Johns Hopkins Engineering

Number Systems for Computation

EN605.204 Computer Organization

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Introduction

- What are bits and how do we interpret them?
- How do we represent data using bits?
- Positive and negative numeric representations
- Number systems: binary, octal, hexadecimal
- Arithmetic in different number systems
- Integer overflow

What is a “bit”?

- A “bit” is a “binary bit: 0/1, True/False, On/Off
- Other type of “_its”:
  - “trit”: trinary bit = {0, 1, 2}
  - “dit”: decimal bit = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - “qubit”: quantum bit = {superpositions of bits}
- Computers implement bits as a high or low voltage
  - -0.5V = 0
  - +0.5V = 1

Interpreting a Stream of Bits

- Computers can stream billions of bits per seconds
- System software decomposes these streams into “instructions”
  - this process is known as “decoding” (more later)
- MIPS instructions are 32-bits/4-bytes long
  - An 8-bit entity is called a “byte”
  - A 4-byte entity is called a “word”
- Before we decode instructions, let’s understand binary number systems

How Binary Works

- Binary is base 2, meaning each digits is a 0 or 1
- Ex: $1101_2 = \underline{1}x2^3 + \underline{1}x2^2 + \underline{0}x2^1 + \underline{1}x2^0 = 8+4+0+1 = 13$
  - Each column goes up a single power or 2 from right to left
- With N bits we can represent N-1 values from 0 to 2^N-1
  - N = 6 bits we can represent 2^6=64 values: {0…63}
- “Unsigned" numbers are always positive
- “Signed” numbers must have the sign (+/-) specified explicitly
- Let’s look at how to represent “signed” numbers…

Sign & Magnitude Numbers

- Allows us to represent “signed” numbers in binary
- The leading bit is used ONLY to represent the sign
- Ex: $4_{10} = 0100_2$, so $-4_{10} = 1100_2$
- Pros:
  - Easy to implement in hardware
- Issues:
  - An entire bit is wasted which reduces the range of numbers we can represent
  - Two zeros: +010 = 00002, -010 = 11112

One’s Complement Numbers

- Allows us to represent “signed” numbers in binary
- “Complement” just means “flipping bits”:
  - 0’s become 1’s and 1’s become 0 (easy!)
- Ex: $4_{10} = 0100_2$, so $-4_{10} = 1011_2$
- Pros:
  - Easy to implement in hardware
- Issues:
  - Two zeros: $+0_{10} = 0000_2$, $-0_{10} = 1111_2$

Two's Complement Numbers

- Allows us to represent negative numbers in binary
- Flip all of the bits and add 1
- Ex: $4_{10} = 0100_2$, so $-4_{10} = 1011_2$ (flip) + 1_2 (add 1) = 1100_2
- Pros:
  - Easy to implement in hardware
  - A single, standard value for 0
- Most computers today use two's complement numbers to represent signed numbers!

Examples of Signed Numbers

- One's Complement
  - 4-bit ex: $+7_{10} = 0111_2$, so $-7_{10} = 1000_2$
  - 5-bit ex: $+11_{10} = 01011_2$, so $-11_{10} = 10100_2$

- Two's Complement
  - 4-bit ex: $+5_{10} = 0101_2$, so $-5_{10} = 1010_2 + 0001_2 = 1011_2$
  - 5-bit ex: $+15_{10} = 01111_2$, $-15_{10} = 10000_2 + 00001_2 = 10001_2$

Pro Tip

- You don’t convert numbers to/from 1’s or 2’s complement, you just interpret them differently:
- Examples:
  - 1101_2 (1’s comp.) = $-(0010)_2$ (flip all bits) = -2_{10}
  - 1101_2 (2’s comp.) = $-(0011)_2$ (flip all bits, add 1) = -3_{10}

Rules for Binary Addition

Addition		Result	Carry
0 + 0	=	0	0
0 + 1	=	1	0
1 + 0	=	1	0
1 + 1	=	0	1

Binary Addition Example

- Two ways to perform addition:
  - Short method: add the bits directly (shown below)
  - Long method: convert the numbers to base 10, add them, convert the sum back to binary

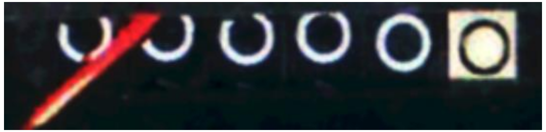
	1	1	1	1		← carry
	1	1	1	0	1	
(+)	1	1	0	1	1	
1	1	1	0	0	0	

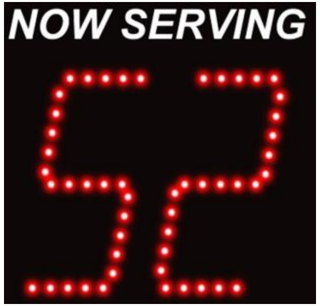
Integer Overflow

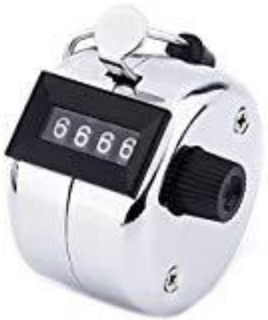
- Overflow occurs when the result of an arithmetic operation exceeds the maximum value that can be represented with the given number of bits
- Recall: given N-bits, we can represent values from 0 to 2^N-1 (unsigned)

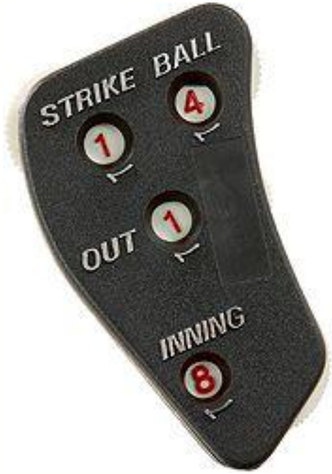
- Example: let's add two unsigned 3-bit numbers together
  - $101_2 + 101_2 = 1010_2 = 10_{10}$
  - The result is too large to represent using only 3 bits and requires a 4th bit; the lead bit is truncated leaving us with $010_2 = 2_{10}$, which is not the correct answer

Integer Overflow – Practical Examples









Octal Numbers

- Base 8
- Values 0-7: 0, 1, 2, 3, 4, 5, 6, 7
  - One octal digit requires 3 bits
- Examples:
  - $14_{10} = 16_8 = 001\ 110_2$
  - $26_{10} = 32_8 = 011\ 010_2$
  - 55_8 (1's complement) $= 101\ 101_2 = -010\ 010_2 = -18_{10}$

Hexadecimal Numbers

- Base 16, commonly referred to as “hex” numbers
  - Values 0-15: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - One hex digit requires 4 bits, 2 hex digits require 1 byte
- Examples:
  - $14_{10} = E_{16} = 1110_2$
  - $26_{10} = 1A_{16} = 00011010_2$
  - CC_{16} (1’s complement) = $1100\ 1100_2 = -0011\ 0011_2 = -51_{10}$
  - $-42_{10} = -2A_{16} = -0010\ 1010_2 = 1101\ 0110_2$ (2’s complement)

Rules for Hexadecimal Addition

- Start adding Hex. Digits from right to left.
- If sum of two Hex. Digits is greater than 15, then divide the sum by Hex. base (16). The quotient becomes the carry value, and the remainder is the sum digit.

		1	1
36	28	28	6A
+ 42	45	58	4B
78	6D	80	B5

21 / 16 = 1, remainder 5