ECEG 431: Project 2

Stewart Thomas

August 28, 2025


ECEG 431
Your PC ran into a problem and needs to restart. We're
just collecting some error info, and then we'll restart for
you.
ECEG 431

NOTES from readings

- none completed yet!

From a mathematical perspective, the number `10` is utterly uninteresting, and, as far as computers go, is a complete nuisance.

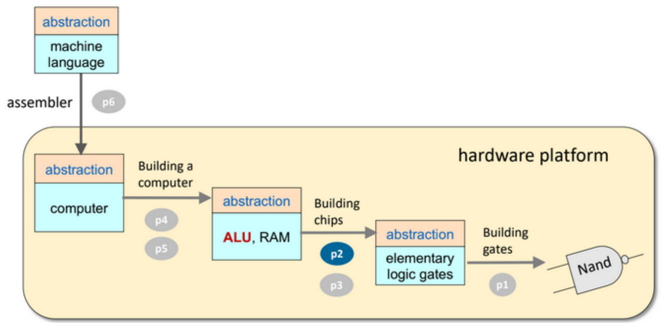
Overview

Project Overview

Last project we started working on our tools. Beginning with a simple NAND gate, we build up the set of tools that we will need for building the rest of the computer.

In this lesson, we will assemble these tools together in a way that we can start performing mathematical operations and doing Boolean Arithmetic.

Finding us on the map



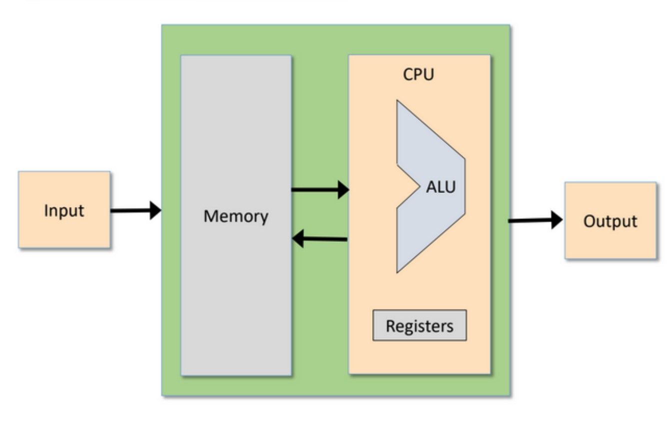
Project 2

Building chips that do arithmetic, ending up with an ALU

We just built 15 elementary logic gates and are now going to build chips that can do arithmetic and end up with an ALU.

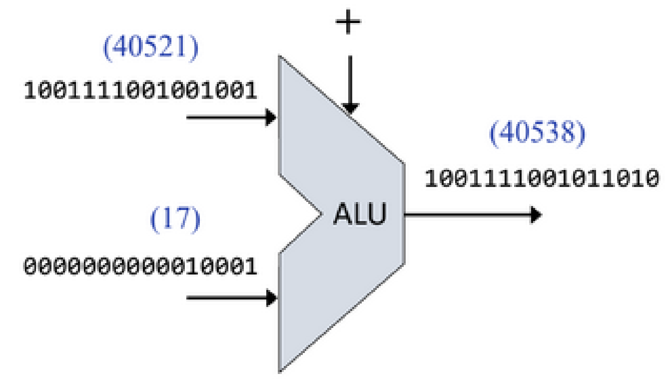
Zooming in a bit

At the moment, we are making a computer. . .



and we are going to focus on the ALU.

What is an ALU?



Arithmetic Logic Unit

The ALU is the heart of a CPU. It computes a function, be it arithmetic or logical, given a set of inputs and provides an output.

Examples of ALU functions are:

- addition
- subtraction
- increment
- decrement
- ANDing
- ORing
- negation
- logical shifting

Arithmetic Logic Unig

All of this must happen on binary signals and make use of the gates we have already created.

So therefore we need to look at:

- Number representation
- Binary numbers
- Boolean arithmetic
- Signed numbers

And then discuss the implementation

Theory

Theory

Numbers have been used for millenia.

- Simple hash marks to represent numbers
- Mayan base-20 system
- Roman numerals, etc.

However these older numeral systems do not scale well, are difficult for arithmetic, and blocked progress of algebra.

Place value

$$\begin{array}{cccc} 3 & 2 & 1 & 0 \\ 6 & 5 & 0 & 7 \end{array}$$

$$\sum_{0}^{n-1} d_i \cdot 10^i = 6 \cdot 10^3 + 5 \cdot 10^2 + 0 \cdot 10^1 + 7 \cdot 10^0 = 6507$$

Understanding place-value is key to understanding number representation.

Binary to decimal:

$$decimal\,(\overset{5\,4\,3\,2\,1\,0}{110101}_2) = 2^5 + 2^4 + 2^2 + 2^0 = 53_{10}$$

Decimal to binary:

$$binary\,(53_{10}) = 2^5 + 2^4 + 2^2 + 2^0 = \overset{5\,4\,3\,2\,1\,0}{110101}_2$$

Boolean arithmetic and ALU design

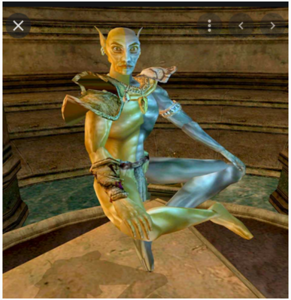
We will implement Addition using logic gates and then get . . .

Subtraction We get it for free!

Multiplication Well that's easy. . . its based on addition!

Division Yawn, again we've got it with what we already have.

Addition is everything.

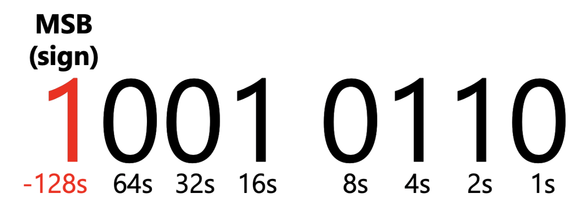


What a grand and intoxicating innocence.

Useful Resources

- link to: Nand2Tetris Project 2 slides [PDF]
- ~~link to: Binary Arithmetic review (slides, Univ. of Pittsburgh)~~ Whoops, this is deleted now. Check the *Project 02* folder from Moodle for the powerpoint.

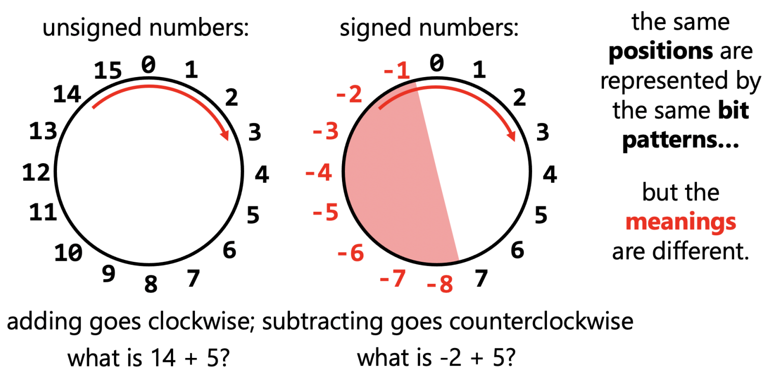
Signed Integers



The MSB is a *negative value*.

The 2s complement number line

In 2s complement, the numbers wrap around. Let's look at 4-bit numbers:



EVERY addition operation is a *modulo* operation.

???

How does a computer know if it is doing signed or unsigned addition?

???

How does a computer know if it is doing signed or unsigned addition?

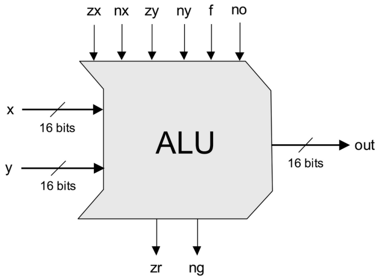
It Doesn't!

Implementation

The HACK ALU

The HACK's ALU is beautiful in its simplicity. It uses six control bits to control the ALU operations, and operates on two 16-bit inputs.

pre-setting the x input		pre-setting the y input		selecting between computing + or &	post-setting the output	Resulting ALU output
zx	**nx**	**zy**	**ny**	**f**	**no**	**out**
`if zx then x=0`	`if nx then x=!x`	`if zy then y=0`	`if ny then y=!y`	`if f then out=x+y else out=x&y`	`if no then out=!out`	`out(x,y)=`



These 6 control bits control:

- `ZERO`'ing and/or negating the inputs
- Selecting between addition or logical `AND`'ing
- Negation of the output

ALU Operation

pre-setting the x input		pre-setting the y input		selecting between computing + or &	post-setting the output	Resulting ALU output
zx	**nx**	**zy**	**ny**	**f**	**no**	**out**
if zx then x=0	if nx then x=!x	if zy then y=0	if ny then y=!y	if f then out=x+y else out=x&y	if no then out=!out	out(x,y)=
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

These operations perform 18 different function … and *(don't forget the two output flags!)*

Hints

- Start with the order the book goes through
  - Half-adder, Adder, etc.
- The HACK CPU is conveniently organized with the control bits controlling
  - “pre-processing” of inputs
  - Selection of a function (addition or ANDing)
  - “post-processing” of output

MORE Hints

We like to think in terms like:

```
if (mode == 1):
    return A + B
else:
    return A - B
```

If `mode == 1` is the second clause run?

MORE Hints

We like to think in terms like:

```
if (mode == 1):
    return A + B
else:
    return A - B
```

If `mode == 1` is the second clause run? NO!

In software, when you *make a decision* only one code path is run and the others never happen.

. . . but in Hardware-land

Our code becomes

```
sum  = A + B
diff = A - B
if (mode == 1):
    return sum
else:
    return diff
```

In hardware, when you *make a decision* you do all possible things and then pick the one you need and ignore the rest.

Other hints

Implementation note

If you need to set a pin `x` to `0` (or `1`) in HDL, use: `x = false` (or `x = true`)

Using multi-bit `truth` / `false` constants:

We can assign values to sub-buses

```
...
// Suppose that x, y, z are 8-bit bus-pins:
chipPart(..., x=true, y=false, z[0..2]=true, z[6..7]=true);
...
```

Slide outline