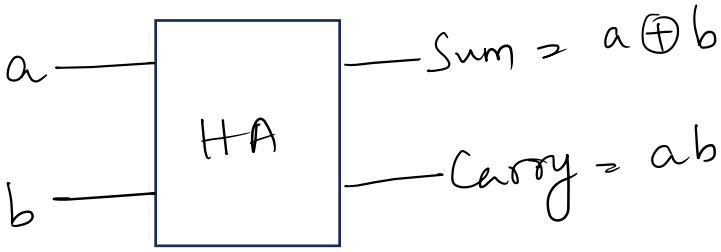
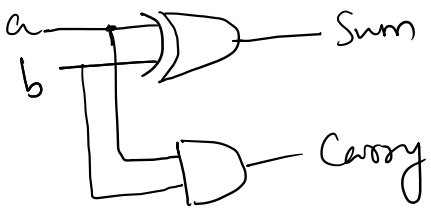
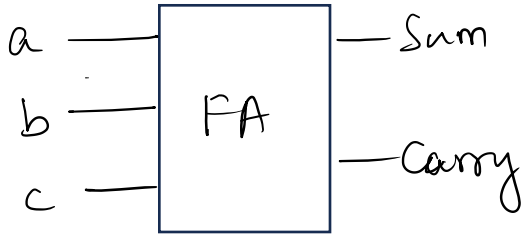
# Project 2

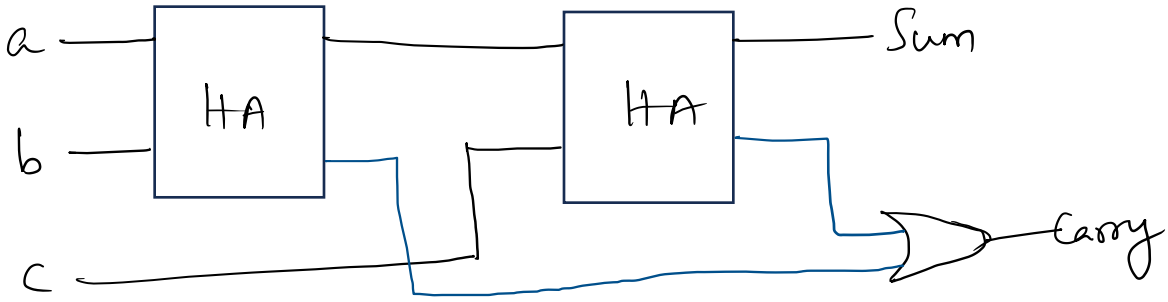
## Half Adder



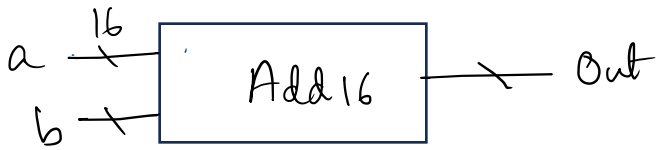



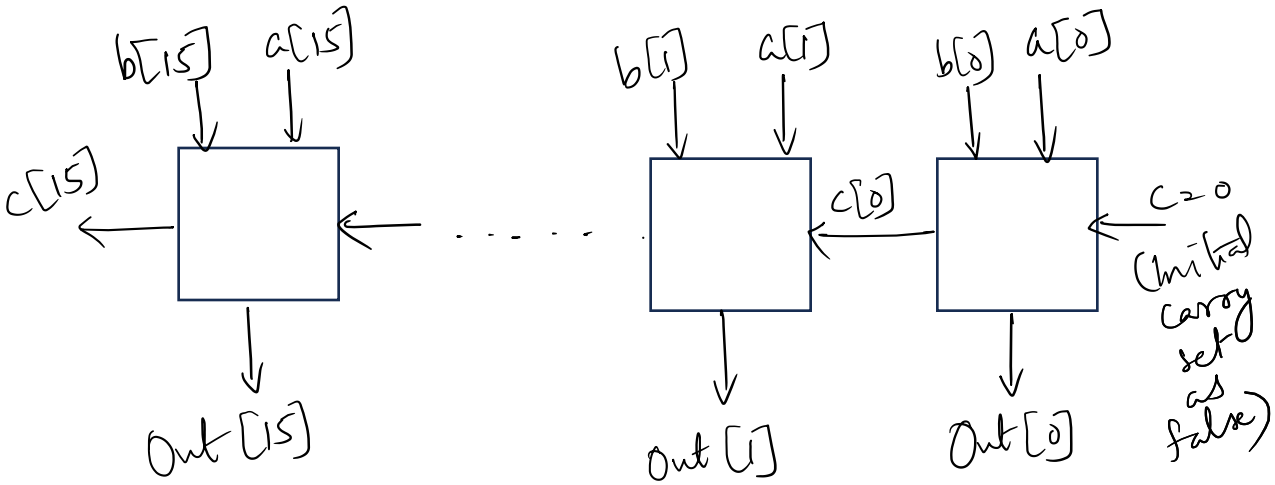

## Full Adder



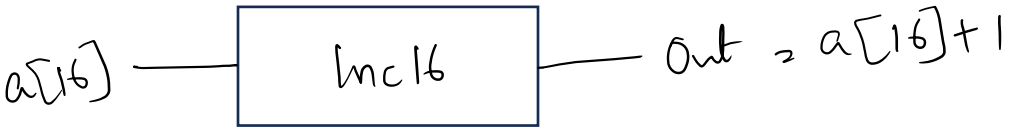



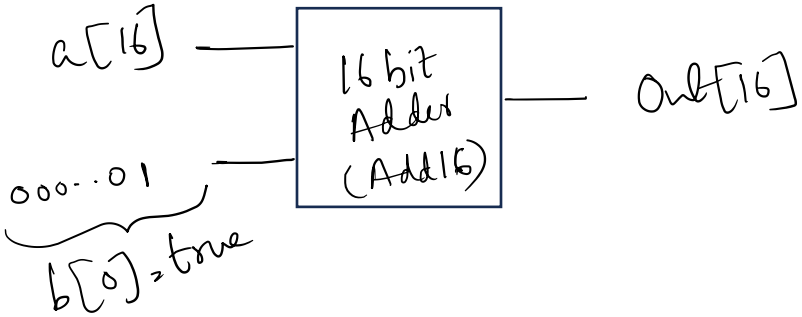
## Add16 (16 bit Adder)



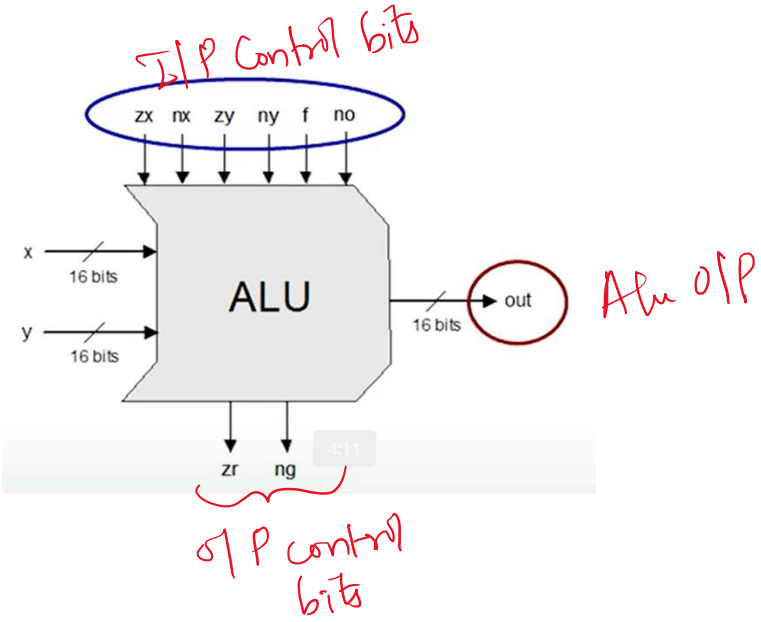




## Inc16



# ALU




| zx | nx | zy | ny | f | no | out |
|---|---|---|---|---|---|---|
| 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 |

Number of operations supported = 18

Shown in the table.

Multiplication and Division will be implemented using software later.

Note : All the operations are done in 2's complement form.

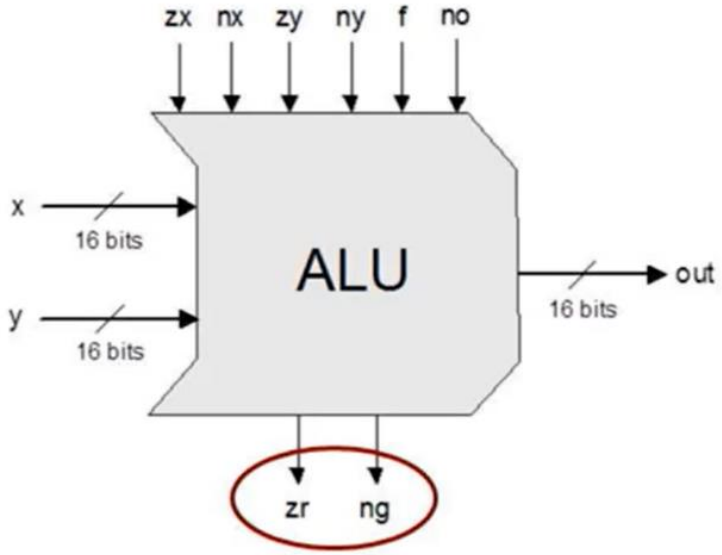
# The Hack ALU operation

| 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)= |


| 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 |

## The Hack ALU output control bits




```
if out == 0 then zr = 1, else zr = 0
if out < 0  then ng = 1, else ng = 0
```

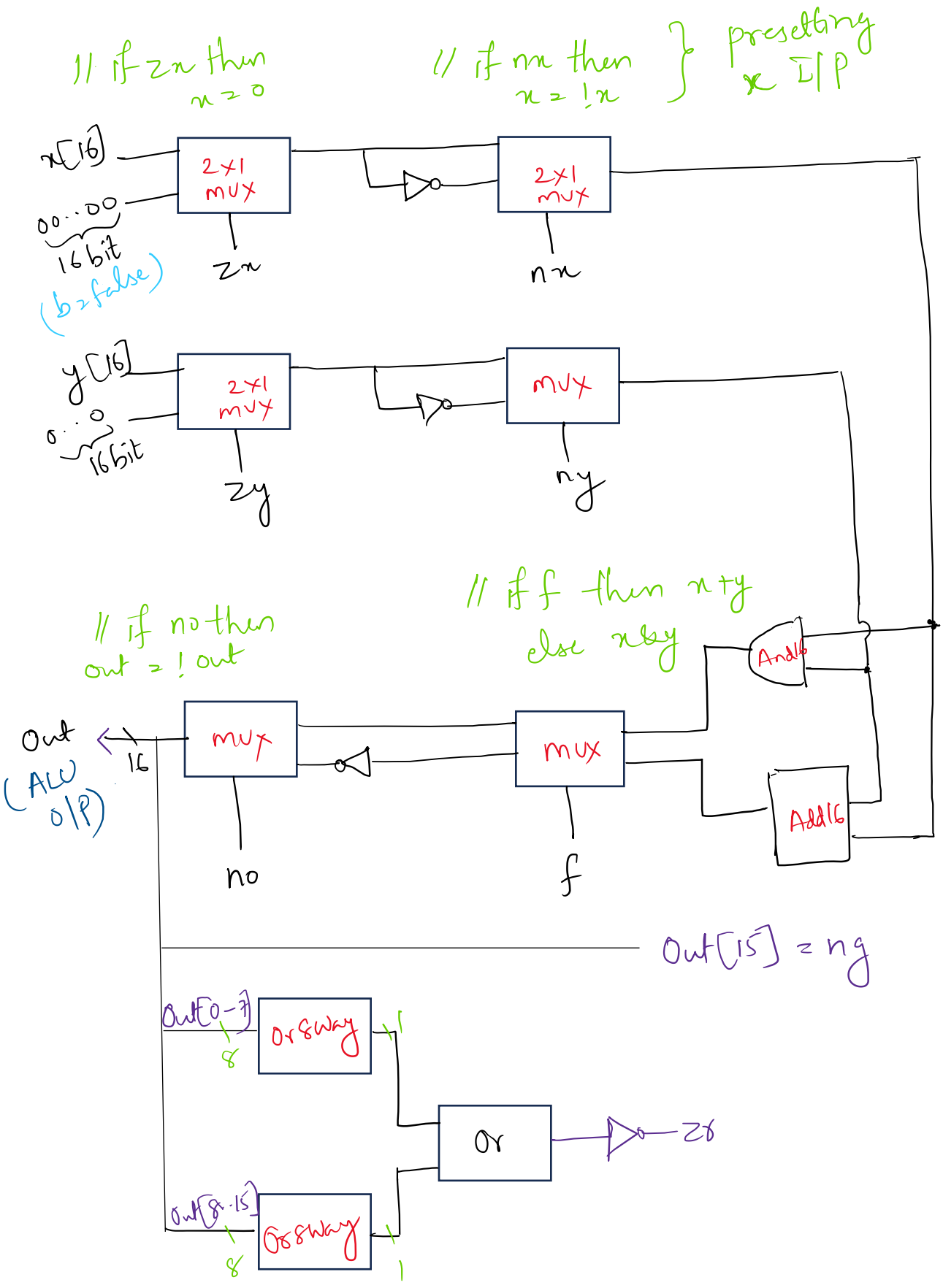
The logic can be easily deduced from the Hack Operation table.

Inputs : x,y,zx,nx,zy,ny,f,no

Outputs : Out, zr,ng

Out=0 >> zr = 1 : For this we “or” all the outputs to check if out=0. If out=0 we set zr=1.

For negative numbers MSB=1. So if MSB is one implies ng=1 else ng =0. So Out[MSB] = ng


// if zx then x = 0
// if nx then x = !x
} preselting x I/P
x[16]
00...00
16bit
(b=false)
2x1 mux
zx
nx
y[16]
0...0
16bit
zy
ny
// if f then x+y
else x&y
// if no then out = !out
Out (ALU O/P)
16
no
f
And16
Add16
Out[15] = ng
Out[0-7]
8
Or8way
Out[8-15]
Or
zr