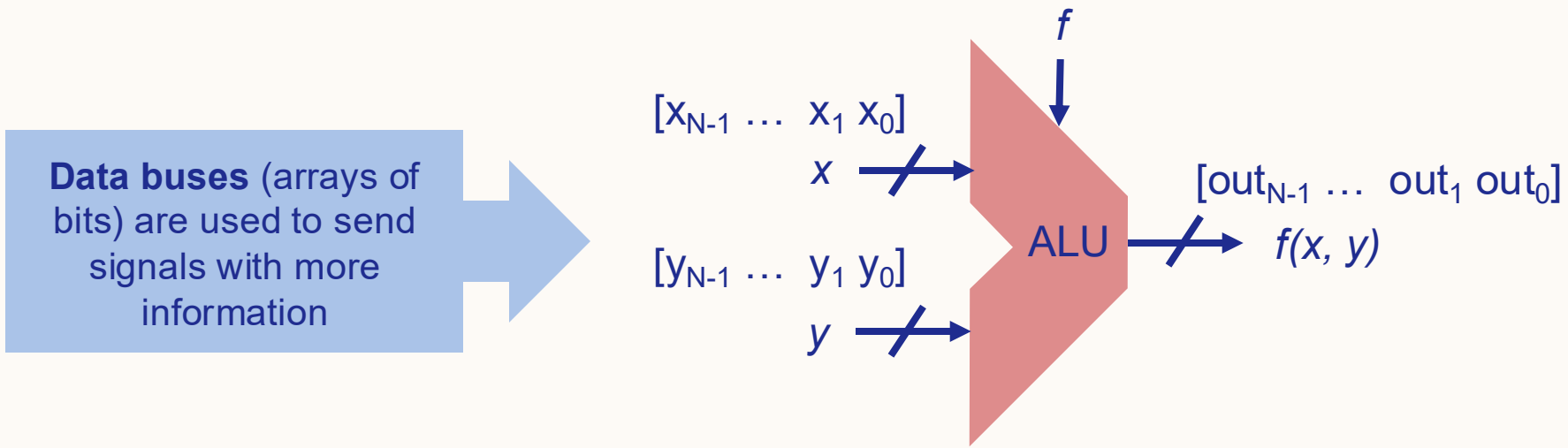
ARITHMETIC LOGIC UNIT

Kira Clements, University of Bristol

ALU

A central part of the CPU; the ALU takes in 2 N-bit inputs, *x* and *y*, performs a function, *f*, between them, and outputs the N-bit result, *f(x, y)*. The operation is one of a set of predefined operations that may be logical or arithmetic.

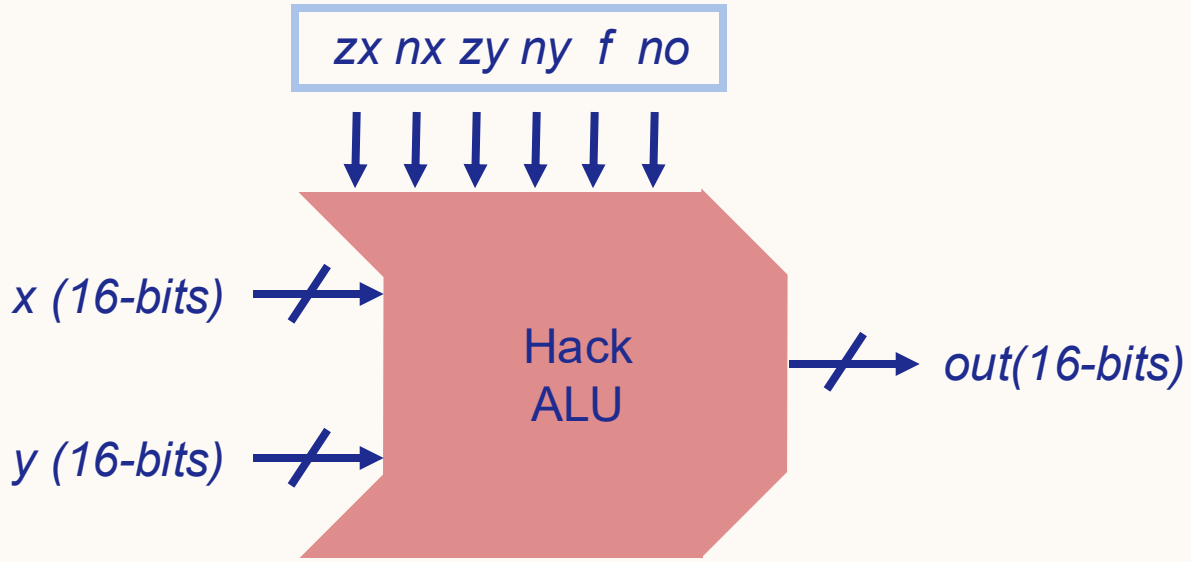


The ALU operates logical functions bitwise e.g. $out_0 = x_0$ AND y_0, $out_1 = x_1$ AND y_1, while arithmetic functions operate using a carry between each bit.

CONTROL BITS

How does an ALU decide which function is calculated?

An ALU uses control bits, also known as **select bits**, to make decisions within the unit. Looking at the Hack ALU, we can see that it uses 6 control bits that each decide between 2 options:



zx: whether to ZERO the x signal

zy: whether to ZERO the y signal

nx: whether to NOT the x signal

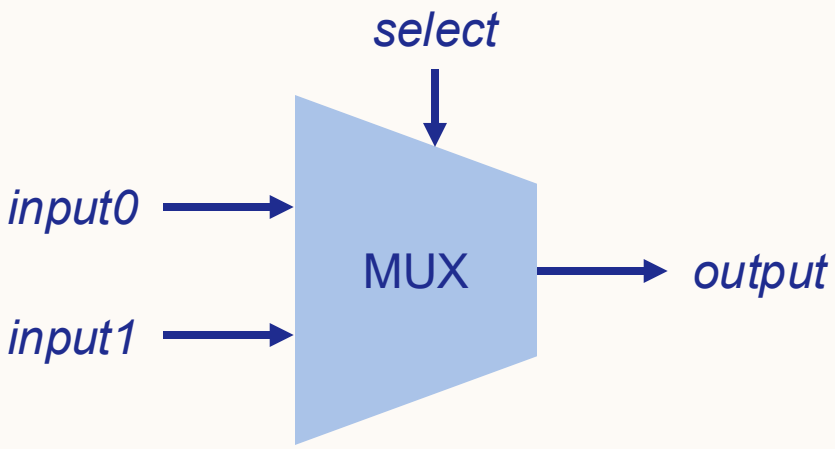
ny: whether to NOT the y signal

f: whether to ADD or AND the input signals

no: whether to NOT the out signal

MULTIPLEXERS

A N-to-1 multiplexer takes in N inputs and uses a control signal to select which of these to propagate to the output. We need **$\log_2$(N) select bits** to choose between N inputs, as this gives us X where $2^X = N$.

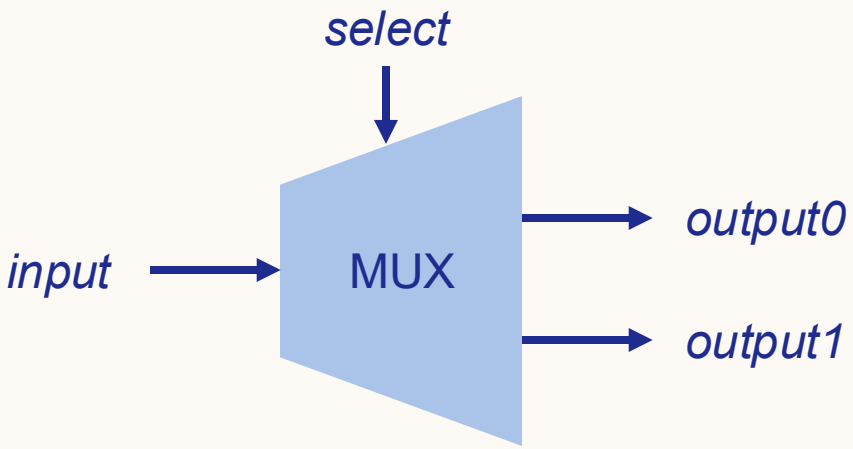



```
if(select == 0){
    output = input0;
} else if(select == 1){
    output = input1;
}
```

A 2-to-1 multiplexer, like that pictured above, has a single select bit. If this bit is set to 0 then the multiplexer will output the value of input0. Else, if this select bit is set to 1, then the multiplexer will output the value of input1.

DEMULTIPLEXERS

On the other hand, we have 1-to-N demultiplexers, which take in 1 input and propagates this signal onto one of N output wires, determined by the control signal. Any output signal not selected will output 0. Again, we need $\log_2(N)$ select bits to choose between N outputs, as this gives us X where $2^X = N$.



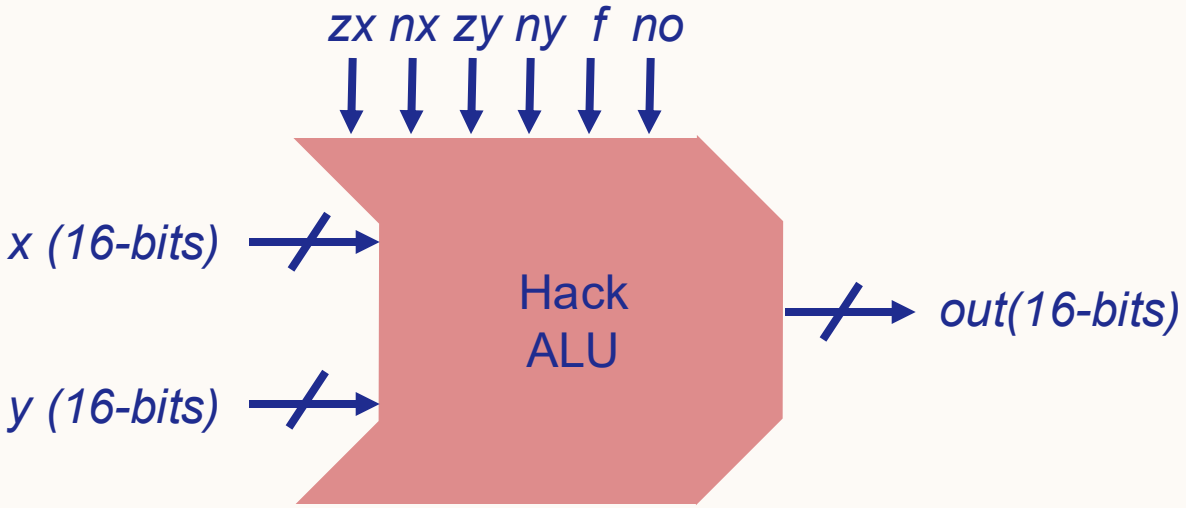

```
if(select == 0){
    output0 = input;
} else if(select == 1){
    output1 = input;
}
```

A 1-to-2 demultiplexer, like that pictured above, has a single select bit. If this bit is set to 0 then the demultiplexer will propagate the value of input onto the output0 signal. Else, if this select bit is set to 1, then the demultiplexer will propagate the value of input into the output1 signal.

HACK ALU

The Hack ALU takes in two inputs with 2's complement values using 16-bit data buses and outputs a resulting 2's complement value using a 16-bit data bus.

Setting the control bits to one of the combinations defined in the below truth table will result in the ALU calculating the function listed in the out column. This is a reduced version of the complete ALU truth table, which list further functions possible given different combinations of control bits.



zx	nx	zy	ny	f	no	out
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

HACK ALU EXAMPLE 1

zx	nx	zy	ny	f	no	out
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

For the control bit combination highlighted on the left, as *zx* and *zy* are both 1, x and y are both set to zero. If we considered these inputs to be 4-bit, this would mean both x and y are set to 0000_2.

As *nx* and *ny* are 1, both the x and y signals are then negated, making each of these input signals set to 1111_2.

As the *f* select bit is 1, these input signals are then added together i.e. $1111_2 + 1111_2 = 1110_2$.

Finally, as *no* is 1, the output is negated, which would make it $0001_2 = 1_{10}$.

HACK ALU EXAMPLE 2

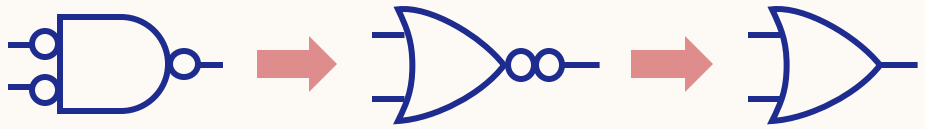
For the control bit combination highlighted on the right, as *nx* and *ny* are 1, both the x and y signals are negated, giving us inputs of ¬x and ¬y.

As the *f* select bit is 0, the function selected is AND, which gives us an output of (¬x ∧ ¬y).

Finally, as *no* is 1, the output is negated, which gives us a final output of ¬(¬x ∧ ¬y).

But the output listed on our truth table is x|y?

Using De Morgan's laws, these are the same thing!



zx	nx	zy	ny	f	no	out
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y