Computer Architecture CS372 – Exam 2

- This exam has 9 pages. Please make sure you have all of them.
- Write your name on this page and initials on every other page now.
- You may only use the green card for this exam, no books, notes or calculators may be used.
- You have 75 minutes for this exam. Budget your time carefully.
- You can tear off the last page to use as reference.
- Please write neatly. I can’t give you points if I can’t read or understand your answer.
- PLEASE READ THE QUESTIONS CAREFULLY AND MAKE SURE YOU HAVE ANSWERED THEM

<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible Score</th>
<th>Your Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Short Answer</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2. Single Cycle Datapath</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3. Multi-cycle Datapath</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>4. Pipelining</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1) Short Answer

a) Pipelining primarily improves **throughput** latency. Circle one (5 points)

b) What are the main functions of the **Cause** and **EPC** registers in exception handling for the MIPS ISA? (5 points)
   
i) **EPC**: Stores the address of the instruction that caused the exception.

   ii) **Cause**: Stores the encoded cause (undefined instruction, overflow) of the exception.

c) Suppose each stage of the instruction takes the following times:

   IF = 7 ns, ID = 8 ns, EX = 15 ns, MA = 10 ns, WB = 8 ns
   
   and there is 2 ns overhead for pipelining. (10 points)
   
i) What is the cycle time for a single cycle datapath?

   The time for the longest instruction, lw = 7 + 8 + 15 + 10 + 8 = 48ns

   ii) What is the cycle time for a multiple cycle datapath?

   The time for the longest stage = 15ns

   iii) What is the cycle time for a pipelined datapath?

   The time for the longest stage plus the overhead = 15 + 2 = 17ns

d) Using the cycle times you calculated above, how long will each take to execute 4 add instructions assuming no hazards occur? Give your answer in nanoseconds. (10 points)

   i) single cycle 48 * 4 = 192ns

   ii) multi cycle There are 4 cycles for 1 add instruction, so 4 * (15 * 4) = 240ns

   iii) pipelined Five cycles for the first add, plus one more for each additional add = (5 + 3) * 17 = 136ns
2) Single Cycle Datapath

The datapath above supports the following instructions:

- **lw_add** rd, (rs), rt  
  \[ R_{rd} = Memory[R_{rs}] + R_{rt}; \]
- **addi_st** (rs), rs, imm  
  \[ Memory[R_{rs}] = R_{rs} + \text{imm}; \]
- **sll_add** rd, rs, rt, imm  
  \[ R_{rd} = (R_{rs} \ll \text{imm}) + R_{rt}; \]

All instructions use the same format (shown below). Not all instructions use all fields.

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>imm</th>
</tr>
</thead>
<tbody>
<tr>
<td>(bits 31-26)</td>
<td>(bits 25-21)</td>
<td>(bits 20-16)</td>
<td>(bits 15-11)</td>
<td>(bits 10-0)</td>
</tr>
</tbody>
</table>

For each of the above instructions, specify how the control signals should be set for correct operation. Use X for don’t care. **ALUOp** can be ADD, SUB, SLL, PASS_A, or PASS_B (e.g., PASS_A means pass through the top operand without change). (20 points)

<table>
<thead>
<tr>
<th>Inst</th>
<th>ALUsrc1</th>
<th>ALUsrc2</th>
<th>ALUsrc3</th>
<th>ALUop1</th>
<th>ALUop2</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>RegWrite</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw_add</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>ADD</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>addi_st</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>PASS_B</td>
<td>ADD</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>sll_add</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SLL</td>
<td>ADD</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
3) Multicycle Datapath

Consider extending the MIPS architecture with the following instruction below which subtracts two registers from a third.

```
sub3 rd, rs, rt, ru  # rd = ru-rs-rt
```

Note that rd=ru-rs-rt can be calculated as rd = ru-(rt+rs).

This will use the same format as R-type instructions (shown here for reference) where the shamt field is used to hold ru.

```
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt/ru</th>
<th>func</th>
</tr>
</thead>
<tbody>
<tr>
<td>(bits 31-26)</td>
<td>(bits 25-21)</td>
<td>(bits 20-16)</td>
<td>(bits 15-11)</td>
<td>(bits 10-6)</td>
<td>(bits 5-0)</td>
</tr>
</tbody>
</table>
```

a) The multicyle datapath from lecture appears below. Show what changes are needed to support sub3. You should only add wires and muxes to the datapath; do not modify the main functional units themselves (i.e. the ALU, Memory, and register file). Try to keep your diagram neat (15 points).

Add a mux here controlled by the signal RegRs. If 0, select IR[25-21], if 1 select IR[10-6]

Add another input, and make ALUSrcB 3 bits. Input 4 should come from ALUOut
b) Complete this finite state machine diagram for the sub3 instruction. Assume control values not shown in each stage are 0. Remember to account for any control signals that you added or modified in the previous part to the question! (20 points).
c) The sub3 instruction can be used in place of two dependent sub instructions, reducing the number of instructions that need to be executed. Below are two functionally equivalent programs; the second of which uses the sub3 instruction.

**Program 1**

```
lw  $t0, 0($a0)
lw  $t1, 4($a0)
lw  $t2, 8($a0)
add $t3, $t2, $t1
sub  $t3, $t0, $t3
sw  $t3, 0($a1)
```

**Program 2**

```
lw  $t0, 0($a0)
lw  $t1, 4($a0)
lw  $t2, 8($a0)
sub3 $t3, $t2, $t3, $t0
sw  $t3, 0($a1)
```

Assuming the datapath and control that you implemented in parts a and b, how much faster (or slower) is program 2 than program 1? You may leave your answer as a fraction. (10 points)

<table>
<thead>
<tr>
<th>Program 1</th>
<th>27 cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 2</td>
<td>24 cycles</td>
</tr>
</tbody>
</table>

Program 2 is faster by 3/27

d) Implementing sub3 instruction in the pipelined datapath we discussed in class would be much more difficult. Give two specific reasons why. (5 points)

i) ALU Collision (need another ALU) – structural hazard

ii) Need 3-port register - structural hazard
4) Pipelined Datapath

Consider the instruction sequence

\[
\begin{align*}
\text{sub} & \quad \text{\$t0, \$t1, \$t2} \\
\text{beq} & \quad \text{\$t0, \$0, loop}
\end{align*}
\]

This sequence causes a stall using the pipelined datapath from class (shown on page 7) because we need \$t0 to resolve the branch the same cycle it is produced as shown below:

\[
\begin{array}{cccccc}
\text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} \\
\hline
\text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB}
\end{array}
\]

Since we don't have forwarding from EX to ID, the pipeline must stall. You need to fix this. Specifically, you need to:

1. Add multiplexers and a forwarding unit to the ID stage of the datapath on the next page. Show the MUX and forwarding unit inputs clearly.

2. Show the forwarding equations for the new forwarding unit to show how the MUX selections are made.
a) Add multiplexers and a forwarding unit to the ID stage to this figure. Show the MUX and forwarding unit inputs clearly.

b) Show the forwarding equations for the new forwarding unit here:

If $\text{ID/EX.RegWrite} = 1$ AND
$\text{IF/ID.RegisterRS} = \text{IF/EX.RegisterRD}$
then $FS = 1$

If $\text{ID/EX.RegWrite} = 1$ AND
$\text{IF/ID.RegisterRT} = \text{IF/EX.RegisterRD}$
then $FT = 1$