Notes and Updates

• **Office hours:** Tues 3:30-5, Wed 9-10:30
  - Come see me for midterm questions
  - I’ll be out of town Wed 10:30-Friday night

• **DECaF Job Fair**
  - Webboard
  - Extra point on HW for a resume!
  - This Friday Resume Workshop Price Center
    Gallery B 11-12 and 1-2

• New homework posted!

• Questions?

---

Mixed Instructions in the Pipeline

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summer REUs
(Research Experience for Undergraduates)

- NOW is the time!
- www.nsf.gov/crssprgm/reu/list_result.cfm?unitid=5049
- Alabama: pervasive/mobile computing
- Santa Cruz: IT
- Harvey Mudd: AI, Systems, Optical Networking
- Boulder: Cybersecurity
- Florida: Autonomous Robots
- Florida: Vision
- Ohio: Networking
- Chicago: Grid and Bioinformatics
- New Orleans: Spatio-Temporal Databases
- NIST
- Maryland: Human Computer Interaction
- Towson: Multidisciplinary
- Oakland: Women only
- Montana: Tribal students only
- Brunswick: International focus
- New Jersey: Radio
- Texas: Distributed Rational Agents
- Virginia: Apps for Medicine

Pipeline Principles

- All instructions that share a pipeline must have the same stages in the same order.
  - therefore, add does nothing during Mem stage
  - sw does nothing during WB stage
- All intermediate values must be latched each cycle.
- There is no functional block reuse
  - example: we need 2 adders and ALU (like in single-cycle)
Pipelined Datapath

Instruction Fetch  Instruction Decode/ Register Fetch  Execute/ Address Calculation  Memory Access  Write Back

The Pipeline in Execution

add $10, $1, $2  Instruction Decode/ Register Fetch  Execute/ Address Calculation  Memory Access  Write Back

add $10, $1, $2  lw $12, 1000($4)  sub $15, $4, $1
The Pipeline in Execution

IW $12, 1000($4)   LW $12, 1000($4)   SUB $15, $4, $1

ADD $10, $1, $2   ADD $10, $1, $2   SUB $15, $4, $1

Memory Access   Write Back

Address Calculation

The Pipeline in Execution

SUB $15, $4, $1   LW $12, 1000($4)   ADD $10, $1, $2

ADD $10, $1, $2   LW $12, 1000($4)   SUB $15, $4, $1

Memory Access   Write Back
The Pipeline in Execution

Instruction Fetch
sub $15, $4, $1
lw $12, 1000($4)
add $10, $1, $2
Write Back

add $10, $1, $2
lw $12, 1000($4)
sub $15, $4, $1

The Pipeline in Execution

Instruction Fetch
Instruction Decode/
Register Fetch
sub $15, $4, $1
lw $12, 1000($4)
add $10, $1, $2

add $10, $1, $2
lw $12, 1000($4)
sub $15, $4, $1
The Pipeline in Execution

Instruction Fetch  Instruction Decode/ Register Fetch  Execute/ Address Calculation

sub $15, $4, $1  lw $12, 1000($4)

add $10, $1, $2  lw $12, 1000($4)  sub $15, $4, $1

The Pipeline in Execution, with controls
Pipelined Control

- FSM isn’t really appropriate
- Combinational Logic (like single-cycle design)!

Self Check

- Why do we only need one register file? It is USED in two different stages of the pipeline...
The Pipeline with Control Logic

Translation: How do we show that program in this style picture?

add $10, $1, $2
lw $12, 1000($4)
sub $15, $4, $1
Is it really that easy?

- Suppose initially, for all registers
  - register i holds the number 2i
- What happens when we execute this program...

```
add $3, $10, $11
lw $r8, 50($3)
sub $11, $8, $7
```

The Pipeline in Execution

```
lw $8, 50($3)    add $3, $10, $11    Execute/  Memory Access  Write Back
           Address Calculation
```
Data Hazards

- When a result is needed in the pipeline before it is available, a “data hazard” can occur.

Three ways to handle remaining Data Hazards

- In Software
  - insert independent instructions (or no-ops)

- In Hardware
  - insert bubbles (i.e. stall the pipeline)
  - data forwarding
Dealing with Data Hazards in Software

Insert enough no-ops (or other instructions that don't use register 2) so that data hazard doesn't occur.

Where are No-ops needed?

- sub $2, $1, $3
- and $4, $2, $5
- or $8, $2, $6
- add $9, $4, $2
Handling Data Hazards in Hardware:
Pipeline Stalls

- To insure proper pipeline execution in light of register dependences, we must:
  - Detect the hazard
  - Stall the pipeline
    - prevent the IF and ID stages from making progress
      - the ID stage because we can’t go on until the dependent instruction completes correctly
      - the IF stage because we do not want to lose any instructions.
    - insert “no-ops” into later stages

Handling Data Hazards in Hardware
Stall the pipeline

- sub $2, $1, $3
- and $12, $2, $5
- or $13, $6, $2
- add $14, $2, $2
• What comparisons tell us when to stall?
  - What do we need to compare?

But that’s not it...

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
</tr>
</tbody>
</table>

- sub $2, $1, $3
- and $12, $5, $5
- or $13, $6, $2
- add $14, $2, $2
- sw $15, 100($2)
• What comparisons tell us when to stall?
  - What do we need to compare?

Actually Stalling the Pipeline

• Prevent the IF and ID stages from proceeding
  - don’t write the PC (PCWrite = 0)
  - don’t rewrite IF/ID register (IF/IDWrite = 0)

• Insert “nops”
  - set all control signals propagating to ID/EX/MEM/WB to zero
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2

PCWrite = 0
IF/ID IorDWrite = 0
Set control lines in ID 0 (nop)

The Pipeline

PCWrite = 0
IF/ID IorDWrite = 0
Set control lines IF/ID 0 (ID/EX 0 (comes from IF/ID))
PCWrite = 1
IF/ID IorDWrite = 1
ID/EX 0, EX/MEM 0 (comes from previous stages)
add $14, $2, $2
Self Check!

- I’ve got a new design for my hardware. Because I hired sub-par engineers, my pipeline breakdown takes 7 stages:
  - IF, ID, EX1, EX2, EX3, MEM, WB
- How many cycles must my hardware stall for:
  Sub $2, $3, $1
  And $12, $2, $5

Can you draw the all the “stall possibilities”?

IF, ID, EX1, EX2, EX3, MEM, WB

<table>
<thead>
<tr>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
<th>CC8</th>
<th>CC9</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub $2, $1, $3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and $12, $2, $5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or $13, $6, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $14, $2, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub $14, $2, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mult $14, $2, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reducing Data Hazards Through Forwarding

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
Is That All??

- How many hazards did we have to deal with in this pipeline?

sub $2, $1, $5

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

Reducing Data Hazards Through Forwarding
Data Forwarding

- Forwarding (just shown) handles two types of data hazards
  - EX hazard
  - MEM hazard
- We’ve already handled the third type (WB) hazard by using a transparent reg file
  - if the register file is asked to read and write the same register in the same cycle, the reg file allows the write data to be forwarded to the output.

Eliminating Data Hazards via Forwarding

Forwarding can grab the defined register value from anywhere in pipeline even though it has not yet been written to register file.
Does Forwarding eliminate all hazards??

lw $2, 10(51)

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)

NO! You may need to stall after loads

lw $2, 10($1)

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Try this one...

Show stalls and forwarding for this code
add $3, $2, $1
lw $4, 100($3)
and $6, $4, $3
sub $7, $6, $2

Some other specific forwarding situations

- Page 412 of book
- Loads forwarding to stores

- AddI not in the datapath shown in book, extra mux needed for it
Data Hazard Key Points

- Pipelining provides high throughput, but does not handle data dependences easily.
- Data dependences cause data hazards.
- Data hazards can be solved by:
  - software (no-ops)
  - hardware stalling
  - hardware forwarding
- Our processor, and indeed all modern processors, use a combination of forwarding and stalling.