2 Computer Design (SWM)

The version of Thacker’s Tiny Computer 3 (TTC3) that was used in the 2012 ECAD Laboratory sessions (instruction set summary is below) has the following pipeline stages:

<table>
<thead>
<tr>
<th>fetch</th>
<th>decode/register</th>
<th>execute/memory</th>
<th>write-back</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fetch</td>
<td>access</td>
<td></td>
</tr>
</tbody>
</table>

Currently the implementation only supports one instruction in the pipeline at a time, i.e. the next instruction is only fetched when the current one finishes in the write-back stage.

If the implementation were to attempt to fetch a new instruction every clock cycle, explain the following microarchitectural issues:

(a) What data hazards would exist and how can they be resolved whilst preserving the programmer’s sequential model? [5 marks]

(b) What are control hazards and how can we avoid exposing them to the programmer? [5 marks]

(c) When are branch target addresses computed on the TTC3 and how many bubbles will be introduced when taking a jump? Assume that such a tiny computer would not have a branch predictor. [5 marks]

(d) On the TTC3, every instruction (except jump) can conditionally skip the next instruction. How might skip be implemented and how many pipeline bubbles need to be introduced? [5 marks]

### TTC3 Instruction Set Summary

<table>
<thead>
<tr>
<th>Rw</th>
<th></th>
<th>Ra</th>
<th>Rb</th>
<th>Func</th>
<th>Shift</th>
<th>Skip</th>
<th>Opcode</th>
</tr>
</thead>
</table>

Shift (rotates right): 0: no shift 1: RCY 1 2: RCY 8 3: RCY16
Skip: 0: never 1: ALU<0 2: ALU=0 3: InRdy
Opcode: 0: normal: Rw=F(Ra,Rb), skip if condition 1: storeDM: DM[Rb]=Ra, Rw=F(Ra,Rb), skip if condition 2: storeIM: IM[Rb]=Ra, Rw=F(Ra,Rb), skip if condition 3: out: OutStrobe, Rw=F(Ra,Rb), skip if condition 4: loadDM: Rw=DM[Rb], ALU=F(Ra,Rb), skip if condition 5: in: Rw=in_data, ALU=F(Ra,Rb), skip if condition 6: jump: Rw=PC+1, PC=F(Ra,Rb), no skip 7: reserved

LC=load constant (bits 23:0 of the instruction), no skip
PC=program counter
ALU=Function(Ra,Rb), where the Function is specified by the Func bits
F(Ra,Rb)=rotate(Shift, ALU), where the rotation is specified by the Shift bits