Hack's ALU Truth Table:

zx	nx	zy	ny	f	no	out
1	0	1	0	1	0	0

$\Rightarrow x=0;\ y=0;\ f=x+y=0;\ out=0$ ✓

zx	nx	zy	ny	f	no	out
1	1	1	1	1	1	1

$x=-1;\ y=-1;\ f=x+y=\underline{-1+(-1)}$

$f=-2\ (1111111111111110)$ ✓

$out=!(-2)=1\ (0000000000000001)$

General properties: (2's complement)

$-x=\sim x+1$

$-x=\sim(x-1)$

zx	nx	zy	ny	f	no	out
1	1	1	0	1	0	-1

$x=-1;\ y=0$ $\quad f=-1$ $\quad out=-1$ ✓

zx	nx	zy	ny	f	no	out
0	0	1	1	0	0	x

x = x

y = -1

f = x & y =

x[15] x[14] ... x[0]
1 1 ... 1
x AND y

= x ✓

zx	nx	zy	ny	f	no	out
1	1	0	0	0	0	y

x = -1 (1111111111111111)

y = y

f = y[15] y[14] ... y[0]
1 1 ... 1

out = y ✓

zx	nx	zy	ny	f	no	out
0	0	1	1	0	1	!x

x = x

y = -1

since x & y = x → out = !x ✓
(1111...111)

f = x & y
out = !(x & y)

→ Same logic for next row: !y ✓

zx	nx	zy	ny	f	no	out
0	0	1	1	1	1	-x

$x = x$ $\quad y = -1$ $\quad f = x+y$ $\quad out = !(x+y)$

$out = !(x-1)$ by general rule

$\Rightarrow !(x-1) = -x$ ✓

→ Same logic for next row : $-y$ ✓

zx	nx	zy	ny	f	no	out
0	1	1	1	1	1	x+1

$x = \sim x$ $\quad y = -1$ $\quad f = x+y$

$= \sim x - 1$

$\Rightarrow out = !(\sim x - 1) \Rightarrow$ apply general 2's complement properties

$= !(-x-1-1) =$

$!(-(x+1)-1)$

$\Rightarrow a = x+1 \Rightarrow !(-a-1) = a$

$out = x+1$ ✓

→ Same logic for $y+1$ ✓

zx	nx	zy	ny	f	no	out
0	0	1	1	1	0	x-1

x=x y=-1 f = x+y = x-1 ✓

→ Same logic for y-1 ✓

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

x=x y=y f = x+y ✓

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

x=nx y=y f = nx+y

out = n(nx+y)

→ (nx = -x-1) (by general properties)

out = n(-x-1+y) = (-(x-y)-1)

if a=x-y

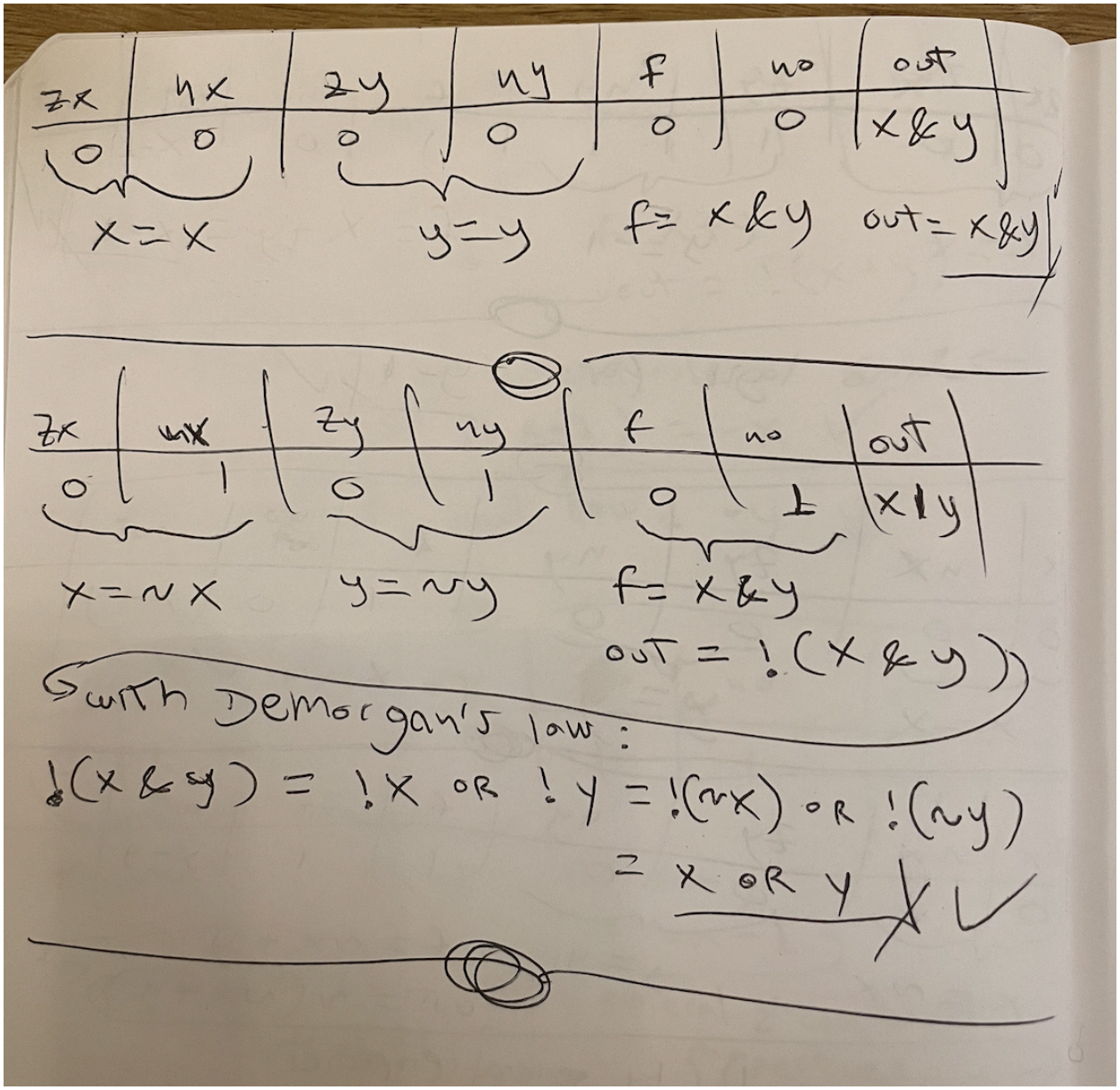
a = n(-a-1)

a = n(-(x-1)-1) = x-y ✓

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

x=x y=ny f = x+ny out = n(x+ny)

ny = -y-1 ⇒ n(x+(-y)-1) = n(-(y-x)-1) = y-x ✓



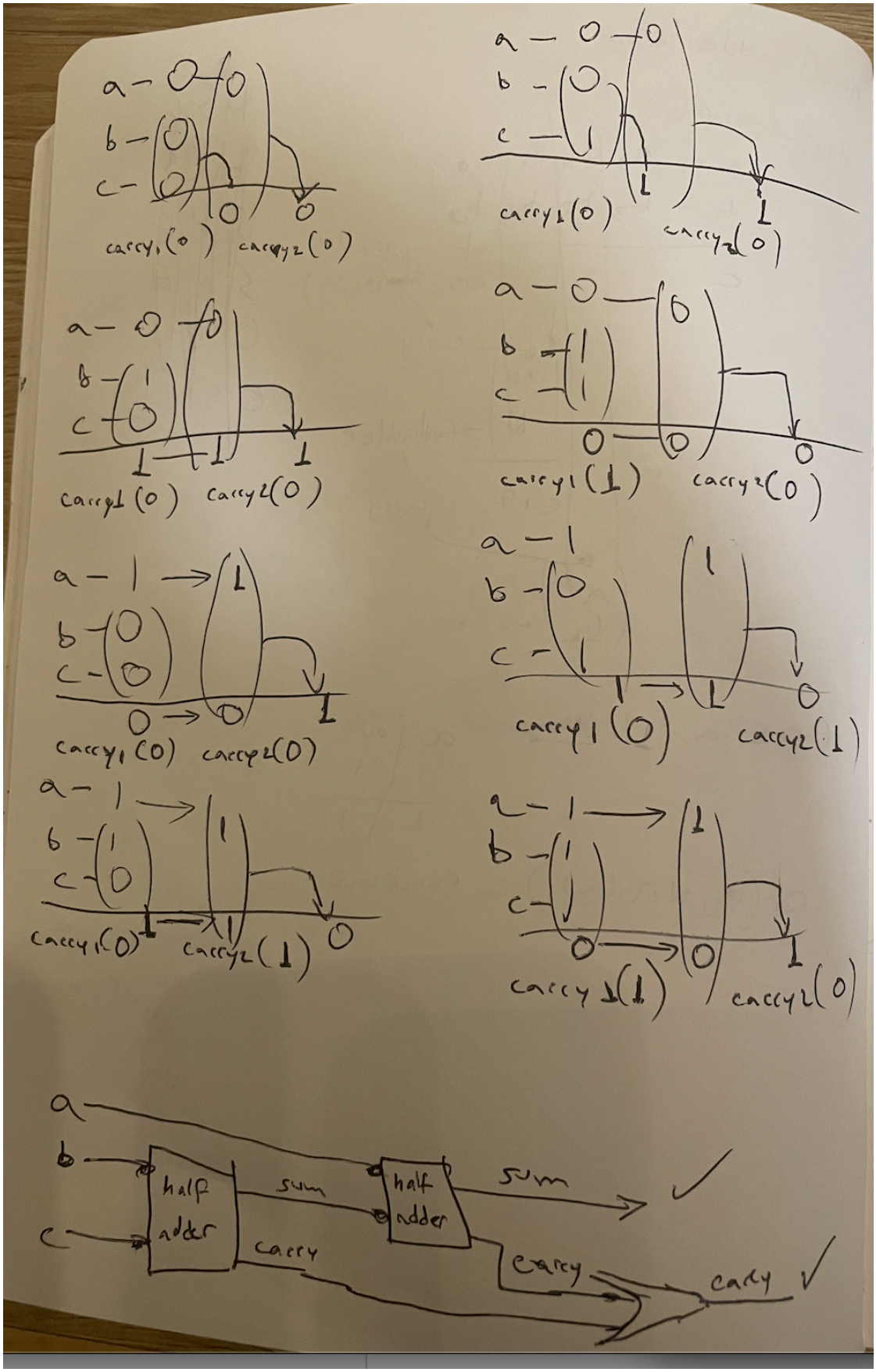
The truth table presented in p. 39, presents the values of the 6 control bits used for the ALU's functions. It's not always obvious the output of each row, so the pages above aim to describe how each of these outputs is achieved.

IMPLEMENTATION

Half-adder:
It can be easily created by looking at its truth table, resulting in two simple gates for carry and sum outputs:

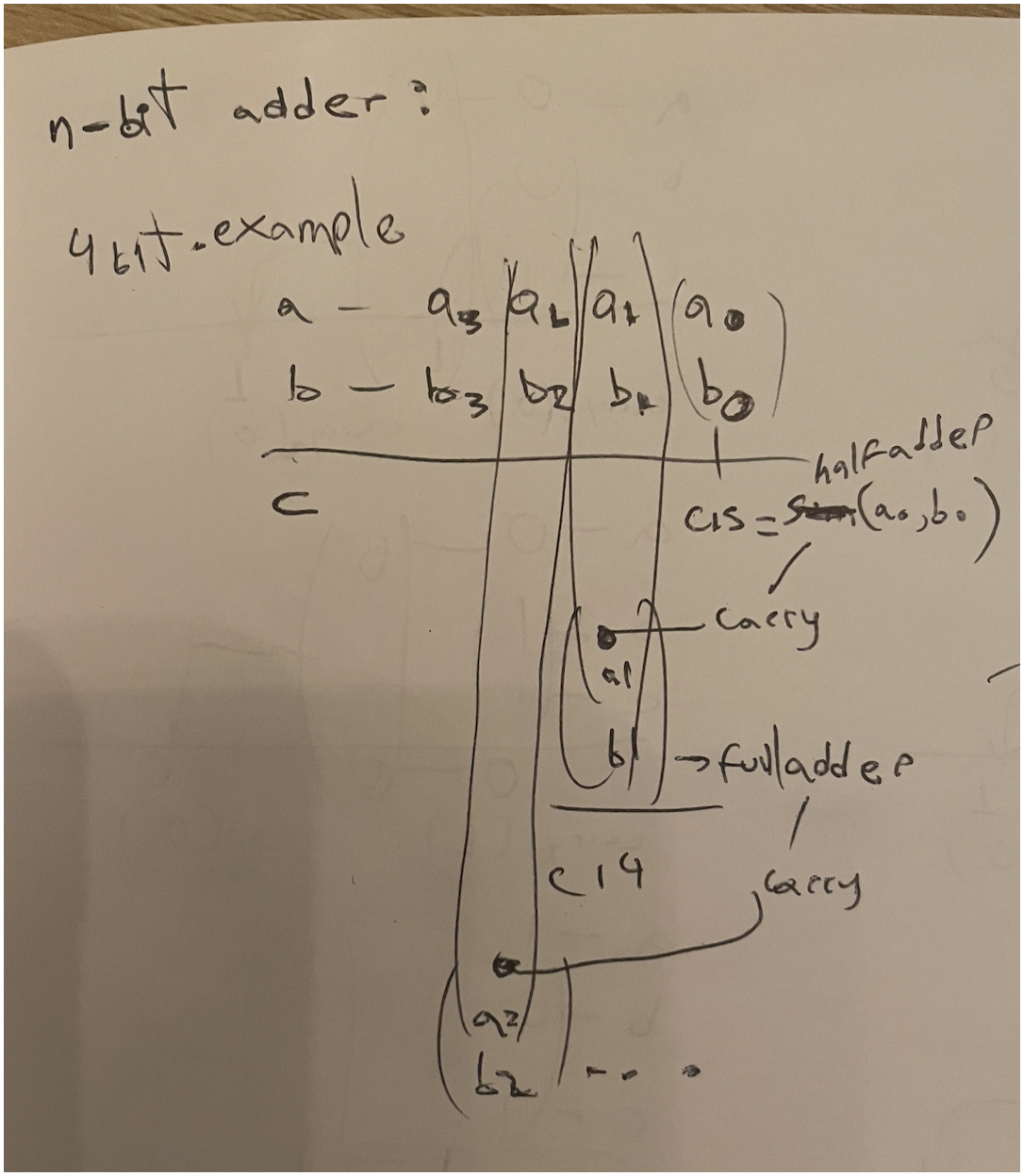
carry = a and b
sum = a xor b

Full-adder:
With the half-adder, the full adder can be composed by using two half-adders and combining the resulting carries with OR. The operation is similar to the common method of adding two numbers and using the carry when moving from right to left: (see page below)


a
b
c
half adder
sum
carry
half adder
sum
carry
carry

Adder:

With half-adders and full-adders available, creating n-bit adders is just combining them one after the other, starting with a half-adder (the two right-most bits don't have carry), and then full-adders to account for the next two bits from right to left plus the carry from the previous operation.



Incrementer:

To add 1 to a variable, in the HDL it can be done with only one Add16 gate, where a is the input variable, and b is an internal bus pin, altered to have the first bit set to true: b[0]=true, all other values for b are assumed to be 0. So this makes b to be: 0000000000000001

ALU:

The ALU can be done using the truth table (see pages above), in groups:

- The zeroing and negation of x and y can be done with simple gates.
- Then, to decide which one to use forward, a multiplexer serves the functionality of an “IF”
- Then, another multiplexer to decide whether f = x + y, or f = x & y
- Same for its negation
- Finally, zr and ng flags also use multiplexers, with the outputs from f as the deciding factor.