CENG-232
Logic Design

Week 9 - Memory and Programmable Logic

Memory

- All sequential circuits depend upon the presence of **memory.**
- A memory unit = collection of storage cells + circuits needed to transfer data in and out

Memory

- Two types of memory used in digital systems:
  - RAM: Random Access Memory
  - ROM: Read-Only Memory
- RAM can perform both read and write
- ROM can perform only the read operation
- ROM is also considered a Programmable Logic Device (more on this later)

RAM: Random Access Memory

- Why random-access?
  - The time it takes to transfer information to or from any desired random location is always the same.
  - Compare this with a tape unit.

Bits, bytes and words

- A memory unit stores binary information in groups of bits called **words**.
- A word is an entity of bits that move in and out of storage as a unit.
- A **memory word** may represent
  - a number,
  - an instruction,
  - one or more alphanumeric characters, or
  - any other binary-coded information.
- Most computer memories use words that are multiples of 8 bits in length which is a **byte**.

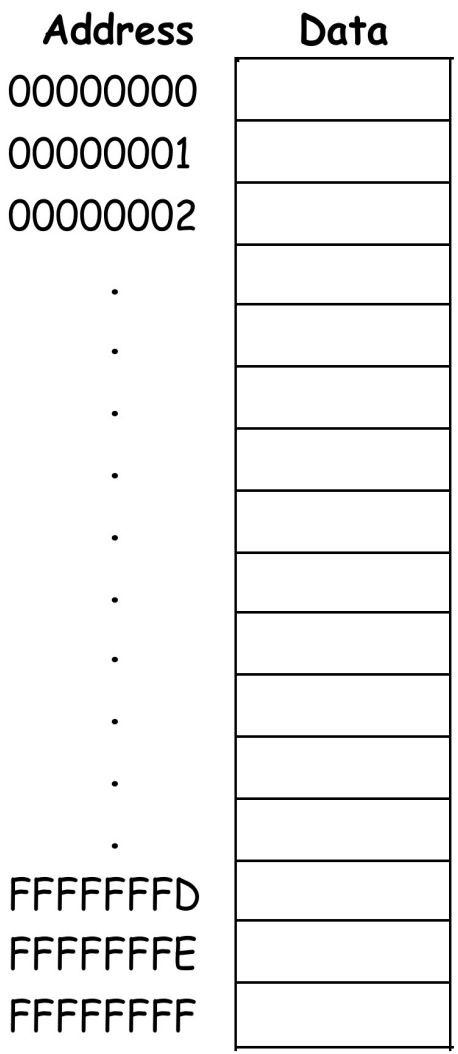
Introduction to RAM

- *Random-Access Memory* (or *RAM*), provides large quantities of temporary storage in a computer system.
- Remember the basic capabilities of a memory:
  - It should be able to *store* a value.
  - You should be able to *read* the value that was saved.
  - You should be able to *change* the stored value.
- A RAM is similar, except that it can store *many* values, and access times for different locations are the same.
  - An *address* will specify which memory value we're interested in.
  - Each value can be a multiple-bit *word* (e.g., 8 bits, 32 bits).
- We'll refine the memory properties as follows:

A RAM should be able to:
- Store many words, one per address
- Read the word that was saved at a particular address
- Change the word that's saved at a particular address

Picture of Memory

- You can think of computer memory as being one big array of data.
  - The address serves as an array index.
  - Each address refers to one word of data.
- You can read or modify the data at any given memory address.
- You've worked with pointers in C (or C++), so you've already worked with memory addresses.



Memory as a Black-Box

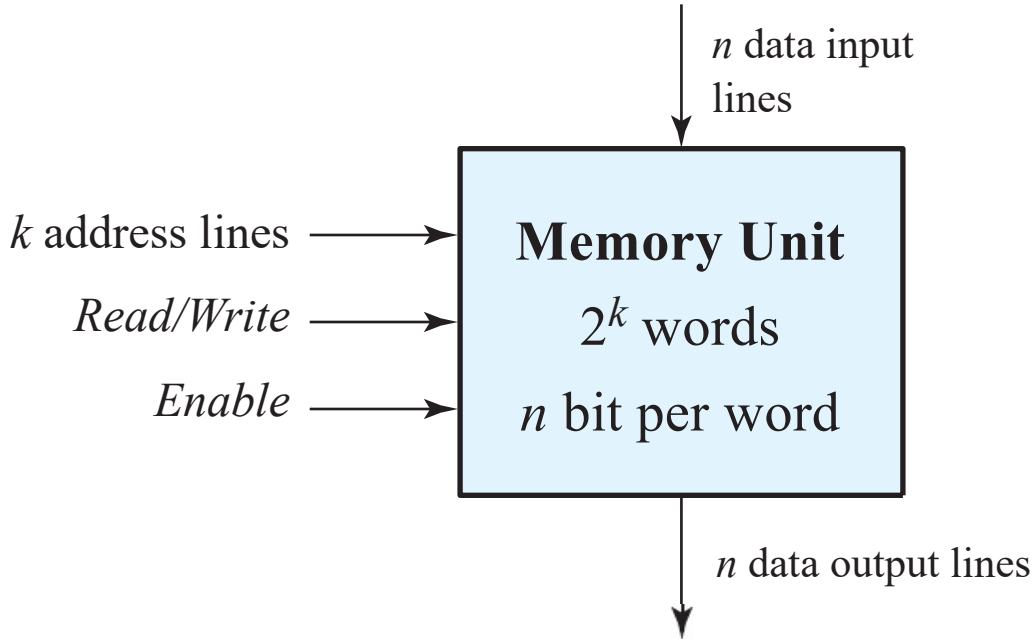


Table 7.1
Control Inputs to Memory Chip

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

The number of address bits needed in a memory

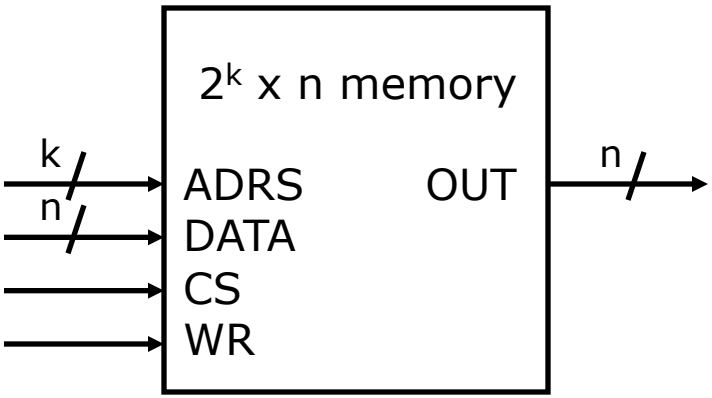
(a) ??? the total number of words that can be stored in the memory

(b) ??? the number of bits in each word.

Replace "???" with "depends on" or "is independent from"

(a) Depends (b) indep

Block diagram of RAM

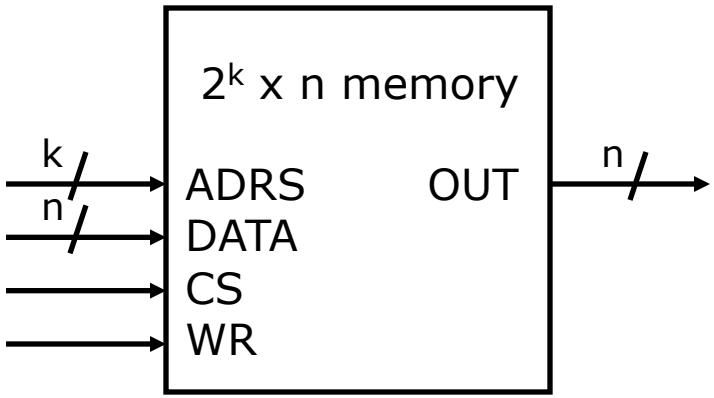


CS	WR	Memory operation
0	x	None
1	0	Read selected word
1	1	Write selected word

- This block diagram introduces the main interface to RAM.
  - A Chip Select, *CS*, enables or disables the RAM.
  - *ADRS* specifies the address or location to read from or write to.
  - *WR* selects between reading from or writing to the memory.
    - To read from memory, WR should be set to 0.
      OUT will be the n-bit value stored at ADRS.
    - To write to memory, we set WR = 1.
      DATA is the n-bit value to save in memory.
- This interface makes it easy to combine RAMs together, as we'll see.

Memory sizes

- We refer to this as a 2^k x n memory.
  - There are k *address lines*, which can specify one of 2^k addresses.
  - Each address contains an n-bit word.



- For example, a 2^{24} x 16 RAM contains 2^{24} = 16M words, each 16 bits long.
  - The RAM would need 24 address lines.
  - The total *storage capacity* is 2^{24} x 16 = 2^{28} bits.

Memory sizes

- Memory sizes are usually specified in numbers of bytes (8 bits), megabytes or gigabytes.
- The 2^{28}-bit memory on the previous page translates into:

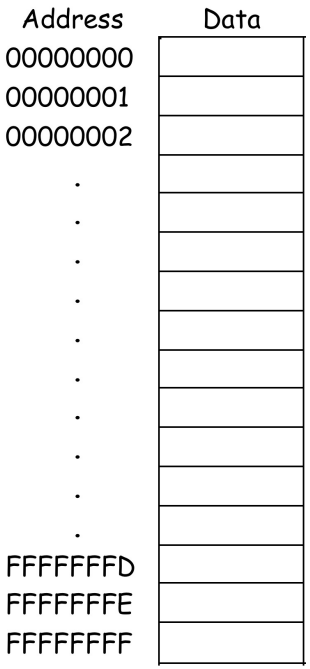
$$2^{28} \text{ bits} / 8 \text{ bits per byte} = 2^{25} \text{ bytes}$$

- With the abbreviations below, this is equivalent to 32 megabytes.

	Prefix	Base 2	Base 10
K	Kilo	$2^{10} = 1{,}024$	$10^3 = 1{,}000$
M	Mega	$2^{20} = 1{,}048{,}576$	$10^6 = 1{,}000{,}000$
G	Giga	$2^{30} = 1{,}073{,}741{,}824$	$10^9 = 1{,}000{,}000{,}000$

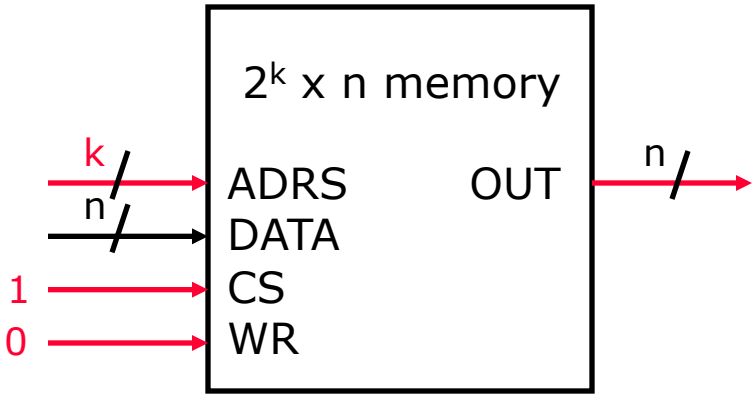
Typical Memory Sizes

- Some typical memory capacities:
  - PCs usually come with 8GB to 16GB of RAM.
  - Smartphones have 2-4GB of memory.
- But… Many operating systems implement *Virtual Memory*, which makes the memory larger than RAM's capacity.
  - If a system uses 32-bit addresses, this works out to 2^{32}, or about four billion, different possible addresses.
  - With a data (word) size of one byte, the result is apparently a 4GB memory.
  - For 64-bit addresses, the theoretical limit is 16 Exabytes. (1 Exabyte = 10^(18) bytes)
  - The operating system uses hard disk space as a substitute for "real" memory (VM).



Reading RAM

- To *read* from this RAM, the controlling circuit must:
  - Enable the chip by ensuring CS = 1.
  - Select the read operation, by setting WR = 0.
  - Send the desired address to the ADRS input.
  - The contents of that address appear on OUT after a little while.
- Notice that the DATA input is unused for read operations.



Memory Read Cycle

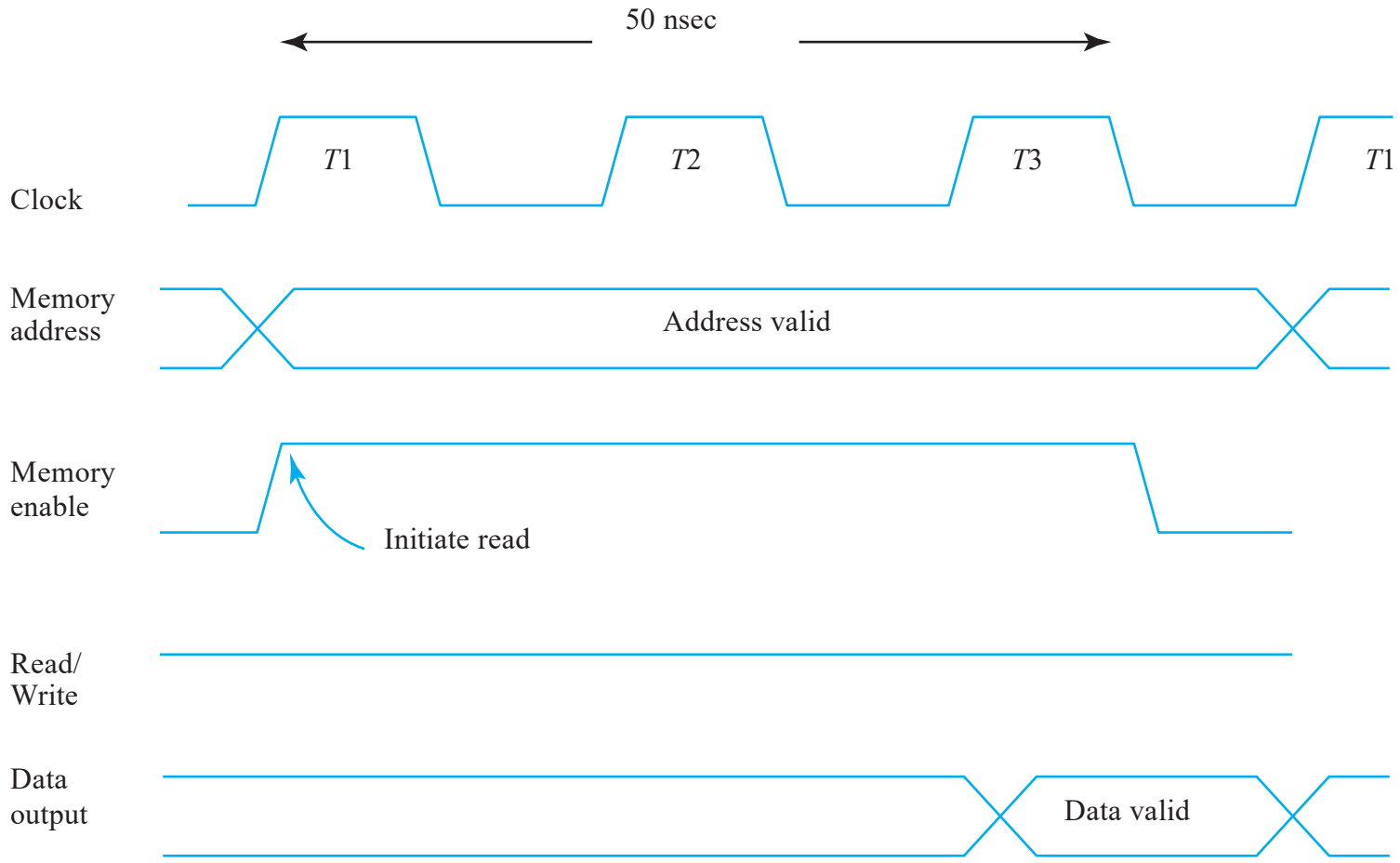
- The memory does not employ an internal clock.
- The operation of the memory unit is controlled by an external device such as a central processing unit (CPU).
- The CPU is usually synchronized by its own clock.
- Memory read and write operations are specified by control inputs.
- The access time of memory is the time required to select a word and **read** it.
- The cycle time of memory is the time required to complete a **write** operation.

Memory Read Cycle

- Timing Diagram example in the next slide.
- Suppose the CPU operates at 50MHz (=20 ns for one clock cycle)
- Also suppose the access and cycle time of the memory do not exceed 50 ns

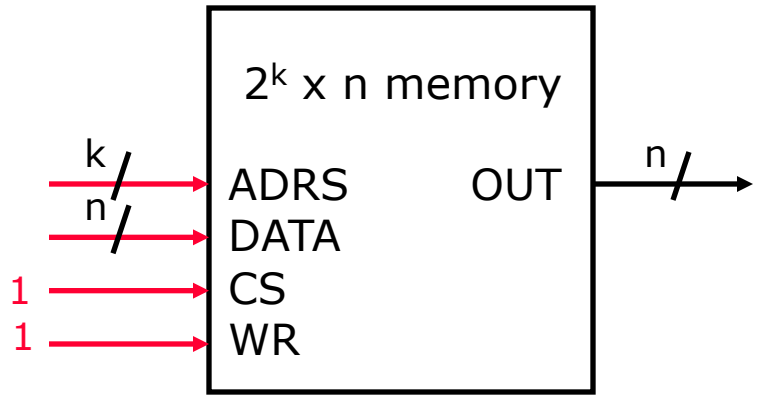
Memory Read Cycle

- Timing Diagram.



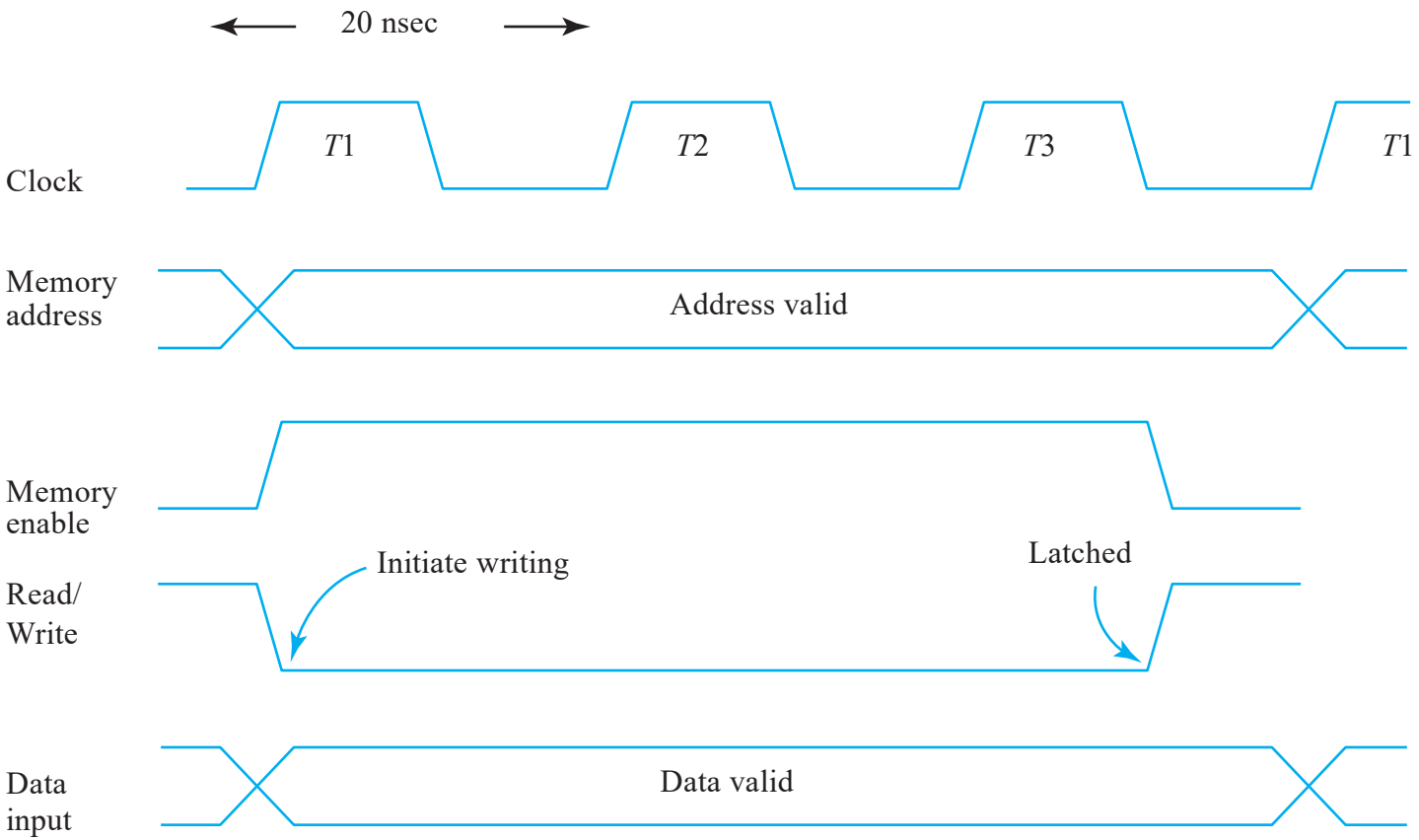
Writing RAM

- To *write* to this RAM, you need to:
  - Enable the chip by setting CS = 1.
  - Select the write operation, by setting WR = 1.
  - Send the desired address to the ADRS input.
  - Send the word to store to the DATA input.
- The output OUT is not needed for memory write operations.



Memory Write Cycle

- Timing Diagram



Types of Memories

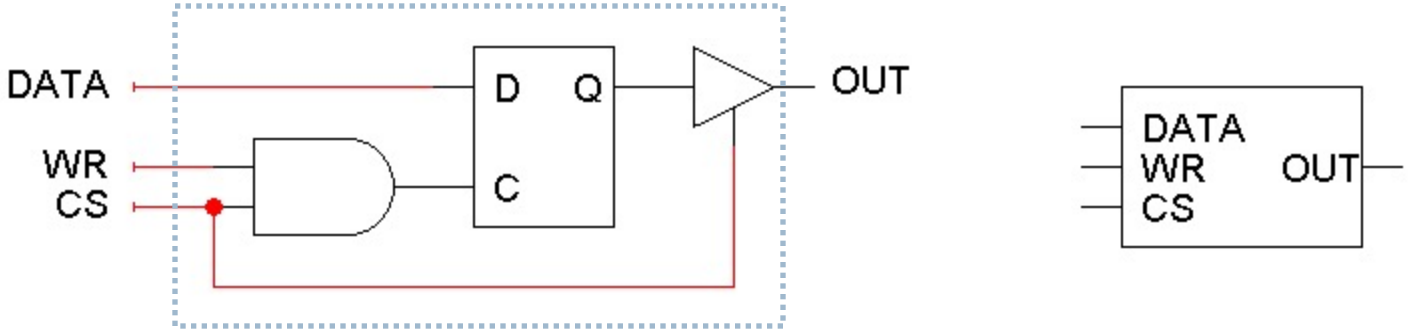
- Static and Dynamic
- Static RAM (SRAM) consists of internal latches
  - The stored data remains valid as long as the power is supplied to the unit.
- Why use latches instead of flip flops?
  - A latch can be made with only two NAND or two NOR gates, but a flip-flop requires at least twice that much hardware.
  - In general, smaller is faster, cheaper and requires less power.
  - The tradeoff is that getting the timing exactly right is a pain.

Dynamic memory

- *Dynamic RAM (DRAM)* is built with capacitors.
  - A stored charge on the capacitor represents a logical 1.
  - No charge represents a logic 0.
- However, capacitors lose their charge after a few milliseconds. The memory requires constant *refreshing* to recharge the capacitors. (That's what's "dynamic" about it.)
- Dynamic RAMs tend to be physically smaller than static RAMs.
  - A single bit of data can be stored with just one capacitor and one transistor, while static RAM cells typically require 4-6 transistors.
  - This means dynamic RAM is cheaper and denser—more bits can be stored in the same physical area.
  - DRAM offers reduced power consumption and larger storage capacity in a single memory chip.
  - SRAM is easier to use and has shorter read and write cycles.

Starting with Latches

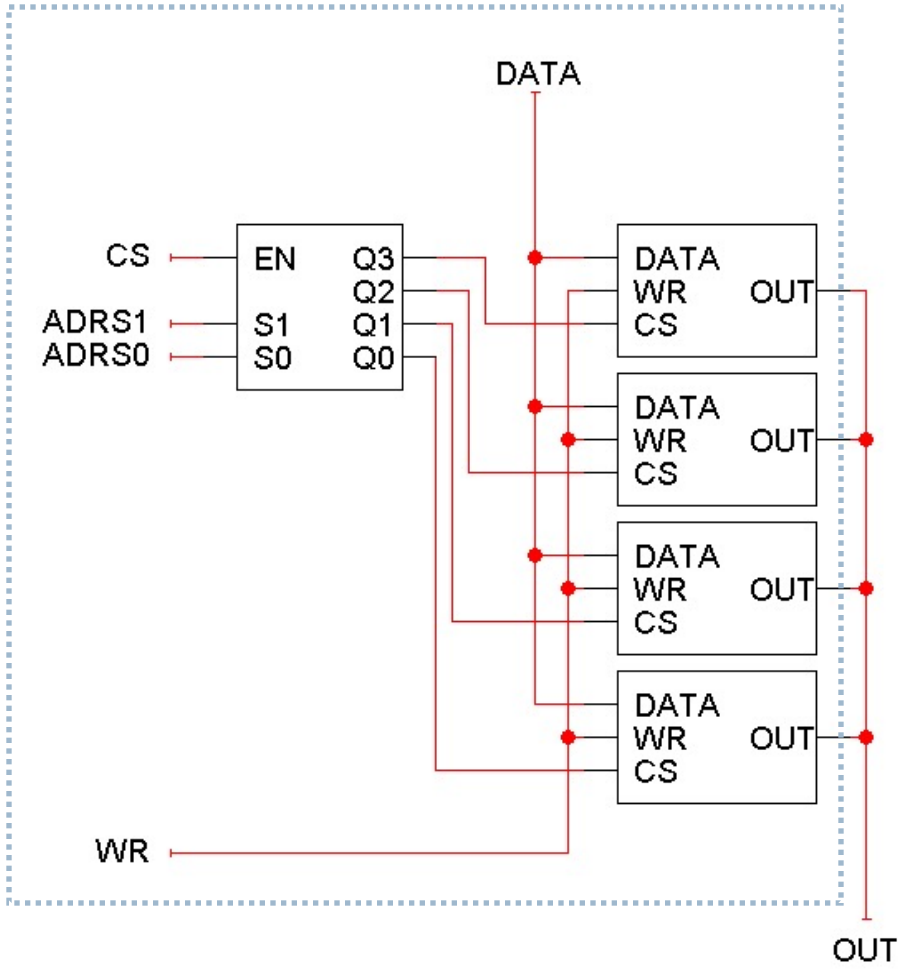
- To start, we can use one latch to store each bit. A one-bit RAM cell is shown here.



- Since this is just a one-bit memory, an ADRS input is not needed.
- Writing to the RAM cell:
  - When CS = 1 and WR = 1, the latch control input will be 1.
  - The DATA input is thus saved in the D latch.
- Reading from the RAM cell and maintaining the current contents:
  - When CS = 0 or when WR = 0, the latch control input is also 0, so the latch just maintains its present state.
  - The current latch contents will appear on OUT.

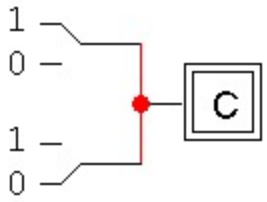
Let's build our first RAM

- We can use these cells to make a 4 x 1 RAM.
- Since there are four words, ADRS is two bits.
- Each word is only one bit, so DATA and OUT are one bit each.
- Word selection is done with a decoder attached to the CS inputs of the RAM cells. Only one cell can be read or written at a time.
- Notice that the outputs are connected together with a *single* line!

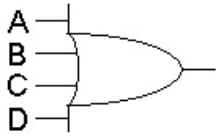


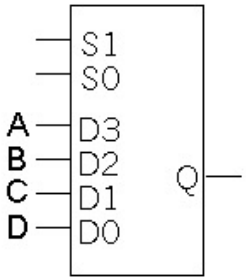
Connecting outputs together

- In normal practice, it's bad to connect outputs together. If the outputs have different values, then a *conflict* arises.



- The standard way to "combine" outputs is to use OR gates (?) or muxes.

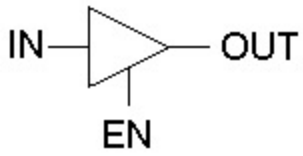




- This can get expensive, with many wires and gates with large fan-ins.

Three-State Buffers

- The triangle represents a *three-state buffer*.
- Unlike regular logic gates, the output can be one of *three* different possibilities, as shown in the table.

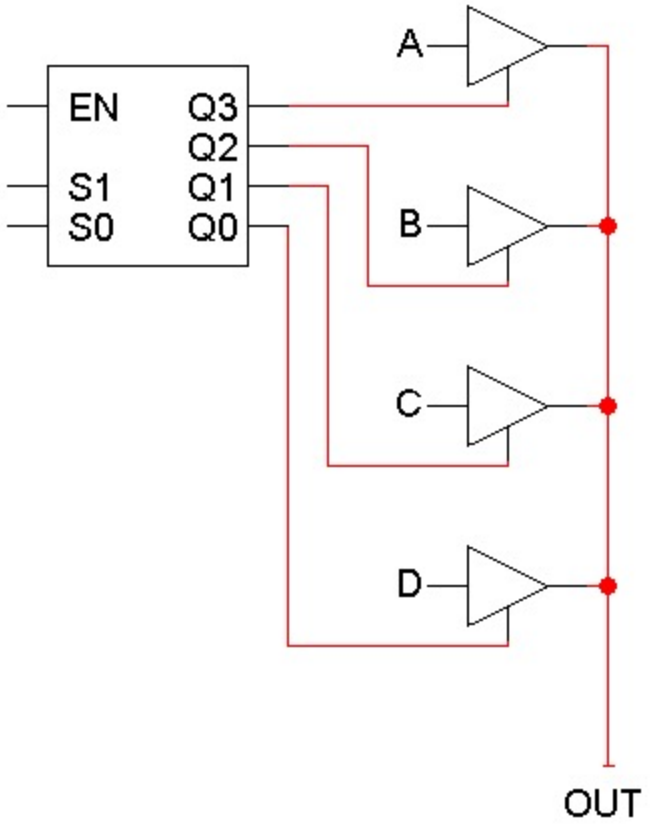


EN	IN	OUT
0	x	Disconnected
1	0	0
1	1	1

- *"Disconnected"* means no output appears at all, in which case it's safe to connect OUT to another output signal.
- The disconnected value is also sometimes called *high impedance* or *Hi-Z*.

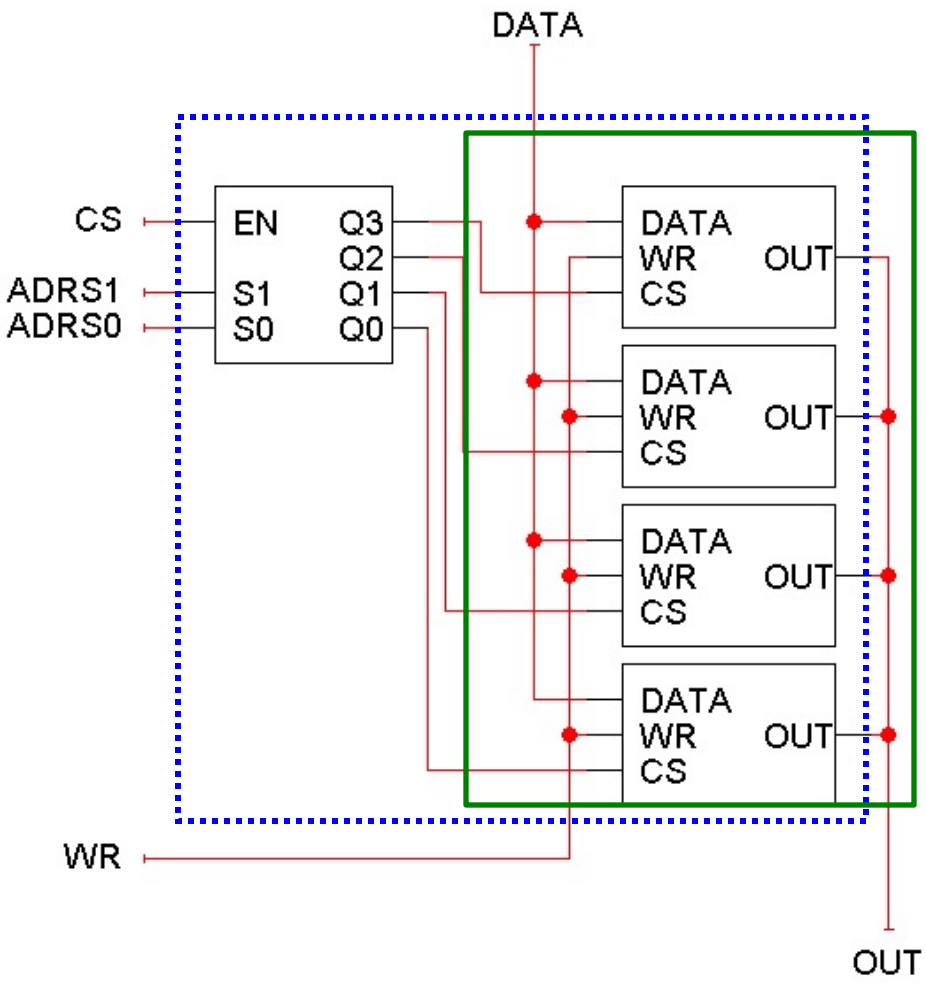
Connecting Three-State Buffers

- You can connect several three-state buffer outputs together if you can *guarantee* that only one of them is enabled at any time.
- The easiest way to do this is to use a decoder!
- If the decoder is disabled, then all the three-state buffers will appear to be disconnected, and OUT will also appear disconnected.
- If the decoder is enabled, then exactly one of its outputs will be true, so only one of the tri-state buffers will be connected and produce an output.
- The net result is we can save some wire and gate costs. We also get a little more flexibility in putting circuits together.



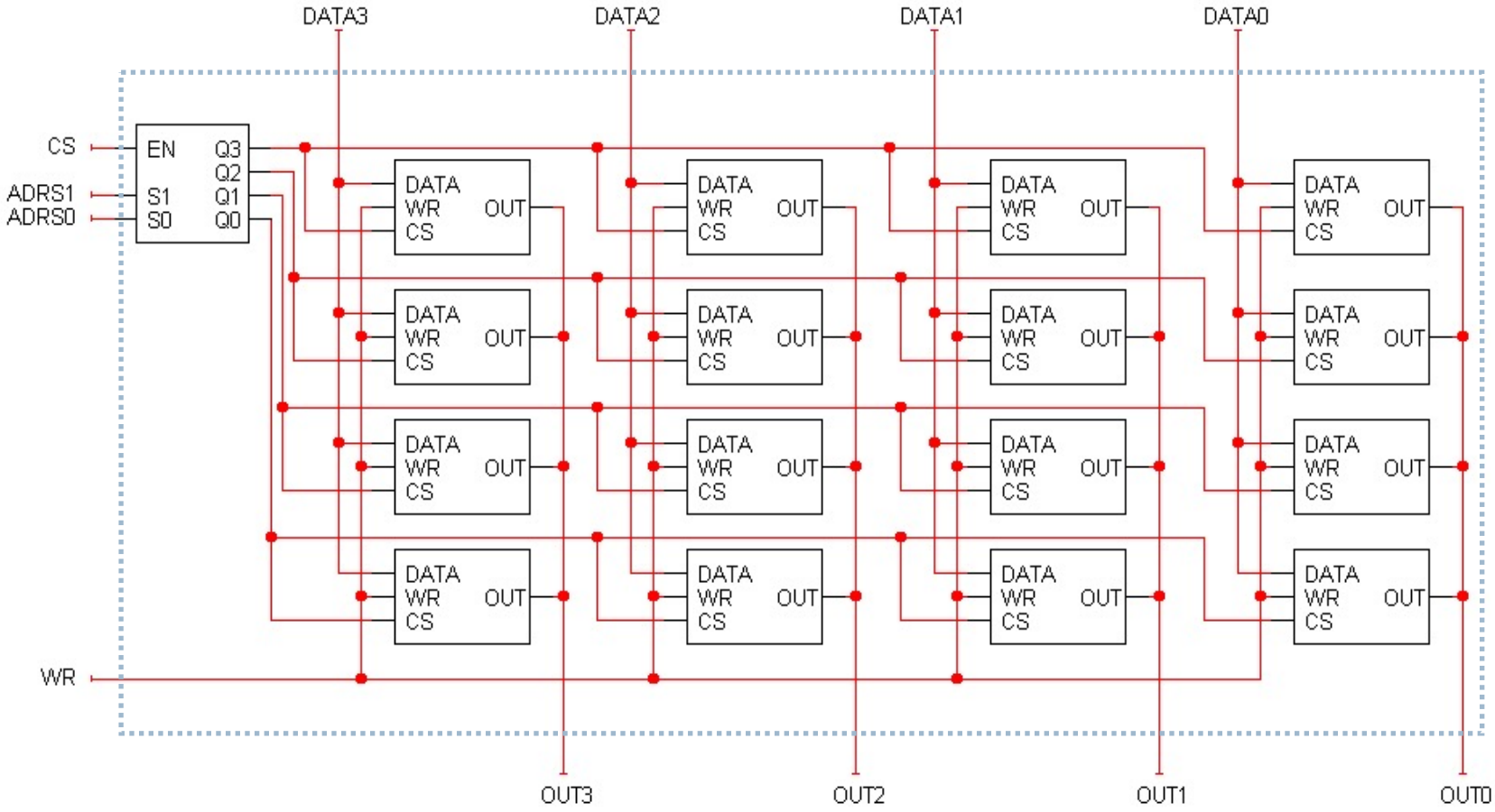
Bigger and Better

- Here is the 4 x 1 RAM once again.
- How can we make a "wider" memory with more bits per word, like maybe a 4 x 4 RAM?
- Duplicate the stuff in the green box!



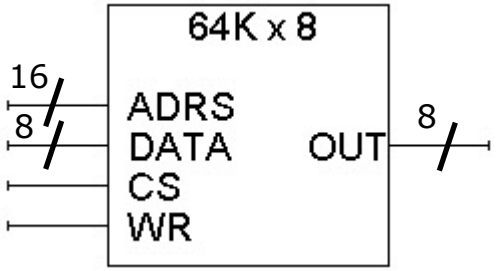
A 4 x 4 RAM

- DATA and OUT are now each *four* bits long, so you can read and write four-bit words.



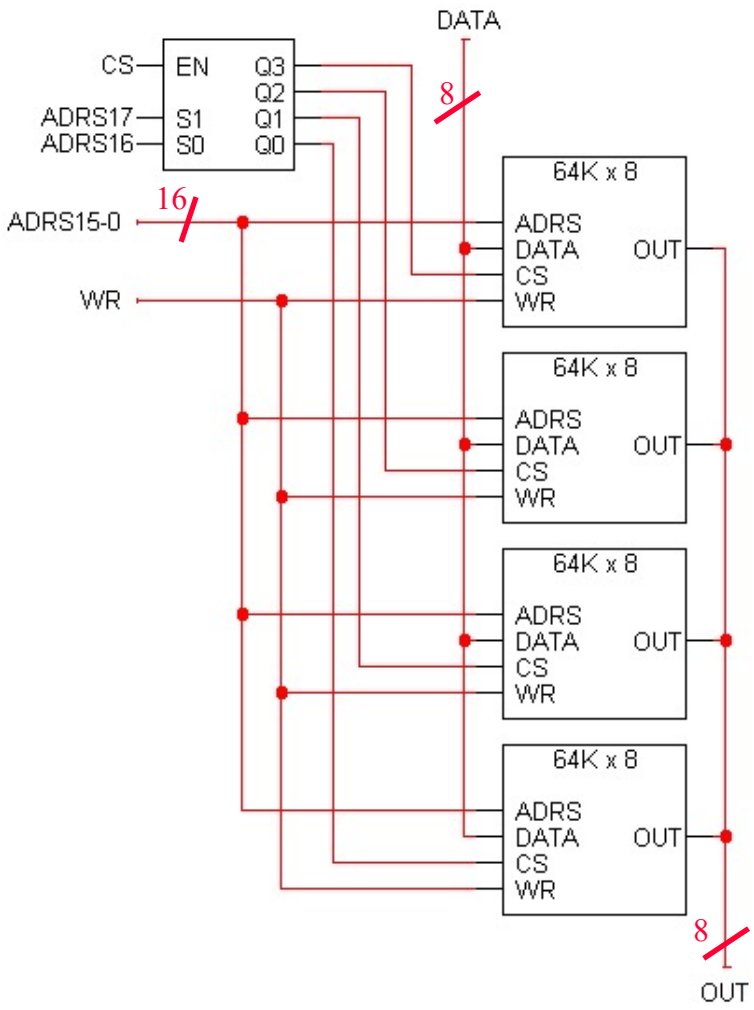
Bigger RAMs from smaller RAMs

- We can use small RAMs as building blocks for making larger memories, by following the same principles as in the previous examples.
- As an example, let's construct a 256Kx8 RAM using 64K x 8 RAMs:
  - How many address lines? How many data lines?
  - 64K = 2^6 x 2^{10} = 2^{16}, so there are 16 address lines.
  - There are 8 data lines.



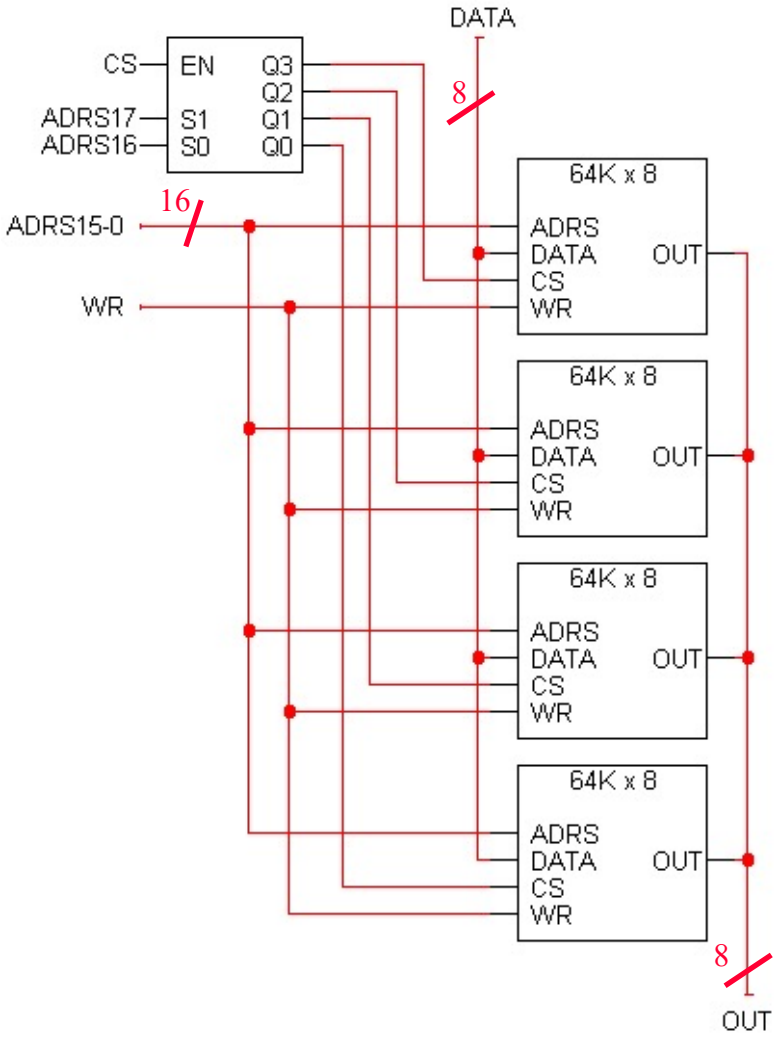
Constructing a larger memory

- We can put four 64K x 8 chips together to make a 256K x 8 memory.
- For 256K words, we need 18 address lines.
  - The two most significant address lines go to the decoder, which selects one of the four 64K x 8 RAM chips.
  - The other 16 address lines are shared by the 64K x 8 chips.
- The 64K x 8 chips also share WR and DATA inputs.
- This assumes the 64K x 8 chips have three-state outputs.

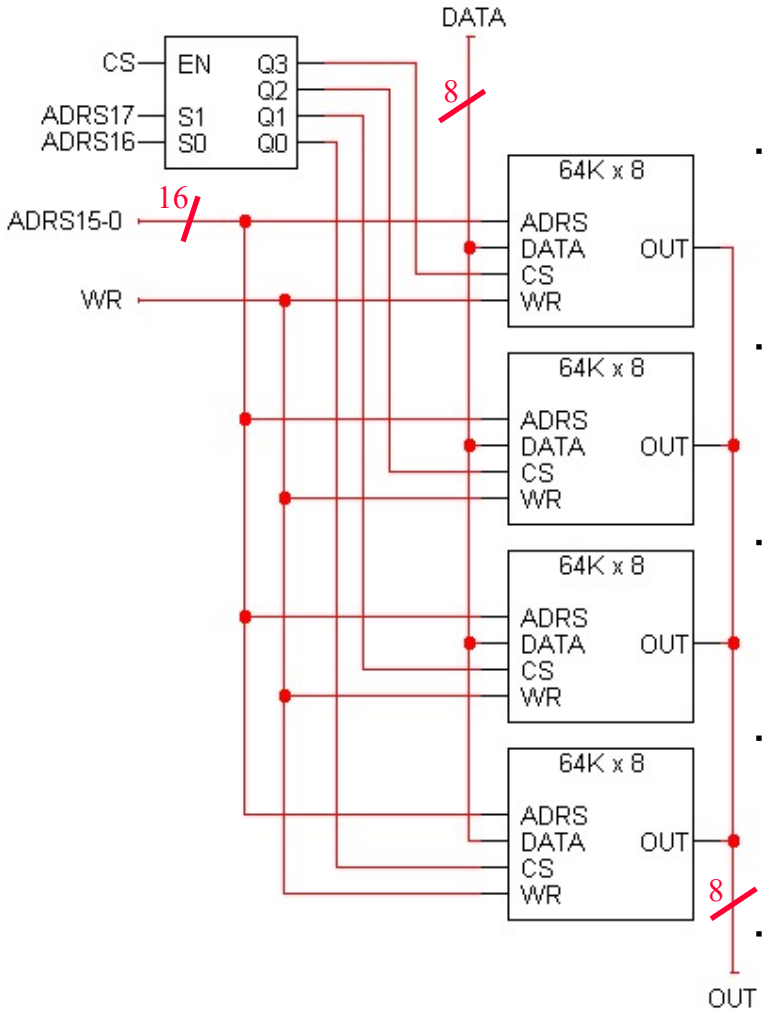


Analyzing the 256K x 8 RAM

- There are 256K words of memory, spread out among the four smaller 64K x 8 RAM chips.
- When the two most significant bits of the address are 00, the bottom RAM chip is selected. It holds data for the first 64K addresses.
- The next chip up is enabled when the address starts with 01. It holds data for the second 64K addresses.
- The third chip up holds data for the next 64K addresses.
- The final chip contains the data of the final 64K addresses.



Address ranges




```
11 1111 1111 1111 1111 (0x3ffff)
to
11 0000 0000 0000 0000 (0x30000)
```

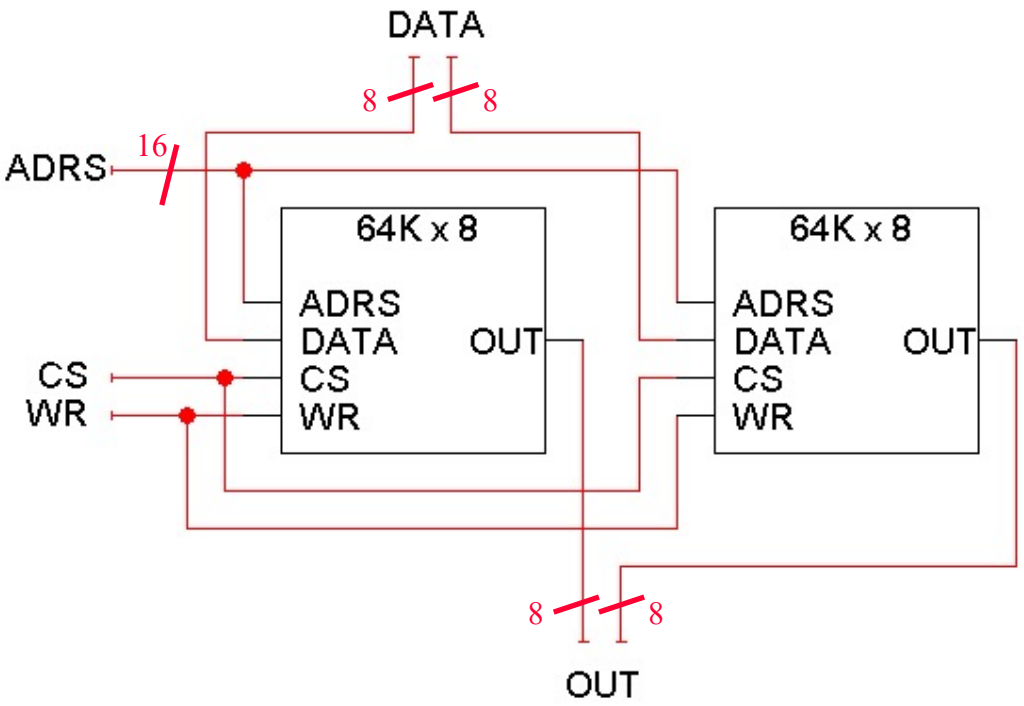
```
10 1111 1111 1111 1111 (0x2ffff)
to
10 0000 0000 0000 0000 (0x20000)
```

```
01 1111 1111 1111 1111 (0x1ffff)
to
01 0000 0000 0000 0000 (0x10000)
```

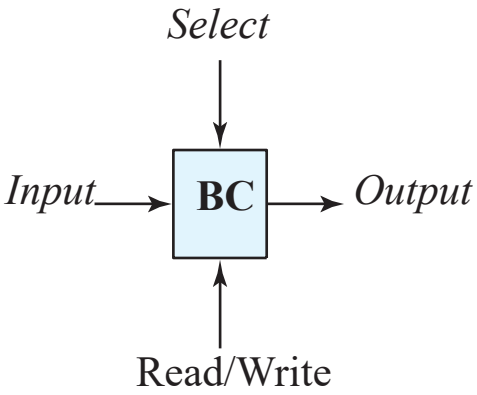
```
00 1111 1111 1111 1111 (0x0ffff)
to
00 0000 0000 0000 0000 (0x00000)
```

Making a wider memory

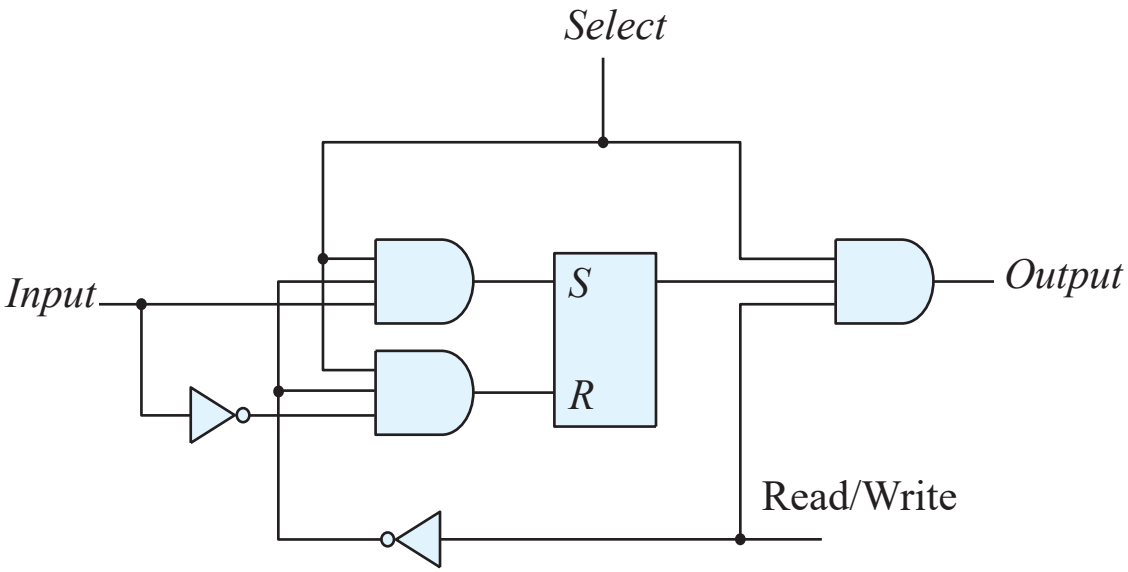
- You can also combine smaller chips to make wider memories, with the same number of addresses but more bits per word.
- Here is a 64K x 16 RAM, created from two 64K x 8 chips.
  - The left chip contains the most significant 8 bits of the data.
  - The right chip contains the lower 8 bits of the data.



Binary Cell



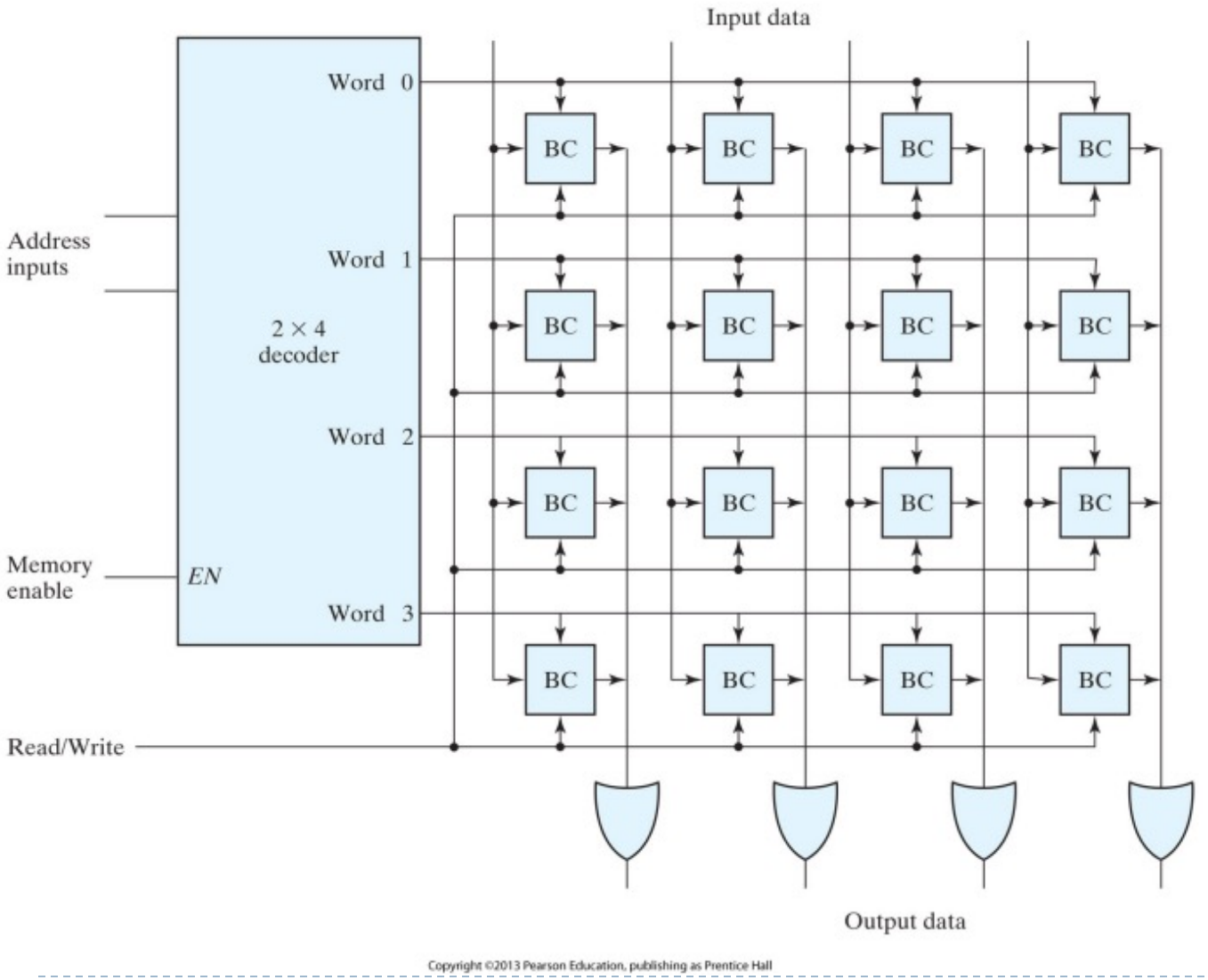
Block Diagram



Logical Equivalent

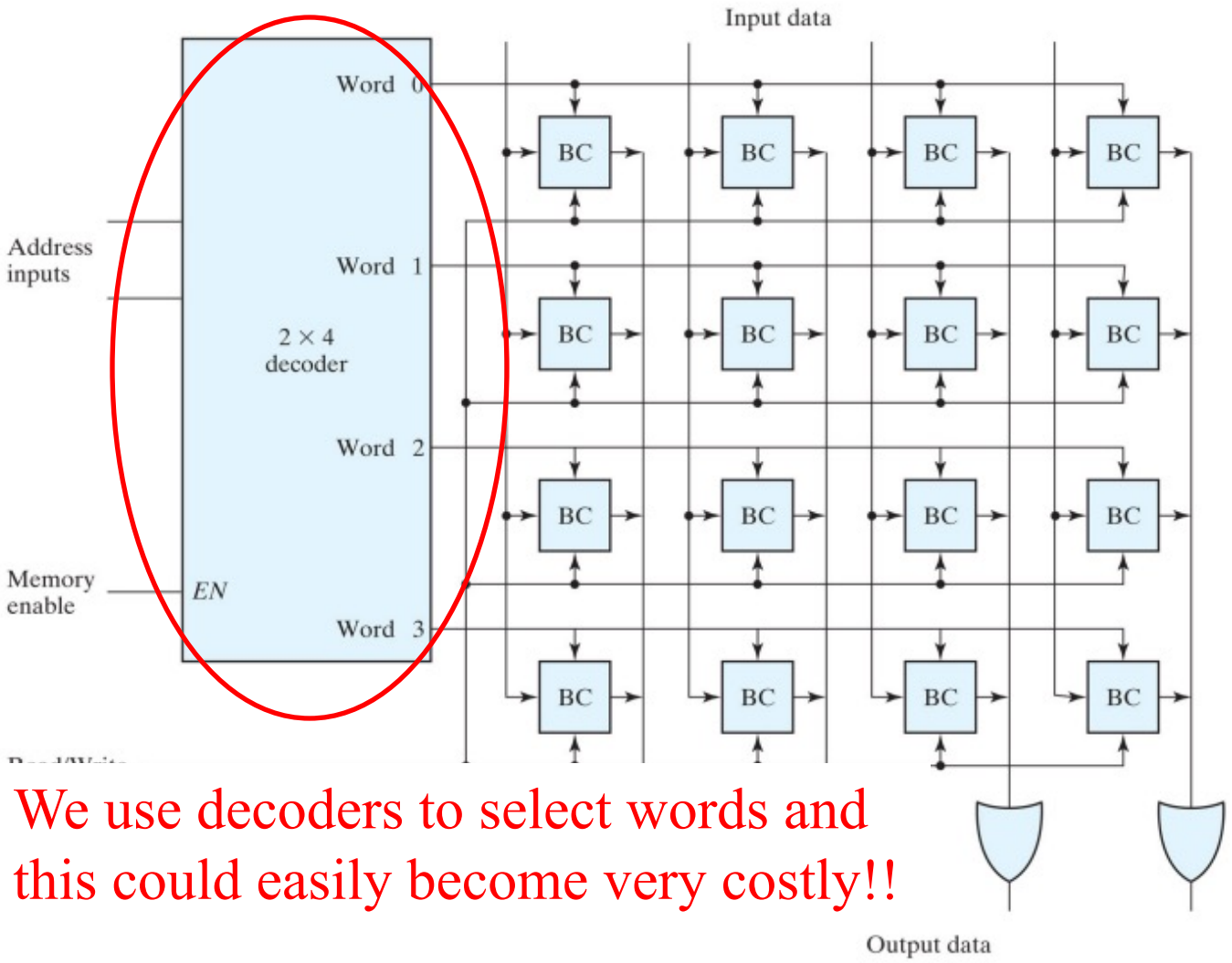
Physical implementation of BC:
SRAM: Uses a latch (has ~6 transistors)
DRAM: A MOS transistor and a capacitor

Cell Access



Each row is a word.

Cell Access

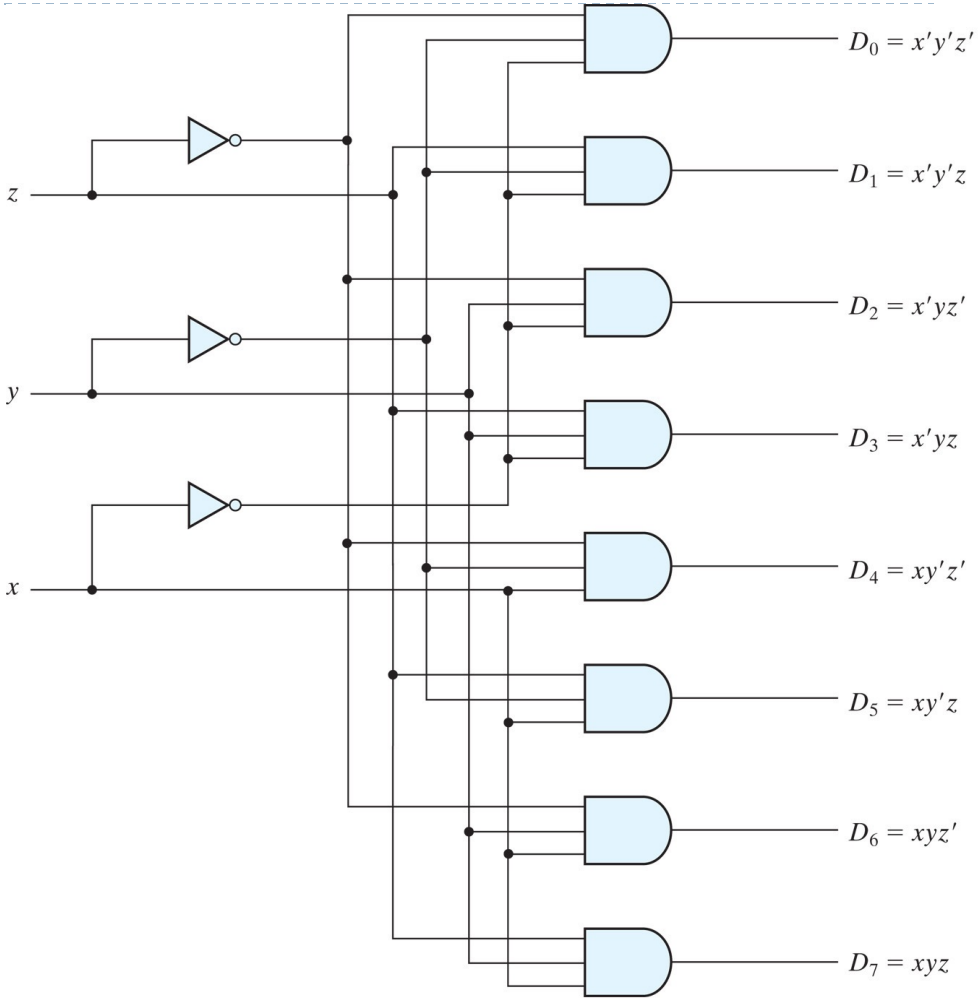


Each row is a word.

We use decoders to select words and this could easily become very costly!!

Decoder reminder

- This is a 3-to-2^3 decoder.

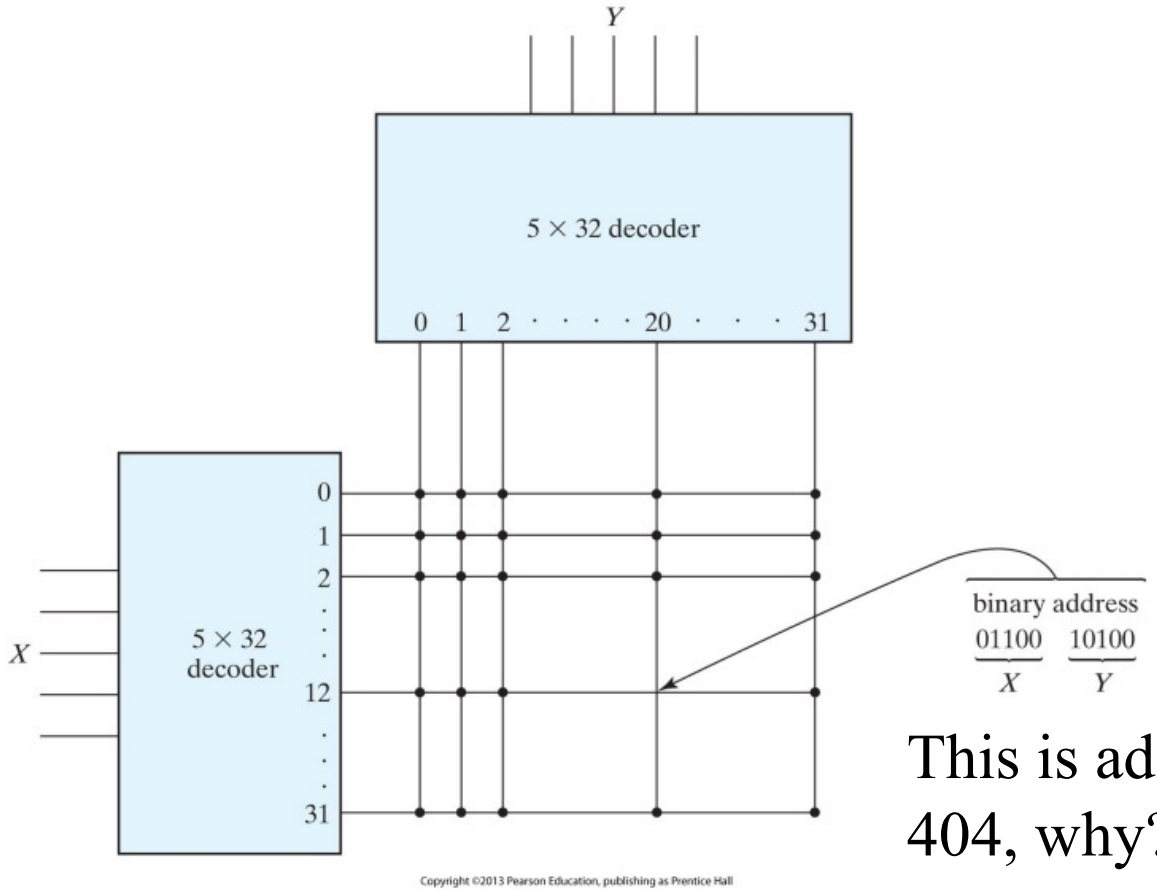


Suppose we want to implement a memory that has 1K (1024) words.

What size decoder do we need?

How many AND gates does it have?

2D Decoding 1K-Word Memory



Goal: reduce the implementation cost of the decoders.

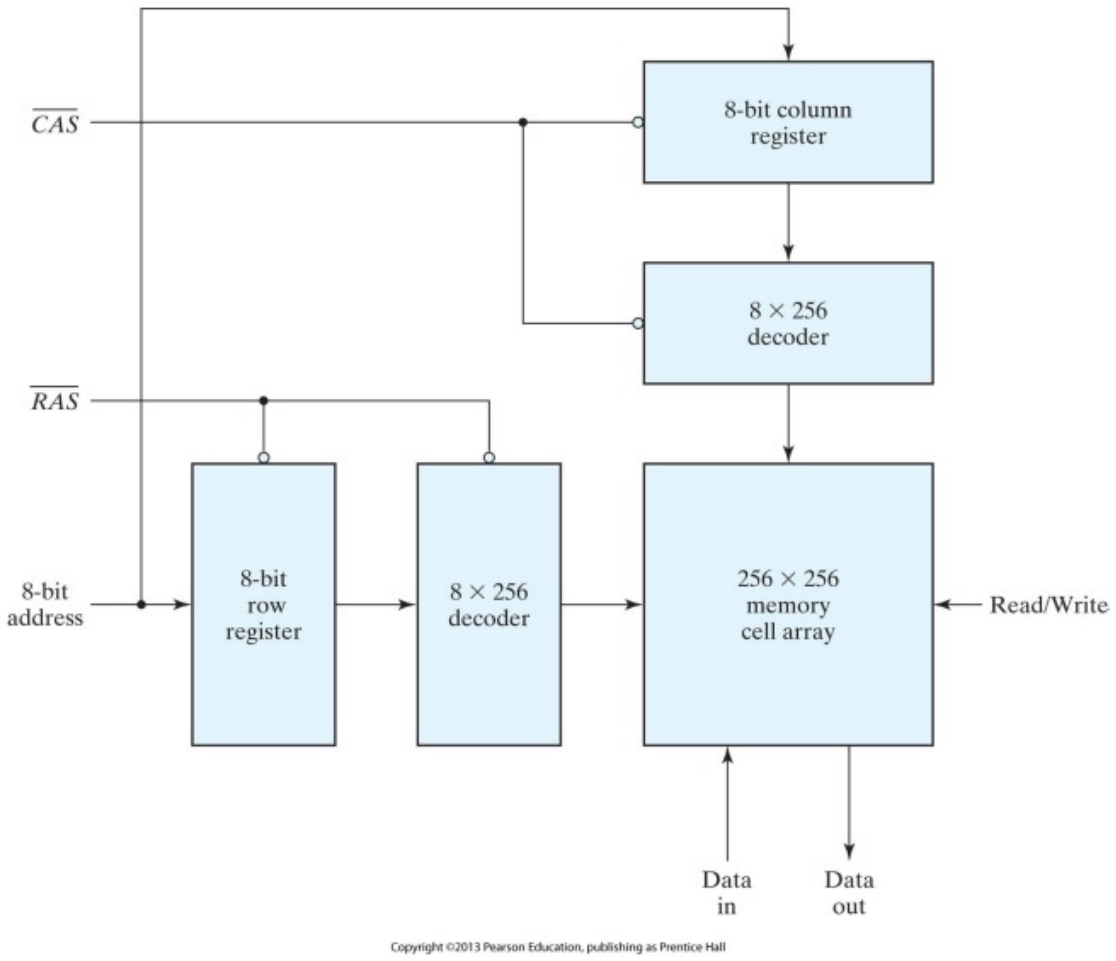
How many AND gates are required in a 10x1024 decoder?

How many AND gates are required in a 5x32 decoder?

This is address 404, why?

Note that each intersection represents a word that may have any number of bits.

Address Multiplexing



Goal: reduce the number of pins in the IC (integrated-circuit) package.

RAS (row address strobe) enables the row register.

CAS (column address strobe) enables the column register.

Error Correction

- Murphy's law: *"Anything that can go wrong, will go wrong."*
- Memory is no exception! Some bit will flip once in a while..
- The dynamic physical interaction of the electrical signals affecting the data path of a memory unit may cause occasional errors in storing and retrieving the binary information.
- We need to
  - Detect whether there is an error
  - Correct it, if possible

Error Correction

- Use extra bits
  - For instance append a parity bit (remember this? How was it implemented?)
  - For more interesting methods read the related section of the book. (Section 7.4)

RAM Summary

- A RAM looks like a bunch of registers connected together, allowing users to select a particular address to read or write.
- Much of the hardware in memory chips supports this selection process:
  - Chip select inputs
  - Decoders
  - Tri-state buffers
- By providing a general interface, it's easy to connect RAMs together to make "longer" and "wider" memories.

Dynamic vs. Static Memory

- Dynamic RAM: uses capacitors and transistors
- Static RAM: uses latches
- In practice, dynamic RAM is used for a computer's main memory, since it's cheap and you can pack a lot of storage into a small space.
  - These days you can buy 2GB of memory for 60 liras.

Dynamic vs. Static Memory

- The disadvantage of dynamic RAM is its speed.
  - Access and cycle times are long compared to CPU's clock period.
- Real systems augment dynamic memory with small but fast sections of static memory called caches.
  - Typical processor caches range in size from 128KB to 320KB.
  - That's small compared to a 4GB main memory, but it's enough to significantly increase a computer's overall speed.
  - You'll study caches later on in CENG331 next semester.

SDRAM

- Synchronous DRAM, or SDRAM, is one of the most common types of PC memory now.
- Memory chips are organized into "modules" that are connected to the CPU via a 64-bit (8-byte) bus.
- Speeds are rated in megahertz: PC66, PC100 and PC133 memory run at 66MHz, 100MHz and 133MHz respectively.
- The memory bandwidth can be computed by multiplying the number of transfers per second by the size of each transfer.
  - PC100 can transfer up to 800MB per second (100MHz x 8 bytes/cycle).
  - PC133 can get over 1 GB per second.

DDR-RAM

- A newer type of memory is Double Data Rate, or DDR-RAM.
- It's very similar to earlier SDRAM, except data can be transferred on both the positive *and* negative clock edges. For 100-133MHz buses, the effective memory speeds appear to be 200-266MHz.
- This memory is confusingly called PC1600 and PC2100 RAM, because
  - 200MHz x 8 bytes/cycle = 1600MB/s
  - 266MHz x 8 bytes/cycle = 2100MB/s.
- DDR-RAM has lower power consumption, using 2.5V instead of 3.3V like SDRAM. This makes it good for notebooks and other mobile devices.

DDR-RAM

- DDR2
  - Power savings are achieved primarily due to an improved manufacturing process through die shrinkage, resulting in a drop in operating voltage (1.8 V compared to DDR's 2.5 V).
- DDR3
  - At 100 MHz, DDR3 SDRAM gives a maximum transfer rate of 6400 MB/s.
  - 1.5V
- DDR4
  - At 200 MHz, DDR4 SDRAM gives a maximum transfer rate of 12800 MB/s.
  - 1.2 V
- DDR5
  - Released in 2020
  - Doubles the rates of DDR4
  - 1.1 V

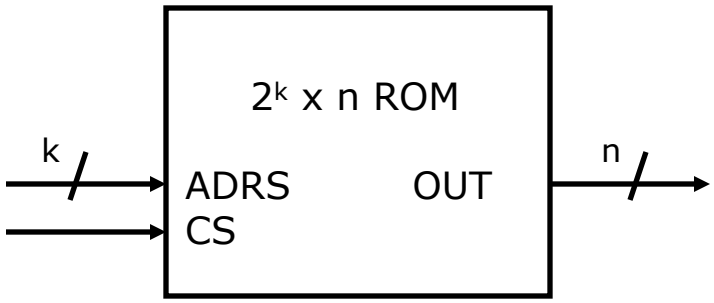
Programmable Logic

Programmable Logic Devices (PLDs)

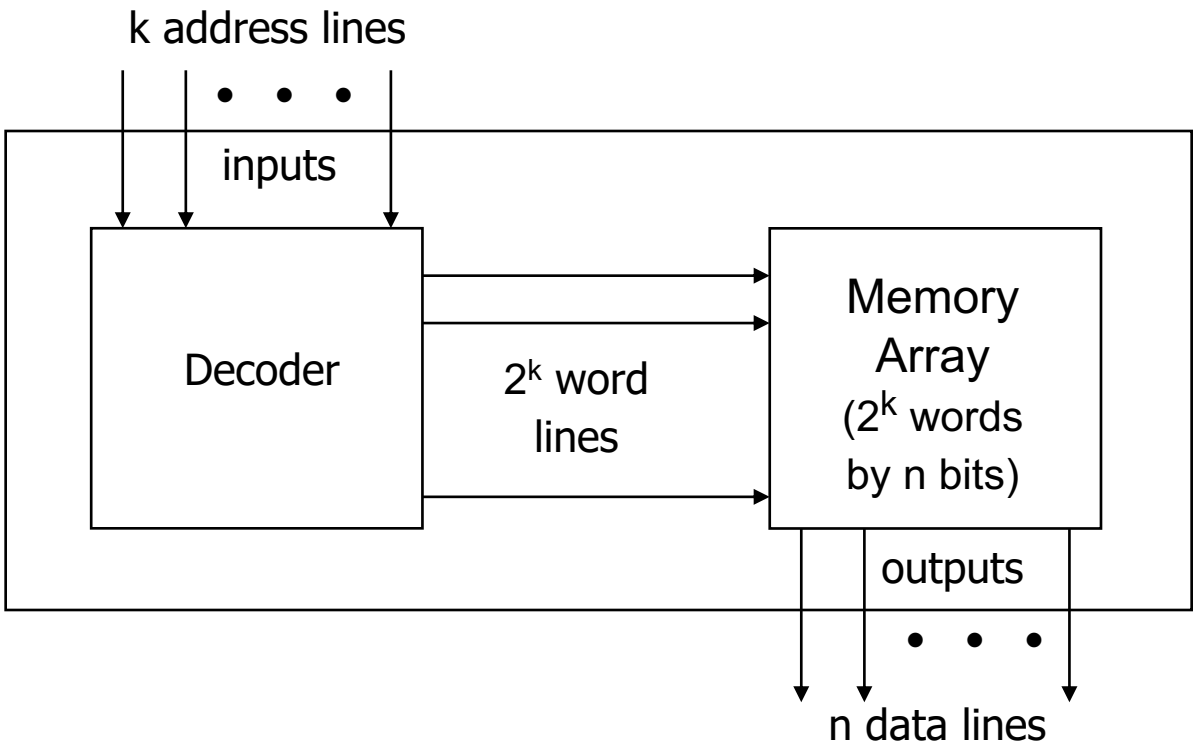
- PLD is a re-configurable IC
- Built with large numbers of gates connected through electronic fuses to implement arbitrary circuits.
- Types:
  - Programmable ROM (read-only memory)
  - Programmable Logic Array (PLA)
  - Programmable Array Logic (PAL)
  - FPGA

Read-Only Memory

- A *read-only memory*, or *ROM*, is a special kind of memory whose contents cannot be easily modified.
  - The WR and DATA inputs that we saw in RAMs are not needed.
  - Data is stored onto a ROM chip using special hardware tools.
- ROMs are useful for holding data that never change.
  - Arithmetic circuits might use tables to speed up computations of logarithms or divisions.
  - Many computers use a ROM to store important programs that should not be modified, such as the system BIOS.
  - PDAs, game machines, cell phones, vending machines and other electronic devices may also contain non-modifiable programs.



ROM Structure



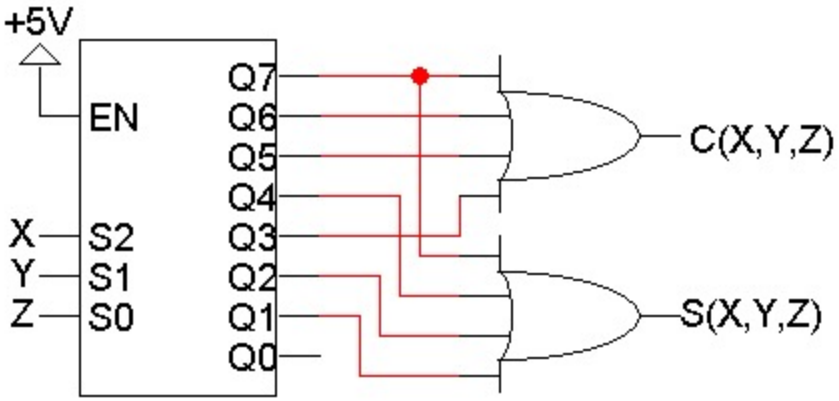
Memories and functions

- ROMs are actually combinational devices, not sequential ones!
  - (Once set) The same address will always contain the same data.
  - You can think of a ROM as a combinational circuit that takes an address as input, and produces some data as the output.
- A ROM table is basically just a truth table.
  - The table shows what data is stored at each ROM address.
  - You can generate that data combinationally, using the address as the input.

Address $A_2A_1A_0$	Data $V_2V_1V_0$
000	000
001	100
010	110
011	100
100	101
101	000
110	011
111	011

Using Decoders

- We can already convert truth tables to circuits easily, with decoders.

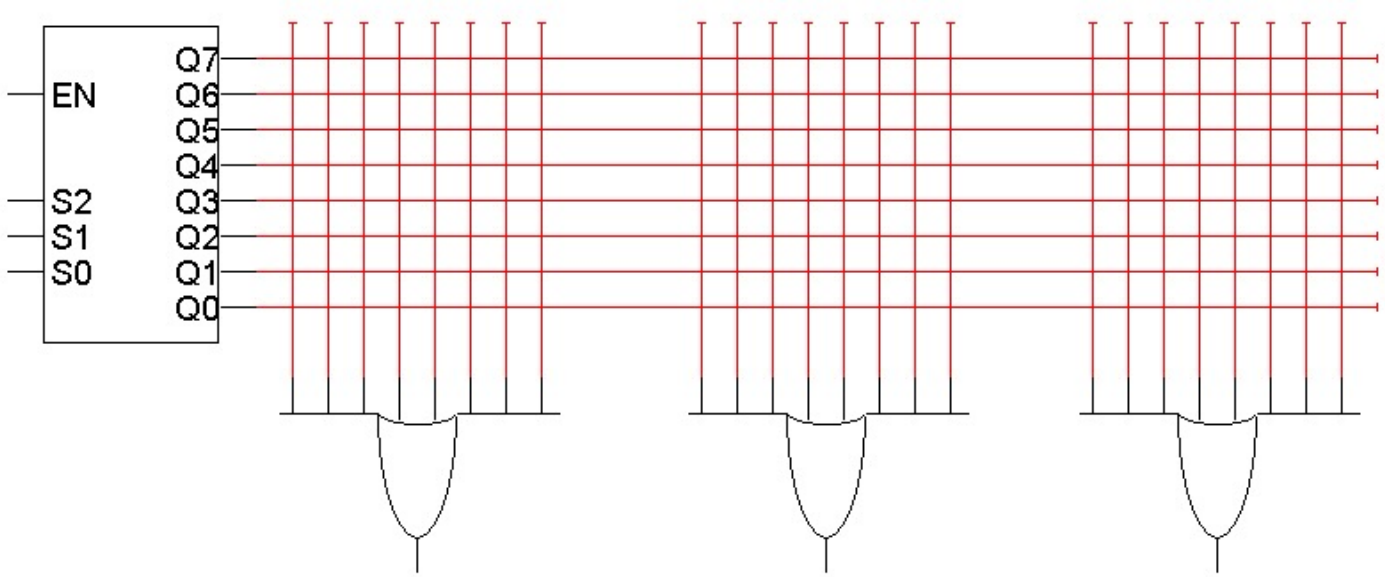


X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- For example, you can think of this old circuit as a memory that "stores" the sum and carry outputs from the truth table on the right.

ROM setup

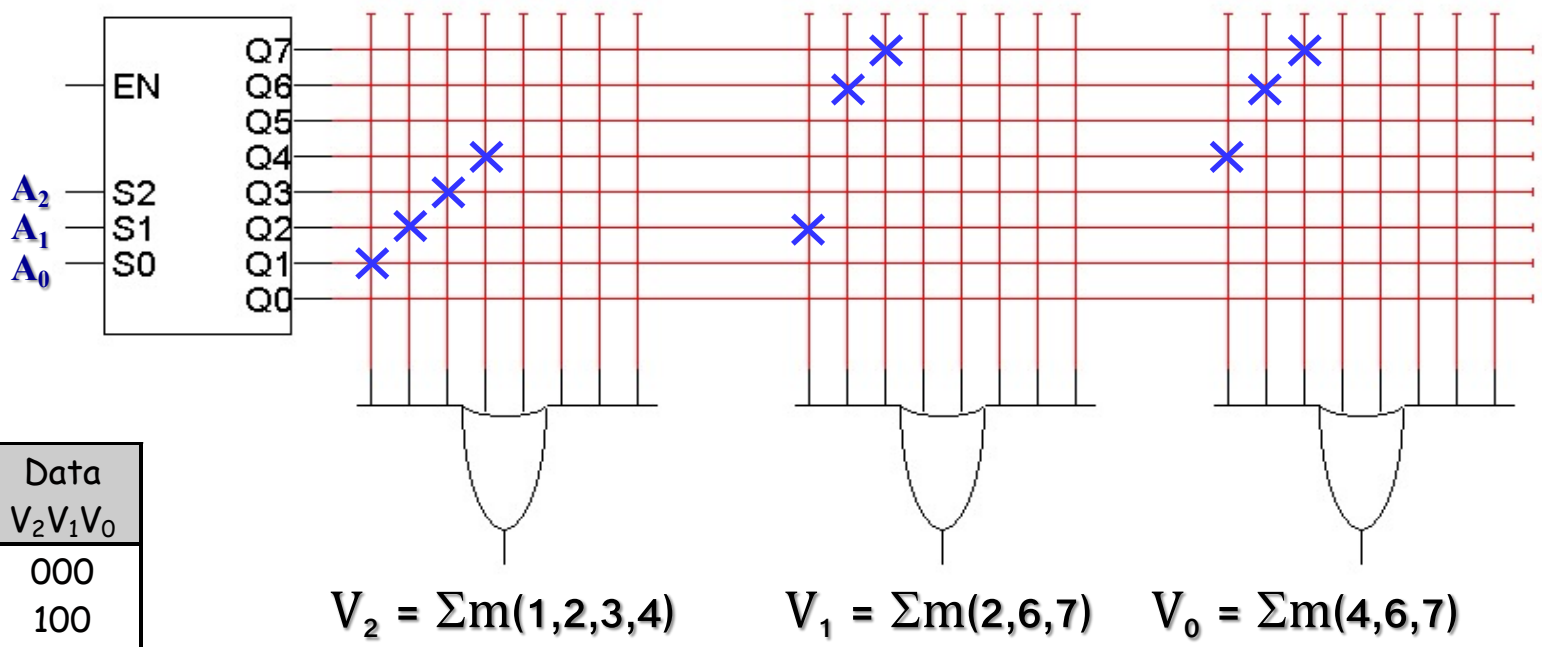
- ROMs are based on this decoder implementation of functions.
  - A blank ROM just provides a decoder and several OR gates.
  - The connections between the decoder and the OR gates are "programmable," so different functions can be implemented.
- To program a ROM, you just make the desired connections between the decoder outputs and the OR gate inputs.



Address $A_2A_1A_0$	Data $V_2V_1V_0$
000	000
001	100
010	110
011	100
100	101
101	000
110	011
111	011

ROM Example

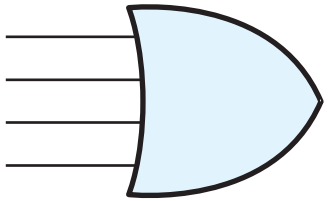
- Here are three functions, $V_2V_1V_0$, implemented with an 8 x 3 ROM.
- Blue crosses (X) indicate connections between decoder outputs and OR gates. Otherwise there is no connection.



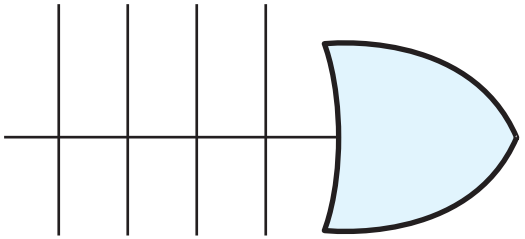
Address $A_2A_1A_0$	Data $V_2V_1V_0$
000	000
001	100
010	110
011	100
100	101
101	000
110	011
111	011

"Fuse" to form a connection, "blow" a fuse to disconnect

A reminder



(a) Conventional symbol



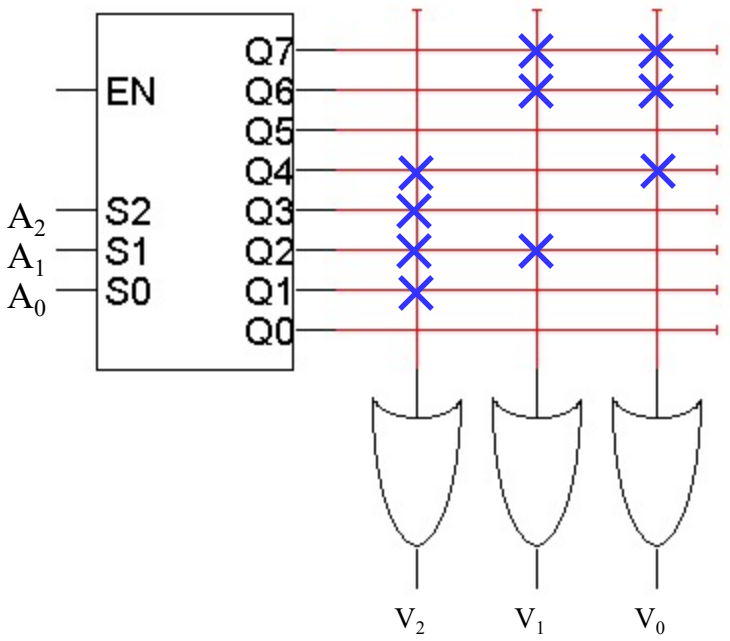
(b) Array logic symbol

In PLD, the symbol in (b) is used.

The same example again …

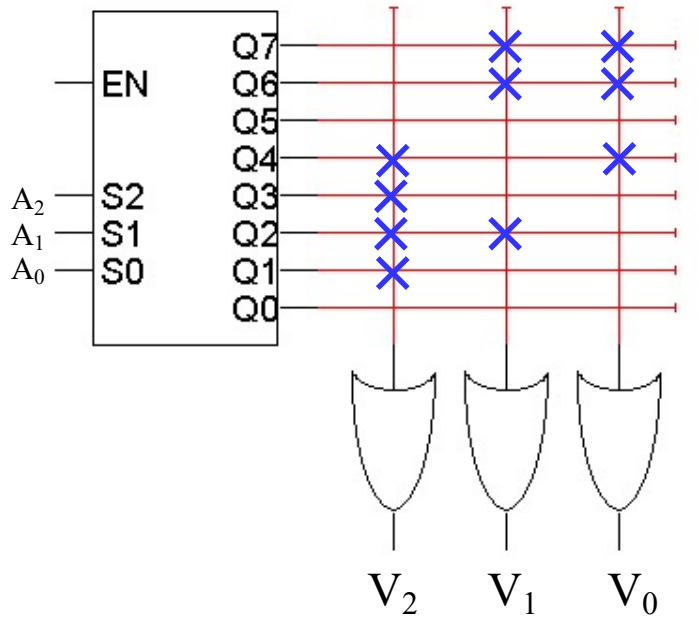
- Here is an alternative presentation of the same 8 x 3 ROM, using *"abbreviated"* OR gates to make the diagram neater.

$V_2 = \Sigma m(1,2,3,4)$
$V_1 = \Sigma m(2,6,7)$
$V_0 = \Sigma m(4,6,7)$



Why is this a "Memory"?

- This combinational circuit can be considered a read-only memory.
  - It stores eight words of data, each consisting of three bits.
  - The decoder inputs form an address, which refers to one of the eight available words.
  - So every input combination corresponds to an address, which is "read" to produce a 3-bit data output.



Address $A_2A_1A_0$	Data $V_2V_1V_0$
000	000
001	100
010	110
011	100
100	101
101	000
110	011
111	011

ROM Programming

Table 7.3
ROM Truth Table (Partial)

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮							⋮			
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

ROM Programming

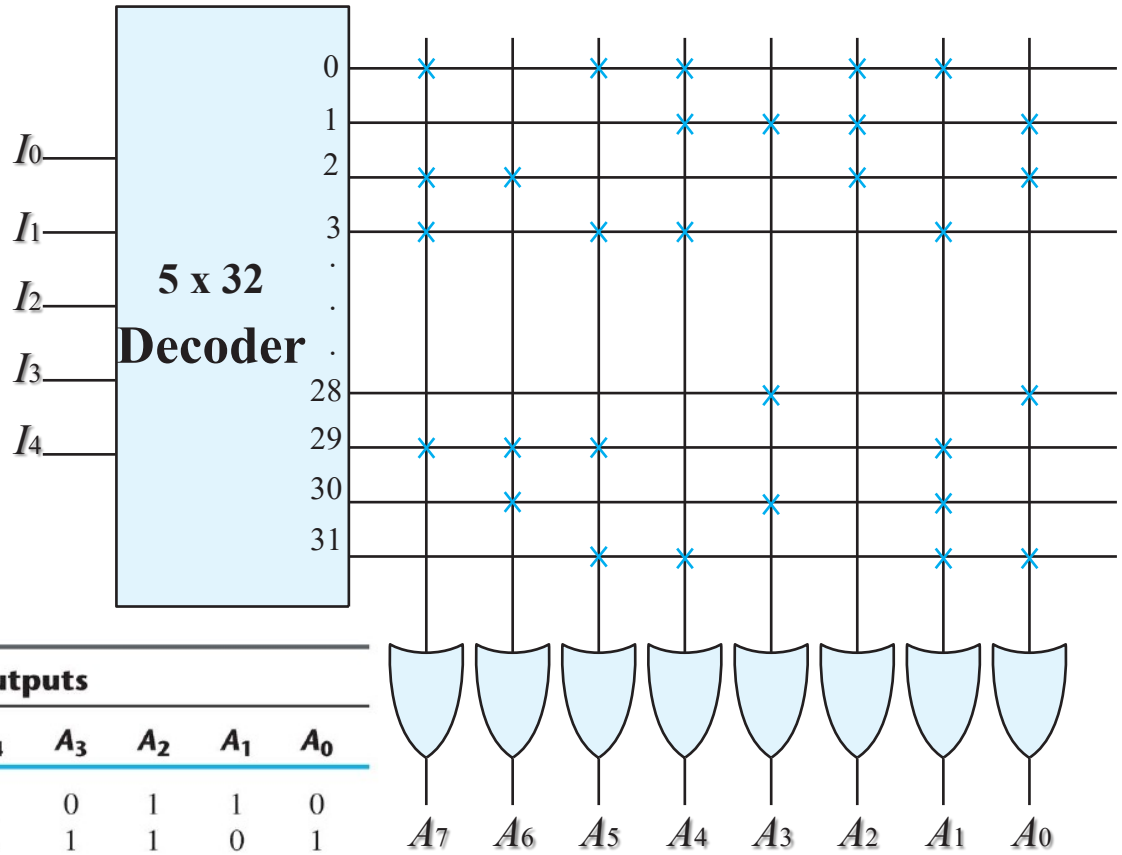


Table 7.3
ROM Truth Table (Partial)

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮						⋮				
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

ROMs vs. RAMs

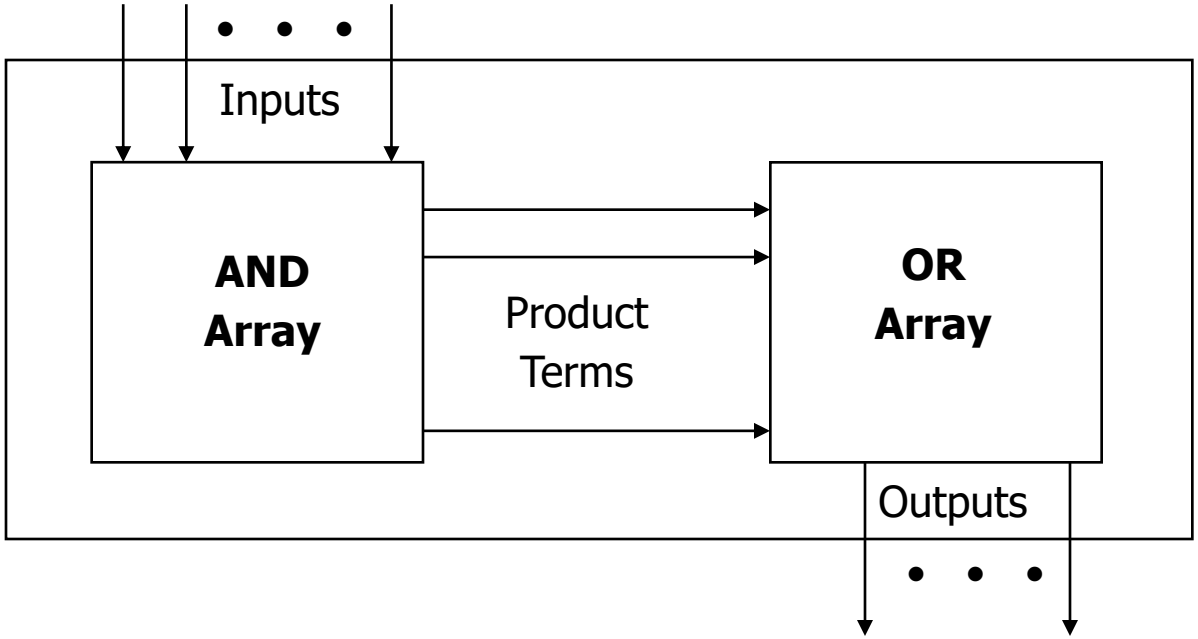
- There are some important differences between ROM and RAM.
  - ROMs are "non-volatile"—data is preserved even without power. On the other hand, RAM contents disappear once power is lost.
  - ROMs require special (and slower) techniques for writing, so they're considered to be "read-only" devices.
- Some newer types of ROMs do allow for easier writing, although the speeds still don't compare with regular RAMs.
  - MP3 players, digital cameras and other toys use CompactFlash, Secure Digital, or MemoryStick cards for non-volatile storage.
  - Many devices allow you to upgrade programs stored in "flash ROM."

Programmable Logic Arrays

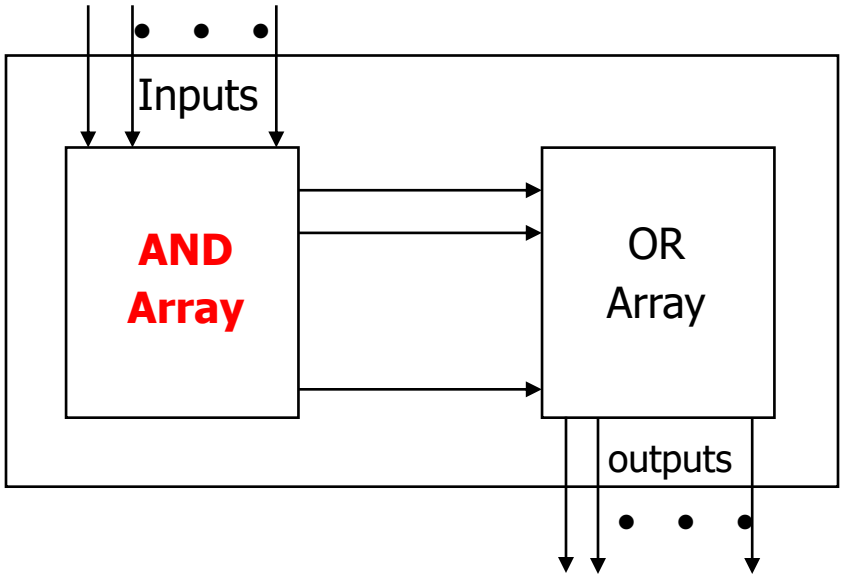
- A ROM is potentially inefficient because it uses a decoder, which generates *all* possible minterms. No circuit minimization is done.
- Using a ROM to implement an n-input function requires:
  - An n-to-2^n decoder (which uses n inverters and 2^n n-input AND gates; and an OR gate with up to 2^n inputs).
  - The number of gates roughly doubles for each additional ROM input.
- A *programmable logic array*, or PLA, makes the decoder part of the ROM "programmable" too.
  - Instead of generating all minterms, you can choose which products (not necessarily minterms) to generate.

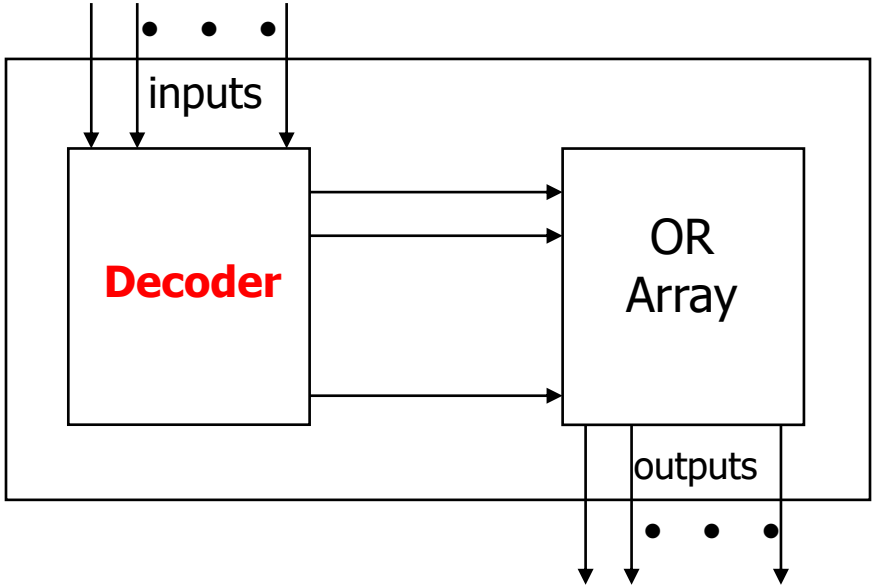
Programmable Logic Array (PLA)

- Pre-fabricated building block of many AND/OR gates
  - Actually NOR or NAND
  - "Personalized" by making or breaking connections among gates
  - Programmable array block diagram for sum of products form



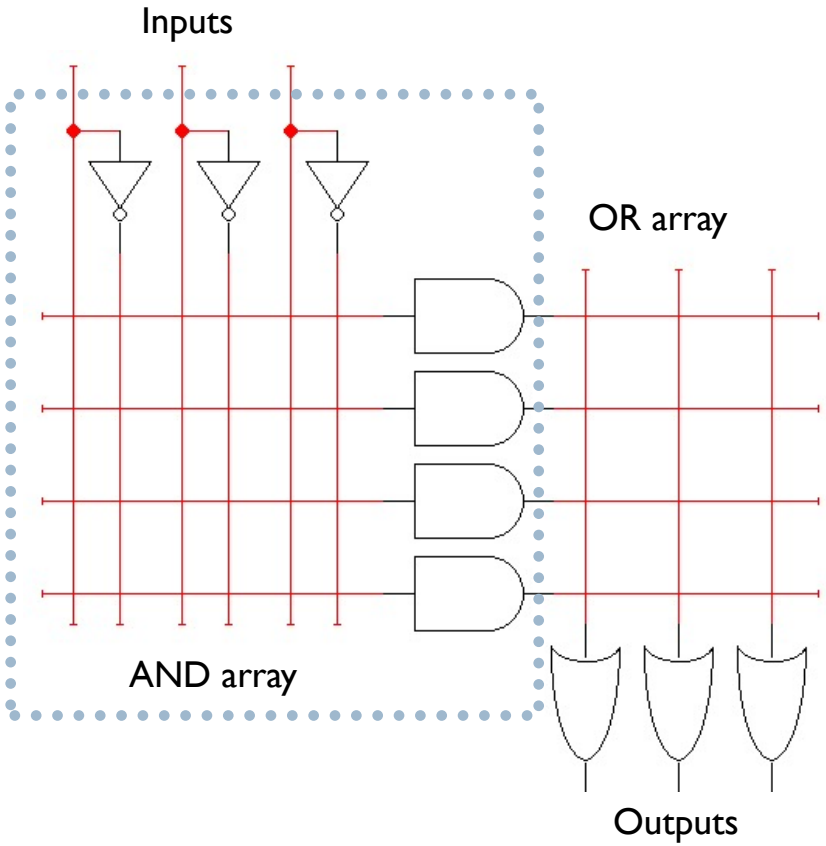
PLA vs ROM


Inputs
AND
Array
OR
Array
outputs


inputs
Decoder
OR
Array
outputs

A blank 3 x 4 x 3 PLA

- This is a 3 x 4 x 3 PLA (3 inputs, up to 4 product terms, and 3 outputs), ready to be programmed.
- The left part of the diagram replaces the decoder used in a ROM.
- Connections can be made in the "AND array" to produce four arbitrary products, instead of 8 minterms as with a ROM.
- Those products can then be summed together in the "OR array."



Regular K-map minimization

- The normal K-map approach is to minimize the number of product terms for each *individual* function.
- For our three functions, this would result in a total of *six* different product terms.

Address $A_2A_1A_0$	Data $V_2V_1V_0$
000	000
001	100
010	110
011	100
100	101
101	000
110	011
111	011

V_2

X \ YZ	00	01	11	10
0	0	1	1	1
1	1	0	0	0

V_1

X \ YZ	00	01	11	10
0	0	0	0	1
1	0	0	1	1

V_0

X \ YZ	00	01	11	10
0	0	0	0	0
1	1	0	1	1

$$V_2 = \Sigma m(1,2,3,4)$$
$$V_1 = \Sigma m(2,6,7)$$
$$V_0 = \Sigma m(4,6,7)$$

PLA minimization

- For a PLA, we should minimize the number of product terms **for *all functions* together**.
- We could express V_2, V_1 and V_0 with just a total of ***four*** products:

$V_2 = xy'z' + x'z + x'yz'$

X \ yz	00	01	11	10
0	0	1	1	1
1	1	0	0	0

$V_1 = x'yz' + xy$

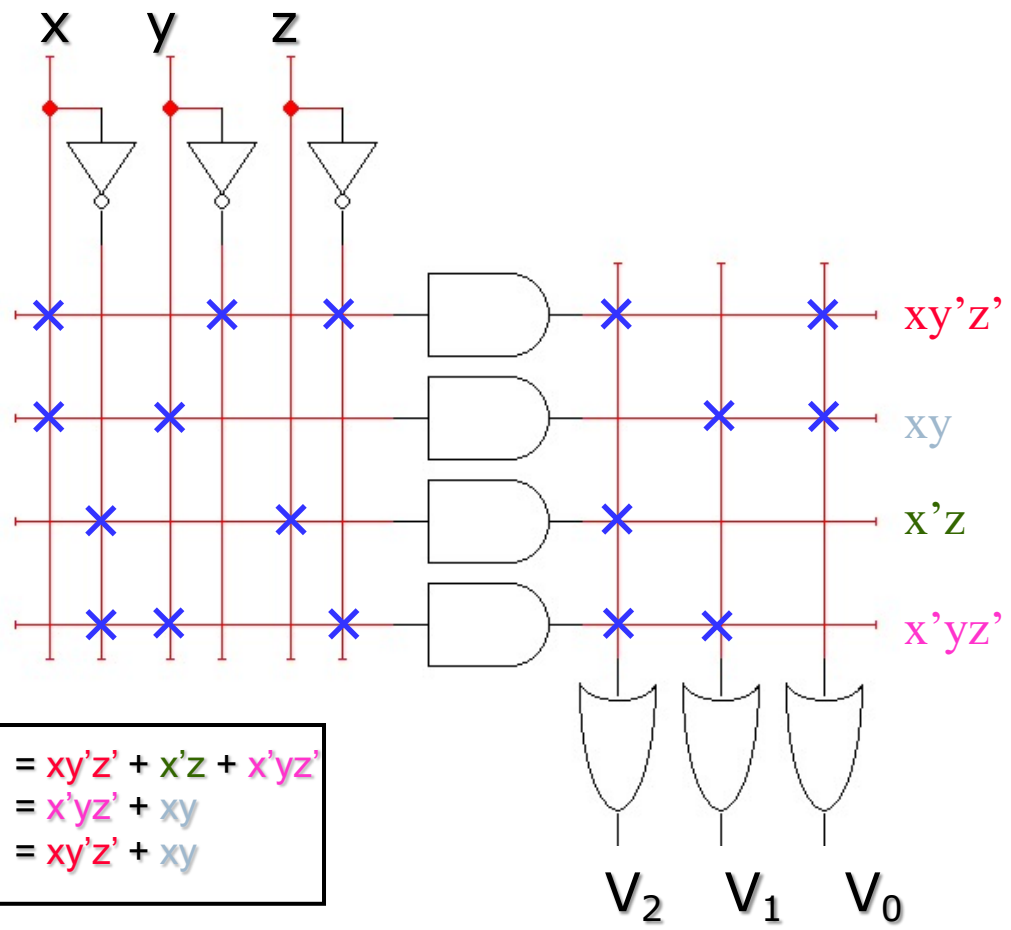
X \ yz	00	01	11	10
0	0	0	0	1
1	0	0	1	1

$V_0 = xy'z' + xy$

X \ yz	00	01	11	10
0	0	0	0	0
1	1	0	1	1

PLA example

- So we can implement these three functions using a 3 x 4 x 3 PLA:



$V_2 = \Sigma m(1,2,3,4) = xy'z' + x'z + x'yz'$
$V_1 = \Sigma m(2,6,7) = x'yz' + xy$
$V_0 = \Sigma m(4,6,7) = xy'z' + xy$

PLA programming example

- Enabling concept: **Shared product terms** among outputs.
- Implement the following function(s) using PLA.

$F_0 = A + B' C'$
$F_1 = A C' + A B$
$F_2 = B' C' + A B$
$F_3 = B' C + A$

PLA programming example

$F_0 = A + B' C'$
$F_1 = A C' + A B$
$F_2 = B' C' + A B$
$F_3 = B' C + A$

Input side:

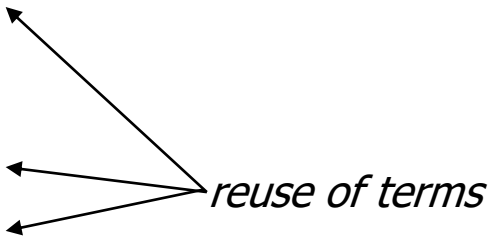
1 : uncomplemented in term
0 : complemented in term
– : does not participate

Output side:

1 : term connected to output
0 : no connection to output

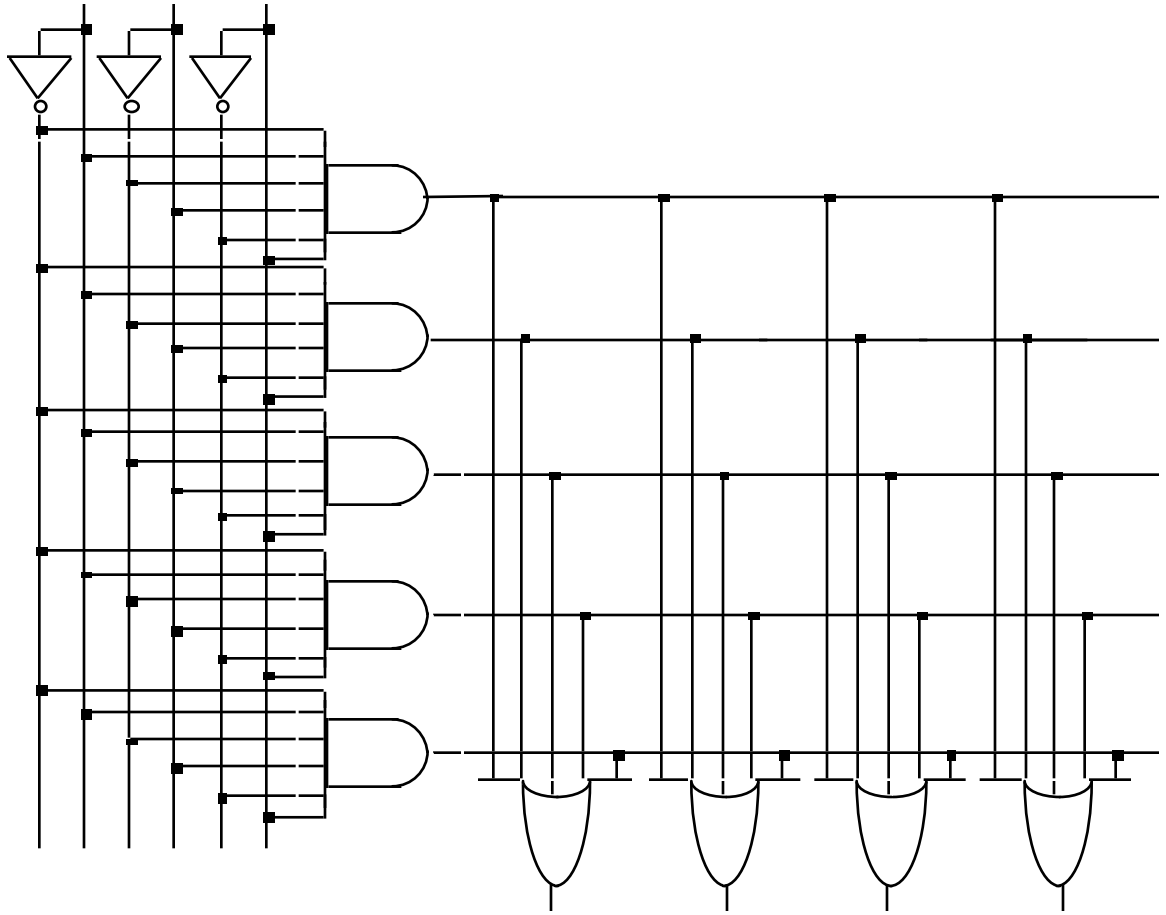
PLA programming table:

Product Term	Inputs A	B	C	Outputs F_0	F_1	F_2	F_3
AB	1	1	–	0	1	1	0
B'C	–	0	1	0	0	0	1
AC'	1	–	0	0	1	0	0
B'C'	–	0	0	1	0	1	0
A	1	–	–	1	0	0	1



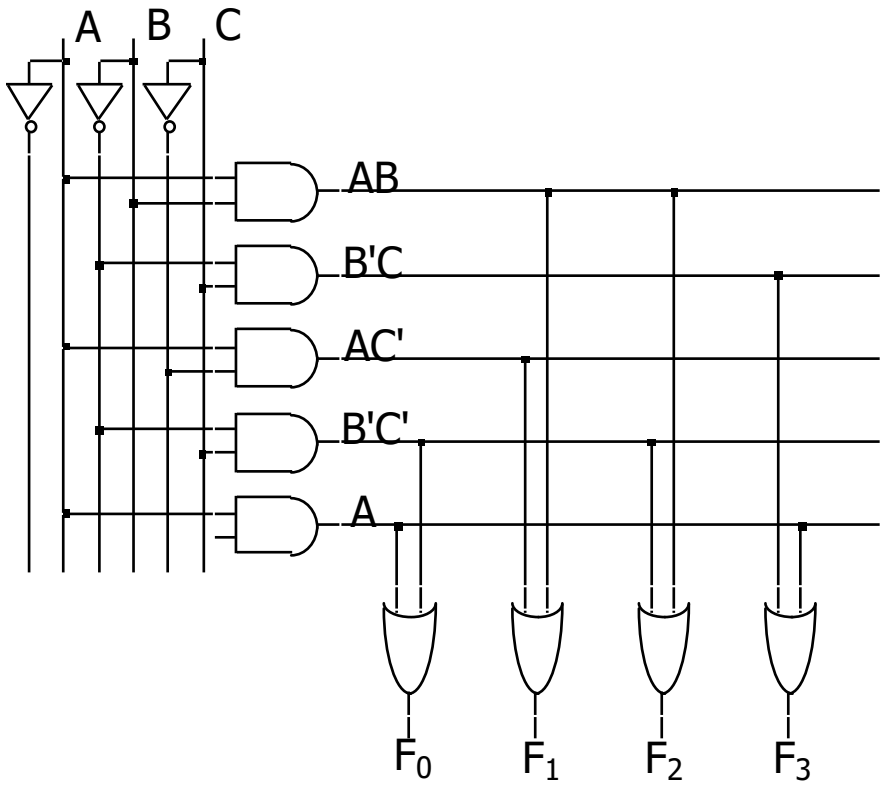
Before Programming

- All possible connections available before "programming"



After Programming

- Unwanted connections are "blown“
- Final, programmed PLA:



$F_0 = A + B' C'$

$F_1 = A C' + A B$

$F_2 = B' C' + A B$

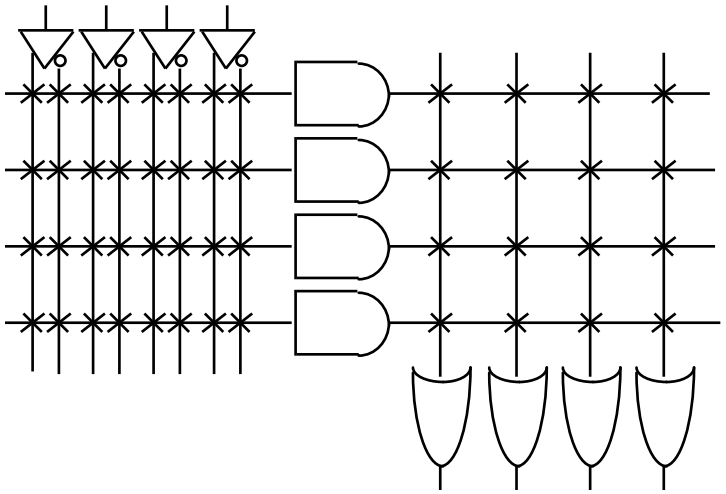
$F_3 = B' C + A$

Another example

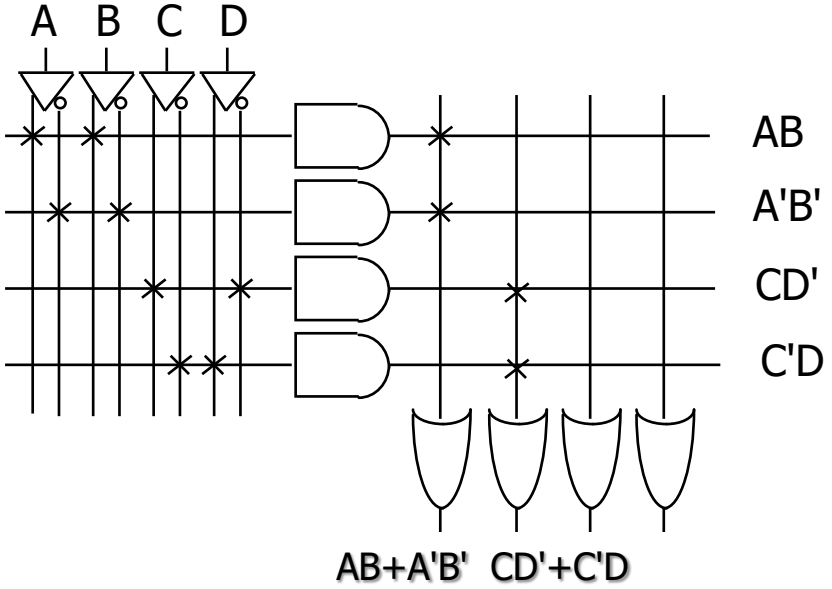
Implement the following using a 4x4x4 PLA:

$$F_0 = A\,B + A'\,B'$$
$$F_1 = C\,D' + C'\,D$$

Before programming:



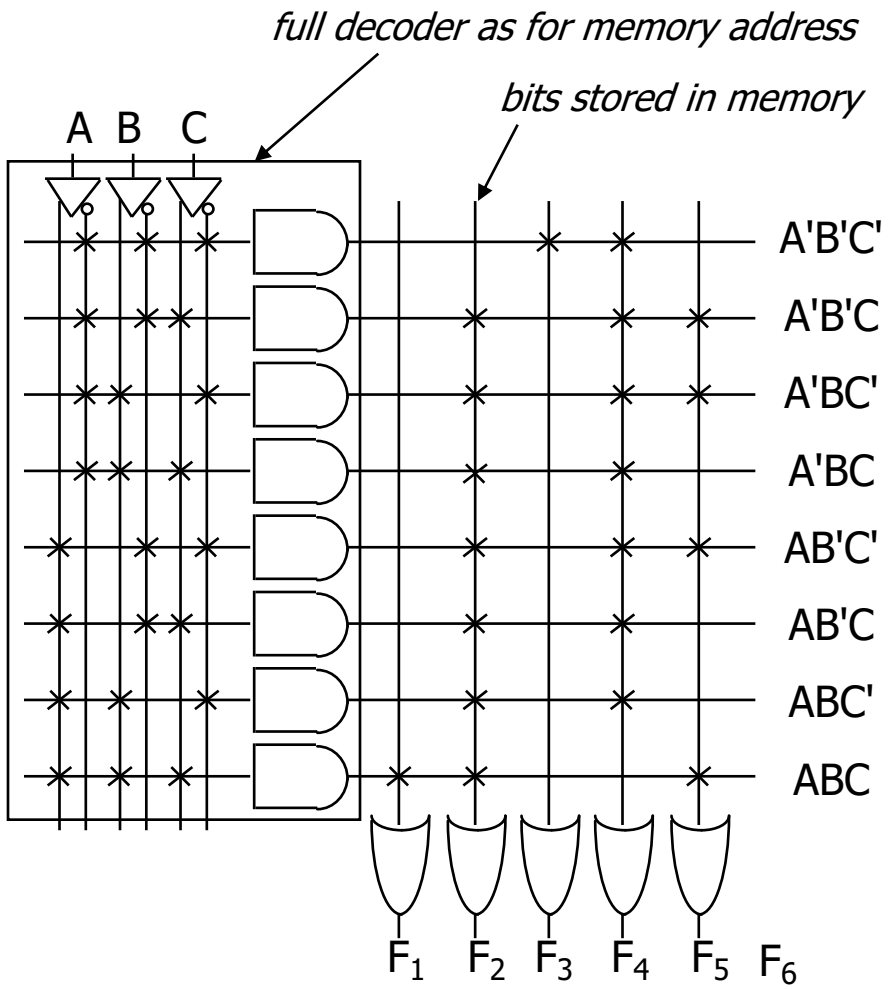
After programming:



One more example: a full decoder can be implemented in the programmable AND part

- Multiple functions of A, B, C
  - F_1 = A B C
  - F_2 = A + B + C
  - F_3 = A' B' C'
  - F_4 = A' + B' + C'
  - F_5 = A xor B xor C
  - F_6 = A xnor B xnor C

A	B	C	F_1	F_2	F_3	F_4	F_5	F_6
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1	1



PLA evaluation

- A k x m x n PLA can implement up to n functions of k inputs, each of which must be expressible with no more than m product terms.
- Unlike ROMs, PLAs allow you to choose which products are generated.
  - This can significantly reduce the fan-in (number of inputs) of gates, as well as the total number of gates.
  - However, a PLA is less general than a ROM. Not all functions may be expressible with the limited number of AND gates in a given PLA.
- In terms of memory, a k x m x n PLA has k address lines, and each of the 2^k addresses references an n-bit data value (so, word size is n bits).
- But again, not all possible data values can be stored.

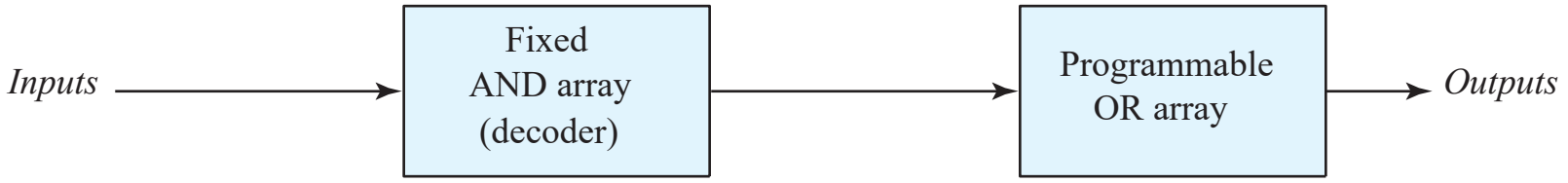
Functions and memories

- ROMs and PLAs give us two more ways to implement functions.
  - A circuit implies that some **calculation** has to be done on the inputs in order to arrive at the output. If the same inputs are given again, we have to repeat that calculation.
  - A truth table lists all possible combinations of inputs and their corresponding outputs. Instead of doing a potentially lengthy calculation, we can just "**look up**" the result of a function.
- The idea behind using a ROM or PLA to implement a function is to "store" the function's truth table, so we don't have to do any (well, very little) computation.
- This is like "memorization" or "caching" techniques in programming.

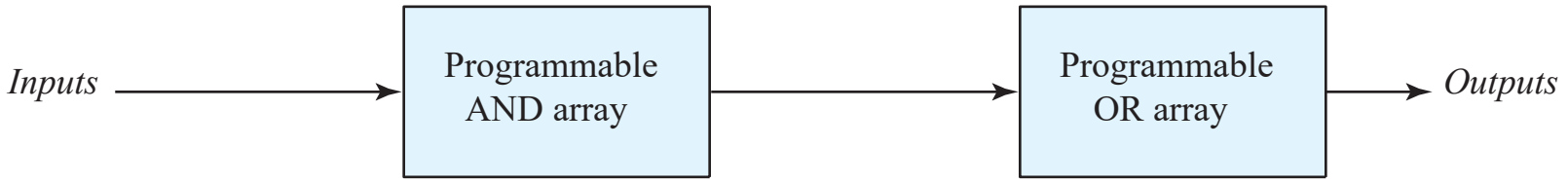
Important points so far…

- There are two main kinds of random access memory.
  - Static RAM costs more, but the memory is faster. Static RAM is often used to implement cache memories.
  - Dynamic RAM costs less and requires less physical space, making it ideal for larger-capacity memories. However, access times are also slower.
- ROMs and PLAs are programmable devices that can implement arbitrary functions, which is equivalent to acting as a read-only memory.
  - ROMs are simpler to program, but contain more gates.
  - PLAs use less hardware, but it requires some effort to minimize a set of functions. Also, the number of AND gates available can limit the number of expressible functions.

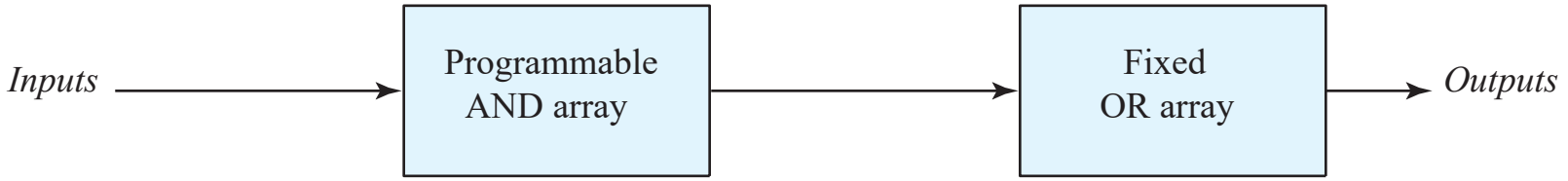
Programmable Logic Devices (PLDs)



(a) Programmable read-only memory (PROM)



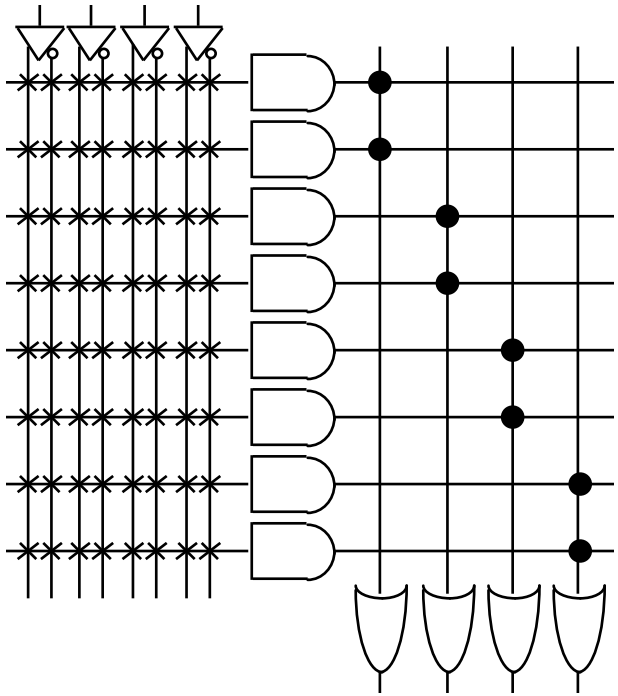
(c) Programmable logic array (PLA)



(b) Programmable array logic (PAL)

PALs and PLAs

- Programmable logic array (PLA)
  - what we've seen so far
  - unconstrained fully-general AND and OR arrays
- Programmable array logic (PAL)
  - Seen on the right
  - constrained topology of the OR array
  - faster and smaller OR plane

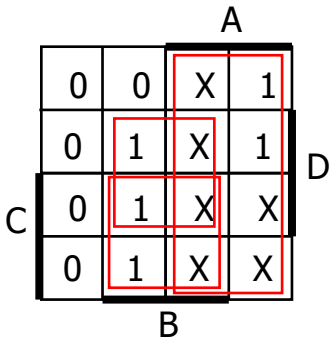


a given column of the OR array has access to only a subset of the possible product terms

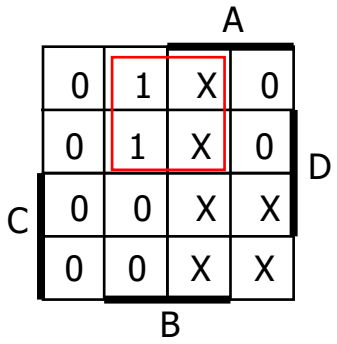
PALs and PLAs: Design Example

- BCD to Gray code converter

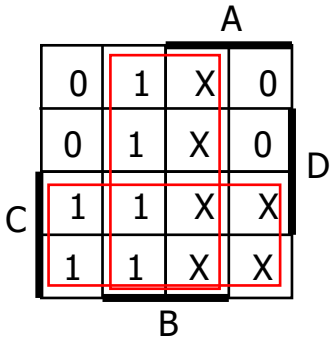
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	–	–	–	–	–
1	1	–	–	–	–	–	–



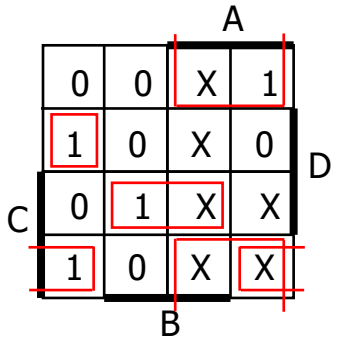
K-map for **W**



K-map for **X**



K-map for **Y**



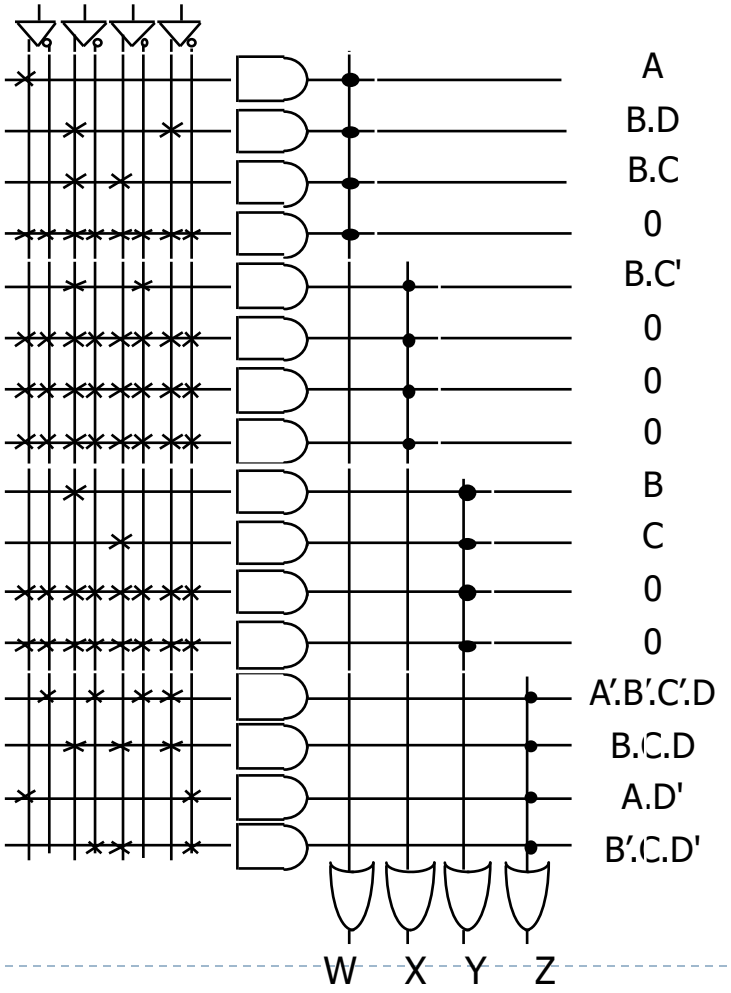
K-map for **Z**

Minimized Functions:

W = A + B D + B C
X = B C'
Y = B + C
Z = A' B' C' D + B C D + A D' + B' C D'

PALs and PLAs: design example (cont'd)

- Code converter: programmed PAL



Minimized Functions:

W = A + B.D + B.C
X = B.C'
Y = B + C
Z = A'.B'.C'.D + B.C.D + A.D' + B'.C.D'

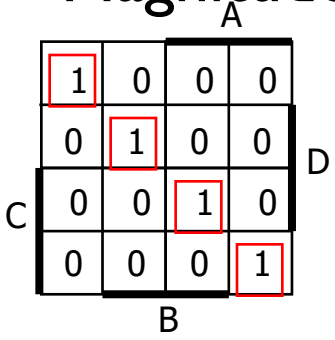
not a particularly good candidate for PAL/PLA implementation since no terms are shared among outputs

however, much more compact and regular implementation when compared with discrete AND and OR gates

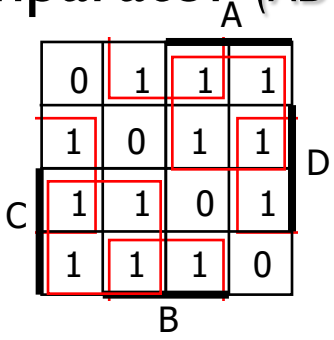
PALs and PLAs: Another Design Example

- Magnitude comparator *(AB vs. CD)*

Is the following a PAL or a PLA?



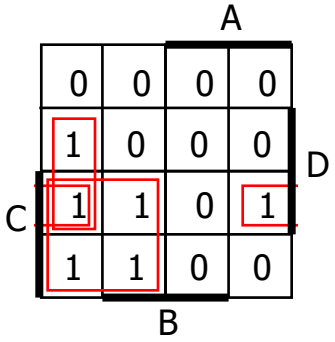
K-map for EQ



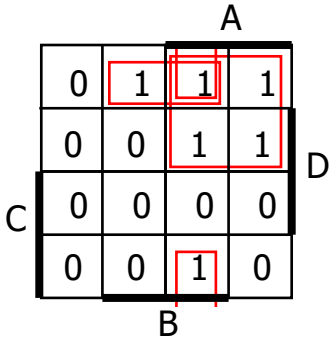
K-map for NE

EQ = A'B'C'D' + A'BC'D + ABCD + AB'CD'

NE = AC' + B'D + BD' + A'C



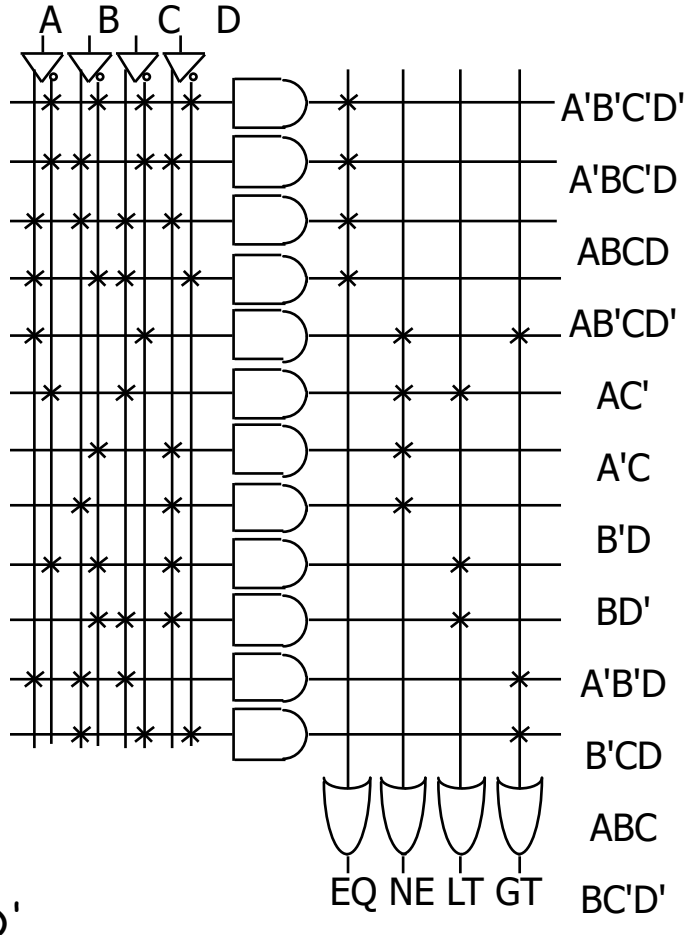
K-map for LT



K-map for GT

LT = A'B'D + A'C + B'CD

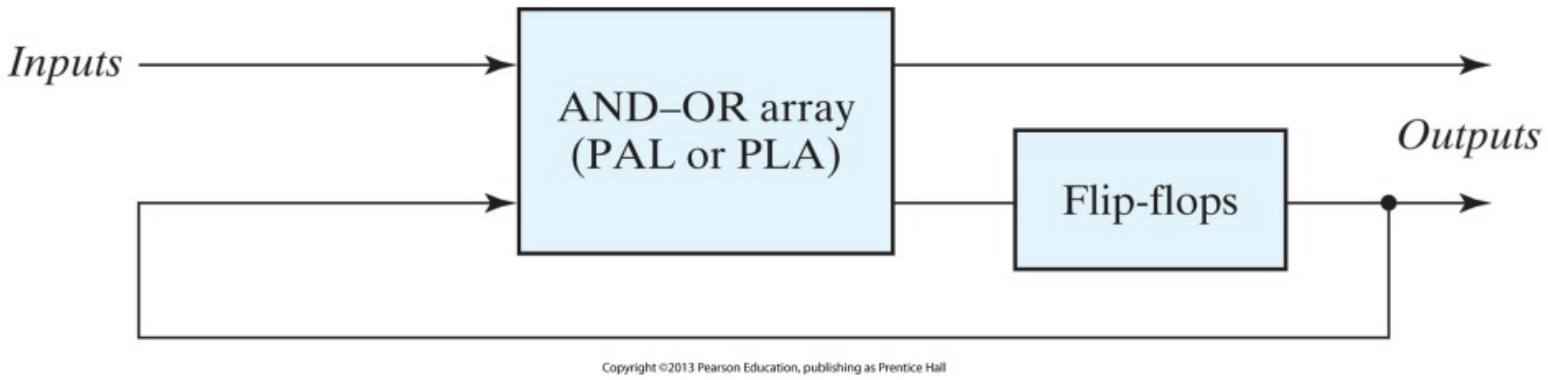
GT = BC'D' + AC' + ABD'



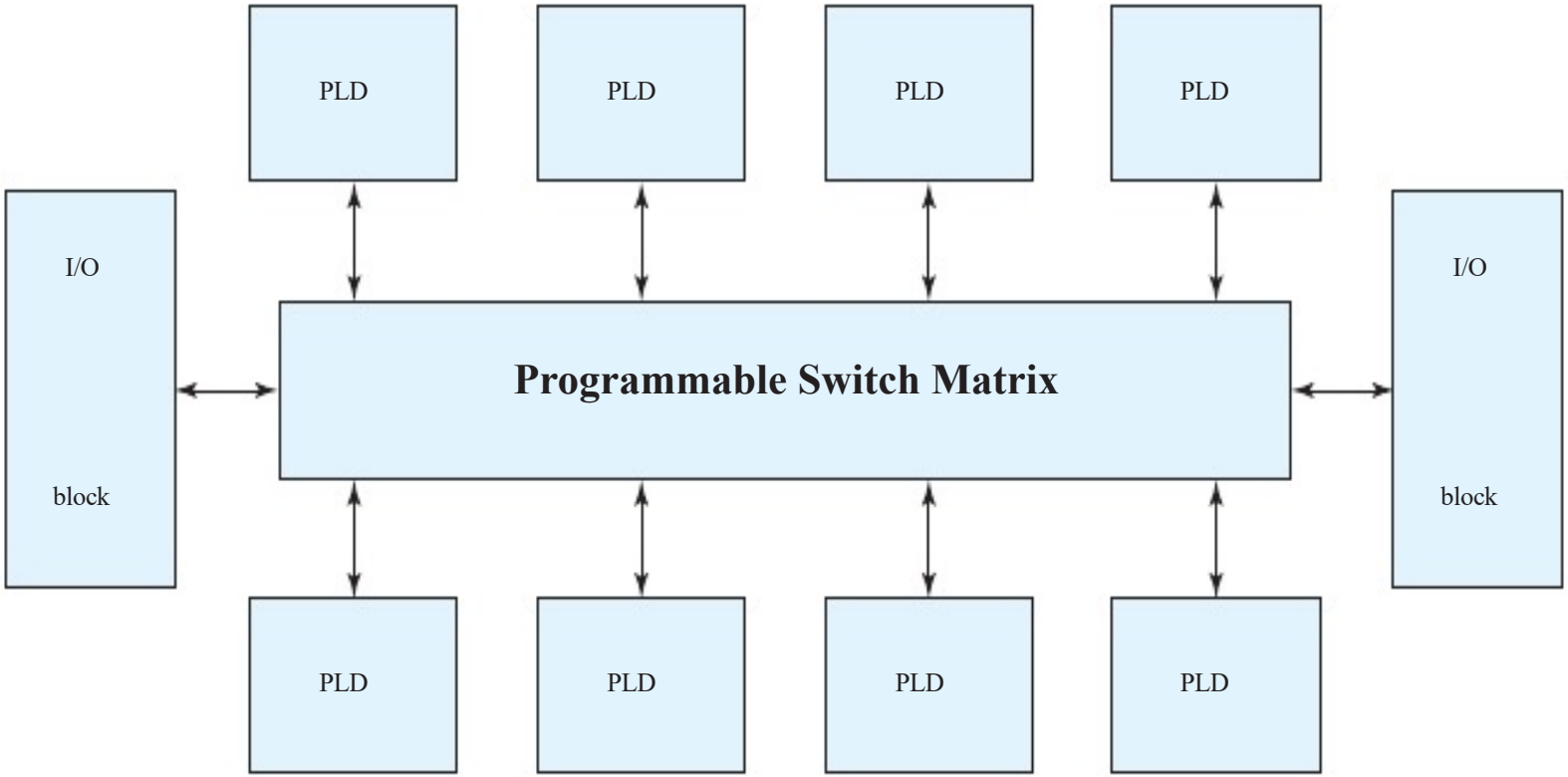
Sequential programmable devices

- Consists of programmable combinational logic + memory
- There are several types:
- 1. Sequential (or simple) programmable logic devices (SPLD)
- 2. Complex programmable logic devices (CPLD)
- 3. Field-programmable gate array (FPGA)

SPLD



CPLD


PLD
PLD
PLD
PLD
I/O
block
Programmable Switch Matrix
I/O
block
PLD
PLD
PLD
PLD

FPGA

- A VLSI circuit that can be programmed at the user's location.
- A typical FPGA consists of an array of millions of logic blocks, surrounded by programmable input and output blocks and connected together via programmable interconnections.
- A typical FPGA logic block consists of lookup tables, multiplexers, gates, and flip-flops.