Notes and Updates

- Midterm Review
- Office hours: Tues 3:30-5, Wed 11-12:30
  - Come see me for midterm questions
- 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?

Midterm Stats

- 51 exams: Avg 68   Median 77   Highest 97
Question 1 and 2: Performance

1. (5 points) A new compiler optimization decreases the instruction count (for program A) by 40% (that is ICnew = 0.6 ICold), but that only resulted in a speedup of 11. How can that be? (Please be specific—ie, give me the numbers AND WORDS to support your conclusion). Hint: start by showing the definition of speedup as defined in this problem.

Question 1 and 2: Performance

2. (10 points) Your typical workload on your machine spends 20% of its time executing operating system code, and the rest executing user (non-OS) code. Adding instruction B to the ISA could decrease average CPI by 20% (but doesn’t change instruction count). On the other hand, we could modify instruction C, which can only be used by the OS, to run 4 times as fast as it originally does. Which modification provides higher performance? Hint: Calculate each optimization in terms of the ETold, showing equations - then compare.
Bit Lines: Do you understand the datapath and how different decisions would affect it?

More in Discussion Section

- **Swap $3, $4, $3**
  - **Register issues**
    - Registers are overwritten each cycle, unless controlled by a “write” control line
    - If you decode values into A and B, they won’t necessarily be there in cycle 4 unless you “redecode”
  - **Best solution:**
    - EX: write A into rd? ALSO redecode B
    - EX2: write B into rs
      - New mux for rs
  - **Can you write a better swap (in terms of instruction format?)**
Chapter 6: Designing a Pipelined CPU

- What are our resources?

- What % of the time are they idle?
Chapter 6: Designing a Pipelined CPU

What % of the time are resources idle?
Chapter 6: Designing a Pipelined CPU

What is our roommate is gone? What happens to the pipeline?

A new(old?) design

- Single Cycle Load: _____ Rtype: ______
  - Cycle time:_____ Total Time:__________
- Multiple Cycle Load:______ Rtype: _____
  - Cycle time:_____ Total Time:__________

IF | ID | EX | MEM | WB
---|----|----|-----|----

Ld $3, 100($2)
Add $1, $5, $6
**A new(old?) design**

- Pipelined Load: _____ Rtype: _____
  - Cycle time: _____ Total Time: _________

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
</table>

**Review -- Instruction Latencies**

**Single-Cycle CPU**

```
Load  Ifetch  Reg/Dec  Exec  Mem  Wr
```

**Multiple Cycle CPU**

```
Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4 | Cycle 5 |
Load  Ifetch  Reg/Dec  Exec  Mem  Wr
Add    Ifetch  Reg/Dec  Exec  Wr```
Self Check!

- If my single cycle CPU has a cycle time of 14ns and my multicycle CPU has a cycle time of 3ns and my pipelined CPU has a cycle time of 3ns, what is the relative performance of my machines?
  - What kind of answer would you provide?
  - What kind of information do you need to know?

Pipelining Advantages

- Higher maximum throughput
- Higher utilization of CPU resources

- But, more hardware needed, perhaps complex control
BREAK!
Building Pedagogical Intelligence

- What was this about?

- “let students in on the tricks and truths of the learning trade”
- “agents of your own learning”, “something you can create and control”
- “when teachers continue to create opportunities for such self-assessment, students get better at identifying and seeking out what they need to advance their knowledge and abilities”
- Please think about how you learn
- In class, take the opportunity to figure out if you have learned and can apply what has been discussed
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>Iw</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>

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

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

The Pipeline in Execution
The Pipeline in Execution

lw $12, 1000($4)  add $10, $1, $2  Execute/Address Calculation
lw $12, 1000($4)  sub $15, $4, $1  Memory Access  Write Back

The Pipeline in Execution

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

sub $15, $4, $1  lw $12, 1000($4)  add $10, $1, $2  Execute/Address Calculation
lw $12, 1000($4)  sub $15, $4, $1  Memory Access  Write Back
sub $15, $4, $1  lw $12, 1000($4)
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

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/Address Calculation  
 Memory Access  
 Write Back

Diagram showing the pipeline stages for instruction execution and memory access.
The Pipeline in Execution

sub $11, $8, $7  lw $8, 50($3)  add $3, $10, $11  Memory Access  Write Back

The Pipeline in Execution

add $10, $1, $2  sub $11, $8, $7  lw $8, 50($3)  add $3, $10, $11  Write Back
Data Hazards

- When a result is needed in the pipeline before it is available, a *data hazard* occurs.

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

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
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
  - IM Reg
  - CC1
  - DM
  - CC3
  - Reg
  - CC5
  - IM Reg
  - CC7
  - IM Reg
  - CC8
- and $12, $2, $5
  - IM
  - Reg
  - DM
  - IM
  - bubble
  - Reg
  - CC5
  - DM
  - IM
  - Reg
  - CC7
  - DM
  - IM
  - bubble
  - Reg
  - CC8
- or $13, $6, $2
- add $14, $2, $2
  - IM
  - Reg
  - DM
  - IM
  - Reg
  - DM
The Pipeline

- What comparisons tell us when to stall?
  - What do we need to compare?

But that’s not it...

R2 Available

<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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

- 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