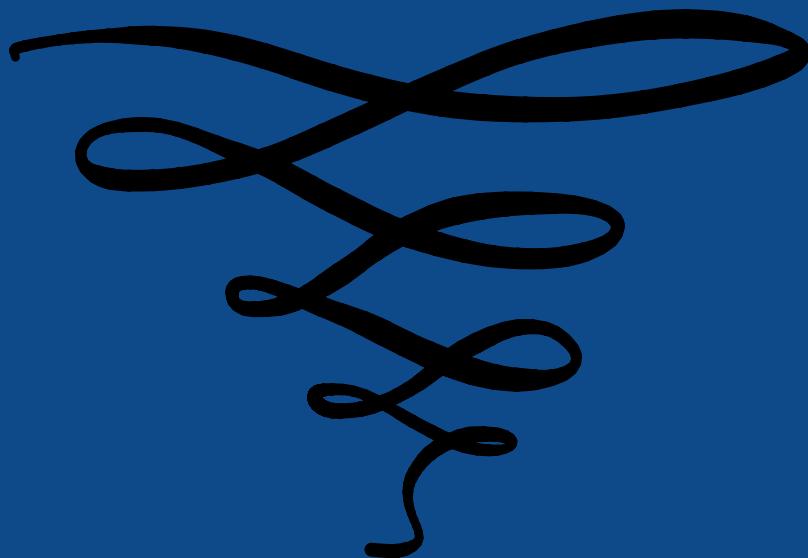


The Elements of Computing Systems

NAND2TETRIS

Notes



20 July 2020 - Ch. 1

Boolean Expressions - AND $\rightarrow (x+y)$

$$\text{OR} \rightarrow x \cdot y = xy$$

$$\text{NOT} \rightarrow \bar{x} = \bar{y} \rightarrow \begin{array}{c|c} 0 & 1 \\ 1 & 0 \end{array}$$

$f(x,y) = z$	$x+y \mid z$
$00 \mid 0$	$00 \mid 0$
$01 \mid 0$	$01 \mid 1$
$10 \mid 0$	$10 \mid 1$
$11 \mid 1$	$11 \mid 1$

But how
to deduce??

FIG 1

Canonical Representation

↳ Use truth table to deduce representation.

↳ Focus on what makes truth table = 1.

↳ Fig 1 → check Row 3 (true)

↳ Row 3: $x=0, y=1, z=0$.

$$\therefore \bar{x}y\bar{z}$$

$$RS = x\bar{y}\bar{z}$$

$$RF = xy\bar{z}$$

$$\sum_{\substack{\text{all true rows} \\ (3, 5, 7)}} f(x,y,z) = f(x,y,z) = \bar{x}\bar{y}\bar{z} + x\bar{y}\bar{z} + xy\bar{z}$$

Can you factor??

$$(\bar{x}y + x\bar{y} + xy) \cdot \bar{z}$$

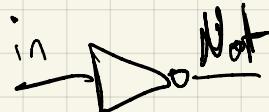
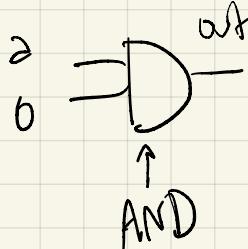
$$x\bar{y} + x(\bar{y} + y)$$

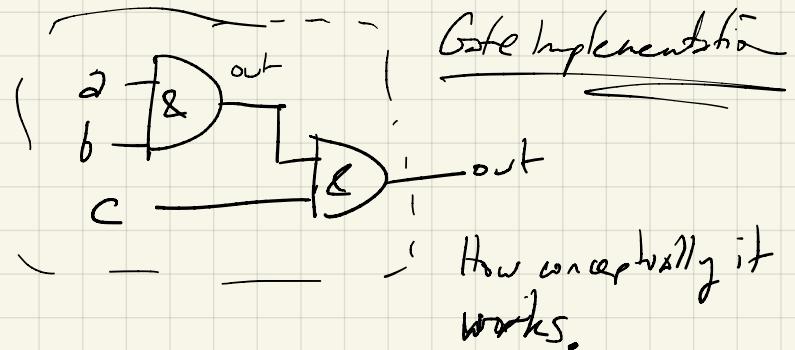
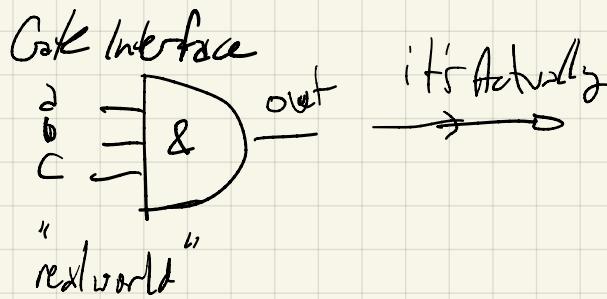
<u>No.</u>	<u>XOR</u>
00	0
01	1
10	1
11	0

Thus, all boolean expressions boil down to

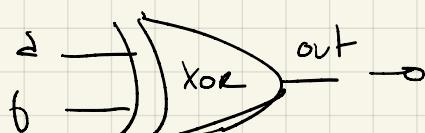
AND, OR & NOT

→ More Advanced versions of the operators include: NOR (\overline{OR}), XOR

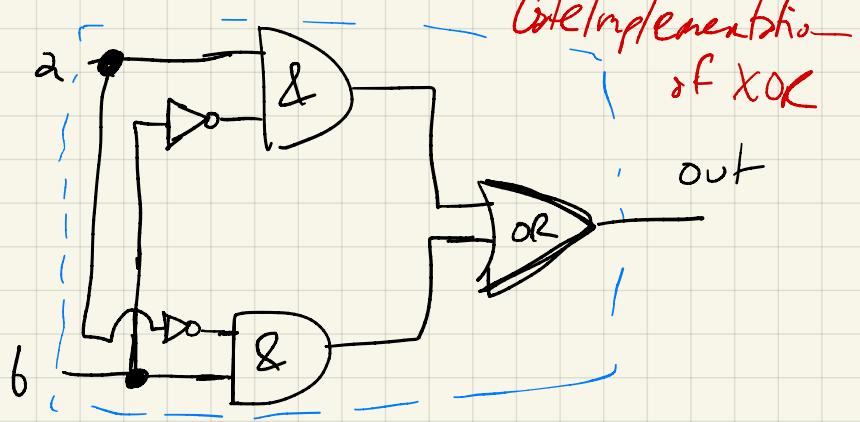




Gate Interface of XOR is unique....



a	b	f
0	0	0
0	1	1
1	0	1
1	1	0



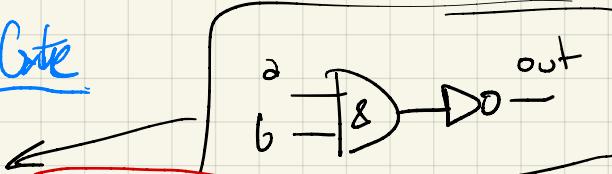
HDL = Hardware Description Language

VHDL \rightarrow Virtual

* Page 16 has HDL Code Example for XOR Gate *

1.2.1 - NAND Gate

a	b	NAND(a,b)
0	0	1
0	1	0
1	0	0
1	1	0



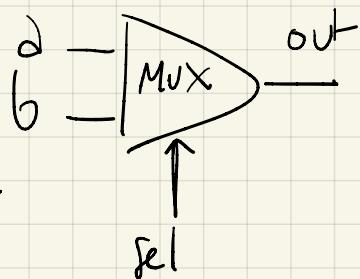
"Primitive" Chip (No need to implement)

"the starting point of our computer architecture is the NAND Gate
 \hookrightarrow From which All other gates & chips are built."

\rightarrow NOT Gate known as "inverter"

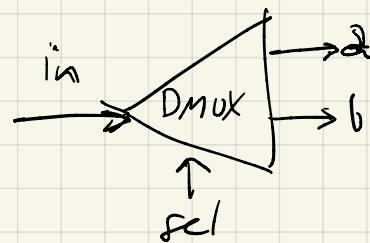
\rightarrow Multiplexer ("MUX") \Rightarrow 3 input gate to select an input as output.

\hookrightarrow i.e. \rightarrow If selector bit = 0, out = a, else out = b



Demultiplexer (DEMUX) \rightarrow

scl	a	b
ϕ	in	ϕ
1	ϕ	in



1) Multibit Gates are n -bit versions of primitive gates that still perform the same function.

* Not $16 \rightarrow$ 16 input converter

* And $16 \rightarrow 2 \times 16$ inputs, 16 pairs of AND Gates. Requires 16 inputs.
etc., etc.

* Mux $16 \rightarrow$ Same as binary Mux except with 16 pairs of inputs but still a single selector.

2) Multi-Way \rightarrow generalizes gates to n -inputs to 1 output.

↳ Ex: n -way OR Gate \rightarrow outputs 1 when at least 1 input of n bits = 1.

↳ An m -Way, n -bit MUX selects one of m n -bit input buses & outputs $\overset{\text{f}}{\rightarrow}$ to a single n -bit output bus. The selection is specified by a set of " k " control bits, where $k = \log_2 m$ (see Fig. 1-10, pg. 24)

→ The platform developed in this book requires two variations of this chip:

A 4-way 16-bit MUX, & an 8-way, 16-bit MUX.

We must construct all gates from NAND.

NDT

AND

OR/XOR

MUX/DMUX

Multibit NOT/AND/OR

Multibit MUX
Multiway Gates.

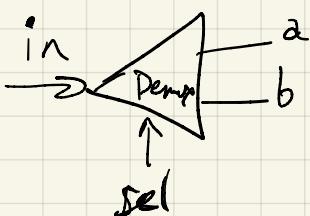
Chapter 1 Project

→ Construct ALL Gates from chapter 0/NAND.

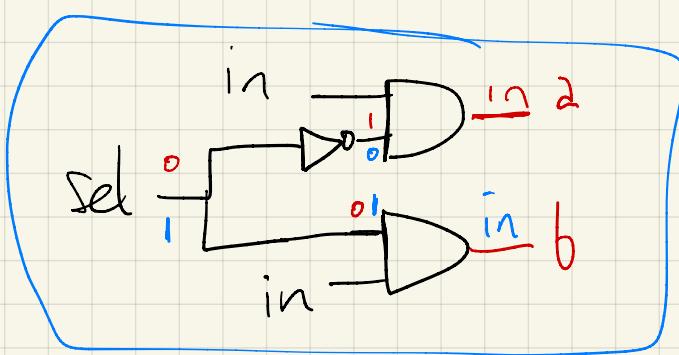
- 1) Read Appendix A, Sections A1-A6 only.
- 2) Go through Hardware Simulator Tutorial, parts I, II, & III only.
- 3) Build & Simulate all chips specified in projects/phi1 directory.
 - ↳ 1) ~~AND, AND16~~
 - 2) ~~DMux, DMux4way, DMux8way,~~
 - 3) ~~MUX, Mux4way16, Mux8Way16~~
 - 4) ~~Not, Not16~~
 - 5) ~~Or, Or8way, Or16~~
 - 6) ~~XOR~~

~~Not, And, Or, Xor, Mux, Demux~~ [Multi-bit Not, And, Or], Multi-bit Mux,
 Multi-Way Gates [~~Not16, And16, Or16, Mux 8way16, DMux8way~~]

sel	a	b
0	in	0
1	0	in



Or(a,b)		Nand(a,b)		And(a,b)		Xor(a,b)	
a	b	a	b	a	b	a	b
0	0	1	0	1	1	0	0
0	1	0	1	0	1	0	1
1	0	1	0	0	1	1	0
1	1	1	1	0	0	1	1



Multibit Basic Gates (Not, And, Or)

1) Multibit Not

Chapter 2 Notes - 8 Aug 2020

→ Goal of chapter 2 is to fully implement & finish our ALU.

↳ Context → Adder Chips

* Binary Addition

$$\begin{array}{r}
 \begin{array}{c}
 0001 \\
 1001 \\
 + 0101 \\
 \hline
 01110
 \end{array}
 \end{array}$$

since $= 0$, \rightarrow Registers involved
 \therefore No overflow

carry Row:
 $1+1=0(+1)$
 carry

$$\begin{array}{r}
 \begin{array}{c}
 1111 \\
 1011 \\
 + 0111 \\
 \hline
 10010
 \end{array}
 \end{array}$$

carry
 $1+1+1=1+(1)$
 overflow!

Signed Binary Numbers → Most significant Bit = negative connotation.

Positive #'s	Negative #'s (Signed)
0	0000
1	1111
2	1110
3	1101
4	1100
5	1011
6	1010
7	1001
8	1000

2's Complement (Radix Complement)

$$\bar{x} = \begin{cases} 2^n - x & \text{if } x \neq \phi \\ \phi & \text{otherwise} \end{cases}$$

Example:

$\boxed{-2}$ for 5-bit system

$$\Rightarrow \bar{-2} = 2^5 - 2_{10} \quad \boxed{\downarrow}$$

$$= 32_{10} - 2_{10} = 30_{10} \quad \boxed{\downarrow}$$

$$\boxed{30_2 = 11110_2 = -2_{10}}$$

$$11110_2 + 00010_2 \quad \boxed{\downarrow}$$

$$\begin{array}{r}
 11110 \\
 + 00010 \\
 \hline
 100000
 \end{array}$$

5-bit system, $= 0!$

Overflow is ignored b/c
book said
so...

Quick Method to achieving negative number in binary:

MSB = 1, invert all other bits, then add 1.

Subtraction is just adding negative # $\rightarrow x - y = x + (-y)$

Adders: 1) Half Adder: Designed to add 2 bits,

2) Full Adder: " " " 3 bits.

3) Adder: " " " n bits.

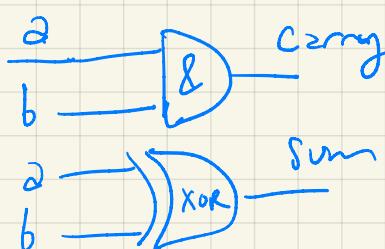
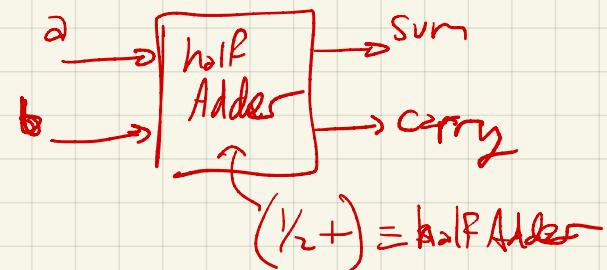
Half-Adder Chip:

$N = 2, 6$

$OUT = sum, carry$

$\hookrightarrow f_n: LSB\ of\ a+b = sum$
 $MSB\ of\ a+b = carry$

a	b	Carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



halfAdder!

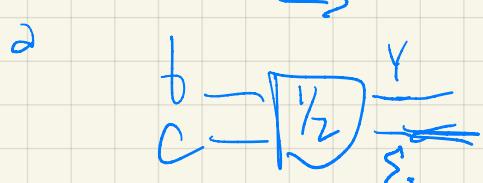
Full-Adder Chip:

$N = 2, 6, C$

$OUT = sum, carry$

$\hookrightarrow f_n: sum = LSB\ of\ a+b+c$
 $carry = MSB\ of\ a+b+c$

$$a+b+c = (b+c) + a$$



a	b	c	Y	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

0	0	0	0	X
0	1	1	1	X

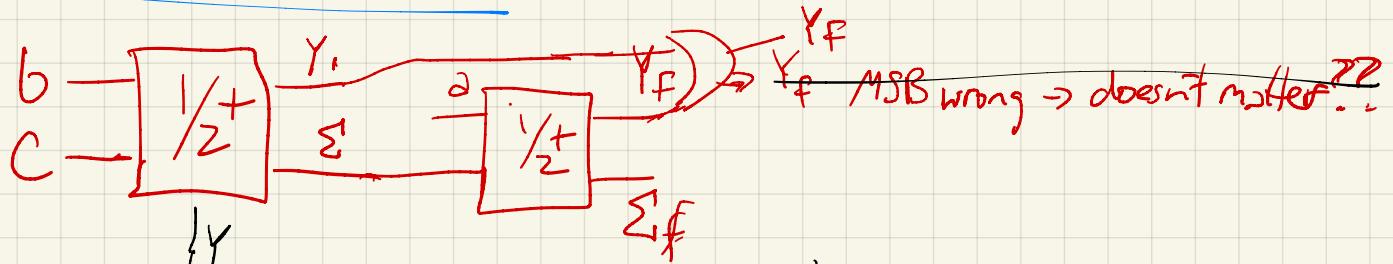
a	Σ	Y	Σ
1	0	0	1
1	1	1	0
1	1	1	0
1	0	0	1

$\text{if } a=0 \rightarrow \Sigma = \text{sum}$

$\text{if } a=1 \rightarrow \Sigma = \text{Notsum}$

do't care

Full Adder via Half Adders: $a+b+c$ (36f) $\Sigma \equiv \text{sum}, Y \equiv \text{carry}$



b	c	Σ
0	0	0
0	1	1
1	0	1
1	1	0

OR

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

AND XOR

a	b	Σ	Y
1	0	0	1
1	1	1	0
1	0	1	0
0	1	0	1

MSB wrong??

16 Bit Adder:

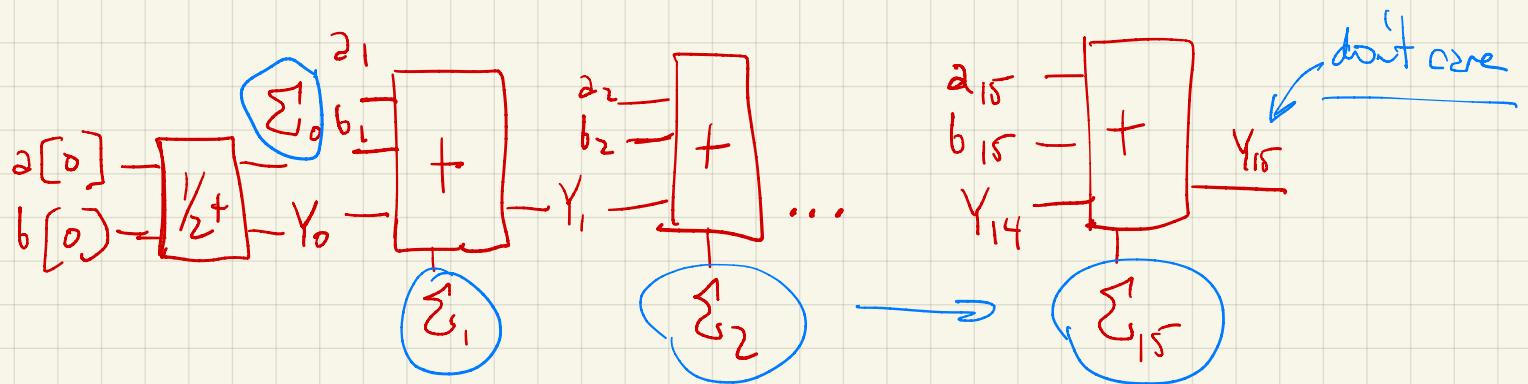
$| N \Rightarrow a[16], b[16]$

$f_n: \text{out} = a+b$

Integer 2's complement Addition
overflow is neither detected nor
handled.

$\text{OUT} \Rightarrow \text{out}[16]$

from § 2.3 (pg 38) \rightarrow the LSBs are added, + the carry fed into the addition of the next pair of significant bits. (full adder, i.e 36 bits).



Incrementer, 16-Bit:

IN \Rightarrow in[16]

fn: out = in + 1

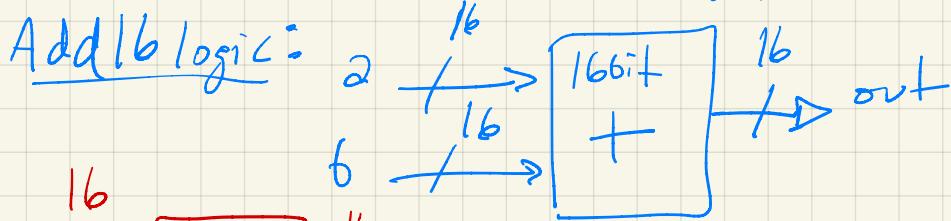
Overflow neither detected nor handled.

OUT \Rightarrow out[16]

Integer 2's complement addition.

§2.3 (pg. 38) \rightarrow "n-bit incrementer can be trivially implemented from an n-bit Adder."

Add 16 logic:



ALU:

IN \Rightarrow x[16], y[16]

6 control
Bit
inputs

$\begin{cases} ZX \rightarrow \text{zero } x\text{-input} \\ NX \rightarrow \text{negate } x\text{-input} \\ ZY \rightarrow \text{zero } y \\ NY \rightarrow \text{negate } y \\ f \rightarrow 0 = \text{And}, 1 = \text{Add} \\ NO \rightarrow \text{negate output.} \end{cases}$

OUT \Rightarrow out[16]

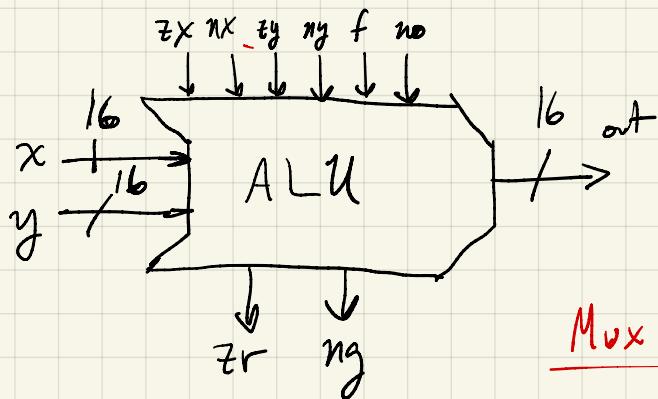
$$Zr \left\{ \begin{array}{l} 1 \\ 0 \end{array} \right.$$

out = 0 \leftarrow 16-bit eq. comparison
out \neq 0

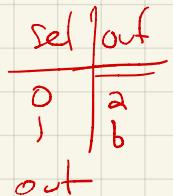
$$ng \left\{ \begin{array}{l} 0 \\ 1 \end{array} \right.$$

out ≥ 0 \leftarrow 16-bit negative comparison
out < 0

§2.3 (pg. 39): "Your first step will likely be to create a logic circuit that manipulates a 16-bit input according to $ZX + NX$ ($+y$). Then integrate the 'f' bit functionality."



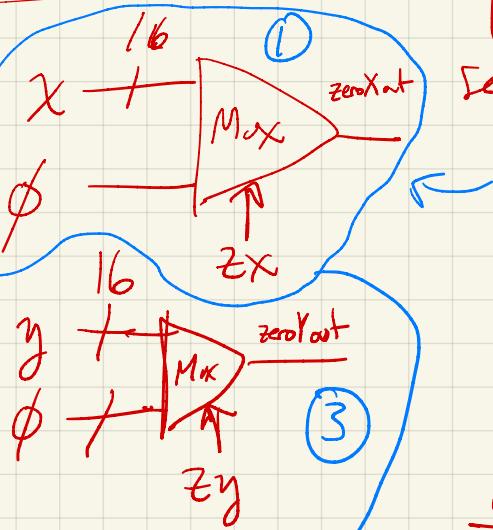
8 inputs, 3 outputs
 - 16 bits
 two 16bit / one 16bit.
 6 bits / 2 bits.



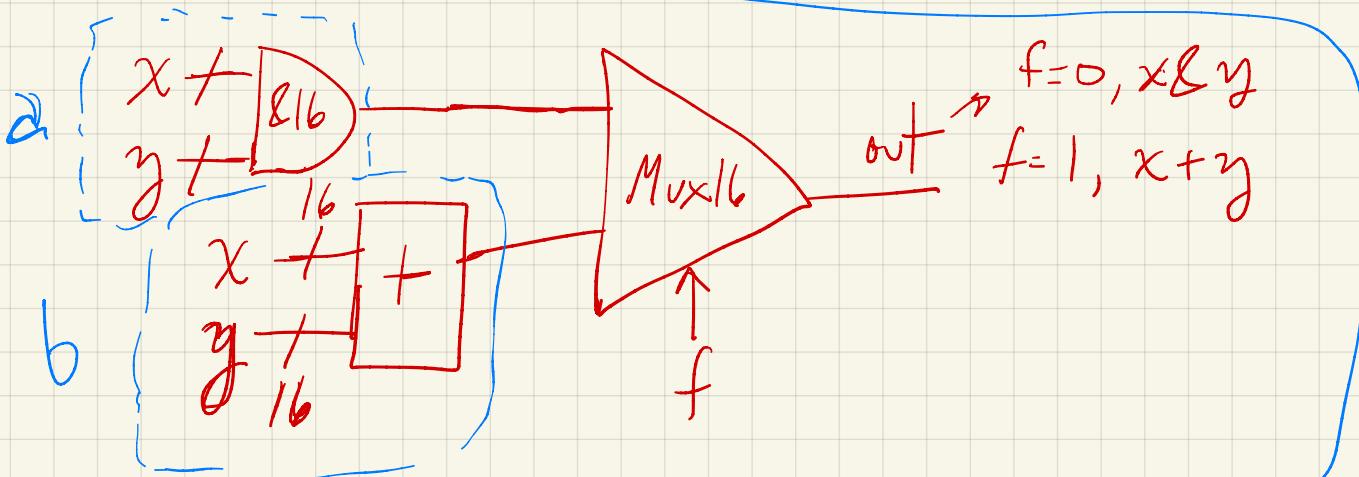
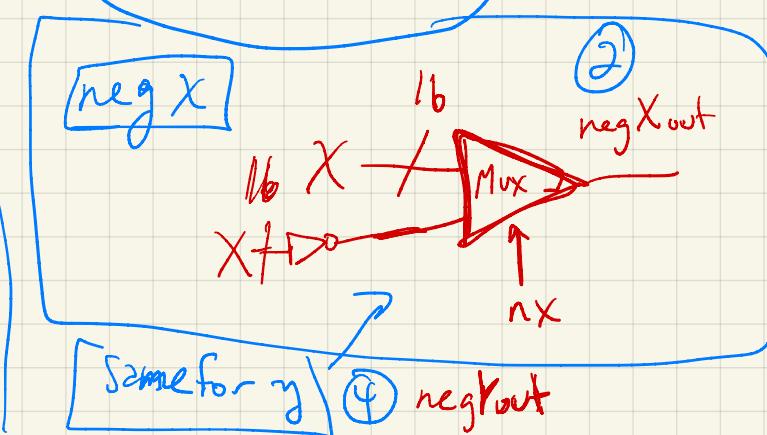
Mux per x or y ??

- 1) construct zero X ✓
- 2) construct neg. X ✓
- 3) zero y ✓
- 4) neg y ✓
- 5) $x+y$ ✓
- 6) $x \cdot y$ ✓
- 7) not output

8) zr check
9) ng check



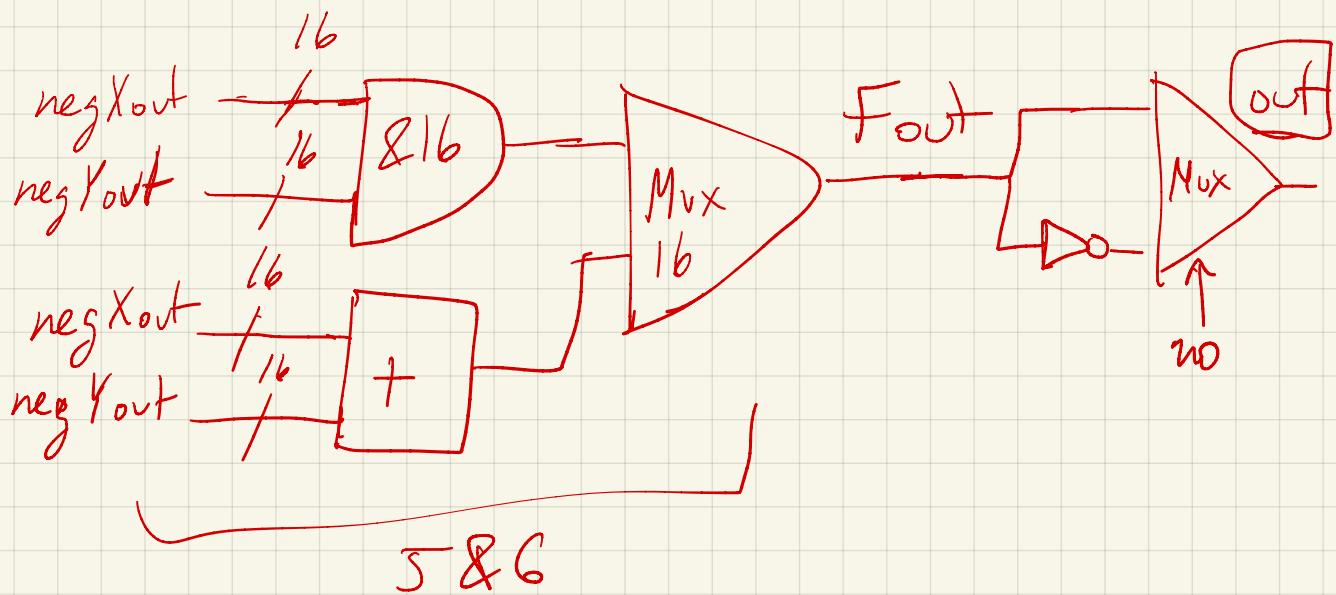
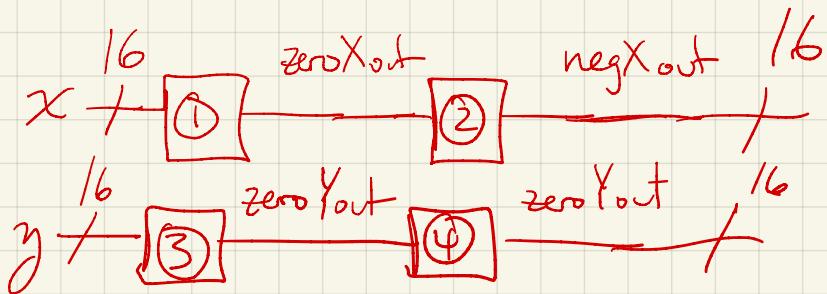
will need to keep track of each OUT during implementation



Chip 1 = zero X
 2 = neg X
 3 = zero Y
 4 = neg Y

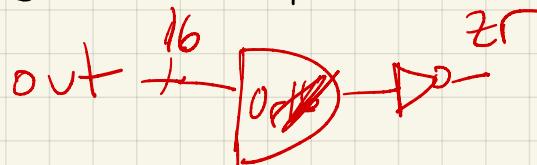
$$5 \& 6 = f$$

7) not out

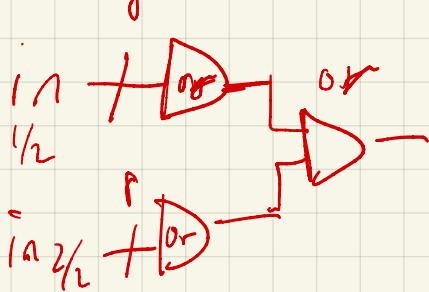


Zr :

$$Zr \left\{ \begin{array}{l} 1 \\ 0 \end{array} \right. \begin{array}{l} \text{out} = 0 \\ \text{out} \neq 0 \end{array}$$



Or/6-way



ng : ($MSB = 1$ or not)

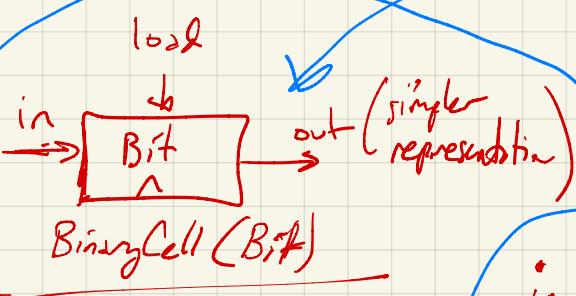
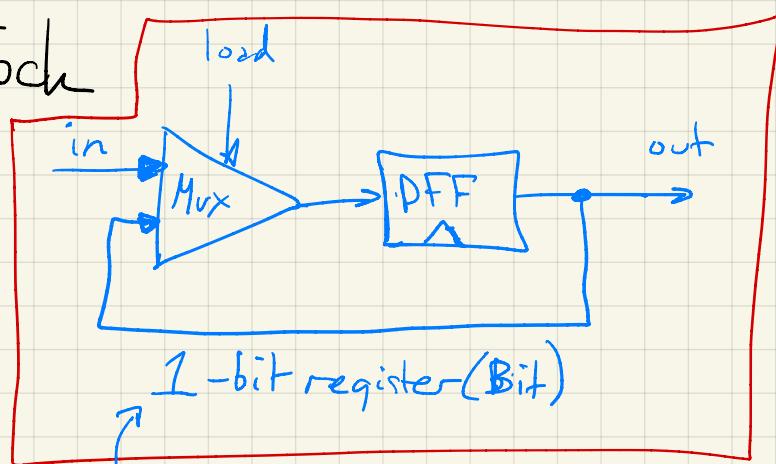
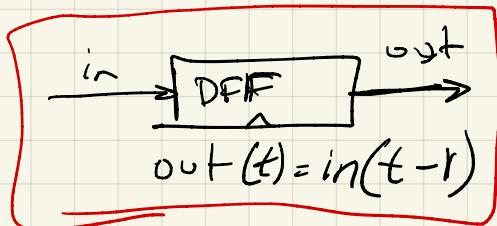
$$ng \left\{ \begin{array}{l} 0 \\ 1 \end{array} \right. \begin{array}{l} \text{out} \geq 0 \\ \text{out} < 0 \end{array}$$

$$\text{out}[15] \rightarrow ng$$

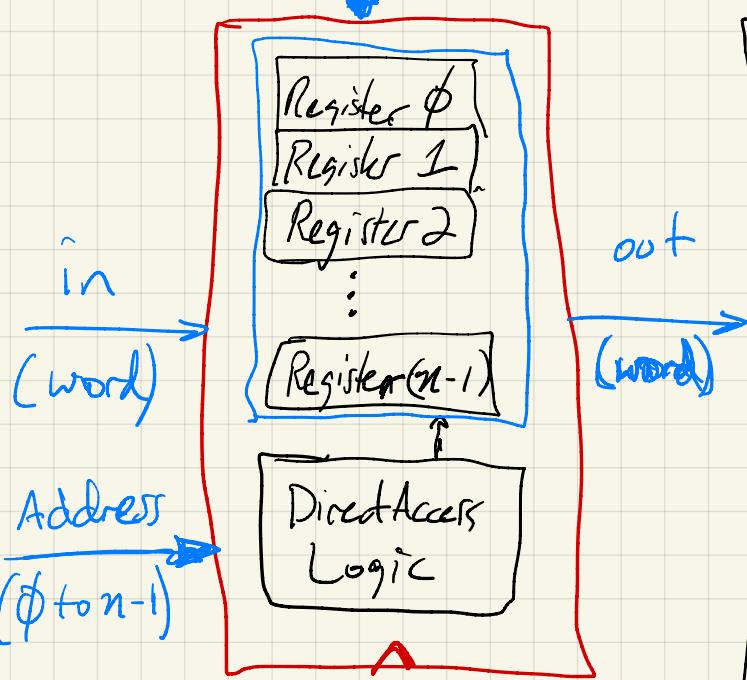
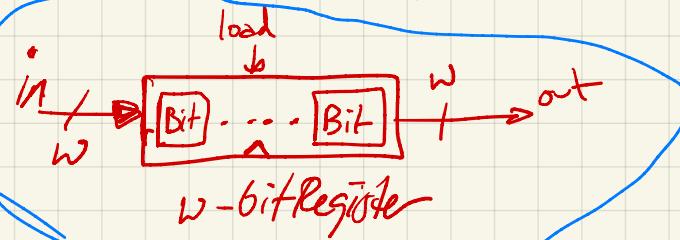
Ch. 3 Notes - 15 Aug 2020

Flip-Flops - basis of our clock. (We use Data Flip-Flop, aka DFF)

Clock Cycle \rightarrow Tick-Tock



if load(t-1) then out(t) = in(t-1)
else out(t) = out(t-1)



RAM Chip Concept Model

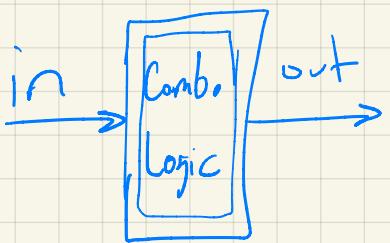
Counters - out(t) = out(t-1) + C, where C = 1 usually..

CPUs include program counters whose output is interpreted as the address of the instruction that should be executed next in the current program.

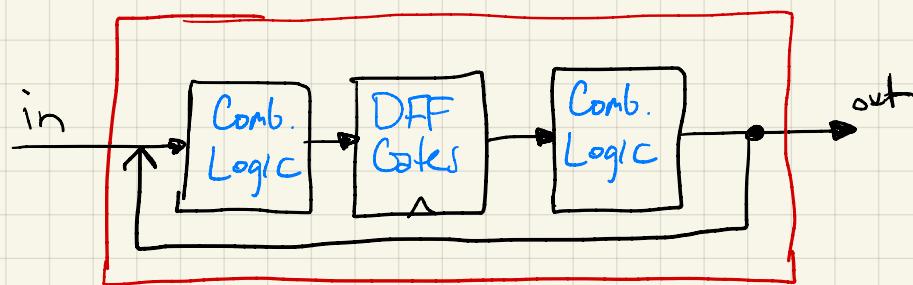
Semiconductor Chips

↳ Semiconductor chips always consist of 2 sandwiched layers of DFFs between optional combinational logic layers.

Combinational Chip (pg. 46)



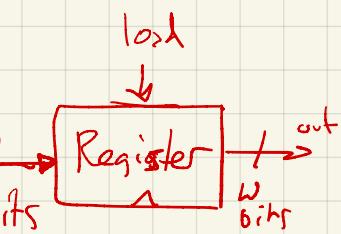
$out = \text{somefunction of } (in)$



$out(t) = \text{somefunction of } (in(t-1), out(t-1))$



Bit
IN: in , load
OUT: out



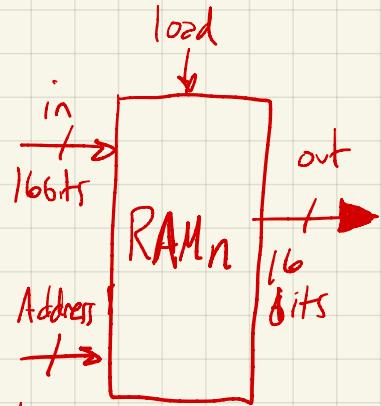
Register (16-bit)
IN: $in[16]$, load
OUT: $out[16]$

fn: If $\text{load}(t-1)$ then: $out(t) = in(t-1)$ } " is a 16-bit operation
L else: $out(t) = out(t-1)$ }

Register is just w -bits wide!

Memory: An array of n w -bit registers equipped w/ direct Access Circuitry.

The # of registers (n) + the width of each register (w) are called the memory's size + width, respectively.



RAM n :

IN: $in[16]$, address $[k]$, load
OUT: $out[16]$

fn: $out(t) = RAM[\text{address}(t)](t)$

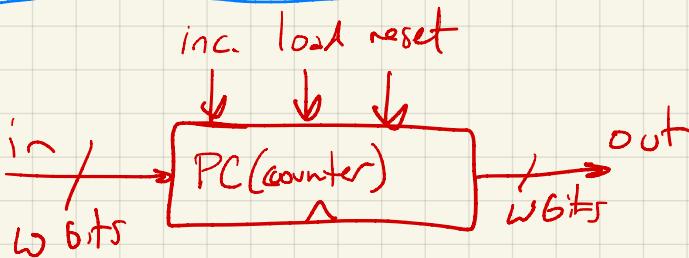
IF: $\text{load}(t-1)$, then

$\rightarrow RAM[\text{address}(t-1)](t) = in(t-1)$

" " is a 16-bit operation

The specific RAM Chips needed for the Hack platform are:

Chip Name	n	k
RAM 8	8	3
RAM 64	64	6
RAM 512	512	9
RAM 4K	4096	12
RAM 16K	16384	14



PC (16-bit Counter) ($W=16$):

IN: $in[16]$, inc, load, reset

OUT: $out[16]$

fn: If: $reset(t-1)$ true, then $out(t) = \phi$

↳ elseif $load(t-1)$ true, then $out(t) = in(t-1)$

↳ elseif $inc(t-1)$ true, then $out(t) = \underline{out(t-1) + 1}$

↳ else $out(t) = out(t-1)$

To Build!

1-Bit Register (Bit)

16-Bit Register (Register)

RAM8, RAM64, RAM512

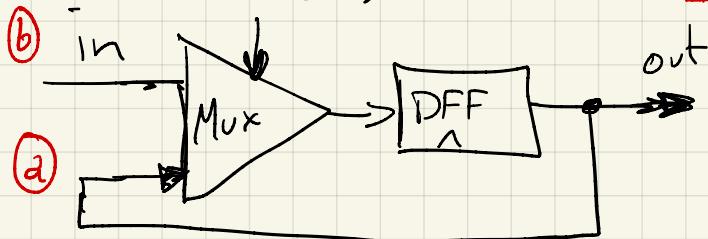
RAM 4K, RAM 16K

16-Bit Counter

1-bit Register (pg. 43)

load(sel) ← b/c

load = 0 → feedback loop
load = 1 → use new input!



To Build!

✓ 1-Bit Register (B&F)

✓ 16-Bit Register (Registers)

RAM8, RAM64, RAM512

RAM 4K, RAM16K

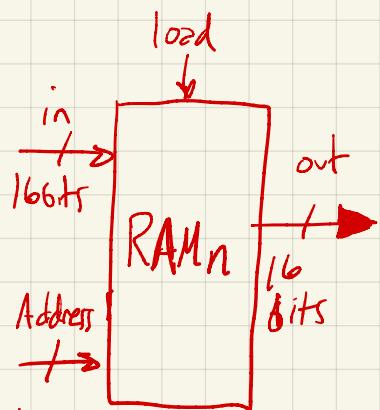
16-Bit Counter.

16-bit Register (pg. 49)



Use 16 Bits from above ...

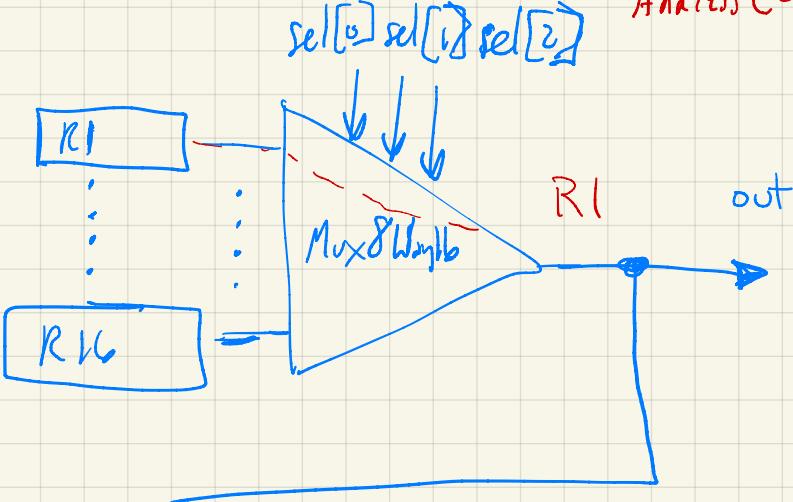
RAM 8: → RAM 8 is 8 16-bit registers...



$$\log_2 n \text{ bits} \Rightarrow \log_2 8 = 3$$

smells like a Mux...

Mux 8 Way [16]



Address[0] → sel[0]

⋮

Address[2] → sel[2]

Address chooses Register to be written...

Load tells register to pick new input.