Basic Pipelining
Long Instructions & MIPS Case Study

Complications With Long Instructions

• So far, all MIPS instructions take 5 cycles

• But haven't talked yet about the floating point instructions

• Take it on faith that floating point instructions are inherently slower than integer arithmetic instructions
How Slow Is Slow?

• Some typical times:
  – **Latency** is the number of cycles between an instruction that produces a result and one that uses it
  – **Initiation interval** is the number of cycles between two instructions of the same kind (for example, two ADD.Fs)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Latency</th>
<th>Initiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU uses</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Load/store</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD.F, SUB.F</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DIV.F</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>

Examples

• If we have a sequence of integer instructions
  – ADD
  – SUB
  – AND
  – OR
  – SLLI

• Then there are no delays in the pipeline, because
  – Initiation=1 means can start one of these instructions every cycle
  – Latency=0 means that results from one instruction will be available when the next instruction needs them
Examples (cont.)

- If we have a sequence of floating point instructions
  - ADD.F
  - SUB.F
- Then initiation=1 means that can start SUB.F one cycle behind ADD.F
- But latency=3 means that this will work right only if SUB.F doesn’t need ADD.F’s results
- If it does need the results, then need 3 instructions in between ADD.F and SUB.F to prevent bubbles in the pipeline

Functional Units

© 2003 Elsevier Science (USA). All rights reserved.
Hazards Caused By Long Instructions

- The floating point adder and multiplier are pipelined, but the divider is not - that is why the initiation interval for divide is 25
  - A program will run very slowly if it does too many of these!
- It will also run slowly if the results of the divide are needed too soon
### FP Stalls From RAW Hazards

<table>
<thead>
<tr>
<th>Inst.</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>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F4,0(R2)</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL.D F0,F4,F6</td>
<td>IF</td>
<td>ID</td>
<td>stall</td>
<td>M1</td>
<td>M2</td>
<td>M3</td>
<td>M4</td>
<td>M5</td>
<td></td>
</tr>
<tr>
<td>ADD.D F2,F0,F8</td>
<td>IF</td>
<td>stall</td>
<td>ID</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.D F2,0(R2)</td>
<td>stall</td>
<td>IF</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inst.</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL.D</td>
<td>M6</td>
<td>M7</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>stall</td>
<td>stall</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>S.D</td>
<td>stall</td>
<td>stall</td>
<td>ID</td>
<td>EX</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>MEM</td>
</tr>
</tbody>
</table>

### Long Instructions (cont.)

- It is possible that two instructions enter the WB stage at the same time

<table>
<thead>
<tr>
<th>ADD.D</th>
<th>IF</th>
<th>ID</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>IF</td>
<td>ID</td>
<td>ALU</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DADD</td>
<td>IF</td>
<td>ID</td>
<td>ALU</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DADD</td>
<td>IF</td>
<td>ID</td>
<td>ALU</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- A structural hazard
Long Instructions (cont.)

- Instructions can finish in the wrong order
- This can cause WAW hazards
- This violation of WB ordering defeats the previous strategy for precise exception handling
  - problem is out-of-order completion

DIV.D F0, F2, F4
ADD R1, R1, R2
SUB.D F10, F12, F14

What happens if sub faults?
And then div?
What about R1?

WAW Structural Hazard

<table>
<thead>
<tr>
<th></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>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL.D F0,F4,F6</td>
<td>IF</td>
<td>ID</td>
<td>M1</td>
<td>M2</td>
<td>M3</td>
<td>M4</td>
<td>M5</td>
<td>M6</td>
<td>M7</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>...</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F2,F4,F6</td>
<td>IF</td>
<td>ID</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F2,0(R2)</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Possible Fixes

- Give up and just do **imprecise exception handling**
  - tempting, but very annoying to users
- **Delay WB until all previous instructions complete**
  - since so many instructions can be active, this is expensive - requires a lot of supporting hardware
- Write, to memory, a **history file** of register and memory changes so can undo instructions if necessary
  - or keep a **future file** of computed results that are waiting for MEM or WB

Possible Fixes (cont.)

- **Let the** exception handler **finish the instructions** in the pipeline and then restart the pipe at the next instruction
- **Have the floating point units** diagnose exceptions in their first or second stages, so **can handle them by methods** that work well for handling integer exceptions
How To Detect Hazards In ID

• Early detection would prevent trouble
• Check for structural hazards:
  – will the divide unit clear in time?
  – will WB be possible when we need it?
• Check for RAW data hazards:
  – will all source registers be available when needed?
• Check for WAW data hazards:
  – Is the destination register for any ADD.D, multiply or divide instruction the same register as the destination for this instruction?
• If anything dangerous could happen, delay the execute cycle so no conflict occurs

Review – MIPS Instruction Format

Register-Register

```
<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td></td>
<td></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>
```

\[ rd \leftarrow rs \text{ OP } rt \]

Examples

```
ADD R1,R2,R3 \quad // R1 \leftarrow R2+R3
MUL R1,R2,R3 \quad // R1 \leftarrow R2*R3
```

Register-Immediate

```
<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>15</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>immediate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

\[ rt \leftarrow rs \text{ OP } \text{immed} \]

Examples

```
ADDI R1,R2,8 \quad // R1 \leftarrow R2+8
LW R1,4(R2) \quad // R1 \leftarrow \text{MEM} (R2+4)
SW R1,4(R2) \quad // \text{MEM} (R2+4) \leftarrow R1 \quad \ldots \leftarrow rs \text{ OP } rt
```
Review – Pipeline Registers

Pipeline register’s instruction register (IR) fields: IR[op] IR[rs] IR[rt] IR[rd]

Data Hazard Without Forwarding

Time (clock cycles)

lw r1,4(r2)
add r3,r1,r4
Data Hazard With Forwarding

Time (clock cycles)

lw r1,4(r2)
add r3,r1,r4

RAW Data Hazard Detection (LW/ADD)

Time (clock cycles)

lw r1,4(r2)
add r3,r1,r4

At time = cycle 2
IF/ID.IR[op] = ADD
IF/ID.IR[rs] = r1
IF/ID.IR[rt] = r4
IF/ID.IR[rd] = r3

ID/EX.IR[op] = LW
ID/EX.IR[rs] = r2
ID/EX.IR[rt] = r1

So insert pipeline stall due to RAW hazard if
ID/EX.IR[op] = LW
IF/ID.IR[op] = ADD
ID/EX.IR[rt] = IF/ID.IR[rs]
OR
ID/EX.IR[rt] = IF/ID.IR[rt]
RAW Data Hazard Detection (LW/LW)

- **Time (clock cycles)**

  1. `lw r1,4(r2)`
  2. `lw r3,8(r1)`

At time = cycle 2

- `IF/ID.IR[op] = LW`
- `IF/ID.IR[rs] = r1`
- `IF/ID.IR[rt] = r3`

- `ID/EX.IR[op] = LW`
- `ID/EX.IR[rs] = r2`
- `ID/EX.IR[rt] = r1`

So insert pipeline stall due to RAW hazard if

- `ID/EX.IR[op] = LW`
- `IF/ID.IR[op] = LW`
- `ID/EX.IR[rs] = IF/ID.IR[rt]`

WAW Data Hazard Detection (MUL.D/ADD.D)

- **MUL.D r1,r2,r3**
- **ADD.D r1,r4,r5**

If at time x

- `IF/ID.IR[op] = ADD.D`
- `IF/ID.IR[rd] = r1`
- `M1/M2.IR[op] = MUL.D`
- `M1/M2.IR[rd] = r1`

So insert pipeline stall due to WAW hazard if

- `M1/M2.IR[op] = MUL.D`
- `IF/ID.IR[op] = ADD.D`
- `M1/M2.IR[rd] = IF/ID.IR[rd]`

Then ADD.D would write to r1 at time x+6, while MUL.D would write r1 at time x=7, reversing order of writes.
A Case Study: MIPS R4000

- **MIPS R4000**
  - Introduced 1991, one of the first 64-bit CPUs
  - Sony PSP (2004) used 0.3 GHz R4000
- **Deep 8 stage pipeline**
  - to get higher clock rates
  - extra stages come from memory accesses
  - techniques called superpipelining

MIPS R4000 Pipeline Stages

- **IF** – 1st half instruction fetch
  - PC selection and start instruction cache access
- **IS** – 2nd half instruction fetch
  - complete instruction cache access
- **RF** – instruction decode, register fetch, hazard checking, instruction cache hit detection
- **EX** – execution
  - includes effective address computation, ALU operation, branch target computation and condition evaluation
MIPS R4000 Pipeline (cont.)

- DF – 1st half data fetch
  - 1st half of data cache access

- DS – 2nd half data fetch
  - complete data cache access

- TC – tag check
  - determine whether data cache access hit

- WB – write back for loads and ALU operations

A 2 cycle load delay
MIPS R4000 Pipeline (cont.)

A 3 cycle branch delay – 1 delay slot + 2 cycle stall for taken branch (untaken just delay slot)

Forwarding

- Deeper pipeline increases number of levels of forwarding for ALU operations
  - 4 possible sources for an ALU bypass
    » EX/DF
    » DF/DS
    » DS/TC
    » TC/WB
Floating Point Pipeline

• 3 functional units
  – divider, multiplier, adder
• Double precision FP ops take
  – from 2 (negate) up to 112 cycles (square root)
• Effectively 8 stages, combined in different orders for various FP operations
  – one copy of each stage, and some instructions use a stage zero or more times, and in different orders
• Overall, rather complicated …
  – see H&P for more details

R4000 Pipeline Performance

• 4 major causes of pipeline stalls
  – load stalls – from using load result 1 or 2 cycles after load
  – branch stalls – 2 cycles on every taken branch, or empty branch delay slot
  – FP result stalls – RAW hazards for an FP operand
  – FP structural stalls – from conflicts for functional units in FP pipeline
SPEC92 Benchmarks

Assuming a perfect cache – 5 integer and 5 floating-point programs

Pitfalls

- **Unexpected hazards do occur …**
  - for example, when a branch is taken before a previous instruction finishes
- **Extensive pipelining can slow a machine down, or lead to worse cost-performance**
  - more complex hardware can cause a longer clock cycle, killing the benefits of more pipelining
Pitfalls (cont.)

- A poor compiler can make a good machine look bad
  - compiler writers need to understand the architecture in order to
    » optimize efficiently and
    » avoid hazards
  - better to eliminate useless instructions, than make them run faster