Lecture Topics

- Today: Pipelined Processors
  (P&H 4.5-4.10)
- Next: continued

Announcements

- Exam #1 – unclaimed exams available
- Milestone #5 (due 3/2)
- Milestone #6 (due 3/16)
Pipeline Summary

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Instruction set design affects complexity of pipeline implementation
- Subject to hazards
  - Structural, data, control

MIPS Pipelined Datapath

Right-to-left flow leads to hazards
Pipelined Control

- Control signals are the same as before
- Used in different pipeline stages

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Effect when deasserted (0)</th>
<th>Effect when asserted (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegDst</td>
<td>The register destination number for the Write register comes from the id field (bits 20-15).</td>
<td>The register destination number for the Write register comes from the id field (bits 15-11).</td>
</tr>
<tr>
<td>RegWrite</td>
<td>None.</td>
<td>The register on the Write register input is written with the value on the Write data input.</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>The second ALU operand comes from the second register file output (Read data 2).</td>
<td>The second ALU operand is the sign-extended, lower 16 bits of the instruction.</td>
</tr>
<tr>
<td>PCSrc</td>
<td>The PC is replaced by the output of the adder that computes the value of PC + 4.</td>
<td>The PC is replaced by the output of the adder that computes the branch target.</td>
</tr>
<tr>
<td>MemRead</td>
<td>None.</td>
<td>Data memory contents designated by the address input are put on the Read data output.</td>
</tr>
<tr>
<td>MemWrite</td>
<td>None.</td>
<td>Data memory contents designated by the address input are replaced by the value on the Write data input.</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>The value fed to the register Write data input comes from the ALU.</td>
<td>The value fed to the register Write data input comes from the data memory.</td>
</tr>
</tbody>
</table>
Pipelined Control

- Control signals derived from instruction (as with single-cycle implementation)

Pipelined Control

- Control signals carried forward to the stage where they are needed
Pipelined Control

- Control signals grouped by stage
- ALU Op (2 bits) – first step in two-level decoding (further decoding in EX stage)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Execute/address calculation stage control lines</th>
<th>Memory access stage control lines</th>
<th>Write-back stage control lines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RegDest</td>
<td>ALUOp1</td>
<td>ALUOp0</td>
</tr>
<tr>
<td>RSubtract</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>LN</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SH</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SUB</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Pipelined Control

- ALUOp (2 bits): derived from instruction
- funct (6 bits): last 6 bits of instruction
- ALU operation (4 bits): control signal for ALU

<table>
<thead>
<tr>
<th>Instruct</th>
<th>ALUOp</th>
<th>Operation</th>
<th>funct</th>
<th>ALU function</th>
<th>ALU operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>00</td>
<td>load word</td>
<td>XXXXX</td>
<td>add</td>
<td>0010</td>
</tr>
<tr>
<td>sw</td>
<td>00</td>
<td>store word</td>
<td>XXXXX</td>
<td>add</td>
<td>0010</td>
</tr>
<tr>
<td>beq</td>
<td>01</td>
<td>branch equal</td>
<td>XXXXX</td>
<td>subtract</td>
<td>0110</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>add</td>
<td>100000</td>
<td>add</td>
<td>0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subtract</td>
<td>100010</td>
<td>subtract</td>
<td>0110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AND</td>
<td>100100</td>
<td>AND</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR</td>
<td>100101</td>
<td>OR</td>
<td>0001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set-on-less-than</td>
<td>101010</td>
<td>set-on-less-than</td>
<td>0111</td>
</tr>
</tbody>
</table>

Data Hazards in ALU Instructions

- Consider this sequence:

  sub $2, $1, $3  
  and $12, $2, $5  
  or $13, $6, $2  
  add $14, $2, $2  
  sw $15, 100($2)

- Hazards can be resolved with forwarding
- Additional logic to detect when to forward
Dependencies & Forwarding

- Pass register numbers along pipeline:
  - ID/EX.RegisterRs = Rs register number in ID/EX pipeline register
  - EX/MEM.RegisterRd = Rd register number in EX/MEM pipeline register

- ALU operand register numbers in EX stage:
  - ID/EX.RegisterRs
  - ID/EX.RegisterRt

Detecting the Need to Forward
Data hazard when:

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

From the example:

```
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
```

Forward from EX/MEM pipeline register to resolve 1a and 1b

Forward from MEM/WB pipeline register to resolve 2a and 2b

But only if forwarding instruction will write to a register

And only if destination register for that instruction is not $0
Forwarding Paths

Forwarding Conditions

- **EX hazard**

  if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
  
  ForwardA = 10

  if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
  
  ForwardB = 10
Forwarding Conditions

- MEM hazard

\[
\text{ForwardA} = 01 \\
\text{ForwardB} = 01
\]

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs))

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt))

Double Data Hazard

- Consider the sequence:

  add \$1, \$1, \$2
  add \$1, \$1, \$3
  add \$1, \$1, \$4

- Both hazards occur: use the most recent value

- Revise MEM hazard condition: only forward if EX hazard condition is not true
MEM hazard

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite
    and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
ForwardA = 01
Datapath with Forwarding

Load-Use Data Hazard

Need to stall for one cycle
Load-Use Hazard Detection

- Check when instruction which uses the result is decoded in ID stage
- ALU operand register numbers in ID stage are IF/ID.RegisterRs, IF/ID.RegisterRt
- From the example:

  \( \text{lw} \ $2, 20($1) \)
  \( \text{and} \ $4, $2, $5 \)

Load-Use Hazard Detection

- Load-use hazard when

  \[
  \text{ID/EX.MemRead and ((ID/EX.RegisterRt = IF/ID.RegisterRs) or (ID/EX.RegisterRt = IF/ID.RegisterRt))}
  \]

- If detected, stall and insert bubble into the pipeline
Stalling the Pipeline

- Force control signals in ID/EX register to 0
  - instruction becomes no-op
  - as instruction advances through pipeline, next 3 stages (EX, MEM and WB) will do nothing

- Prevent update of PC and IF/ID register
  - Instruction with hazard is decoded again
  - Following instruction is fetched again

Stall/Bubble in the Pipeline

![Diagram showing the pipeline with a stall inserted at a specific point]

Stall inserted here
Stall/Bubble in the Pipeline

Datapath with Hazard Detection
Stalls and Performance

- Stalls reduce performance, but are required to get correct results.
- Compiler, assembler and/or programmer can arrange code to avoid hazards and stalls
  - Re-arrange instructions without changing the meaning of the program
  - Requires knowledge of the pipeline structure

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
Control Hazards

- Control transfer instructions create control hazards (branch hazards) in the pipeline
- The target address is not available until after at least one other instruction has entered the pipeline
- Handled by rearranging the pipeline and stalling the pipeline as needed

Example

```
beq $20, $21, labelA
add $1, $10, $11
sub $2, $10, $11
xor $3, $10, $11
nor $4, $10, $11
and $5, $10, $11
.
.
labelA:
addi $6, $10, 0x40
ori $7, $10, 0x40
```
Pipelined Implementation

- Target address and condition value (true/false) computed in EX stage, used in MEM stage

Pipelined Implementation

When condition is false (same as before)

```
beq $20, $21, labelA
add $1, $10, $11
sub $2, $10, $11
xor $3, $10, $11
nor $4, $10, $11
and $5, $10, $11
```

Pipelined Implementation

# When condition is true

- beq $20, $21, labelA
- add $1, $10, $11
- sub $2, $10, $11
- xor $3, $10, $11
- addi $6, $10, 0x40
- ori $7, $10, 0x40

Three more instructions have entered the pipeline before control transferred to target address.

Handling Control Hazards

- Stall the pipeline: nullify those three instructions (convert them to no-ops)
- Rearrange and stall: compute the target address and branch condition in the ID stage, nullify next instruction
- Rearrange and use delay slot: compute the target address and branch condition in the ID stage, leave next instruction alone
Control Hazards

- When branch outcome determined in MEM

Reducing Branch Delay

- Determine outcome in ID stage (instead of MEM stage)
  - Move target address adder
  - Register comparator (instead of using ALU to subtract the two operands)
Reducing Branch Delay

- Example: branch taken

36:   sub $10, $4, $8
40:   beq $1, $3, 7
44:   and $12, $2, $5
48:   or $13, $2, $6
52:   add $14, $4, $2
56:  slt $15, $6, $7

...  

72:   lw $4, 50($7)

Example: Branch Taken
Example: Branch Taken

- Hazard if a comparison register is a destination of 2\textsuperscript{nd} or 3\textsuperscript{rd} preceding ALU instruction

- Can resolve using forwarding
Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2\textsuperscript{nd} preceding load instruction, need to stall one cycle

\begin{verbatim}
  lw $1, addr  IF ID EX MEM WB
  add $4, $5, $6 IF ID EX MEM WB
  beq stalled   IF ID
  beq $1, $4, target IF ID EX MEM WB
\end{verbatim}

Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction, need to stall two cycles

\begin{verbatim}
  lw $1, addr  IF ID EX MEM WB
  beq stalled  IF ID
  beq stalled  IF ID
  beq $1, $0, target IF ID EX MEM WB
\end{verbatim}