SI232
Slide Set #9:
Computer Arithmetic (Chapter 3)

RECAP – HOMEWORK RESPONSIBILITIES

• I will
  – Ensure problems from text are clear, or write new ones
  – Answer your questions promptly (start sending them!)
  – Continue to be available for EI
  – Provide a stapler

• You should
  – Email/EI questions if you are confused or need help
  – Read the directions carefully
  – Expect to spend some time: for learning, not just rehash of class
  – Start early
  – (optional) Collaborate (though not for projects)

• Suggestions
  – Review your notes from class – sometime the same day
  – Practice and understand problems/exercises

ADMIN

• Reading
  – Read 3.1, 3.2, 3.3, 3.4
  – Skim 3.5
  – Read 3.6 (Floating point – skim details on addition, multiplication, rounding – but pay attention to representation and MIPS instructions)
  – Read 3.8

Chapter Goals

• Introduce 2’s complement numbers
  – Addition and subtraction
  – Sketch multiplication, division

• Overview of ALU (arithmetic logic unit)

• Floating point numbers
  – Representation
  – Arithmetic operations
  – MIPS instructions
Bits

- What do these two binary strings represent?
  0000 0000 0000 0000 0000 0000 0001 0101
  0000 0001 0010 0011 0100 0101 0110 0111

- Bits are...

- ___________ define relationship between ___________ and ___________

Integers: Possible 3-bit Representations of 2 and -2

1. Unsigned

2. Sign and Magnitude

3. One’s Complement

4. Two’s Complement

Bits as Numbers: Complications

- Numbers are finite

- Fractions and real numbers

- Negative numbers

- MIPS typically uses 32 bits for a number
  - But we’ll often demonstrate with fewer for simplicity

- MSB vs LSB

Example Representations

<table>
<thead>
<tr>
<th></th>
<th>Unsigned</th>
<th>Sign Mag.</th>
<th>One’s Comp.</th>
<th>Two’s Comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>+0</td>
<td>000 = +0</td>
<td>000 = +0</td>
<td>000 = +0</td>
</tr>
<tr>
<td>001</td>
<td>+1</td>
<td>001 = +1</td>
<td>001 = +1</td>
<td>001 = +1</td>
</tr>
<tr>
<td>010</td>
<td>+2</td>
<td>010 = +2</td>
<td>010 = +2</td>
<td>010 = +2</td>
</tr>
<tr>
<td>011</td>
<td>+3</td>
<td>011 = +3</td>
<td>011 = +3</td>
<td>011 = +3</td>
</tr>
<tr>
<td>100</td>
<td>+4</td>
<td>100 = -0</td>
<td>100 = -0</td>
<td>100 = -0</td>
</tr>
<tr>
<td>101</td>
<td>+5</td>
<td>101 = -1</td>
<td>101 = -1</td>
<td>101 = -1</td>
</tr>
<tr>
<td>110</td>
<td>+6</td>
<td>110 = -2</td>
<td>110 = -2</td>
<td>110 = -2</td>
</tr>
<tr>
<td>111</td>
<td>+7</td>
<td>111 = -3</td>
<td>111 = -3</td>
<td>111 = -3</td>
</tr>
</tbody>
</table>
Two's Complement Operations

- Negating a two's complement number: invert all bits and add 1
- But must write down leading zero bits if there!
- Example:
  - Express -6₁₀ in 8-bit binary 2's complement:

Exercise #1

- Assume we have 4 bits. Convert the given decimal numbers to the stated binary representations.

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed Magnitude</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One's Comp.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two's Comp.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2

- Convert the given decimal numbers to the stated binary representations.

<table>
<thead>
<tr>
<th></th>
<th>-3 (using 4 bits)</th>
<th>-3 (using 6 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sign Magnitude</td>
<td>One's Comp.</td>
</tr>
<tr>
<td></td>
<td>Two's Comp.</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #3

- Assume the following is in binary two’s complement form. What do they represent in decimal?

001011

111011

- Now negate these numbers and show the new binary form:

-(001011) =

-(111011) =
Exercise #4 – Stretch

- Given N bits, what is the largest and smallest number that each of the following can represent?

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign Magnitude</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ones Complement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Twos Complement</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MIPS

- MIPS signed numbers use...

- 32 bit signed numbers:

  - 0000 0000 0000 0000 0000 0000 0000 0000two = 0ten
  - 0000 0000 0000 0000 0000 0000 0000 0001two = + 1ten
  - 0000 0000 0000 0000 0000 0000 0000 0010two = + 2ten
  - ...1111 1111 1111 1111 1111 1111 1111 1110two = + 2,147,483,646ten
  - 0111 1111 1111 1111 1111 1111 1111 1111two = + 2,147,483,647ten
  - 1000 0000 0000 0000 0000 0000 0000 0000two = – 2,147,483,648ten
  - 1000 0000 0000 0000 0000 0000 0000 0001two = – 2,147,483,647ten
  - 1000 0000 0000 0000 0000 0000 0000 0010two = – 2,147,483,646ten
  - ...1111 1111 1111 1111 1111 1111 1111 1110two = – 3ten
  - 1111 1111 1111 1111 1111 1111 1111 1111two = – 2ten
  - 1111 1111 1111 1111 1111 1111 1111 1111two = – 1ten

Two's Complement Operations

- Converting n bit numbers into numbers with more than n bits:
  - MIPS 16 bit immediate gets converted to 32 bits for arithmetic
  - copy the most significant bit (the sign bit) into the other bits
  - 4 -> 8 bit example:
    - 0010  ->
      - 1010  ->
  - This is called

Signed vs. unsigned numbers

- Some values don’t make sense as negative numbers

- MIPS allows values to be signed or unsigned
- Different instructions to deal with each case
  - add vs. addu
  - lb vs. lbu
  - addi vs. addiu
  - slli vs srlu
  - Usually, the unsigned version will not ___________________

- Exception:
Addition & Subtraction

- Just like in grade school (carry/borrow 1s)
  
  \[
  \begin{array}{lll}
  0001 & 0111 & 0110 \\
  + 0101 & -0110 & -0101 \\
  \end{array}
  \]

- Easier way to subtract?

Detecting Overflow

- Overflow -- result too large for finite computer word
- Is overflow possible if adding...
  - a positive and a negative number?
  - two positive numbers?
  - two negative numbers?

- Subtraction:
  - Invert the second number to test
  - So no overflow possible when signs are...

Addition & Subtraction

- Another example:
  
  \[
  \begin{array}{ll}
  0111 \\
  + 0001 \\
  \end{array}
  \]

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
  - C always ignores overflow
- Don't always want to detect overflow
  - "Unsigned" arithmetic instructions will ignore: `addu, addiu, subu`
Summary: Advantages of Two’s Complement

- How to negate a number?
- How many zeros?
- How add positive and negative numbers?
- Consequently, essentially all modern computers use this