Computer Architecture
EE 4720
Midterm Examination
20 October 1999, 10:40–11:30 CDT

Problem 1 (33 pts)
Problem 2 (33 pts)
Problem 3 (34 pts)

Problem 1 (33 pts)
Problem 2 (33 pts)
Problem 3 (34 pts)

Alias
Exam Total (100 pts)

Good Luck!
Problem 1:

(a) Add exactly the bypass paths that are needed so the code below executes as shown in the pipeline execution diagram. (Don’t add any bypass paths that are not needed by the code.) (15 pts)

! r1 = 0x1010, r2 = 0x2020, r3 = 0x3030, r4 = 0x4040, r5 = 0x5050
! Cycle 0 1 2 3 4 5 6 7
LINE1: LINE1 = 0x100
lw r1, 16(r2) IF ID EX MEM WB IF ID EX ...
addi r2, r2, #4 IF ID EX MEM WB IF ID ...
sw 20(r2), r1 IF ID EX MEM WB IF ...
beqz r2, LINE1 IF ID EX MEM WB ...
and r3, r4, r5 IFx

(b) Show the values of the registers listed below in cycle 4 of the execution above using the added bypass paths. Assume that the load instruction retrieves a 0x1234 from memory. Instructions are squashed by replacing them with an or r0, r0, r0. (18 pts)
Problem 2:

(a) Show a pipeline execution diagram for the code below on the implementation shown until $lw$ is fetched a second time. The first branch is not taken but the last one is. The only bypass paths available are the ones shown. (18 pts)

```
LOOP:
lw r1, 0(r2)
andi r3, r1, #4
beqz r3, LINE1
add r4, r4, r1
j LINE2
LINE1:
add r5, r5, r1
LINE2:
sw 4(r2), r1
addi r2, r2, #8
bneq r2, LOOP
xor r5, r6, r7
```
(b) Rewrite the code below (which is the same as the code on the previous page) using one-cycle delayed branches and predicated instructions and schedule the code so that it executes as quickly as possible. (Do not unroll the loop.) Assume that bypass paths are provided for the predicated instructions. It should be possible to remove all stalls, but if any remain point them out (for partial credit). (15 pts)

```assembly
LOOP:
   lw r1, 0(r2)
   andi r3, r1, #4
   beqz r3, LINE1
   add r4, r4, r1
   j LINE2
LINE1:
   add r5, r5, r1
LINE2:
   sw 4(r2), r1
   addi r2, r2, #8
   bneq r2, LOOP
   xor r5, r6, r7
```
Problem 3: Answer each question below.

(a) Why do DLX branches (and branches in many other ISAs) use displacement addressing? Why don’t branches use indirect addressing (destination address in a register) instead of displacement addressing? (8 pts)

(b) The code below executes on an implementation that uses a reservation register to detect WB structural hazards. At cycle zero the reservation register contains all zeros. Show the state of the reservation register at the end of each cycle below. Indicate which (if any) bit positions are tested in each cycle. (9 pts)

<table>
<thead>
<tr>
<th>Cycle</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>
</tr>
</thead>
<tbody>
<tr>
<td>multif f0, f1, f2</td>
<td>IF</td>
<td>ID</td>
<td>M0</td>
<td>M1</td>
<td>M2</td>
<td>M3</td>
<td>M4</td>
<td>M5</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addf f3, f4, f5</td>
<td>IF</td>
<td>ID</td>
<td>A0</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subf f6, f7, f8</td>
<td>IF</td>
<td>ID</td>
<td>-&gt;</td>
<td>A0</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gtf f9, f10, f11</td>
<td>IF</td>
<td>-&gt;</td>
<td>ID</td>
<td>A0</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</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>
| nop               |    |    |    |    |    |    |    |    |    |    |...

(c) Many packed operand instructions perform saturating arithmetic. What is saturating arithmetic? Provide an example. (8 pts)

(d) In homework 3, a special return address register (ERA) was used to hold exception return addresses. The jump and link instructions, jal and jalr, use r31 for the return address; is this an option for exceptions? Explain. (9 pts)