Number Representation

COMP375 Computer Organization
and Architecture

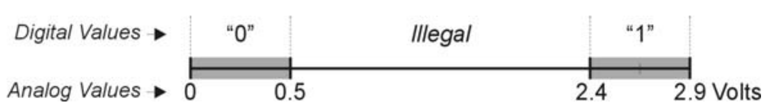
How do we represent data in a computer?

- At the lowest level, a computer is an electronic machine.
  - works by controlling the flow of electrons
- Easy to recognize two conditions:
  1. presence of a voltage – we'll call this state "1"
  2. absence of a voltage – we'll call this state "0"
- Could base state on *value* of voltage, but control and detection circuits more complex.
  - compare turning on a light switch to measuring or regulating voltage

Computer is a binary digital system.

Digital system:
- finite number of symbols

Binary (base two) system:
- has two states: 0 and 1



- Basic unit of information is the *binary digit*, or *bit*.

What kinds of data do we need to represent?

- Numbers – signed, unsigned, integers, floating point, complex, rational, irrational, …
- Text – characters, strings, …
- Images – pixels, colors, shapes, …
- Sound
- Logical – true, false
- Instructions
- …

Integer Numbers

- Integers are almost universally stored as binary numbers.
- When you enter an integer from the keyboard, software converts the ASCII or Unicode characters to a binary integer.

Integers

- Weighted positional notation
  - like decimal numbers: “329”
  - “3” is worth 300, because of its position, while “9” is only worth 9

329

10^2 10^1 10^0

3x100 + 2x10 + 9x1 = 329

most significant 101 *least significant*

2^2 2^1 2^0

1x4 + 0x2 + 1x1 = 5

Binary Fractions

- Each position is twice the value of the position to the right.

2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
8	4	2	1	.	1/2	1/4	1/8
1	0	1	0	.	0	0	1

What is this number in decimal?

Binary Arithmetic

- Base-2 addition – just like base-10
  - add from right to left, propagating carry

```
  10010        10010         1111
+  1001      +  1011      +     1
  11011        11101        10000
```

carry

```
  10111
+   111
```

Negative Integers

- Almost all systems for storing negative binary numbers set the left most bit (MSB) to indicate the sign of a number.

Common formats:
- –Signed Magnitude
- –Ones Complement
- –Twos Complement

Signed Magnitude

- Negative number are the same as positive with the sign bit set

Three bit example

000	001	010	011	100	101	110	111
0	1	2	3	-0	-1	-2	-3

•There are two zeroes, positive and negative.

Signed Magnitude Arithmetic

- Normal addition does not work for negative signed magnitude numbers.

```
  1001         -1
+ 0100         +4
  1101  (-5)    3
```

Ones Complement

- Negative number are the logical inverse of positive numbers

Three bit example

000	001	010	011	100	101	110	111
0	1	2	3	-3	-2	-1	-0

Mathematically positive and negative zero are the same, but they are different bit patterns.

Ones Complement Arithmetic

- The carry out of the sign position needs to be added to the number

```
carry
 101      -2
+110    + -1
 011
  +1    add carry out of sign
 100      -3
```

Twos Complement

- Negative number are the logical inverse of positive numbers plus 1.

Three bit example

000	001	010	011	100	101	110	111
0	1	2	3	-4	-3	-2	-1

Normal binary arithmetic works for positive and negative numbers.

Two's Complement Representation

- If number is positive or zero,
  - normal binary representation, zero in upper bit
- If number is negative,
  - start with positive number
  - flip every bit (i.e., take the one's complement)
  - then add one

```
  00101  (5)          01010  (10)
  11010  (1's comp)   10101  (1's comp)
+     1             +     1
  11011  (-5)         10110  (-10)
```

Two's Complement Shortcut

- To take the two's complement of a number:
  - copy bits from right to left up to and including the first "1" bit
  - flip remaining bits to the left

```
  011010000                      011|10000
  100101111  (1's comp)   (flip)    |      (copy)
+         1
  100110000                      100|10000
```

Two's Complement Signed Integers

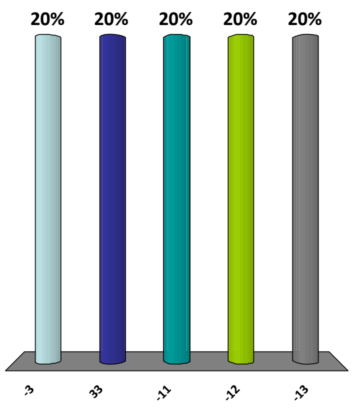
2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

2^3	2^2	2^1	2^0	
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

What is the decimal value of the 6 bit two's complement number?

110011

A. -3
B. 33
C. -11
D. -12
E. -13



2's Complement Addition

- Use normal binary addition regardless of sign.
- Ignore carry out

```
  01101000 (104)      11110110  (-10)
+ 11110000 (-16)    + 11110111  (-9)
  01011000 (98)       11101101  (-19)
```

Assuming 8-bit 2's complement numbers.

Subtraction

- Negate subtrahend (2nd number) and add.

```
  00001000      8
- 00000101     -5

  00001000      8
+ 11111011   + -5
  00000011      3
```

Assuming 8-bit 2's complement numbers.

Sign Extension

- When moving an integer into a larger register the upper bits must be set to the sign bit.
- If we just pad with zeroes on the left:

4-bit		**8-bit**	
0100	(4)	00000100	(still 4)
1100	(-4)	00001100	(12, not -4)

- Instead, replicate the MS bit -- the sign bit:

4-bit		**8-bit**	
0100	(4)	00000100	(still 4)
1100	(-4)	11111100	(still -4)

Overflow

- If operands are too big, then sum cannot be represented as an *n*-bit 2's complement number.

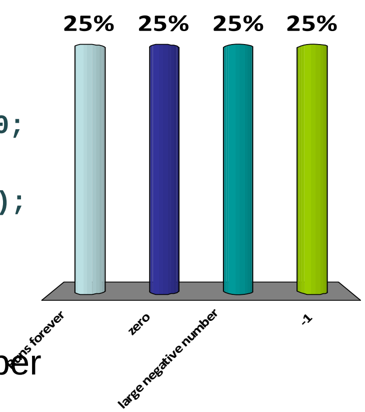
```
  carry into sign bit        carry out of sign bit
   01000  (8)                 11000  (-8)
+  01001  (9)              +  10111  (-9)
   10001  (-15)               01111  (+15)
```

- We have overflow if:
  - signs of both operands are the same, and
  - sign of sum is different.
- Another test -- easy for hardware:
  - carry into left most bit does not equal carry out

What is printed by this program?

```
int num;
num = 100000000;
while (num > 0) {
  num = num + 100000000;
}
System.out.println(num);
```

1. Runs forever
2. zero
3. large negative number
4. -1



Decimal Numbers

- Some systems *(i.e. Intel Pentium)* support decimal numbers in packed or unpacked format.

packed decimal uses 4 bit fields for each digit

9375 =

1001,0011,0111,0101

unpacked decimal uses a byte per digit (ASCII)

9375 =

00111001,00110011,00110111,00110101

Number Order

- With normal binary numbers, many bits may change from number to number as you count up.

	bits changed
000	
001	1
010	2
011	1
100	3
101	1
110	2
111	1

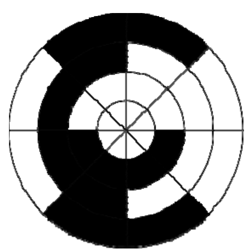
Gray Code

- With gray code numbers, only 1 bit changes from number to number as you count up.

	bits changed
000	
001	1
011	1
010	1
110	1
111	1
101	1
100	1

Gray Code Input

- When reading sensors to measure the position of something, gray codes reduce the probability of incorrect input.



Counting in Gray Code

- To make the list twice as long, copy and reverse the list and append a 1 bit to the left.