Computer Systems Architecture
http://cs.nott.ac.uk/~txa/g51csa/

Thorsten Altenkirch and Liyang Hu

School of Computer Science
University of Nottingham

Lecture 06: Binary Addition and Signed Numbers
# Long Addition

## Long Addition in Decimal

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>2</th>
<th>9</th>
<th>5</th>
<th>1</th>
<th>4</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

## Long Addition in Binary

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

76\textsubscript{16} = 118  
D5\textsubscript{16} = 213  
14B\textsubscript{16} = 331
Single-Bit Adders

**Half-Adder**

\[ S = A \oplus B \]
\[ C = A \land B \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Full-Adder**

\[ S = A \oplus B \oplus C_{in} \]
\[ C_{out} = (A \land B) \lor (B \land C_{in}) \lor (A \land C_{in}) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_{in}</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Ripple Carry Adder

- A chain of full-adders can sum as many bits as needed
  - The $C_{\text{out}}$ of each adder is wired to the $C_{\text{in}}$ of the next
  - Present the $n^{\text{th}}$ bit of inputs $A$ and $B$ to the $n^{\text{th}}$ adder
  - Set $C_0$ to 0; carry out $C_4$ indicates overflow
  - Calculates $S = A + B$, where $S_n$ is the $n^{\text{th}}$ bit of $S$...
- The carry bits *ripples* across the full adders
- Simplest, but not the fastest (c.f. carry look-ahead)

4-Bit Ripple Carry Adder
Simulating addition in C

- **Inputs** $x, y$;
- **Initial values**
  
  $ci=0$;
  $z=0$;

  ```c
  for(i=0;i<sizeof(x)*8;i++) {
    xi=x&(1<<i);
    yi=y&(1<<i);
    zi=xi^yi^ci;
    csi=(xi&yi)|(yi&ci)|(xi&ci);
    z=z|zi;
    ci=csi;
  }
  ```

- **Output**: $z$.
- **For illustration only!**
  Nobody would implement addition in software!
Representing Negative Numbers

- How are negative numbers represented on a computer?
- Sign and Magnitude
  - What we use in decimal notation: +/- and 0, 1, 2, ...
- Ones’ Complement
  - Represent $-x$ by its bitwise inverse (NOT)
- Excess-$n$
  - Zero represented by binary $n$: $0_2$ is thus $-n$
- Two’s Complement
  - For an $n$-bit number, represent $-x$ by $2^n - x$
- How do we add two integers in the above systems?
# Example of 4-Bit Signed Encodings

<table>
<thead>
<tr>
<th>Sign and Mag.</th>
<th>Ones' Comp.</th>
<th>Excess-3</th>
<th>Two’s Comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>1000</td>
<td>0000</td>
<td>1000</td>
</tr>
<tr>
<td>1110</td>
<td>1001</td>
<td>0001</td>
<td>1001</td>
</tr>
<tr>
<td>1101</td>
<td>1010</td>
<td>0010</td>
<td>1010</td>
</tr>
<tr>
<td>1100</td>
<td>1011</td>
<td>0011</td>
<td>1011</td>
</tr>
<tr>
<td>1011</td>
<td>1100</td>
<td>0100</td>
<td>1100</td>
</tr>
<tr>
<td>1010</td>
<td>1101</td>
<td>0101</td>
<td>1101</td>
</tr>
<tr>
<td>1001</td>
<td>1110</td>
<td>0110</td>
<td>1110</td>
</tr>
<tr>
<td>1000</td>
<td>1111</td>
<td>0111</td>
<td>1111</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
<td>1001</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
<td>1010</td>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
<td>1011</td>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
<td>1100</td>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
<td>0101</td>
<td>1101</td>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
<td>1110</td>
<td>0110</td>
</tr>
<tr>
<td>0111</td>
<td>0111</td>
<td>1111</td>
<td>0111</td>
</tr>
</tbody>
</table>
Method of Complements

- In base $b$ with $n$ digits, represent $-x$ using $b^n - x$
- But there’s a quicker way to calculate $b^n - x$:
  - Subtract each digit of $x$ from $b - 1$, then add 1 to result
- Addition as for unsigned numbers; just ignore carry digit

**Calculate 8192 − 4235 Using Ten’s Complement**

9999

\[
\begin{array}{c}
\text{9999} \\
{-4235} \\
\hline
\text{5764}
\end{array}
\]

Negate: $5764$, then add: $5765$

\[
\begin{array}{c}
\text{5764} \\
+1 \\
\hline
\text{5765}
\end{array}
\]

Drop the carry digit: thus $8192 − 4235 = 3957$
Two’s Complements

- In base 2 with $n$ bits, represent $-x$ using $2^n - x$
- But there’s a quicker way to calculate $2^n - x$:
  - Take the bitwise inverse (NOT) of $x$, then add 1 to result
- Addition as for unsigned numbers; just ignore carry bit
- Conventionally we take the MSB to represent the sign
  - 0 for positive, 1 for negative
- Thus $n$ bits can represent from $-2^{n-1}$ up to $2^{n-1} - 1$
- All modern processors primarily use two’s complement
  - Includes all the MIPS’s integer arithmetic instructions
  - Different instruction variants for signed and unsigned
  - But floating point (future lecture) uses *sign and magnitude* as well as *excess-$n$*
Alternative View

8-Bit Two’s Complement ($-128 \leq x < 127$)

<table>
<thead>
<tr>
<th>Bit</th>
<th>$7^{th}$</th>
<th>$6^{th}$</th>
<th>$5^{th}$</th>
<th>$4^{th}$</th>
<th>$3^{rd}$</th>
<th>$2^{nd}$</th>
<th>$1^{st}$</th>
<th>$0^{th}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>$-2^7$</td>
<td>$2^6$</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
</tbody>
</table>

Exercises

- Calculate $81 - 42$ in 8-bit two’s complement
- We can negate $s0$ using `nor $s0$, $zero$, $s0`
  `addi $s0$, $s0$, 1`
  For what number will this not work?
Thursday quiz

Most of the following questions are multiple choice. There is at least one correct choice but there may be several. For each of the questions list all the roman numerals corresponding to correct answers but none of the incorrect ones.

Questions are marked as follows:

- **no errors**: 5 points
- **1 error**: 3 points
- **2 errors**: 1 point
- **≥3 errors**: 0 points
1. Which of the following statements about MIPS machine code are correct?
   a. MIPS machine code consists of symbolic instructions?
   b. Each instruction is 32 bytes long.
   c. There are 3 basic formats: R,I,J.
   d. The opcode always occupies the leftmost 6 bits.
   e. The position of the opcode varies with the instruction.
Thursday quiz

2. What do we mean by *data transfer instructions*?
   a. Instructions to load memory data into registers.
   b. Instructions to store registers into memory.
   c. Instructions to read data from a keyboard.
   d. Instructions to write data on the console.
   e. Instructions to transfer data over the internet.
2. What do we mean by *data transfer instructions*?
   a. Instructions to load memory data into registers.
   b. Instructions to store registers into memory.
   c. Instructions to read data from a keyboard.
   d. Instructions to write data on the console.
   e. Instructions to transfer data over the internet.
3. What does the following program fragment do?

```assembly
num: .word 0
...
la $s0,num
li $t0,1
sb $t0,1($s0)
lw $t1,($s0)
```

- It stores 1 in register $t1.
- It stores 256 in register $t1.
- It modifies a byte in memory.
- It divides the content of $t0 by 256.
- It multiplies the content of $t0 by 256.
Thursday quiz

4. What can we say about goto?

   a. It is the recommended way to implement loops in high level languages.

   b. Dijkstra wrote a letter titled *Goto Statment considered useful*.

   c. In machine language goto statments (or jumps) are used to implement control flow.

   d. Java has a goto statement.

   e. C++ has a goto statement.
Thursday quiz

5. What is bge in MIPS assembler code?
   a. A machine instruction.
   b. A pseudoinstruction.
   c. A conditional jump.
   d. An unconditional jump.
   e. A data transfer instruction.