Review Problem 12

- How would the ALU be used to help with each of the following branches? The first is filled in for you:
  - **cmp**
  - **B.EQ:** SUBS X31, <val1>, <val2>; use zero flag
  - **B.NE:** SUBS X31, u1, u2 \_ use zero
  - **B.GE:** SUBS X31, u1, u2 \_ use zero + overflow
  - **B.GT:** not specified (Negative + overflow) + zero
  - **B.LE:** not specified (Negative + overflow) + zero
  - **B.LT:** SUBS not specified; Negative + overflow
Multiplication

Example

\[
\begin{array}{cccccc}
\text{Multiplicand:} & 0 & 1 & 1 & 0 & 6 \\
\text{Multiplier:} & 0 & 1 & 0 & 1 & 5 \\
\hline
\end{array}
\]

4 partial products

\[
\begin{array}{cccccccc}
+ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 1 & 1 & 1 & 0 & 30 \\
\end{array}
\]

Repeat n times:
Compute partial product; shift; add

NOTE: Each bit of partial products is just an AND operation
Parallel Multipliers
Computer “Performance”

Readings: 1.6-1.8

BIPS (Billion Instructions Per Second) vs. GHz (Giga Cycles Per Second)

Throughput (jobs/seconds) vs. Latency (time to complete a job)

Measuring “best” in a computer

3.0 GHz

The PowerBook G4 outguns Pentium III-based notebooks by up to 30 percent.*

* Based on Adobe Photoshop tests comparing a 500MHz PowerBook G4 to 850MHz Pentium III-based portable computers

Hyper Pipelined Technology
Performance Example: Homebuilders

<table>
<thead>
<tr>
<th>Builder</th>
<th>Time per House</th>
<th>Houses Per Month</th>
<th>House Options</th>
<th>Dollars Per House</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-build</td>
<td>24 months</td>
<td>1/24</td>
<td>Infinite</td>
<td>$200,000</td>
</tr>
<tr>
<td>Contractor</td>
<td>3 months</td>
<td>1</td>
<td>100</td>
<td>$400,000</td>
</tr>
<tr>
<td>Prefab</td>
<td>6 months</td>
<td>1,000</td>
<td>1</td>
<td>$250,000</td>
</tr>
</tbody>
</table>

Which is the “best” home builder?
- Homeowner on a budget?
- Rebuilding Haiti?
- Moving to wilds of Alaska?

Which is the “speediest” builder?
- Latency: how fast is one house built?
- Throughput: how long will it take to build a large number of houses?
Computer Performance

Primary goal: execution time (time from program start to program completion)

\[ \text{Performance} = \frac{1}{\text{Execution Time}} \]

To compare machines, we say "X is n times faster than Y"

\[ n = \frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution Time}_y}{\text{Execution Time}_x} \]

Example: Machine Orange and Grape run a program

Orange takes 5 seconds, Grape takes 10 seconds

\[ \frac{\text{Perf}_{\text{Orange}}}{\text{Perf}_{\text{Grape}}} = \frac{\text{Exec}_{\text{Orange}}}{\text{Exec}_{\text{Grape}}} = \frac{10}{5} = 2 \]

Orange is \( 2 \times \) times faster than Grape
Execution Time

Elapsed Time
counts everything \textit{(disk and memory accesses, I/O, etc.)}
a useful number, but often not good for comparison purposes

CPU time
doesn't count I/O or time spent running other programs
can be broken up into system time, and user time

Example: Unix "time" command

\begin{verbatim}
linux15.ee.washington.edu> time javac CircuitViewer.java
3.370u 0.570s 0:12.44 31.6% < user + OS
\end{verbatim}

Our focus: user CPU time
time spent executing the lines of code that are "in" our program
CPU Time

CPU execution time for a program = CPU clock cycles for a program * Clock period

CPU execution time for a program = CPU clock cycles for a program * \( \frac{1}{\text{Clock rate}} \)

Application example:
A program takes 10 seconds on computer Orange, with a 400MHz clock. Our design team is developing a machine Grape with a much higher clock rate, but it will require 1.2 times as many clock cycles. If we want to be able to run the program in 6 second, how fast must the clock rate be?

\[ 10 = \frac{\text{cycles}}{0.4 \times 10^9} \]

\[ \text{cycles} = 10 \times 0.4 \times 10^9 = 4 \times 10^9 \]

\[ 6 = 1.2 \times 4 \times 10^9 \times \frac{1}{\text{Rate}} \]

\[ \text{Rate} = \frac{1.2 \times 4 \times 10^9}{6} = \frac{0.2 \times 4 \times 10^9}{0.8 \times 10^9} = 800 \text{ MHz} \]
CPI

How do the # of instructions in a program relate to the execution time?

\[
\text{CPU clock cycles for a program} = \text{Instructions for a program} \times \text{Average Clock Cycles per Instruction (CPI)}
\]

\[
\text{CPU execution time for a program} = \text{Instructions for a program} \times \text{CPI} \times \frac{1}{\text{Clock rate}}
\]
CPI Example

Suppose we have two implementations of the same instruction set (ISA).

For some program

Machine A has a clock cycle time of 10 ns. and a CPI of 2.0
Machine B has a clock cycle time of 20 ns. and a CPI of 1.2

What machine is faster for this program, and by how much?

\[
\frac{\text{Perf}_A}{\text{Perf}_B} = \frac{\text{Exec}_B}{\text{Exec}_A} = \frac{\text{same inst}_B \times CPI_B \times \text{Period}_B}{\text{inst}_A \times CPI_A \times \text{Period}_A} = \frac{1.2 \times 20}{2.0 \times 10} = 1.2 \times
\]
Review Problem 15

- Orange runs at 1GHz, and provides a unit making all floating point operations take 1 cycle. Grape runs at 1.2 GHz by deleting the unit, meaning floating point operations take 20 cycles. Which machine is better?

Depends on what's your usage.
Computing CPI

Different types of instructions can take very different amounts of cycles
Memory accesses, integer math, floating point, control flow

\[ CPI = \sum_{types} \left( Cycles_{type} \times Frequency_{type} \right) \]

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Type Cycles</th>
<th>Type Frequency</th>
<th>Cycles * Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>1</td>
<td>50%</td>
<td>= 0.5</td>
</tr>
<tr>
<td>Load</td>
<td>5</td>
<td>20%</td>
<td>= 1.0</td>
</tr>
<tr>
<td>Store</td>
<td>3</td>
<td>10%</td>
<td>= 0.3</td>
</tr>
<tr>
<td>Branch</td>
<td>2</td>
<td>20%</td>
<td>= 0.4</td>
</tr>
</tbody>
</table>

CPI: 2.2
CPI & Processor Tradeoffs

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Type Cycles</th>
<th>Type Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>1</td>
<td>50%</td>
</tr>
<tr>
<td>Load</td>
<td>5</td>
<td>20%</td>
</tr>
<tr>
<td>Store</td>
<td>3</td>
<td>10%</td>
</tr>
<tr>
<td>Branch</td>
<td>2</td>
<td>20%</td>
</tr>
</tbody>
</table>

How much faster would the machine be if:

1. A data cache reduced the average load time to 2 cycles?

\[
\frac{2.2}{(2.2 - 1 \times 2)} = \frac{2.2}{(2.2 - 0.6)} = \frac{2.2}{1.6} = 1.375 \times \text{Best}
\]

2. Branch prediction shaved a cycle off the branch time?

\[
\frac{2.2}{(2.2 - 1 \times 2)} = \frac{2.2}{2.0} = 1.1 \times \text{Worst}
\]

3. Two ALU instructions could be executed at once?

\[
\frac{2.2}{(2.2 - 0.5 \times 0.5)} = \frac{2.2}{(2.2 - 0.25)} = \frac{2.2}{1.95} = 1.13 \times
\]
Warning 1: Amdahl’s Law

The impact of a performance improvement is limited by what is NOT improved:

\[
\frac{\text{Execution time after improvement}}{4} = \frac{\text{Execution time of unaffected}}{20} + \frac{\text{Execution time affected}}{\frac{80}{\text{improve}}} * \frac{1}{\text{Amount of improvement}}
\]

Example: Assume a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to speed up multiply to make the program run 4 times faster?

\[
\frac{100}{4} = 20 + \frac{80}{\text{improve}}
\]

5 times faster?

\[
\frac{100}{5} = 20 + \frac{80}{\text{improve}}
\]

\[
20 = 20 + \frac{80}{\text{improve}} \quad 0 = \frac{80}{\text{improve}}
\]

\[
\text{improve} = \infty
\]
Warning 2: BIPs, GHz ≠ Performance

Higher MHz (clock rate) doesn’t always mean better CPU
Orange computer: 1000 MHz, CPI: 2.5, 1 billion instruction program
\[
\text{Time} = 1 \times 10^9 \times 2.5 \times \frac{1}{1 \times 10^9} = 2.5 \text{ seconds}
\]

Grape computer: 500MHz, CPI: 1.1, 1 billion instruction program
\[
\text{Time} = 1 \times 10^9 \times 1.1 \times \frac{1}{0.5 \times 10^9} = 2 \times 1.1 = 2.2 \text{ seconds}
\]

Higher MIPs (million instructions per second) doesn’t always mean better CPU
1 GHz machine, with two different compilers
Compiler A on program X: 10 Billion ALU, 1 Billion Load
Compiler B on program X: 5 Billion ALU, 1 Billion Load

\[
A = \frac{10B \times 1 + 1B \times 5}{1Ghz} \quad \text{Execution Time: A} \quad 15 \text{ sec} \\
B = \frac{5B \times 1 + 1B \times 5}{1Ghz} \\
\]

\[
A = \frac{11B}{15 \text{ sec}} = 733 \text{ MIPs} \\
B = \frac{6B}{105 \text{ sec}} = 600 \text{ MIPs}
\]

\[
\text{MIPS: A} \quad 733 \quad \text{B} \quad 600
\]

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Type Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>5</td>
</tr>
<tr>
<td>Store</td>
<td>3</td>
</tr>
<tr>
<td>Branch</td>
<td>2</td>
</tr>
</tbody>
</table>
Processor Performance Summary

Machine performance:

\[
\text{CPU execution time for a program} = \frac{\text{Instructions for a program} \times \text{CPI} \times 1}{\text{Clock rate}}
\]

Better performance:

- \(\downarrow\) number of instructions to implement computations
- \(\downarrow\) CPI
- \(\uparrow\) Clock rate

Improving performance must balance each constraint

Example: RISC vs. CISC

CISC: Complex Instruction Set Comp.
RISC: Reduced Instruction Set Comp.
Datapath & Control

Readings: 4.1-4.4

Datapath: System for performing operations on data, plus memory access.

Control: Control the datapath in response to instructions.
Simple CPU

Develop complete CPU for subset of instruction set

**Memory:** LDUR, STUR

<table>
<thead>
<tr>
<th>Opcode</th>
<th>DAddr9</th>
<th>00</th>
<th>Rn</th>
<th>Rd</th>
</tr>
</thead>
</table>

**Branch:** B

<table>
<thead>
<tr>
<th>Opcode</th>
<th>BrAddr26</th>
</tr>
</thead>
</table>

**Conditional Branch:** CBZ

<table>
<thead>
<tr>
<th>Opcode</th>
<th>CondAddr19</th>
<th>Rd</th>
</tr>
</thead>
</table>

**Arithmetic:** ADD, SUB

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Rm</th>
<th>SHAMT</th>
<th>Rn</th>
<th>Rd</th>
</tr>
</thead>
</table>

Most other instructions similar
Execution Cycle

- **Instruction Fetch**: Obtain instruction from program storage
- **Instruction Decode**: Determine required actions and instruction size
- **Operand Fetch**: Locate and obtain operand data
- **Execute**: Compute result value or status
- **Result Store**: Deposit results in storage for later use
- **Next Instruction**: Determine successor instruction
Processor Overview

Overall Dataflow
- PC fetches instructions
- Instructions select operand registers, ALU immediate values
- ALU computes values
- Load/Store addresses computed in ALU
- Result goes to register file or Data memory