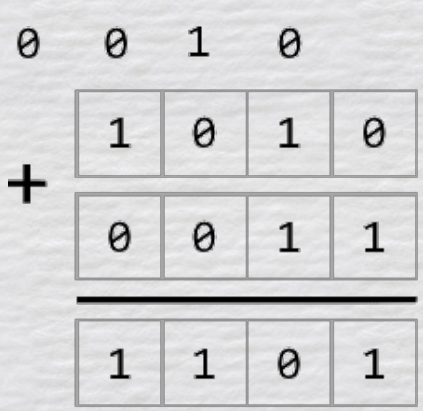
Computer Organization

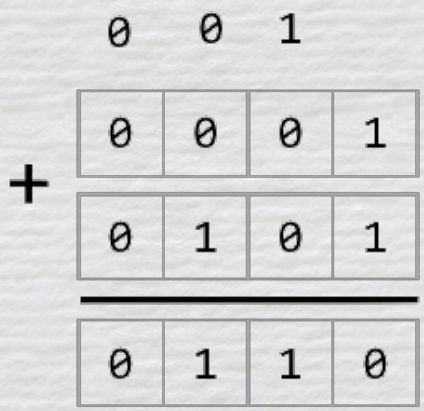
Arithmetic Gates

Adding Numbers

- Adding numbers starts from a basic concept
  - Add 2 inputs, get the sum and carry
  - Move to the next to add the next 2 inputs and the previous carry



Binary addition

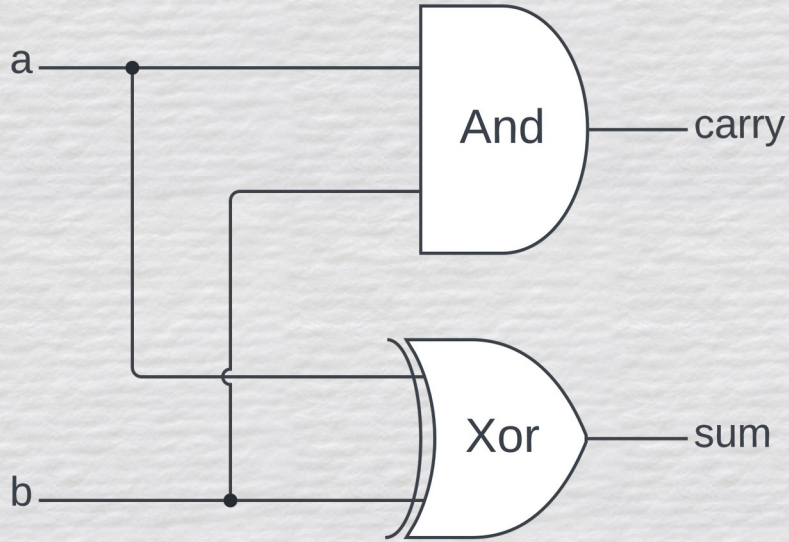


Another example

Overflow

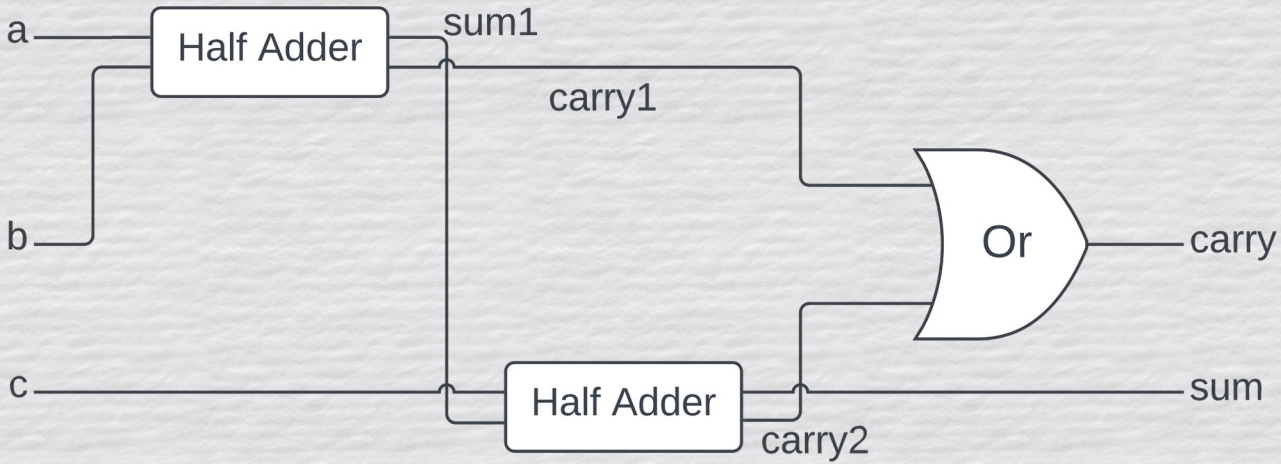
Half Adder

- Half Adders take 2 inputs and output a sum and a carry.
  - Used to kick start an addition operation



Full Adder

- Full Adders are used as the continuous elements of addition operation since they account for carry bits
  - Takes in 2 inputs and a carry input to output a sum and a carry



Add Bit Sequences (Add 16)

- Using half adders and full adders, it's very simple to compute the addition of 2 bit sequences
  - Pass the Least Significant Bit ([0] for the .hdl language) into a Half Adder
  - Pass the rest of the 15 bits and the previous carry bits into Full Adders until you've accounted for all bits

```
6   /**
7    * Adds two 16-bit values.
8    * The most significant carry bit is ignored.
9    */
10
11  CHIP Add16 {
12      IN a[16], b[16];
13      OUT out[16];
14
15      PARTS:
16     // Put you code here:
17     HalfAdder(a=a[0],b=b[0],sum=out[0],carry=carry1);
18     FullAdder(a=a[1],b=b[1],c=carry1,sum=out[1],carry=carry2);
19
20      ...
21
22     FullAdder(a=a[15],b=b[15],c=carry15,sum=out[15],carry=carry16);
23  }
```

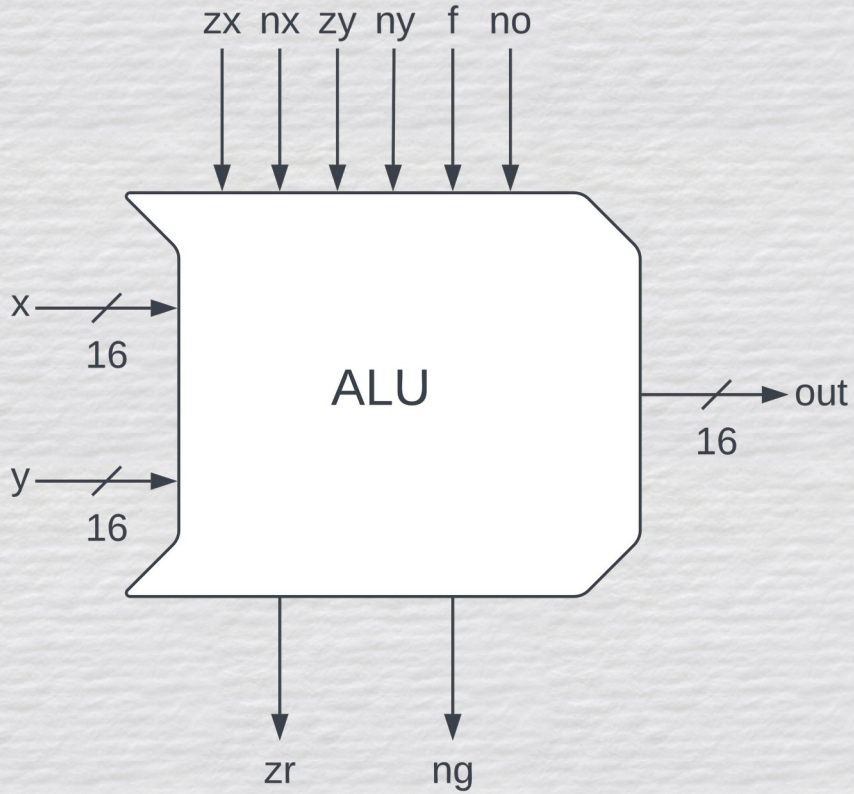
Incrementing a Bit Sequence (Inc 16)

- Incrementing a bit sequence is incredibly simple
  - Pass the entire first input into an Add-N (16 for our use cases)
  - Then, pass **true** in for the LSB of the next bit sequence and **false** for all other bits
  - This creates **0000000000000001**

```
6   /**
7    * 16-bit incrementer:
8    * out = in + 1 (arithmetic addition)
9    */
10
11  CHIP Inc16 {
12      IN in[16];
13      OUT out[16];
14
15      PARTS:
16     // Put you code here:
17     Add16(a=in,b[0]=true,b[1..15]=false,out=out);
18  }
```

The ALU

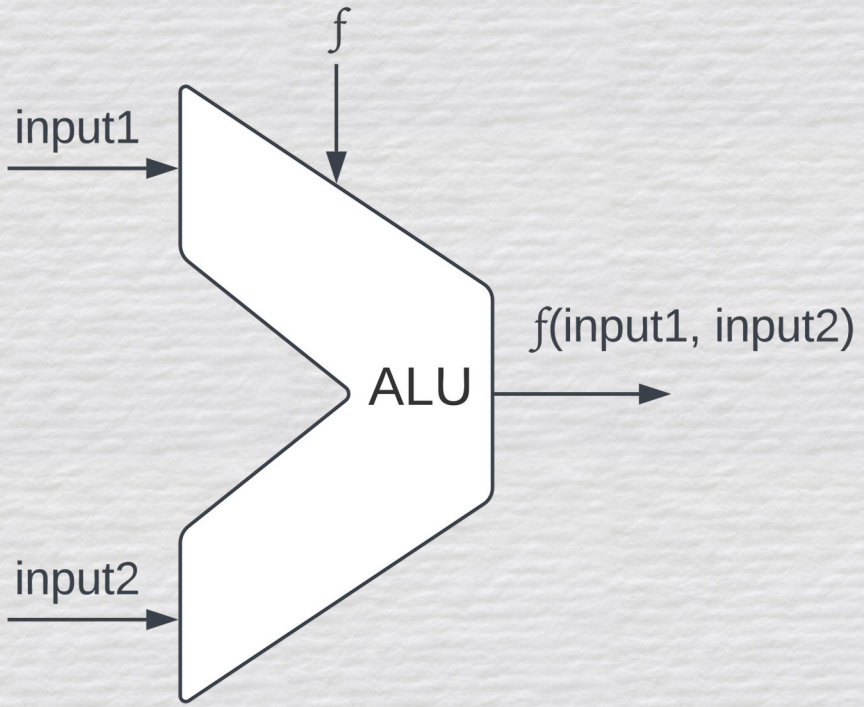
- What even is this?



The ALU – cont.

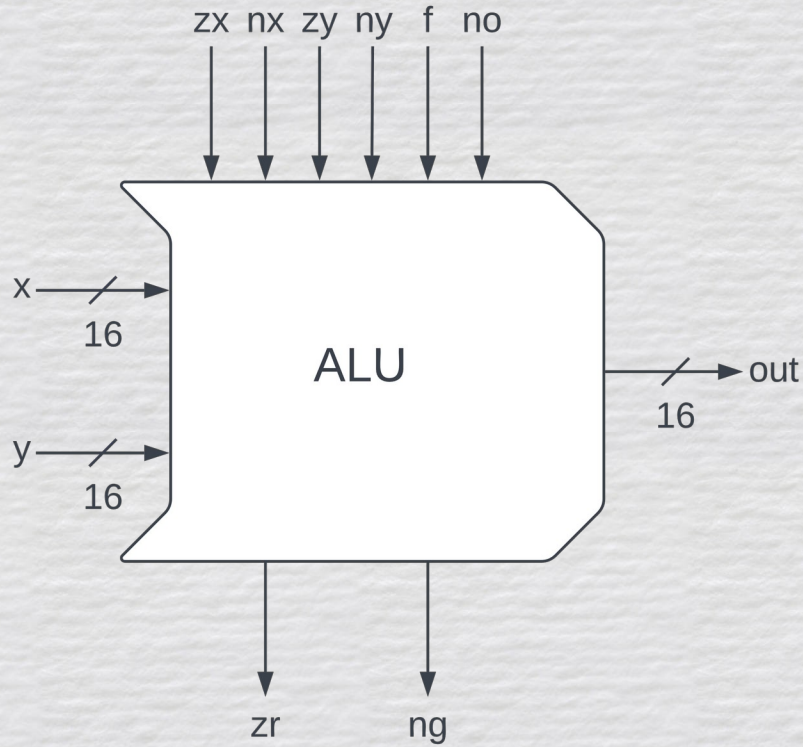
- The ALU computes a given function on its two given data inputs, and outputs the result

- f: one out of a family of pre-defined arithmetic functions (*add, subtract, multiply, ...*) and logical functions (*And, Or, Xor, …*)



The Hack ALU

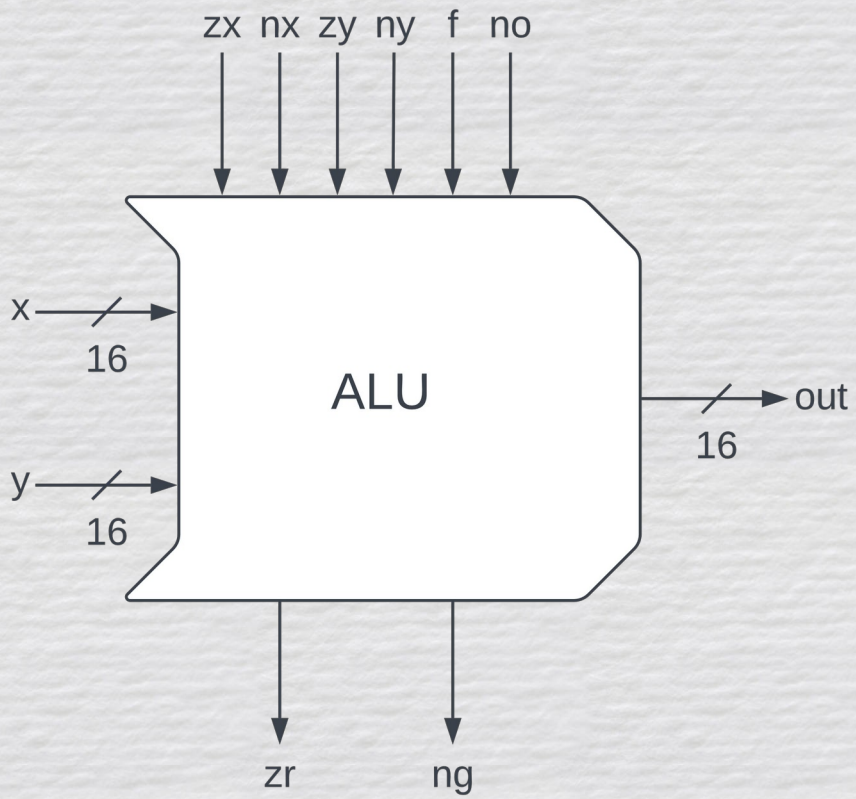
- Inputs
  - Two 16-bit, two's complement values
  - Six 1-bit control values
- Outputs
  - One 16-bit, two's complement value
  - Two 1-bit values



out
0
1
-1
x
y
!x
!y
-x
-y
x+1
y+1
x-1
y-1
x+y
x-y
y-x
x&y
x|y

The Hack ALU – cont.

- The control bits determine the ALU's function



control bits

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

The Control Bits

- We can treat the 6 control bits as if statements that string together to form a single function

- But how do we do this in hardware?

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

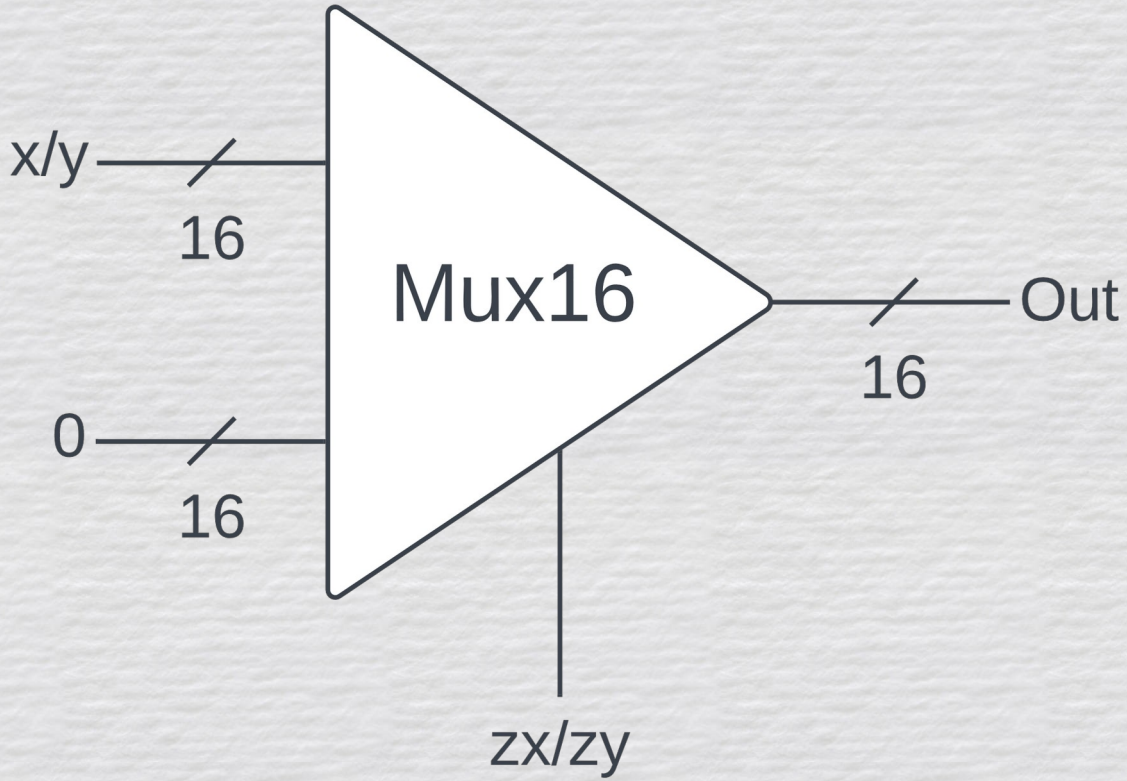
The Control Bits – cont.

- We can treat the 6 control bits as if statements that string together to form a single function

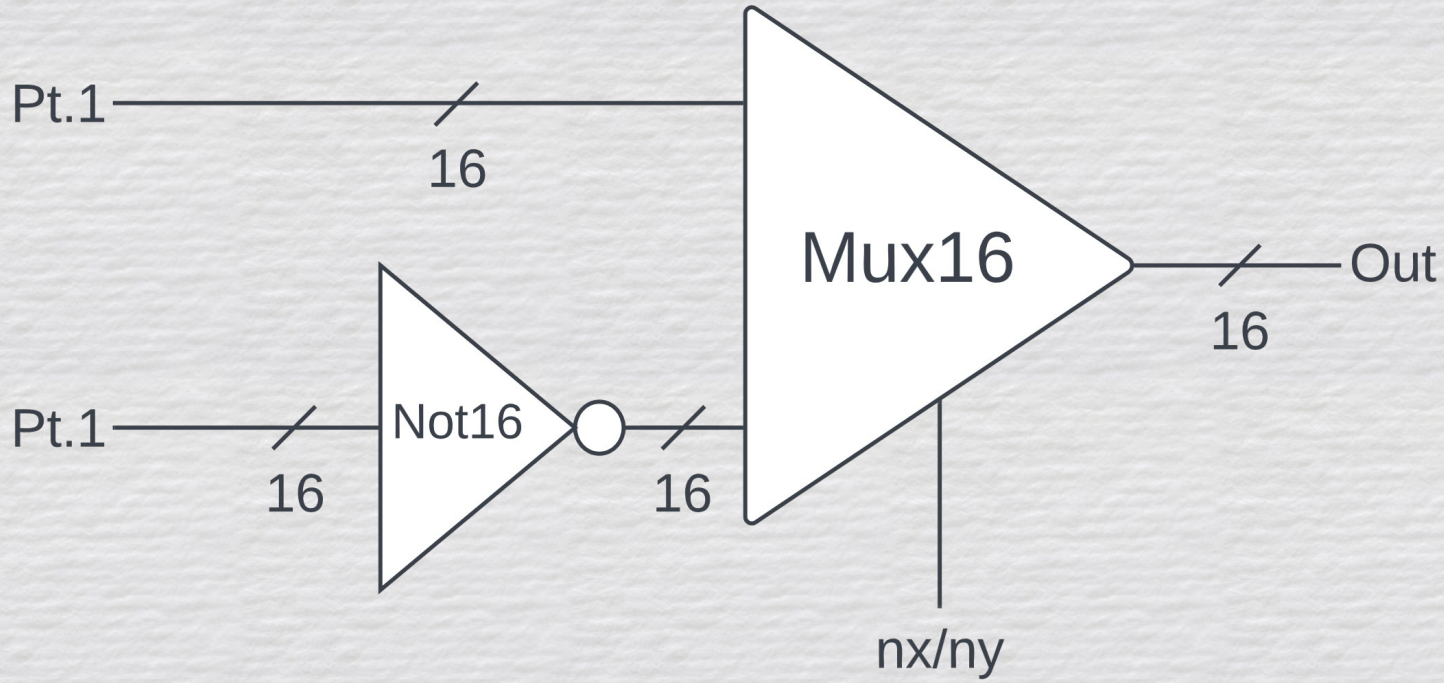
- But how do we do this in hardware? **Muxes**

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

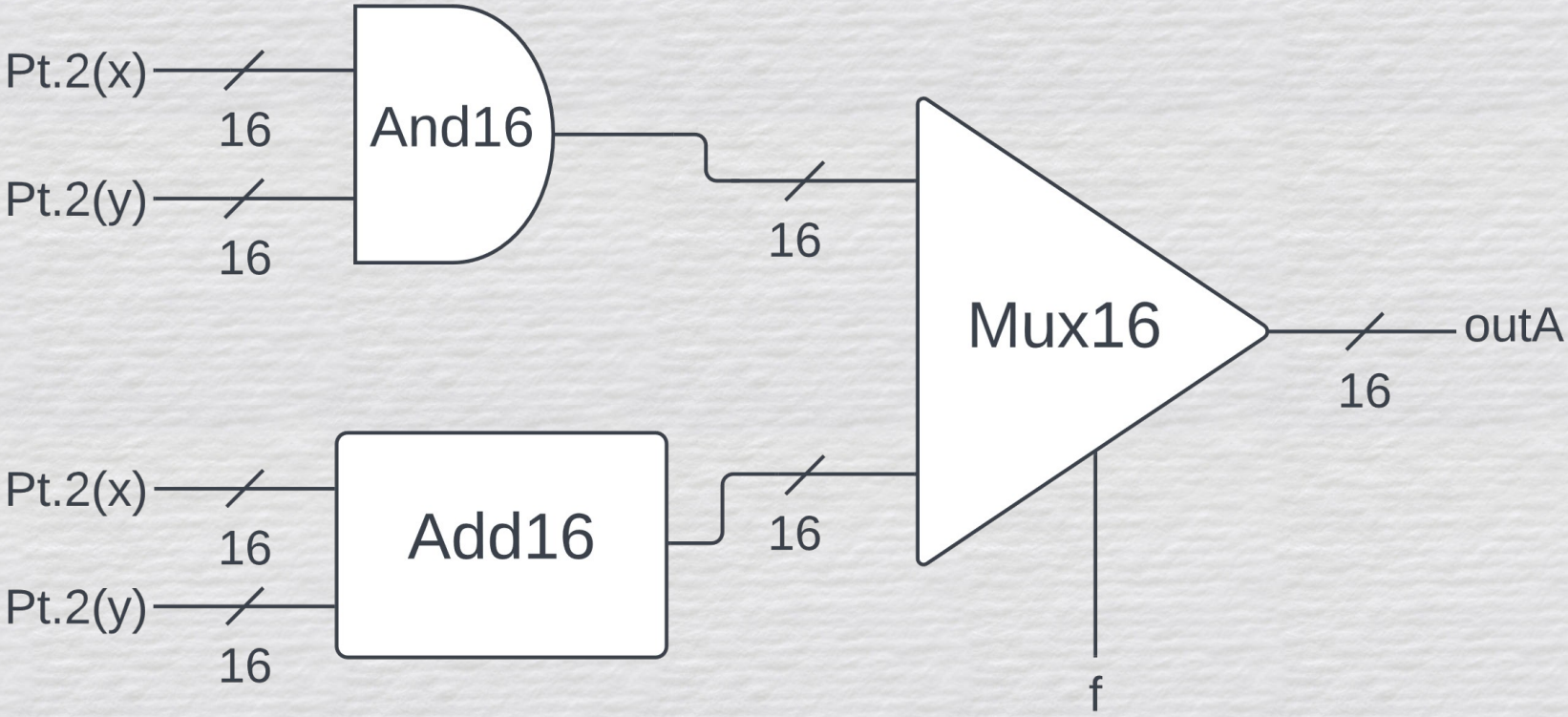
The Control Bits: A Thrilling Series – 1



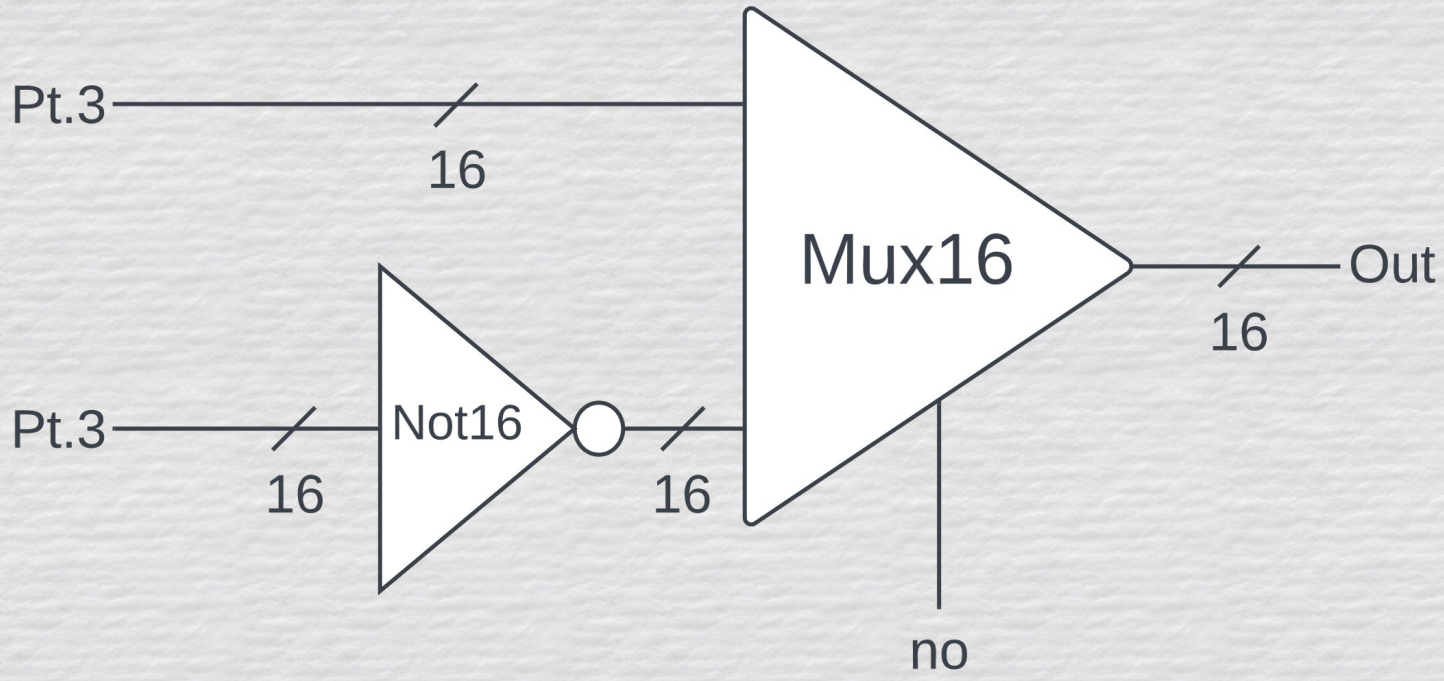
The Control Bits: A Thrilling Series – 2



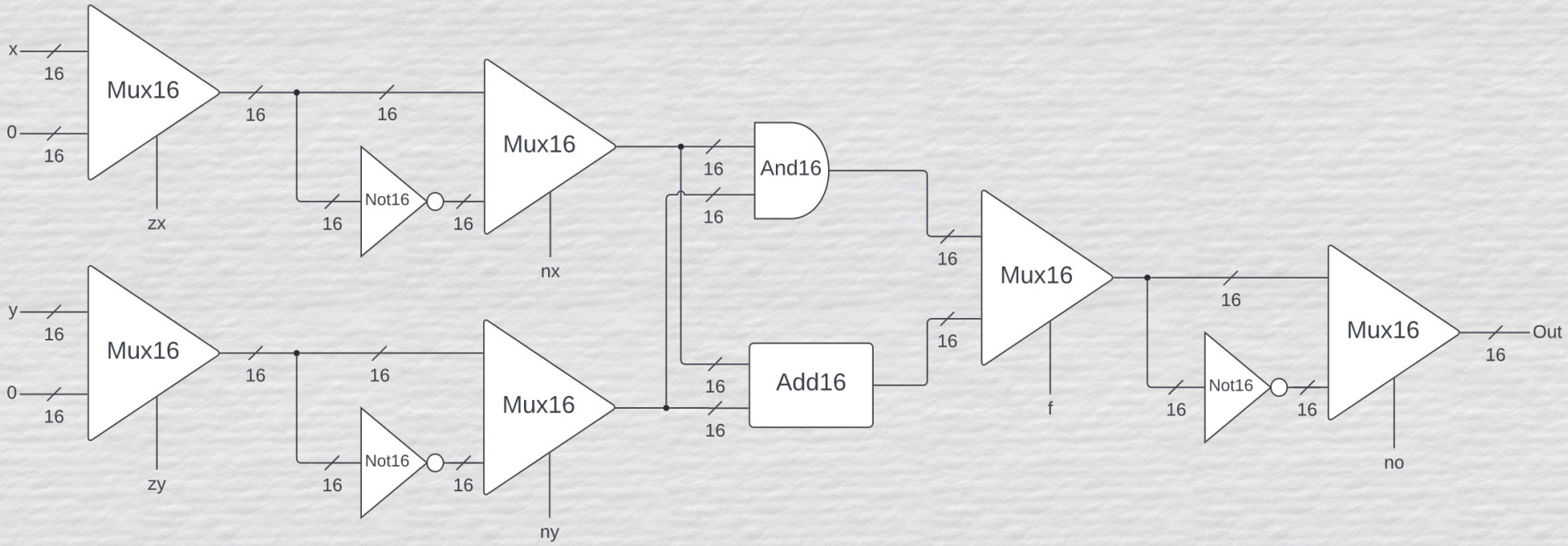
The Control Bits: A Thrilling Series – 3



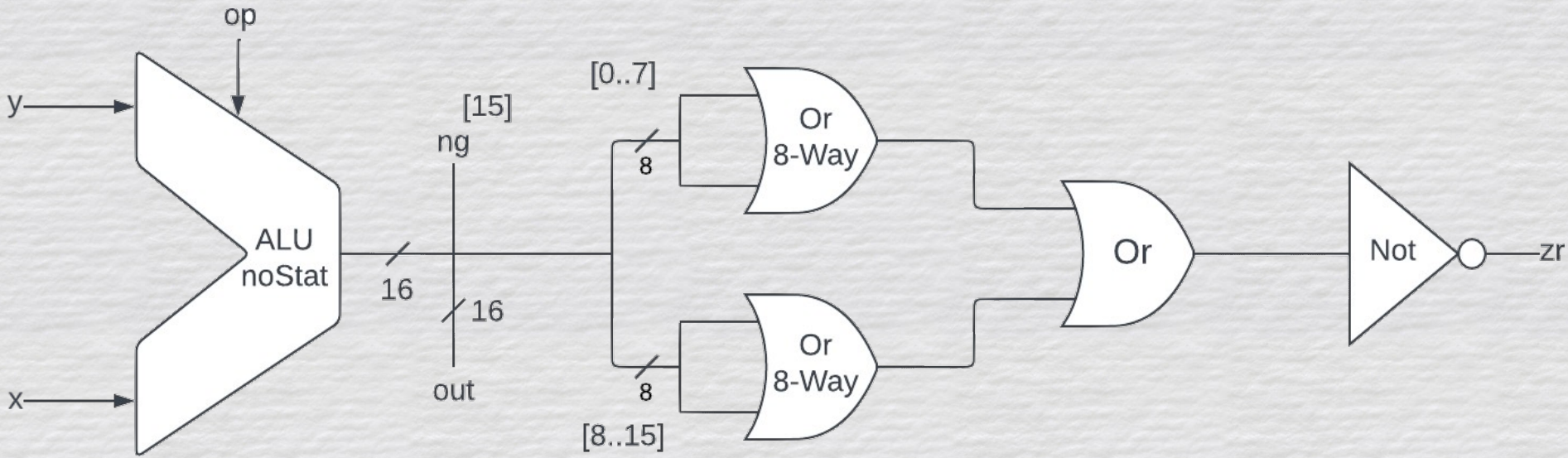
The Control Bits: A Thrilling Series – 4



The ALU: NoStat



The ALU


op
y
x
ALU
noStat
16
[15]
ng
16
out
[0..7]
8
Or
8-Way
8
[8..15]
Or
8-Way
Or
Not
zr