Spacial parallelism → dupe. hardware.

Temporal parallelism /pipelining. , multiple stages.

Throughput # tokens produced per unit time. (with expense of latency.
rate of finishing task...

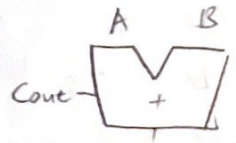
Ch. 5 p1-8.

Intro. Arith. Num systems. seq. building blocks, mem / logic arrays.

Building blocks: gates multiplex. decoders, registers, arith circuits, counters. memo array, logic array.

1-Bit adder.

half adder



add A and B.

A	B	Cout	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

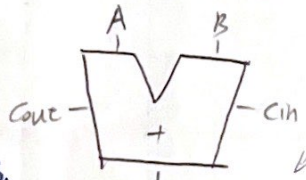
$S = A \oplus B$ ← carry... exclusive or. (two inputs)

$Cout = AB$ ← and.

XOR gate is hard to implement!

Try using half/full adders.

Full adder



adds 3 one-bit inputs

Cin	A	B	Cout	S
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

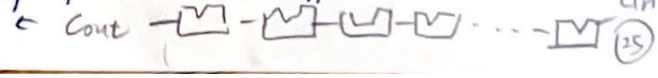
add up to 3 bc. ∃ Cin.

$S = A \oplus B \oplus Cin$

$Cout = AB + ACin + BCin$

carry out .. majority function. (more 1's than 0's) ⇒ 1.

potentially faster.. dedicate specific circuitry for ripple



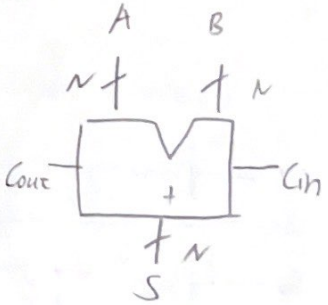
carry propagate adders.

Multibit Adders (CPA)s.

- Ripple-carry (slow)
- Carry-lookahead (fast)
- Prefix (faster).

precomputation. requires some hardware.

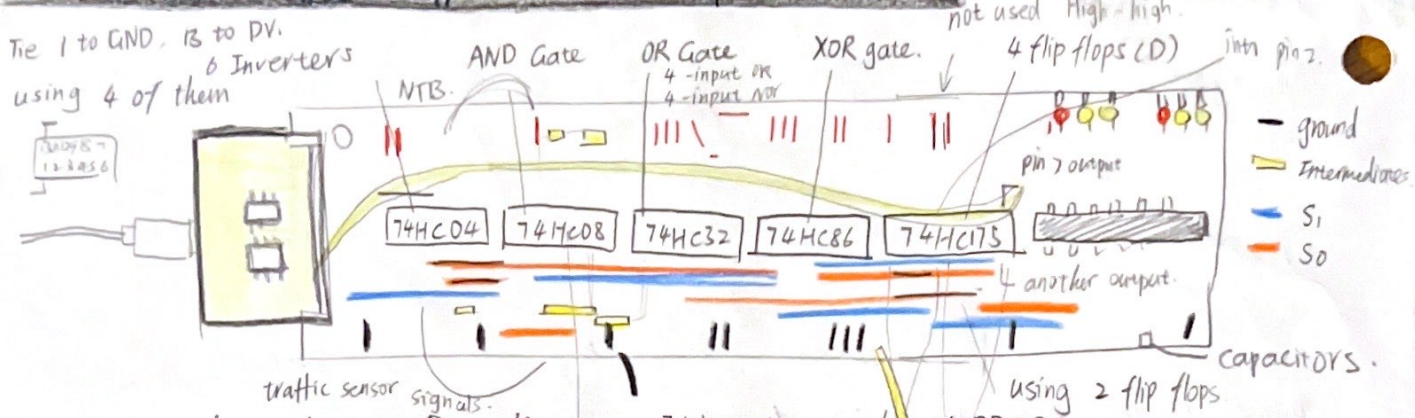
carry select



Ripple carry. - chain 1-bit adders together.

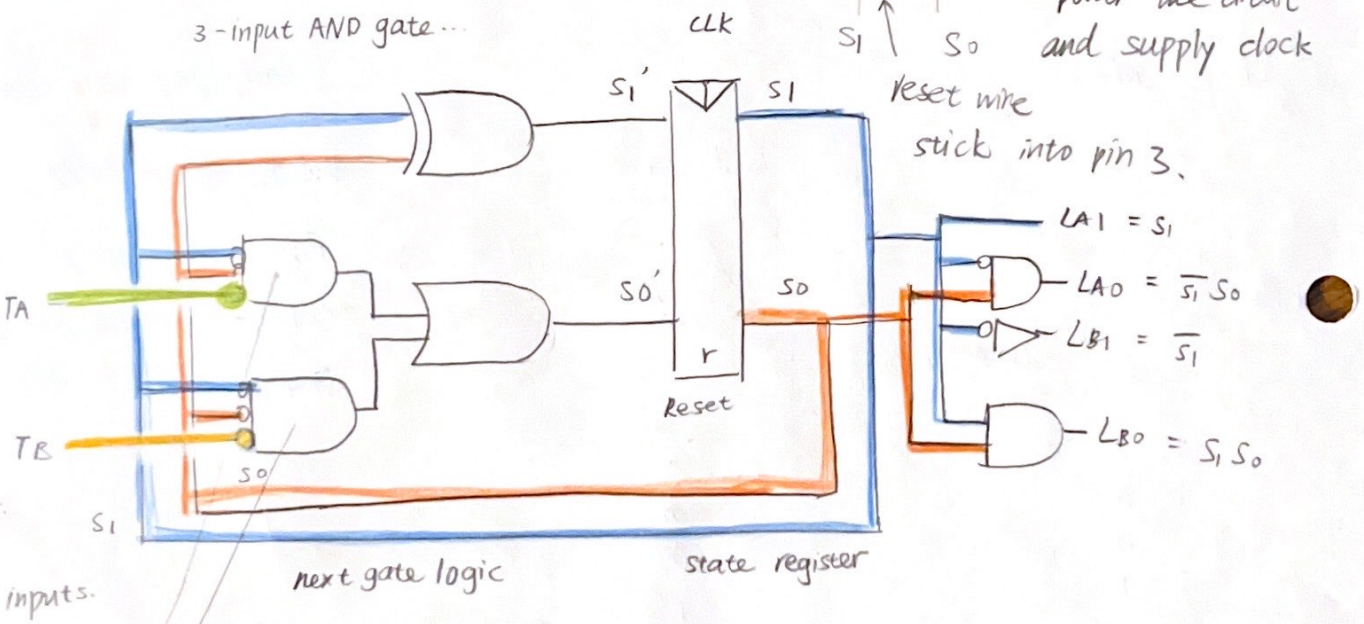
slow bc. linear computation...

7A Bread Board Traffic Circuit Explained 3/10 course website down



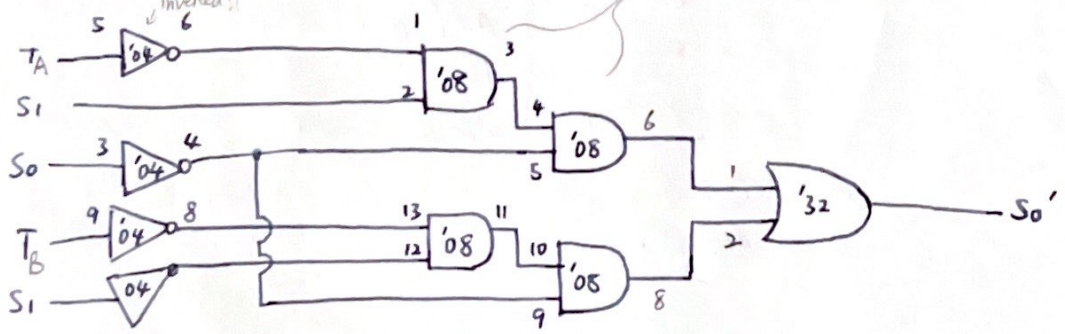
Look up data sheet: Digi-Key. sn 74hc175 search. 16 PDIP.

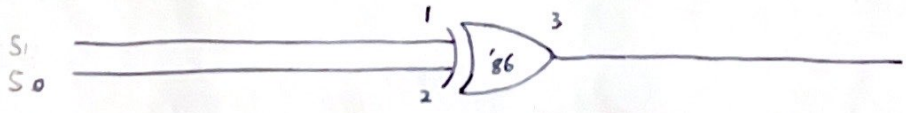
- power the circuit and supply clock

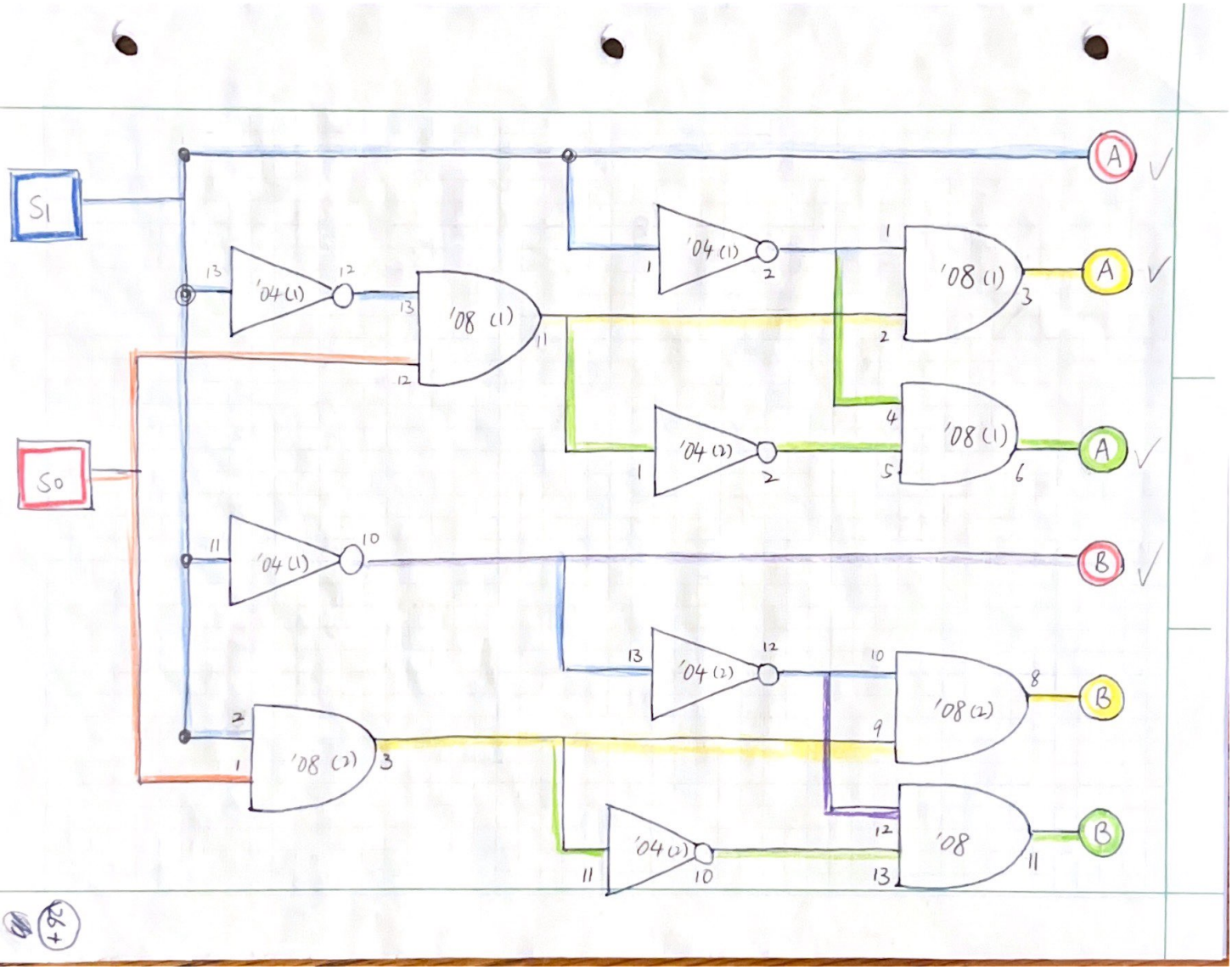


inverters invert S0, S1, TA, TB.

3-input AND gates are composed of 2 AND gates.






S1
S0
A
A
A
B
B
B
'04 (1)
'04 (1)
'08 (1)
'08 (1)
'04 (2)
'08 (1)
'04 (1)
'04 (2)
'08 (2)
'08 (2)
'04 (3)
'08

PCB template on MATLAB. pcb_traffic.m.

```
clk = 2;    % clock on Pin 2
nRst = 3;   % reset on pin 3

pic - setl (clk);   % initialize to 0.
pic - setl (nRst);

% set output mode.
pic - dig - O (clk)
pic - dic - O (nRst),

% pause
 pause (1);

% Deassert nRst.
pic - seth (nRst);
    set high
```

~~Pin 5~~
Pin 15 can't be in output mode.
Spits error. physically impossible.
No internal clocks...
Pin 1 / 16 can't be input

```
% Toggle clock every second.
while true
    pause (1);
    pic - seth (clk);
    pause (1);
    pic - setl (clk);
end
```

} loop doesn't stop.

→ Then the clock/LED flashes...
→ connect 2 boards

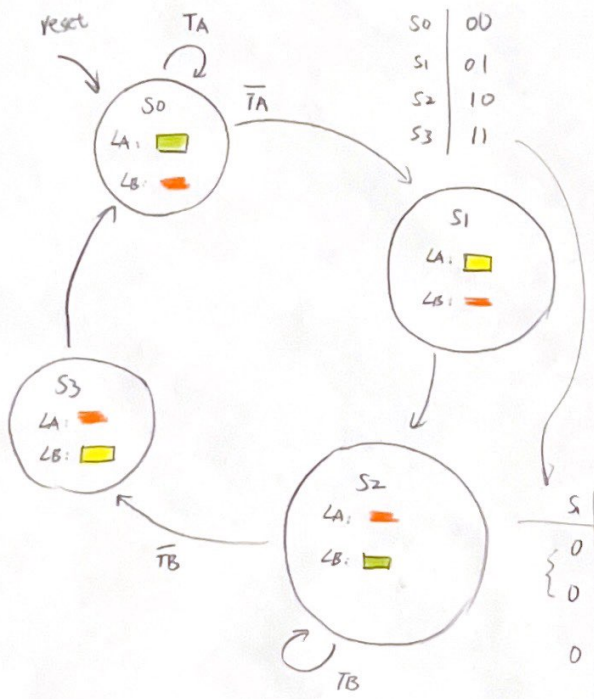
3/13 Prep Work.

Slides Ch.3 P32-45.

- Has the FSM on slide #43.
- Need to control 2 sets of LED's. → 6 LED signals from 2-state bits S_1 and S_0.
- T_A and T_B grounded.
- Hold on green when one of T_A or T_B is ~~grounded~~ connected to 5V.
- 300 Ω resistor (orange, black, brown, gold) → green. } LED's.
  160 Ω red, amber, yellow → blue

↑ "while has traffic"

Notes 3/15



Encoding	
S0	00
S1	01
S2	10
S3	11

curr S	Inputs TA	Inputs TB	next state S'	
S0	0	X	S1	← A no traffic. change
S0	1	X	S0	← A has traffic. Stay.
S1	X	X	S2	← always transitions.
S2	X	0	S3	← B no traffic. △
S2	X	1	S2	← B has traffic. Stay.
S3	X	X	S0	← always △ (yellow)

S_1	S_0	T_A	T_B	S_1'	S_0'
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

↙ exclusive or.

$S_1' = S_1 \oplus S_0$

$S_0' = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\,\overline{S_0}\,\overline{T_B}$

① Clock set up?
- Powered?
- Output?
- Four wires where?

② What's the goal?

③ Breadboard Explain ...

④ Debug? where's everything?

⑤ what to do w. the LA_1, LA_0, LB_1, LB_0 out wires? 3 lights?

What's on the slides P32-45? #43, #35. description

output table.

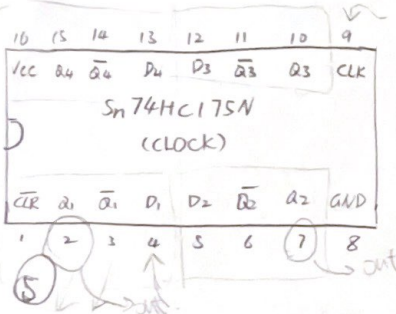
Encoding		curr S_1	curr S_0	LA_1	LA_0	LB_1	LB_0	
green	00	(S_0)	0	0	0	1	0	✓
yellow	01	(S_1)	0	1	0	1	0	✓
red	10	(S_2)	1	0	1	0	0	✓
		(S_3)	1	1	1	0	1	✓

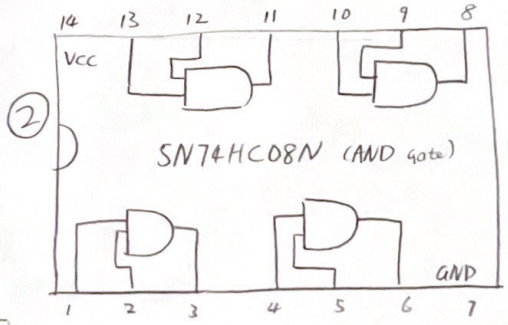
$LA_1 = S_1$

$LA_0 = \overline{S_1}\, S_0$

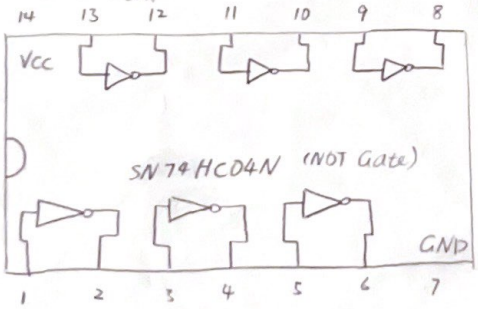
$LB_1 = \overline{S_1}$

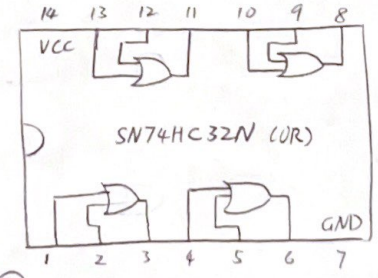
$LB_0 = S_1 S_0$

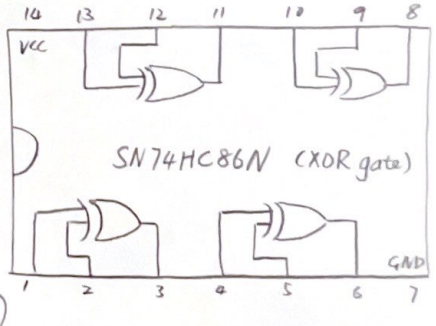


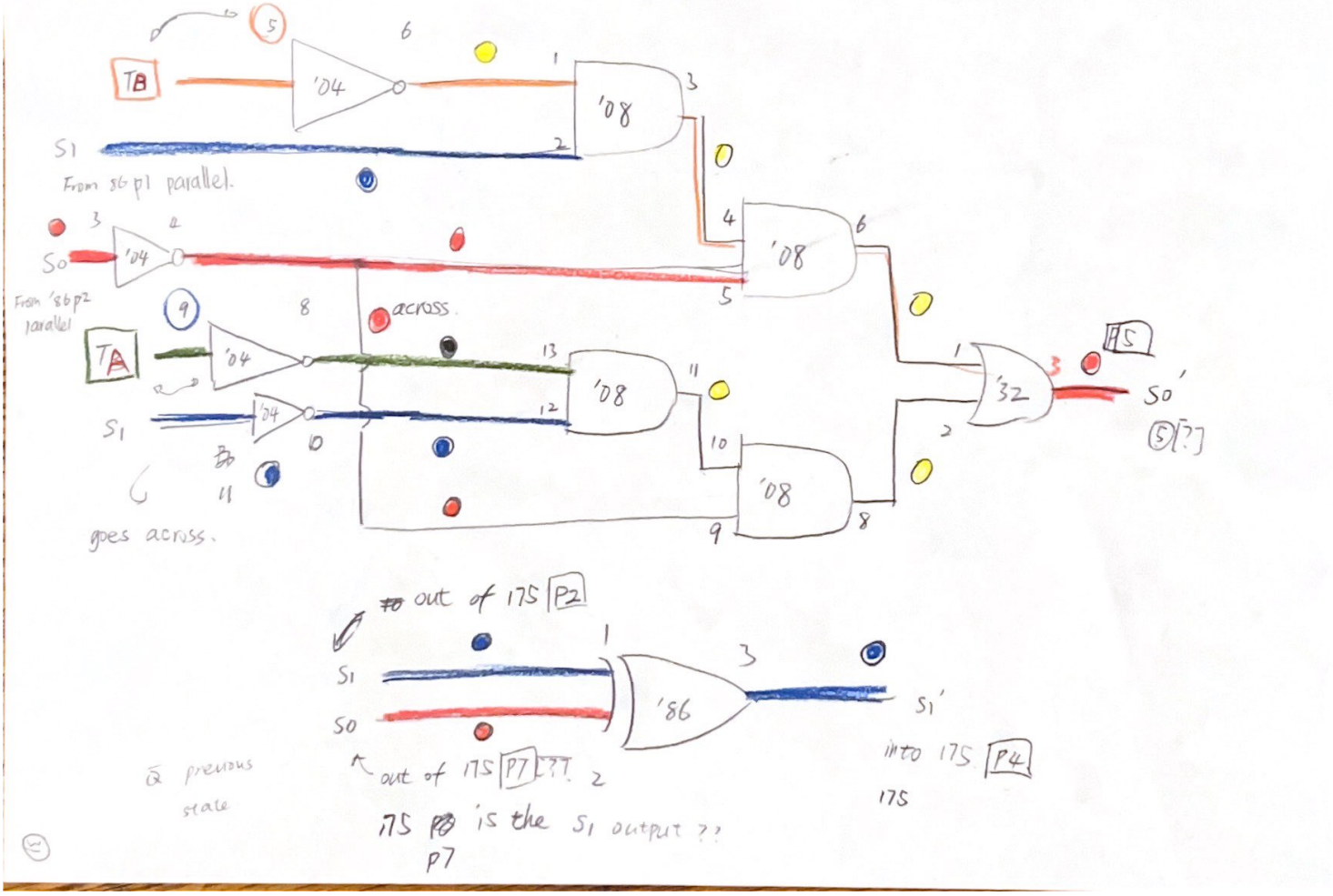


12 inches.
6ft



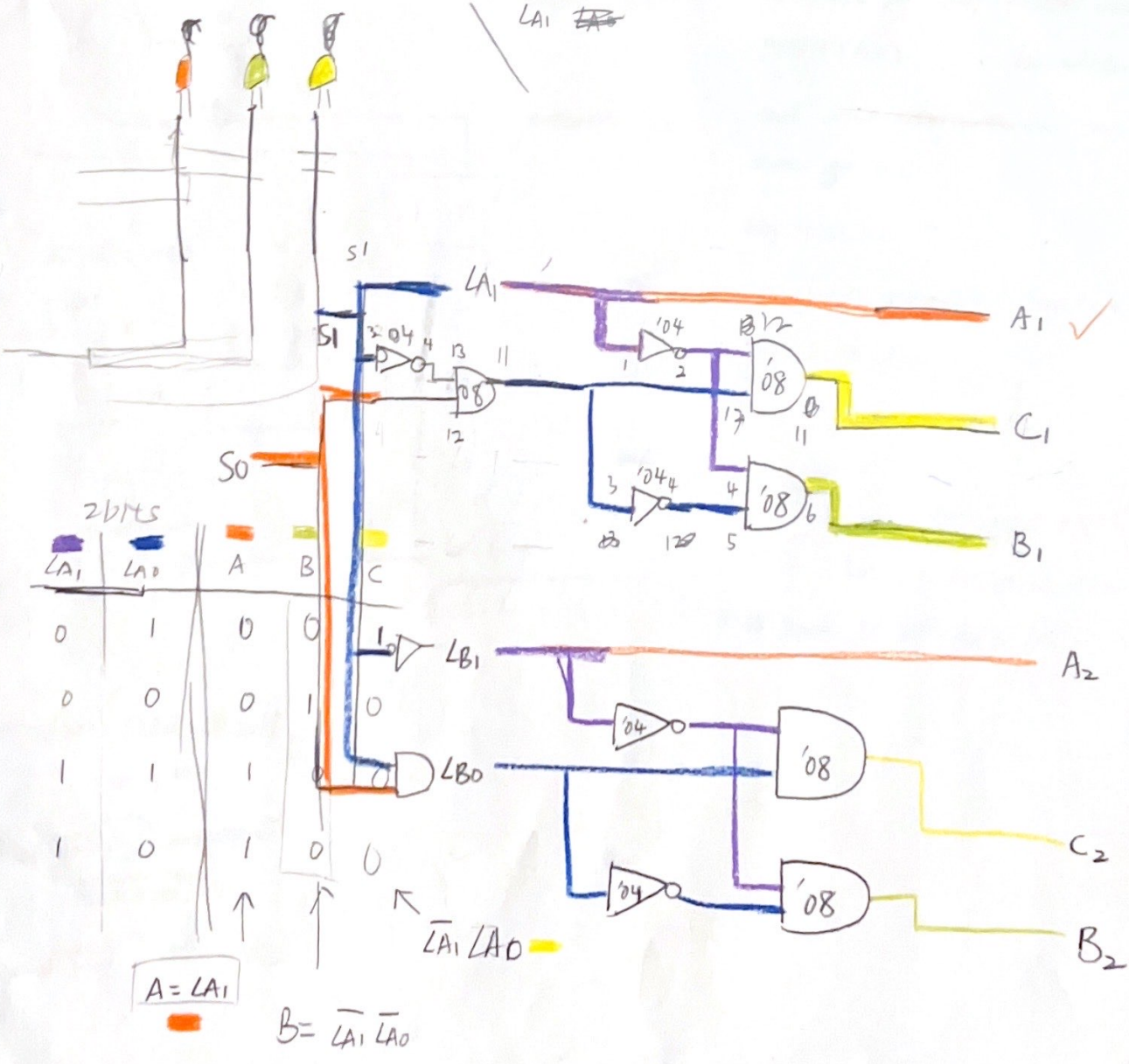





TB
'04
'08
S1
From 86 p1 parallel.
S0
From '86 p2 parallel
TA
across.
'32
S0'
goes across.
out of 175 P2
'86
S1
S0
S1'
into 175 P4
out of 175 P7 ??
175 P7 is the S1 output ??
previous state

L_{A1}	L_{A0}	Light Colors
0	1	
0	0	
1	1	
1	0	

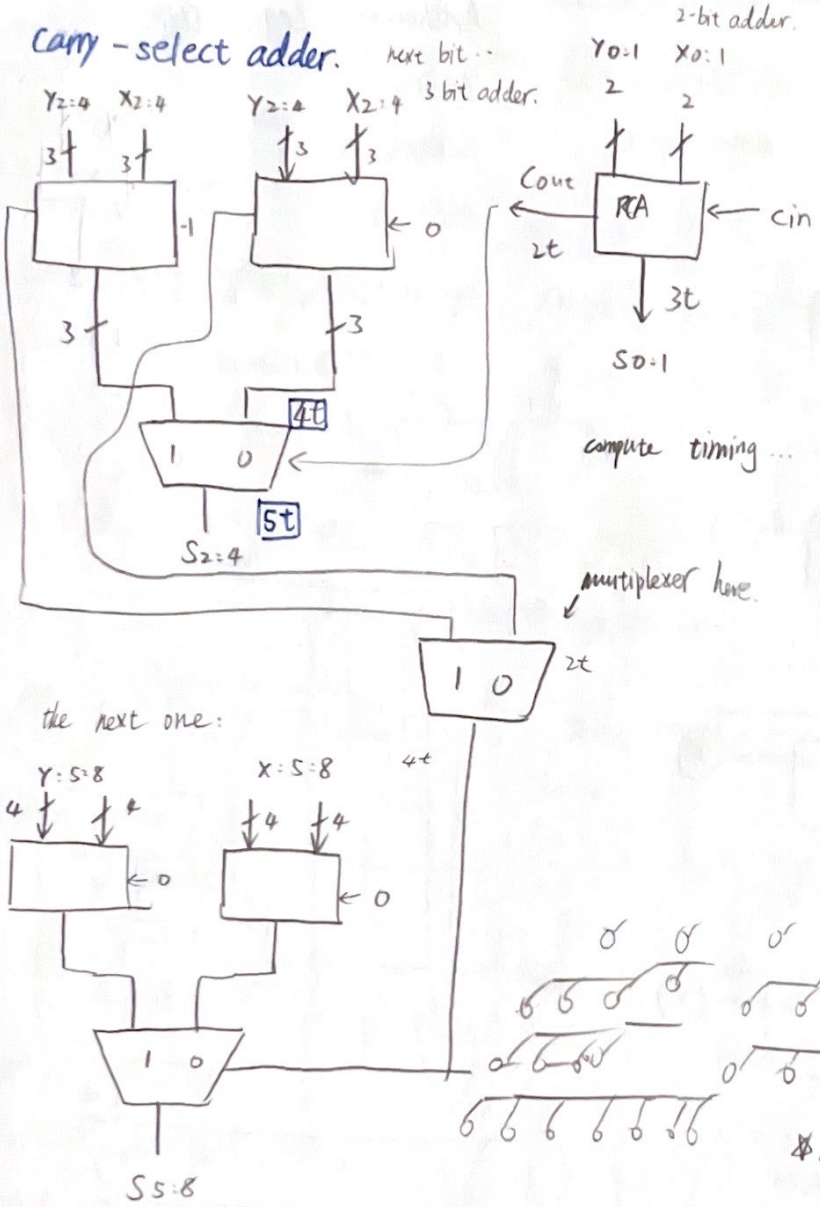
L_{B1}	L_{B0}	Light Colors
0	1	
0	0	
1	1	
1	0	



8A Carry Look Ahead, Ripple, Carry - Select. 3/17

No synchronizer needed ... full credit. (173).

Carry - select adder.



compute timing ...

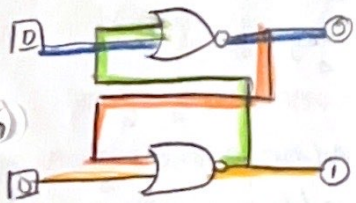
the next one:

good for large # of bits but not for small ones.

* no need to implement this!

Logism build circuit.

nor gate



VCC	16
RE3	15 ← Only digital input
RA1	14
RA5	10
RB7	9
RB1	3
RB0	2
GND	1

* # of number of bits ↑ account for propogation delay. progression thru multiplexer.

each new component has one more ~~go~~ bit.

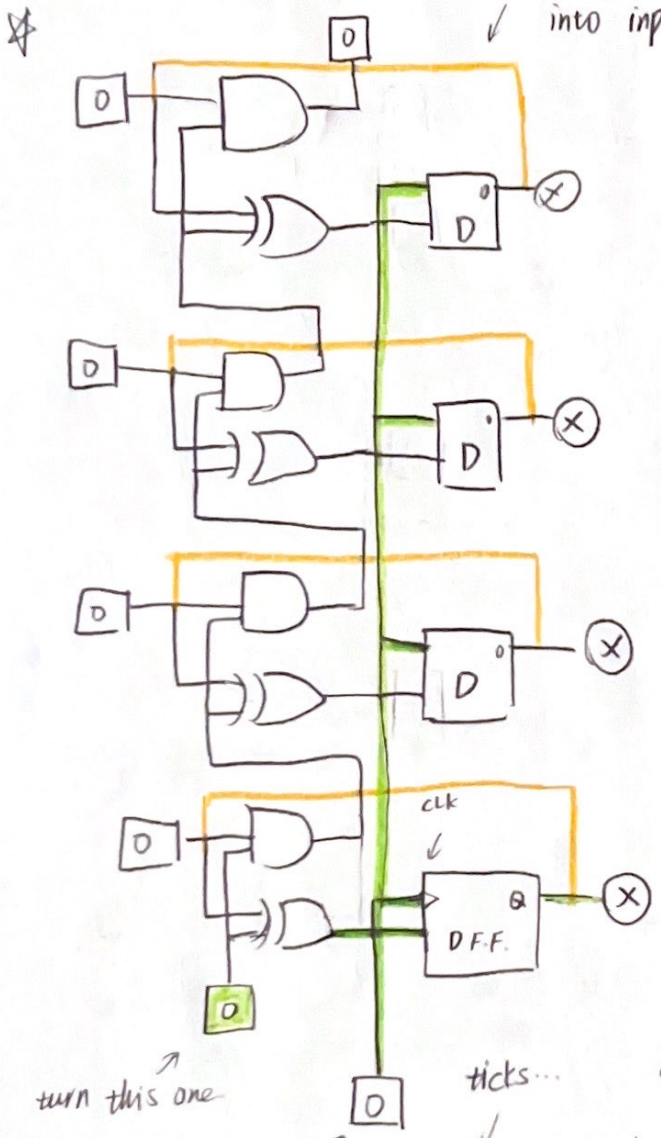
Very fast ...

Can choose group size. Logarithmic propogation delay

8B Adder Logism, Matlab, Arithmetic Logic Unit / Status Flags, Shifters, Multipliers, Dividers. 3/19.

4-bit increment circuit.

connect outputs back into input



turn this one

input. ← change clock

ticks...

Keep toggling this. It counts from $0 - 2^3$.

overflow Carry

Ch. 5 P43.

some extra components for status flags.

Arithmetic Logic Unit

Addition

Subtraction

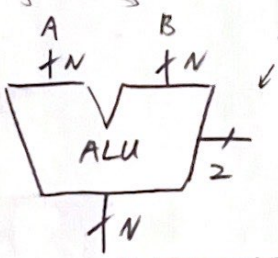
AND OR

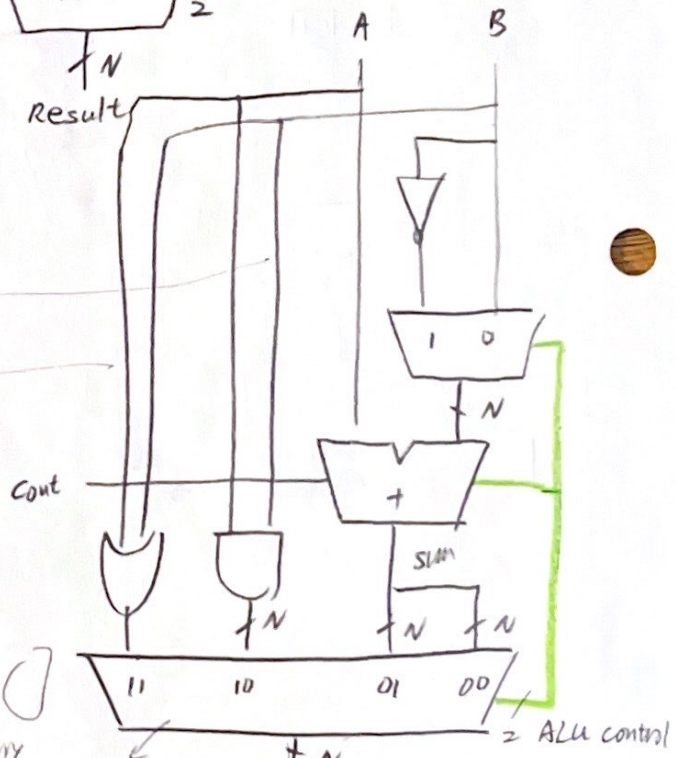
logical. bit-wise basis.

ctrl. 1:0	
0 0	Add
0 1	Subtract
1 0	And
1 1	Or.

inputs

ALU control. ↙ to the code





multiplexer chooses result.

neg. zero.

Status flag. N - negative

Z - result is 0 (zero)

C - Adder produces carry out

V - adder overflowed

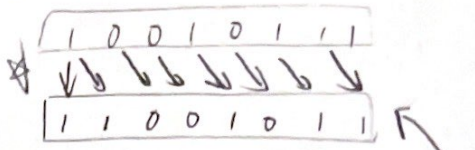
* overflow. ⊕ + ⊕ = ⊖ ! issue!

same-signed #'s produce a sum w. opposite sign.

V=1 if ALU is doing ⊕ or ⊖ (ctrl = 0.)

↳ And. A and Sum have opposite signs.

↳ And. same signs upon addition OR different signs upon subtraction.



Always shift right the MSB

What to do with the flags ?

Shifters

Logical shifter — filled with 0's.

11001 >> 2 = 00110

11001 << 2 = 00100

Arith shifter.

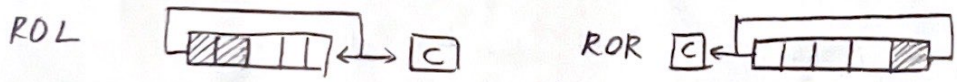
11001 >>> 2 = 11110 ← right shift, preserve the sign

11001 <<< 2 = 00100 left shift stays the same

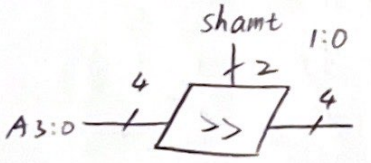
Rotator

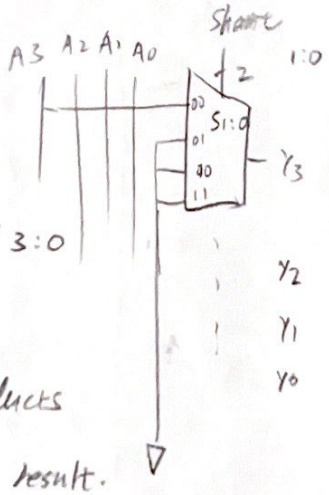
11001 ROR 2 = 01110 — rotate the 2 rightest bits.

11001 ROL 2 = 00111 — rotate the 2 leftest bits.



Shifter design





partial products

shifted partial products

summed to form result.

log n # of multiplexers to go thru.

Shift to the left = multiply by 2.

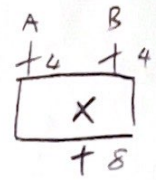
take 2's comp. shift to the left.

11101 << 2 = 10100 ($-3 \times 2^2 = -12$)

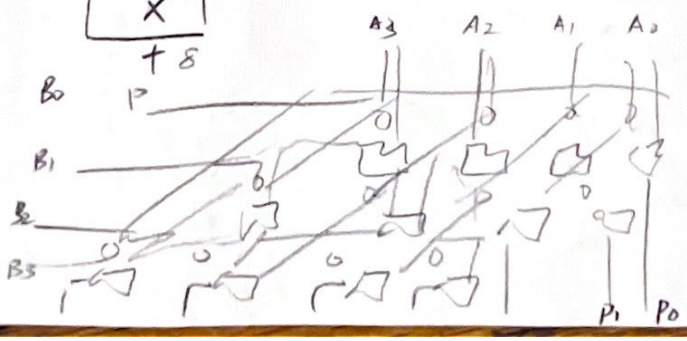
A >>> N = $A \div 2^N$.

10000 >>> 2 = 11100 ($-16 \div 2^2 = -4$)

```
  01 01  (5)
X 01 11  (7)
-------
  01 01
 0101
 0101
+0000
-------
 0100011
 5x7 = 35
```



Not efficient



full adders and AND gates.

result:

P7 P6 P5 P4 P3 P2 P1 P0.

8C Dividers, Floating Pt. Calculations. 3/23

$A/B = Q + R$ — separate quotient & remainder.

Decimal example: 2584 / 15 = 172 R4

$\frac{A}{B} \rightarrow Q, R$ such that $A = B \times Q + R$, $0 \le R < B$. might be negative...

Long hand revisited:

```
      172 R4
15 ) 2584
```

① 0002 − 15 → −13 bc. failed. — 0 _ _ _

② 0025 − 15 = 10 — 01 _ _

③ 0108 − 105 = 3 success — 017 _

④ 0034 − 30 = 4 — 0172

bring the next.

success. bring in the next bit

binary.

Binary: 1101 / 10 = 0110 R1

△ crank from orig. if fail

△ add from next if succ.

grab the ~~right~~ left most piece & subtract

```
 0 0 0 1 | 0 _ _ _
-0 0 1 0
 1 1 1 1
```

fails bc. result is −1.

test w. extra bit. from 1 to the right

```
 0 0 1 1 | 0 1 _ _
-0 0 1 0
 0 0 0 1
```

success

shift to left [?] why

```
-0 0 1 0   (3rd digit)
 0 0 1 0 | 0 1 1 _
 0 0 0 0
```

success.

only use the rightmost

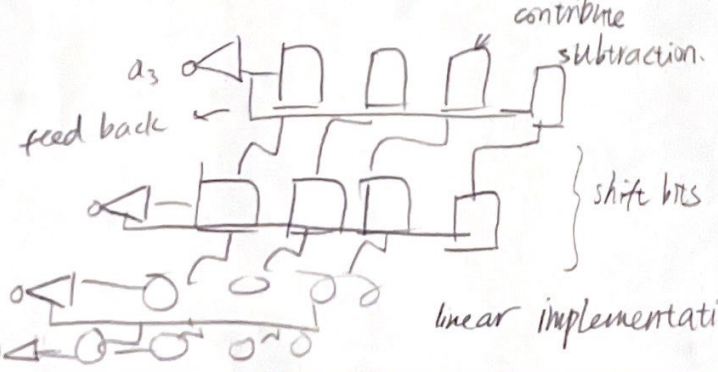
```
-0 0 0 1 | 0 1 1 0  R1
 0 0 1 0
 1 1 1 1
```

now.

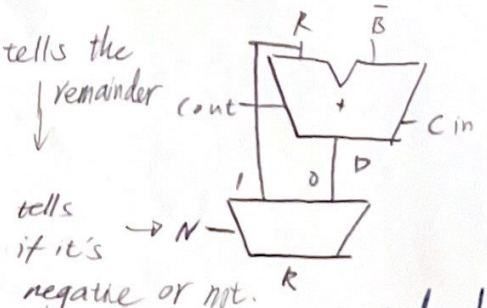
retain the last value.

Algorithm.

```
R' = 0
for i = N-1 to 0
    R = {R' << 1, Ai}
    D = R - B
    if D < 0   Qi = 0 ; R' = R
    else       Qi = 1 ; R' = D
R = R'
```



also tells the remainder



Redundant

linear implementation. straightforward but slow.

Number systems

positive # unsigned bin

neg # two's complement. sign / mag numbers.

fractions { fixed - point
floating - point binary point floats to the sig 1.
similar to sci. notation.

6.75 has 4 int bits and 4 frac bits.

0 1 1 0 . 1 1 0 0

$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$ exponent.

$273 = 2.73 \times 10^2$. $\pm M \times B^E$

M B mantissa base

* Mantissa needs to be specified and ~~0 ≤ M ≤ 10.~~ $0 < M < 10$.

Ex: 75_{10} -75_{10}

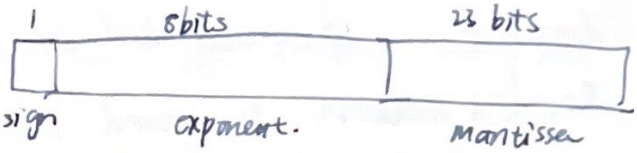
0 1 1 1 . 1 0 0 0 +7.5 0111 1000

7 0.5 inv. bits 1000 0111

add 1 to lsb + 1

1000 1000

* $-8 + \frac{1}{2}$



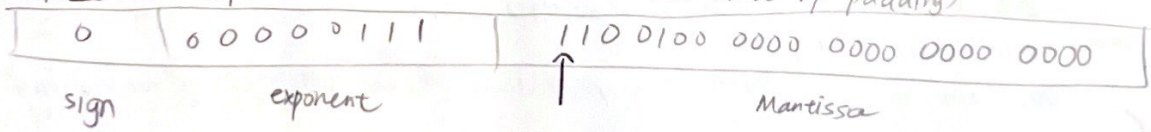
32 - bit floating pt.

IEEE 754 floating - point standard.

Ex: convert 288_{10}

$288_{10} = 1\,1100100_2$

~~rep1~~ binary sci. note: $1.1100100_2 = 1.1100\,1_2 \times 2^7$



(positive is 0).

rep 2 : implicit leading 1. bc. no need to store it.

rep 3 = biased rep. Exp of 7 $127 + 7 = 134 = 0 \times 10000110_2$ Exponent encoding --

for better comparisons

Ex. -58.25_{10} → $58.25_{10} = 111010.01_2$ (only convert abs!)

→ 1.1101001×2^5

→ sign 1 (neg.

8 exp: $(127 + 5) = 132 = 1000\,0100_2$

23 frac: 110 1001 0000 0000 0000 0000 in hex: 0x C2690000

output: 1 100 0010 0 110 1001 0000 0000 0000 0000

specials.

	Sign	Ex	Frac
0	X	00000000	00000000000000000000000
∞	0	11111111	
-∞	1	11111111	
NaN	X	11111111	non-zero

} ds computations used in subsequent computations. debug

double pres 64 bit 1 sign 11 exp 52 frac bias = 1023 $(2^N - 1)$

not supported in microprocessors.

denormals ... 0 exponent and non-zero fractions.

Rounding modes: Ex. round 1.100101 (1.578125) to 3 frac bits.

down : 1.100

up : 1.101

→0 : 1.100

→ nearest : 1.101

1.625 is closer to 1.578125 than 1.5 is.

floating pt additions is hard. Procedure.

① Extract exponent

② prepend leading 1 fm mantissa

③ compare exponents

④ shift smaller mantissa if necessary

⑤ Add mantissas.

⑥ Norm. mantissa & adjust exps if necessary

⑦ Round result

⑧ Assemble exponent & fraction back into floating-point format and then round.

Ex: ① get exponent

S=0 E=127 F=.1

S=0 E=128 F=.101

0 01111111 100 0000 0000 0000 0000 0000

0 10000000 101 0000 0000 0000 0000 0000

② Prepend leading 1.

$N_1 = 1.1_2$

$N_2 = 1.101_2$

③ compare exponents. 127 - 128 = -1 NOT THE SAME!

N_1's shifted mantissa: 1.1 >> 1 = 0.11 ($\times 2^1$)

Add

0.11×2^1

$+ 1.101 \times 2^1$

10.011×2^1

⑥ $10.011 \times 2^1 = 1.0011 \times 2^2$ $(0 < M < 1)$

⑦ round: no need.

⑧ assemble. 0 10000001 001 1000 0000 0000 0000 0000

$A \times B = (a_m \times b_m) \times 10^{(a_m + b_m)}$

multiplication: multiply mantissas and ⊕ exponents.

9B

Multiplication

```
  1010   Multiplicand (M)
  1101   Multiplier   (Q)
  ----
  1010
  0000
  1010
  1010
  ----
  10000010
```

carries

use a const amt of resources bc. one bit at a time.

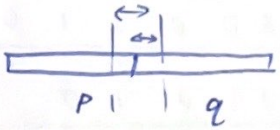
Same answer

prevent overflow

```
  1010
  1101
  ----
  00000000
  01010   ← M·q0
  ----
  010100000
```

gen. add. cycle... accumulate.

shift to the right.



shifting / sharing bits between p & q.

needs 9 bits rather than 12.

```
  q3 q2 q1 q0
   1  1  0  1
```

```
  001010000
+   01010   ← M·q1
  ----
  001011000
  ↘↘↘↘↘
  000101000  ← M·q2
+   01010
  ----
  011001000
    ↘↘↘      ← M·q3
+   00110010
  ----
  010000100
    ↘↘↘↘↘↘
+   10000010
  ----
  100000110
```

Negative numbers.

```
  -6   1010   (M)
  -3   1101   (Q)
  ----
  0000000000
  11010          M·q0
  ----
  1101000000
shift↘
  1110100000
sign extend 0000000000   M·q1
  ----
  1111010000
shift↘
  1111101000
  10110          M·q2
  ----
  1100010000
  ↘↘↘
  1111000100
- 11010          M·q3
  ----
 0 10110010000
  ↘↘↘↘
  1011001100
```

this should be a subtraction.

-8 +4 +2 +1

q3 q2 q1 q0

1 1 0 1

q3's weight

This is -78, not -18...

```
  111000100
- 11010
  ----
  000100100
↳ 000010010 = -18
```

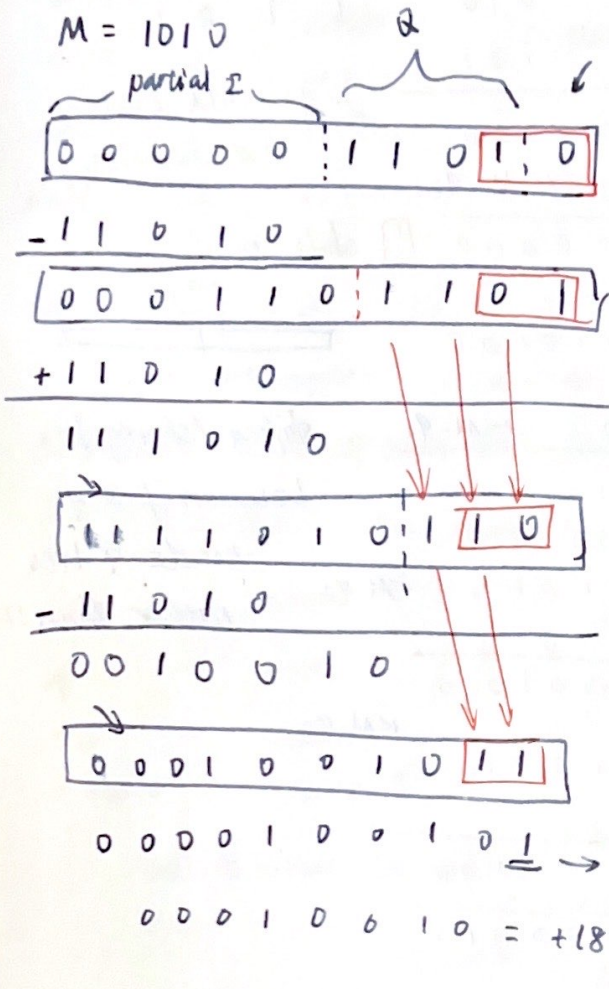
Special quick adds:

Q = 00010001 (=17)

8 shifts, 2 adds.

Q = 00011110 ← lots of borrows...

Booth's Algorithms: skip over a bunch of 0's w. $2^{j-1} - 2^j$



- These are both 2's complement.
- This is really formalized!
- Marks for P, Q being ⊕ and ⊖.
- When # < 0, either subtract (for ⊖) or do nothing bc. the previous ones can do subtractions for us.
- △ Downside... Look at various 2's.

Ex: Q: 0 0 1 0 1 1 1 1, 0

Q^* = 0 1 1 1 0 0 0 1 — 2 adds, 2 subtracts.

Q = 0 1 0 1 0 1 0 1, 0 — 4 adds, 4 subtracts.

especially when there are dis-continuous 1's

! booth's alg. doesn't always help!

Recoding ??? → do 4) 2-bit shifts at a time. instead of 8 1-bit shifts.

q_{i+1}	q_i	q_{i-1}	q_{i+1}^*	q_i^*
0	0	0	0	0
0	0	1	0	1
0	1	0	1	⊥
0	1	1	1	0
1	0	0	⊥	0
1	0	1	⊥	⊥
1	1	0	0	⊥
1	1	1	0	0

⊥ subtract 1 add 0 do nothing

i	$i+1$	
0	0	= 0
+M	0	= +M
-M	+2M	= M
0	+2M	= 2M
0	-2M	= -2M
+M	-2M	= -M
-M	0	= -M
0	0	= 0

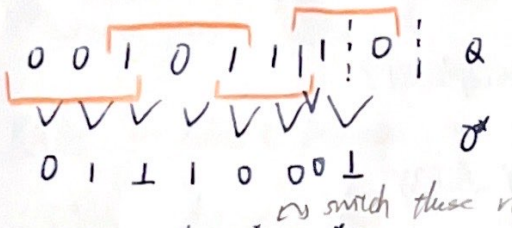
9C Shifters, Counters, RAM, final project explanation 3/29

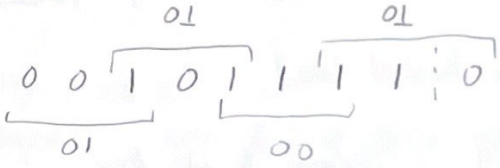
Rest of proj. 5.

- Extra breadboards. Collect parts. Go to Barton Hall a lot.

Booth's Alg.

00, 11 → 0
01 → 1
10 → ⊥





↳ switch these rows

q_{j+1}	q_j	q_{j-1}	q_j^*	$q_j'^{+1}$	
0	0	0	0	0	no op
0	0	1	1	0	→ add m
0	1	0	-1	1	at j : −M; at j+1 : +2M } net +M → add m
0	1	1	0	1	→ add 2m
1	0	0	0	-1	either row has 0... → add 2m
1	0	1	1	-1	at j = +M → sum M; at j+1 : −2M → sub M
1	1	0	-1	0	
1	1	1	0	0	no op

← use a multiplexer ... and some wiring.

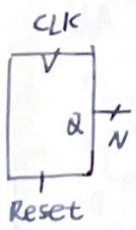
PPT. Floating Pt Addition.

Counters

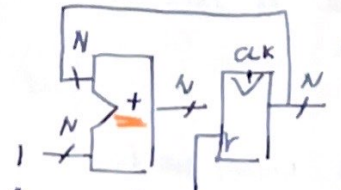
- Increments on clk edge.
- Cycle thru nums.

000 001 010 011 100 ...

- Digital clock displays.
- Program counters.



Implementation



chunck of F.F

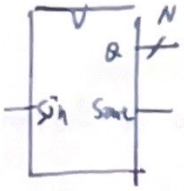
the 1 is the next-state logic.

The chip is in the kit counts up or down.

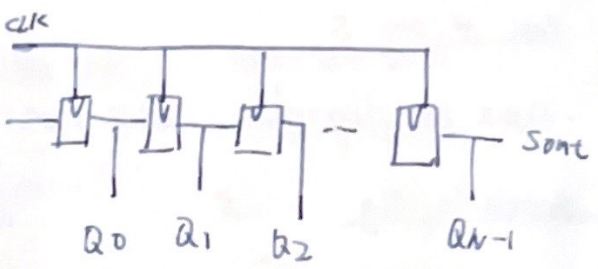
sync. counter is controlled by the carry-in input.

Shift Registers.

- shift a bit on CLK edge.
  - in/out
- Serial to // converter.



Imp:



Shift serial data. ↓

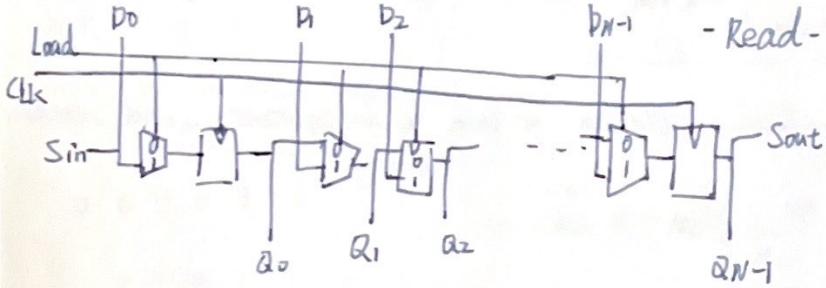
the otherway: parallel load (multiplexer).

SR. w. Parallel load.

Load = 1 = Normal N-bit reg.

Load = 0 : shift regis.

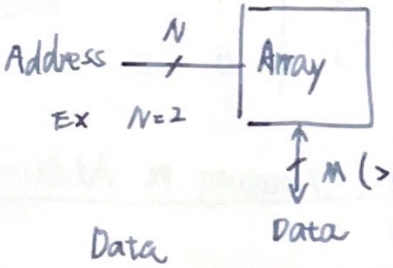
both ways!



Memory Arrays

- Stores large amount of data.
- Dynamic random access memory (DRAM)
- Static random access memory (SRAM)
- Read-only array (ROM)

M-bit data val read/written at each unique N-bit addresses.



Ex N=2

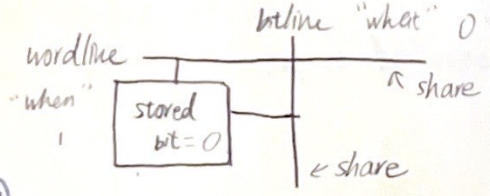
- 2-dim array of bit cells
- Each bit cell stores 1 bit.
- N address bits and M data bits.
  - 2^N rows & M colums
  - Depth (num rows (num words)
  - Width (num cols (size of word))
  - Array size: depth x width = $2^N \times M$

Addr.	Data		
11	0	1	0
10	1	0	0
01	1	1	0
00	0	1	1

depth ↑↓

← width →

the 3-bit word at addr. 10 is "100".



in computer

RAM: volatile read & write quickly

loses data when power off

ROM: nonvolatile

retains data.

writing is impossible or slow w/o power.

Non-sequential access.

easy access to any memory

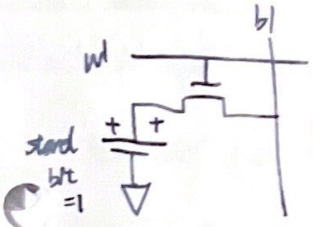
flash mem. in cams, thumb drives, digital cameras.

"burning fuses" 烧光保. This is no longer the case for flash memory.

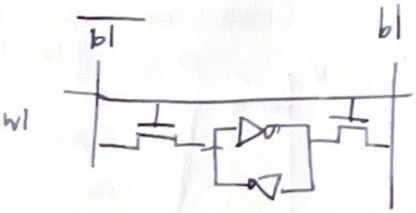
RAM ← Robert Dennard 1932- cheap.

DRAM dynamic ram. → capacitor (computers -- refreshed. or data gone). very small cells.

SRAM static RAM → cross-coupled inverters. same circuitry as a latch. as compact as possible.



SRAM

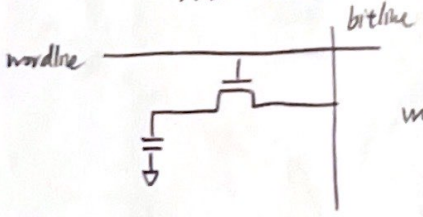


sense △ and flip.

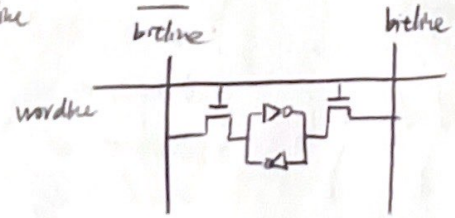
refresh ....

10A Memory Arrays, ROM (read-only memory) Dot Notation, Multi-Ported Memories, PLA's, FPGA's, LE (logic element 411

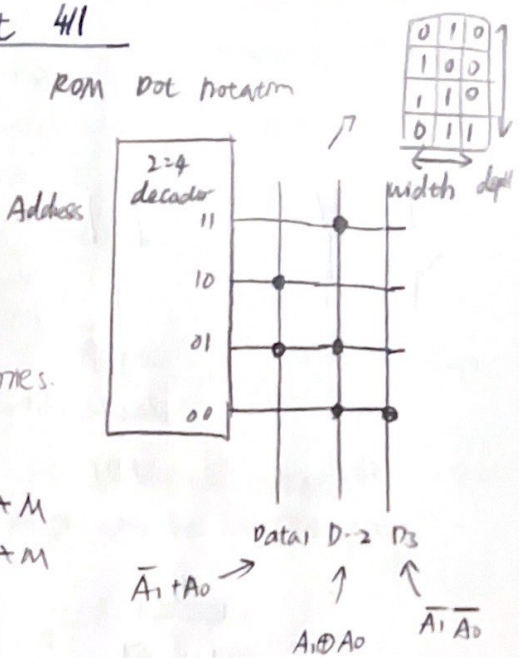
DRAM bit cell



SRAM bit cell



ROM Dot notation



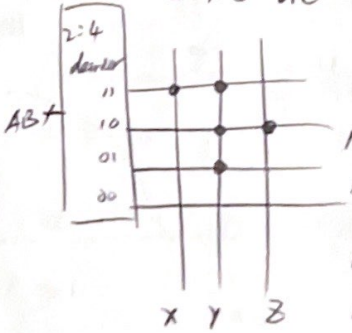
$2^2 \times 3$-bit

Ex:

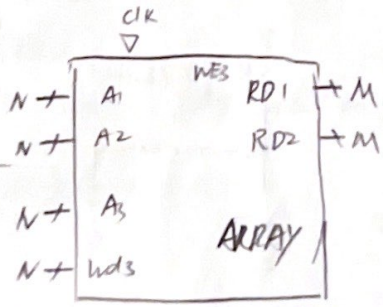
$X = AB$

$Y = A + B$

$Z = A\bar{B}$



Multi-ported memories.

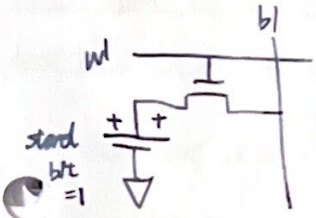


RAM: volatile (in computer) read & write quickly
loses data when power off
ROM: nonvolatile
retains data.
writing is impossible or slow w/o power.

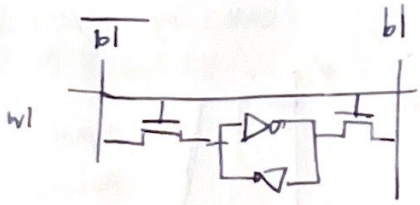
Non-sequential access.
easy access to any memory

flash mem. in cams, thumb drives, digital cameras.
"burning fuses" 烧光碟. This is no longer the case for flash memory.

RAM ← Robert Dennard 1932- cheap.
DRAM dynamic ram. → capacitor (computers -- refreshed. or data gone).
very small cells.
SRAM static RAM → cross-coupled inverters. same circuitry as a latch.
as compact as possible.

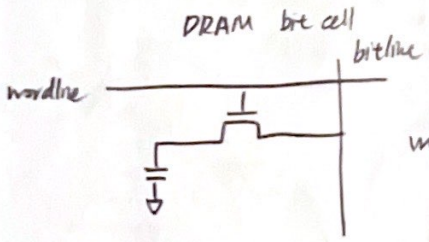


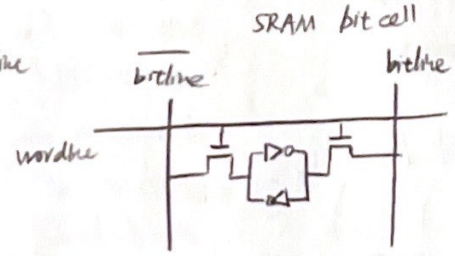
SRAM

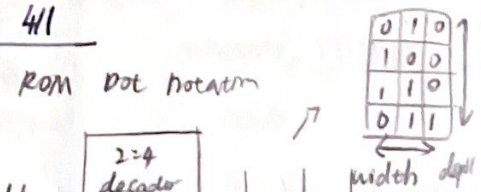


sense Δ and flip.
refresh ----

10A Memory Arrays, ROM (read-only memory) Dot Notation, Multi-Ported Memories, PLA's, FPGA's, LE (logic element 411

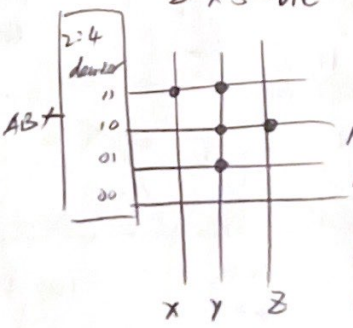




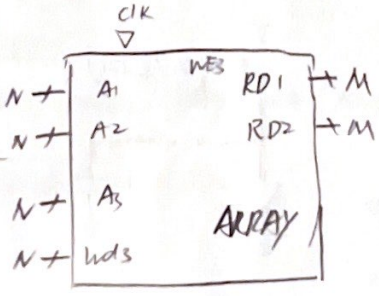


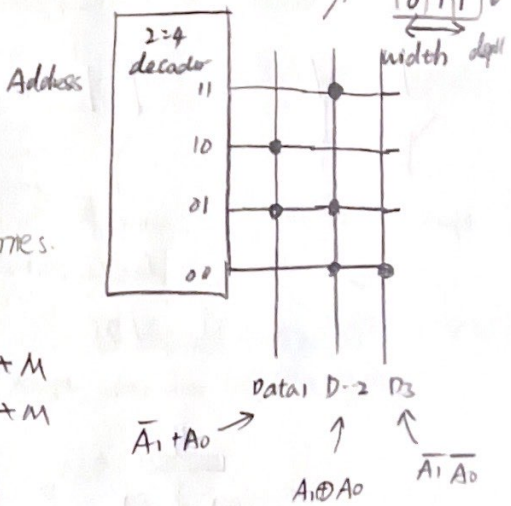
$2^2 \times 3$-bit

Ex:
$X = AB$
$Y = A + B$
$Z = A\bar{B}$



Multi-ported memories.

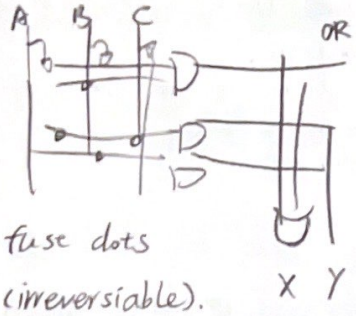




- ~~SystemVerilog~~ Memory Arrays. (code)

- PLA (programmable logic arrays) Fixed internal connections. AND arr + OR arr.

FPGA (Field programmable gate arrays) ① Arrs of logic elems.



fuse dots (irreversiable).

② combinational & seq. logic

③ programmable internal connections.

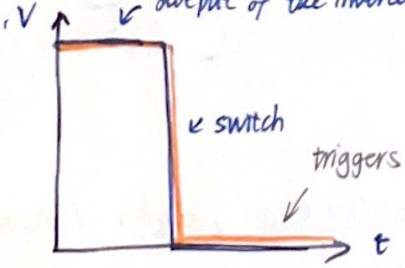
LE = Logic element

- LUT (look up tables) perform combinational logic.
- Flip - Flops : perform seq. logic
- Multiplexers : connect LUT's and ff.'s

Switches, delay

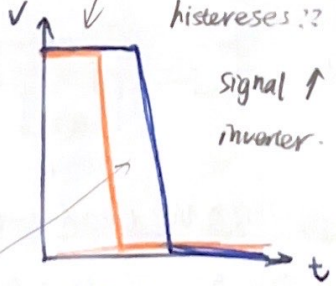
10B ~~Fancy Matlab Stuff~~ 4/3

SCHMMI.

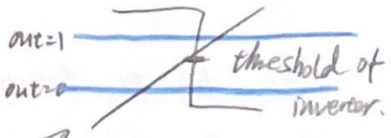


74HC04 Cinverter.

SCHMII inputs intentionally histereses ??



Signal ↑ when time ↑. inverter.



74HC14

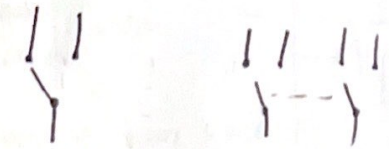
window large enough to handle most cases.

SMCHII has 2 thresholds depending on the output to avoid bad oscillations.

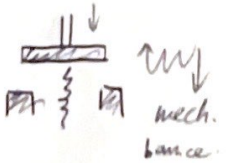
wb. noisy inputs?

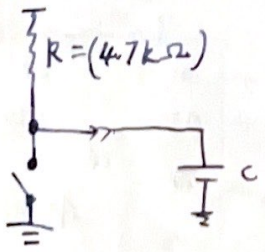
bad oscillations. stability issues.

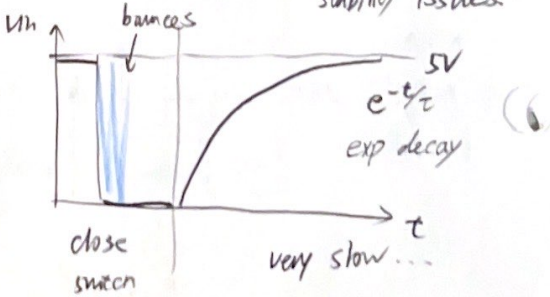
SPST single pole single throw



SPDT





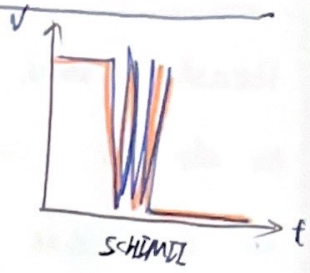


(54)

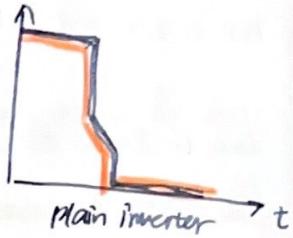
minimize bance: tact switches. (tactile switch)

If remove capacitator (stabalizer ???

bad oscillations in SCHEMIT trigger in intermediate state.



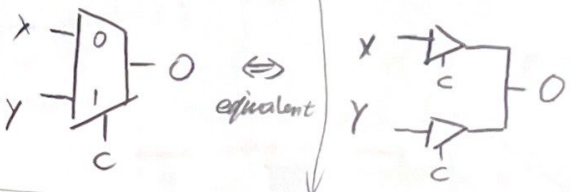
TTL : totem-pole } almost everything rail to rail | open collector } or
CMOS : push-pull } open drain } or nothing.



TTL/CMOS : tri-state } 0V, 5V, or nothing ? pascent set

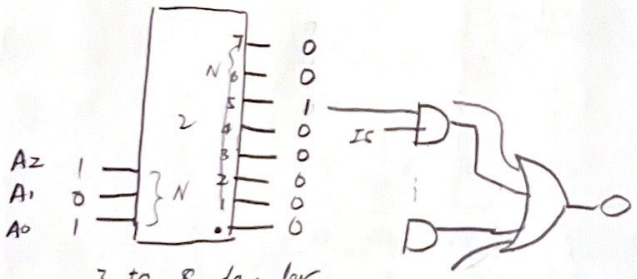
bi-directional signals

split multiplexor



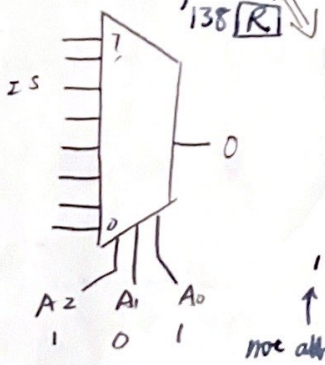
10C How to use Encoders and Decoders on Bread Board (HW6) 4/3

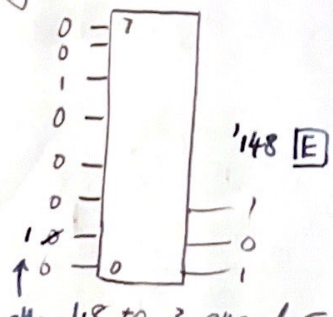
Decoder is RAM (memory array).



3 to 8 decoder

'138 R



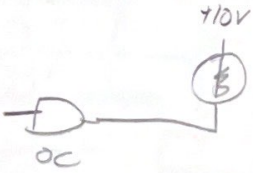


more allowed! 8 to 3 encoder

only one input can be active. Only set/reset can be high at once.

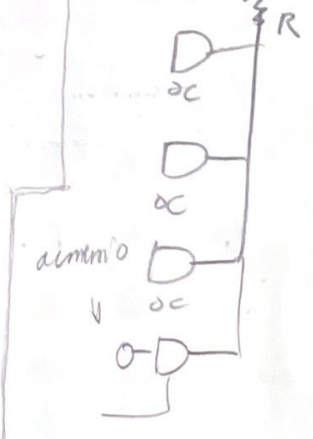
Or undefined behavior.

so.. we use the priority ~~encoder~~ encoder



Uses: ① High voltage (Lamp...)

② wired or



safe way to prove outputs bc. output is never high.

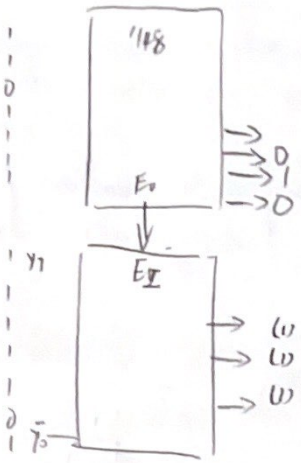
unknown # of signals

Datasheet, thuth table. priority encoder.

For chip 138

G_1	$\overline{G_{2A}}$	$\overline{G_{2B}}$
H	L	L

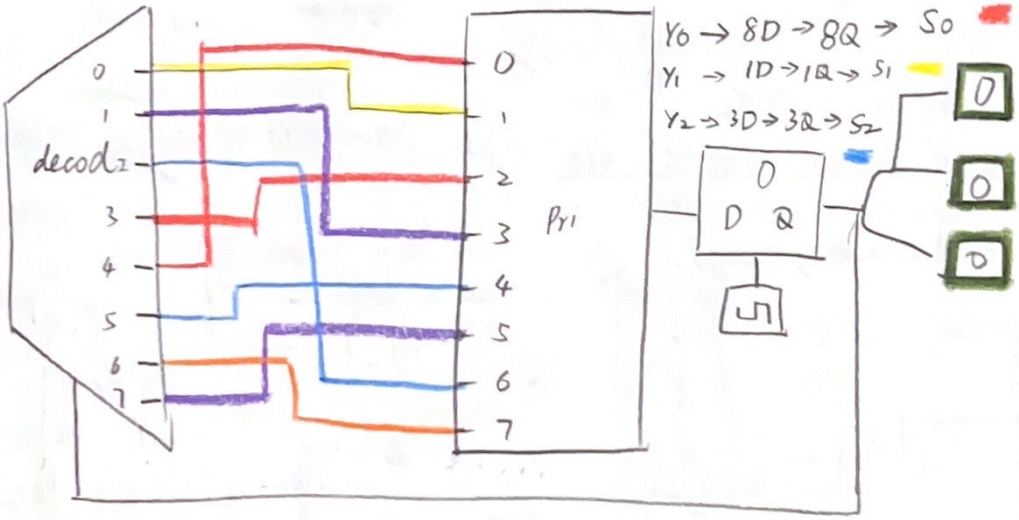
↑ has to have this for the chip to work.



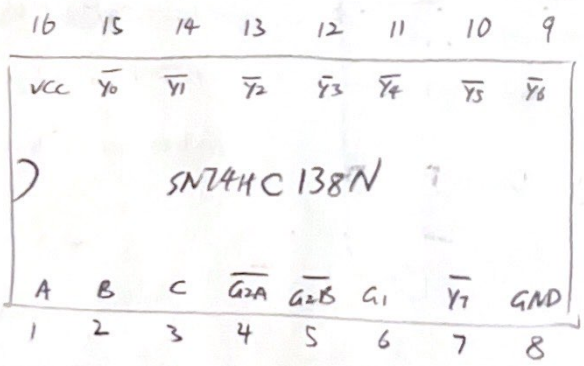
Gray's numberly

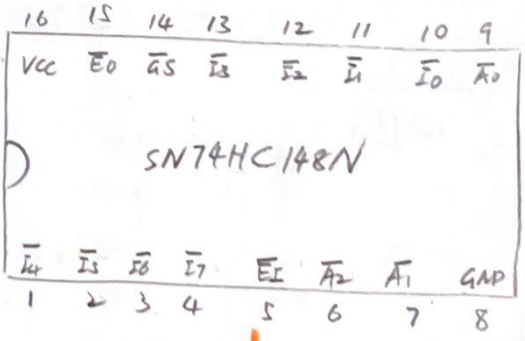
0	0	0	0
0	0	1	1
0	1	1	3
0	1	0	2
1	1	0	6
1	1	1	7
1	0	1	5
1	0	0	4

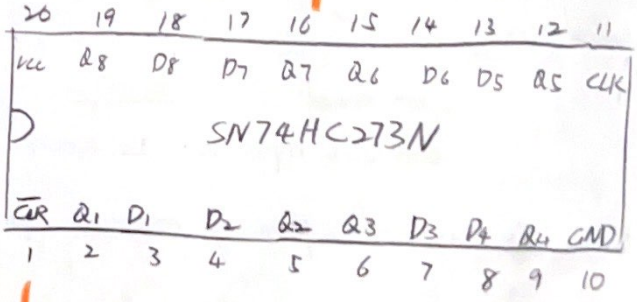
148 273 138



3-bit number







56

FSM next state. EN DE finds nex state.

decoder

0
1
2

encoder

1 state on decoder. 6 (next state is 1.)

3 stat.

011

keeps going.

next state.

current

none is inverted.

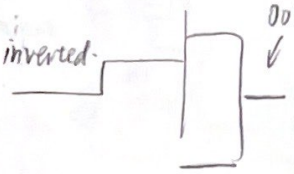
in 000 → 001.

active ~~high~~ low

c only the active one is low.

actualy want it to be 110

this is 6.

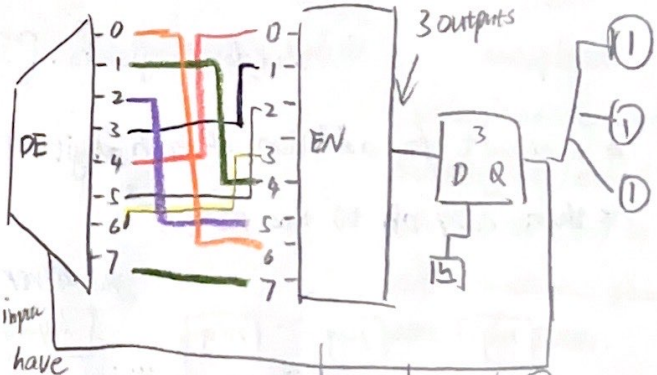


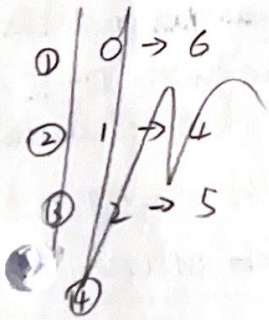
encoder sees 6,

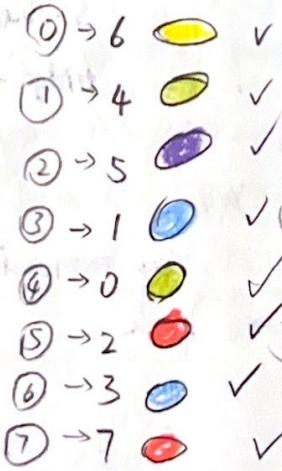
it outputs 001.

active low. invert

0	✓	0	0	0	0	1	1	1	7
1	✓	0	0	1	1	1	1	0	6
2	✓	0	1	1	3	1	0	0	4
3	✓	0	1	0	2	1	0	1	5
4	✓	1	1	0	6	0	0	1	1
5		1	1	1	7	0	0	0	0
6	✓	1	0	1	5	0	1	0	2
7	✓	1	0	0	4	0	1	1	3







wont have