Pipelining Wrapup

- Brief overview of the rest of chapter 3
  - Exceptions and the pipeline
  - Multicycle pipelines: Floating Point

Exceptions

- An exception is when the normal execution order of instructions is changed. This has many names:
  - Interrupt
  - Fault
  - Exception
- Examples:
  - I/O device request
  - Invoking OS service
  - Page Fault
  - Malfunction
  - Undefined instruction
  - Overflow/Arithmetic Anomaly
  - Etc!

Exception Characteristics

- Synchronous vs. asynchronous
  - Synchronous when invoked by current instruction
  - Asynchronous when external device
- User requested vs. coerced
  - Requested is predictable
- User maskable vs. non-maskable
  - Can sometimes ignore some interrupts, e.g. overflows
- Within vs. Between Instructions
  - Exception can happen anywhere in the pipeline
- Resume vs. Terminate
  - Terminate if execution stops, resume if we need to return to some code and restart execution, must store some state

Stopping/Restarting Execution

- DLX – occurs in MEM or EX stages
- Pipeline must be shut down
  - PC saved for restart
  - Branches must be re-executed, condition code must not change
- DLX steps to restart
  - Force trap instruction into pipe on next IF
  - Erase following instructions by writing all 0's to pipeline latches
  - Allow preceding instructions to complete if possible
  - Let all preceding instructions complete if they can; this freezes the state at the time the exception is handled
  - After OS exception handling routine starts, it must save the PC of the faulting instruction
Complications

- Saving the single PC sometimes isn’t enough
- Using delayed branches, given two delay slots
  - Both delay slots contain branch instructions
    - Recall with delayed branches, we’ll always execute the instructions in the delay slots
  - Say there is an exception processing the 1st delay slot; the 2nd delay slot is erased
  - Upon return, the restart position is the PC which becomes the 1st delay slot
    - We’ll then continue to execute the 2nd delay slot instruction AND the following instruction!
  - Complication arises from interaction with effective ordering in the delayed branch
    - Solution: save needed delay slots and PC

---

DLX Exceptions

<table>
<thead>
<tr>
<th>Pipeline Stage</th>
<th>Exception Possibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>Page fault on IF, misaligned memory access, memory-protection violation</td>
</tr>
<tr>
<td>D</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; misaligned memory access; memory protection violation</td>
</tr>
<tr>
<td>WB</td>
<td>None</td>
</tr>
</tbody>
</table>

---

MultiCycle Operations

- Unfortunately, it is impractical to require all DLX floating point operations to complete in one clock cycle (or even two)
  - Could, but it would result in a seriously slow clock!
  - Consider we do this and we have the following units:
    - Integer EX
    - FP Multiple
    - FP Add
    - FP Divide
  - The FP units merely require multiple cycles to complete

---

Unpipelined FP Units

<table>
<thead>
<tr>
<th>Unit</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>0</td>
</tr>
<tr>
<td>FPAdd</td>
<td>3</td>
</tr>
<tr>
<td>FPMult</td>
<td>6</td>
</tr>
<tr>
<td>FPDiv</td>
<td>24</td>
</tr>
</tbody>
</table>

Solution: Pipeline FP units
Pipelined FP Units

Allows 4 outstanding adds, 7 multiplies, 1 int, 1 divide

Not pipelined
Need 24 cycles

New Hazard Problems!

- Structural hazards with divide unit not fully pipelined
- WAW hazards now possible since instructions can reach WB stage at different times
  - At least WAR hazards not possible, since reads still occur early in the ID stage
- Instructions can complete in a different order than issued, causing more problems with exception handling
- Longer latency increases frequency of stalls for RAW hazards

How would you tell if the efforts here are worth it?

Example FP Sequence with RAW Hazard

Instruction 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
LD F4, 0(R2) IF ID EX MEM WB
MULTD F0, F4, F6 IF ID M1 M2 M3 M4 M5 M6 M7 MEM WB
ADDD F2, F0, F8 IF ID A1 A2 A3 A4 MEM WB
Sd 0(R2), F2 IF ID A1 A2 A3 A4 MEM WB

Uses forwarding for each stage when data is available
SD stalled one extra cycle for MEM to not conflict with ADDD

Example FP Sequence with Hazards

Instruction 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
MULTD F0, F4, F6 IF ID M1 M2 M3 M4 M5 M6 M7 MEM WB
...
ADDD F2, F4, F6 IF ID EX MEM WB
...
LD F2, 0(R2)

Cycle 9: three requirements for memory
Cycle 11: three requirements for write-back
More stalls

What if the last instruction was issued one cycle earlier?
We have a WAW conflict
WinDLX Code Example

.data
.align 4
X: .byte 50,50,23,25 ; Random FP Number

.text
.global main

main:
  lf f1, X
  divf f1, f1, f1
  addi r2, r0, #3
  lf f1, X

Finish: ;*** end
  trap 0

FP Pipelining Performance

- Given all the new problems, is it worth it?
  - See book for details
  - Overall answer is yes
    - Latency varies from 46-59% of functional units on the benchmarks
    - Fortunately, divides are rare
    - As before, compiler scheduling can help a lot

Try inserting other addi’s here!

Causes WAW stall