Delayed Branch

- Assume branch delay of one cycle
- If branch taken, execution is:
  - Branch instruction
  - Branch delay instruction
  - Branch target
- If branch not taken, execution is:
  - Branch instruction
  - Branch delay instruction
  - Branch instruction + 2

- Instruction immediately following branch is executed irrespective of whether the branch is taken or not
- Rely on compiler to make successor instructions valid and useful

Behavior of Delayed Branch

BEQZ R1, L1
branch delay instruction
instruction + 2
instruction + 3
branch target
branch target + 1
branch target + 2
Scheduling the Branch Delay Slot

Impact of Control Hazards

- Impact of control hazards on performance depends on:
  - Frequency of branch instructions
  - Accuracy of branch prediction
  - Stall penalty for each misprediction
  - 1 or 2 cycles on a simple 5-stage MIPS
  - Higher in deeper pipelines and more complex processors

Example

- Assume that branches comprise 20% of all instructions. Also assume that the branch prediction is 80% accurate and incurs a 2 cycle stall on each misprediction. What is the impact of control hazards on the CPI of the pipelined processor? Ignore all other sources of pipeline hazards.

  - Solution:
    CPI without control hazards = 1
    Added CPI due to control hazards = Branch frequency * (1 – Branch prediction accuracy) * Stall penalty = 20% * (1 - 80%) * 2 = 0.08
    CPI with control hazards = 1 + 0.08 = 1.08

Another Example

<table>
<thead>
<tr>
<th>Branch scheme</th>
<th>Penalty unconditional</th>
<th>Penalty untaken</th>
<th>Penalty taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash pipeline</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Predicted taken</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Predicted untaken</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Speedup of “predicted-not-taken” over “stall pipeline” = 1.56 / 1.38 = 1.13
Implementation of MIPS Pipeline

MIPS Datapath

Pipelining the MIPS Datapath

Events in Pipeline Stages
### Detecting Hazards

<table>
<thead>
<tr>
<th>Situation</th>
<th>Example code sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>No dependence</td>
<td>LD R1,45(R2)</td>
<td>No hazard possible because no dependence exists on R1 in the immediately following three instructions.</td>
</tr>
<tr>
<td>Dependence requiring stall</td>
<td>DADD R5, R1, R7</td>
<td>Comparators detect the use of R1 in the DADD and stall the DADD (and DSUB and OR) before the DADD begins EX.</td>
</tr>
<tr>
<td>Dependence overcome by</td>
<td>DSUB R8, R1, R7</td>
<td>Comparators detect use of R1 in DSUB and forward result of load to ALU in time for DSUB to begin EX.</td>
</tr>
<tr>
<td>with accesses in order</td>
<td>OR R9, R1, R7</td>
<td>No action required because the read of R1 by OR occurs in the second half of the ID phase, while the write of the loaded data occurred in the first half.</td>
</tr>
</tbody>
</table>

### What Makes Pipelining Difficult to Implement

- Exceptions
  - Interruptions to “normal” program flow
  - Usually outside of program’s control
  - Common “mechanism”
    - Save processor state (PC, flags etc.)
    - Handle the exception
    - Resume execution from point of interruption

### Exceptions

- Common mechanism for variety of reasons
  - I/O device request
  - User program invoking OS call
  - Breakpoints
  - Integer arithmetic overflow
  - Page fault
  - Misaligned memory address reference
  - Memory protection violation
  - Undefined instruction

---

**Figure A.23** Forwarding of results to the ALU requires the addition of three extra inputs on each ALU multiplexer and the addition of three paths to the new inputs. The paths correspond to a bypass of (1) the ALU output at the end of the EX, (2) the ALU output at the end of the MEM stage, and (3) the memory output at the end of the MEM stage.
Exceptions

- Variety of names used for different systems
  - Exception
  - Trap
  - Fault
  - Interrupt
  - Machine check

Exceptions

- Can be characterized along 5 independent axes:
  - Synchronous vs. Asynchronous
  - User requested vs. Coerced
  - User maskable vs. user nonmaskable
  - Within vs. between instructions
  - Resume vs. terminate

Exceptions

- Most difficulty arises with exceptions that occur within instructions and which must permit resumption of the program
  - Requires “restartable” pipeline
    - Instruction is “in execution” and has been “partly executed”
    - Succeeding instructions may also be in execution
  - “Precise” vs. “Imprecise” exceptions
    - Precise: pipeline can be “stopped” such that instructions just before the faulting instruction are completed but the faulting instruction and those following are able to be restarted correctly

Exceptions in MIPS

<table>
<thead>
<tr>
<th>Pipeline Stage</th>
<th>Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>Page fault on instruction fetch</td>
</tr>
<tr>
<td>IF</td>
<td>Misaligned memory access</td>
</tr>
<tr>
<td>IF</td>
<td>Memory protection violation</td>
</tr>
<tr>
<td>ID</td>
<td>Undefined/illegal opcode</td>
</tr>
<tr>
<td>EX</td>
<td>Arithmetic exception</td>
</tr>
<tr>
<td>MEM</td>
<td>Page fault on data fetch</td>
</tr>
<tr>
<td>MEM</td>
<td>Misaligned memory reference</td>
</tr>
<tr>
<td>MEM</td>
<td>Memory protection violation</td>
</tr>
<tr>
<td>WB</td>
<td>None</td>
</tr>
</tbody>
</table>
Exceptions in MIPS

• Multiple exceptions can occur in same clock cycle:
  
  LD R2, 100(R3) ; page fault in MEM
  ADD R5, R6, R7 ; arithmetic exception in EX

• Exceptions may occur out of order:
  
  LD R2, 100(R3) ; page fault in MEM
  ADD R5, R6, R7 ; page fault in IF (occurs earlier)