UNIVERSITY of VIRGINIA

# Mux, Binary Arithmetic

## CS 2130: Computer Systems and Organization 1

**Xinyao Yi** Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

## Announcement

- No class Monday
- We will have lab on Tuesday

## Multiplexer (mux)

x=a ? b : c

A multiplexer (mux) is commonly drawn as a trapezoid in circuit diagrams.

## Multiplexer (mux)

UNIVERSITY of VIRGINIA

## Multiplexer (mux)

## Multiplexer (mux)

## 2-bit Multiplexer (mux)

2-bit values instead of 1-bit values

## Multi-bit Values

So far, only talking about 2 things: 0 and 1

Next:

Numbers, strings, objects, ...

## Numbers

From our oldest cultures, how do we mark numbers?

- unary representation: make marks, one per ”thing”
  - Awkward for large numbers, ex: CS 2130?
  - Hard to tell how many marks there are
- Update: group them!
- Romans used new symbols:

## Numbers

Arabic numerals

- Positional numbering system

## Numbers

Arabic numerals

- Positional numbering system
- The 10 is significant:
    - 10 symbols, using 10 as base of exponent

## 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! $\pi$, 2130, 2, ...

## Base-8 Example

Try to turn $134_8$ into base-10:

## 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

***2 digits: 0, 1***

Try to turn $1100101_2$ into base-10

## Binary

Any downsides to binary?

Turn $2130_{10}$ into base-2:

*hint: find largest power of 2 and subtract*

## 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
- Effectively base-1000

## Long Numbers in Binary - Readability

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

100001010010

## Long Numbers in Binary - Readability

- 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?

100001010010

## Long Numbers in Binary - Readability

- 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 - Readability

- 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**

100001010010

## Long Numbers in Binary - Readability

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

100001010010

## Long Numbers in Binary - Readability

- 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

100001010010

## Hexadecimal

Need more than 10 digits. What next?

1110

## Hexadecimal Exercise

Consider the following hexadecimal number:

852dab1e

Is it even or odd?

## Using Different Bases in Code

| | Old Languages | New Languages |
|---|---|---|
| binary | | |
| octal | | |
| decimal | | |
| hexadecimal | | |

## Binary Addition

01101011 + 01100101

11101011 + 11100101

## Binary Subtraction

01111011 - 01100101

## 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

- Can we use the minus sign?

## Negative Integers

Representing negative integers

- Can we use the minus sign?
- In binary we only have 2 symbols, must do something else!

## Two’s Complement

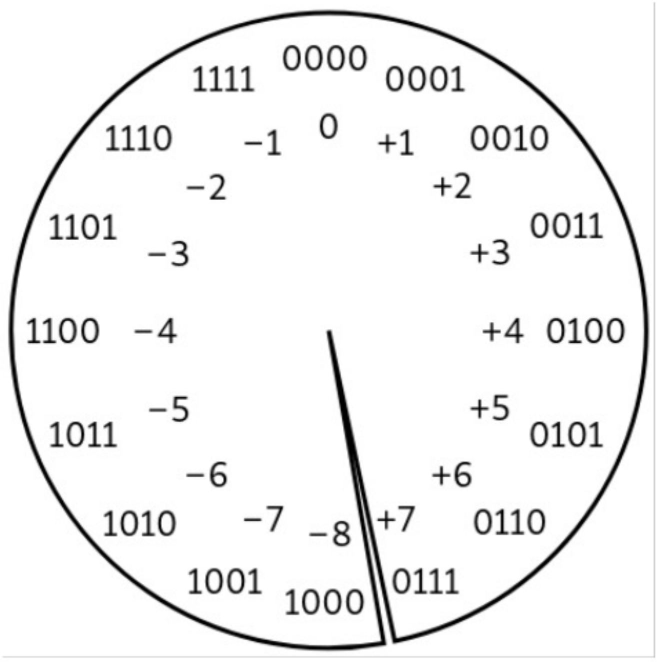
The scheme is called Two’s Complement

**Why do we need Two’s Complement?**

- We want the computer to represent both positive and negative numbers.
- And we want addition and subtraction to use the *same* hardware (just one adder), instead of building a separate “subtractor.”

**How does it work?**

- The **leftmost bit (MSB)** is treated as negative.
  - In normal binary: the leftmost bit is +128 (for 8-bit).
  - In two’s complement: the leftmost bit is −128.
- That’s why $10000000_2 = -128$ instead of +128.



## 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

## Values of Two’s Complement Numbers

**Why “invert the bits and add 1”?**

- Because in 8 bits, we have 256 total values (0–255).
- A negative number is stored as 256 − (its absolute value).
- The “invert + 1” trick is just a fast way to compute that.

## Values of Two’s Complement Numbers

Consider the following decimal number:

-117

What is its value in 8-bit binary binary?

## 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

Any Questions?