Lecture 4: Advanced Pipelines

• Control hazards, multi-cycle in-order pipelines, static ILP (Appendix A.4-A.10, Sections 2.1-2.2)
Data Dependence Example

lw R1, 8(R2)

sw R1, 8(R3)
Summary

• For the 5-stage pipeline, bypassing can eliminate delays between the following example pairs of instructions:
  
  - add/sub R1, R2, R3
  - add/sub/lw/sw R4, R1, R5

  lw R1, 8(R2)
  sw R1, 4(R3)

• The following pairs of instructions will have intermediate stalls:
  
  - lw R1, 8(R2)
  - add/sub/lw R3, R1, R4 or sw R3, 8(R1)

  fmul F1, F2, F3
  fadd F5, F1, F4
Control Hazards

- Simple techniques to handle control hazard stalls:
  - for every branch, introduce a stall cycle (note: every 6th instruction is a branch!)
  - assume the branch is not taken and start fetching the next instruction – if the branch is taken, need hardware to cancel the effect of the wrong-path instruction
  - fetch the next instruction (branch delay slot) and execute it anyway – if the instruction turns out to be on the correct path, useful work was done – if the instruction turns out to be on the wrong path, hopefully program state is not lost
Branch Delay Slots

(a) From before

DADD R1, R2, R3
if R2 = 0 then
Delay slot
becomes
if R2 = 0 then
DADD R1, R2, R3

(b) From target

DSUB R4, R5, R6
DADD R1, R2, R3
if R1 = 0 then
Delay slot
becomes
DSUB R4, R5, R6
DADD R1, R2, R3
if R1 = 0 then
DSUB R4, R5, R6

(c) From fall-through

DADD R1, R2, R3
if R1 = 0 then
Delay slot
OR R7, R8, R9
DSUB R4, R5, R6
becomes
DADD R1, R2, R3
if R1 = 0 then
OR R7, R8, R9
DSUB R4, R5, R6
Slowdowns from Stalls

• Perfect pipelining with no hazards → an instruction completes every cycle (total cycles ~ num instructions) → speedup = increase in clock speed = num pipeline stages

• With hazards and stalls, some cycles (= stall time) go by during which no instruction completes, and then the stalled instruction completes

• Total cycles = number of instructions + stall cycles

• Slowdown because of stalls = 1/ (1 + stall cycles per instr)
Pipeline Implementation

- Signals for the muxes have to be generated – some of this can happen during ID
- Need look-up tables to identify situations that merit bypassing/stalling – the number of inputs to the muxes goes up
## Detecting Control Signals

<table>
<thead>
<tr>
<th>Situation</th>
<th>Example code</th>
<th>Action</th>
</tr>
</thead>
</table>
| No dependence                 | LD  **R1**, 45(R2)  
DADD R5, R6, R7  
DSUB R8, R6, R7  
OR  R9, R6, R7 | No hazards                                                 |
| Dependence requiring stall     | LD  **R1**, 45(R2)  
DADD R5, **R1**, R7  
DSUB R8, R6, R7  
OR  R9, R6, R7 | Detect use of R1 during ID of DADD and stall               |
| Dependence overcome by forwarding | LD  **R1**, 45(R2)  
DADD R5, R6, R7  
DSUB R8, **R1**, R7  
OR  R9, R6, R7 | Detect use of R1 during ID of DSUB and set mux control signal that accepts result from bypass path |
| Dependence with accesses in order | LD  **R1**, 45(R2)  
DADD R5, R6, R7  
DSUB R8, R6, R7  
OR  R9, **R1**, R7 | No action required                                         |
Multicycle Instructions

<table>
<thead>
<tr>
<th>Functional unit</th>
<th>Latency</th>
<th>Initiation interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer ALU</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Data memory</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>FP add</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>FP multiply</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>FP divide</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>
Effects of Multicycle Instructions

- Structural hazards if the unit is not fully pipelined (divider)
- Frequent RAW hazard stalls
- Potentially multiple writes to the register file in a cycle
- WAW hazards because of out-of-order instr completion
- Imprecise exceptions because of o-o-o instr completion

Note: Can also increase the “width” of the processor: handle multiple instructions at the same time: for example, fetch two instructions, read registers for both, execute both, etc.
Precise Exceptions

• On an exception:
  ➢ must save PC of instruction where program must resume
  ➢ all instructions after that PC that might be in the pipeline must be converted to NOPs (other instructions continue to execute and may raise exceptions of their own)
  ➢ temporary program state not in memory (in other words, registers) has to be stored in memory
  ➢ potential problems if a later instruction has already modified memory or registers

• A processor that fulfils all the above conditions is said to provide precise exceptions (useful for debugging and of course, correctness)
Dealing with these Effects

- Multiple writes to the register file: increase the number of ports, stall one of the writers during ID, stall one of the writers during WB (the stall will propagate)

- WAW hazards: detect the hazard during ID and stall the later instruction

- Imprecise exceptions: buffer the results if they complete early or save more pipeline state so that you can return to exactly the same state that you left at
ILP

- Instruction-level parallelism: overlap among instructions: pipelining or multiple instruction execution

- What determines the degree of ILP?
  - dependences: property of the program
  - hazards: property of the pipeline
Types of Dependences

- Data dependences: an instr produces a result for another (true dependence, results in RAW hazards in a pipeline)

- Name dependences: two instrs that use the same names (anti and output dependences, result in WAR and WAW hazards in a pipeline)

- Control dependences: an instruction’s execution depends on the result of a branch – re-ordering should preserve exception behavior and dataflow
Title

• Bullet