C335
Computer Structures

Number Representation
Part #2

Dr. David R. Surma
Department of Computer and Information Sciences
Adapted from Morgan Kaufmann, Dr. L. Zhang and others
How to Represent Negative Numbers?

- So far we have only looked at unsigned numbers. How do we represent signed numbers?
  - Obvious solution: define leftmost bit to be sign!
    - 0 ⇒ +, 1 ⇒ –
    - Rest of bits can be numerical value of number
  - Representation called sign and magnitude
- MIPS uses 32-bit integers. \(+1_{\text{ten}}\) would be:
  
  \[0000 0000 0000 0000 0000 0000 0000 0001\]
- And \(-1_{\text{ten}}\) in sign and magnitude would be:
  
  \[1000 0000 0000 0000 0000 0000 0000 0001\]
Shortcomings of sign-magnitude representation

- Arithmetic circuit complicated
  - Special steps depending whether signs are the same or not
  - Extra step to set the sign
- Also, there are two representations for zero!
  - $0x00000000 = +0_{\text{ten}}$
  - $0x80000000 = -0_{\text{ten}}$
  - What would two 0s mean for programming?

Therefore sign-magnitude representation is abandoned.
Another try: complement the bits

- Example: \(7_{10} = 00111_2\) \(-7_{10} = 11000_2\)
- Called One’s Complement
- Note: positive numbers have leading 0s, negative numbers have leading 1s.

- What is \(-00000\)?
  - Answer: 11111
- How many positive numbers in \(N\) bits?
- How many negative numbers?
Shortcomings of One’s complement

- Arithmetic still somewhat complicated.
- Still two zeros
  - $0\times00000000 = +0_{ten}$
  - $0\timesFFFFFFFF = -0_{ten}$
- Although used for awhile on some computer products, one’s complement was eventually abandoned because another solution was better.
What is result for unsigned numbers if try to subtract large number from a small one?

- Would try to borrow from string of leading 0s, so result would have a string of leading 1s
  - $3 - 5 \Rightarrow 00\ldots0011 - 00\ldots0101 = 11\ldots1110$

- With no obvious better alternative, pick representation that made the hardware simple

- As with 1’s complement, leading 0s $\Rightarrow$ positive, leading 1s $\Rightarrow$ negative
  - $000000\ldotsxxx \geq 0$, $111111\ldotsxxx < 0$
  - except $1\ldots1111$ is -1, not -0 (as in 1’s complement.)

This representation is **Two’s Complement**
2’s Complement Number “line”: N = 5

- $2^{N-1}$ non-negatives
- $2^{N-1}$ negatives
- One zero
- How many positives?

00000 00001 01111
11111 11110 10000
10001
10000 10001 10010 10011 10100 10101 10110 10111 11000 11001 11010 11011 11100 11101 11110 11111

00000 00001 01111
11111 11110 10000
10001
10000 10001 10010 10011 10100 10101 10110 10111 11000 11001 11010 11011 11100 11101 11110 11111
Two’s Complement for N=32

0000 ... 0000 0000 0000 0000\text{two} = 0\text{ten}

0000 ... 0000 0000 0000 0001\text{two} = 1\text{ten}

0000 ... 0000 0000 0000 0010\text{two} = 2\text{ten}

... 

0111 ... 1111 1111 1111 1101\text{two} = 2,147,483,645\text{ten}

0111 ... 1111 1111 1111 1110\text{two} = 2,147,483,646\text{ten}

0111 ... 1111 1111 1111 1111\text{two} = 2,147,483,647\text{ten}

1000 ... 0000 0000 0000 0000\text{two} = −2,147,483,648\text{ten}

1000 ... 0000 0000 0000 0001\text{two} = −2,147,483,647\text{ten}

1000 ... 0000 0000 0000 0010\text{two} = −2,147,483,646\text{ten}

... 

1111 ... 1111 1111 1111 1101\text{two} = −3\text{ten}

1111 ... 1111 1111 1111 1110\text{two} = −2\text{ten}

1111 ... 1111 1111 1111 1111\text{two} = −1\text{ten}

- One zero; 1st bit called sign bit
- 1 “extra” negative: no positive 2,147,483,648\text{ten}
Two’s Complement Formula

- 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: \( 1101_{\text{two}} \)
  \[ = 1 \times -(2^3) + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \]
  \[ = -2^3 + 2^2 + 0 + 2^0 \]
  \[ = -8 + 4 + 0 + 1 \]
  \[ = -8 + 5 \]
  \[ = -3_{\text{ten}} \]
Two's Complement shortcut: Negation

- Change every 0 to 1 and 1 to 0 (invert or complement), then add 1 to the result

- Proof*: Sum of number and its (one's) complement must be $111\ldots111_{\text{two}}$
  
  However, $111\ldots111_{\text{two}} = -1_{\text{ten}}$
  
  Let $x' \Rightarrow$ one's complement representation of $x$
  
  Then $x + x' = -1 \Rightarrow x + x' + 1 = 0 \Rightarrow -x = x' + 1$

- Example: -3 to +3 to -3

<table>
<thead>
<tr>
<th>$x$ :</th>
<th>1111 1111 1111 1111 1111 1111 1111 1101$_{\text{two}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x'$ :</td>
<td>0000 0000 0000 0000 0000 0000 0000 0010$_{\text{two}}$</td>
</tr>
<tr>
<td>$+1$:</td>
<td>0000 0000 0000 0000 0000 0000 0000 0011$_{\text{two}}$</td>
</tr>
<tr>
<td>()' :</td>
<td>1111 1111 1111 1111 1111 1111 1111 1100$_{\text{two}}$</td>
</tr>
<tr>
<td>$+1$:</td>
<td>1111 1111 1111 1111 1111 1111 1111 1101$_{\text{two}}$</td>
</tr>
</tbody>
</table>
Exercise

- Convert 0b11100111 to decimal (assume 2’s complement is used)
Two’s comp. shortcut: Sign extension

- Convert 2’s complement number rep. using n bits to more than n bits

- Simply replicate the most significant bit (sign bit) of smaller to fill new bits
  - 2’s comp. positive number has infinite 0s
  - 2’s comp. negative number has infinite 1s
  - Binary representation hides leading bits; sign extension restores some of them
  - 16-bit $-4_{\text{ten}}$ to 32-bit:
    
    $\begin{align*}
    &1111 1111 1111 1100_{\text{two}} \\
    &1111 1111 1111 1111 1111 1111 1100_{\text{two}}
    \end{align*}$
What if too big?

- Binary bit patterns above are simply **representatives** of numbers. Strictly speaking they are called “numerals”.

- Numbers really have an $\infty$ number of digits
  - with almost all being same (00…0 or 11…1) except for a few of the rightmost digits
  - Just don’t normally show leading digits

- If result of add (or -, *, / ) cannot be represented by these rightmost bits, **overflow** is said to have occurred.

```
  00000  00001  00010  11110  11111
```

unsigned
Number summary...

- We represent “things” in computers as particular bit patterns: \( N \) bits \( \Rightarrow 2^N \)

- Decimal for human calculations, binary for computers, hex to write binary more easily

- 1’s complement - mostly abandoned

- 2’s complement universal in computing: cannot avoid, so learn

- Overflow: numbers \( \infty \); computers finite, errors!
Exercise

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

A. X > Y (if signed)
B. X > Y (if unsigned)
C. To identify 30 registers, you need at least 5 bits
Exercise

- Convert \(-98_{10}\) to binary (8 bits), then to hexadecimal (assume 1’s complement is used)

- Convert \(-98_{10}\) to binary (8 bits), then to hexadecimal (assume 2’s complement is used)
Exercise

- Convert 0xFFB5 to binary then to Decimal (assume 1’s complement is used)

- Convert 0xFFB5 to binary then to Decimal (assume 2’s complement is used)