Project 2

In this section we are suppose to add/Substract two binary Values.

Here's a small intro to what Binary numbers are know to convert it to decimal & vice-versa.

Binary Numbers look like this 0101001101. This is because down to chip level, computers just understant on or off.

Convert Binary to Decimal

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	1	0	0	1	0	1	0

Step 1 for each box multiply the value with the power of 2 mentioned above

U'll get something lik this

X	512	0	128	64	0	0	8	0	2	0

Step 2 just add these Values

512 + 128 + 64 + 8 + 2 = 714

Now, lets go the other way around

Decimal to Binary

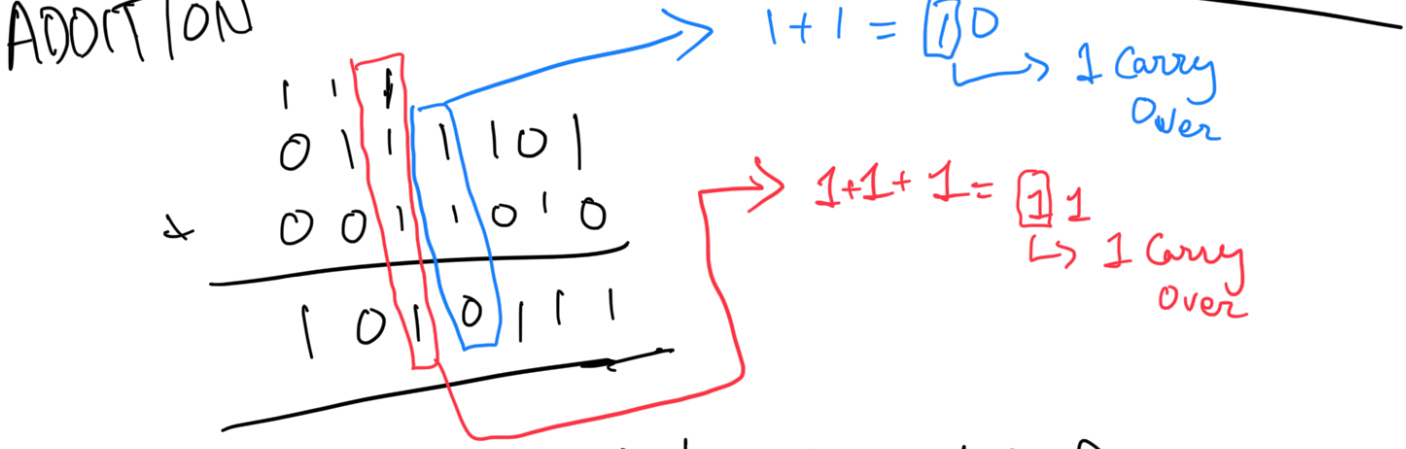
Step 1 Divide 714 by 2 & store the difference

Step 2 Arrange them from Bottom to top

-> 1 0 1 1 0 0 1 0 1

2 | 714 - 0
2 | 357 - 1
2 | 178 - 0
2 | 89 - 1
2 | 44 - 0
2 | 22 - 0
2 | 11 - 1
2 | 5 - 1
2 | 2 -> 0
1 -> 1

ADDITION



There are three chips to add Binary Numbers

-> Half Adder

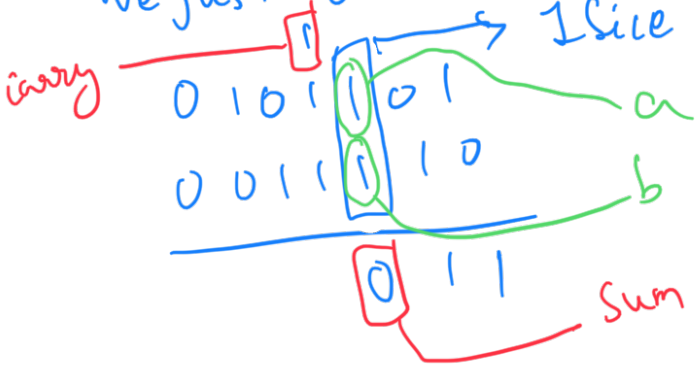
→ Full Adder

→ Adder

Lets Start with Half Adder

notice that we dont care about the carry from previous Slice

Half Adder has 2 inputs → a, b
& 2 outputs → Carry, Sum



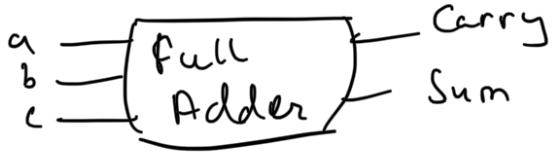
a	b	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Try this one, its pretty simple !

Carry is just Add (a=a, b=b, out = carry)
& Sum is just xor (a=a, b=b, out = Sum)

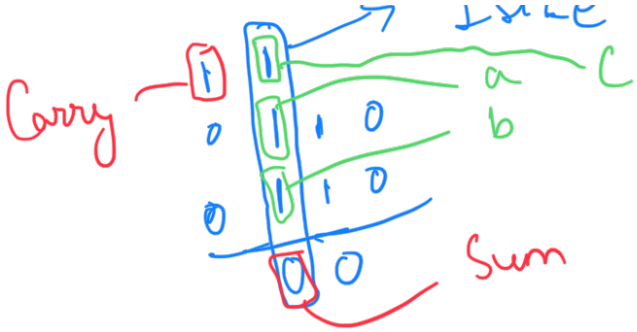
Full Adder

→ 3 inputs a, b, c
→ 2 out Carry Sum



In full adder we can add 3 bits, but we are still dealing with One Slice

a	b	c	Sum	Carry



0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

tempSum, tempCarry | Sum, Carry

0 0 | 1 0
0 0 | 0 1
0 0 | 0 1
1 1 | 1 1
0

There may exist a better way of doing this, below is the method I came up with.

Sum or tempCarry

0 0 → 0
1 0 → 1
0 0 → 1
1

first use HalfAdder(a=b, b=c, sum=tempSum, carry=tempCarry);

tempb bcoz we are not sure yet

then we use

Not(in=tempSum, out=tempSum2);

bcoz this is the other half of Sum & we are not sure which one is correct.

We finally use Mux(a=tempSum, b=tempSum2, sel=a, out=Sum);

we finally get the desired sum result

then we do the Or(a=tempSum, b=tempCarry, out=tempCarry2);

this is to get the desired Carry2

0
1

then we finally choose one of two "carry"

Mux16 a=temp1carry, b=temp1carry2, sel=a, out=carry);

Increment InC16

in → [InC16] → out=inc+1

Add16(a=in, b[0]=true, out=out);

[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]

↓

adds this to in

ALV

IN- x[16], y[16]

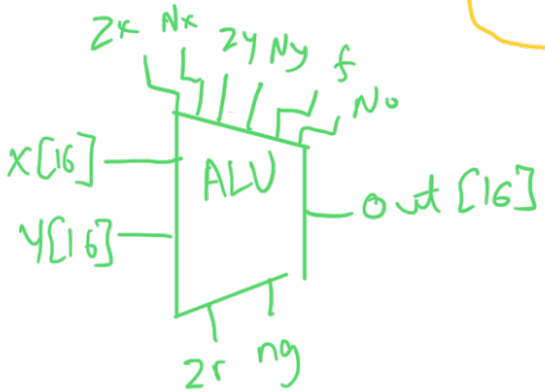
zx
nx
zy
ny
f
no

Out = out[16]

zr

ng

ALV is the Biggest chip yet but it is simpler than it looks just follow the below rules & you can easily make it



The Hack 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

↑→ Do the tasks in the given order, don't worry much about the truth table

↑↓ Change the x values to 0 if 2 x is 1. Use this new x moving forward

if Nx is 1 Change the new x to !x → Use this as new x

Do same with y.

→ Calculate both x+y & x&y now use Mux to decide which one we need. Store this out using temp variable

→ Calculate Not of temp output & use Mux to decide if we should put final output as Notout or out.

Tip: Create two outputs so that we can pass one to out pin & use other to calculate Zr & Ng

Zr is 1 if out == 0 else 0

Ng is 1 if out < 0 or 0

for Ng → Output is <0 when output[15] is 1. So we will use output[15] as sel for Mux and use a & b as true or false

for Zr → Output is 0 when none of its 16 bits contain 1. So we need to use Or to check that. If the output of Or is 1, number is not 0 else it is 0

Tip [for Zr I tried using 2 Or8Way & 1 Or to get the final Or output. But that way is complicated & you will create a lot of variables. So I created a Or16Way chip to simplify the process. Do the same]

Below is the code if You Couldn't Understand.

```
CHIP ALU {
    IN
        x[16], y[16],  // 16-bit inputs
        zx, // zero the x input?
        nx, // negate the x input?
        zy, // zero the y input?
        ny, // negate the y input?
        f,  // compute out = x + y (if 1) or x & y (if 0)
        no; // negate the out output?

    OUT
        out[16], // 16-bit output
        zr, // 1 if (out == 0), 0 otherwise
        ng; // 1 if (out < 0),  0 otherwise

    PARTS:
        Mux16(a=x,b=false,sel=zx,out=X);
        Not16(in=X,out=NotX);
        Mux16(a=X,b=NotX,sel=nx,out=zX);

        Mux16(a=y,b=false,sel=zy,out=Y);
        Not16(in=Y,out=NotY);
        Mux16(a=Y,b=NotY,sel=ny,out=zY);

        Add16(a=zX,b=zY,out=XplusY);
        And16(a=zX,b=zY,out=XAndY);
        Mux16(a=XAndY,b=XplusY,sel=f,out=F);

        Not16(in=F,out=NotF);
        Mux16(a=F,b=NotF,sel=no,out=out,out=outcopy,out[15]=outcopy2);

        Or16Way(in=outcopy,out=notzr);
        Not(in=notzr,out=zr);

        Mux(a=false,b=true,sel=outcopy2,out=ng);
```