ECE 411, Exam 2

- This exam has 6 problems. Make sure you have a complete exam before you begin.
- Write your name on every page in case pages become separated during grading.
- You will have three hours to complete this exam.
- Write all of your answers on the exam itself. If you need more space to answer a given problem, continue on the back of the page, but clearly indicate that you have done so.
- This exam is closed-book. You may use a calculator and allowed 1-sheet of cheat sheet.
- DO NOT do anything that might be perceived as cheating. The minimum penalty will be a grade of zero.
- Show all of your work on all problems. Correct answers that do not include work demonstrating how they were generated may not receive full credit, and answers that show no work cannot receive partial credit.

- The exam is meant to test your understanding. Ample time has been provided. So, be patient. Read the questions/problems carefully before you answer.

- **Good luck!**

Problem 1 (25 pts): __________
Problem 2 (15 pts): __________
Problem 3 (15 pts): __________
Problem 4 (15 pts): __________
Problem 5 (15 pts): __________
Problem 6 (15 pts): __________
Total (100 pts): __________
1. Short answer questions (25 points)

Answer the following questions in no more than 30 words.

a) For a two-bit branch predictor, if a loop is executed only once and iterates 15 times, how many branch prediction misses will be incurred by the conditional branch that takes the execution back to the beginning of the loop (loop-back branch)? (3 points)

Solution: Assume initial state weakly taken, 2 times, once when the loop-back branch is first executed and once when the loop exits.

b) What is the functionality of a BTB (branch target buffer)? (2 points)

Solution: The branch target buffer provides the branch target PC so that the branch target instruction can be fetched at the immediate following cycle after the branch instruction is fetched.

c) What is the difficulty of handling return instructions with branch target buffer? (3 points)

Solution: The target PC of a return instruction depends on where the returning function is called from. The next one may not be the last one.

d) What is a Gap 2-level branch predictor? (3 points)

Solution: A 2-level predictor with a global branch history table (BHT) for all branches and an adaptive, private pattern history table (PHT) for each branch instruction.

e) How does register bypass reduce the number of data hazard stalls in a pipeline? (2 points)

Solution: The register bypass deliver the execution results from a later pipeline stage to an earlier pipeline stage to eliminate the need for stalling the instruction at that earlier stage.

f) What is the advantage of a 2-level page table over a 1-level page table? (2 points)

Solution: In a 2-level page table design, the bulk of the page table entries can be placed into virtual memory, paged in and out as needed and moved around in the physical address space as needed.
g) This question tests your knowledge on relative merit of one-level vs. two-level page table organizations. (10 points)

If you don’t believe that the specifications given in the following three subproblems are enough to determine your answer for sure, then say which option the given information points to, as evidence of its optimality.

i. Computer G is a real-time system, and requires minimal worst-case delay for hardware operations used by all its algorithms in order to meet its guarantees. G has been carefully designed to use a set of pre-determined algorithms with good worst-case total memory usage, and has memory to spare. Would a one-level or two-level page table be better? Why?

Solution: a one-level page table, because the second level, while saving space in the average case, causes additional delay in the worst-case for page misses.

ii. Computer H is tightly constrained by memory space, and in the worst case can use all available memory. Reductions of available memory due to hardware can be tolerated by compressing more of the program data during its execution, but this is expensive. The choice of how much to compress is very fine-grained. It uses 32-bit physical addresses, and 32-bit virtual addresses. The mission-critical code which can use all the memory is built into the operating system and will always run alone, with no other process active. It can have up to 1000 threads active at once. Would a one-level or two-level page table be better? Why?

Solution: a one-level page table, because the physical address space is as big as the virtual address space, meaning an identity mapping of virtual addresses to physical addresses is a possibility, and for a system using almost all virtual pages, and as a multi-level page table takes more memory. The 1000 threads don’t matter, because threads share an address space.

iii. Computer I is a desktop personal computer with 64-bit virtual addresses, and 32-bit physical addresses. It will be running up to 100 processes concurrently. It will be used for standard consumer software where occasional extra delays can be tolerated. Would a one-level or two-level page table be better? Why?

Solution: a two-level page table, because the memory cost otherwise will be huge.
2. Machine Problem (15 points)

This question tests your knowledge on MP2.2 and the concept of critical paths. Please refer to the following cache datapath, controller, and component delay value table. You will find the zoomed version of the datapath as well as the cache way block at the end of the exam draft.

<table>
<thead>
<tr>
<th>Component</th>
<th>Delay [ns]</th>
<th>Component</th>
<th>Delay [ns]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Period</td>
<td>50</td>
<td>Reg</td>
<td>5</td>
</tr>
<tr>
<td>Logic2</td>
<td>1</td>
<td>DataArray_128B</td>
<td>14</td>
</tr>
<tr>
<td>Logic3</td>
<td>2</td>
<td>DataArray_256B</td>
<td>20</td>
</tr>
<tr>
<td>Logic4</td>
<td>2</td>
<td>DataArray_512B</td>
<td>25</td>
</tr>
<tr>
<td>MUX2</td>
<td>2</td>
<td>DataArray_1KB</td>
<td>35</td>
</tr>
<tr>
<td>MUX4</td>
<td>4</td>
<td>DataArray_2KB</td>
<td>45</td>
</tr>
<tr>
<td>MUX8</td>
<td>6</td>
<td>DataArray_4KB</td>
<td>60</td>
</tr>
<tr>
<td>Compare8</td>
<td>3</td>
<td>ROM</td>
<td>3</td>
</tr>
<tr>
<td>Compare16</td>
<td>4</td>
<td>MEM</td>
<td>500</td>
</tr>
</tbody>
</table>
Assume the following cache specification:

- Cache line size (data only): 16 bytes
- Number of lines per way: 8 lines
- Set-associativity: 2-way
- LRU replacement policy: True LRU
- Write back
- Address: 16 bit, byte-addressable

a) Given the cache specification, what is the total size of the data array?

Solution: $16 \text{ bytes} \times 8 \text{ lines per way} \times 2 \text{ ways} = 256 \text{ bytes}$

b) There are tag array, dirty bit array, valid bit array, and LRU array in addition to data array in the cache design. Find the total size of the cache including all arrays.

Solution: Offset bits = 4 (byte-addressable, $2^4 = 16$), Index bits = 3 ($2^3 = 8$ sets per way), Tag bits = 16-3-4 = 9 bits
- Tag array size = $9 \times 8 \times 2 = 144 \text{ bits} = 18 \text{ bytes}$
- Valid array size = $1 \times 8 \times 2 = 16 \text{ bytes} = 2 \text{ bytes}$
- Dirty array size = $1 \times 8 \times 2 = 16 \text{ bytes} = 2 \text{ bytes}$
- LRU array size = 1 bit (for distinguishing between $2^1=2$ -way in each set) * 8 = 8 bits = 1 byte
- Total overhead = 256 (data array) + 18 + 2 + 2 + 1 = 279 \text{ bytes}

c) Assume that all the arrays (tag, valid bit, dirty bit array, LRU array) have the same delay as the data array (largest delay). Every single array access in that cache (data, tag, valid, LRU) will incur the specified delay, regardless of how you split up/organize the data. This delay does not include any logic or control that is external to the arrays. What is the path and delay value of the cache read hit? Show your calculations.

Solution:
- Address splitter (0ns) + Data array (DataArray_256B: 20ns) + Tag Compare (Compare16: 4ns) + Valid Check (And2=Logic2: 1ns) + MAX(2-way Check (OR2=Logic2: 1ns)), 2-way Select/LRU (Mux2: 2ns)) + Line to Word Convert (Mux8: 6ns) = 33 ns.
3. Cache and Virtual Memory Interaction (15 points)

This question tests your knowledge on the details of accessing data in a system with both TLB and cache.

A small memory system has the following attributes. (From Exam 1, remember?)
- 10-bit virtual addresses
- 8-bit physical addresses
- Byte-addressed
- Fully associative TLB with 2 entries
  - Random replacement policy
- 64B pages
- Single level page table
- 2-way set associative cache
  - Single level
  - 32B capacity
  - Block size of 4B
  - LRU replacement
  - Virtually indexed, physically tagged
  - Write-back policy
  - No write-allocation

Flow-chart showing what can happen in a memory access is attached at the end of the exam sheet. Please feel free to tear the page for convenience. States/events are labeled with numbers.

Suppose the contents of the TLB, cache, and page table are as follows.

<table>
<thead>
<tr>
<th>TLB Entries</th>
<th>Virtual Page Number</th>
<th>Physical Page Number</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<p>| Cache | Way 0 | | Way 1 | | |
|-------|-------| |-------| | |</p>
<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>Dirty</th>
<th>Tag</th>
<th>Valid</th>
<th>Dirty</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>1</td>
<td>0</td>
<td>0110</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0101</td>
<td>1</td>
<td>0</td>
<td>0100</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1110</td>
<td>1</td>
<td>0</td>
<td>0011</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1011</td>
<td>1</td>
<td>1</td>
<td>1101</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Virtual Page Number</td>
<td>Physical Page Number</td>
<td>Valid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For the following series of requests, list out (separated by commas) the numbers of the states in order in the flow-chart that the memory system goes through in responding to the requests. When states are active in parallel, their names should be separated by slashes. For example, “A, B, C, D/E, F” if it performed D and E in parallel. You can assume no pages are flushed to disk. Feel free to cross things out and write in new values in the given tables if it helps you work, but only the state/event sequence will be graded.

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Read/Write</th>
<th>State/Event Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x221</td>
<td>W</td>
<td>1, 8/11, 9/11, 10/11, 12, 13, 14, 17, 18, 16 (TLB hit, cache write hit)</td>
</tr>
<tr>
<td>0x257</td>
<td>R</td>
<td>1, 8/11, 2/11, 3/11, 4, 5/11, 6/11, 10/11, 12, 13, 19, 21, (22?), 23, 24, 14, 15, 16 (TLB miss, page fault, cache read miss)</td>
</tr>
<tr>
<td>0x244</td>
<td>W</td>
<td>1, 8/11, 9/11, 10/11, 12, 13, 19, 20, 16 (TLB hit, cache write miss)</td>
</tr>
<tr>
<td>0x247</td>
<td>R</td>
<td>1, 8/11, 9/11, 10/11, 12, 13, 19, 21, (22?), 23, 24, 14, 15, 16 (TLB hit, cache read miss)</td>
</tr>
</tbody>
</table>
4. Pipeline: Data Hazards (15 points)

Consider the 7-stage pipeline.

F1 - F2 - DE - EX - M1 - M2 - WB

The behavior of the pipeline is similar to the 5-stage pipeline discussed in class except the Fetch and Memory stages have each been broken into two halves. Memory address calculation and access are started during the first half of the Fetch and Memory stages (F1 and M1) and always complete during the F2 and M2 stages (2-cycle cache hit). The memory system is pipelined to be handled the concurrent memory accesses.

1. LDR R1, R0, 0
2. ADD R4, R4, R1
3. ADD R5, R4, R6
4. ADD R3, R3, -1
5. STR R5, R6, 2
6. LDR R0, R0, 4
7. LDR R1, R0, 0
8. ADD R2, R0, R0

a) Assume that no data forwarding or hazard detection is present. Identify where and how many NOPs need to be inserted into the above assembly code to ensure that the program runs as intended. Assume the cache always hits in two cycles. Also, assume that a value written into the register file becomes valid on the next clock cycle.

Solution:
LDR R1, R0, 0
NOP
NOP
NOP
NOP
ADD R4, R4, R1
NOP
NOP
NOP
NOP
ADD R5, R4, R6
ADD R3, R3, -1
NOP
NOP
NOP
STR R5, R6, 2
LDR R0, R0, 4
NOP
NOP
b) Suppose data forwarding and hazard detection has been added to the datapath and the original code is executed. How many stalls need to be inserted by the datapath for the code to run as intended? How many cycles does the code take to execute? What is the speedup over a)?

Solution:
LDR R1, R0, 0
STALL
ADD R4, R4, R1
ADD R5, R4, R6
ADD R3, R3, -1
STR R5, R0, 2
LDR R0, R0, 4
STALL
LDR R1, R0, 0
ADD R2, R0, R0

6+11=17 cycles to execute, Speedup = (6+23)/17 = 1.7

c) Now, assume that no data forwarding or hazard detection is present. Can the original code be reordered so that it executes in a shorter time? If yes, give the reordering that executes in the shortest time. If no, explain why.

Solution:
LDR R1, R0, 0
LDR R0, R0, 4
NOP
NOP
NOP
ADD R4, R4, R1
LDR R1, R0, 0
ADD R2, R0, R0
NOP
NOP
ADD R5, R4, R6
ADD R3, R3, -1
NOP
NOP
NOP
STR R5, R6, 2
5. Pipeline: Control Hazards (15 points)

Below is the code used to calculate the inner product of two vectors. Note that there are two conditional branches (Branch 1 and Branch 2) other than unconditional branches. Please answer the following questions. Assume all registers are set to 0 initially.

SEGMENT CodeSegment:
    LEA R2, A
    LEA R3, B
    LDR R1, R0, LENGTH
LOOP:
    ADD R1, R1, -1
    BRn HALT ;Branch 1
    LDR R4, R2, 0
    LDR R5, R3, 0
    ADD R2, R2, 2
    ADD R3, R3, 2
    ADD R6, R0, 0
MULTIPLY:
    ADD R6, R6, R4
    ADD R5, R5, -1
    BRp MULTIPLY ;Branch 2
    ADD R7, R7, R6
    BRnzp LOOP
HALT:
    BRnzp HALT ; Infinite loop

LENGTH: DATA2 4x0005
A:
    DATA2 4x0021
    DATA2 4x0123
    DATA2 4x0321
    DATA2 4x0001
    DATA2 4x0099
B:
    DATA2 4x0003
    DATA2 4x0002
    DATA2 4x0004
    DATA2 4x0005
    DATA2 4x0004

a) What are the actual behaviors (Taken: T, Not Taken: N) of each branches?
   Branch 1 – NNNNNT
   Branch 2 – TTN TN TTTN TTTN TTTN
b) What is the number of mispredictions for each branch prediction techniques for Branch 1? For 2-bit predictor, there are total 4 states: Strongly Taken: ST, Weakly Taken: WT, Weakly Not Taken: WN, Strongly Not Taken: SN.

- Static Predict Taken – 5
- Static Predict Not Taken – 1
- 1-bit Predictor (Initial: T) – 2
- 1-bit Predictor (Initial: N) – 1
- 2-bit Predictor (Initial: ST) – 3
- 2-bit Predictor (Initial: WT) – 2
- 2-bit Predictor (Initial: WN) – 1
- 2-bit Predictor (Initial: SN) – 1

c) How about Branch 2?

- Static Predict Taken – 5
- Static Predict Not Taken – 13
- 1-bit Predictor (Initial: T) – 9
- 1-bit Predictor (Initial: N) – 10
- 2-bit Predictor (Initial: ST) – 5
- 2-bit Predictor (Initial: WT) – 5
- 2-bit Predictor (Initial: WN) – 6
- 2-bit Predictor (Initial: SN) – 9

d) From the result you have chosen from above, which branch prediction techniques would you choose for both Branch 1 and Branch 2?

Solution: 2-bit predictor WT, and 2 bit predictor WN. (7 mispredictions).
6. Instruction-Level Optimization (15 points)

This program copies the value of A[i] to B[i] for i between 0 and N-1, using the following code:

```
LDR R1, R0, A
LDR R2, R0, B
LDR R7, R0, N
LOOP:
    LDR R3, R1, 0
    STR R3, R2, 0
    ADD R1, R1, 2
    ADD R2, R2, 2
    ADD R7, R7, -1
    BRzp LOOP
```

a) Unroll the loop from the above code 2 times (including the original loop) and reschedule to minimize the stalls. No extra registers can be used. Assume all data forwarding is available, BR resolves in the EXECUTE stage and also assume N is a multiple of 2.

Solution: One of the possible solutions:
```
LDR R1, R0, A
LDR R2, R0, B
LDR R7, R0, N
LOOP:
    LDR R3, R1, 0
    ADD R1, R1, 4
    STR R3, R2, 0
    ADD R7, R7, -2
    LDR R3, R1, -2
    ADD R7, R7, -2
    BRzp LOOP
    STR R3, R2, 2
    ADD R2, R2, 4
```
b) Now N can be any positive integer. Write the prologue and any additional changes to the loop above. No extra registers can be used. Assumptions are same as above a).

Solution:
LDR R7, R0, N
AND R1, R7, 1
BRz LOOP
LDR R1, R0, A
LDR R2, R0, B

LDR R3, R1, 0
ADD R1, R1, 2
ADD R7, R7, -1
BRn END
STR R3, R2, 0
ADD R2, R2, 2

LOOP:
....(same as b)...
END:
Problem 2. Cache Datapath Zoomed

<Cache Datapath Lower part>
<Cache Way Block>
Problem 3. Cache and Virtual Memory Interaction Flow Chart

1. Request
   - TLB lookup
     - TLB miss
       - Fault
       - Page walk
         - Bring page from disk to memory
         - Invalidate cache entry matching physical address
         - Update page table
       - Page table hit
         - Update TLB
         - Compare tag
           - Hit
             - Cache hit
               - Read
               - Write
               - Update cache
             - Cache miss
               - Evict
                 - Copy from memory to cache
               - Set dirty bit
             - Done
           - Cache miss
             - Write lower memory
           - Write back
         - Miss
           - Update lower memory
           - Copy from memory to cache
           - Clear dirty bit
           - Set dirty bit
           - Done