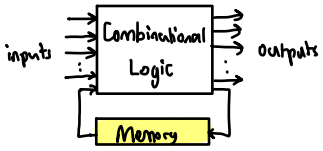
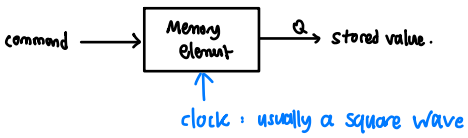
Sequential Logic

- Combinational Circuit
  - each output depends entirely on the immediate (present) inputs.
- Sequential Circuit
  - each output depends on both present inputs and (previous) state.



  - 2 types
    - Synchronous – output change only at specific time
    - Asynchronous – output change at any time.
  - Multivibrator – a class of sequential circuit
    - ✓ Bistable (2 stable states)
    - Monostable or one-shot (1 stable state)
    - Astable (no stable state).
  - Bistable logic devices
    - eg. Latches and flip-flops
    - differ in the methods used for changing their state.

- Memory Element
  - Defn: a device which can remember value indefinitely, or change value on command from its inputs.

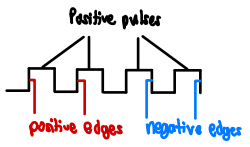


- Characteristic table:

Command (at time t)	Q(t)	Q(t+1)
Set	X	1
Reset	X	0
Memorise/ no change	0	0
	1	1

} no change.

Q(t) or Q : current state
(curr. val. that mem. elem. remembers.)
Q(t+1) or Q^+ : next state



- Command is usually only effective at a particular time (i.e. rising/falling edge / high/low pulse)

Edge-triggered
- Flip-flops
- positive: ON = 0 to 1, OFF = others
- negative: ON = 1 to 0, OFF = others

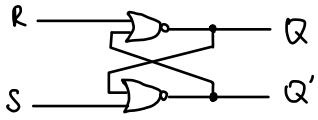
Pulse-triggered
- Latches
- ON = 1, OFF = 0.

S-R Latch

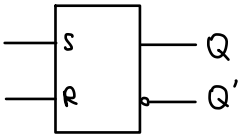
- 2 inputs: Set and Reset.
- 2 outputs: Q and Q' (complimentary)
  - ↳ HIGH: Set state
  - ↳ LOW: Reset state

- For active-high input S-R latch (aka NOR gate latch)
  - R = HIGH, S = LOW → Q becomes LOW (Reset state)
  - S = HIGH, R = LOW → Q becomes HIGH (Set state)
  - Both R and S are LOW → No change in output Q.
  - Both R and S are HIGH → Outputs Q and Q' are both LOW (Invalid)

    ↳ Drawback:
    invalid conditions exists and must be avoided!



Block diagram:



S	R	Q	Q'	
1	0	1	0	initial
0	0	1	0	(after S=1, R=0)
0	1	0	1	
0	0	0	1	(after S=0, R=1)
1	1	0	0	invalid!

- Characteristic table:

S	R	Q	Q'	
0	0	NC	NC	No change. Latch remained in present state.
1	0	1	0	Latch SET.
0	1	0	1	Latch RESET.
1	1	0	0	Invalid condition.

S	R	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	indeterminate	

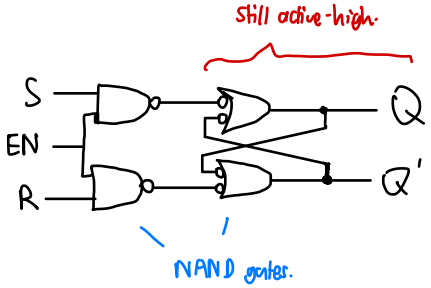
$Q(t+1) = S + R' \cdot Q$

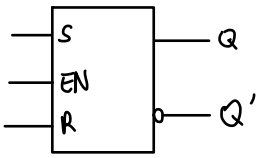
$S \cdot R = 0$

- Active-low S-R Latch: use NAND gates instead.

Gated S-R Latch

- S-R Latch + enable input (EN) and 2 NAND gates
  → gated S-R latch

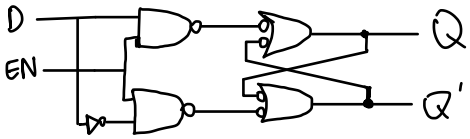


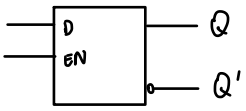


- Output change (if necessary) only when EN is high.

Gated D Latch

- Make input R equal to S' → gated D latch
- D latch eliminates the undesirable condition of invalid state in the S-R latch.





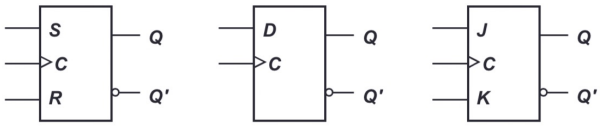
- When EN is high,
  D = HIGH → latch is SET.
  D = LOW → latch is RESET.
- Hence, when EN is high, Q "follows" the D (data) input.
- Characteristic table:

EN	D	Q(t+1)	
1	0	0	Reset
1	1	1	Set
0	X	Q(t)	No change

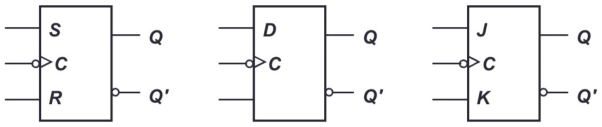
When *EN*=1, ***Q(t+1) = D***

Flip-flops

- synchronous bistable devices
- output changes state at a specified point on a triggering input called the clock
- change state either at the positive (rising) edge, or at the negative (falling) edge of the clock signal.
- S-R flip-flop, D flip-flops, and J-K flip-flops.
- Note the ">" symbol at the clock input.



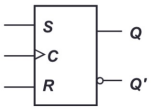
Positive edge-triggered flip-flops



Negative edge-triggered flip-flops

4.1 *S-R* Flip-flop

- *S-R* flip-flop: On the triggering edge (either rising/falling) of the clock pulse,
  - *R* = HIGH and *S* = LOW ➔ *Q* becomes LOW (RESET state)
  - *S* = HIGH and *R* = LOW ➔ *Q* becomes HIGH (SET state)
  - Both *R* and *S* are LOW ➔No change in output *Q*
  - Both *R* and *S* are HIGH ➔Invalid!
- Characteristic table of positive edge-triggered *S-R* flip-flop:



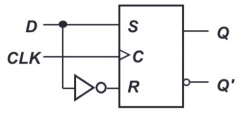
S	R	CLK	Q(t+1)	Comments
0	0	X	Q(t)	No change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	?	Invalid

(Similar to latch.)

X = irrelevant ("don't care")
↑ = clock transition LOW to HIGH (rather than enable.)

4.2 *D* Flip-flop (1/2)

- *D* flip-flop: Single input *D* (data). On the triggering edge of the clock pulse,
  - *D* = HIGH ➔ *Q* becomes HIGH (SET state)
  - *D* = LOW ➔ *Q* becomes LOW (RESET state)
- Hence, *Q* "follows" *D* at the clock edge.
- Convert *S-R* flip-flop into a *D* flip-flop: add an inverter.



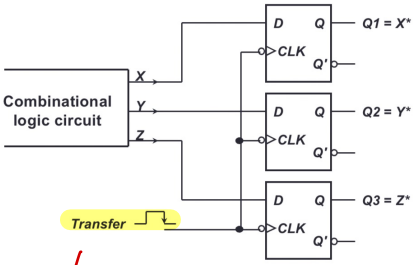
A positive edge-triggered D flip-flop formed with an S-R flip-flop.

D	CLK	Q(t+1)	Comments
1	↑	1	Set
0	↑	0	Reset

↑ = clock transition LOW to HIGH

Application: Parallel data transfer

To transfer logic-circuit outputs X, Y, Z to flip-flops Q1, Q2 and Q3 for storage.



* After occurrence of negative-going transition

value retains if no pulse

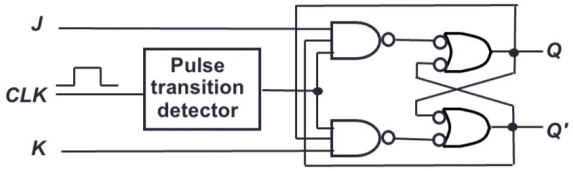
. boost voltage levels so can transfer longer distance.

J-K flip-flops

- Q and Q' are fed back to the pulse-steering NAND gates.
- No invalid state
- Include a toggle state
  - *J* = HIGH and *K* = LOW → *Q* becomes HIGH (SET state)
  - *K* = HIGH and *J* = LOW → *Q* becomes LOW (RESET state)
  - Both *J* and *K* are LOW →No change in output *Q*
  - Both *J* and *K* are HIGH →Toggle

J ≡ S
K ≡ R

- *J-K* flip-flop circuit:



- Characteristic table:

J	K	CLK	Q(t+1)	Comments
0	0		Q(t)	No change
0	1		0	Reset
1	0		1	Set
1	1		Q(t)'	Toggle

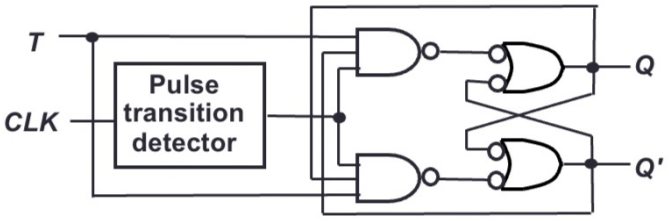
$Q(t+1) = J \cdot Q' + K' \cdot Q$

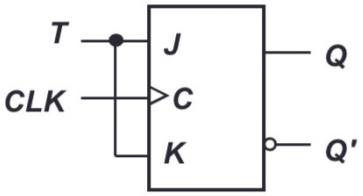
flip state of Q

Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

T flip-flop

Single input version of the J-K flip-flop, formed by tying both inputs together.





- Characteristic table:

T	CLK	Q(t+1)	Comments
0	↑	Q(t)	No change
1	↑	Q(t)'	Toggle

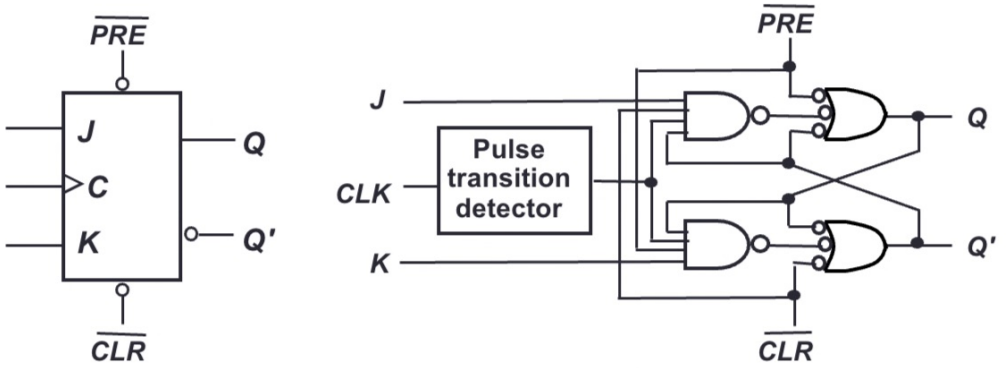
$Q(t+1) = T \cdot Q' + T' \cdot Q$

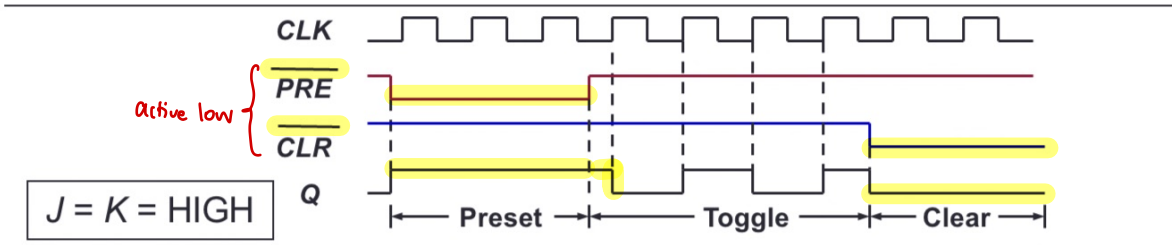
Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

5. Asynchronous Inputs (1/2)

- *S-R*, *D* and *J-K* inputs are synchronous inputs, as data on these inputs are transferred to the flip-flop's output only on the triggered edge of the clock pulse.
- Asynchronous inputs affect the state of the flip-flop independent of the clock; example: *preset* (*PRE*) and *clear* (*CLR*) [or *direct set* (*SD*) and *direct reset* (*RD*)].
- When *PRE*=HIGH, *Q* is immediately set to HIGH.
- When *CLR*=HIGH, *Q* is immediately cleared to LOW.
- Flip-flop in normal operation mode when both *PRE* and *CLR* are LOW.

- A *J-K* flip-flop with active-low PRESET and CLEAR asynchronous inputs.





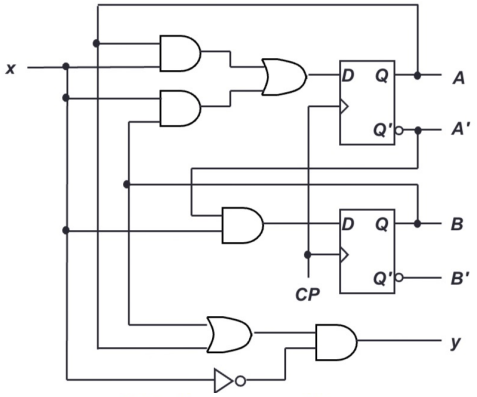
Analyse Sequential Circuits

- By deriving its state table and hence its state diagram
- Requires state equations to be derived for the flip-flop inputs, as well as output function for the circuit outputs that do not come from the flip-flops (if any)
- A(t) and A(t+1) [or A and A^+] to represent the present and next state, respectively, of a flip-flop represented by A.

Example using *D* flip-flops

State equations:
DA = $A^+ = A \cdot x + B \cdot x$
DB = $B^+ = A' \cdot x$

Output function:
$y = (A + B) \cdot x'$



State equations:

$A^+ = A^1 \cdot x^1 + B^1 \cdot x^1$ (or)

$B^+ = A'^0 \cdot x^1$

Output function:

$y = (A + B) \cdot x'$

$= A^1 \cdot x'^0 + B^1 \cdot x'^0$

State Table

- consisting of all possible binary combinations of present states and inputs.
- Similar to truth table
- Inputs and present state on the left side.
- Outputs and next state on the right side.
- n flip-flops and n inputs → 2^{m+n} rows.

Present State		Input	Next State		Output
A	B	x	A^+	B^+	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Full Table ←

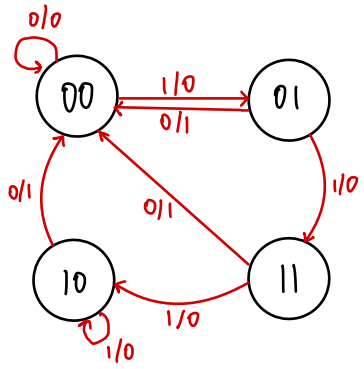
Compact Table ←

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
AB	A^+B^+	A^+B^+	y	y
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0

State Diagram

- Derived from the State Table
- Each state is denoted by a circle
- Each arrow (between 2 circles) denotes a transition of the sequential circuit. (a row in state table)
- A label of the form a/b is attached to each arrow where
  - a (if there is one) denotes the inputs while
  - b (if there is one) denotes the outputs of the circuit in that transition.
- Each combination of the flip-flop value represents a state.
  Hence, m flip-flops → up to 2^m states.
- Eg.

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
AB	A^+B^+	A^+B^+	y	y
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0

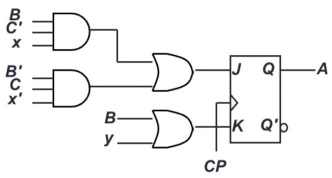


Flip-flop input functions.

- The outputs of a sequential circuit are functions of the present states of the flip-flops and the inputs. These are described algebraically by the circuit output functions.
  - i.e. $y = (A+B) \cdot x'$
- ∴ Inputs are described algebraically by the flip-flop input functions
  - ↳ determines the next-state generation (together with the characteristic table)
- eg.

2 letters: input of flip-flop, name of flip-flop.

$JA = B \cdot C' \cdot x + B' \cdot C \cdot x'$
$KA = B + y$



Full Analysis Example

Given Figure 2, a sequential circuit with two *J-K* flip-flops *A* and *B*, and one input *x*.

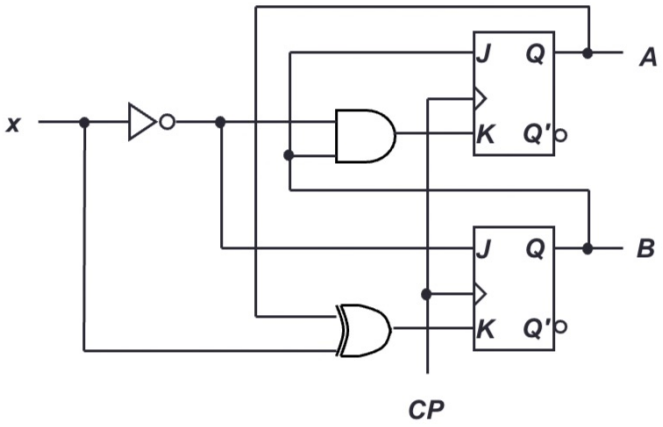


Figure 2

Obtain the flip-flop input functions from the circuit:

$JA = B$ **$JB = x'$** via minterms.

$KA = B \cdot x'$ **$KB = A' \cdot x + A \cdot x' = A \oplus x$** (places where A and x have diff. vals.)

Fill the state table using the above functions, knowing the characteristics of the flip-flops used.

Characteristic Table for J-K ff

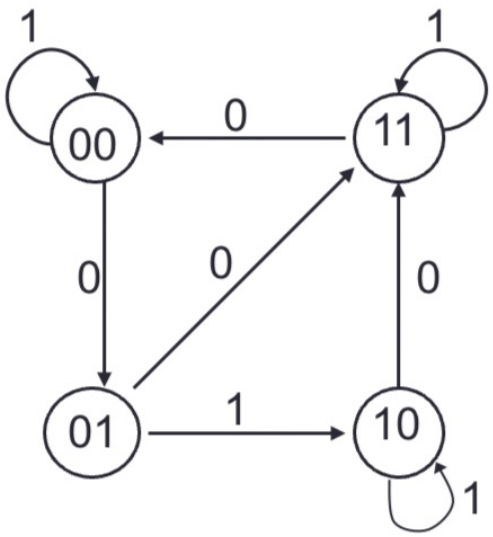
J	K	Q(t+1)	Comments
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q(t)'	Toggle

Present state		Input	Next state		Flip-flop inputs			
A	B	x	A^+	B^+	JA	KA	JB	KB
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

copy

etc

State Diagram:



Full Analysis Example 2

Derive the state table and state diagram of this circuit.

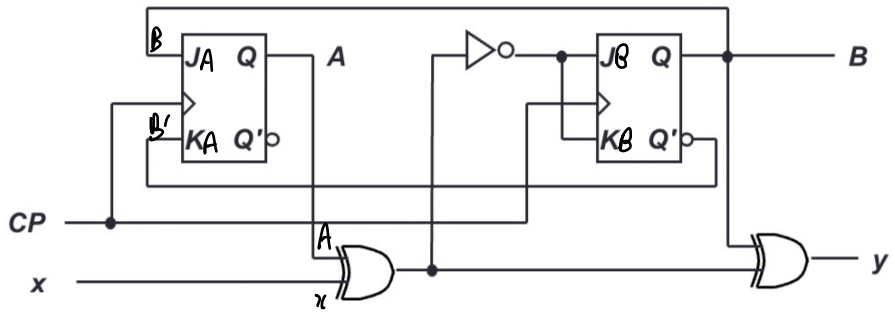


Figure 3

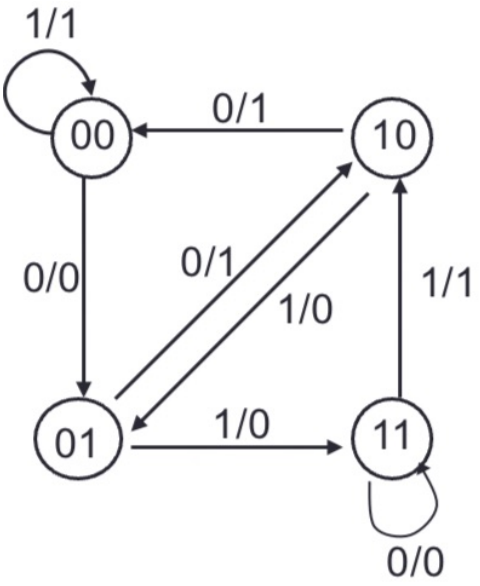
Flip-flop input functions:

via minterms.

$JA = B$ $\qquad JB = KB = (A \oplus x)' = A \cdot x + A' \cdot x'$

$KA = B'$ $\qquad y = B \oplus A \oplus x$

Present state		Input	Next state		Output	Flip-flop inputs			
A	B	x	A^+	B^+	y	JA	KA	JB	KB
0	0	0	0	1	0	0	1	1	1
0	0	1	0	0	1	0	1	0	0
0	1	0	1	0	1	1	0	1	1
0	1	1	1	1	0	1	0	0	0
1	0	0	0	0	1	0	1	0	0
1	0	1	0	1	0	0	1	1	1
1	1	0	1	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1	1



Sequential Circuit Design

* Excitation tables – given the required transition from present state to next state, determine the flip-flop input(s).

$Q \rightarrow Q^+$	J	K	
0 0	0	X	M/R
0 1	1	X	S/T
1 0	X	1	R/T
1 1	X	0	S/M

JK Flip-flop

Q	Q^+	S	R	
0	0	0	X	M/R
0	1	1	0	S
1	0	0	1	R
1	1	X	0	S/M

SR Flip-flop

Notation:
M = memorize, 0,0
S = set, 1,0
R = reset, 0,1
T = toggle, 1,1

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

D Flip-flop

Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

T Flip-flop — "toggle"

Design procedure:

- Start with circuit specifications – description of circuit behaviour, usually a state diagram or state table. (or design statement)
- Derive the state table. (if not available)
- Perform state reduction if necessary. (get rid of redundant/repeated states)
- Perform state assignment. (binary values assigned to diff. states)
- Determine number of flip-flops and label them.
- Choose the type of flip-flop to be used.
- Derive circuit excitation and output tables from the state table. (decide what kind of values the flip-flop input is needed) → augmented state tables
- Derive circuit output functions and flip-flop input functions. → using K-maps.
- Draw the logic diagram.

* Self-correcting:

A circuit is **self-correcting** if for some reason the circuit enters into any unused (invalid) state, it is able to transit to a valid state after a finite number of transitions. Is your circuit self-correcting and why?

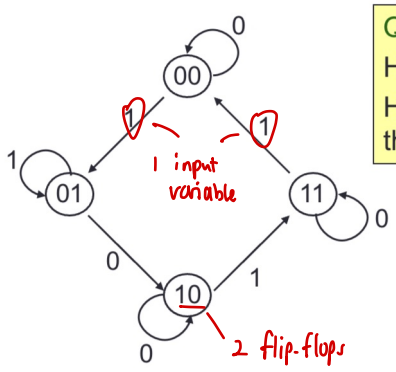
A state is called a ***sink*** if once the circuit enters this state, it never moves out of that state.

* Note: $A \text{ XOR } B = A \cdot B' + A' \cdot B$.

$A \text{ XNOR } B = A \cdot B + A' \cdot B'$

Design Example

Given the following state diagram, design the sequential circuit using *JK* flip-flops.



Questions:
How many flip-flops are needed?
How many input variable are there?

Answers:
Two flip-flops.
Let's call them *A* and *B*.
One input variable.
Let's call it *x*.

Derive State Table

Present State	Next State x=0	Next State x=1
AB	A^+B^+	A^+B^+
00	00	01
01	10	01
10	10	11
11	11	00

Augmented State Table

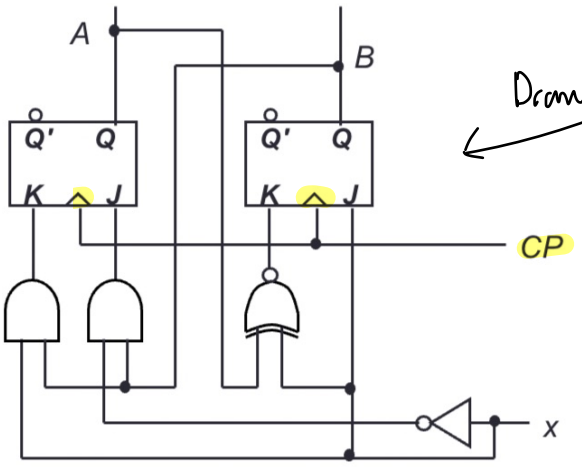
↓ excitation table

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

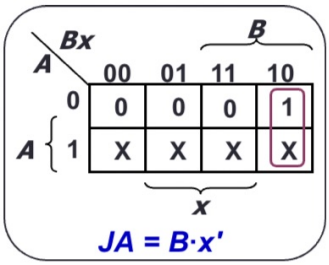
JK Flip-flop's excitation table.

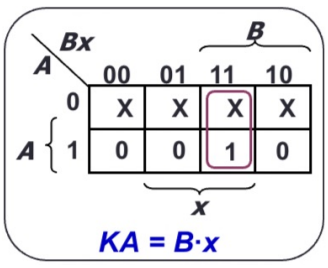
Present state		Input	Next state		Flip-flop inputs			
A	B	x	A^+	B^+	JA	KA	JB	KB
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

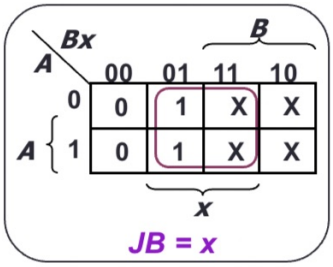
Get flip-flop input functions!
use K-maps!

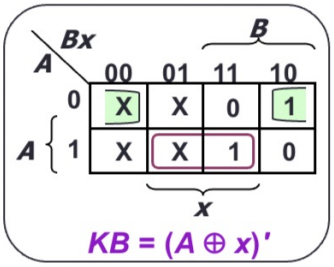


Draw!









$= A' \cdot x' + A \cdot x$

↳ (XNOR) = (XOR)'

Design Example 2

$A^+ = DA$, $B^+ = DB$.

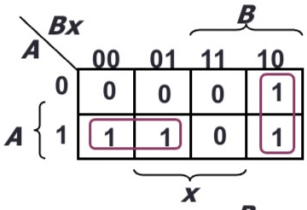
Using *D* flip-flops, design the circuit based on the state table below. (Exercise: Design it using *JK* flip-flops.)

Present state		Input	Next state		Output
A	*B*	*x*	A^+	B^+	*y*
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

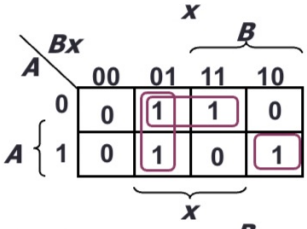
$DA(A,B,x) = \Sigma\, m(2,4,5,6)$
$DB(A,B,x) = \Sigma\, m(1,3,5,6)$
$y(A,B,x) = \Sigma\, m(1,5)$

DA DB ⇒ no need enumerated state table.

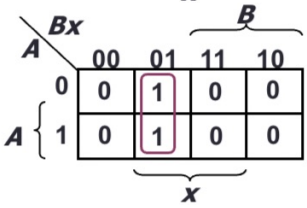
K-maps



$DA = A \cdot B' + B \cdot x'$

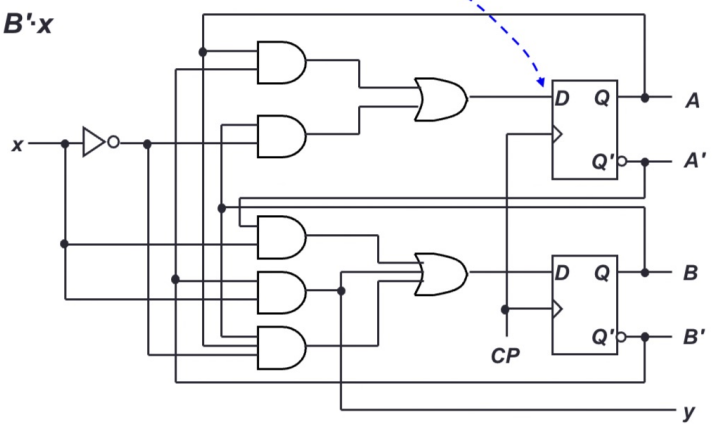


$DB = A' \cdot x + B' \cdot x + A \cdot B \cdot x'$



$y = B' \cdot x$

$DA = A \cdot B' + B \cdot x'$
$DB = A' \cdot x + B' \cdot x + A.B \cdot x'$
$y = B' \cdot x$



Design Example 3 (Missing States)

Present state			Input	Next state			Flip-flop inputs						Output
A	B	C	x	A^+	B^+	C^+	SA	RA	SB	RB	SC	RC	y
0	0	1	0	0	0	1	0	X	0	X	X	0	0
0	0	1	1	0	1	0	0	X	1	0	0	1	0
0	1	0	0	0	1	1	0	X	X	0	1	0	0
0	1	0	1	1	0	0	1	0	0	1	0	X	0
0	1	1	0	0	0	1	0	X	0	1	X	0	0
0	1	1	1	1	0	0	1	0	0	1	0	1	0
1	0	0	0	1	0	1	X	0	0	X	1	0	0
1	0	0	1	1	0	0	X	0	0	X	0	X	1
1	0	1	0	0	0	1	0	1	0	X	X	0	0
1	0	1	1	1	0	0	X	0	0	X	0	1	1

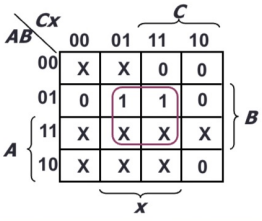
Given these — Desire these

For all unused states - don't care where the flip-flop inputs are

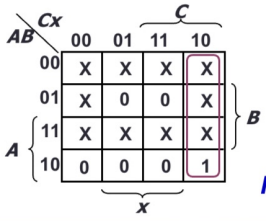
Other unused:
1 1 0
1 1 1

Unused state 000:

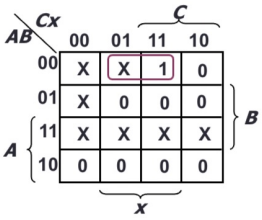
0	0	0	0	X	X	X	X	X	X	X	X	X	X
0	0	0	1	X	X	X	X	X	X	X	X	X	X



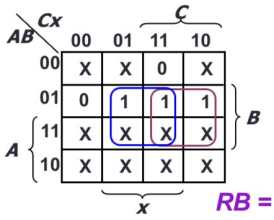
SA = B·x



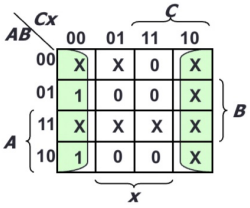
RA = C·x'



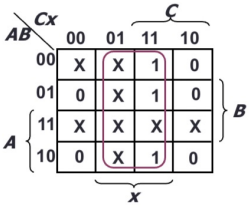
SB = A'·B'·x



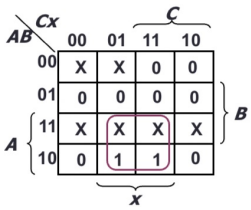
RB = B·C + B·x



SC = x'

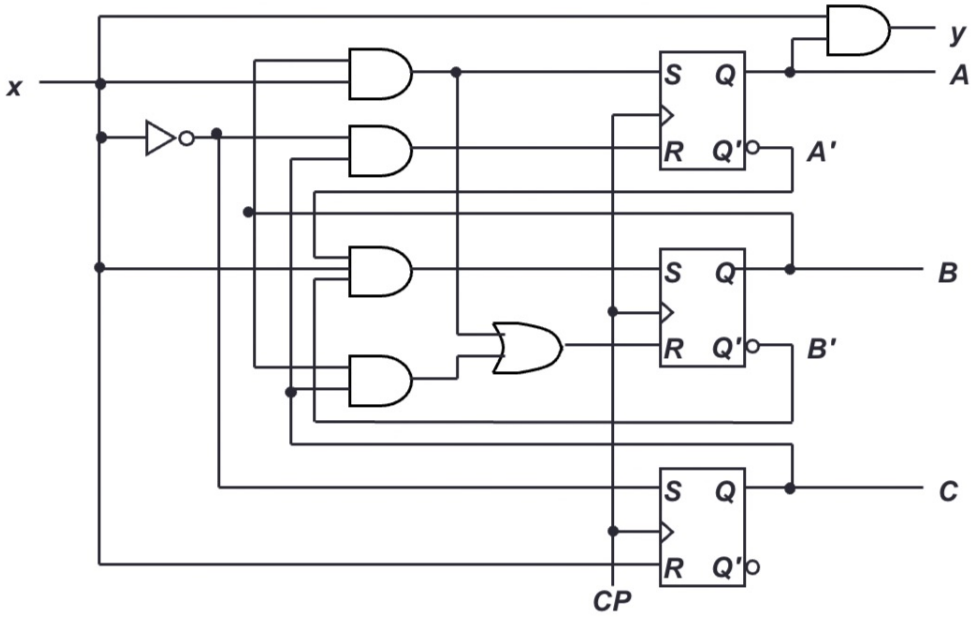


RC = x



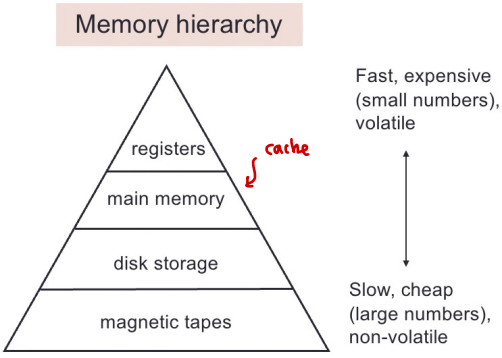
y = A·x

$SA = B \cdot x$	$SB = A' \cdot B' \cdot x$	$SC = x'$	$y = A \cdot x$
$RA = C \cdot x'$	$RB = B \cdot C + B \cdot x$	$RC = x$	



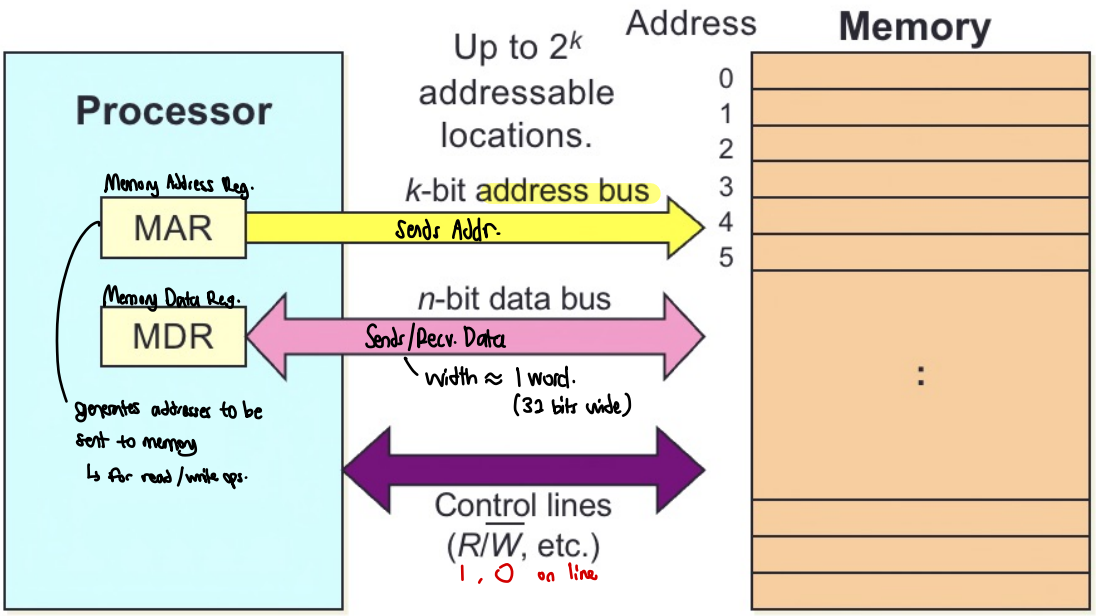
Memory

- Memory stores programs and data
- Defn:
  - 1 byte = 8 bits.
  - 1 word = in multiple of bytes, a unit of transfer btw main memory and registers, usually size of registers.
  - 1 KB = 2^{10} bytes.
  - 1 MB = 2^{20} bytes.
  - 1 GB = 2^{30} bytes
  - 1 TB = 2^{40} bytes.



- Desirable properties: fast access, large capacity, economical cost, non-volatile.
  - ↳ hope for a large amount of memory running reasonably fast

Data transfer



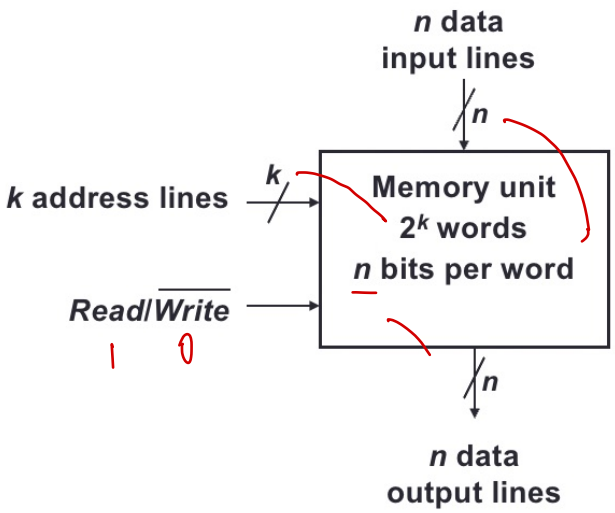
A memory unit stores binary information in groups of bits called *words*.

The data consists of *n* lines (for *n*-bit words). Data input lines provide the information to be stored (*written*) into the memory, while data output lines carry the information out (*read*) from the memory.

The address consists of *k* lines which specify which word (among the 2^k words available) to be selected for reading or writing.

The control lines *Read* and *Write* (usually combined into a single control line *Read/Write*) specifies the direction of transfer of the data.

Block diagram of a memory unit:

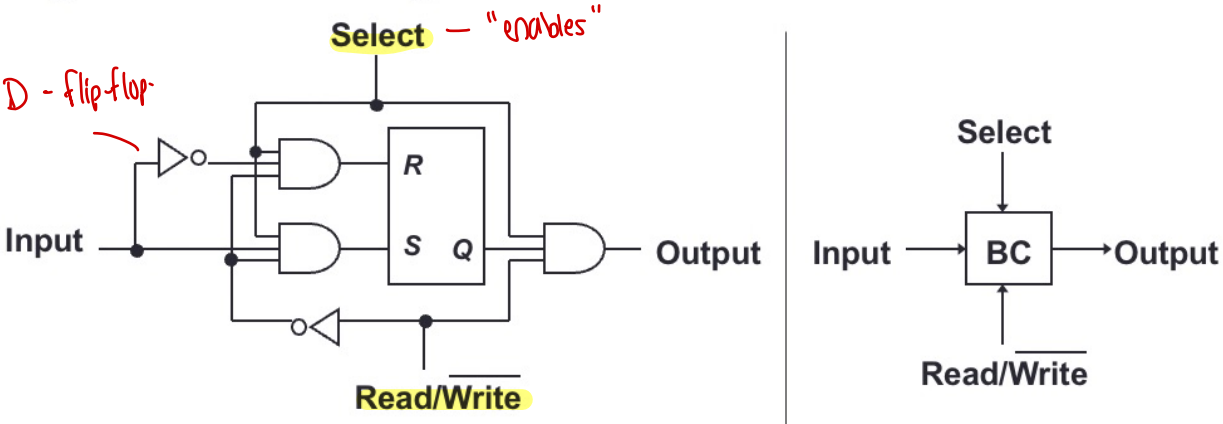


- **Write** operation:
  - Transfers the address of the desired word to the address lines.
  - Transfers the data bits (the word) to be stored in memory to the data input lines.
  - Activates the *Write* control line (set *Read/Write* to 0).
- **Read** operation:
  - Transfers the address of the desired word to the address lines.
  - Activates the *Read* control line (set *Read/Write* to 1).

Memory Enable	*Read/Write*	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

7.3 Memory Cell

- Two types of RAM
  - Static RAMs use flip-flops as the memory cells. → used in caches
  - Dynamic RAMs use capacitor charges to represent data. Though simpler in circuitry, they have to be constantly refreshed. → main system memory. ↘ high data density.
- A single memory cell of the static RAM has the following logic and block diagrams:



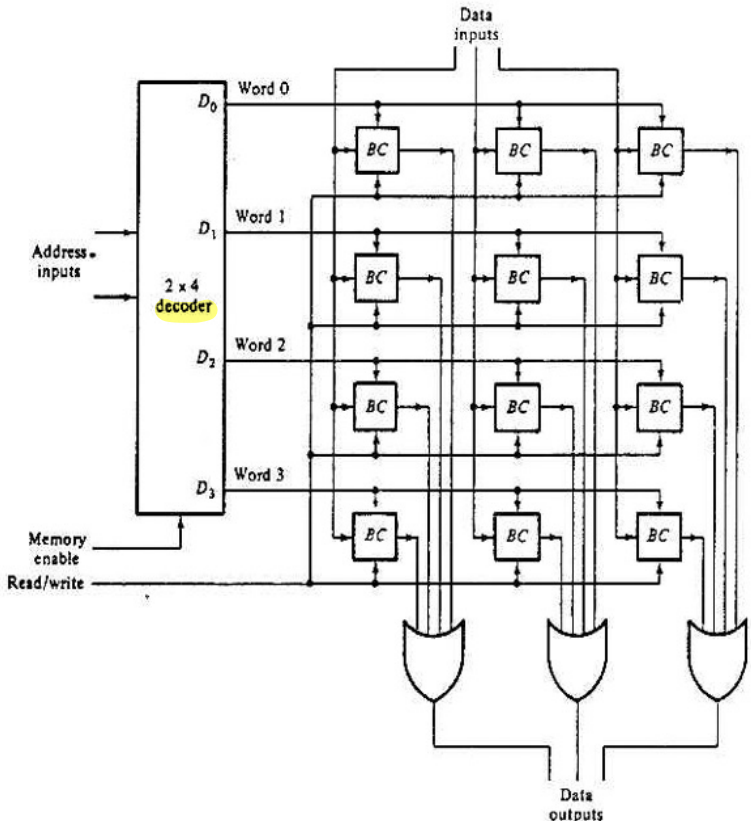
Logic diagram

Block diagram

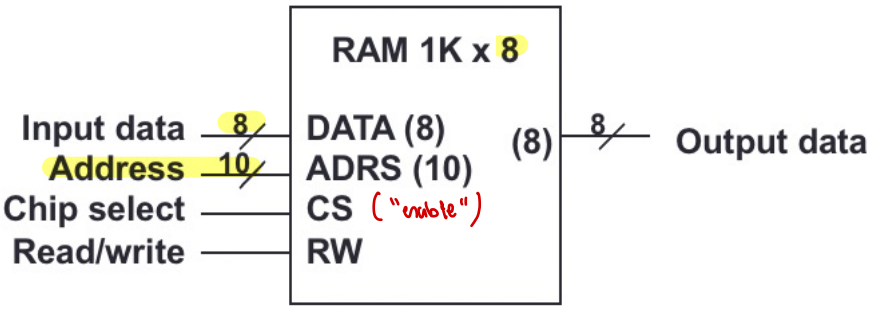
Memory Arrays

- Logic construction of a 4×3 RAM (with decoder and OR gates):

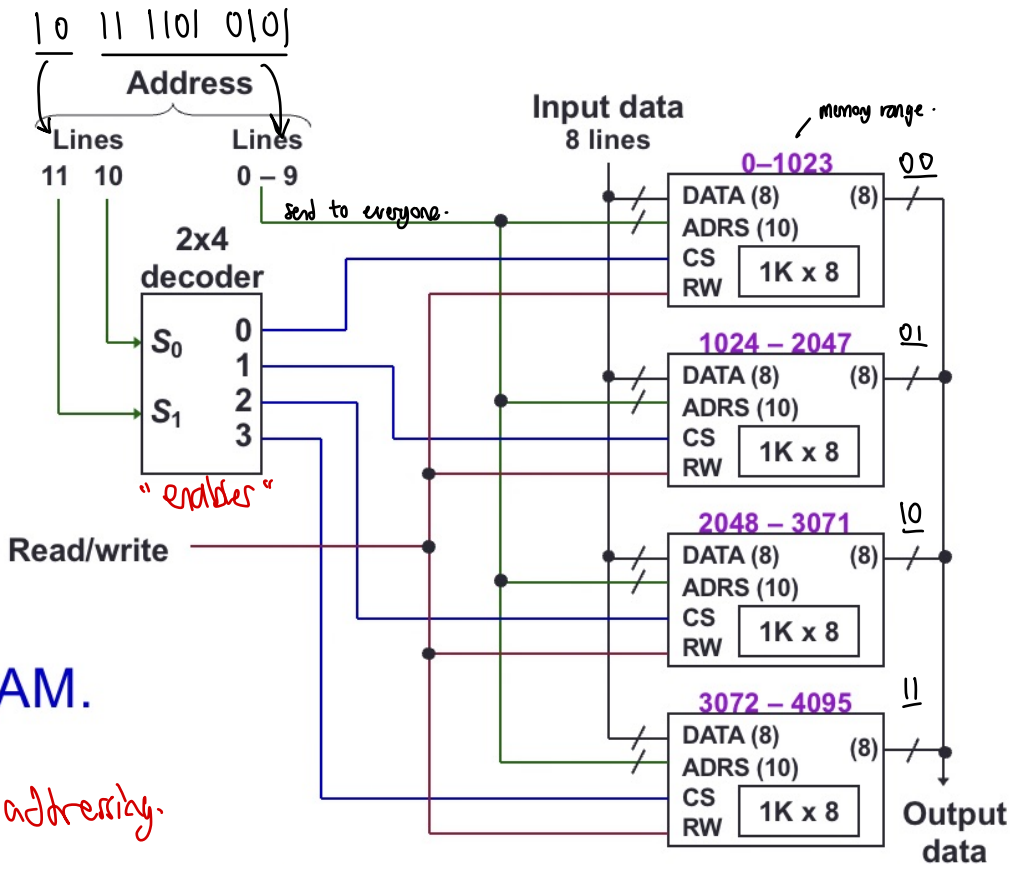
every row 3 bits, total 4 rows.



- An array of RAM chips: memory chips are combined to form larger memory.
- A 1K × 8-bit RAM chip:

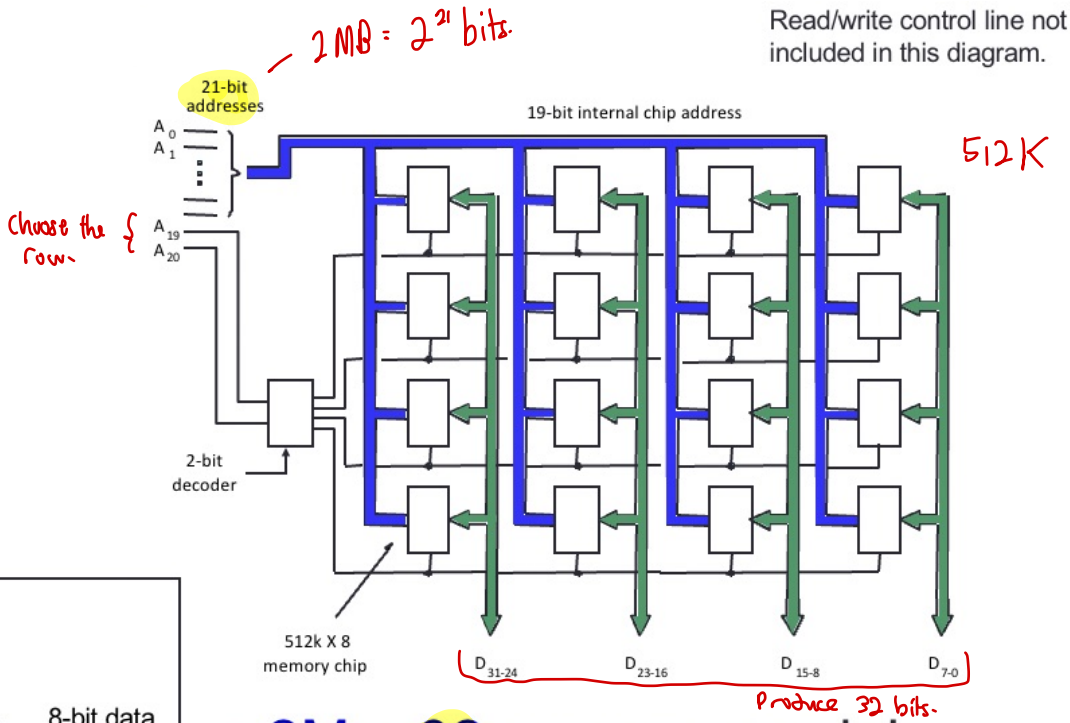


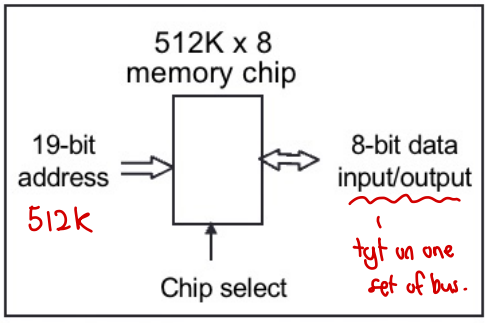
Block diagram of a 1K x 8 RAM chip



- 4K × 8 RAM.

12 bits of addressing.





- 2M × 32 memory module
  - Using 512K × 8 memory chips.

4 x per row ⇒ 4 ram chips