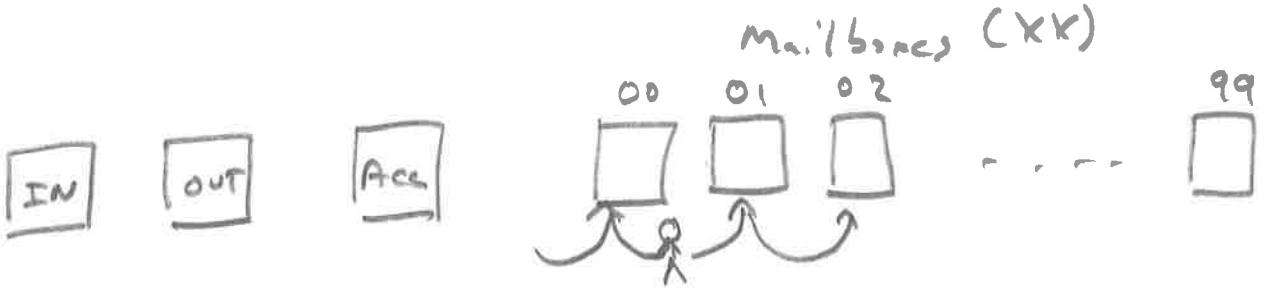
Microprocessors

and

Assembly

Little Man Computer



Input → user input

Output → e.g. to screen

Acc → accumulator (for addition + subtraction)

Program Counter → starts at zero, increments prior to each instruction, but may be overwritten by the instruction.
(Next instruction taken from address in PC)

Mail boxes: Stores data and program.

LMC Instruction Set

1XX	ADD	Add value in mail box XX to accumulator
2XX	SUB	Subtract value "
3XX	STA	Store contents of Acc into mail box XX.
5XX	LDA	Load contents of XX into the accumulator
6XX	BRA	Set PC to mailbox XX
7XX	BRZ	Set PC mailbox XX if Acc is zero
8XX	BRP	Set PC mail box XX if Acc is zero or positive
901	IND	load input to Acc
902	OUT	Send Acc to Out
000	COB	Coffee Break

Example : Adder (A + B)

```
901     INP
350     STA  FIRST
901     INP
150     ADD  FIRST
902     OUT
000     COB
FIRST   DAT
```

Example Subtracter (A - B)

```
901     INP
350     STA  FIRST
901     INP
351     STA  SECOND
550     LDA  FIRST
251     SUB  SECOND
902     OUT
000     COB
```

Example Multiplier

```
00   901   INP          input "a"
01   350   STA A        store "a"
02   901   INP          input "b"
03   351   STA B        store "b"
04   711   LOOP BRZ DONE
05   552   LDA SUM
06   150   ADD A
07   352   STA SUM
08   551   LDA B
09   253   SUB ONE
10   604   BRA LOOP
11   552   DONE LDA SUM
12   902   OUT
13   000   COB
--   --
50   -     A DAT
51   -     B DAT
52   0     SUM DAT 0
53   1     ONE DAT 1
```

* H.W. LMZ to integer divide A/B

6 3 → 2
5 3 → 1
7 3 → 2

Comparator A > B

AND 

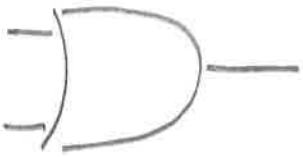
AB

A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

NAND 

$\overline{AB}$

0	0	1
0	1	1
1	0	1
1	1	0

OR 

$A + B$

0	0	0
0	1	1
1	0	1
1	1	1

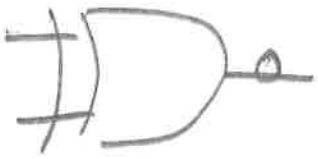
NOR 

0	0	1
0	1	0
1	0	0
1	1	0

XOR

A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

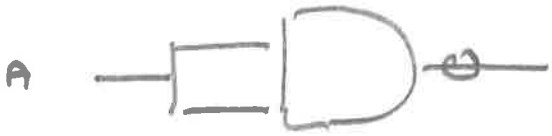


0	0	1
0	1	0
1	0	0
1	1	1

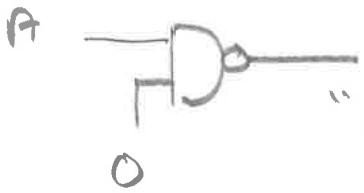
For the minimalists...

NAND and NOR are each functionally complete... you can construct all logic from enough of either.

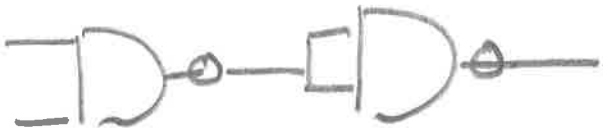
Main tricks, inverter:



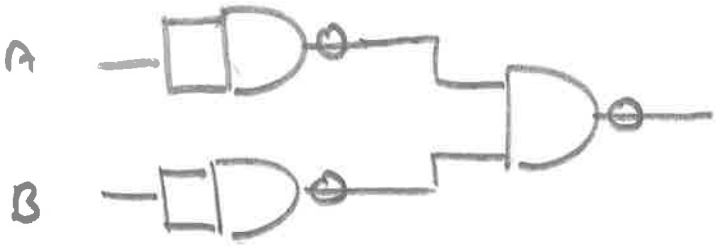
Ignore:



AND:

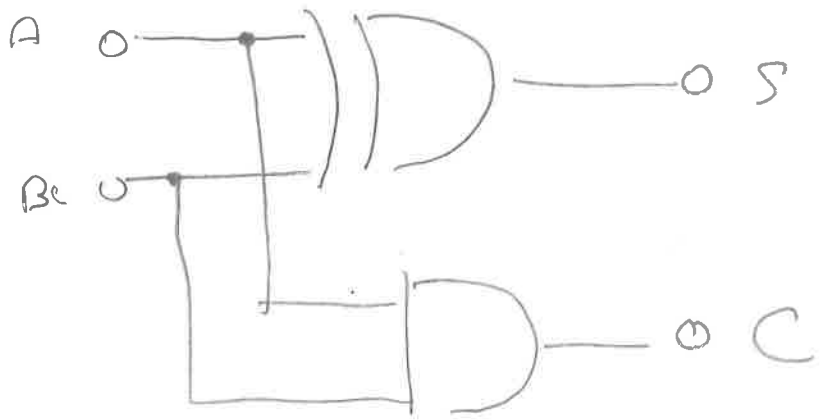


OR:

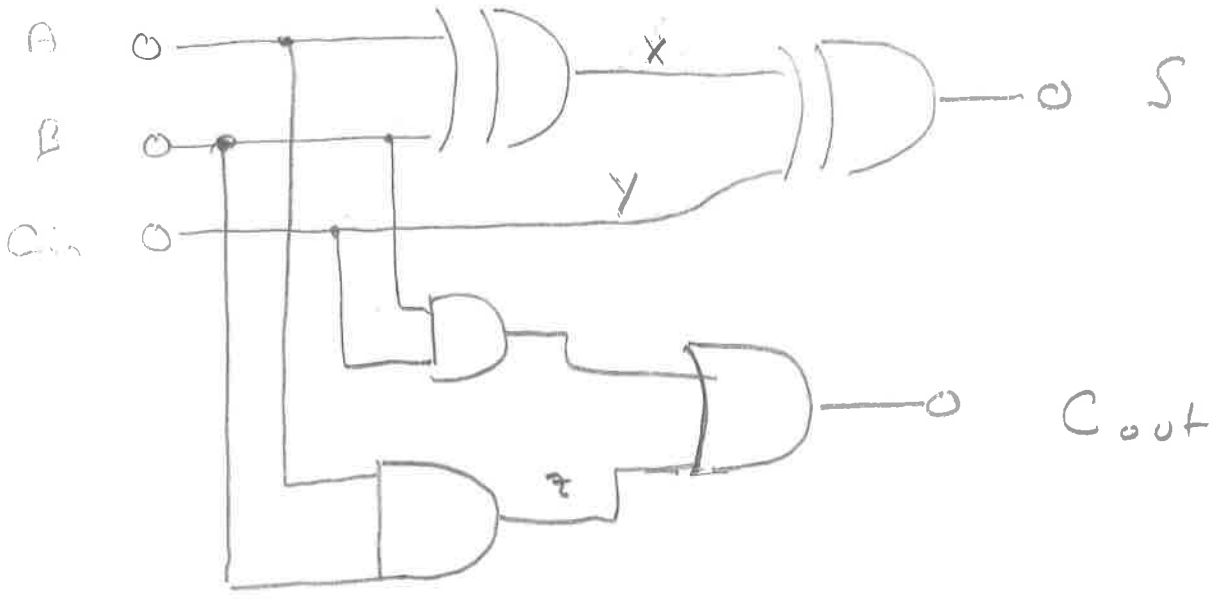


Half Adder

0	1	1	1
0	0	1	1
0	0	0	1
00	01	10	11

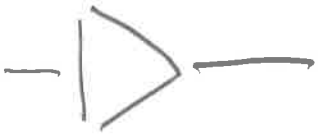


Full Adder



"Trick" to understanding is that
$1+1+(1) = 11 = 3$ is biggest number possible
$(\Leftrightarrow \quad z = 1 \quad \Rightarrow \quad x = 0)$

Buffer



A	O
0	0
1	1

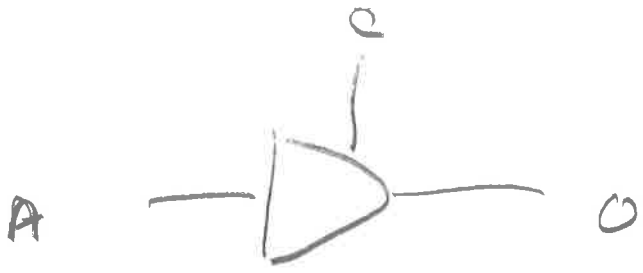
Inverter

A	O
1	0
0	1

Buffer from Inverter

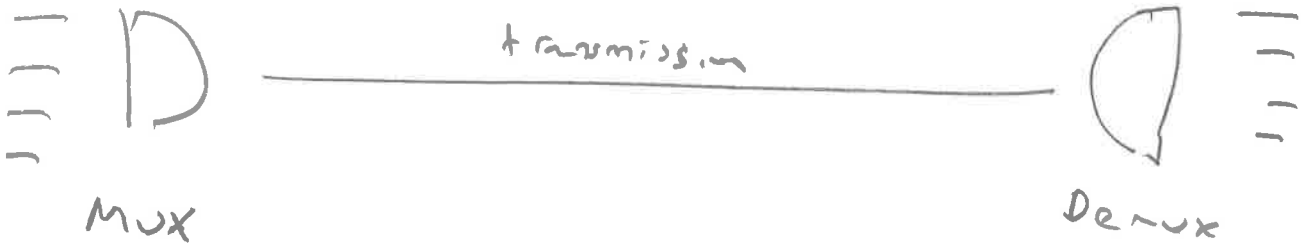
Inverter from Buffer

Tri-State Buffer

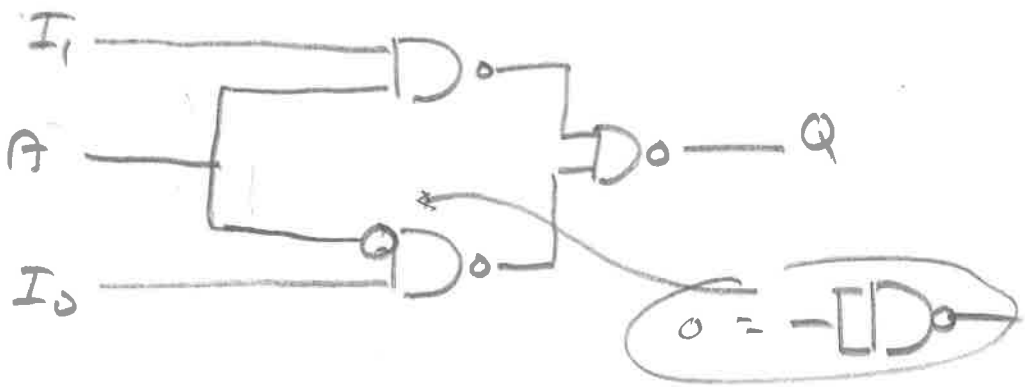


A	C	O
0	0	Z
1	0	Z
0	1	0
1	1	1

Multiplexing

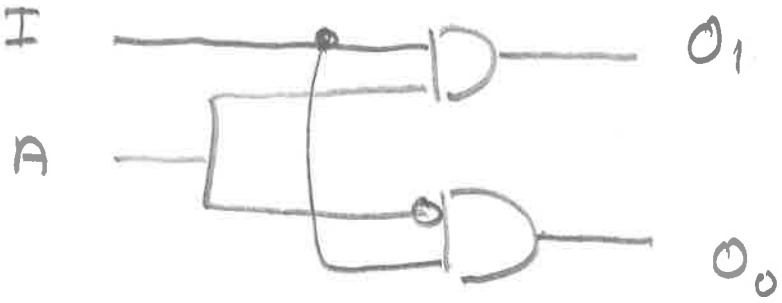


1-bit Mux



A	Q
0	I_0
1	I_1

1-bit Demux



A	Q_0	O_1
0	I	0
1	0	I

MUX, DEMUX, Tri-State Buffer

MUX

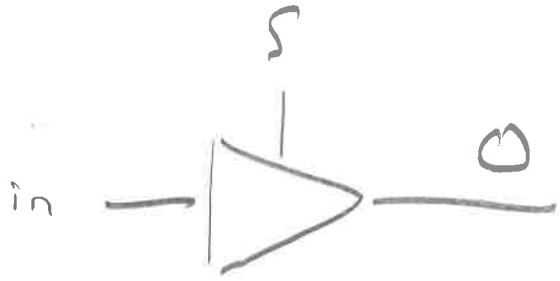
A	B	S	O
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

S	O
0	A
1	B

DEMUX

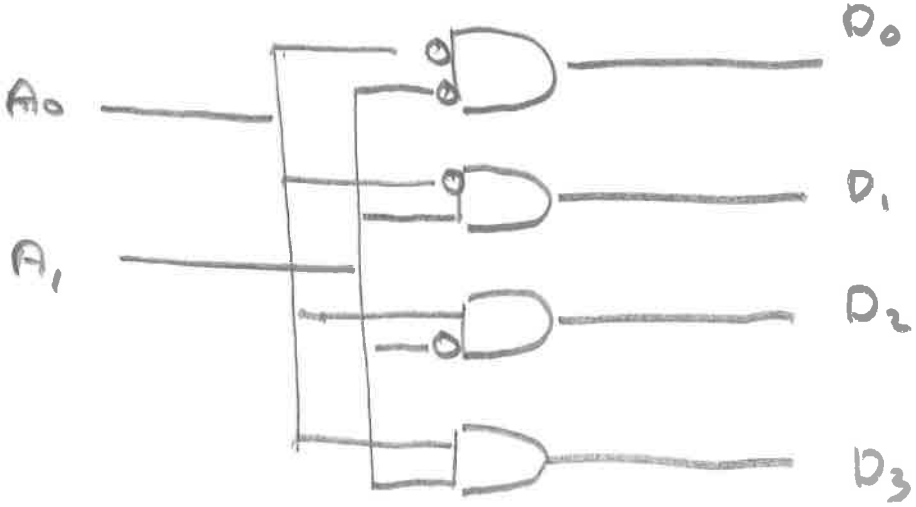
A	S	C	D
0	0	0	0
1	0	1	0
0	1	0	0
1	1	0	1

S	C	D
0	in	0
1	0	in



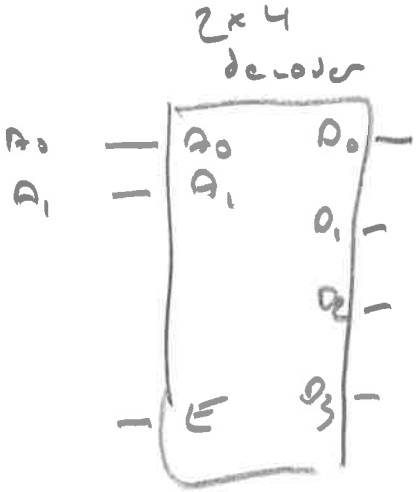
S	O
0	Z
1	in

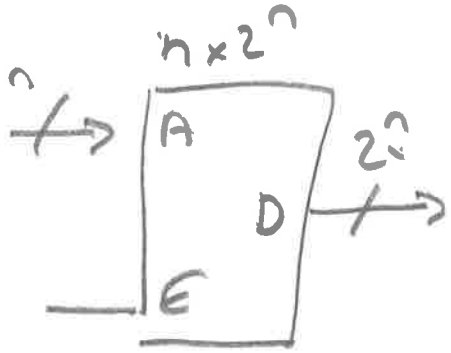
De coder

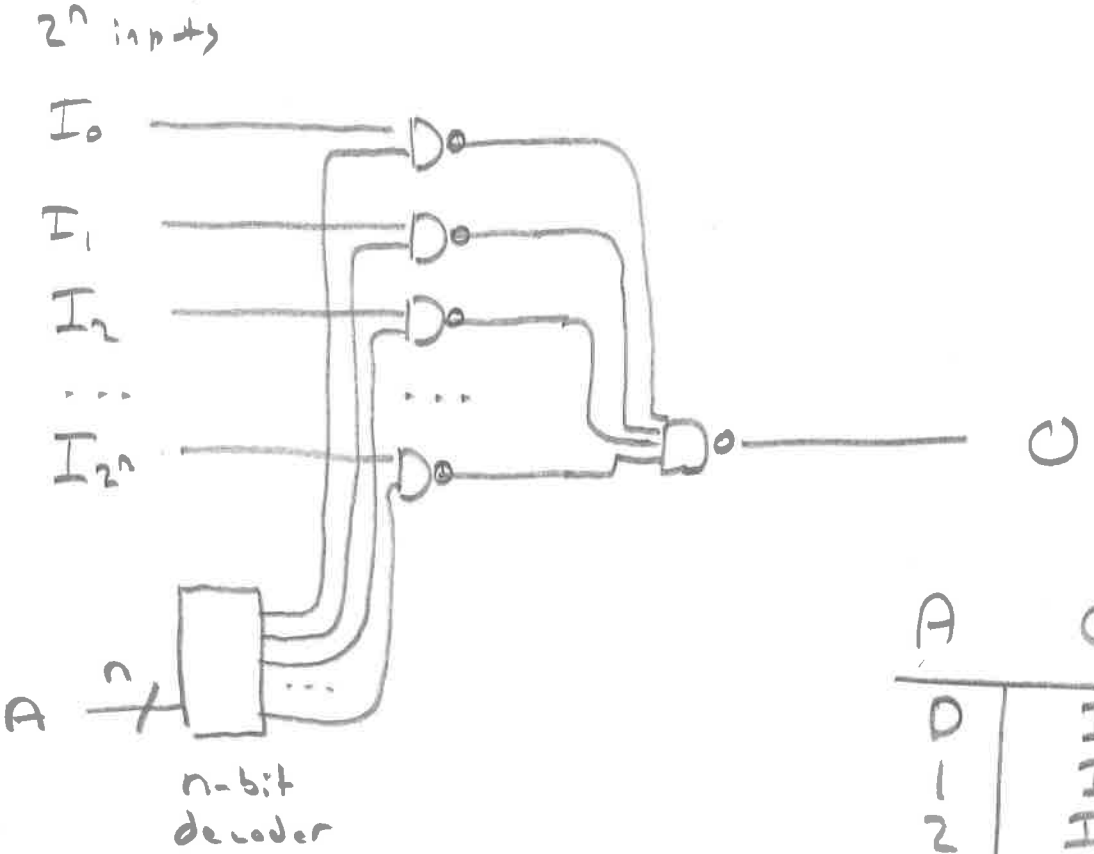


3-bit:

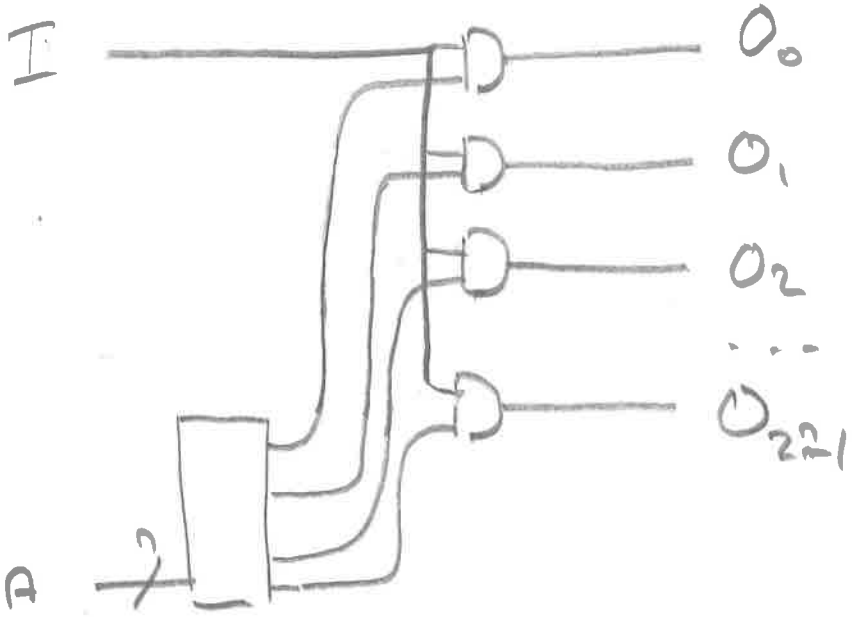
A_0	A_1	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1







A	O
0	I_0
1	I_1
2	I_2
...	...
(2^n-1)	$I_{(2^n-1)}$



A	O_0	O_1	O_2	...	O_{2^n-1}
0	I	0	0		0
1	0	I	0		0
2	0	0	I		0
...					
2^n-1	0	0	0		I

Multiplexer

Number of inputs : N_i

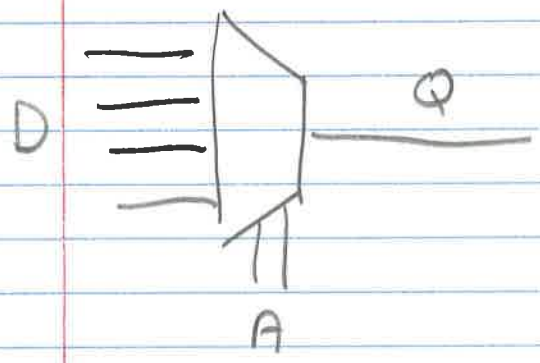
Number of outputs $N_o \geq 1$

Number of address bits: N_A

$$2^{N_A} = N_i / N_o$$

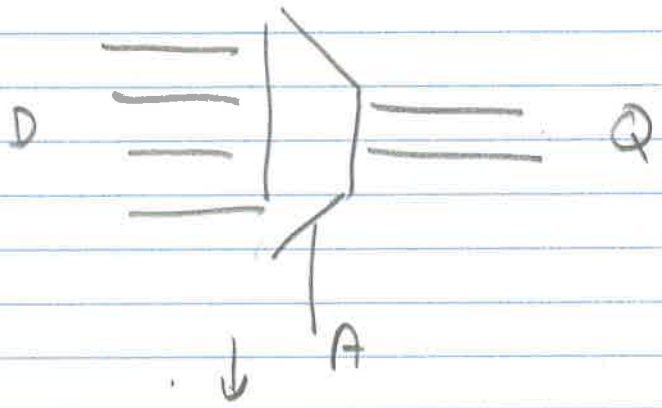
e.g.

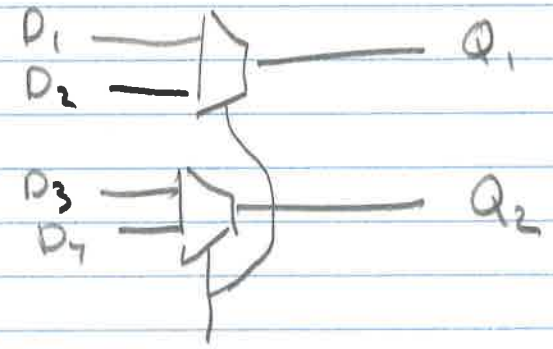
4:1



$2^2 = 4/1$

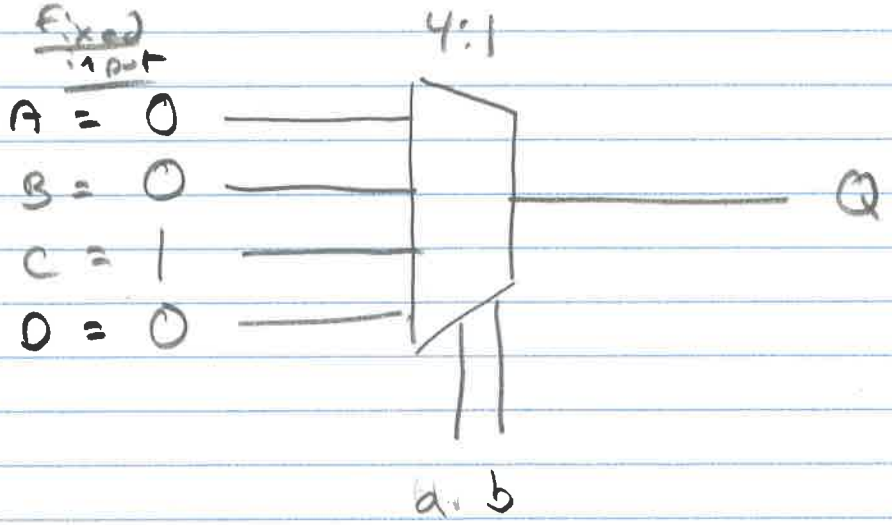
4:2





$2^1 = 4/2$

Mux as Generic Logic



a	b	Q	
0	0	A	
0	1	B	
1	0	C	
1	1	D	

Set to what we want

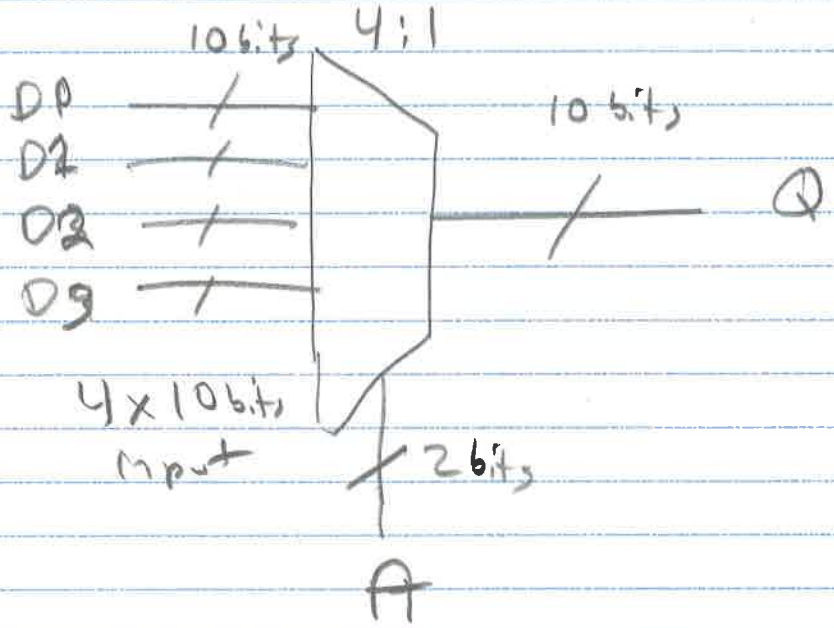
Configurable Logic!

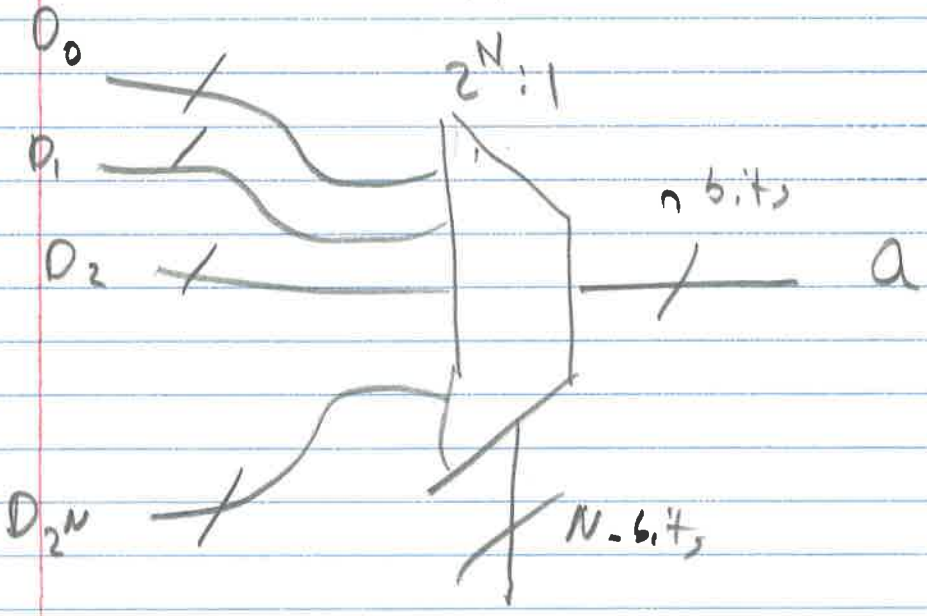
Width:

Often, we might want to associate m-bits with a particular address

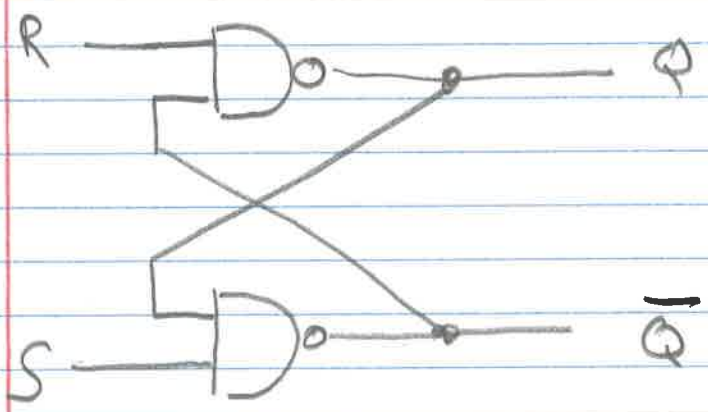
(Accumulator in LMC requires 10 bits ≤ 1024)
and Mailboxes

→ 10 bits at address 0x3
10 bits at address 0x2





RS latch



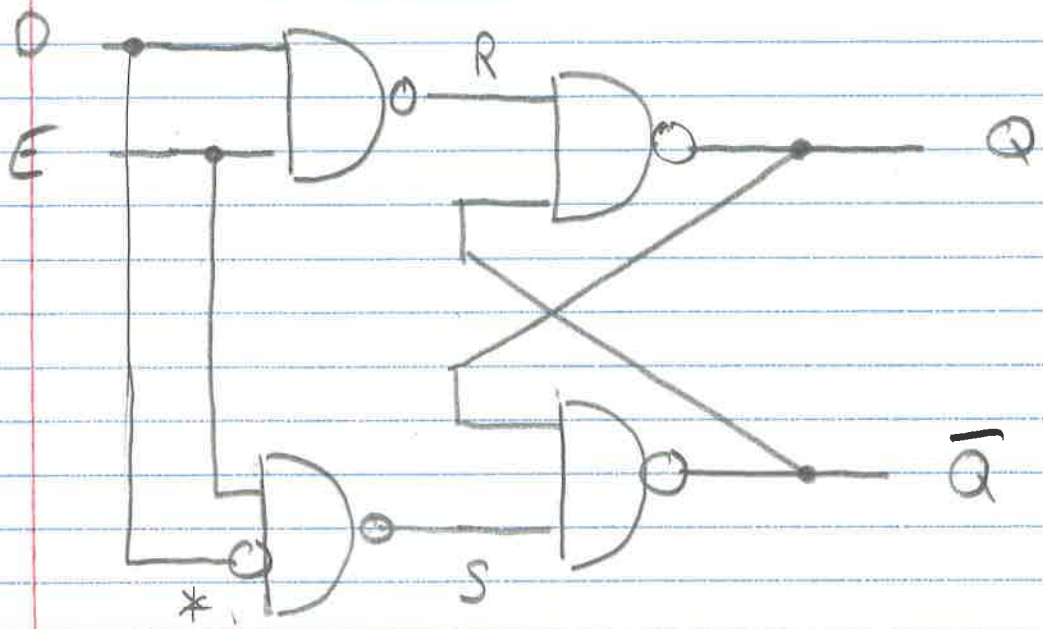
RS both high $Q=0$ $\overline{Q}=1$ stable
$Q=1$ $\overline{Q}=0$ stable
($Q=\overline{Q}$ not stable)

$R=1$ $S=0$, $S=0$ forces $\overline{Q}=1$ (not)
$\overline{Q}=1$ and $R=1$ forces $Q=0$

$R=0$ $S=1$ $R=0$ forces $Q=1$
$Q=1$ and $S=1$ forces $\overline{Q}=0$

RS both 0 → $Q=\overline{Q}=0$ invalid state

D-type Flip-Flop



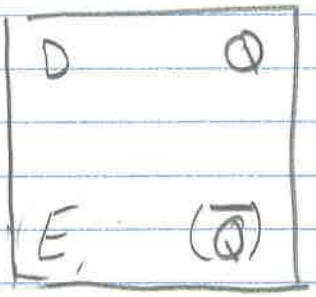
$E = 0 \Rightarrow R = S = 1 \Rightarrow Q$ constant

$E = 1 \Rightarrow R = \bar{D} \quad S = D \Rightarrow Q = D \quad \bar{Q} = \bar{D}$

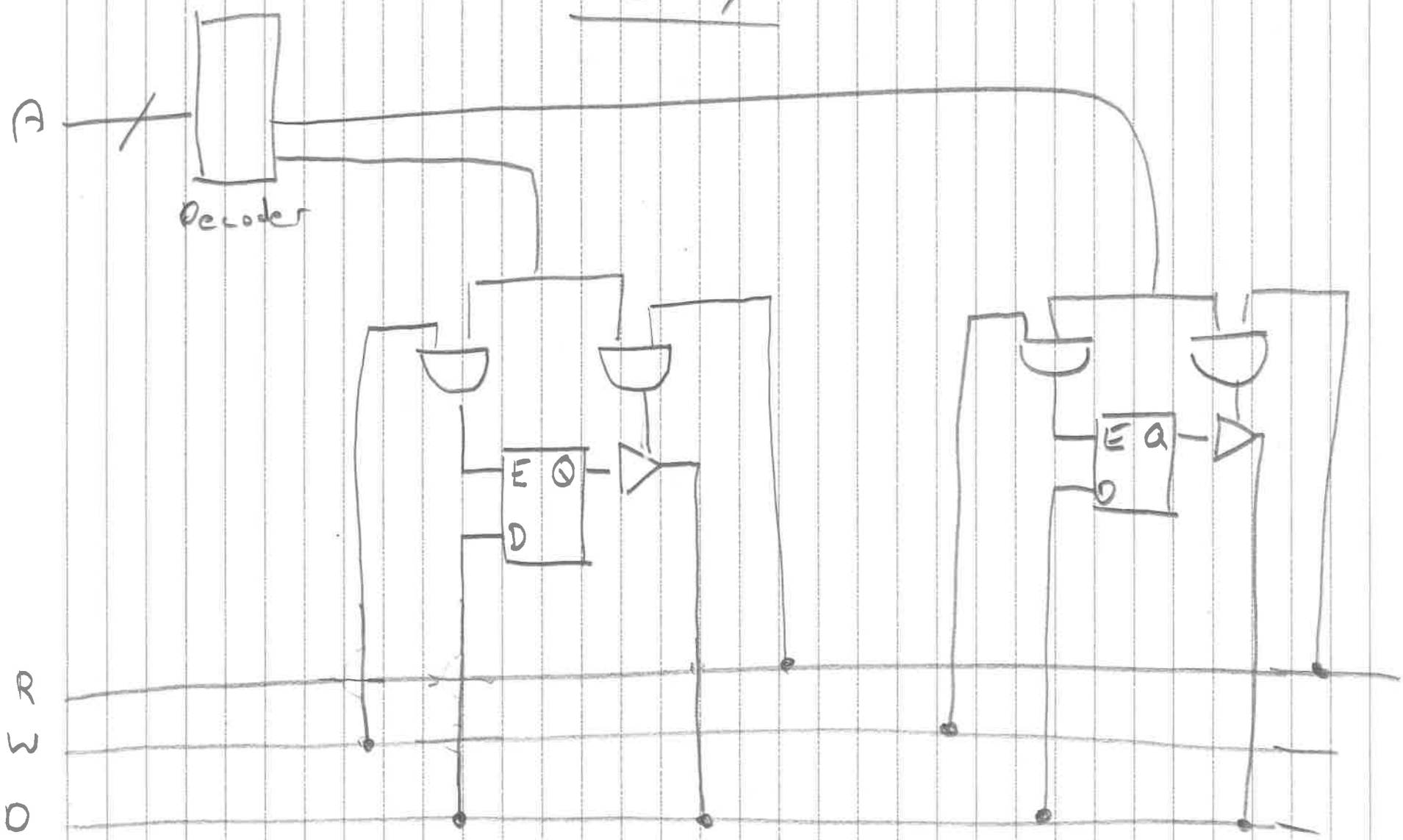
Note: $R = S = 0$ is not possible due to NOT at (*).

(Btw: that $\bigcirc = -\!\!\square\!\!\bigcirc$ for NAND construction... 5 NANDs per D ff)

Symbol

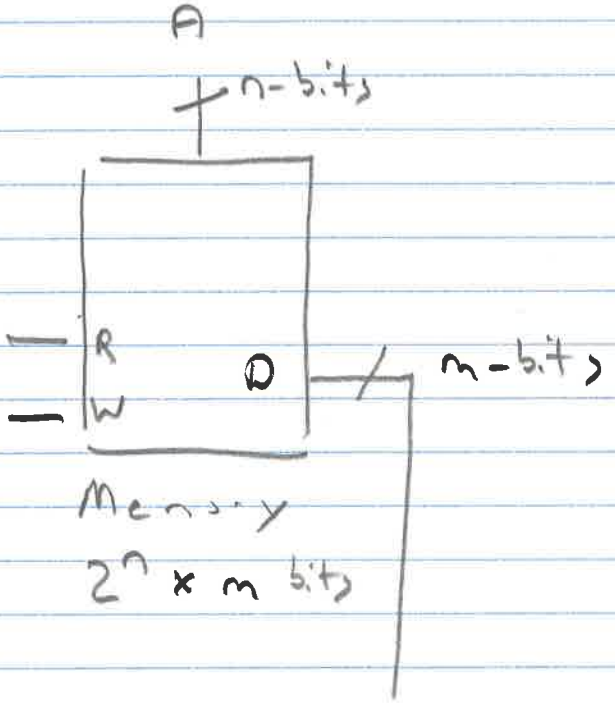


Memory

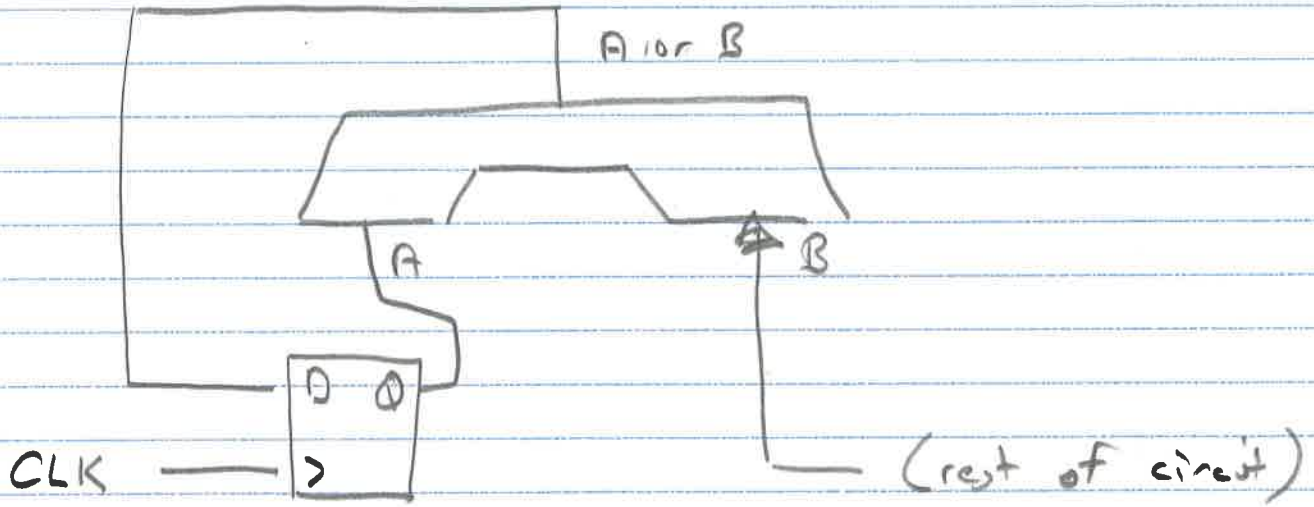


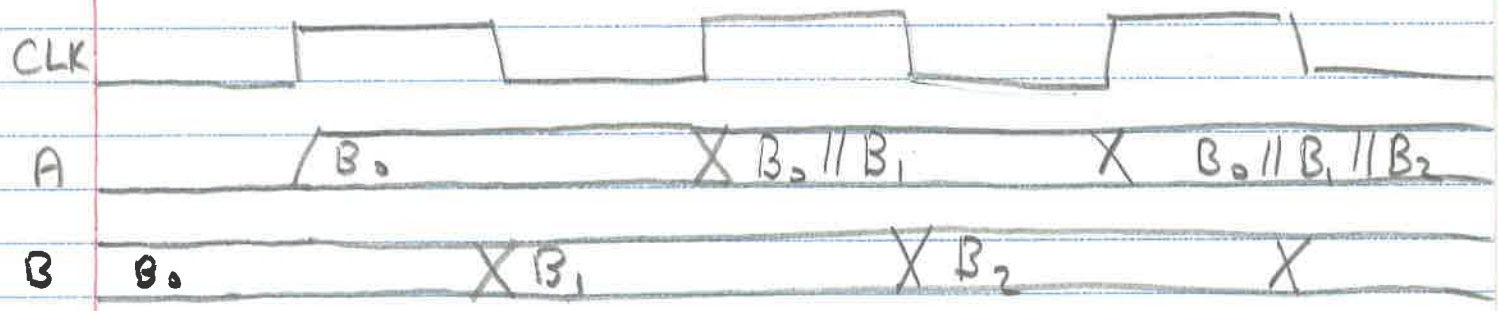
Any D-ff can be set with W, or read with R

Memory Symbol



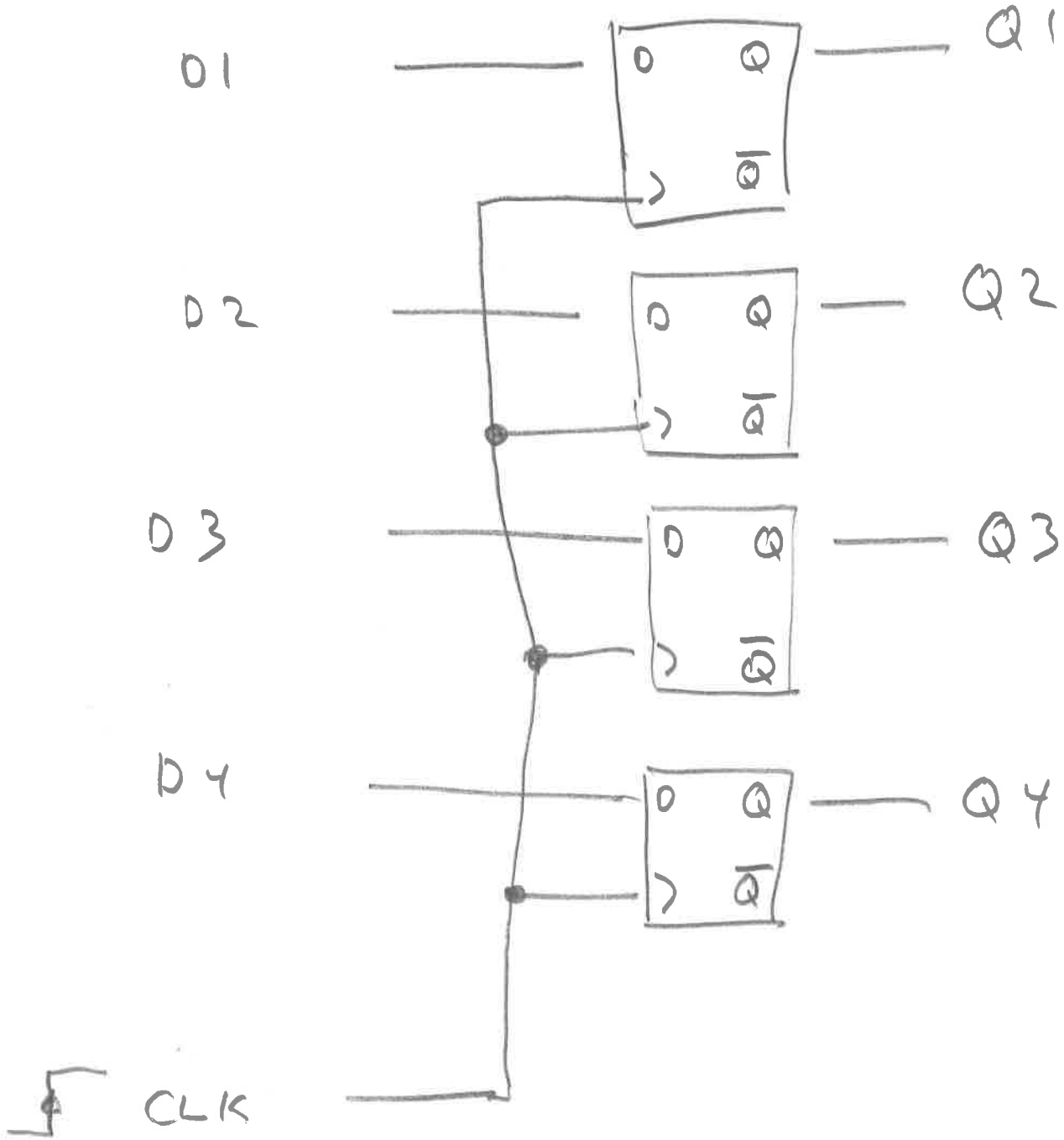
Synchronous Accumulator

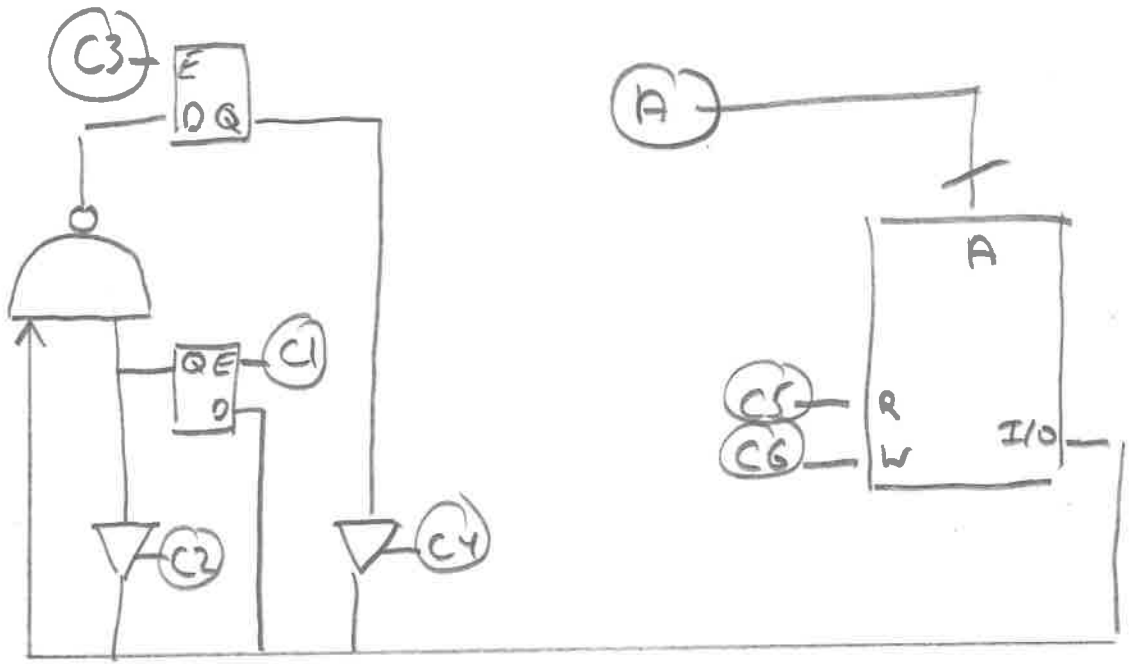




As long as we feed B $\frac{1}{2}$ cycle ahead, accumulator handles one new input per cycle.

Register

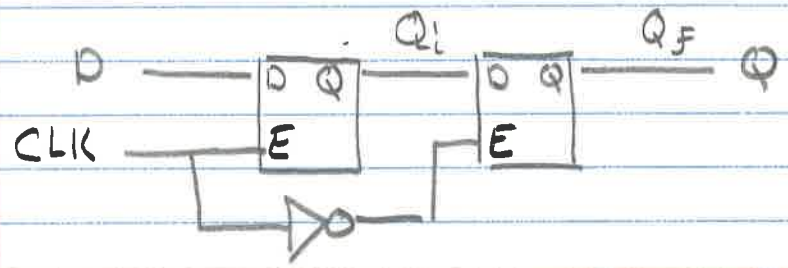


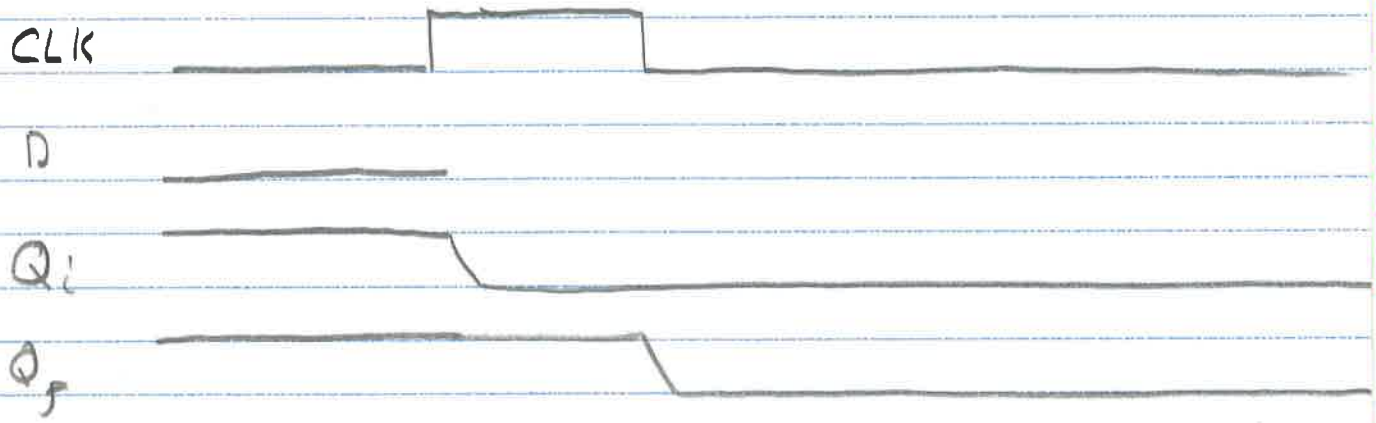


1-bit asynchronous NAND computer

	A	C1	C2	C3	C4	C5	C6	
	X	/	0	0	0	/	0	
LDA X	X	0	/	0	0	0	/	
STA X	X	0	0	/	0	/	0	(1)
NAND X	—	/	0	0	/	0	0	(2)

Edge-Triggered D-Flip flop

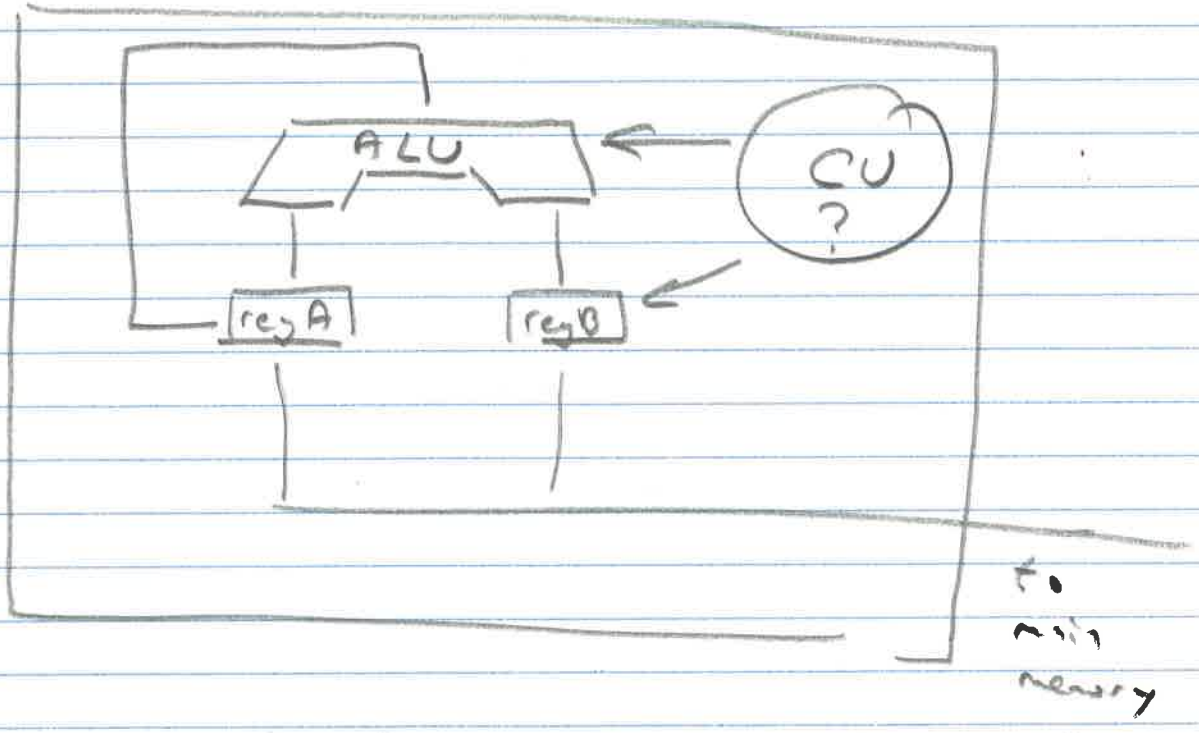




* Won't see new value until after falling edge, ie. Next Clock Cycle

Symbols

CPU: Central Processing Unit



By now understand workings of ALU for each assembly command as a fixed set of steps:

→ moving data into registers
→ digital synchronous logic

all driven by control signals (C_i)

Fixed Program Computer

Early computers applied a fixed set of steps to variable input data

C_1 0 0 1
C_2 0 1 0
⋮
C_N 0 0 1

} Known by Designer

R1 ? ? ?
R2 ? ? ?
R3 ? ? ?

} Computed at run time.

First computers had a hard-wired CU, no easy way to change it.

Turing / Von Neuman / Zuse

developed idea to

Store Instructions in Memory.

Incredibly powerful:

→ General Purpose Computer

New Algorithms Possible based on recursion (GOTO) and branching (conditional)

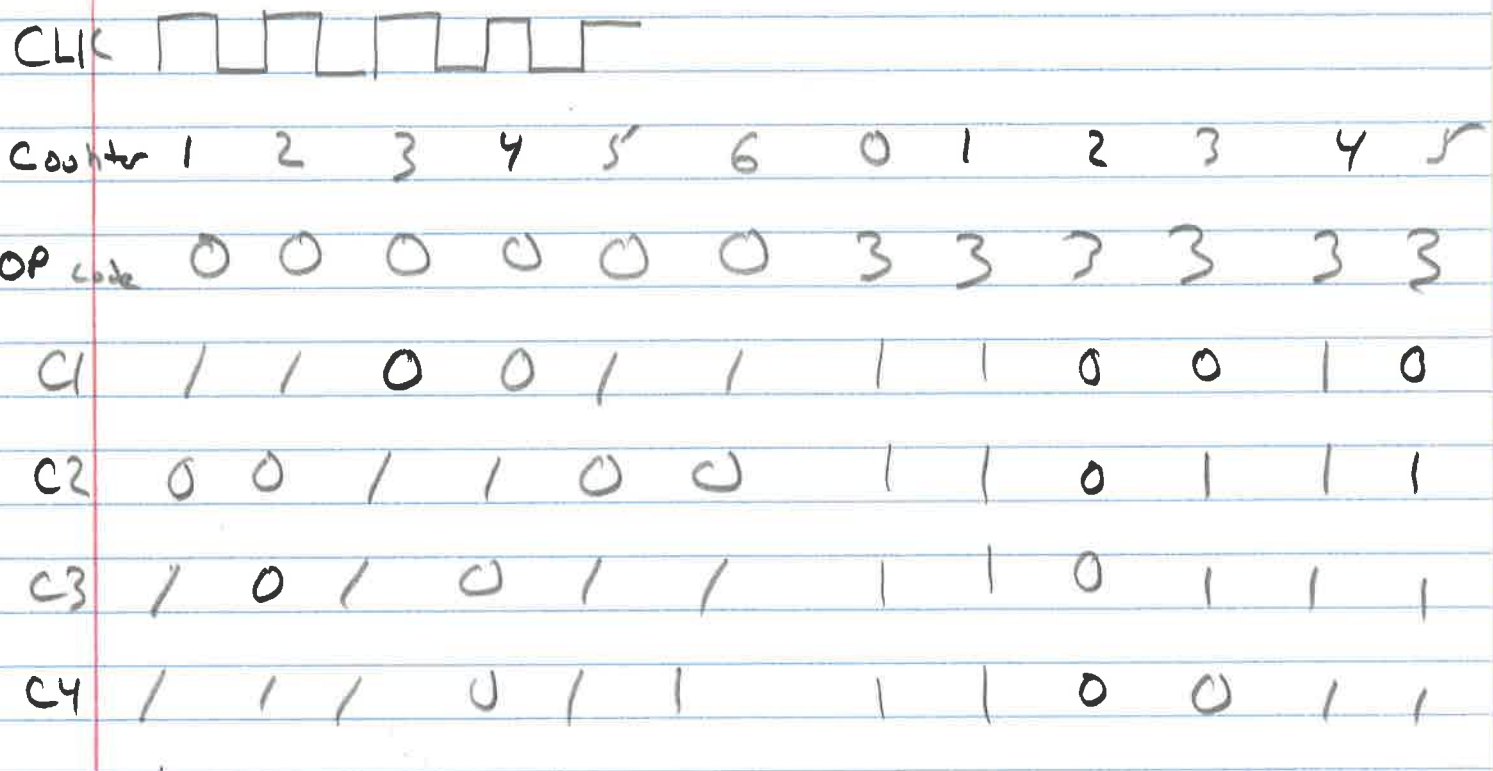
Control Unit

→ fetch instruction word from memory at current address in program counter (IR)

→ increment program counter (PC++)

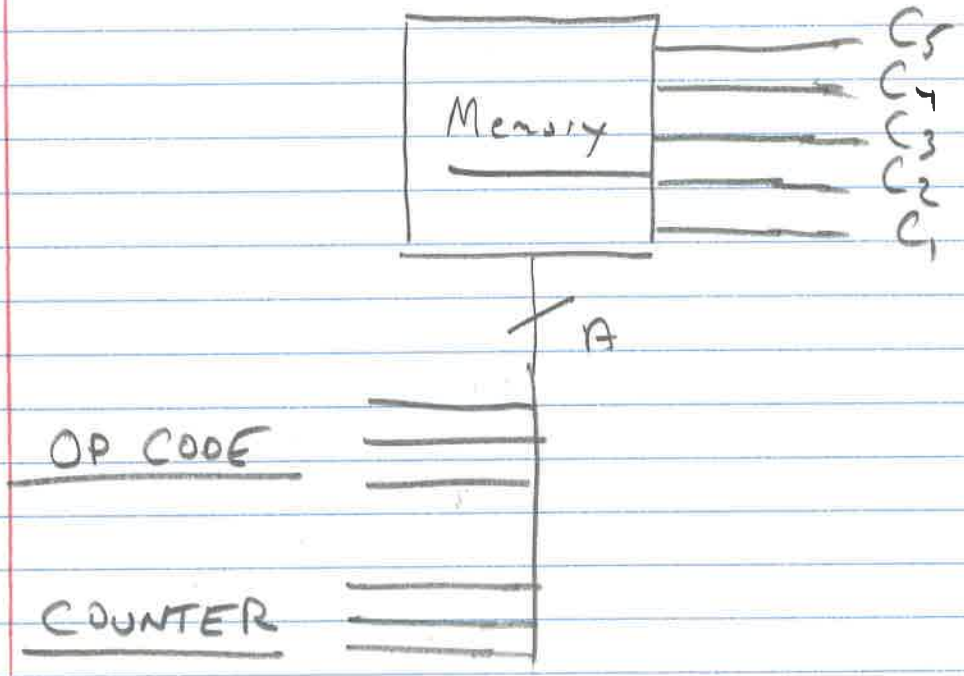
→ IR contains both OP code and address.

→ For each operation, there are a fixed set of steps that take place.



Key point: the control operations can, eg. change program counter, providing flow control, and conditional execution.

CU in ROM



That's really it....

except for literally Trillions of $ invested to improve performance!

Microsoft	—	$500 B
Apple	—	$700 B
Google	—	$574 B
Amazon	—	$373 B