Binary Arithmetic, Bitwise Operations

CS 2130: Computer Systems and Organization 1
January 25, 2023

Announcements

- My Office Hours
    - Wednesdays 2:30-4:30pm, Rice 210
    - Thursdays 2-3pm, Discord
    - *This week only: Wed until 4:15, Thurs in Rice 210*
- TA Office Hours starting soon
- Discord link coming soon
- Homework 1 due Feb 6 (Mon)

Numbers

From our oldest cultures, how do we mark numbers?

- Arabic numerals
  - Positional numbering system
  - The `10` is significant:
    - 10 symbols, using 10 as base of exponent
  - The `10` is *arbitrary*
  - *We can use other bases!* π, 2130, 2, ...

2130

Bases

We will discuss a few in this class

- Base-10 (decimal) - talking to humans
- Base-8 (octal) - shows up occasionally
- Base-2 (binary) - most important! (we've been discussing 2 things!)
- Base-16 (hexadecimal) - nice grouping of bits

Binary

Powers of 2: $\overset{0}{1}$ $\overset{1}{2}$ $\overset{2}{4}$ $\overset{3}{8}$ $\overset{4}{16}$ $\overset{5}{32}$ $\overset{6}{64}$ $\overset{7}{128}$ $\overset{8}{256}$ $\overset{9}{512}$ $\overset{10}{1024}$

$\underset{11}{2048}$ $\underset{12}{4096}$

Any downsides to binary?

Turn 2130_{10} into base-2:

hint: find largest power of 2 and subtract

$$\begin{array}{r} \scriptstyle 1\ 1\ {}^{10}\ {}^{10} \\ 2130 \\ -\,2048 \\ \hline 00\overset{7}{\cancel{8}}2 \\ -\,64 \\ \hline 18 \\ -\,16 \\ \hline 2 \end{array} \quad = 2^{11}$$

$$\underset{2^{11}}{\underline{1}}\ \underline{0}\ \underline{0}\ \underline{0}\ \underline{0}\ \underline{1}\ \underline{0}\ \underset{\cdots}{\underline{1}}\ \underset{2^3}{\underline{0}}\ \underset{2^2}{\underline{0}}\ \underset{2^1}{\underline{1}}\ \underset{2^0}{\underline{0}}$$

Long Numbers

How do we deal with numbers too long to read?

Long Numbers

How do we deal with numbers too long to read?

- Group them by 3 (right to left)

Long Numbers

How do we deal with numbers too long to read?

- Group them by 3 (right to left)
- In decimal, use commas: **,**
- Numbers between commas: 000 - 999

Long Numbers

How do we deal with numbers too long to read?

- Group them by 3 (right to left)
- In decimal, use commas: **,**
- Numbers between commas: 000 - 999
- Effectively base-1000

$10^3 = 1000$

Long Numbers in Binary

Making binary more readable

- Typical to group by 3 or 4 bits
- No need for commas *Why?*

100,001,010,010

Long Numbers in Binary

Making binary more readable

- Typical to group by 3 or 4 bits
- No need for commas *Why?*
- We can use a separate symbol per group
- How many do we need for groups of 3?

$$\left.\begin{matrix}0\\ \vdots\\ 7\end{matrix}\right\}8$$

$$\begin{matrix}4 & 2 & 1\\ 2^2 & 2^1 & 2^0\\ 1 & 0 & 0\end{matrix}$$

4

$$2^3 = 8$$

$$\underbrace{100}_{4}\underbrace{001}_{1}\underbrace{010}_{2}\underbrace{010}_{2}\,{}_{8}$$

Long Numbers in Binary

Making binary more readable

- Typical to group by 3 or 4 bits
- No need for commas *Why?*
- We can use a separate symbol per group
- How many do we need for groups of 3?
- Turn each group into decimal representation

100001010010

Long Numbers in Binary

Making binary more readable

- Typical to group by 3 or 4 bits
- No need for commas *Why?*
- We can use a separate symbol per group
- How many do we need for groups of 3?
- Turn each group into decimal representation
- Converts binary to **octal**

$$\underset{1}{\underline{1}}\ \underset{3}{\underline{011}}\ \underset{3}{\underline{011}}\ {}_8$$

$$\underset{4}{\underline{100}}\ \underset{1}{\underline{001}}\ \underset{2}{\underline{010}}\ \underset{2}{\underline{010}}\ {}_8$$

Long Numbers in Binary

Making binary more readable

- Groups of 4 more common
- How many symbols do we need for groups of 4?

$2^4 = 16$

1000001010010

Long Numbers in Binary

Making binary more readable

- Groups of 4 more common
- How many symbols do we need for groups of 4?
- Converts binary to **hexadecimal**
- Base-16 is very common in computing

$$\underbrace{1000}_{8}\underbrace{\overset{8}{0}\overset{4}{1}\overset{2}{0}\overset{1}{1}}_{5}\underbrace{0010}_{2}{}_{16}$$

Hexadecimal

Need more than 10 digits. What next?

1110

14 = E

0
1
2
⋮
8
9
a = 10
b = 11
c = 12
d = 13
e = 14
f = 15

Hexadecimal Exercise

Consider the following hexadecimal number:

16^7 16^6 16^5 16^4 16^3 16^2 16^1 16^0

852dab1e

Is it even or odd?

Using Different Bases in Code

08

	Old Languages	New Languages
binary	no way	0b 01101 0110
octal	073	0o 725
decimal	2130	4282
hexadecimal	0x 3af	0x 3af

Finally, Numbers!

Storing Integers

- Use binary representation of decimal numbers
- Usually have a limited number of bits (ex: 32, 64)
    - Depending on language
    - Depending on hardware

Finally, Numbers!

Storing Integers

- Use binary representation of decimal numbers
- Usually have a limited number of bits (ex: 32, 64)
  - Depending on language
  - Depending on hardware
- Is there something missing?

Negative Integers

Representing negative integers

-25

Negative Integers

Representing negative integers

- Can we use the minus sign?

Negative Integers

Representing negative integers

$$\begin{array}{rrrrrr} & & 9 & 9 & 9 & \\ & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & 0 \\ - & & & & & 1 \\ \hline \end{array}$$

- Can we use the minus sign?
- In binary we only have 2 symbols, must do something else!
- Almost all hardware uses the following observation:

$$\begin{array}{r|rrrr} & 9 & 9 & 9 & 10 \\ 1 & 1 & 1 & 1 & \\ 1 & 0 & 0 & 0 & 0 \\ - & 0 & 0 & 0 & 1 \\ \hline 0 & 9 & 9 & 9 & 9 \end{array}$$

$$\begin{array}{rrrrr} & 9 & 9 & 9 & 10 \\ 1 & 1 & 1 & 1 & \\ & 0 & 0 & 0 & 0 \\ - & & & & 1 \\ \hline & 9 & 9 & 9 & 9 \end{array}$$

Negative Integers

Representing negative integers

- Computers store numbers in fixed number of wires
- Ex: consider 4-digit decimal numbers

Negative Integers

Representing negative integers

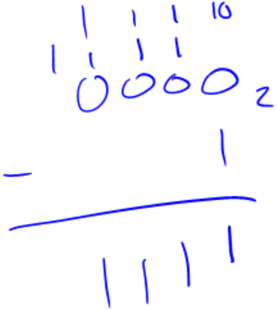
- Computers store numbers in fixed number of wires
- Ex: consider 4-digit decimal numbers
- Throw away the last borrow:
  - 0000 - 0001 = $\underline{9999}$ == -1
  - 9999 - 0001 = $\underline{9998}$ == -2
  - Normal subtraction/addition still works
  - Ex: -2 + 3

$$\begin{array}{r} 1\;1\,1\,1 \\ 9998 \\ +\quad 3 \\ \hline \boxed{0001} \end{array}$$

Negative Integers

Representing negative integers

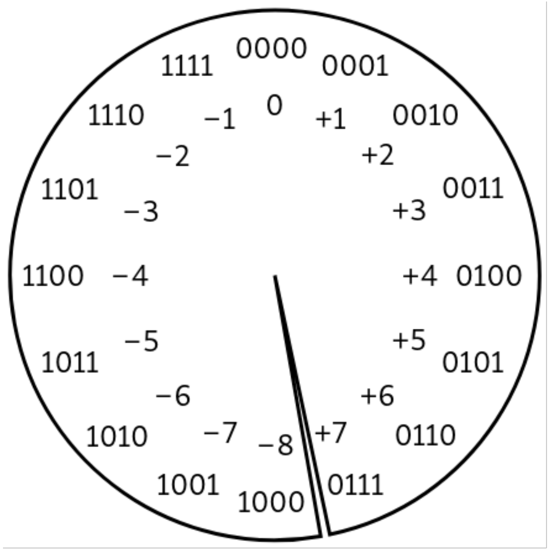
- Computers store numbers in fixed number of wires
- Ex: consider 4-digit decimal numbers
- Throw away the last borrow:
    - 0000 - 0001 = 9999 == -1
    - 9999 - 0001 = 9998 == -2
    - Normal subtraction/addition still works
    - Ex: -2 + 3
- This works the same in binary



Two’s Complement

This scheme is called **Two’s Complement**

- More generically, a *signed* integer
- There is a break as far away from 0 as possible
- First bit acts vaguely like a minus sign
- Works as long as we do not pass number too large to represent



Two’s Complement

Questions?

Values of Two's Complement Numbers

Consider the following 8-bit binary number in Two's Complement:

11010011

What is its value in decimal?

Values of Two's Complement Numbers

Consider the following 8-bit binary number in Two's Complement:

11010011

What is its value in decimal?

1. Flip all bits
2. Add 1

Operations

So far, we have discussed:

- Addition: $x + y$
  - Can get multiplication
- Subtraction: $x - y$
  - Can get division, but more difficult
- Unary minus (negative): $-x$
  - Flip the bits and add 1

Operations (on Integers)

Bit vector: fixed-length sequence of bits (ex: bits in an integer)

- Manipulated by bitwise operations

Bitwise operations: operate over the bits in a bit vector

- Bitwise not: `~x` - flips all bits (unary)
- Bitwise and: `x & y` - set bit to 1 if *x*, *y* have 1 in same bit
- Bitwise or: `x | y` - set bit to 1 if either *x* or *y* have 1
- Bitwise xor: `x ^ y` - set bit to 1 if *x*, *y* bit differs

Example: Bitwise AND

```
  11001010
& 01111100
----------
```

Example: Bitwise OR

```
  11001010
| 01111100
```

Example: Bitwise XOR

```
  11001010
^ 01111100
----------
```

Your Turn!

What is: `0x1a ^ 0x72`

Operations (on Integers)

- Logical not: $!x$
  - $!0 = 1$ and $!x = 0, \forall x \neq 0$
  - Useful in C, no booleans
  - Some languages name this one differently
- Left shift: $x << y$ - move bits to the left
  - Effectively multiply by powers of 2
- Right shift: $x >> y$ - move bits to the right
  - Effectively divide by powers of 2
  - Signed (extend sign bit) vs unsigned (extend 0)