Practice Quiz #3

Please enter your name, Athena login name, and recitation section above. Enter your answers in the spaces provided below. You can use the extra white space and the backs of the pages for scratch work.

Problem 1. Potpourri (10 points)

For the following questions, circle the correct of the two choices.

(A) (2 points) In a system with virtual memory:
   a. User processes need to use system calls to communicate with peripherals that are using MMIO addresses outside of the process’s address space.
   b. User processes can directly load from and store to MMIO addresses because MMIO addresses are never virtual addresses. Virtual memory is deactivated for them.

(B) (2 points) In the processors that we studied in class and in part II (and part III) of Lab 7, virtual-to-physical address translation happens for:
   a. Every memory operation emitted in user mode (instruction loads, data loads, and data stores get their address translated).
   b. Only data memory operations emitted in user mode (data loads and data stores get their address translated).

(C) (2 points) After an interrupt in a process at pc X returns to the original running process, the control flow is restored to:
   a. Address X
   b. Address X + 4

(D) (2 points) An exception always returns control to the process that caused the exception.
   a. True
   b. False

   e.g., sleep syscall in lab 7, returns to another process
(E) (2 points) System calls (invoked using `ecall` in RISC-V):

a. Behave like `jal`, so the user process must save all caller-saved registers before invoking a system call.

b. Behave differently from `jal`, so the user process does not need to save all caller-saved registers before invoking a system call.

All registers treated as callee-saved registers in system calls.
Problem 2. Virtual Memory (16 points)

A standard RISC-V CPU is connected to a memory management unit (MMU) that uses a page table to translate 32-bit virtual addresses to 28-bit physical addressing using a page size of $2^{16}$ bytes.

(A) (6 points) Including the D (dirty) and R (resident) control bits, please give the number of entries in the page map and the number of bits required for each entry in page map.

Number of entries in the page map: $2^{16}$

Number of bits required for each page map entry: 14

(B) (10 points) The following program fragment is executed and a record is made of the inputs and outputs of the MMU. The record is shown is shown in the table on the right.

```assembly
lw x12, 0(x10)  
addi x10, x10, 4  
slli x12, x12, 5  
sw x11, 0(x12)
```

Using information from the program and the table above, please deduce the contents of as many entries as possible in the page table. Please make an entry in the table below for each page table entry we learn about, giving the VPN, D and R controls bits, and the PPN, showing the state of the page table after the execution of the program fragment. If you can’t deduce the value of a field, please leave the field blank. Assume that pages holding instructions are read-only.

<table>
<thead>
<tr>
<th>Access type</th>
<th>Virtual address</th>
<th>Physical address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inst. fetch</td>
<td>0x13FFF8</td>
<td>0x3FFFF8</td>
</tr>
<tr>
<td>Data read</td>
<td>0x987654</td>
<td>0x897654</td>
</tr>
<tr>
<td>Inst. fetch</td>
<td>0x13FFFC</td>
<td>0x3FFFFC</td>
</tr>
<tr>
<td>Inst. fetch</td>
<td>0x140000</td>
<td>0x420000</td>
</tr>
<tr>
<td>Inst. fetch</td>
<td>0x140004</td>
<td>0x420004</td>
</tr>
<tr>
<td>Data write</td>
<td>0x226004</td>
<td>0x006004</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VPN</th>
<th>D</th>
<th>R</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x13</td>
<td>0</td>
<td>1</td>
<td>0x3F</td>
</tr>
<tr>
<td>0x98</td>
<td>1</td>
<td></td>
<td>0x89</td>
</tr>
<tr>
<td>0x14</td>
<td>0</td>
<td>1</td>
<td>0x42</td>
</tr>
<tr>
<td>0x22</td>
<td>1</td>
<td>1</td>
<td>0x00</td>
</tr>
</tbody>
</table>

VA: VPN: 16 bits, page offset: 16 bits
PA: PPN: 12 bits, page offset: 16 bits
Problem 3. Pipelining Combinational Circuits (18 points)

For each of the questions below, please create a valid K-stage pipeline of the given circuit. Each component in the circuit is annotated with its propagation delay. Show your pipelining contours and place large black circles (●) on the signal arrows to indicate the placement of pipeline registers. Give the latency and throughput of each design, assuming ideal registers ($t_{PD}=0$, $t_{SETUP}=0$). Remember that our convention is to place a pipeline register on each output.

(A) (3 points) Show the maximum-throughput 1-stage pipeline.

Latency (ns): __11_____
Throughput (ns$^{-1}$): ___$1/11$_____

$t_{CLK} = 11 \text{ ns}$

(B) (5 points) Show the maximum-throughput 2-stage pipeline using a minimal number of registers.

Latency (ns): ___14_____
Throughput (ns$^{-1}$): ___$1/7$_____

$t_{CLK} = 7 \text{ ns}; \text{ latency } = 2 \times 7 = 14$

(C) (5 points) Show the maximum-throughput pipeline using a minimal number of registers.

Latency (ns): ___12_____
Throughput (ns$^{-1}$): ___$1/4$_____

$t_{CLK} = 4 \text{ ns}; \text{ latency } = 3 \times 4 = 12$
(D) (5 points) You manage to reimplement the slowest combinational component in the previous circuit (the one with a propagation delay of 4 ns) using two components with propagation delays of 2ns, as shown right. Show the maximum-throughput pipeline using a minimal number of registers.

Latency (ns): __12_______

Throughput (ns\(^{-1}\)): __1/3_______

t\(_{CLK}\) = 3 ns; latency = 4 * 3 = 12
Problem 4. Pipelined Processors (15 points)

Consider the execution of the following code sequence on a 5-stage pipelined RISC-V processor, which is fully bypassed, predicts all branches not-taken, and kills instructions following taken branches. Assume that branch taken/not taken decisions are made in the EXE stage. Also, assume that the results of load operations are not available until the WB stage. The loop sums the first 100 elements of the integer array at address 0x1000 and stores the result at address 0x2000. Assume execution halts at instruction unimp.

```
lui x11, 1       // x11 = 0x1000 (array)
lui x15, 2       // x15 = 0x2000 (result)
addi x12, x0, 0  // x12 holds sum
addi x14, x11, 400 // address of array element 101

A: lw x13, 0(x11)    // load next array element
   addi x11, x11, 4   // addr of next array element
   add x12, x12, x13  // add element value to sum
   bne x11, x14, A    // loop until element 101
   sw x12, 0(x15)     // store result
   xor x2, x3, x4
   unimp
```

To help answer the following questions, fill in the pipeline diagram below showing execution of the loop assuming that the loop was previously executed, and will be repeated again after this iteration. Extra copies of this table are provided at the end of the quiz.

<table>
<thead>
<tr>
<th>Cycle</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>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>lw</td>
<td>addi</td>
<td>add</td>
<td>bne</td>
<td>bne</td>
<td>sw</td>
<td>xor</td>
<td>lw</td>
<td>addi</td>
<td>add</td>
</tr>
<tr>
<td>DEC</td>
<td>lw</td>
<td>addi</td>
<td>add</td>
<td>add</td>
<td>bne</td>
<td>sw</td>
<td>NOP</td>
<td>lw</td>
<td>addi</td>
<td></td>
</tr>
<tr>
<td>EXE</td>
<td>lw</td>
<td>addi</td>
<td>NOP</td>
<td>add</td>
<td>bne</td>
<td>NOP</td>
<td>NOP</td>
<td>lw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td>lw</td>
<td>addi</td>
<td>NOP</td>
<td>add</td>
<td>bne</td>
<td>NOP</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td>lw</td>
<td>addi</td>
<td>NOP</td>
<td>add</td>
<td>bne</td>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bypass paths
- Cycle 5: lw to add
- Cycle 6: addi to lw
- Cycle 7: add to sw
(A) (3 points) Are there points in the execution of the sequence when data is bypassed from the EXE stage back to the DEC stage? If so, give the instruction(s) in the DEC stage at each such point; otherwise, enter “NONE”.

Instruction(s) in DEC, or NONE: _______NONE_________

(B) (3 points) Are there points in the execution of the sequence when data is bypassed from the WB stage back to the DEC stage? If so, give the instruction(s) in the DEC stage at each such point; otherwise, enter “NONE”.

Instruction(s) in DEC, or NONE: _______add, bne_________

(C) (3 points) Are there points during the execution of the sequence when the pipeline is stalled? If so, give the instruction(s) in the DEC stage at each such point; otherwise, enter “NONE”.

Instruction(s) in DEC, or NONE: _______add_________

(D) (6 points) Fill in the contents of the WB stage below. You may have blank spaces indicating the time it takes for the lw instruction to reach the WB stage.

<table>
<thead>
<tr>
<th>cycle</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>
</tr>
</thead>
<tbody>
<tr>
<td>WB stage</td>
<td></td>
<td></td>
<td></td>
<td>lw</td>
<td>addi</td>
<td>NOP</td>
<td>add</td>
<td>bne</td>
<td>NOP</td>
<td></td>
</tr>
</tbody>
</table>
Problem 5. Processor Pipeline Performance (20 points)

You are designing a four stage RISC-V processor (IF, DEC, EXE, WB) with a BTB for next address prediction and a scoreboard for stalling on data hazards. Currently you are trying to decide whether to include bypassing through the register file from the write-back stage to the decode stage. As part of this evaluation, you construct two processors:

Processor A: No bypassing from WB to DEC.
Processor B: Bypassing from WB to DEC through the register file.

(A) (2 points) To construct processor B in Bluespec, you start with processor A and replace the register file with a bypassing register file. After this change, the WB rule conflicts with the DEC rule because of the scoreboard. What scheduling constraint do you need between the following methods of your scoreboard to get concurrent execution of WB and DEC with the bypassing register file? (you may answer <, >, CF, or C)

scoreboard.search __>___ scoreboard.remove

You are using the following loop of an important program to evaluate the performance of the processor:

L1: lw t0, 0(a0)
    add a1, a1, t0
    addi a0, a0, 4
    blt a0, a2, L1

For the following questions, assume this loop has been running for a long time.

(B) (4 points) How many cycles per loop iteration does the decode stage stall due to read after write hazards in the following cases?

Processor A decode stall cycles per iteration: _____4__________

Processor B decode stall cycles per iteration: _____2__________

(C) (2 points) How many cycles does this loop take to execute in the following cases?

Processor A cycles per iteration: _____8__________

Processor B cycles per iteration: _____6__________

<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>lw</th>
<th>add</th>
<th>addi</th>
<th>addi</th>
<th>addi</th>
<th>blt</th>
<th>lw</th>
<th>lw</th>
<th>lw</th>
<th>add</th>
<th>addi</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>DEC</td>
<td>lw</td>
<td>add</td>
<td>add</td>
<td>addi</td>
<td>addi</td>
<td>blt</td>
<td>lw</td>
<td>lw</td>
<td>lw</td>
<td>add</td>
<td>add</td>
</tr>
<tr>
<td></td>
<td>EXE</td>
<td>lw</td>
<td>NOP</td>
<td>NOP</td>
<td>Addi</td>
<td>NOP</td>
<td>NOP</td>
<td>blt</td>
<td>lw</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WB</td>
<td>lw</td>
<td>NOP</td>
<td>NOP</td>
<td>Addi</td>
<td>NOP</td>
<td>NOP</td>
<td>blt</td>
<td>lw</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>lw</th>
<th>add</th>
<th>addi</th>
<th>addi</th>
<th>addi</th>
<th>blt</th>
<th>lw</th>
<th>lw</th>
<th>add</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>DEC</td>
<td>lw</td>
<td>add</td>
<td>add</td>
<td>addi</td>
<td>blt</td>
<td>lw</td>
<td>lw</td>
<td>add</td>
<td>add</td>
</tr>
<tr>
<td></td>
<td>EXE</td>
<td>lw</td>
<td>NOP</td>
<td>addi</td>
<td>NOP</td>
<td>blt</td>
<td>lw</td>
<td>NOP</td>
<td>add</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WB</td>
<td>lw</td>
<td>NOP</td>
<td>addi</td>
<td>NOP</td>
<td>blt</td>
<td>lw</td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Processor A has the following propagation delays for each of the pipeline stages:

- IF: 2 ns
- DEC: 3.0 ns
- EX: 3.5 ns
- WB: 1.0 ns

The logic for the bypassing path of processor B can be viewed as taking the output from the DEC and WB stages of processor A and adding an additional bypass logic (BYP) as shown in the picture below.

Assuming the BYP logic has a propagation delay of 1 ns.

(D) (4 points) What is the minimum clock period for each processor?

Clock period for processor A: ___ 3.5 ns _________

Clock period for processor B: ___ 4 ns _________

(E) (4 points) For the loop shown above, what is the average cycles per instruction for the two processors:

Average cycles per instruction for processor A: ___ 8/4 = 2 _________

Average cycles per instruction for processor B: ___ 6/4 = 3/2 _________

(F) (4 points) For the loop shown above, what is the average number of instructions per second for the two processors:

Average number of instructions per second for processor A: ___ 1/(7ns) _________

Average number of instructions per second for processor B: ___ 1/(6ns) _________

Instr/sec = 1/(cyc/instr)(sec/cycle)
A: 1/(2*3.5ns) = 1/(7ns)
B: 1/((3/2)*4ns) = 1/(6ns)
Blank pipeline diagrams for Problem 3

<table>
<thead>
<tr>
<th>Cycle</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>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td></td>
<td>lw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

END OF PRACTICE QUIZ 3!