Instructions:

- **Do not open this exam until told by the proctor.** You will have 120 minutes to finish it.
- **Make sure your phone is turned OFF (not to vibrate!) before the exam starts.**
- Food, gum, and drink are strictly forbidden.
- **You may not use your phone or open your bag for any reason**, including to retrieve or put away pens or pencils, until you have left the exam room.
- This exam is closed-book, closed-notes, and closed-computational devices.
- If you get stuck on a problem, it may be to your benefit to move on to another question and come back later.
- Do not separate the pages. If a page becomes loose, reattach it with the provided staplers.
- Staple all scratch paper to your exam. Do not take any sheets of paper with you.
- If you require extra paper, please use the backs of the exam pages or the extra pages provided at the end of the exam. Clearly indicate on the question page where the graders can find the remainder of your work (e.g., “back of page” or “on extra sheet”).
- Use a pencil, or blue or black pen to complete the exam.
- If you have any questions, raise your hand and a proctor will come to answer them.
- When you turn in your exam, you may be required to show ID. If you forgot to bring your ID, talk to an exam proctor immediately.
- Good luck!
0.) The Easy One  
   (1 point total)
   • Check to make certain that your exam has all 6 pages (excluding the cover sheet).
   • Write your name and PennKey (username) on the front of the exam. Leave the recitation field blank.
   • Sign the certification that you comply with the Penn Academic Integrity Code.

1.) Virtual Memory  
   (7 points total)

1.1) (2 points) Which of the following conditions (there may be more than one) are considered a page fault?
   (a) A requested page was loaded incorrectly from main memory;
   (b) A requested page was not listed in the processor’s virtual memory hardware;
   (c) A page did something wrong and is being blamed for it;
   (d) The wrong page was loaded from main memory;
   (e) None of the above.

1.2) (1 point) How many bytes of virtual memory are available on a 16-bit, byte-addressed machine?

1.3) (2 points) How many virtual pages can be accessed on a 16-bit, byte-addressed machine if each page is 2 KB?

1.4) (2 points) What well-established phrase does each of the following three-letter acronyms stand for? Circle any that are related to virtual memory hardware.
   (a) TLA
   (b) TLB
   (c) TLC
   (d) TLD
2.) **Branch Davidians**  

(12 points total)

As an ardent follower of your friendly TA David Mally, you naturally hark to his teachings on how to predict the future. Among other things, you are of course well-versed in the art of branch prediction.

2.1) **(4 points)** What does each of the following acronyms stand for? Explain, in at most ten words the purpose of each one.

(a) RAS:

(b) BTB:

2.2) **(4 points)** Which of the following reasons explains why the branch predictor uses only the low-order bits of the branch instruction’s address? Circle all correct answers.

(a) Indexing based on the low-order bits is much faster than a “better” hash function;
(b) The predictor’s size is linear in the number of address bits used;
(c) The predictor’s size is quadratic in the number of address bits used;
(d) The predictor’s size is exponential in the number of address bits used;
(e) Most programs have all their branch instructions clustered together anyway;
(f) Branch instructions that are far apart in memory rarely share the same low-order bits;
(g) Branch instructions that are far apart in memory rarely execute in quick succession;
(h) None of the above.

2.3) **(4 points)** GPU designers disregard David’s branch teachings, and omit prediction hardware. Extend an olive branch to them by explaining, in at most twenty words why they often don’t use branch prediction.
3.) Keystone XL  (25 points total)

In order to handle the copious amounts of data coming out of Canada, you propose an extra long version of the superscalar design from Lab 4. In the new design, each stage is split in two and clock frequency is doubled. The stages are conveniently named F1, F2, D1, D2, X1, X2, M1, M2, W1, and W2. In general, values have to be available at the beginning of X1 and M1 depending on the instruction, and results are produced at the end of X2 and M2. However ALU operations except multiply, divide, and modulo produce their results at the end of X1.

3.1) (15 points) In the list below of all possible forwarding paths, cross out any that are not necessary for a fully bypassed pipeline. Assume that each path forwards rs, rt, and NZP bits as appropriate.

A. X2 -> A.X1  A.X2 -> B.X1
B. X2 -> A.X1  B.X2 -> B.X1
A. M1 -> A.X2  A.M1 -> B.X2  A.M1 -> A.X1  A.M1 -> B.X1
B. M1 -> A.X2  B.M1 -> B.X2  B.M1 -> A.X1  B.M1 -> B.X1
A. M2 -> A.M1  A.M2 -> B.M1  A.M2 -> A.X2  A.M2 -> B.X2  A.M2 -> A.X1  A.M2 -> B.X1
B. M2 -> A.M1  B.M2 -> B.M1  B.M2 -> A.X2  B.M2 -> B.X2  B.M2 -> A.X1  B.M2 -> B.X1

3.2) (2 points) You also plan to press the issue, by supporting N pipes with the ability to dispatch an instruction to every pipe on every cycle. How many read ports and how many write ports will your register file need, as a function of N?

3.3) (8 points) You expect all the groups that opposed the Pentium 4 pipeline to oppose your 10-stage, N-issue design as well. To defend your design, prepare responses explaining how you will overcome each of the potential objections below. Each response must be at most twenty words, but you do not need to write complete sentences.

(a) Latency, counted in instructions, increases by a factor of N

(b) Register file complexity increases by a factor of $N^2$

(c) Bypassing requires $N^2$ forwarding paths

(d) Cycle time increases by a factor of N
Consider a dynamically scheduled pipeline with the stages Fetch, Dispatch, Issue, RegRead, Execute, Memory, Writeback, and Commit, where the F and D stages form the in-order front end, and the I, R, X, M, and W stages form the out-of-order back-end. Assume your micro-architecture has 10 physical registers, labeled P0 through P9.

Rewrite the code below, renaming all architectural registers to physical registers. Assume all physical registers are free at the start of this code, and are on the free queue in increasing numerical order. Indicate all RAW, WAW, and WAR hazards on architectural registers (R0–R8) as comments. The first few lines have been completed for you.

```
.CODE
.ADDR 0x0000
    I0  CONST R1, #1
    I1  CONST R2, #15
    I2  MUL R7, R1, R2
        MUL P2, P0, P1; RAW R2
    I3  ADD R2, R1, R0
    I4  SLL R2, R2, #11
    I5  NOT R2, R1
    I6  XOR R2, R0, R0
    I7  MUL R0, R1, R2
    I8  ADD R1, R2, R3
    I9  LDR R3, R1, #0
    I10 SRA R3, R3, #2
    I11 ADD R7, R2, #0
    I12 MOD R7, R7, R3
    I13 JSRR R3
    I14 JSR _end
_end
    RET
```
5.) **This Question Also Out Of Order**  \hspace{1cm} (24 points total)

Fill in the out-of-order pipeline cycle diagram for the code in the previous question, beginning with `I0 \text{ CONST} R1, \#1` and ending with `I12 \text{ MOD} R7, R7, R3`. Assume the processor is three-way superscalar, triple-issue, fully bypassed, and has a single load-store unit. In other words, it has the same constraints on issuing instructions as your superscalar design from Lab 4, except that instructions can issue out-of-order. For bypassing purposes, your Lab 4's decode stage is replaced by the RegRead stage. The first line has been completed for you.

<table>
<thead>
<tr>
<th>Insn</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>I0</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>R</td>
<td>X</td>
<td>W</td>
<td>M</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
6.) Cache Incoherence        (26 points total)
Assuming a 16-bit architecture, with word-addressed memory (i.e. LC4), complete the tables below of
the L1 cache contents for each of the described cache architectures, and at each point in the stream
of loads and store operations in the left column. Also give the number of bits required for the tag in
each cache structure.
You may write all values as either hexadecimal, decimal, or binary. Each cache variant contains 8
entries, with a block size of 16 bits (one word), and employs the LRU replacement algorithm. The
only difference is in their associativity. In the column headers, “V” stands for “Valid,” and “D” stands
for “Dirty.” All cache frames start out invalid.

Operations:
Read 0x0000 from addr 0x1230
Read 0x9999 from addr 0x7899
Read 0x2020 from addr 0xABC0
Read 0x6666 from addr 0x4566
Read 0xB888 from addr 0x7898
Read 0x4444 from addr 0x4564
Read 0x2222 from addr 0x1232
Read 0xFFFF from addr 0xABC0
Write 0x0123 to addr 0xDEF0
Read 0x0000 from addr 0x7899
Read 0x4444 from addr 0x4564
Write 0x4567 to addr 0x4566
Read 0x0123 from addr 0xDEF0
Read 0x4444 from addr 0x4564
Write 0x4567 to addr 0x4566
Read 0x789A from addr 0xAAAA
Read 0x4567 from addr 0x4566

### 2-Way Set Associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
<th>V</th>
<th>D</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td></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>
</tr>
<tr>
<td></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>
</tr>
</tbody>
</table>

### Fully Associative

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data</th>
<th>V</th>
<th>D</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Tag Bits: _______

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data</th>
<th>V</th>
<th>D</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>