finished executing, while C, D and E have progressed as far along as possible. *Be sure to label the head and tail of the ROB.* You are to place instruction A in the RoB in the same place it would have been placed had the prediction been initially correct. When arbitrary decisions need to be made, you are to just make them. **[20]**

(A second copy is available on the following page)

A copy of the problem…

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

ROB					
Buffer Number	**PC**	**Done with EX?**	**Dest. Arch Reg #**	**Value**	
0	40	N	0	---	←Head
1	44	Y	2	-5	
2	48	Y	--	---	
3	52	Y	4	4	
4	56	N	0	---	←Tail
5					
6					
7					

RS						
RS#	**Op type**	**Op1 ready?**	**Op1 RoB/value**	**Op2 ready?**	**Op2 RoB/value**	**Dest ROB**
0	Mult	Y	21	Y	3	0
1	Add	N	0	N	0	4
2						
3						
4						

ARF	**Reg#**	**0**	**1**	**2**	**3**	**4**	**5**
	Value	1	3	6	10	15	21

The instruction at PC 48 is a branch that has been predicted not-taken, but it is actually taken. The destination of the branch is PC 100, where the following code resides:

```
R0=R0+R1              // A
R1=R3*R0              // B
R2=R4+R0              // C
R3=R2-R3              // D
R2=R2*R4              // E
```

Show the state of the above tables if instruction A has retired, inst B has not finished executing, while C, D and E have progressed as far along as possible. *Be sure to label the head and tail of the ROB.* You are to place instruction A in the RoB in the same place it would have been placed had the prediction been initially correct. When arbitrary decisions need to be made, you are to just make them.

IF YOU WANT THIS ONE GRADED YOU MUST **CLEARLY** ***CROSS OUT THE FIRST ONE***