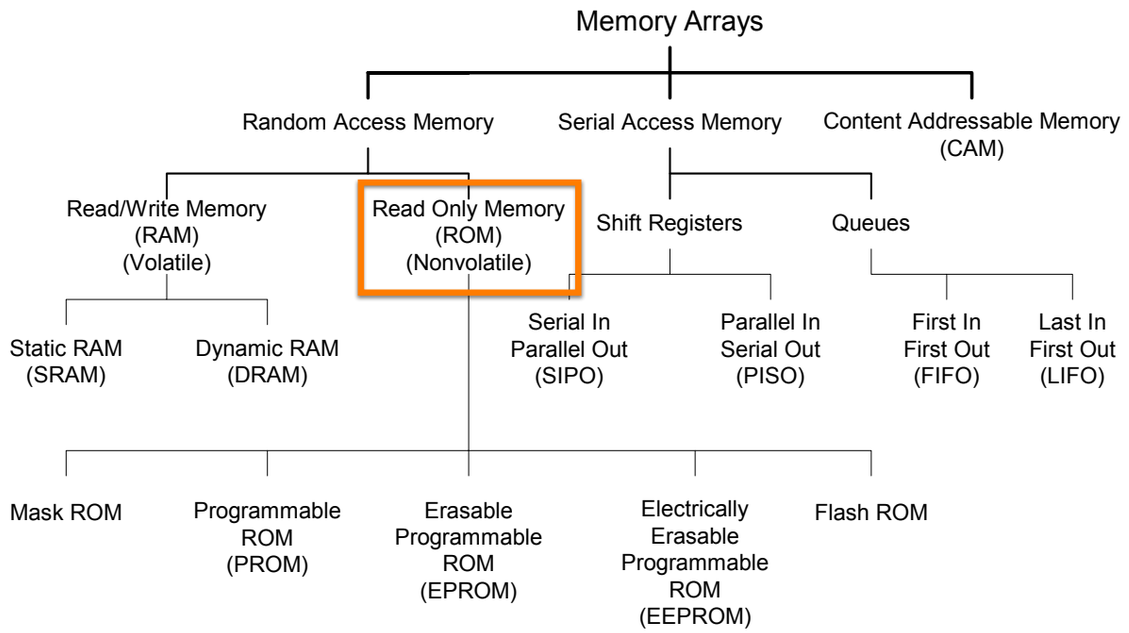
CENG 4480
L09 Memory 2

Bei Yu

Reference:

- Chapter 11 Memories
- **CMOS VLSI Design—A Circuits and Systems Perspective**
- by H.E.Weste and D.M.Harris

Memory Arrays



You might be familiar with this figure, as I covered this one two week ago. Today I will cover CAM & ROM.

[click] ROM is misleading that many of them can be written as well.
Compared with RAM, a more useful classification is volatile ['välətl] or nonvolatile. Volatile memory retains its data as long as power is applied, while nonvolatile memory will hold data.
So ROM is a nonvolatile memory.

[click] CAM determines which address contain that matches specified input data. Essentially, given input data, CAM would determine whether this data is stored, as well as where is the data.

Read-Only Memories

- Read-Only Memories are nonvolatile
  - Retain their contents when power is removed
- Mask-programmed ROMs use one transistor per bit
  - Presence or absence determines 1 or 0

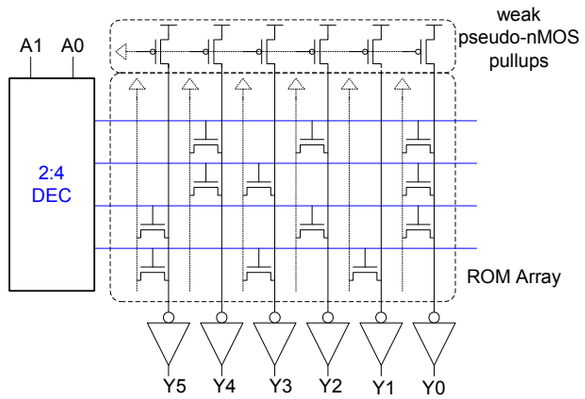
A ROM is a nonvolatile memory structure in that the state is retained indefinitely—even without power. A ROM array is commonly implemented as a single-ended NOR array.

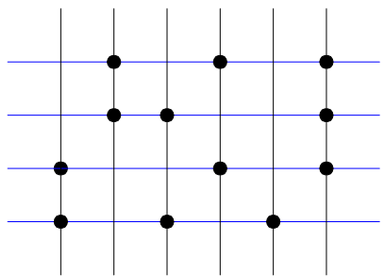
BIOS

NOR ROM

- 4-word x 6-bit NOR-ROM
  - Selected word-line high
  - Represented with dot diagram

```
Word 0: 010101
Word 1: 011001
Word 2: 100101
Word 3: 101010
```





Looks like 6 4-input pseudo-nMOS NORs

It's also called NOR ROM.

1) the selected word-line is pre-charged to high
2) if there is a nmos transistor on the word-line, corresponding bit-line would be discharged to low

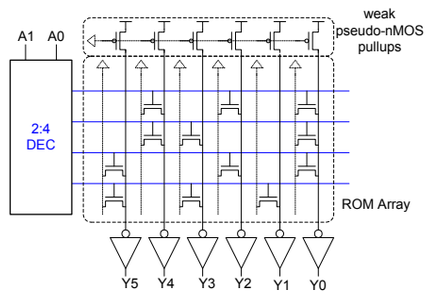
[Analyze] word 0 - 3

The contents of the ROM can be symbolically represented with a dot diagram in which dots indicate the presence of 1s, as shown in Figure 12.53. The dots correspond to nMOS transistors connected to the bitlines, but the outputs are inverted.

EX: NOR ROM

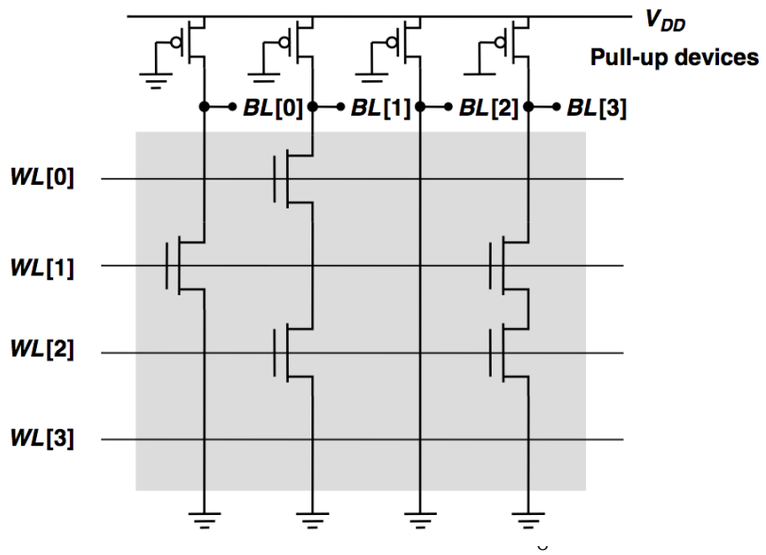
- Draw 4-word 4-bit NOR-ROM structure and dot diagram

```
Word 0: 0100
Word 1: 1001
Word 2: 0101
Word 3: 0000
```



NAND ROM

- 4-word x 4-bit NAND-ROM
  - All word-lines high with exception of selected row



[Analyze] Words 0 - 3: 0100, 1001, 0101, 0000

All the word-lines are pre-charged to high, except the selected row.

1) If one nmos transistor on the row, means this n-transistor is OFF. Then corresponding BL would be 1.
2) If no nmos transistor on the row, the transistors on other row are ON, BL would be discharged to 0.

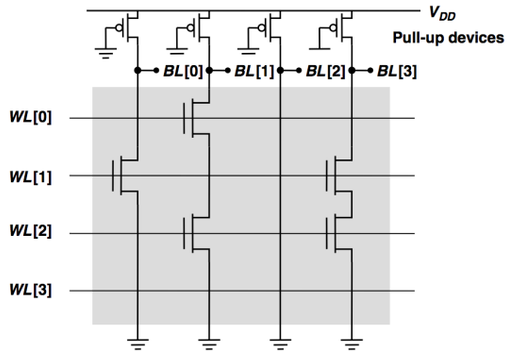
[Draw] NOR: (+) faster, (-) more expensive

NAND: (+)higher density (no contact to VDD/GND)

(-) slower (delay grows quadratically with the number of series transistors discharging the bitline. NAND structures with more than 8–16 series transistors become extremely slow)

EX. NAND ROM

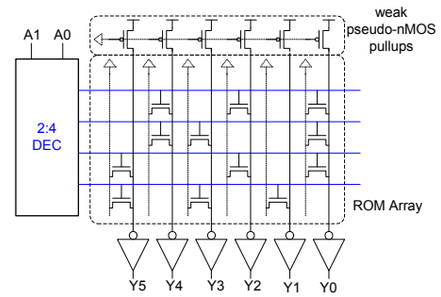
- What's it function?




```
WL[0]=0:
WL[1]=0:
WL[2]=0:
WL[3]=0:
```

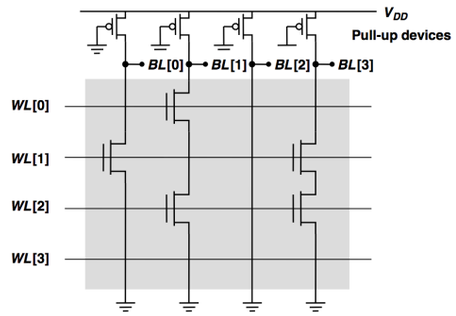
NOR ROM v.s. NAND ROM

- NOR ROM:
  - (+) Faster
  - (-) Larger Area (VDD lines)



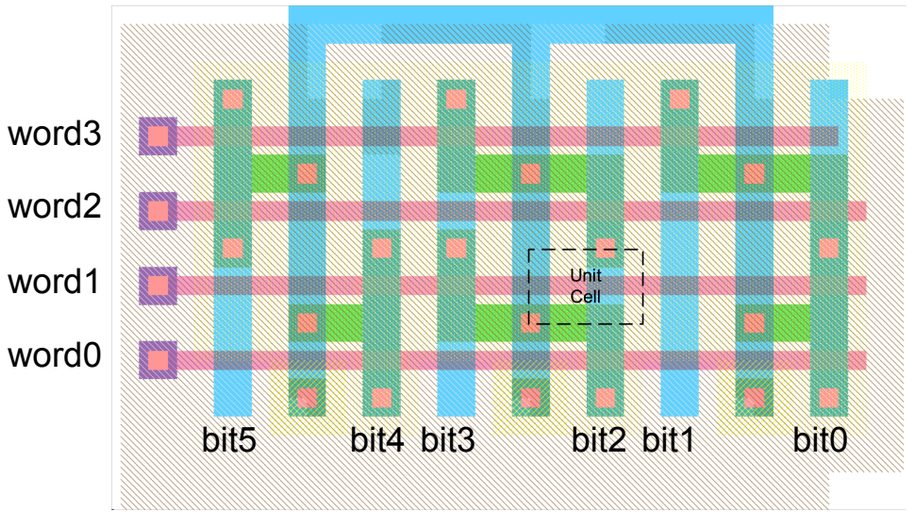
- NAND ROM:
  - (+) High density, small area
  - (-) Slower

delay grows quadratically with the number of series transistors discharging the bitline.



NOR ROM Array Layout*

- Unit cell is 12 x 8 λ (about 1/10 size of SRAM)



pseudo-nMOS ROM

red: poly

dark green: nmos

light green: substrate contacts

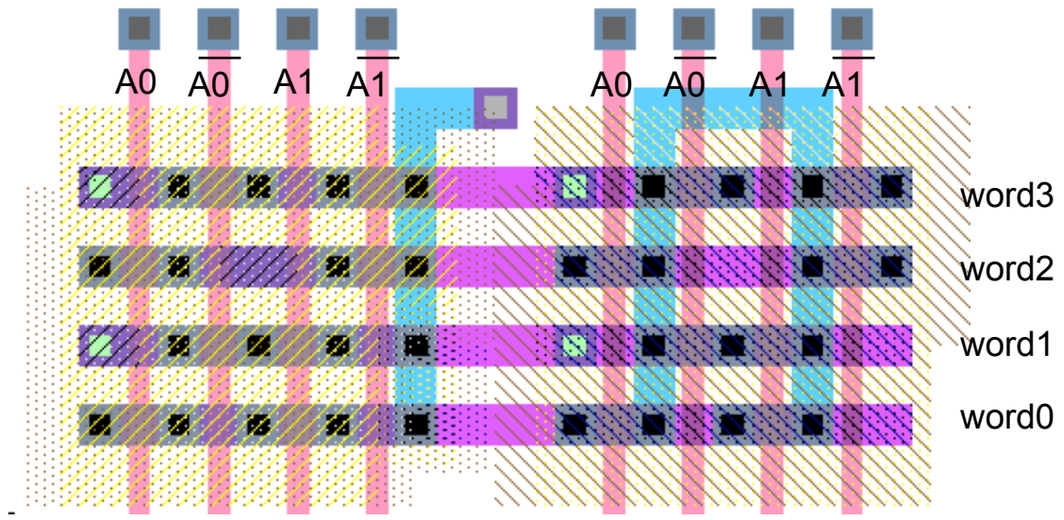
blue: metal

* encoding method is quite important

7 x 8 for NAND ROM

Row Decoders*

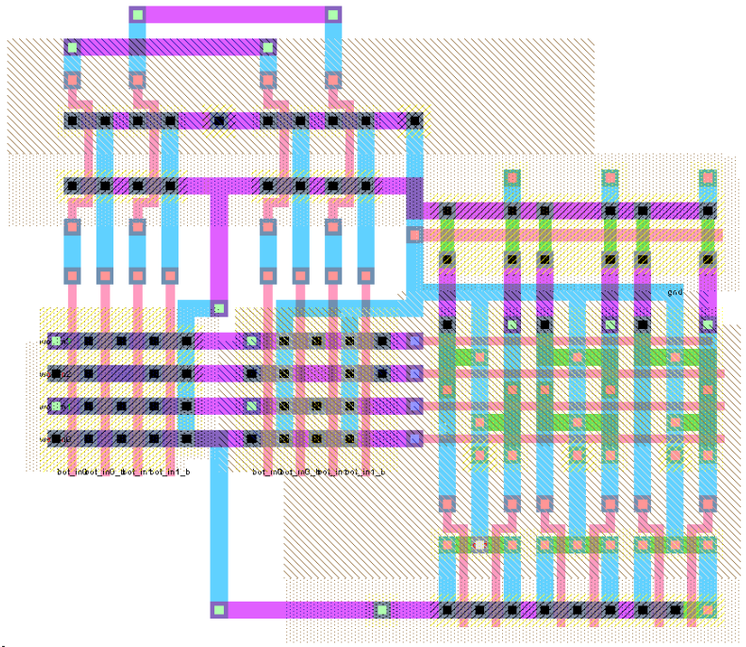
- ROM row decoders must pitch-match with ROM
  - Only a single track per word!



Similar to that in SRAM, the decoder must be pitch-matched to the ROM array. That is, the height of each decoder gate must match the height of the row it drives.

This figure shows a layout on a pitch that is tighter and independent of the number of inputs.
The blue lines are Metal-1, read lines are poly.
We can see that this is a kind of NOR gate structure, the left pMOS transistors are connected in serious, while the nMOS transistors are connected in parallel.

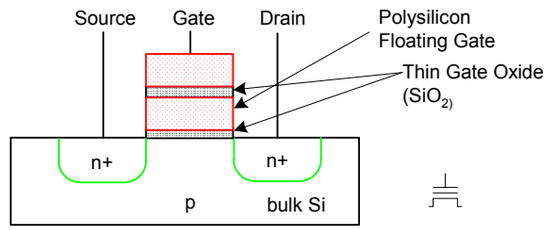
Complete ROM Layout*



a complete pseudo-nMOS ROM including row decoder, cell array, pMOS pullups, and output inverters.

PROMs and EPROMs*

- Programmable ROMs
  - Build array with transistors at every site
  - Burn out fuses to disable unwanted transistors
- Electrically Programmable ROMs
  - Use floating gate to turn off unwanted transistors
  - EPROM, EEPROM, Flash



* Programmable ROMs can be fabricated as conventional ROMs fully populated at every site. The user typically configures the ROM in a specialized PROM programmer before putting it in the system. As there is no way to repair a blown fuse, PROMs are also referred to as one-time programmable memories.

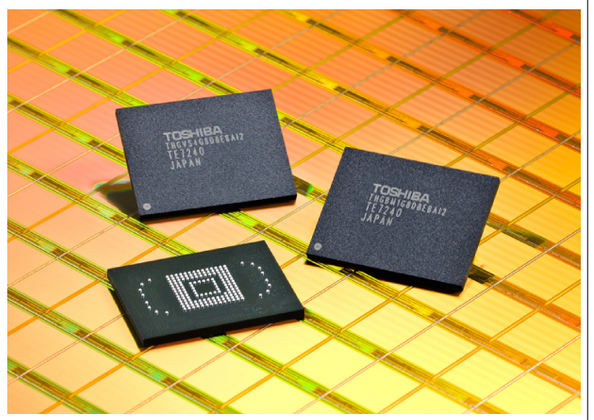
* We can see this structure is similar to a traditional MOS device, except that an extra poly strip [strip] is inserted between the gate and channel. On top is the control gate [CG], as in other MOS transistors, but below this there is a floating gate [FG].
* [draw] systematic symbol

* XXX. Applying a high voltage to the control gate causes electrons to jump through the thin oxide onto the floating gate. Injecting the electrons induces a negative voltage on the floating gate, effectively increasing the threshold voltage to the point that this transistor is always OFF.

Similar structure can be extended to EEPROM and the Flash memory.

NOR / NAND Flash Memory*

- NOR flash: Intel 1988
- NAND flash: Toshiba 1989



[Toshiba'08]

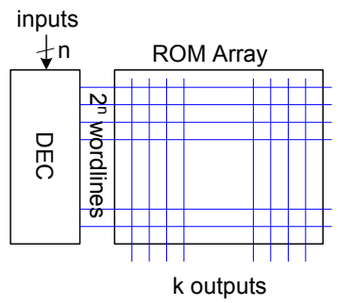
- NOR: faster, more expensive
- NAND: higher density

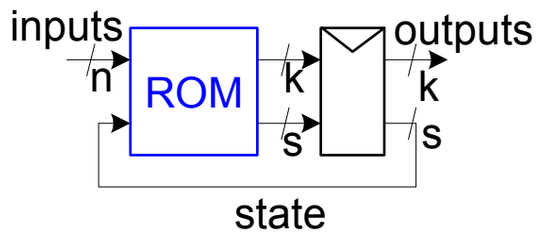
* NOR flash was first introduced by Intel in 1988. NAND flash was introduced by Toshiba in 1989. The two chips work differently. NAND has significantly higher storage capacity than NOR. NOR flash is faster, but it's also more expensive. Some mobile devices use both NAND and NOR. A pocket PC, for instance, may use embedded NOR to boot up the operating system and a removable NAND card for all its other memory/storage requirements. Generally speaking, however, when someone talks about a flash solid state drive, they are referring to NAND flash memory.
* It shall be noted that Flash memory works much faster than traditional EEPROMs because it writes data in chunks, usually 512 bytes [baiz] in size, instead of 1 byte at a time.

* 2008, 32GB NAND chips fabricated with Toshiba's 43nm process

Building Logic with ROMs

- ROM as lookup table containing truth table
  - n inputs, k outputs requires 2^n words x k bits
  - Changing function is easy – reprogram ROM
- **Finite State Machine**
  - n inputs, k outputs, s bits of state
  - Build with 2^{n+s} x (k+s) bit ROM and (k+s) bit reg





Example: RoboAnt

Let's build an Ant

Sensors: Antennae

(L,R) – 1 when in contact

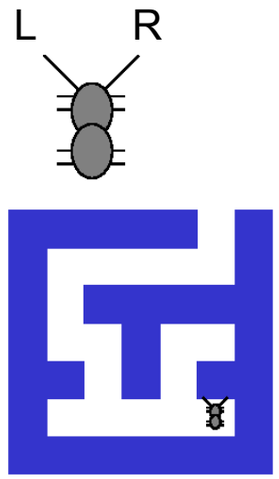
Actuators: Legs

Forward step F

Ten degree turns TL, TR

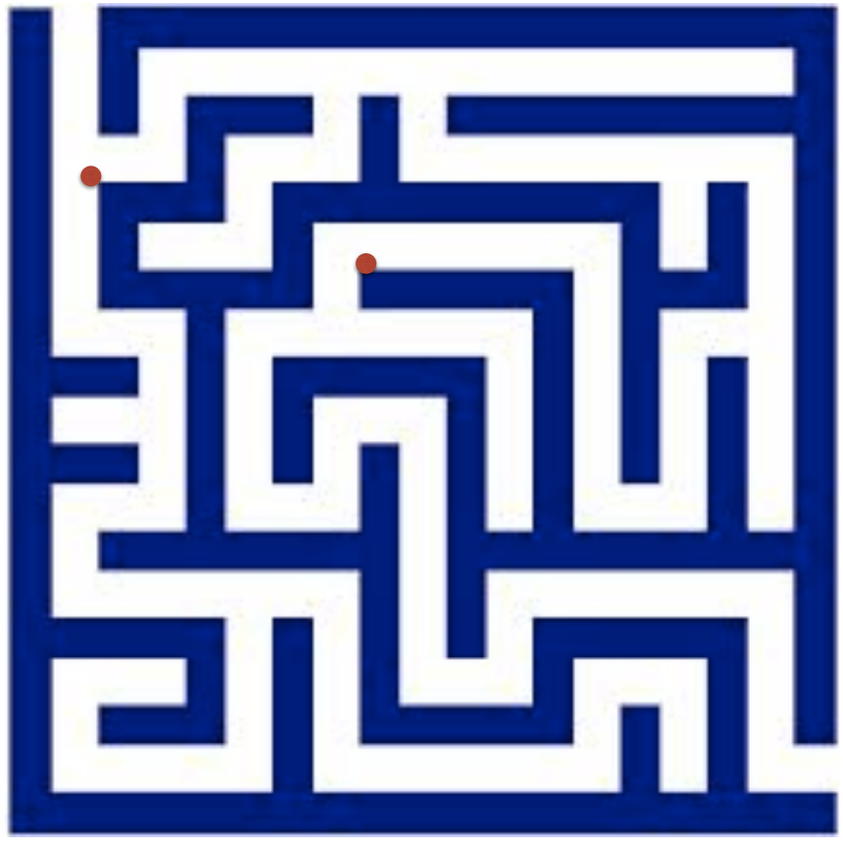
Goal: make our ant smart enough to get out of a maze

Strategy: keep right antenna on wall



(RoboAnt adapted from MIT 6.004 2002 OpenCourseWare by Ward and Terman)

[anˈtenə]



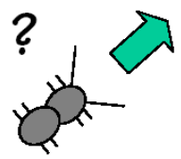
randomly select one initial point

local Vision

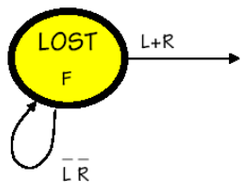
keep right antenna touching the walls

this strategy can guarantee to get out of the maze, but some initial position may be quite ineffective.

Lost in space

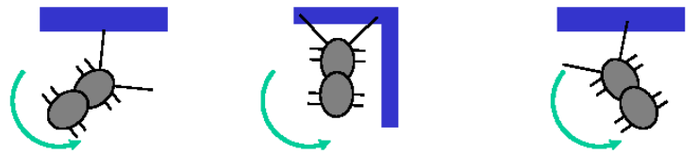


- Action: go forward until we hit something
  - Initial state

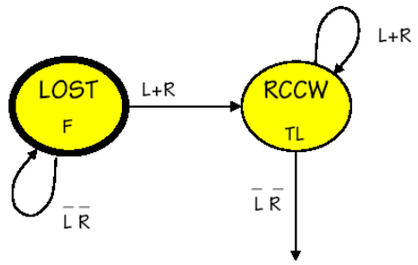


[draw]

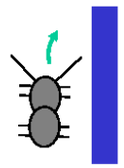
Bonk!!!



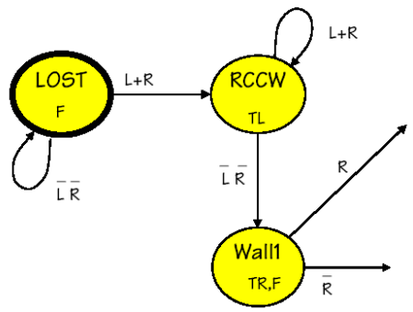
- Action: turn left (rotate counterclockwise)
  - Until we don't touch anymore



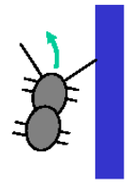
A little to the right



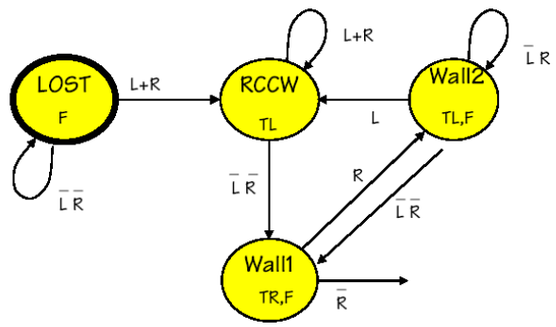
- Action: step forward and turn right a little
  - Looking for wall



Then a little to the right

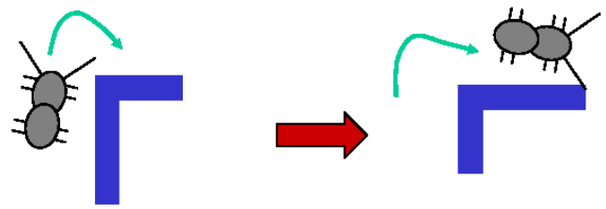


- Action: step and turn left a little, until not touching

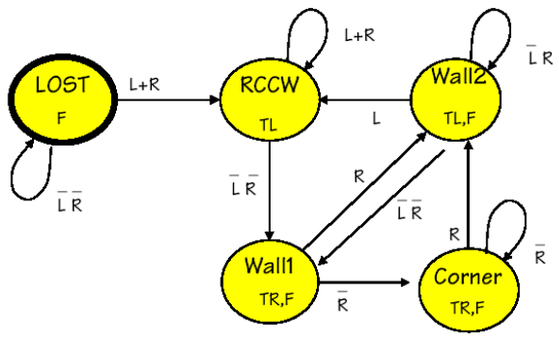


Wall2 if L=1, then left rotate

Whoops – a corner!

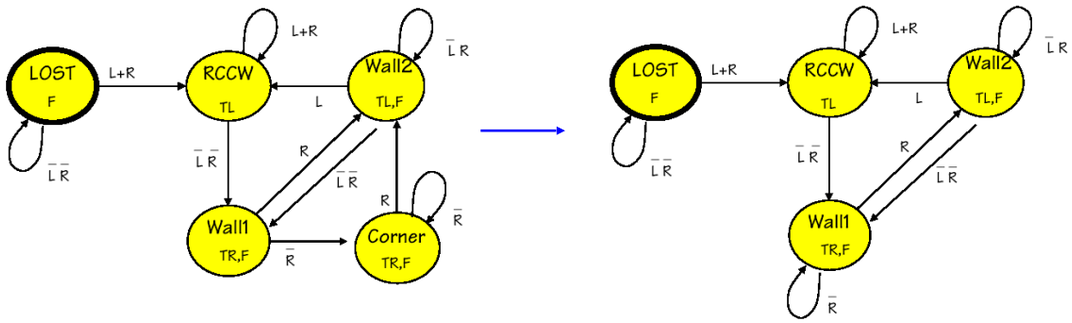


- Action: step and turn right until hitting next wall



Simplification

- Merge equivalent states where possible



1) Identical output behavior on all input strings

2) continue, until

State Transition Table

Current state Inputs Next state Output values

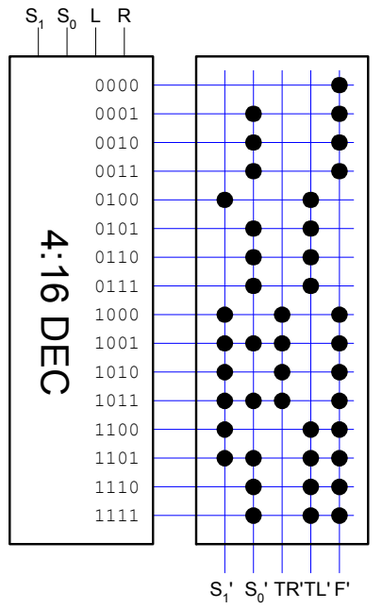
	$S_{1:0}$	L	R	$S_{1:0}$'	TR	TL	F
Lost	00	0	0	00	0	0	1
	00	1	X	01	0	0	1
	00	0	1	01	0	0	1
RCCW	01	1	X	01	0	1	0
	01	0	1	01	0	1	0
	01	0	0	10	0	1	0
Wall1	10	X	0	10	1	0	1
	10	X	1	11	1	0	1
Wall2	11	1	X	01	0	1	1
	11	0	0	10	0	1	1
	11	0	1	11	0	1	1

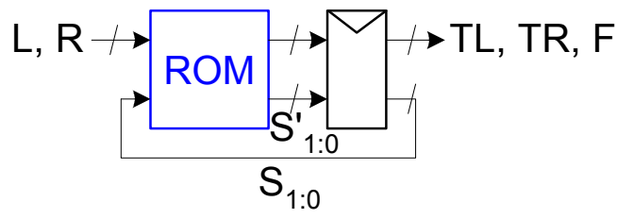
Recall that a state Transition diagram specifies the function of a state machine, not its implementation. Next, we have to convert our specification to gates and registers.

To do so, we rewrite our state transition diagram as a truth table. Each arc of the graph contributes one or more rows to our truth table.

ROM Implementation

- 16-word x 5 bit ROM





one drawback of ROM implementation is that: if there n input signals, there would be 2 to the power of n rows required.

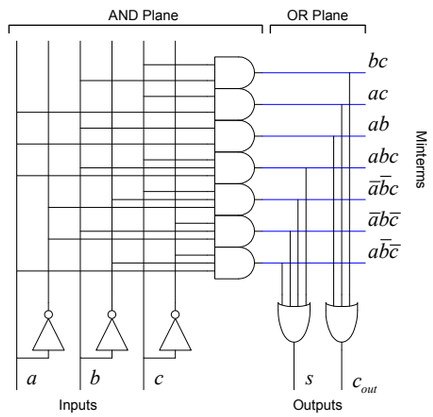
PLAs

- A *Programmable Logic Array* performs any function in sum-of-products form.
- *Literals*: inputs & complements
- *Products / Minterms*: AND of literals
- *Outputs*: OR of Minterms

- Example: Full Adder

$$s = a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + abc$$

$$c_{\text{out}} = ab + bc + ac$$

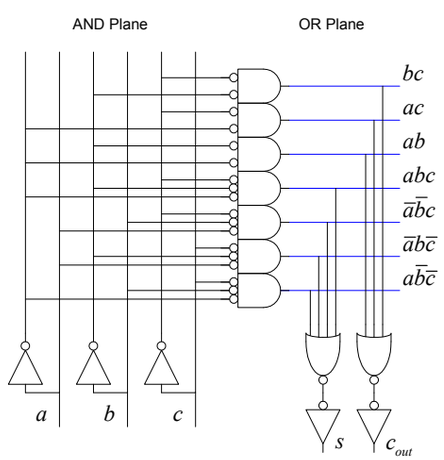


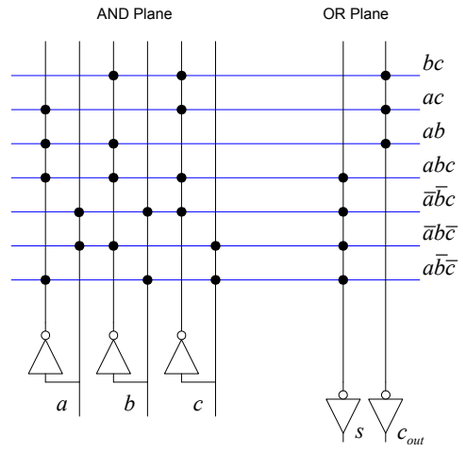
Compared with ROM, PLA can provide an effective implementation to reduce the ROW#.
A PLA provides a regular structure for implementing combinational logic specified in sum-of-products form. It shall be noted that Any logic function can be expressed in sum-of-products form; i.e., where each output is the OR (sum) of the ANDs (products) of true and complementary inputs.

The inputs and their complements are called **literals**. The AND of a set of literals is called a product or **minterm**. The outputs are ORs of minterms. The PLA consists of an **AND plane** to compute the minterms and an **OR plane** to compute the outputs.

NOR-NOR PLAs

- ANDs and ORs not very efficient in CMOS
- Dynamic or Pseudo-nMOS NORs very efficient
- Use DeMorgan's Law to convert to all NORs



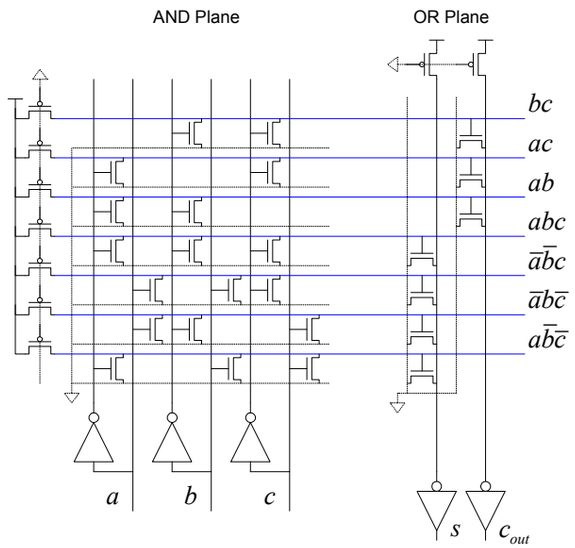


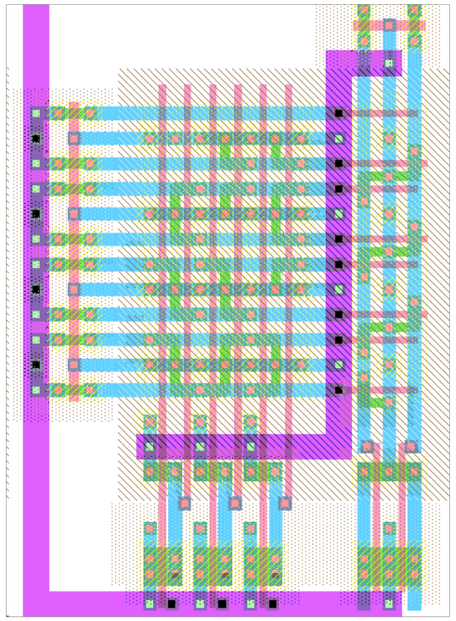
AND/OR are not efficient since we cannot directly implement them in silicon. In other words, more inverters may be required. On the other hand, we want to implement circuit with NAND or NOR gates, because they are generally faster and use fewer components than AND or OR gates.

DeMorgan's law can be used to convert the AND/OR gates to NOR gates. [draw] AND/OR => NOR
We can prove that any logic function can be implemented using only NAND or only NOR gates.

[draw] bc = /(/b + /c)

PLA Schematic & Layout





Both AND plane and OR plane are implemented through the psudo-pMOS NOR gate structures.

[draw]

1) both b=c=0, bit-lines \b \c would be selected
2) both n-transistors are OFF
3) bc line is HIGH
4) c_out before inverter would be discharged to LOW

Layout: nmos, inverters, psudo-pmos pull-up networks, VDD/GND

PLAs vs. ROMs

- The OR plane of the PLA is like the ROM array
- The AND plane of the PLA is like the ROM decoder
- PLAs are more flexible than ROMs
  - No need to have 2^n rows for n inputs
  - Only generate the minterms that are needed
  - Take advantage of logic simplification

RoboAnt PLA*

- Convert state transition table to logic
- Karnaugh map

$S_{1:0}$	L	R	$S_{1:0}$'	TR	TL	F
00	0	0	00	0	0	1
00	1	X	01	0	0	1
00	0	1	01	0	0	1
01	1		01	0	1	0
01	0	1	01	0	1	0
01	0	0	10	0	1	0
10	X	0	10	1	0	1
10	X	1	11	1	0	1
11	1	X	01	0	1	1
11	0	0	10	0	1	1
11	0	1	11	0	1	1

S1′

LR \ S_1S_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	1
10	0	0	0	1

$S_1' =$

S0′

LR \ S_1S_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	0

$S_0' =$

$TR =$

$TL =$

$F =$

If outputs are fed back to inputs through registers,
PLAs also can form finite state machines.

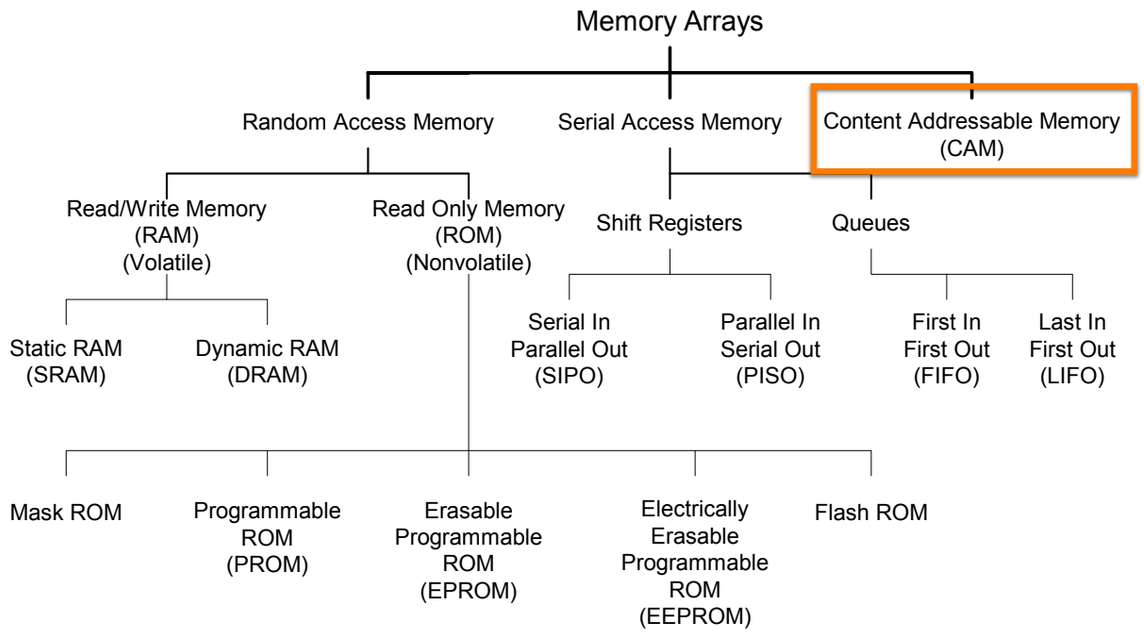
The row and column indices are ordered in Gray code, so that only one variable changes between each pair of adjacent cells. Each cell of the completed Karnaugh map contains a binary digit representing the function's output for that combination of inputs.

Karnaugh is an American scientist, developing the Karnaugh map in 1954 when he was in Bell Labs. Then he worked for IBM for many years. Interestingly, he is famous and IEEE Fellow for the logic simplification and encoding, but he got his PhD degree from Yale University in Physics.

EX. RoboAnt Dot Diagram*

$$S1' = S_1\overline{S_0} + \overline{L}S_1 + \overline{L}\overline{R}S_0$$
$$S0' = R + L\overline{S_1} + LS_0$$
$$TR = S_1\overline{S_0}$$
$$TL = S_0$$
$$F = S_1 + \overline{S_0}$$

Memory Arrays*



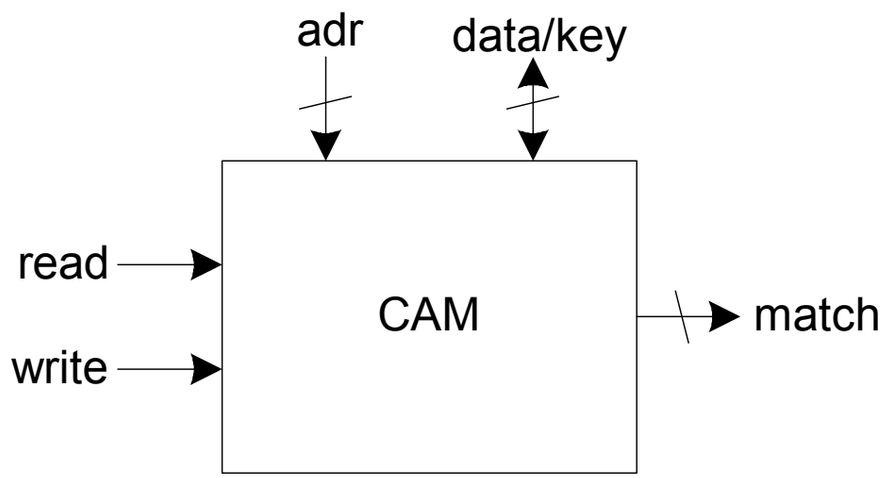
You might be familiar with this figure, as I covered this one two week ago. Today I will cover CAM & ROM.

[click] CAM determines which address contain that matches specified input data. Essentially, given input data, CAM would determine whether this data is stored, as well as where is the data.

[click] ROM is misleading that many of them can be written as well.
Compared with RAM, a more useful classification is volatile ['välətl] or nonvolatile. Volatile memory retains its data as long as power is applied, while nonvolatile memory will hold data.
So ROM is a nonvolatile memory.

CAMs*

- Extension of ordinary memory (e.g. SRAM)
  - Read and write memory as usual
  - Also *match* to see which words contain a *key*

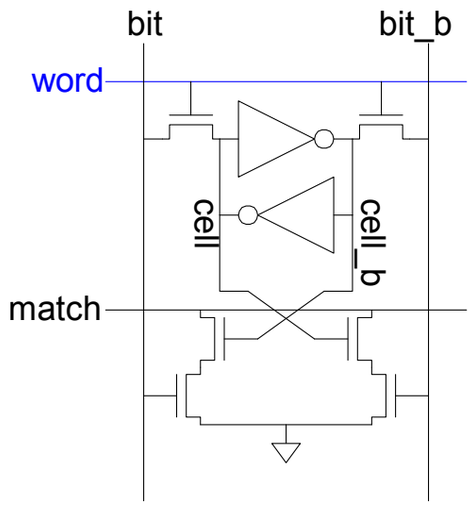


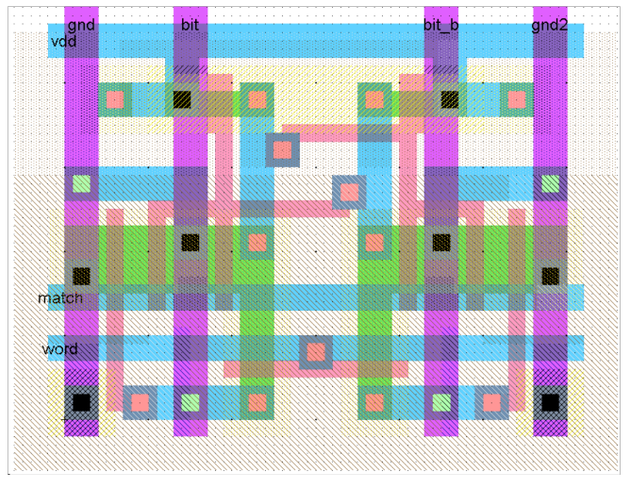
shows the symbol for a content-addressable memory (CAM). The CAM is like a conventional SRAM that can be read or written given adr and data. In addition, CAM also performs matching operations. For each word, if the CAM contains a specified data/key, the related match-line would be high.

[draw] A common application of CAMs is translation look-aside buffers (TLBs) in microprocessors supporting virtual memory. The virtual address is given as the key to the TLB CAM. If this address is in the CAM, the corresponding match-line is asserted. This match-line can serve as the word-line to access a RAM containing the associated physical address.

10T CAM Cell*

- Add four match transistors to 6T SRAM
  - 56 x 43 λ unit cell





10T CAM cell consisting of a normal SRAM cell with additional transistors to perform the match.

[draw]

Read: Precharge bit, bit_b; Raise wordline

Write: Drive data onto bit, bit_b; Raise wordline

Match: 1) Pre-charge match-line, wordline low, 2) drive data onto bit bit_b, 3) if match, match-line down

(if cell=bit=0)

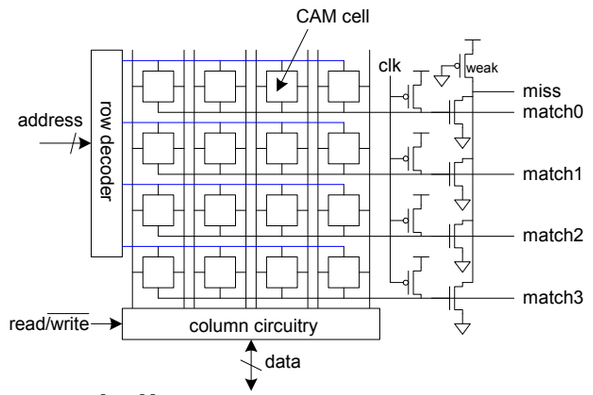
(The match-line is either pre-charged or pulled high as a distributed pseudo-nMOS gate. The key is placed on the bitlines. If the key and the value stored in the cell differ, the match-line will be pulled down.)

The key can contain a “don’t care” by setting both bit and bit_b low.Sometimes the key is provided on separate search-lines rather than on the bitlines to reduce the capacitance and power consumption of a search.

The inside front cover shows a layout of this cell in a 56 × 43 area; CAMs generally have about twice the area of SRAM cells.

CAM Cell Operation*

- Read and write like ordinary SRAM
- For matching:
  - Leave wordline low
  - Precharge matchlines
  - Place key on bitlines
  - Matchlines evaluate
- Miss line
  - Pseudo-nMOS NOR of match lines
  - Goes high if no words match



4 × 4 CAM array. Like an SRAM, it consists of an array of cells, a decoder, and column circuitry. In addition, each row also produces a dynamic match-line. The match-lines are pre-charged with the clocked pMOS transistors.

The miss signal is produced with a distributed pseudo-nMOS NOR.