Chapter 4.4 and 4.5

Principles of pipelining

Pipeline hazards

Remedies

Multi-stage process

Sequential execution
  - One process begins after previous finishes
  - E.g., six stage process; each stage takes 5 sec
  - Time to complete is 30 sec
  - Throughput = 2/minute

Pipelining
  - Begin a stage of the next process as soon as the stage of the previous process is finished
  - Third car in soap, second car in wash, first car in dry
  - Time to complete the wash is ?
  - Throughput is =

Pipelining for instruction execution

Each Instruction has several sub-steps
Each instruction has 5 or 6 stages
- IF, ID, EX, MEM, WB
- Multiple cycles per instruction
- Typical cycle time (1 GHz Processor – 1 nsec)
- Average instruction can take 3 to 5 cycles
- Sequential execution of one instruction after another
- Too slow
- Alternatives?

Multicycle

Multicycle Seq implementation:
Laundry example (Wash, dry, fold, store)

Ideal Pipelining

<table>
<thead>
<tr>
<th>Cycle</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>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+1</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+2</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+3</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+4</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ideal throughput: 1 instruction/cycle

Pipeline hazards

- Car wash example
  - One stage (soap) may take longer (additional rust proof)
  - Car behind (soak stage) need to idle
  - Pipeline stall
- Clothes wash example
- Resource may not be available
  - Washer-dryer combo
  - Resource shared and in use
  - Dried or washed clothes not emptied
  - Cycle taking longer
  - Pipeline stall

Pipelining hazards in CPU

- Hazards: A stage of the instruction cannot execute in the next clock cycle
- Three types of hazards
- Structural hazards
  - Two different instructions use the same hardware
  - Resource conflict (e.g., one memory port)
- Data hazards
  - Two different instructions have a dependency on data
  - Two instructions use the same register/memory
- Control hazards
  - One instruction’s execution affects which instruction is next
  - E.g., jmp
  - Instruction that affects PC (%eip)
Instruction Order

MOV 100(%ebp), %eax
MOV 200(%ebp), %ebx
MOV 300(%ebp), %ecx
MOV 400(%ebp), %edx

Structural Hazard

- Cant read memory twice in the same cycle

Dealing with Structural hazards

- Stall
  - Block until previous operation completes
  - Decreases throughput or Ins/cycle
- Replicate resource
  - Separate memory banks
  - Dual port memory
  - Can read different addresses in the same cycle
  - Need H/W, cost
- Pipeline the resource
  - Can be used for multi-cycle resource

Pipeline Data Hazards

- Data hazards
  - An instruction uses the result of a previous instruction
    - ADD %eax, %edx or MOV %eax, 100(%ebp)
    - MOV %eax, %edx

Data Hazards (RAW)

- Cycle
  - Write Data to %eax Here
  - Read from %eax Here
  - ADD %ebx, %eax
  - ADD %eax, %ebx
Dealing with data hazards

- Stalling
  - Hardware detects RWA conflicts and stalls
- Forwarding
  - Connect new value directly to next stage
  - Make data available internally before it is stored

Data Hazards - Stalling

Data Hazards - Forwarding

- Key idea: connect new value directly to next stage
- Input to next stage is the new result
- Problem: what about move from memory instructions?

Data Hazards - Forwarding

- Data to be loaded in %eax available only after memory read of location 100 + (%ebp)
Data hazards; Compiler optimization

With loads, data available only after M stage; requires stalls
Compiler can reorder instructions
Execute instructions out of order to fill in for pipeline stalls

 mov -8(%ebp), %eax
 mov -12(%ebp), %edx
 add %edx, %eax
 mov %eax, 4(%ebp)
 mov -16(%ebp), %ecx
 mov %eax, 8(%ebp)

mov -8(%ebp), %eax
mov -12(%ebp), %edx
mov -16(%ebp), %ecx
add %edx, %eax
mov -20(%ebp), %edx
mov %eax, 4(%ebp)
sub %ecx, %eax
mov %eax, 8(%ebp)

With loads, data available only after M stage; requires stalls
Compiler can reorder instructions
Execute instructions out of order to fill in for pipeline stalls

Other Types of Data Hazards

- WAW (write after write)
  - variable-length pipeline
  - EX can take 3 cycles
  - later instruction must write after earlier instruction writes
- WAR (write after read)
  - Variable length pipelines
  - Instructions with late read or memory access
  - later instruction must write after earlier instruction reads

Pipeline Data Hazards

- Control hazards
  - the location of an instruction depends on a previous instruction

We can't fetch the next instruction until we know the branch address
Branch taken PC ← Immediate
Branch not taken PC ← PC + 4

Control Hazards: Need to know which to fetch
Control Hazards: Assume branch not taken

Control Hazards: In Some CPUs, address available after EX stage

Summary

- Pipelining principles
- Hazards
- Hazard prevention