Lecture 8: Addition, Multiplication & Division

• Today’s topics:
  - Signed/Unsigned
  - Addition
  - Multiplication
  - Division
MIPS Instructions

Consider a comparison instruction:

\[
\text{slt} \quad \$t0, \$t1, \$zero
\]

and \$t1 contains the 32-bit number \text{1111 01…01}

What gets stored in \$t0?
The result depends on whether \$t1 is a signed or unsigned number – the compiler/programmer must track this and accordingly use either \text{slt} or \text{sltu}

\[
\text{slt} \quad \$t0, \$t1, \$zero \quad \text{stores 1 in } \$t0
\]

\[
\text{sltu} \quad \$t0, \$t1, \$zero \quad \text{stores 0 in } \$t0
\]
Sign Extension

• Occasionally, 16-bit signed numbers must be converted into 32-bit signed numbers – for example, when doing an add with an immediate operand.

• The conversion is simple: take the most significant bit and use it to fill up the additional bits on the left – known as sign extension.

So $2_{10}$ goes from $0000\ 0000\ 0000\ 0010$ to $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010$.

and $-2_{10}$ goes from $1111\ 1111\ 1111\ 1110$ to $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110$. 
Alternative Representations

• The following two (intuitive) representations were discarded because they required additional conversion steps before arithmetic could be performed on the numbers

  ▪ sign-and-magnitude: the most significant bit represents +/- and the remaining bits express the magnitude

  ▪ one’s complement: -x is represented by inverting all the bits of x

Both representations above suffer from two zeroes
Addition and Subtraction

• Addition is similar to decimal arithmetic

• For subtraction, simply add the negative number – hence, subtract A-B involves negating B’s bits, adding 1 and A

Source: H&P textbook
Overflows

• For an unsigned number, overflow happens when the last carry (1) cannot be accommodated

• For a signed number, overflow happens when the most significant bit is not the same as every bit to its left
  ▪ when the sum of two positive numbers is a negative result
  ▪ when the sum of two negative numbers is a positive result
  ▪ The sum of a positive and negative number will never overflow

• MIPS allows addu and subu instructions that work with unsigned integers and never flag an overflow – to detect the overflow, other instructions will have to be executed
Multiplication Example

Multiplicand  
\[1000_{\text{ten}}\]

Multiplier  
\[x \ 1001_{\text{ten}}\]

\[\begin{array}{c}
1000 \\
0000 \\
0000 \\
1000 \\
\end{array}\]

Product  
\[1001000_{\text{ten}}\]

In every step:
- multiplicand is shifted
- next bit of multiplier is examined (also a shifting step)
- if this bit is 1, shifted multiplicand is added to the product
HW Algorithm 1

In every step
- multiplicand is shifted
- next bit of multiplier is examined (also a shifting step)
- if this bit is 1, shifted multiplicand is added to the product

Source: H&P textbook
HW Algorithm 2

- 32-bit ALU and multiplicand is untouched
- the sum keeps shifting right
- at every step, number of bits in product + multiplier = 64, hence, they share a single 64-bit register

Source: H&P textbook
Notes

• The previous algorithm also works for signed numbers (negative numbers in 2’s complement form)

• We can also convert negative numbers to positive, multiply the magnitudes, and convert to negative if signs disagree

• The product of two 32-bit numbers can be a 64-bit number -- hence, in MIPS, the product is saved in two 32-bit registers
MIPS Instructions

mult  $s2, $s3  computes the product and stores it in two “internal” registers that can be referred to as hi and lo

mfhi  $s0  moves the value in hi into $s0
mflo  $s1  moves the value in lo into $s1

Similarly for multu
Fast Algorithm

- The previous algorithm requires a clock to ensure that the earlier addition has completed before shifting.

- This algorithm can quickly set up most inputs – it then has to wait for the result of each add to propagate down – faster because no clock is involved.

-- Note: high transistor cost

Source: H&P textbook
Division

\[
\begin{array}{c|c}
\text{Divisor} & 1000_{\text{ten}} \\
\hline
\text{Dividend} & 1001010_{\text{ten}} \\
\hline
\end{array}
\]

At every step,
- shift divisor right and compare it with current dividend
- if divisor is larger, shift 0 as the next bit of the quotient
- if divisor is smaller, subtract to get new dividend and shift 1 as the next bit of the quotient
At every step,

- shift divisor right and compare it with current dividend
- if divisor is larger, shift 0 as the next bit of the quotient
- if divisor is smaller, subtract to get new dividend and shift 1 as the next bit of the quotient
## Divide Example

- Divide $7_{ten} \ (0000 \ 0111_{two})$ by $2_{ten} \ (0010_{two})$

<table>
<thead>
<tr>
<th>Iter</th>
<th>Step</th>
<th>Quot</th>
<th>Divisor</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial values</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Divide Example

- Divide \( 7_{\text{ten}} \) (\(0000\ 0111_{\text{two}}\)) by \( 2_{\text{ten}} \) (\(0010_{\text{two}}\))

<table>
<thead>
<tr>
<th>Iter</th>
<th>Step</th>
<th>Quot</th>
<th>Divisor</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial values</td>
<td>0000</td>
<td>0010 0000</td>
<td>0000 0111</td>
</tr>
<tr>
<td>1</td>
<td>Rem = Rem – Div</td>
<td>0000</td>
<td>0010 0000</td>
<td>1110 0111</td>
</tr>
<tr>
<td></td>
<td>Rem &lt; 0 (\Rightarrow) +Div, shift 0 into Q</td>
<td>0000</td>
<td>0010 0000</td>
<td>0000 0111</td>
</tr>
<tr>
<td></td>
<td>Shift Div right</td>
<td>0000</td>
<td>0001 0000</td>
<td>0000 0111</td>
</tr>
<tr>
<td>2</td>
<td>Same steps as 1</td>
<td>0000</td>
<td>0001 0000</td>
<td>1111 0111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000</td>
<td>0001 0000</td>
<td>0000 0111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000</td>
<td>0000 1000</td>
<td>0000 0111</td>
</tr>
<tr>
<td>3</td>
<td>Same steps as 1</td>
<td>0000</td>
<td>0000 0100</td>
<td>0000 0111</td>
</tr>
<tr>
<td>4</td>
<td>Rem = Rem – Div</td>
<td>0000</td>
<td>0000 0100</td>
<td>0000 0011</td>
</tr>
<tr>
<td></td>
<td>Rem &gt;= 0 (\Rightarrow) shift 1 into Q</td>
<td>0001</td>
<td>0000 0100</td>
<td>0000 0011</td>
</tr>
<tr>
<td></td>
<td>Shift Div right</td>
<td>0001</td>
<td>0000 0010</td>
<td>0000 0011</td>
</tr>
<tr>
<td>5</td>
<td>Same steps as 4</td>
<td>0011</td>
<td>0000 0001</td>
<td>0000 0001</td>
</tr>
</tbody>
</table>