Chapter 3

Pipelining (continued)

Hazards

• Structural Hazard
• Data Hazard
• Control Hazard

Control Hazard

• Caused by PC-changing instructions
  (Branch, Jump, Call/Return)

<table>
<thead>
<tr>
<th>Branch sequence</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch sequence ≤ 1</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WR</td>
</tr>
<tr>
<td>Branch sequence ≥ 2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WR</td>
</tr>
<tr>
<td>Branch sequence ≥ 3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WR</td>
</tr>
<tr>
<td>Branch sequence ≥ 4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WR</td>
</tr>
<tr>
<td>Branch sequence ≥ 5</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WR</td>
</tr>
</tbody>
</table>

For 5-stage pipeline, 3 cycle penalty
30% branch frequency. CPI = 1.9
Branches Taken

- 67% of conditional branches are taken
- Can compute the forward and backward branch frequency
  - 60% of forward branches are taken
  - 85% of backward branches are taken

Branch Performance

Solutions to Control Hazard

- Optimized branch processing
- Branch prediction
- Delayed branch

Optimized Branch Processing

1. Find out branch taken or not early → simplified branch condition
2. Compute branch target address early → extra hardware
**Optimized Branch Processing (DLX)**

1. Find out branch taken or not early
   → test register value with 0 at end of ID cycle
2. Compute branch target address early
   → extra adder required

- **IF Stage**
  \[
  \text{IF/ID.exe} = \text{IF/ID.exe}_{\text{old}} \text{op} (\text{IF/ID.exe}_{\text{old}}) \quad \text{for} \quad (\text{IF/ID.exe}_{\text{old}} \neq \text{branch}) \text{and} \quad (\text{IF/ID.exe}_{\text{old}} \neq \text{branch})
  \]
  \[
  \text{IF/ID.exe}_{\text{old}} = \text{IF/ID.exe}_{\text{old}} \quad \text{for} \quad (\text{IF/ID.exe}_{\text{old}} = \text{branch})
  \]

- **ID Stage**
  \[
  \text{ID.EX.A} = \text{Reg}[\text{IF/ID.IR}_{6..10}]; \quad \text{ID.EX.B} = \text{Reg}[\text{IF/ID.IR}_{11..15}];
  \quad \text{ID.EX.IR} = \text{IF/ID.IR}; \quad \text{ID.EX.imm} = \text{IF/ID.IR}_{16..31}
  \]

---

**Solutions to Control Hazard**

- Optimized branch processing
- Branch prediction
- Delayed branch

---

**Modified DLX Pipeline Stages**

**Branch Prediction**

- **Predict-not-taken**
  - Simplest scheme
  - Have to be able to back out if branch is taken
- **Predict-taken**
  - No benefit in DLX
  - Useful if target is known prior to condition
Solutions to Control Hazard

- Optimized branch processing
- Branch prediction
- Delayed branch

Delayed Branch

- Semantics of delayed branch
  - Branch-delay slot
  - Sequential successor instruction

<table>
<thead>
<tr>
<th>Untaken branch instruction</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch delay instruction (i + 1)</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction + 2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction + 3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction + 4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Taken/branch instruction</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch delay instruction (i + 1)</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Branch target</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Branch target + 1</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Branch target + 2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

(a): Branch must not depend on the rescheduled instructions.
→ Always improves performance

(b): Must be OK to execute rescheduled instructions if branch is not taken. May need to duplicate instructions.
→ Improves performance when branch is taken. May enlarge program if instructions are duplicated.
(c): Must be OK to execute instructions if branch is taken.

→ Improves performance when branch is not taken.

**Delayed Branch**

- Hardware Assists: Cancelling or Nullifying branch
- Instruction includes direction of branch prediction
- Turn branch-delay slot to no-op if prediction is wrong

---

**What makes pipelining hard to implement?**

- If you thought it was bad up to here
- Life can get worse!!
Exceptions
- I/O Device Request
- Invoking an Operating System service request
- Tracing instruction execution
- Breakpoint
- Integer Arithmetic Overflow or underflow
- FP arithmetic anomaly
- Page fault
- Misaligned memory access
- Memory Protection Violation
- Using undefined instruction
- Hardware malfunction
- Power failure

Exception Characteristics
- Synchronous vs. Asynchronous
- User requested vs. coerced
- User maskable vs. unmaskable
- Within vs. between instructions
- Resume vs. terminate

Interrupt Handling

Interrupts in Pipelined Machines
- Interrupts occur within instructions
- Must be restartable

To restart, the state of the machine must be saved at a point from which the systematic execution of instructions can begin.
Saving State

- Force a trap instruction into the pipeline on the next IF
- Until the trap is taken, turn off all writes for the faulting instruction and all instructions that follow in the pipeline.
  - This prevents any state changes for instruction that will not be completed before the interrupt is handled
- After the interrupt handling routine gets control, save PC of the faulting instruction

Precise Exceptions

- When the pipeline can be stopped so that the instructions just before the faulting instruction are completed and those after it can be restarted from scratch.
- Simplifies the operating system interface.
- Precise exceptions may be supported in hardware or with some software support.
  - In some cases, difficult to implement because instructions may change the state before they are guaranteed to complete.
  - Leads to decreased instruction parallelism - Performance

Two Modes

- Some high performance machines have two modes of operation regarding interrupt handling
  - Precise interrupt mode
  - Imprecise interrupt mode
  - Gives higher performance
- In machines such as Alpha 21064, Power-2, MIPS R8000, precise mode is more than ten times slower!!

Precise Exceptions in DLX

- Multiple exceptions may occur in one cycle
- Exceptions may occur out of order

<table>
<thead>
<tr>
<th>Stage</th>
<th>Problem exceptions occurring</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>Page fault on instruction fetch,</td>
</tr>
<tr>
<td></td>
<td>Memory protection violation,</td>
</tr>
<tr>
<td></td>
<td>Misaligned access</td>
</tr>
<tr>
<td>ID</td>
<td>Undefined or 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></td>
<td>Misaligned memory access,</td>
</tr>
<tr>
<td></td>
<td>Memory protection violation</td>
</tr>
<tr>
<td>WB</td>
<td>None</td>
</tr>
</tbody>
</table>
Precise Exception Violation

- Interrupts can take place as soon as they occur
  - Interrupts occur in order different from the order or the instructions

**Instruction Page Fault**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>IP</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction-1</td>
<td>IP</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction-2</td>
<td>IP</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction-3</td>
<td>IP</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Instruction-4</td>
<td>IP</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

Data Page Fault

- Implementing Precise Exceptions

- Hardware posts each interrupt in a status vector
  - Instruction carries the vector in the pipe
  - Vector is checked when instruction enters the WB stage
  - If any interrupts are posted they are handled in the time order
  - All writes are prevented once interrupt flag has been entered
  - Guarantees that all interrupts of instruction i are seen before any of instruction i+1.

Instruction Set Complications

- When an instruction is guaranteed to complete it is called committed
- In DLX all instructions are committed once they complete MEM stage - No instruction updates the state before that stage
- Complications:
  - If state is changed in the middle
    - Example - autoincrement instructions
  - If memory is updated in the middle of the execution
    - Machines using condition codes set implicitly
  - Multicycle operations

Multicycle Operations: Floating Point Operations for DLX

- Some floating point operations may take long time
- Pipeline operations have to be carried out still
- Couple of changes for DLX
  - EX cycle may be repeated multiple times
  - May have several functional units
DLX Pipeline
with 3 unpipelined floating point units

Multiple EX Units
• Execution stages are not pipelined
• No other instruction can enter the functional unit until the first one is done
• If an instruction can not proceed - stall those behind it
→ Quite restrictive; need to generalize

Latency and Repeat Interval
• Latency
  – Number of intervening cycles between an instruction that produces a result and an instruction that uses the result
  – Usually, number of stages after EX that an instruction produces a result (or, 1 minus depth of EX stages)
• Repeat Interval or Initiation Interval
  – Number of cycles that must elapse between issuing two operations of a given type

<table>
<thead>
<tr>
<th>Functional Unit</th>
<th>Latency</th>
<th>Repeat Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer ALU</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Data memory (integer &amp; FP Loads)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FP Add</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FP (Integer) Multiply</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>FP (Integer) Divide</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>

Latency is the number of stage after EX that an instruction produces results

Pipeline supporting multiple FP operations
Hazards and Forwarding

- Divide unit is not pipelined - structural hazard
- As instructions have varying running times, more than one register writes may be required in a cycle
- WAW hazards are possible since instructions no longer reach WB stage in order
- Instructions complete in different order - exception problems
- Long latency leads to frequent RAW hazards

Precise Exceptions

<table>
<thead>
<tr>
<th>DIVF</th>
<th>F6, F2, F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADF</td>
<td>F10, F10, F8</td>
</tr>
<tr>
<td>SUBF</td>
<td>F12, F12, F14</td>
</tr>
<tr>
<td>Finish before DIVF</td>
<td></td>
</tr>
</tbody>
</table>

- Out of order completion causes imprecise exception conditions.
- Four approaches
  - Ignore the problem
  - Have two modes of execution
  - Buffer the results until all operations issued earlier are done
    - Requires large number of comparators and a large MUX
    - CYBER 180/980 uses a history file which keeps track of the original value of a register
    - Use future file which keeps the newer value of a register
  - Let exceptions to become somewhat imprecise
    - Let trap handling routines make the results precise
  - Issue only after ascertaining that all earlier instructions will complete

Instruction Set Design and Pipelining: Complications lead to inefficient pipelining

- Variable instruction length and running times
  - lead to
    - Imbalance among stages
    - Complicate hazard detection and precise exceptions
    - Caches have similar effect
    - Machines freeze pipeline
- Complex addressing modes
  - Update registers, e.g. autoincrement/decrement
  - Multiple memory accesses
- Allow writes into instruction space
  - Self modifying instructions cause pipelining problems
- Implicitly set condition codes

EXAMPLE - MIPS 4000

- 64 bit instruction set (MIPS-3 instruction set)
- Uses deeper pipeline
- Uses clock rate of 100 - 200 MHz
- Decomposes memory accesses into stages
  - Approach is known as SUPERPIELINING
Example - MIPS R4000

Pipeline Stages
- IF - First half of instruction fetch
- IS - Second half of instruction fetch
- RF - Instruction decode and register fetch
- EX - Execution and
  - effective address calculation
  - ALU operation
  - branch target computation
  - condition evaluation
- DF - Data fetch - first half
- DS - Data fetch - second half
- TC - Tag check - determine if data cache access hit
- WB - Write back for loads and register-register operations

Branch Delay of 3 cycles

MIPS Pipeline with 2 cycle load delay