Signed Numbers

- We have already talked about unsigned binary numbers
  » each bit position represents a power of 2
  » range of values is 0 to $2^n-1$
- How can we indicate negative values?
  » two states: positive or negative
  » a binary bit indicates one of two states: 0 or 1
  ⇒ use one bit for the sign bit

Where is the sign bit?

- Could use an additional bit to indicate sign
  » each value would require 33 bits
  » would really foul up the hardware design
- Could use any bit in the 32-bit word
  » any bit but the left-most (high order) would complicate the hardware tremendously
- The high order bit (left-most) is the sign bit
  » remaining bits indicate the value
Format of 32-bit signed integer

- Bit 31 is the sign bit
  - 0 for positive numbers, 1 for negative numbers
  - aka most significant bit (msb), high order bit

Example: 4-bit signed numbers

<table>
<thead>
<tr>
<th>Hex</th>
<th>Bin</th>
<th>Unsigned</th>
<th>Signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>1111</td>
<td>15</td>
<td>-1</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
<td>14</td>
<td>-2</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
<td>13</td>
<td>-3</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
<td>12</td>
<td>-4</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
<td>11</td>
<td>-5</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
<td>10</td>
<td>-6</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
<td>-7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
<td>-8</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0111</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Two’s complement notation

- Note special arrangement of negative values
- One zero value, one extra negative value
- The representation is exactly what you get by doing a subtraction

Why “two’s” complement?

- In an n-bit word, negative x is represented by the value of $2^n-x$
- 4-bit example
  - $2^4 = 16$. What is the representation of -6?
Negating a number

Given x, how do we represent negative x?

\[ \text{negative}(x) = 2^n - x \]

and

\[ x + \text{complement}(x) = 2^n - 1 \]

so

\[ \text{negative}(x) = 2^n - x = \text{complement}(x) + 1 \]

The easy shortcut

» write down the value in binary
» complement all the bits
» add 1

Example: the negation shortcut

\[ \text{decimal } 6 = 0110 = +6 \]
\[ \text{complement} = 1001 \]
\[ \text{add } 1 = 1010 = -6 \]

\[ \text{decimal } -6 = 1010 = -6 \]
\[ \text{complement} = 0101 \]
\[ \text{add } 1 = 0110 = +6 \]

Signed and Unsigned Compares

<table>
<thead>
<tr>
<th>Hex</th>
<th>Bin</th>
<th>Unsigned Decimal</th>
<th>Signed Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>1111</td>
<td>15 -1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
<td>14 -2</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
<td>13 -3</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
<td>12 -4</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
<td>11 -5</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
<td>10 -6</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9 -7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8 -8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6 6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5 5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3 3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2 2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

Note: using 4-bit signed numbers in this example. The same relationships exist with 32-bit signed values.

Loading bytes

- Unsigned:   `lbu $reg, a($reg)`
  » the byte is 0-extended into the register

```
0000 0000 0000 0000 0000 0000 xxxxxxxxxxxx
```

- Signed:     `lb $reg, a($reg)`
  » bit 7 is extended through bit 31

```
0000 0000 0000 0000 0000 0000 0xxxx xxxxxxxxx
```

```
1111 1111 1111 1111 1111 1111 lxxxx xxxxxxxxx
```
Why Floating Point?

• The numbers we have talked about so far have all been integers in the range 0 to 4B or -2B to +2B
• What about numbers outside that range?
  » population of the planet: 6 billion+
• What about numbers that have a fractional part in addition to the integer part?
  » π = 3.1415926535...

Could use scaling to get fractions

• Assume that every numeric value in memory was scaled by a factor of 1000
  3000 => represents 3.000
  3010 => represents 3.010
• Problems
  » one scale factor for all numbers?
  » impossible to choose one “best” scale factor for all numbers that we might want to represent

A scale factor for each number

• This is the same as scientific notation
  » 6 x 10^9, 3.1415926535 x 10^0
• A floating point number contains two parts
  » mantissa (or significand): the value
  » exponent: the exponent of the scale factor
• Normalized form
  » a non-zero single digit before the decimal point

“Binary scientific notation”

• The computer only stores binary numbers
  » So we use powers of 2 rather than 10
  » Normalized numbers have a leading 1
• 6,000,000,000 = 6.0 x 10^9
  » 1.396983861910 x 2^{32}
• π ≈ 3.141592653589793238462643383
  » 1.57079632679489661923132169163975 x 2^1
Storage format: fixed width fields

- How big can the exponent be?
  » what is the range it represents?
- How big can the mantissa be?
  » what are the values it represents?
- We have to select a storage format and allocate specific fields to various purposes
  » single precision: one 32-bit word
  » double precision: two 32-bit words

IEEE 754 Standard

- Chaos in the 70s and 80s as each system designer chose new formats and rules
- IEEE 754 standard
  » format of the fields
  » rounding: up, down, towards 0, nearest
  » exceptional values: ±infinity, NaN (not a number)
  » action to take on exceptional values

Floating Point Storage

- Single Precision
  » one word (32 bits)
- Double Precision
  » two words (64 bits)
  » the order of the words depends on endianness of the machine being used
- Defined by IEEE 754

Single Precision Format

<table>
<thead>
<tr>
<th>s</th>
<th>exponent</th>
<th>mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
<tr>
<td>0</td>
<td>1 0 0 0 0 0 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>
Double Precision Format

Double Precision Mantissa Fields

- Sign bit
  » 1 bit sign for the value
- Mantissa
  » 52 bits for the value
  » by definition, the leading digit is always a 1
  » so we don’t need to actually store it
  » and we actually have 53 bits of information

Double Precision Exponent Field

- Field range
  » 11 bits: range $2^{11} = 2048$ possible values
- Special values
  » exponent = 2047 $\Rightarrow$ value=special (inf, NaN)
  » exponent = 0 $\Rightarrow$ value=0

Biased Notation

- Need exponent range - negative and positive
- If positive exponents are bigger numbers than the negative exponents, then floating point numbers can be sorted as integers
- Exponent is stored as (E+1023)
  » most positive exponent is +1023 (stored as 2046)
  » most negative exponent is -1022 (stored as 1)
  » this is not two’s complement notation
**Example: 6,174,015,488**

- 6174015488
  - $6.174015488 \times 10^9 = 1.437510 \times 2^{32}$
- Exponent
  - $32+1023 = 1055 = 41_{16}$
- Mantissa
  - $.437510 = .0111_2 = 7_{16}$

---

**Roundoff Error**

- Adding a very small floating point number to a very large floating point number may not have any effect
  - any one number has only 53 significant bits
- Adding a number with a fractional part to another number over and over will probably never yield an exactly integer result
  - so don’t use floating point loop indexes

---

**Loss of precision**

- These are not unusual numbers
  - 53248 and 0.0001983642578125
- Very few bits of mantissa required
- But their sum requires a mantissa with at least 32 bits or there will lost significant bits