Overview

- Signed Numbers: 2'Complement representation
- Addition, Subtraction and Comparison in 2's Complement
- Comparison in signed and unsigned numbers
- Condition code flags

Limits of Computer Numbers

- Bits can represent anything!
- Characters?
  - 26 letter => 5 bits
  - upper/lower case + punctuation => 7 bits (in 8) (ASCII)
  - rest of the world’s languages => 16 bits (unicode)
- Logical values?
  - 0 => False, 1 => True
- Colors?
- Locations / addresses? commands?
- but N bits => only $2^N$ things

Review: Two’s Complement

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0000</td>
<td>1</td>
</tr>
<tr>
<td>0000</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0111</td>
<td>2,147,483,645</td>
</tr>
<tr>
<td>0111</td>
<td>2,147,483,646</td>
</tr>
<tr>
<td>0111</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1000</td>
<td>-2,147,483,648</td>
</tr>
<tr>
<td>1000</td>
<td>-2,147,483,647</td>
</tr>
<tr>
<td>1000</td>
<td>-2,147,483,646</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1111</td>
<td>-3</td>
</tr>
<tr>
<td>1111</td>
<td>-2</td>
</tr>
<tr>
<td>1111</td>
<td>-1</td>
</tr>
</tbody>
</table>

- One zero, 31st bit => >=0 or <0, called **sign bit**
  - but one negative with no positive $-2,147,483,648$
Review: Two’s Complement Formula

- Recognizing role of sign bit, can represent positive and negative numbers in terms of the bit value times a power of 2:
  - \( d_{31} \times (-2^{31}) + d_{30} \times 2^{30} + \ldots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0 \)

Example

1111 1111 1111 1111 1111 1111 1111 1100\(_{\text{two}}\)

\[= 1 \times (-2^{31}) + 1 \times 2^{30} + 1 \times 2^{29} + \ldots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \]

\[= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^2 + 0 + 0 \]

\[= -2,147,483,648_{\text{ten}} + 2,147,483,644_{\text{ten}} \]

\[= -4_{\text{ten}} \]

Review: Two’s complement shortcut: Negation

- Invert every 0 to 1 and every 1 to 0, then add 1 to the result
  - Sum of number and its inverted representation must be 111...111\(_{\text{two}}\)
  - Let \( x' \) mean the inverted representation of \( x \)
  - Then \( x + x' = -1 \Rightarrow x + x' + 1 = 0 \Rightarrow x' + 1 = -x \)

Example: -4 to +4 to -4

\( x \) : 1111 1111 1111 1111 1111 1111 1111 1100\(_{\text{two}}\)

\( x' \) : 0000 0000 0000 0000 0000 0000 0000 0011\(_{\text{two}}\)

+1 : 0000 0000 0000 0000 0000 0000 0000 0100\(_{\text{two}}\)

(') : 1111 1111 1111 1111 1111 1111 1111 1011\(_{\text{two}}\)

+1 : 1111 1111 1111 1111 1111 1111 1111 1100\(_{\text{two}}\)

Two’s Complement’s Arithmetic Example

Example: 20 – 4 = 16

\( 20 - 4 = 20 + (-4) \)

\( X \): 0100\(_{\text{two}}\)

\( Y \): 0100\(_{\text{two}}\)

\( X + (-Y) \): 0100\(_{\text{two}}\)

Ignore Carry out

Two’s Complement’s Arithmetic Example

Example: –2,147,483,648 –2 = –2,147,483,650 ?

\( X \): 1000 0000 0000 0000 0000 0000 0000 0000\(_{\text{two}}\)

\( Y \): 0000 0000 0000 0000 0000 0000 0000 0010\(_{\text{two}}\)

\( X + (-Y) \): 0111 1111 1111 1111 1111 1111 1111 1110\(_{\text{two}}\)

\[= -2,147,483,646_{\text{ten}} \]

\[\neq -2,147,483,650_{\text{ten}} \]

OVERFLOW

\[0111 1111 1111 1111 1111 1111 1111 1110_{\text{two}}\]

\[= 2,147,483,646_{\text{ten}} \]

\[\neq -2,147,483,650_{\text{ten}} \]
Signed vs. Unsigned Numbers

- **C declaration int**
  - Declares a signed number
  - Uses two's complement

- **C declaration unsigned int**
  - Declares an unsigned number
  - Treats 32-bit number as unsigned integer, so most significant bit is part of the number, not a sign bit

**NOTE:**

- Hardware does all arithmetic in 2's complement.
- It is up to programmer to interpret numbers as signed or unsigned.
- Hardware provide some information to interpret numbers as signed or unsigned (check Slides 17-20!)

Overflow for Two’s Complement Numbers?

- Adding (or subtracting) 2 32-bit numbers can yield a result that needs 33 bits
  - Sign bit set with value of result instead of proper sign of result
  - Since need just 1 extra bit, only sign bit can be wrong

<table>
<thead>
<tr>
<th>Op</th>
<th>A</th>
<th>B</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A + B</td>
<td>&gt;=0</td>
<td>&gt;=0</td>
<td>&lt;0</td>
</tr>
<tr>
<td>A + B</td>
<td>&lt;0</td>
<td>&lt;0</td>
<td>&gt;=0</td>
</tr>
<tr>
<td>A - B</td>
<td>&gt;=0</td>
<td>&lt;0</td>
<td>&lt;0</td>
</tr>
<tr>
<td>A - B</td>
<td>&lt;0</td>
<td>&gt;=0</td>
<td>&gt;=0</td>
</tr>
</tbody>
</table>

When adding operands with different signs (subtracting with same signs) Overflow cannot occur.

Signed v. Unsigned Comparisons

- \( X = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{\text{two}} \)
- \( Y = 0011\ 1011\ 1001\ 1010\ 1000\ 1010\ 0000\ 0000_{\text{two}} \)

Is \( X > Y ? \) Do the Subtraction \( X - Y \) and check result

\[ \begin{align*}
X &= 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{\text{two}} - \\
Y &= 0011\ 1011\ 1001\ 1010\ 1000\ 1010\ 0000\ 0000_{\text{two}} \\
R &= 1100\ 0100\ 0110\ 0101\ 0111\ 0110\ 0000\ 0000_{\text{two}} \\
\end{align*} \]

Carry out

- Ve sign indicates \( X \) is NOT greater than \( Y \)
- Ve sign and carry out indicates \( X \) is greater than \( Y \) if numbers were unsigned.
Numbers are stored at addresses

° Memory is a place to store bits
° A word is a fixed number of bits (e.g., 32) at an address
  • also fixed no. of bits
° Addresses are naturally represented as unsigned numbers

11111 = \(2^k - 1\)

Status Flags in Program Status Register CPSR

Copies of the ALU status flags (latched for some instructions).

* Condition Code Flags
N = Negative result from ALU flag.
Z = Zero result from ALU flag.
C = ALU operation Carried out
V = ALU operation overflowed (carry into the msb ≠ carry out of msb)

ARM Terminology:
GT (Greater) \(X > Y\) (signed Arithmetic)
HI (Higher) \(X > Y\) (unsigned Arithmetic)

Condition Flags

<table>
<thead>
<tr>
<th>Flags</th>
<th>Arithmetic Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative (N='1')</td>
<td>Bit 31 of the result has been set</td>
</tr>
<tr>
<td></td>
<td>Indicates a negative number in signed operations</td>
</tr>
<tr>
<td>Zero (Z='1')</td>
<td>Result of operation was zero</td>
</tr>
<tr>
<td>Carry (C='1')</td>
<td>Result was greater than 32 bits</td>
</tr>
<tr>
<td>oVerflow (V='1')</td>
<td>Result was greater than 31 bits</td>
</tr>
<tr>
<td></td>
<td>Indicates a possible corruption of the sign bit in signed numbers</td>
</tr>
</tbody>
</table>

And in Conclusion...

° Computers do arithmetic in 2’s complement
° Interpretation of numbers can be signed or unsigned
° The Arithmetic operation results should be interpreted depending on the signed or unsigned interpretation of the operands.