Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
  - overflow when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive
- Consider the operations $A + B$, and $A - B$
  - Can overflow occur if $B$ is 0?
  - Can overflow occur if $A$ is 0?

Effects of Overflow

- An exception (interrupt) occurs
  - Control jumps to predefined address for exception
  - Interrupted address is saved for possible resumption
- Details based on software system / language
  - example: flight control vs. homework assignment
- Don’t always want to detect overflow
  — new MIPS instructions: addu, addiu, subu
  — note: addiu still sign-extends!
  — note: sltu, sltiu for unsigned comparisons

Multiplication

- More complicated than addition
  - accomplished via shifting and addition
- More time and more area
- Let’s look at 3 versions based on a gradeschool algorithm
  \[
  \begin{array}{c}
    \text{0010 (multiplicand)} \\
    \times \text{1011 (multiplier)} \\
  \end{array}
  \]
- Negative numbers: convert and multiply
  - there are better techniques, we won’t look at them

Multiplication: Implementation
Floating Point (a brief look)

- We need a way to represent
  - numbers with fractions, e.g., 3.1416
  - very small numbers, e.g., .00000001
  - very large numbers, e.g., 3.15576 \times 10^9
- Representation:
  - sign, exponent, significand: \((-1)^{\text{sign}} \times \text{significand} \times 2^{\text{exponent}}\)
  - more bits for significand gives more accuracy
  - more bits for exponent increases range
- IEEE 754 floating point standard:
  - single precision: 8 bit exponent, 23 bit significand
  - double precision: 11 bit exponent, 52 bit significand

Floating point addition

- 
  - Leading “1” bit of significand is implicit
  - Exponent is “biased” to make sorting easier
    - all 0s is smallest exponent all 1s is largest
    - bias of 127 for single precision and 1023 for double precision
    - summary: \((-1)^{\text{sign}} \times (1+\text{significand}) \times 2^{\text{exponent}-\text{bias}}\)
  - Example:
    - decimal: -.75 = -\left( \frac{1}{2} + \frac{1}{4} \right)
    - binary: -.11 = -1.1 \times 2^1
    - floating point: exponent = 126 = 01111110
    - IEEE single precision: 10111110100000000000000000000000

IEEE 754 floating-point standard

- Leading “1” bit of significand is implicit
- Exponent is “biased” to make sorting easier
  - all 0s is smallest exponent all 1s is largest
  - bias of 127 for single precision and 1023 for double precision
  - summary: \((-1)^{\text{sign}} \times (1+\text{significand}) \times 2^{\text{exponent}-\text{bias}}\)
- Example:
  - decimal: -.75 = -\left( \frac{1}{2} + \frac{1}{4} \right)
  - binary: -.11 = -1.1 \times 2^1
  - floating point: exponent = 126 = 01111110
  - IEEE single precision: 10111110100000000000000000000000
Floating Point Complexities

- Operations are somewhat more complicated (see text)
- In addition to overflow we can have “underflow”
- Accuracy can be a big problem
  - IEEE 754 keeps two extra bits, guard and round
  - four rounding modes
  - positive divided by zero yields “infinity”
  - zero divide by zero yields “not a number”
  - other complexities
- Implementing the standard can be tricky
- Not using the standard can be even worse
  - see text for description of 80x86 and Pentium bug!

Chapter Three Summary

- Computer arithmetic is constrained by limited precision
- Bit patterns have no inherent meaning but standards do exist
  - two’s complement
  - IEEE 754 floating point
- Computer instructions determine “meaning” of the bit patterns
- Performance and accuracy are important so there are many complexities in real machines
- Algorithm choice is important and may lead to hardware optimizations for both space and time (e.g., multiplication)
- You may want to look back (Section 3.10 is great reading!)

Chapter 4

- Measure, Report, and Summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation

Why is some hardware better than others for different programs?

What factors of system performance are hardware related?
  (e.g., Do we need a new machine, or a new operating system?)

How does the machine’s instruction set affect performance?
Which of these airplanes has the best performance?

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Passengers</th>
<th>Range (mi)</th>
<th>Speed (mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 737-100</td>
<td>101</td>
<td>630</td>
<td>598</td>
</tr>
<tr>
<td>Boeing 747</td>
<td>470</td>
<td>4150</td>
<td>610</td>
</tr>
<tr>
<td>BAC/Toa Concorde</td>
<td>132</td>
<td>4000</td>
<td>1350</td>
</tr>
<tr>
<td>Douglas DC-8</td>
<td>146</td>
<td>8720</td>
<td>544</td>
</tr>
</tbody>
</table>

• How much faster is the Concorde compared to the 747?
• How much bigger is the 747 than the Douglas DC-8?

Computer Performance: TIME, TIME, TIME

• Response Time (latency)
  — How long does it take for my job to run?
  — How long does it take to execute a job?
  — How long must I wait for the database query?

• Throughput
  — How many jobs can the machine run at once?
  — What is the average execution rate?
  — How much work is getting done?

• If we upgrade a machine with a new processor what do we increase?
• If we add a new machine to the lab what do we increase?

Execution Time

• Elapsed Time
  — counts everything (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
• Our focus: user CPU time
  — time spent executing the lines of code that are “in” our program

Book’s Definition of Performance

• For some program running on machine X,
  \[
  \text{Performance}_X = 1 / \text{Execution time}_X
  \]
• ”X is n times faster than Y”
  \[
  \text{Performance}_X / \text{Performance}_Y = n
  \]
• Problem:
  — machine A runs a program in 20 seconds
  — machine B runs the same program in 25 seconds
Clock Cycles

- Instead of reporting execution time in seconds, we often use cycles

\[
\text{seconds program} = \frac{\text{cycles program}}{\text{seconds cycle}}
\]

- Clock “ticks” indicate when to start activities (one abstraction):

\[
\text{cycle time} = \text{time between ticks} = \text{seconds per cycle}
\]

- Clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)

A 4 Ghz. clock has a cycle time of

\[
\frac{1}{4\times10^9} = 10^{12} = 290 \text{ picoseconds (ps)}
\]