Lecture Topics

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

Announcements

- Exam #1 returned today
- Milestone #4 (due 2/23)
- Milestone #5 (due 3/2)
- Milestone #6 (due 3/16)
Milestone #4

Shift Unit Control Signals:

SO -- shift unit operation code
V -- asserted when variable shift

Math Unit Control Signals:

MO -- math unit operation code
I -- invert second operand
M2 -- select source for second operand

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>contents of REG[rt]</td>
</tr>
<tr>
<td>1</td>
<td>sign-extended immediate-16 value</td>
</tr>
<tr>
<td>2</td>
<td>zero-extended immediate-16 value</td>
</tr>
<tr>
<td>3</td>
<td>appropriate value for LUI instructions</td>
</tr>
</tbody>
</table>
General Control Signals:

F1 -- rs field identifies a source operand
F2 -- rt field identifies a source operand
MR -- instr reads from the Data Memory
MW -- instr writes into the Data Memory
RW -- instr writes into the Register Unit
DR -- select destination register number
DO -- select operand to be written

DR (destination register):

Port 2:  rd
Port 1:  31
Port 0:  rt

DO (destination operand):

Port 3:  output of the Branch Unit
Port 2:  output of the Shift Unit
Port 1:  output of the Math Unit
Port 0:  output of the Memory Unit
MIPS Pipelined Implementation

Classic five-stage model:

- **IF:** Fetch instruction from memory
- **ID:** Decode instruction and read registers
- **EX:** Execute operation or calculate address
- **MEM:** Access memory operand
- **WB:** Write result back to register

- Pipeline registers capture results at end of stages
- All instructions forced into same framework
Example

Sequence of five MIPS instructions:

Example

Sequence of five MIPS instructions:
Pipeline Hazards

- Situations where there are conflicts between two instructions in the pipeline are called hazards

- Three categories:
  - Structural hazards
  - Data hazards
  - Control hazards

Structural Hazards

- A structural hazard occurs when the datapath does not contain the necessary resources to perform two operations at the same time

- Solution: add resources to the datapath (perhaps by replicating existing resources)
Structural Hazards

- Example: separate instruction cache and data cache
  
  If single cache, could not simultaneously fetch an instruction and perform the MEM stage of LW or SW instructions

- Example: separate adder for PC
  
  Without a separate adder, PC+4 would have to be done in the ALU; could not simultaneously update PC and perform the EX stage

Data Hazards

- A data hazard occurs when the result of one instruction is an input to the next instruction

- Solution: freeze early stages of the pipeline (*stall the pipeline*)

- Solution in some cases: use data forwarding
Example: Data Hazard

Second instruction dependent on first:

\[
\begin{align*}
\text{add} & \quad s0, t0, t1 \\
\text{sub} & \quad t2, s0, t3
\end{align*}
\]

Old value of $s0$

Delay for two clock ticks:

\[
\begin{align*}
\text{add} & \quad s0, t0, t1 \\
\text{nop} & \\
\text{nop} & \\
\text{sub} & \quad t2, s0, t3
\end{align*}
\]
Pipeline Bubbles

- Cause needed delays by inserting **bubbles** into the pipeline (cycles when no useful work is done in some stage)

- Several strategies for inserting bubbles
  - Programmer required to insert no-ops
  - Assembler inserts no-ops
  - Hardware freezes early stages and converts "nullified" instructions into no-ops

Hardware inserts two bubbles:

\[
\begin{align*}
\text{add} & \quad \$s0, \quad \$t0, \quad \$t1 \\
\text{sub} & \quad \$t2, \quad \$s0, \quad \$t3
\end{align*}
\]
Forwarding (Bypassing)

- Required value is sometimes available earlier in the pipeline
- Add pathways to forward the value

Program execution order (in instructions)

Example: Data Hazard

Second instruction dependent on value loaded from memory by first instruction:

```assembly
lw  $s0, 20 ($t1)
sub  $t2, $s0, $t3
```

Called a load-use data hazard
Required value is available after the MEM stage, so stall pipeline for one cycle and user forwarding (another new pathway)

Program execution order (in instructions)

lw $t0, 20($t1)
sub $t2, $t0, $t3
lw $t1, 0($t0)
lw $t2, 4($t0)
add $t3, $t1, $t2
sw $t3, 12($t0)
lw $t4, 8($t0)
add $t5, $t1, $t4
sw $t5, 16($t0)

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

```assembly
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
```
Single-Cycle Implementation

# When condition is false

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

Single-Cycle Implementation

# When condition is true

    beq  $20, $21, 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

```assembly
beq $20, $21, labelA
addi $1, $10, 0x40
ori $7, $10, 0x40
```

Three more instr
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