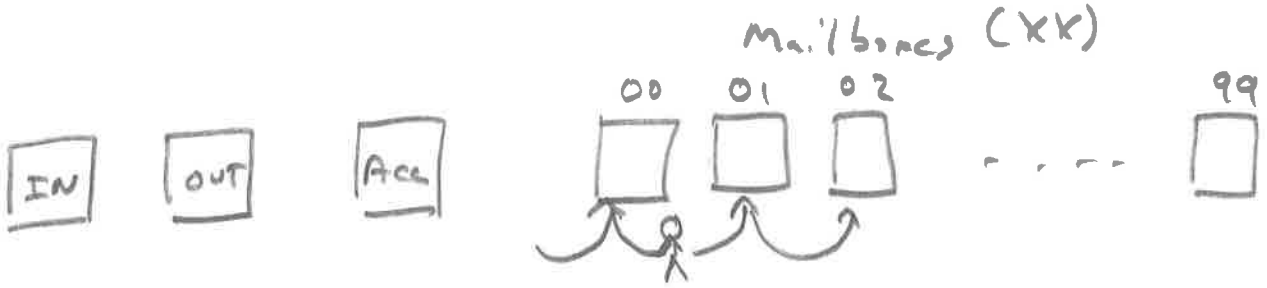
Microprocessors

and

Assembly

Little Man Computer



Input → user input

Output → eg. 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)

Mailboxes: 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	INP	load input to Acc
902	OUT	Send Acc to Out
000	COB	Coffee Break

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

Example : Adder (A+B)

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

Example Subtractor (A - B)

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

AND 

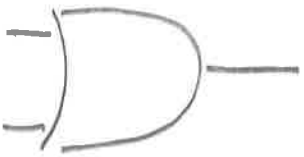
AB

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 

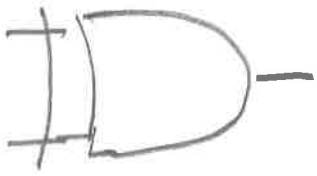
$A + B$

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

NOR 

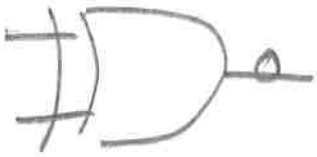
0	0	1
0	1	0
1	0	0
1	1	0

XOR



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

XNOR



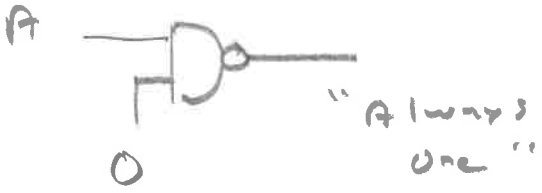
0	0	1
0	1	0
1	0	0
1	1	1

For the minimalists...

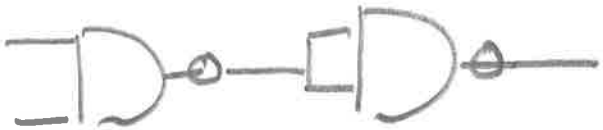
NAND and NOR are each functionally complete... you can construct all logic from enough of either.

Main tricks, invert:

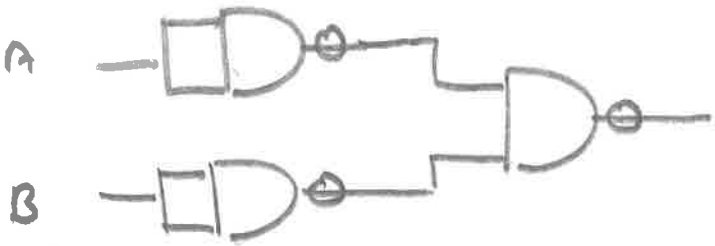
Ignore:



AND:

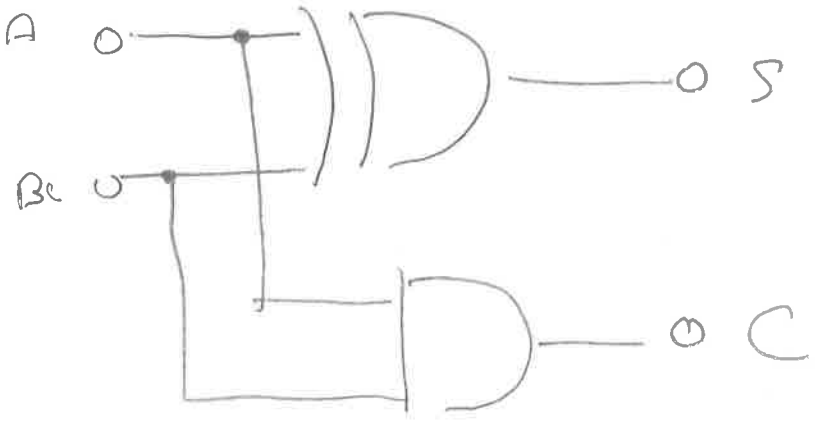


OR:

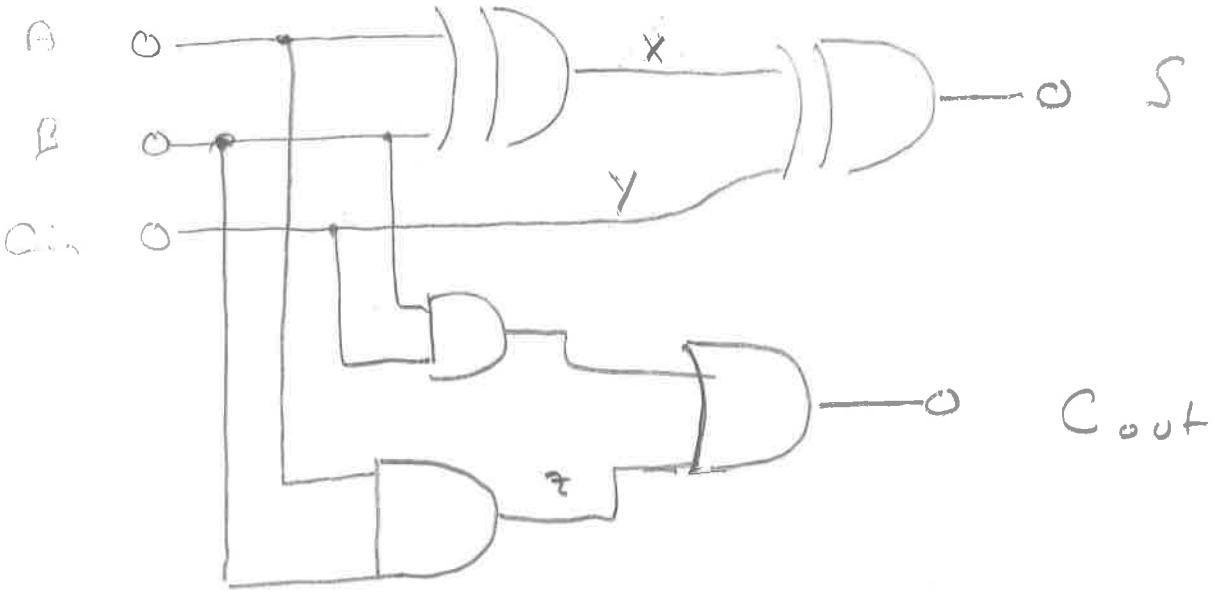


Half Adder

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

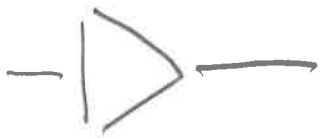


Full Adder



"Trick" to understanding is that
$1+1+(1) = 11 = 3$ is biggest number possible
$(\Leftrightarrow \quad 2 = 1 \quad \Rightarrow \quad x = 0)$

Buffer



A	O
0	0
1	1

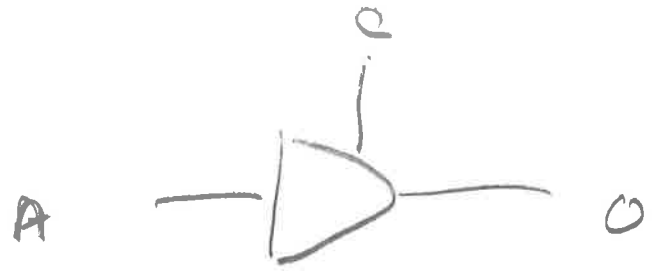
Inverter

A	O
1	0
0	1

Buffer from Inverter

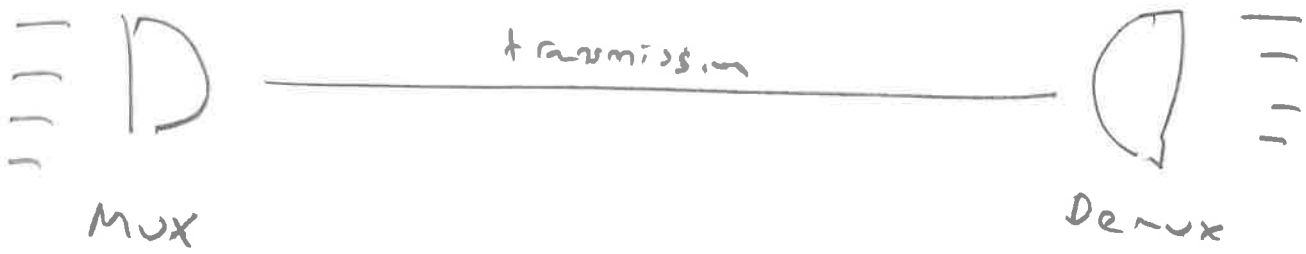
Inverter from Buffer

Tri-State Buffer :

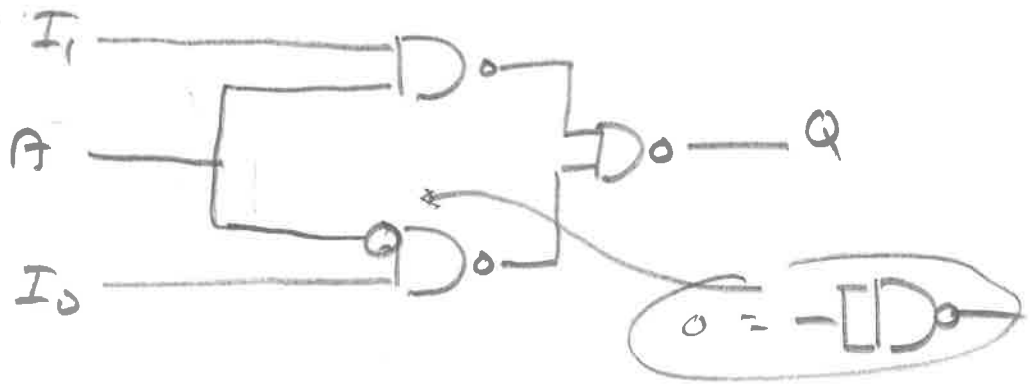


A	C	O
0	0	Z
1	0	Z
0	1	0
1	1	1

Multiplexing

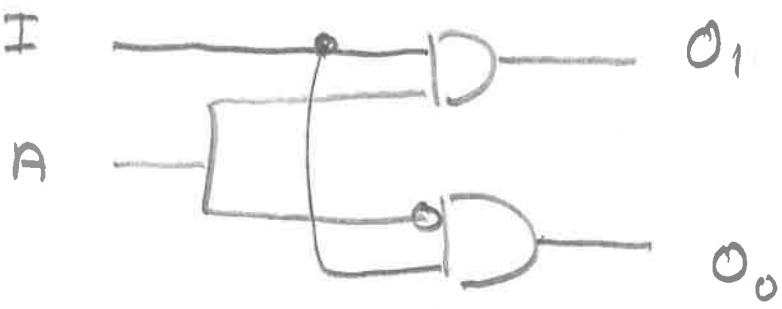


1-bit Mux



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

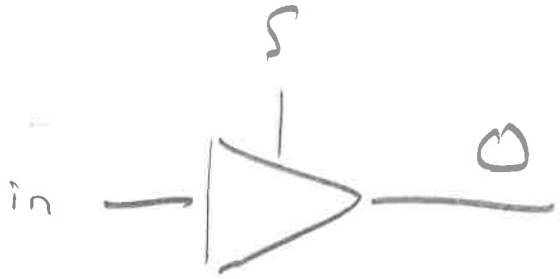
A	B	S	O
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

S	O
0	A
1	B

DEMUX

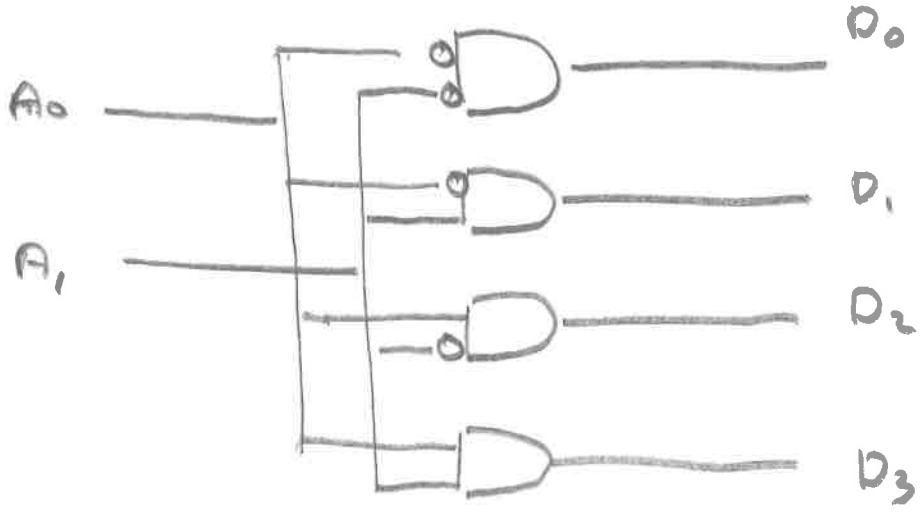
A	S	C	D
0	0	0	0
1	0	1	0
0	1	0	0
1	1	0	1

S	C	D
0	in	0
1	0	in



S	O
0	Z
1	in

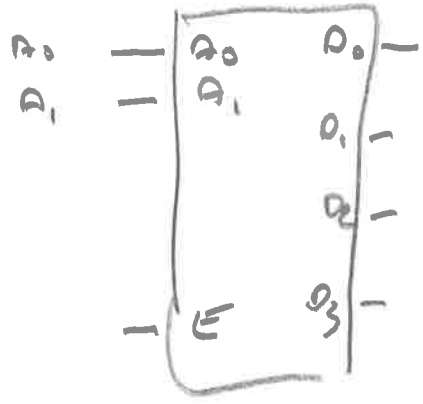
De coder

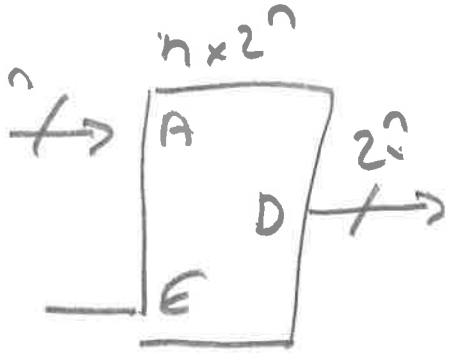


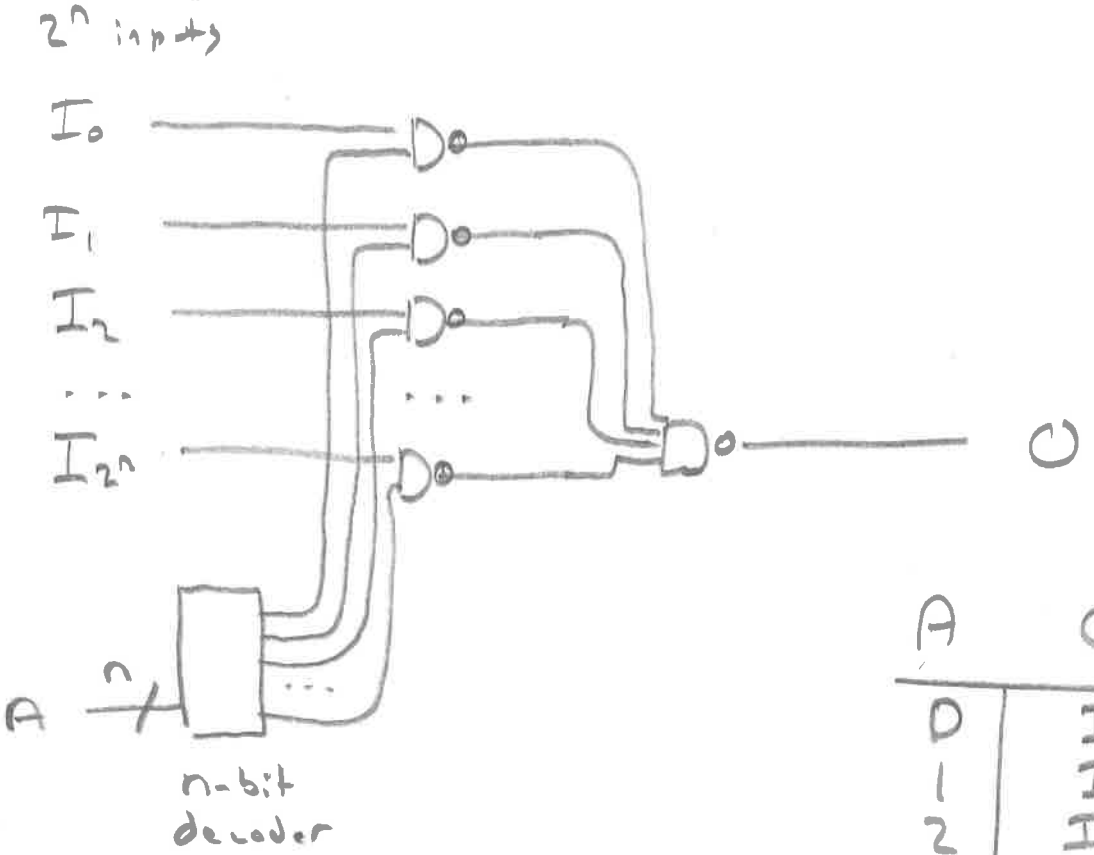
3-bit:

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

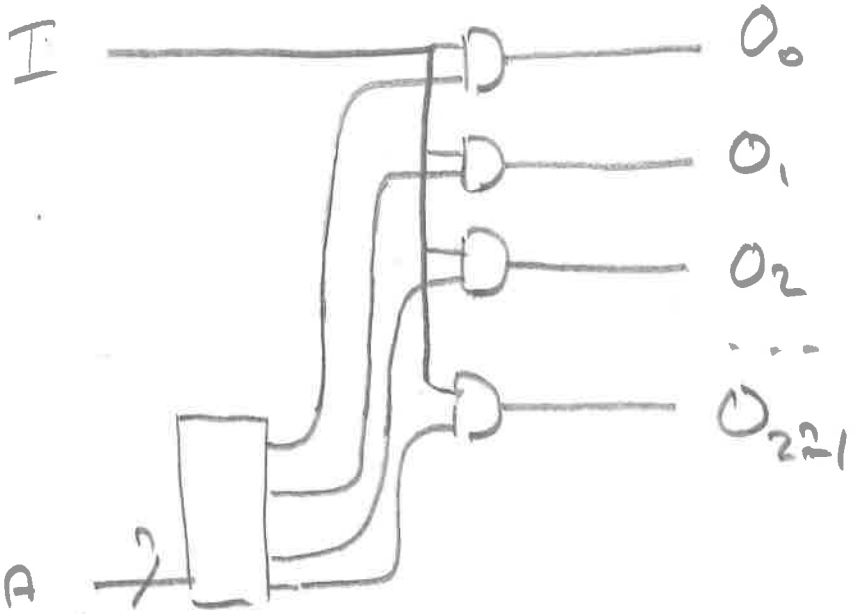
2x4 decoder







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

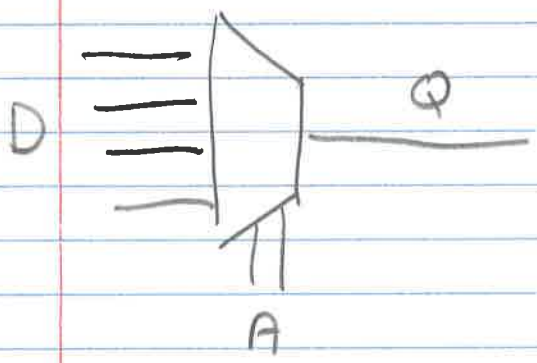
Number of outputs $N_o \geq 1$

Number of address bits: N_A

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

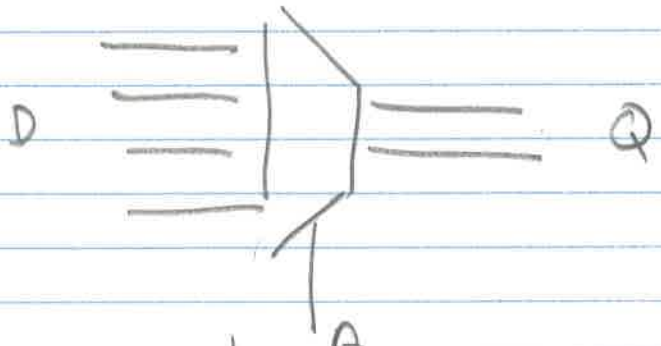
eg.

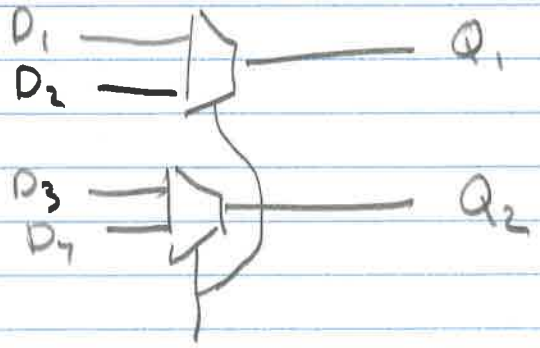
4:1



$2^2 = 4/1$

4:2





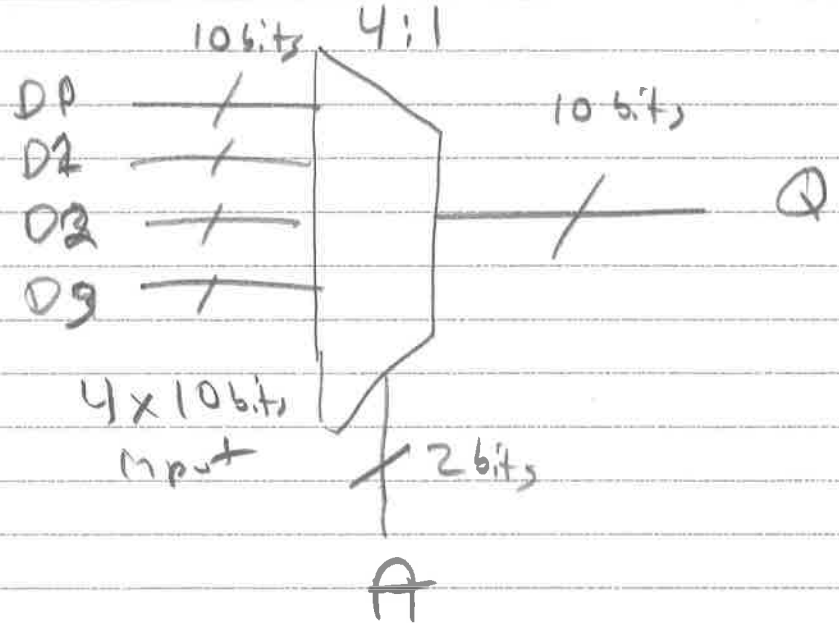
$2^1 = 4/2$

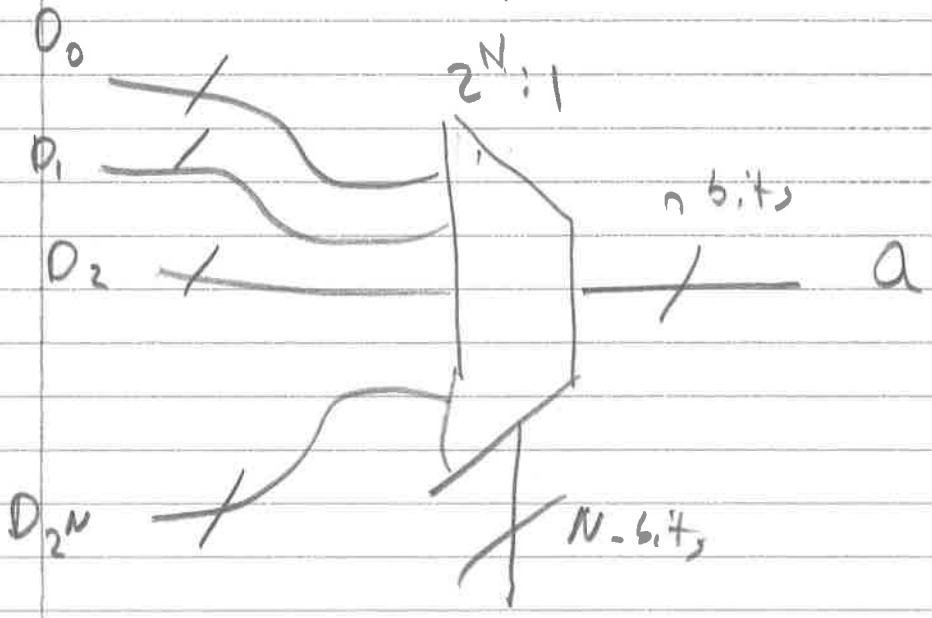
With:

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

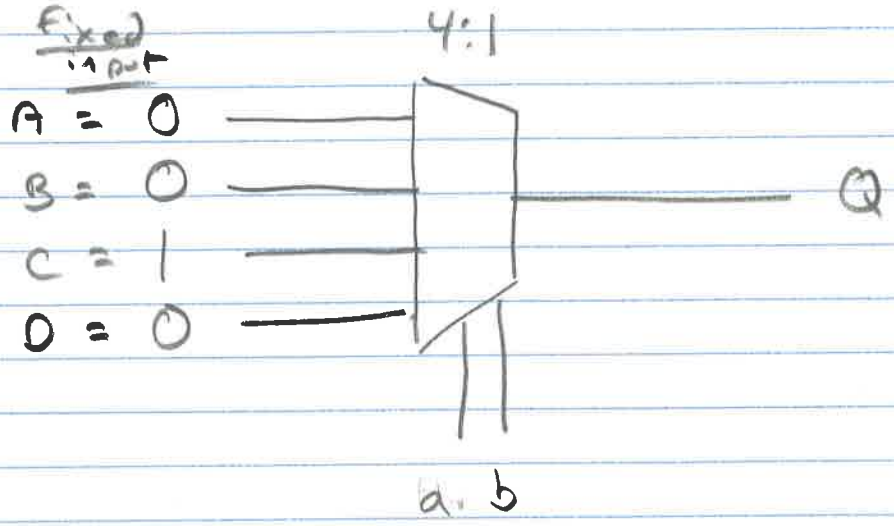
(Accumulator in LMC requires 10 bits ≤ 1024)
and Mailboxes

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





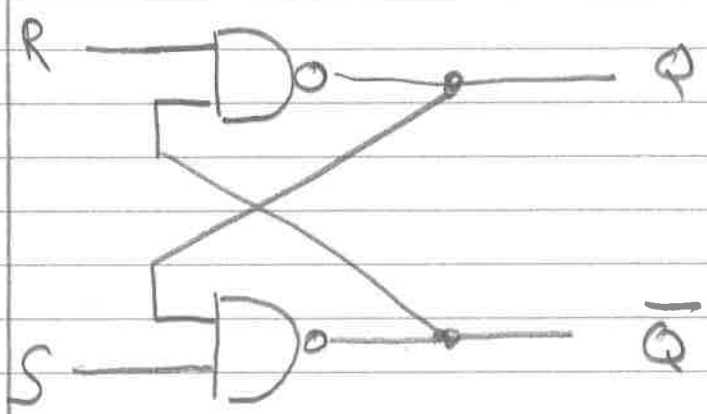
Mux as Generic Logic



a	b	Q	
0	0	A	Set to what we want
0	1	B	
1	0	C	
1	1	D	

Configurable Logic!

RS latch



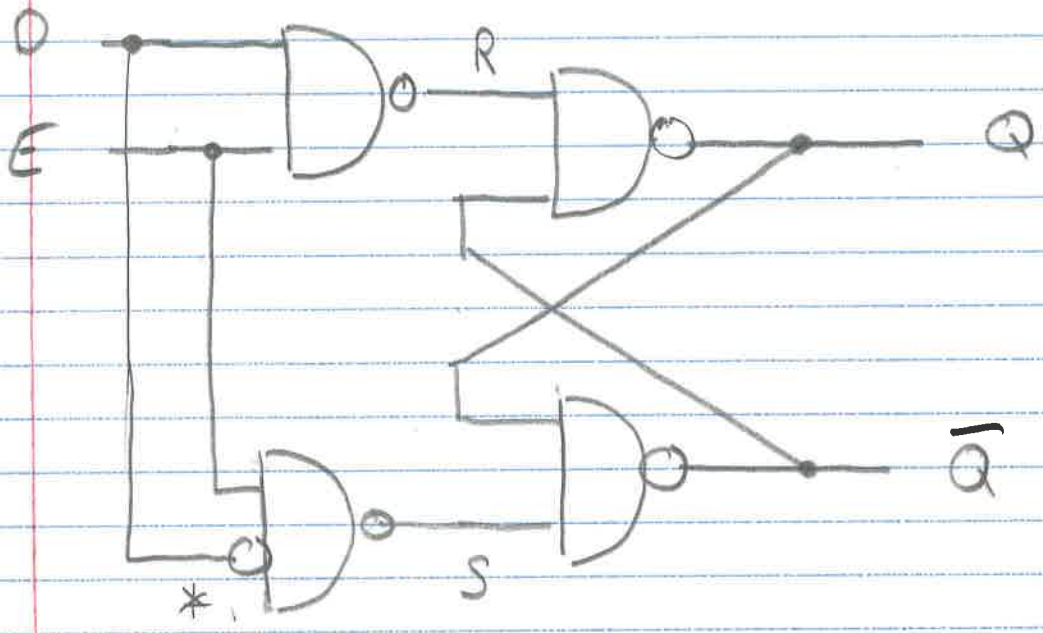
RS both high $Q=0$ $\overline{Q}=1$ stable
$Q=1$ $\overline{Q}=0$ stable
($Q=\overline{Q}$ not stable)

$R=1$ $S=0$, $S=0$ 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



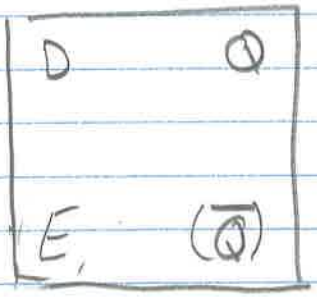
$E = 0 \Rightarrow R = S = 1 \Rightarrow Q$ constant

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

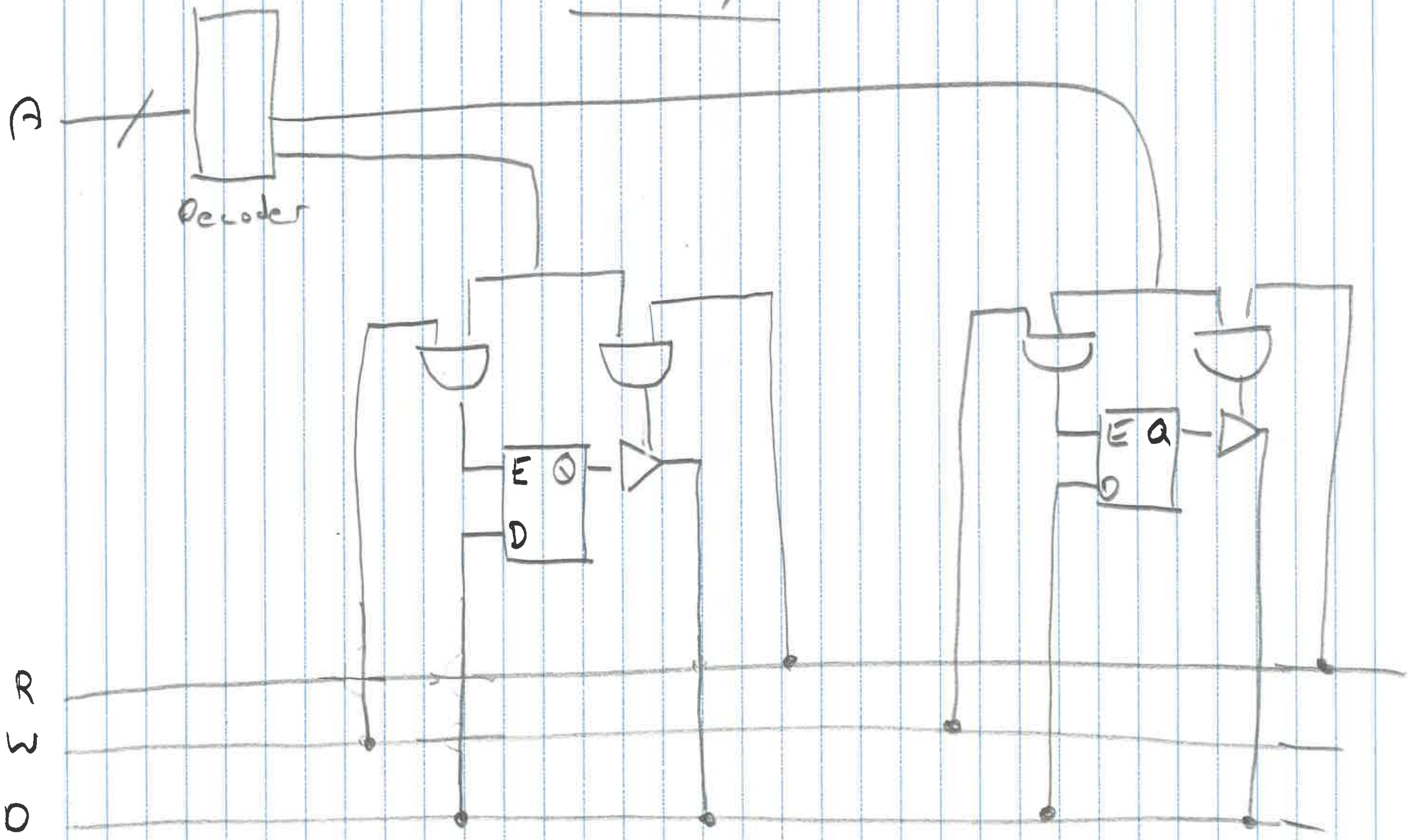
Note: $R = S = 0$ is not possible due to NOT at (*).

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

Symbol

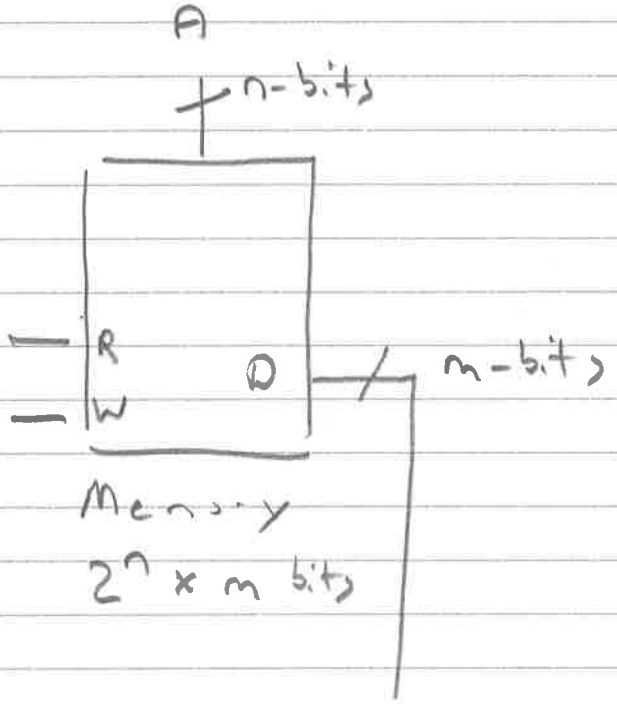


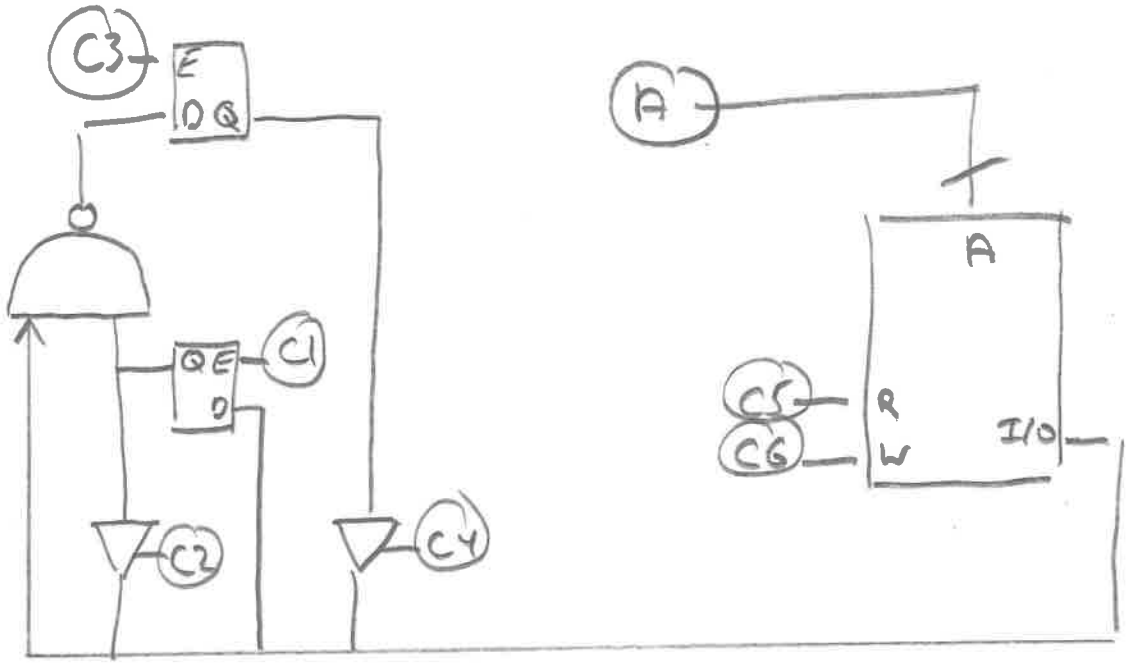
Memory



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

Memory Symbol

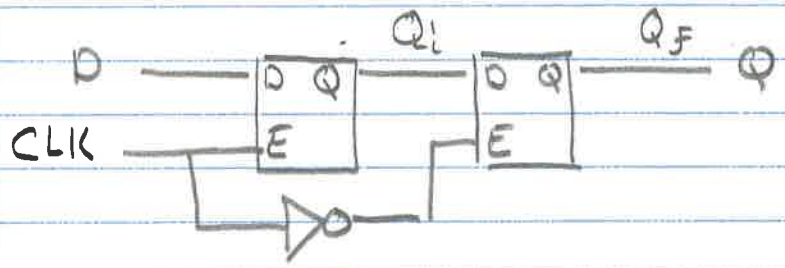


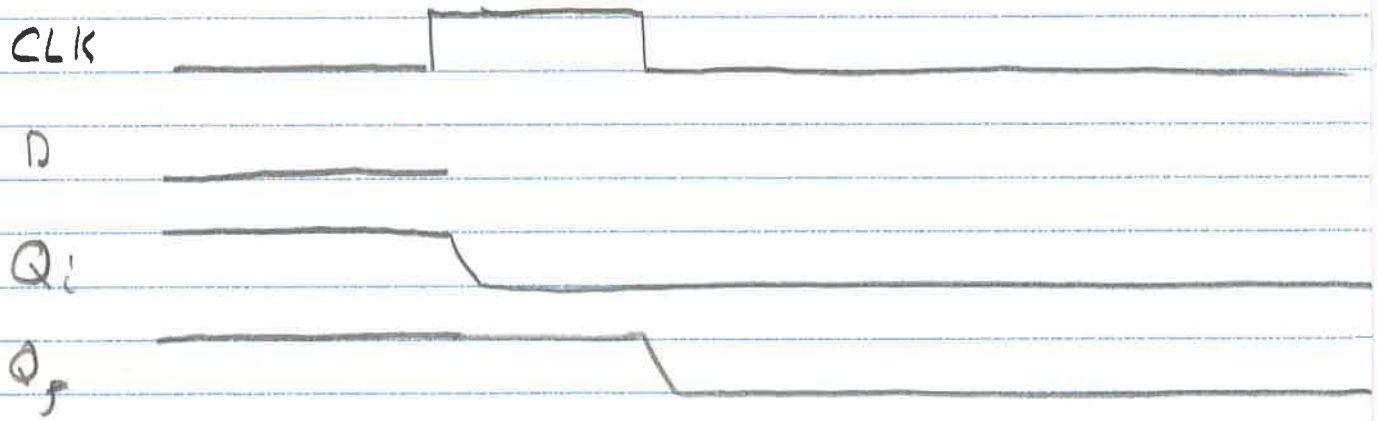


1-bit asynchronous NAND computer

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

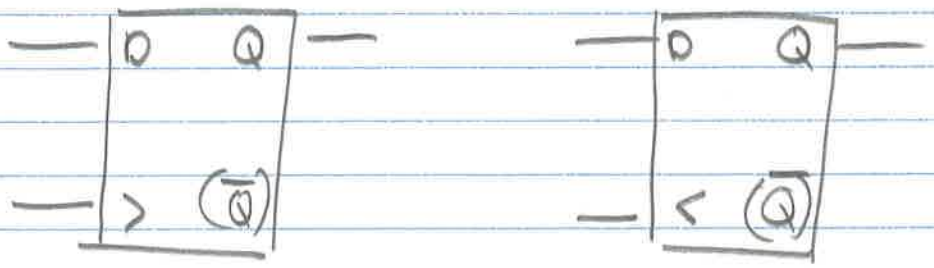
Edge - Triggered D - flip flop



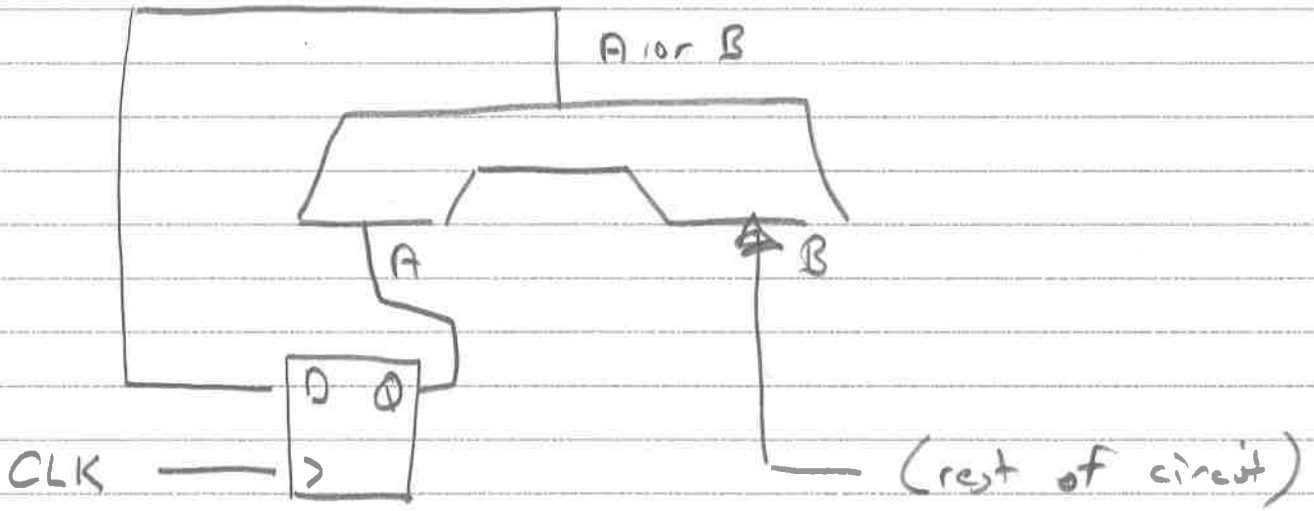


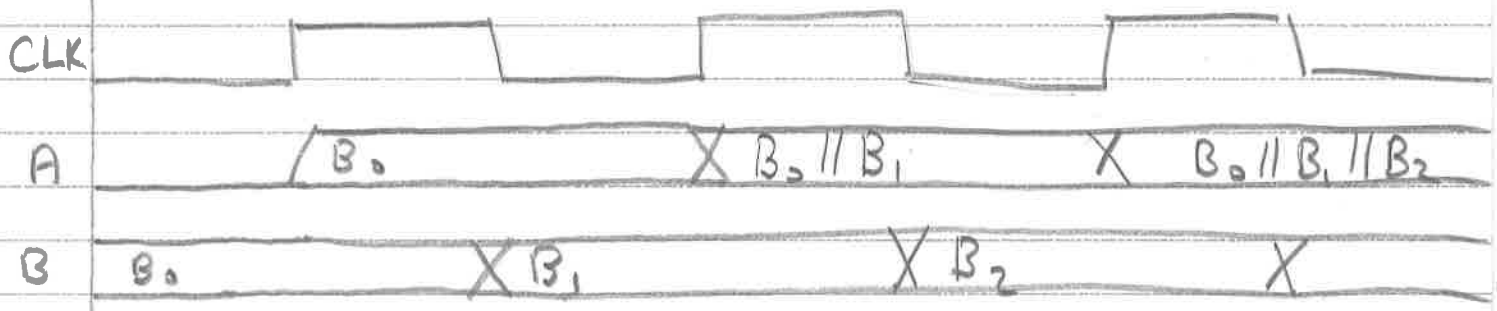
* Won't see new value until after falling edge, ie, Next Clock Cycle

Symbols



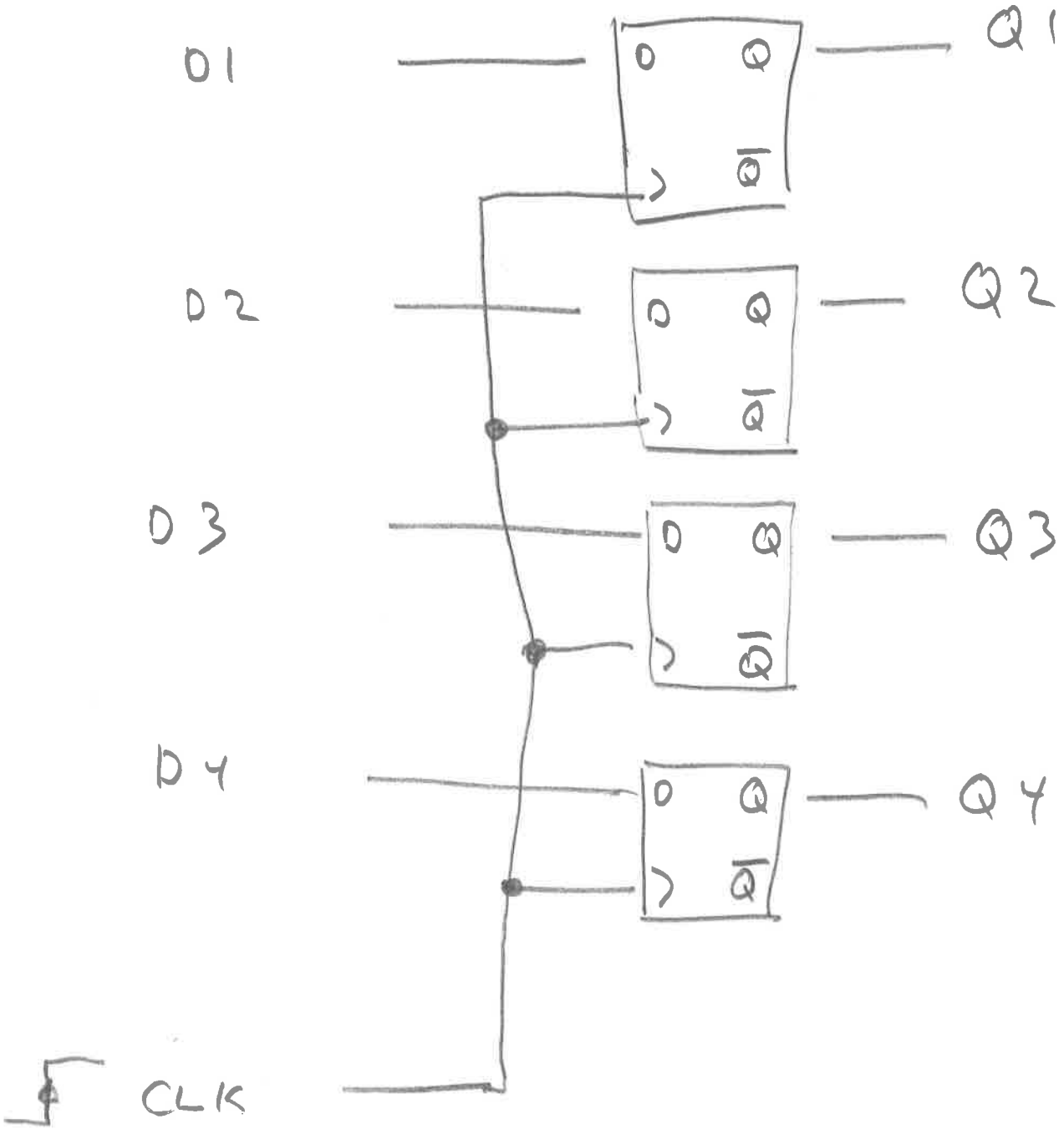
Synchronous Accumulator



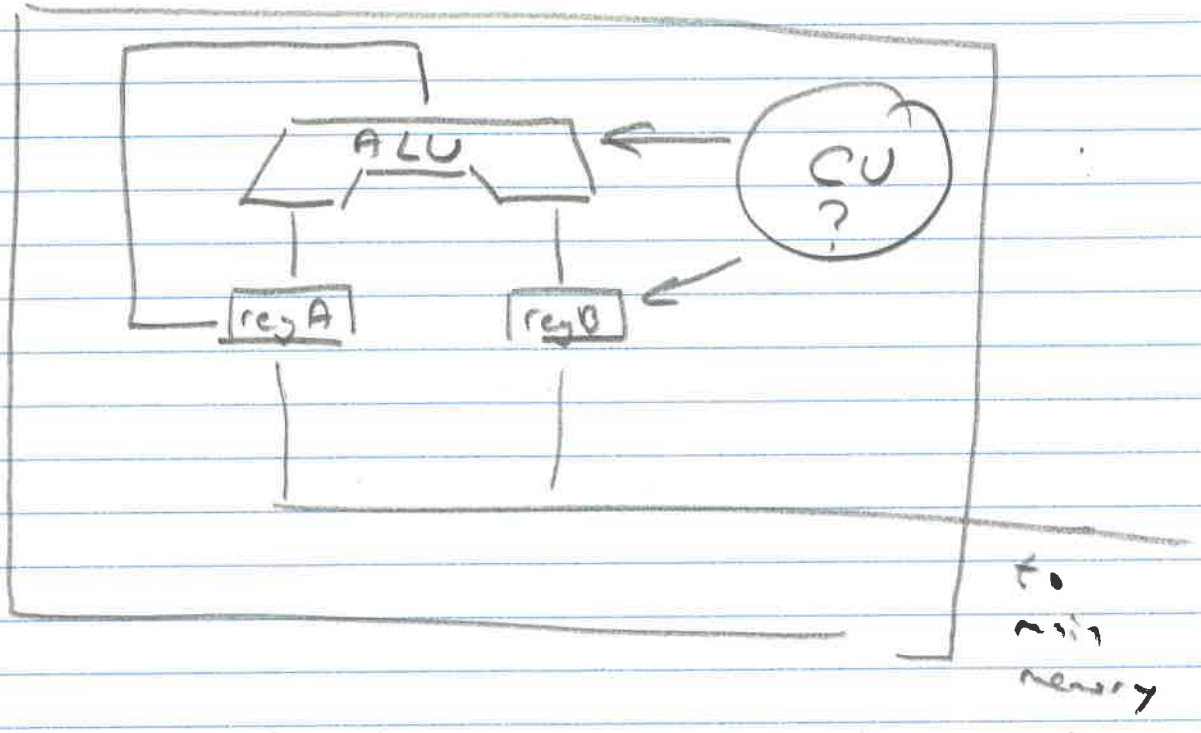


As long as we feed B $\frac{1}{2}$ cycle ahead, accumulator handles one new input per cycle.

Register



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	} Known by Designer
C_2	0	1	0	
⋮				
C_N	0	0	1	
R1	?	?	?	} Computed at run time.
R2	?	?	?	
R3	?	?	?	

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

Turing / Von Neuman / Zuse

developed idea to

Store Instructions in Memory,

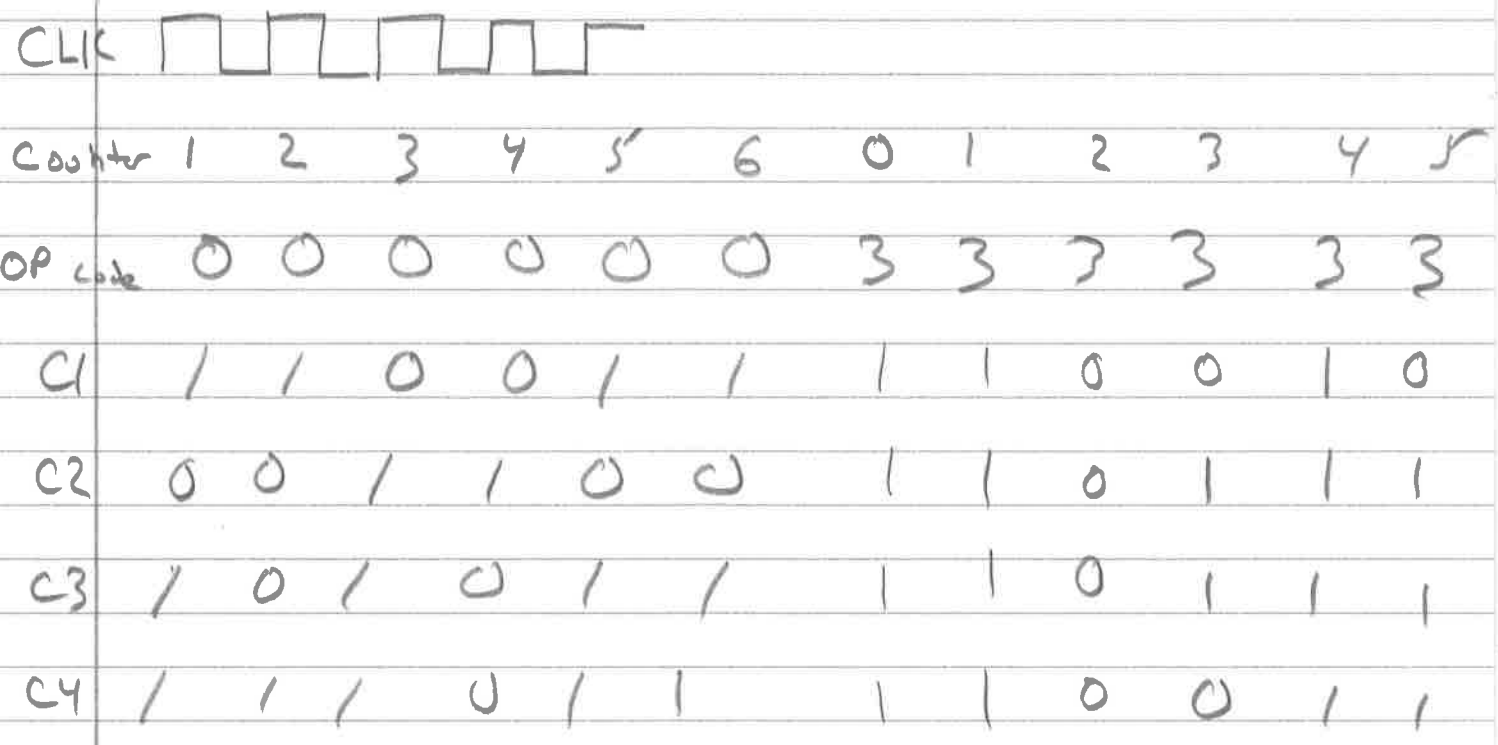
Incredibly powerful:

→ General Purpose Computer

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

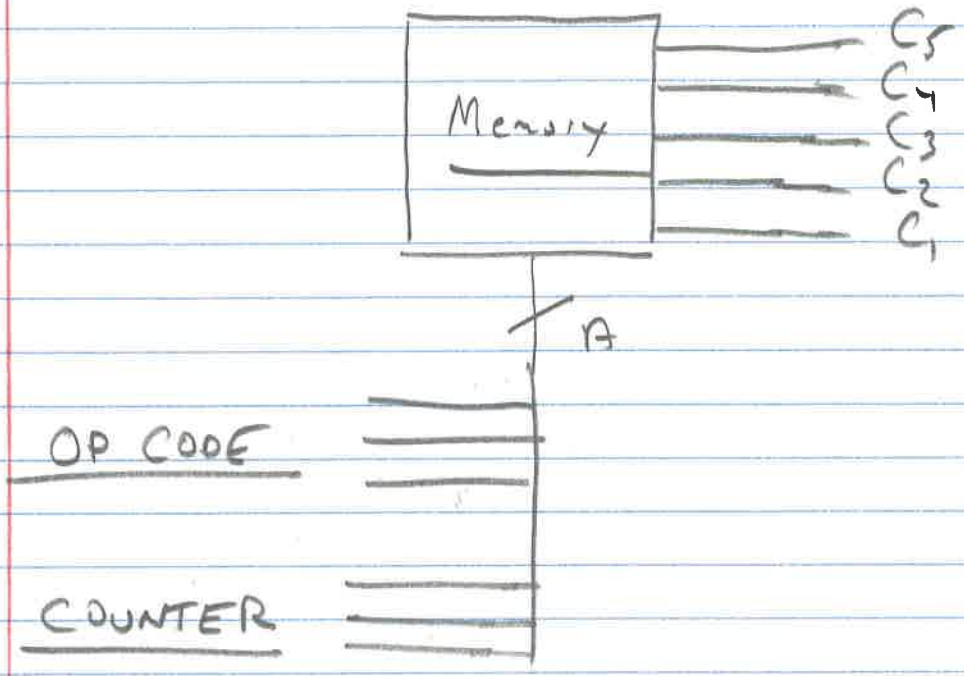
Control Unit

→ 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 ≈ ROM



That's really it....

except for literally Trillions of $ invested to improve performance!

Microsoft	—	$500 B
Apple	—	$700 B
Google	—	$534 B
Amazon	—	$373 B

Python:

→ Very different (complementary) to C/C++

→ dynamic type checking
(no int x = 2)

→ automatic memory management
(no new/delete, no pointers)

→ readability → indentation used instead of {}?

Great resource at:

www.python.org

For example, tutorials:

http://developers.google.com/edu/python/

Intro

Strings (& unicode)

Lists

(numpy, matplotlib)

Interpreter or Command Line

→ Show Calculator in interactive mode

→ print 'hello world'

As script.py

Subroutines:

```
def f(x, y)
```

Conditionals: if / elif / else

while / break

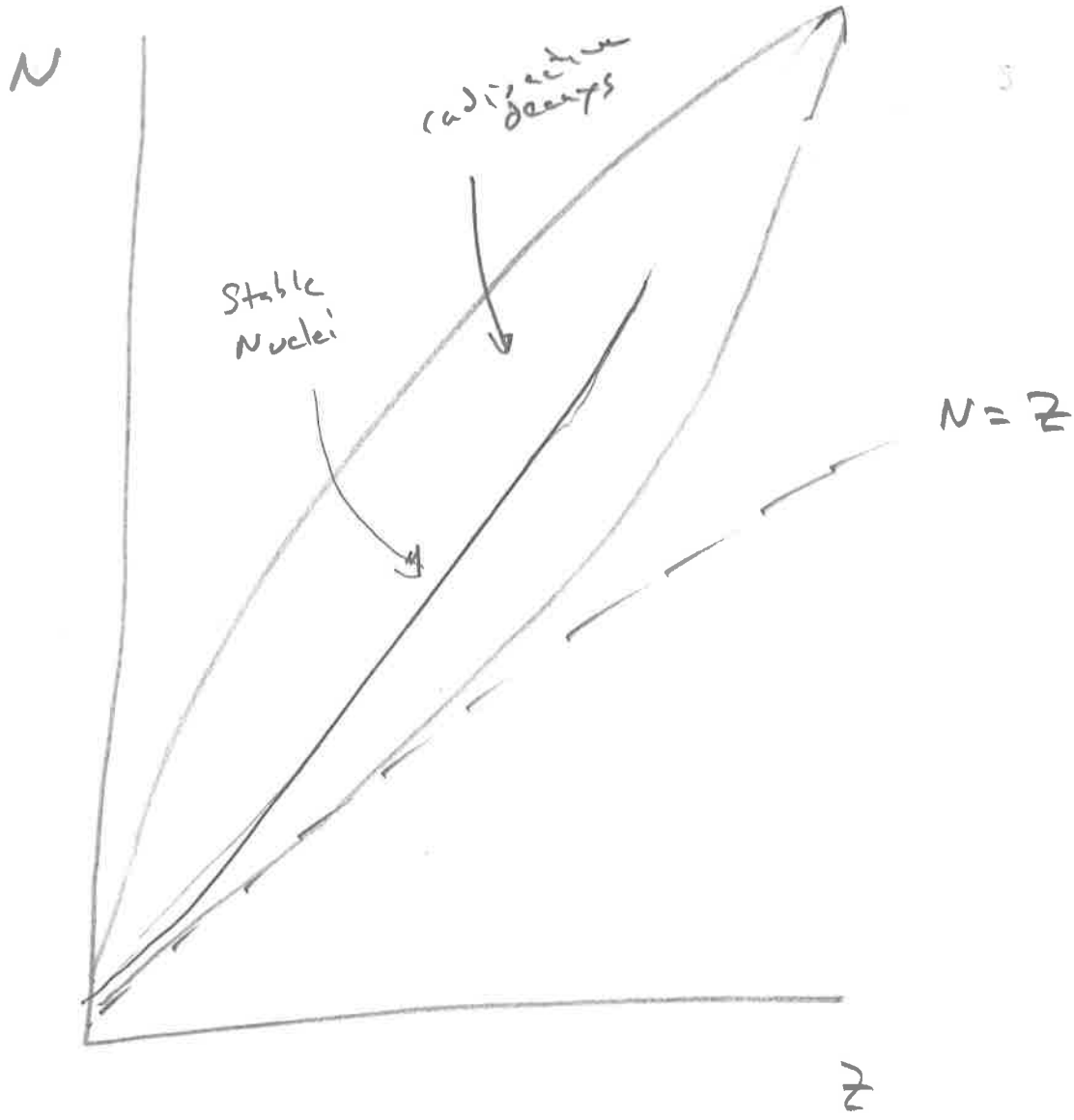
For: for x in [1, 2, 3]

for y in range(10)

Lists:

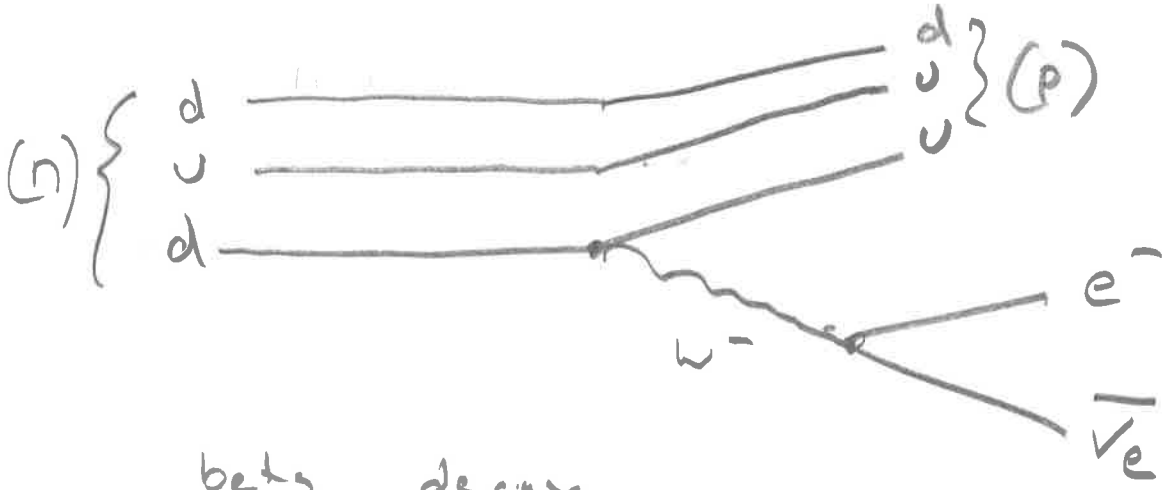
```
x = []
x.append(5)
x.append(2)
print len(x)
```

Radioactive Decay



After Hydrogen, need a mix of p/n to reach stable nuclei...

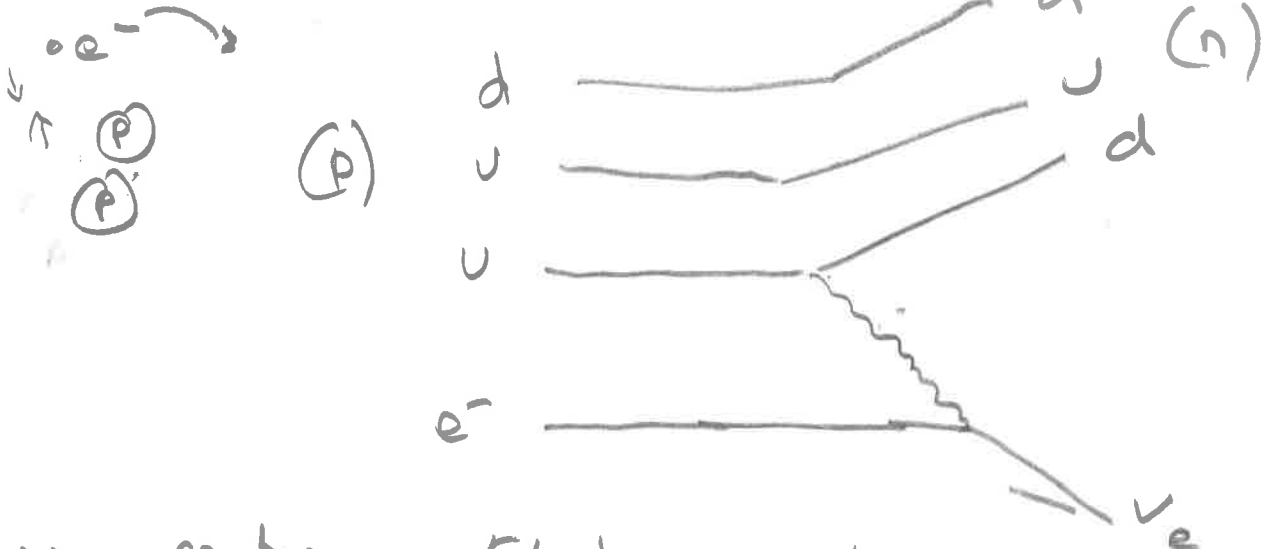
→ n are unstable (outside of nuclei)



beta decay...

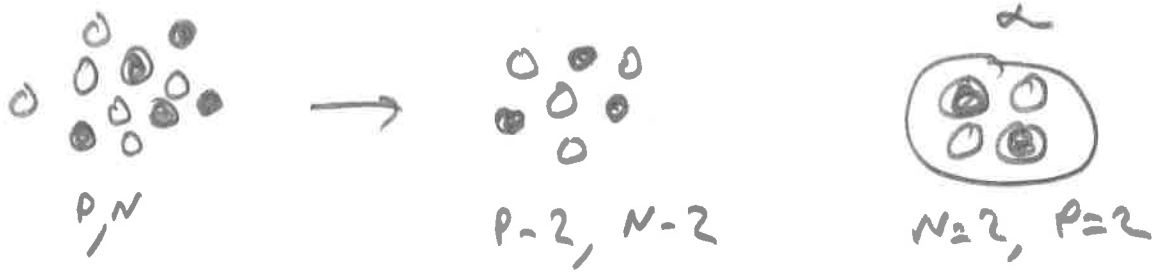
Too many neutrons, beta decay (B^-)

→ p are stable outside nuclei, but too many, Coulomb forces dominates



Too many protons, Electron capture (B^+)

Too many protons or neutrons:



which is really just special case instance of nuclear fission:

Other (rarer) decay processes:

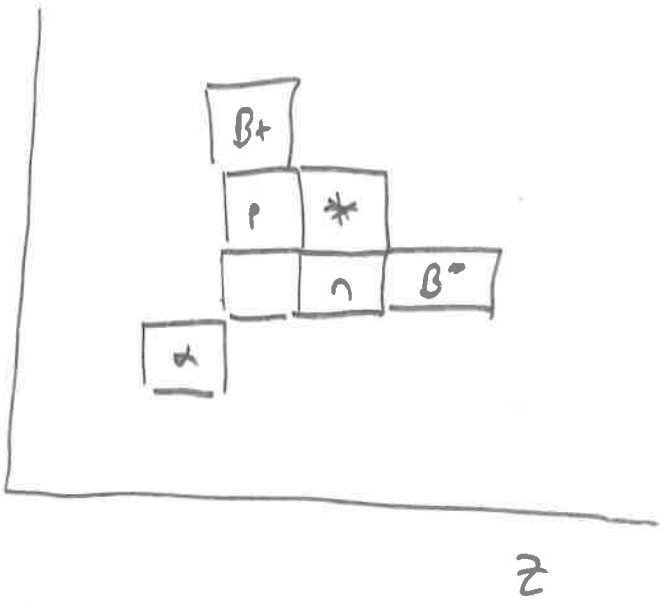
→ proton emission

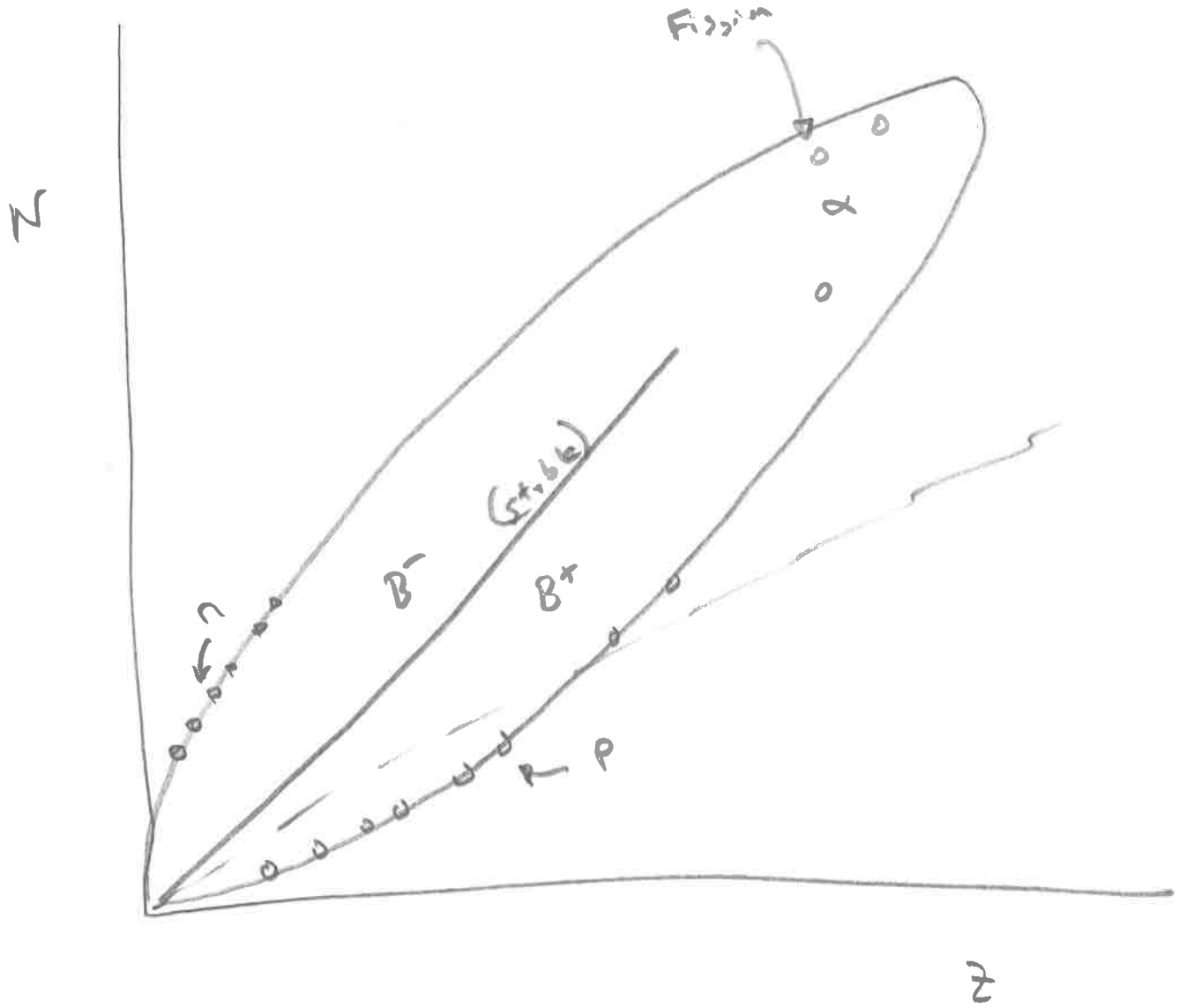
→ neutron emission

Excess p or n simply ejected.

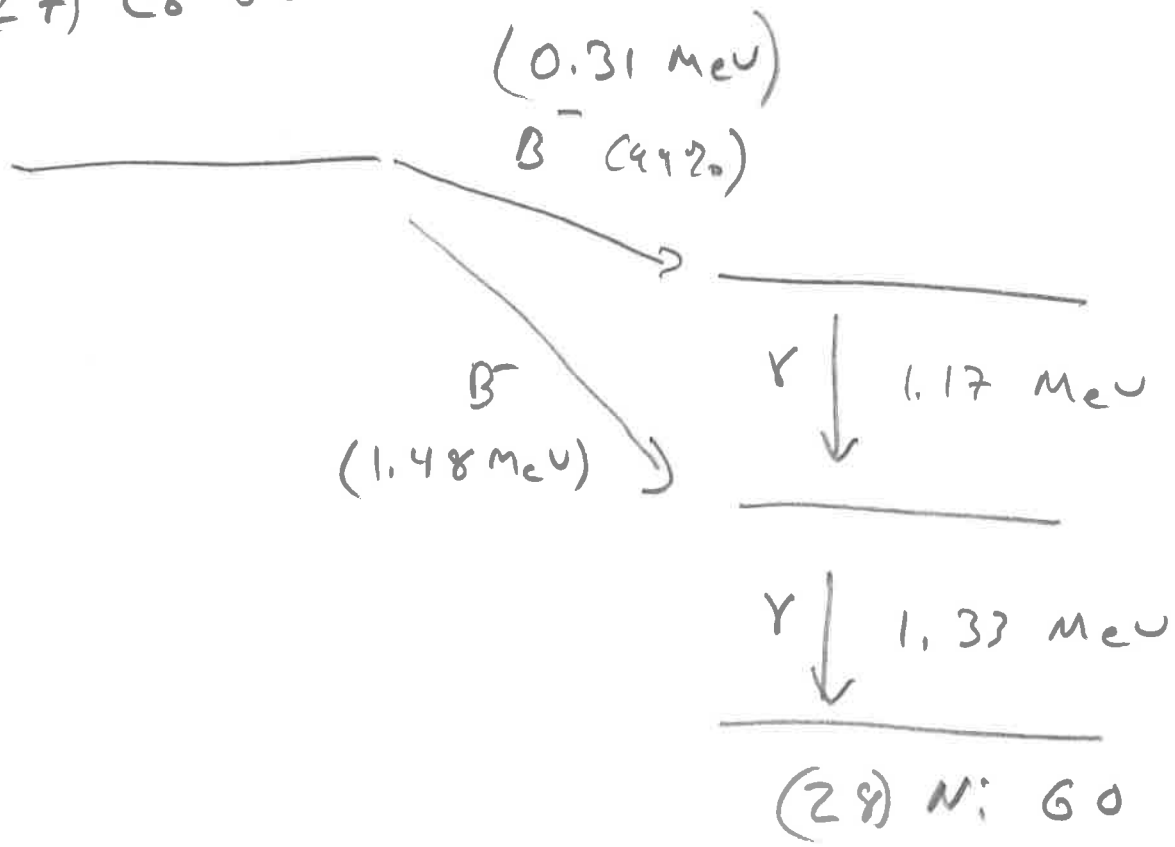
→ neutron emission is common as result of fission (extra neutrons!)

→ proton emission rare, but interesting experimentally (quantum tunelling)


N
Z
B+
p
n
B-
α


N
Z
Fission
α
B-
(Stable)
B+
n
p
R

(27) Co 60



Radio-active Decay

→ Totally random which (when a particular) nucleus decays. (unlike humans!)

→ Equally probable that any nucleus will decay at any time (unlike humans)

(Nuclei don't get old and die!)

$$\Rightarrow \quad \frac{dN}{dt} = -kN$$

$$N(t) = N \exp(-kt)$$

$$N(0) = N_0$$

$$N(T_{1/2}) = N_0/2$$

$$N(t) = N_0 \exp(-kt)$$

$$N(T_{1/2}) = N_0 \exp(-kT_{1/2}) = N_0/2$$

$$\Rightarrow \quad 2 = \exp(kT_{1/2})$$

$$\log 2 = kT_{1/2}$$

$$k = \frac{1}{\log 2 \, T_{1/2}}$$

(Fix)

$$N(t) = N_0 \exp\left(\frac{-t}{\log(2)\, T_{1/2}}\right)$$

Passage of charged particles:

As charged particles move through medium, loose energy in a number of ways.

→ Ionize atoms by liberating electrons

→ Multible Scattering, from nuclei

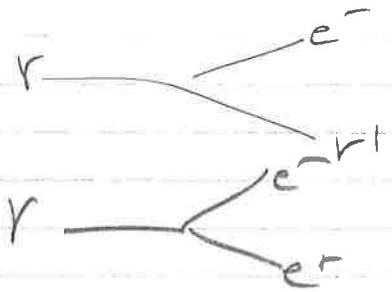
→ For light particles acceleration causes radiation (bremsstrahlung)

Photons

Energy ↓

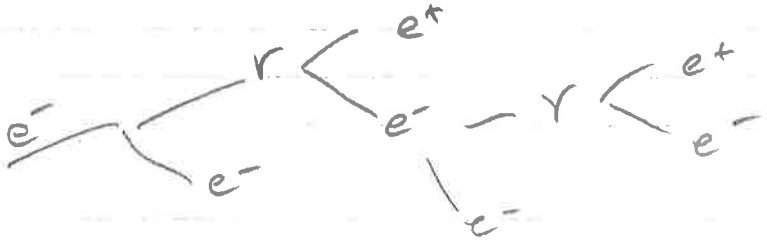
(a) Photo-electric effect

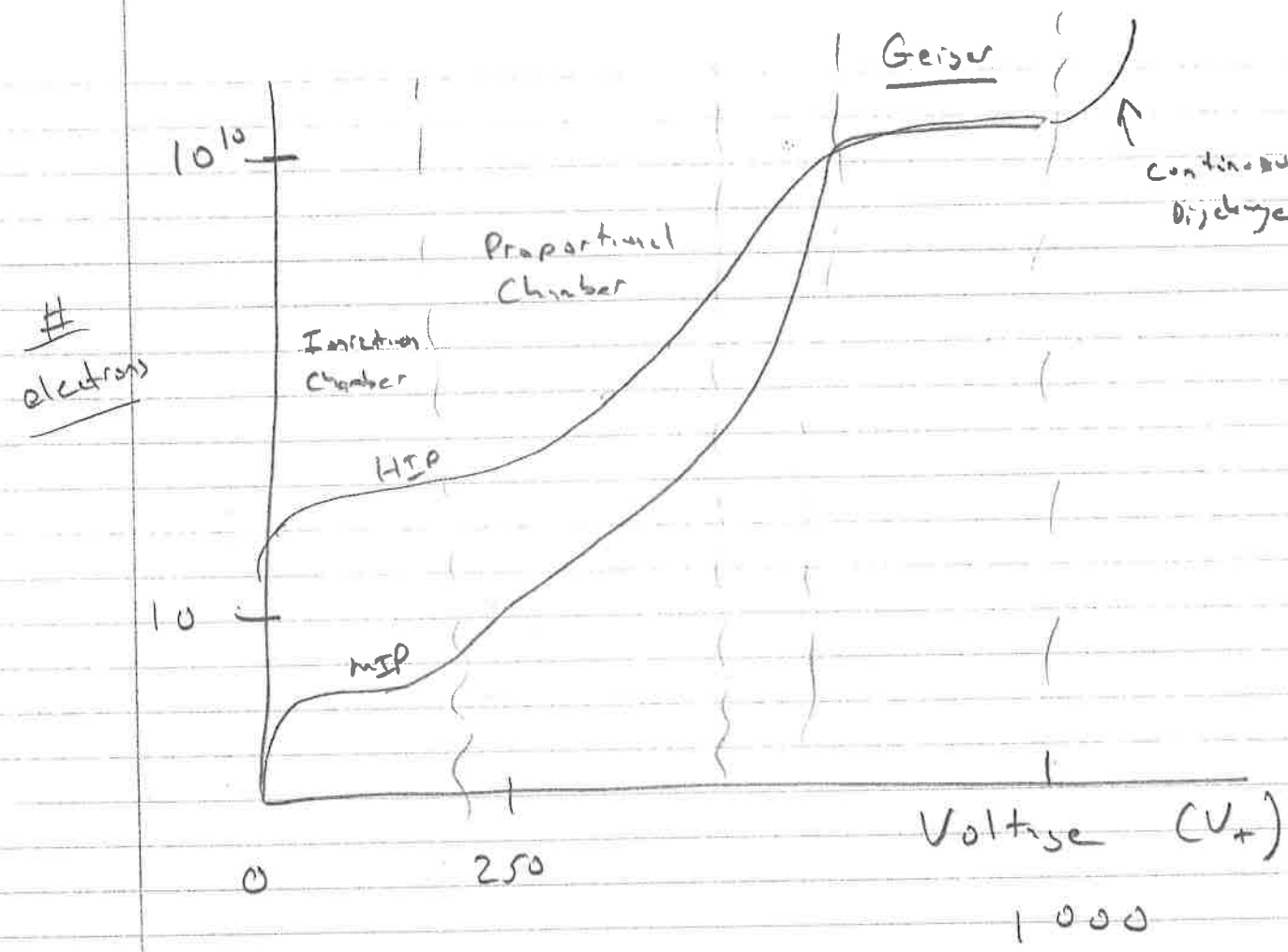
(b) Compton Scattering



(c) Pair production

Cascades:





As voltage increased:

(1) Collect ionized charge

(2) Voltage accelerates electrons to high energy energy to cause additional collisions, liberating additional electrons

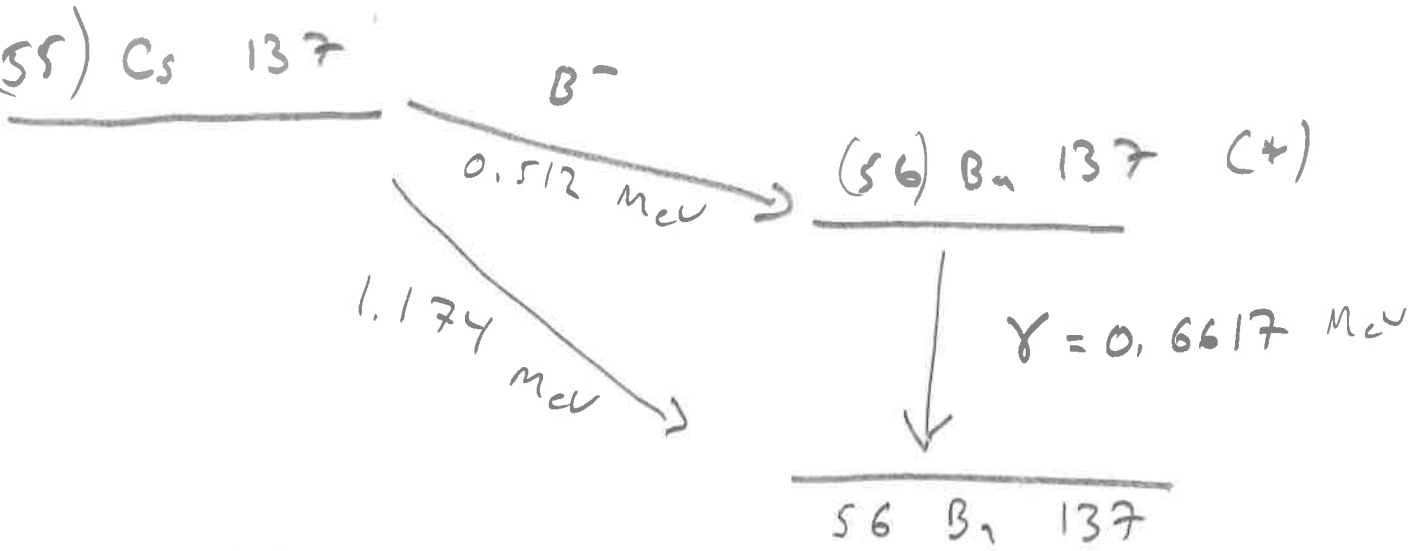
(3) Saturation

From plot

$$V = \frac{Q}{C} = \frac{1.6 \times 10^{-19} C \times 10^{10}}{10^{-9} F} = \boxed{1.6\ V}$$

$\Rightarrow$ Proportion Chambers require fancy electronics,

Often, nuclear decays result in a nucleus that is not in its ground state. It reaches ground state by γ emission



55 Cs 137 : 136.90709 amu

56 Ba 137 : 136.90582 amu

Δm = 0.00127 amu

(1 amu) c^2 = 931×10^6 eV

$\Delta m c^2 \sim$ 1.18 MeV

Geiger Counter