Computer Systems Principles

Data Representation- Bits and Bytes

UMASSCS
SCHOOL OF COMPUTER SCIENCE

Today’s Class

1. Learn data representation in binary and hexadecimal
   – represent real numbers, negative numbers and fractions.
2. Perform operations
   - addition, subtraction, multiplication, and division

• Announcements: Moodle Quiz 2 is up.

Group Task

- Form partners
- Using only the three symbols @#& represent:
  - integers 0 – 10
  - represent movements:

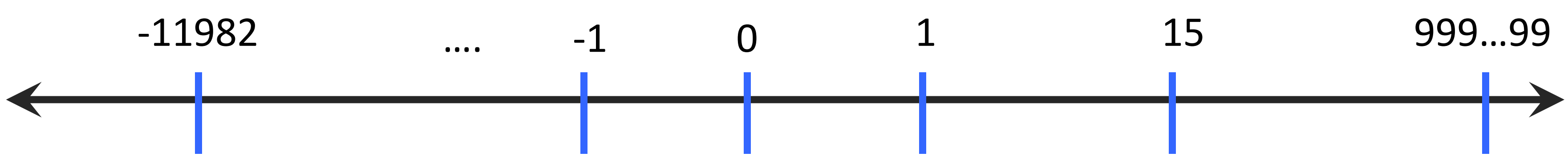
  1) stand, 2) sit, 3) walk , 4)raise hand, 5) turn around 6) hop
- Using only your representation write down a series of commands and integers (raise arm - 3, turn - 2)
  - must have at least 5 commands

How do we think about numbers?

- Representing and reasoning numbers?
  - Computers store variables (data).
  - Data is typically composed of numbers and characters.
  - Want a representation that is sensible.

Decimal Number Systems

- Digits 0-9
- Positional
- Digits are composed to make larger numbers
  $12 = 1*10^1+2*10^0$
- Base 10 representation

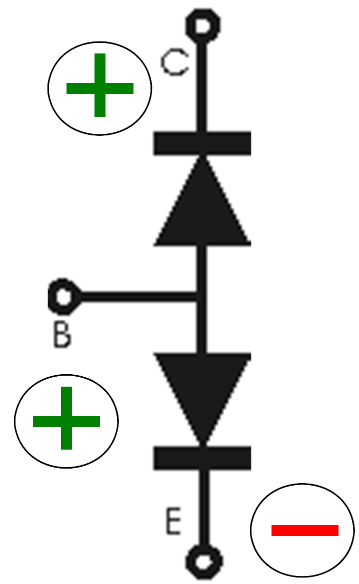


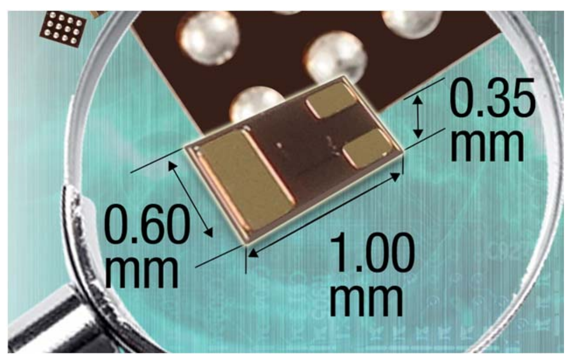
Given four positions: $[X\ X\ X\ X]_{10}$ what is the largest number you can represent?

- 9999_{10}

Binary Number Systems

- Transistors: On or Off
- Optical: Light or No light
- Magnetic: Positive or Negative





Binary Digits: (BITs)

- Everything is bits
- Thinking in Base 2

Dec	0	1	2	3	4	…	10	…
Bin	0	01	10	11	100	…	1010	…

The number line

Binary Digits: (BITS)

$$\begin{array}{c} 7\ 6\ 5\ 4\ 3\ 2\ 1\ 0 \\ \underline{1}000111\underline{1} \end{array}$$

Most significant bit → $\underline{1}000111\underline{1}$ ← Least significant bit

Representation:

$$1 \times 2^7 + 0 \times 2^6 \ldots\ldots + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$10001111 = 143$$

$$01011000\ 00010101\ 00101110\ 11100111$$

Represents: $0 \times 2^{31} + 1 \times 2^{30} + 0 \times 2^{29} + \ldots + 1 \times 2^0$

- Sequence of eight bits: byte
- Sequence of four bits: nibble

Conversion: Decimal to Binary

Method 1:

We have a decimal number (x):

1. Highest power of two less than or equal to the decimal number (y)
2. Subtract the power of two (y) from the decimal number (x) as $x = x-y$.
3. If $x = 0$, we have our result! Else: Repeat

Conversion: Decimal to Binary

Method 2:

We have a decimal number (x):

1. Divide (x) by 2 to obtain a quotient and remainder.
   - Remainder is 0 or 1.
2. If quotient ≠ 0, assign (x) = quotient back to step 1.
3. If the quotient = 0 proceed!
4. The sequence of remainders is the binary representation.

Example conversion by Method 2

quotient	remainder
35	1
17	1
8	0
4	0
2	0
1	1
0	

read upward:

$35_{10} = 100011_2$

Note: remainders always 0 or 1:
Easy to see by even versus odd

Hexadecimal Representation

Translate the binary to decimal?

1001000100001011001010001101001010001101

- Hexadecimal numbers use 16 digits: {0-9, A-F}
  - does not distinguish between upper and lower case!
- Encoding byte values using Octal numbers: use 8 digits {0 – 7}

DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Hexadecimal Representation

- Bit patterns as base-16 numbers
- Convert binary to hexadecimal: by splitting into groups of 4 bits each.
- If not a multiple of 4 (nibble):
  - pad the number with leading zeros.
- Example: $11\ 1100\ 1010\ 1101\ 1011\ 0011_2 = 3CADB3_{16}$

Bin	11	1100	1010	1101	1011	0011
Hex	3	C	A	D	B	3

iClicker Activity

Convert the decimal number 231 to binary and hexadecimal equivalents.

a) 1110 1111, F7

b) 1110 0110, E6

c) 1110 0111, E7

d) 0110 0111, 57

DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Binary Addition

Rules

- 1+0 = 1
- 1+1 = 10
- 1+1+1 = 11

	1	1	1	0		Carry in
		1	0	1	1	Augend (A)
+		1	1	1	0	Addend (B)
	1	1	0	0	1	Sum

Binary Multiplication

Example:

		1	1
	X	1	3
		3	3
+	1	1	-
	1	4	3

					1	0	1	1
				X	1	1	0	1
					1	0	1	1
				^{1}0	0	0	0	-
			^{1}1	0	1	1	-	-
+		^{1}1	0	1	1	-	-	-
	1	0	0	0	1	1	1	1

- 1+0 = 1
- 1+1 = 10

Binary Multiplication

- Since we always multiply by either 0 or 1, the partial products are always either 0000 or the multiplicand (in this example: 1011).
- There are four partial products which are added to form the result.

iClicker Activity

Multiply the two numbers: 1101 x 1001

a) 110 1001
b) 100 1111
c) 110 1101
d) 111 0101

iClicker Activity

- Largest binary integer that can be stored in 3 bits?

a) 001

b) 100

c) 111

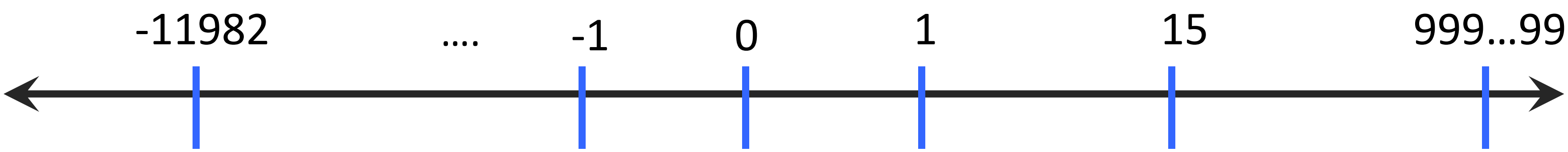
d) None of these

What about the smallest?

Unsigned Binary representation

- Issues:
  - complexity of arithmetic operations
  - negative numbers
  - maximum representable number

Negative Binary Numbers



- Which bit should we pick?
  - Left-most bit
  - Also called the most significant bit

Negative Binary: Sign-Value

To get the negative number, set the sign bit to 1 and leave all the other bits unchanged.

81 =	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1
-81=	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1

Called sign-magnitude representation: sign bit + value

0 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 =	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Two zeroes! Not fun ….

More difficulties …

- It is not easy to add, subtract, or compare numbers in sign-magnitude format
- Everything get complicated by whether the signs of the numbers are the same or different, etc.
- Values do not fall in a natural order as compared with unsigned numbers

An alternative: Ones Complement

To get the negative of a number, invert or complement all the bits.

Complement = (~)

- ~1 = 0 and ~0 = 1

81 =	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1
-81=	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0

This solves the "unnatural order" problem

0 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

But we still have two zeroes!

Two's Complement

To get the negative of a number, invert all the bits and then add 1.

If the addition causes a carry bit past the most significant bit, discard the high carry.

81 =	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1

~81 =	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0

+1

-81 =	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1

Two's Complement

Do we have a unique representation of the value zero? Yes!

0 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
~0 =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
																+1
-0 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Two’s Complement: Advantages

- ✓ Unique representation of zero
- ✓ Exactly same method for addition, multiplication, etc. as unsigned integers except throw away the high carry (high borrow for subtraction).
- ✓ Also turns out to lead to reasonable simple electronic circuits: nice for building CPUs!

Two's Complement: Addition, Subtraction

45 – 14 = 45 + (-14) = 31

45 =	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1
-14 =	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0
31 =	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

ignore high carry 1!

Two's Complement Range

- Representing negative and positive numbers in b bits:
  - Unsigned: 0 to 2^b-1
  - Signed: $-2^{(b-1)}$ to $2^{(b-1)}$ -1
- Example: range for 8 bits is:
  - Unsigned: 0 to 2^8 - 1 = 0 …. 255
  - Signed: $-2^{(7)}$ to $2^{(7)}$ -1 = -128 …. 127
- Can think of the numbers as being arranged in a large circle
  - Signed and unsigned simply start at different places …
  - … and label differently the values with high bit of 1

Overflow & Underflow

- For an unsigned number: overflow happens when the last carry cannot be accommodated.
- For a signed number: overflow happens when the most significant bit is not the same as every bit to its left.
  - sum of two positive numbers is negative: overflow
  - sum of two negative numbers is positive: underflow
  - sum of a positive and negative number will never overflow!

Overflow: Example

Consider the 8-bit two’s complement addition:

				1	1	1	1	1	1	
1	2	7	0	1	1	1	1	1	1	1
	+	1							+	1
1	2	8	1	0	0	0	0	0	0	0

- The result should be +128, but the leftmost bit is 1, therefore the result is -128!
- This is an overflow: an arithmetic operation that should be positive gives a negative result.

Two's Complement Underflow

- Just as overflow is caused by a value near the upper limit of the range, an underflow is caused by values near the lower limit of the range.

iClicker Activity

- What is the result of the following 8-bit two’s complement addition 1000 0000 - 1?

a) 0111 1111

b) 1111 1111

c) 0000 0000

d) 0000 0001

Hint: Try converting it back to decimal and compare!

Underflow: Example

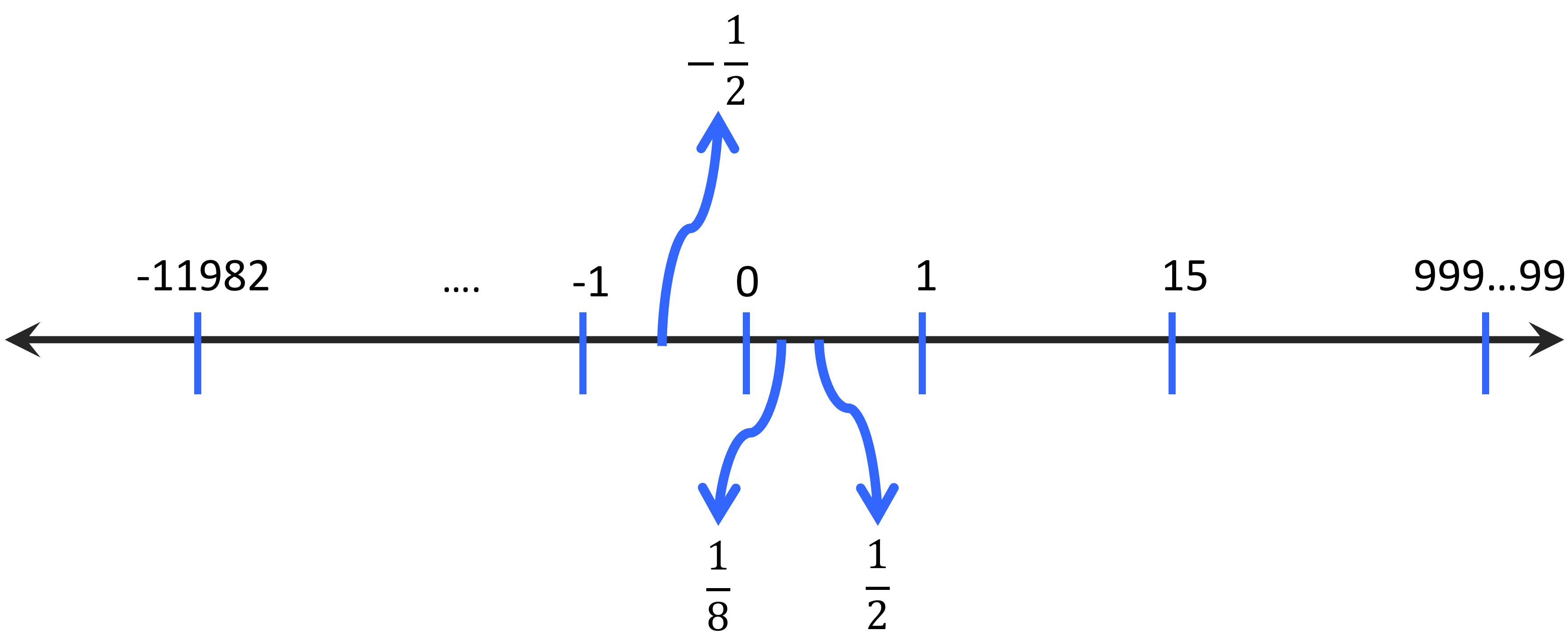
Consider the 8-bit two's complement addition:

-	1	2	8		1	0	0	0	0	0	0	0
		-	1	-								1
-	1	2	9		0	1	1	1	1	1	1	1

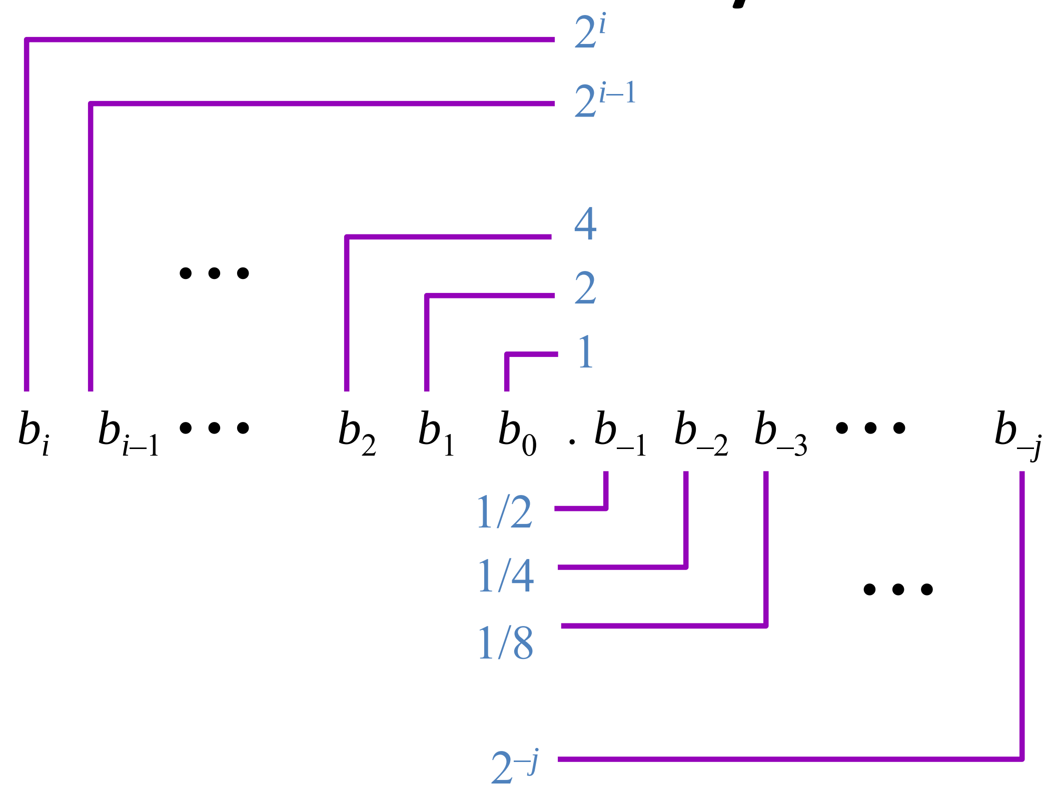
- The result should be -129, but the leftmost bit is 0, therefore the result is +127!
- This is an underflow: as an arithmetic operation that should be negative gives a positive result.

Fractional binary numbers

How do we represent fractions in binary?



Fractional Binary Numbers



- Representation
  - Bits to right of "binary point" represent fractional powers of 2
  - Represents the rational number: $$\sum_{k=-j}^{i} b_k \cdot 2^k$$

IEEE Single Precision Floating point

- sign bit
- fraction
- exponent

s	exp	frac

- $(-1)^s \times M \times 2^E$
  1. Sign bit s determines whether number is negative or positive
  2. Fractional bit M normally a fractional value in range [1.0,2.0).
  3. Exponent E weights value by power of two

Next Class

- Lets represent Binary operations in C!
- Overflow conditions
  - How do we represent overflow?
  - What's the impact of an overflow?
  - How do we mitigate it?
- Readings posted on website