Introduction to Computing Systems
from bits & gates to C & beyond

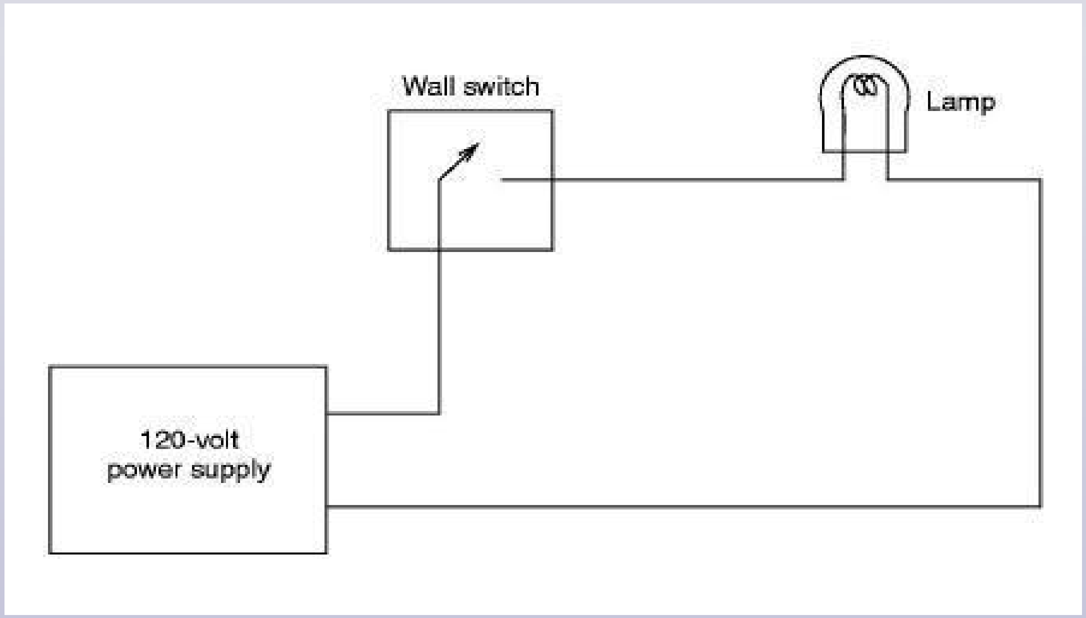
Chapter 3

Digital Logic Structures

- *Transistors*
- *Logic gates & Boolean logic*
- *Combinational logic*
- *Storage Elements*
- *Memory*

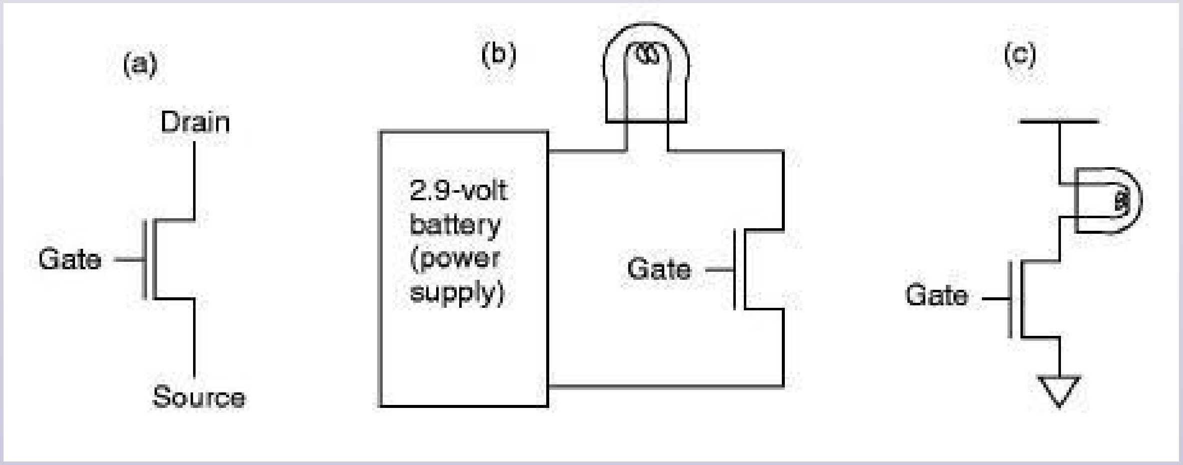
Electronic ones and zeros

- An electronic switch
  - like a light switch or faucet
  - switches between insulator (open circuit) and conductor (closed circuit)
  - We can call the presence of a voltage “1” and its absence “0”



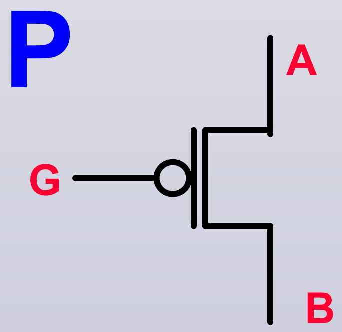
Transistors

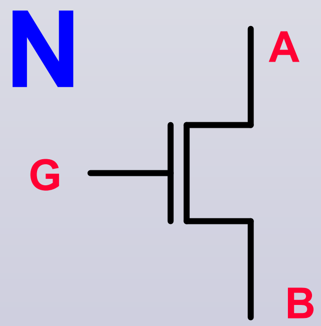
- **An electronic switch that is open or closed between the *source* and the *drain* depending on the voltage on the *gate*.**



CMOS Transistors

- **CMOS**
  - **= Complementary Metal-Oxide Semiconductor**
  - **Standard type for digital applications**
  - **Two versions: P-type (positive) and N-type (negative)**
  - **P and N-type transistors operate in inverse modes**

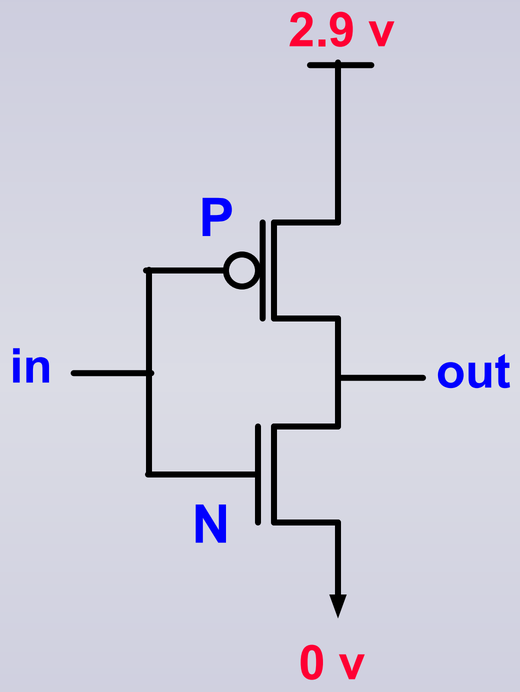




Open (insulating) if gate is “on” = 1
Closed (conducting) if gate is “off” = 0

Open (insulating) if gate is “off” = 0
Closed (conducting) if gate is “on” = 1

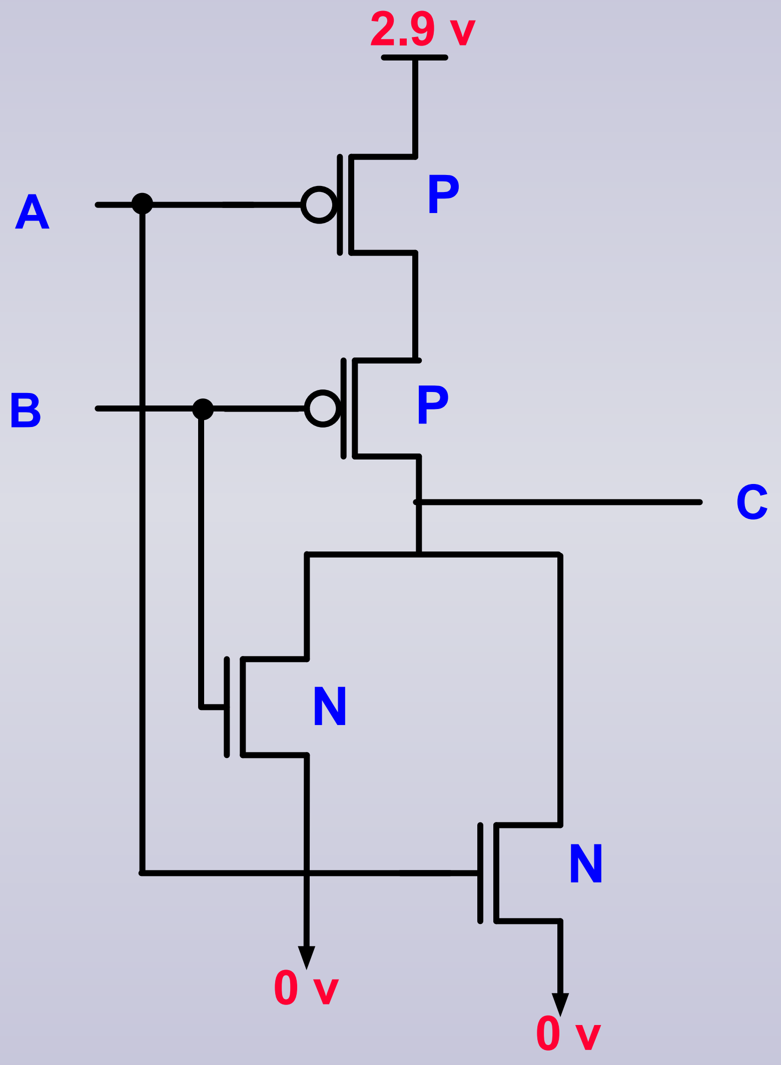
Inverter Gate



In	Out
0	1
1	0

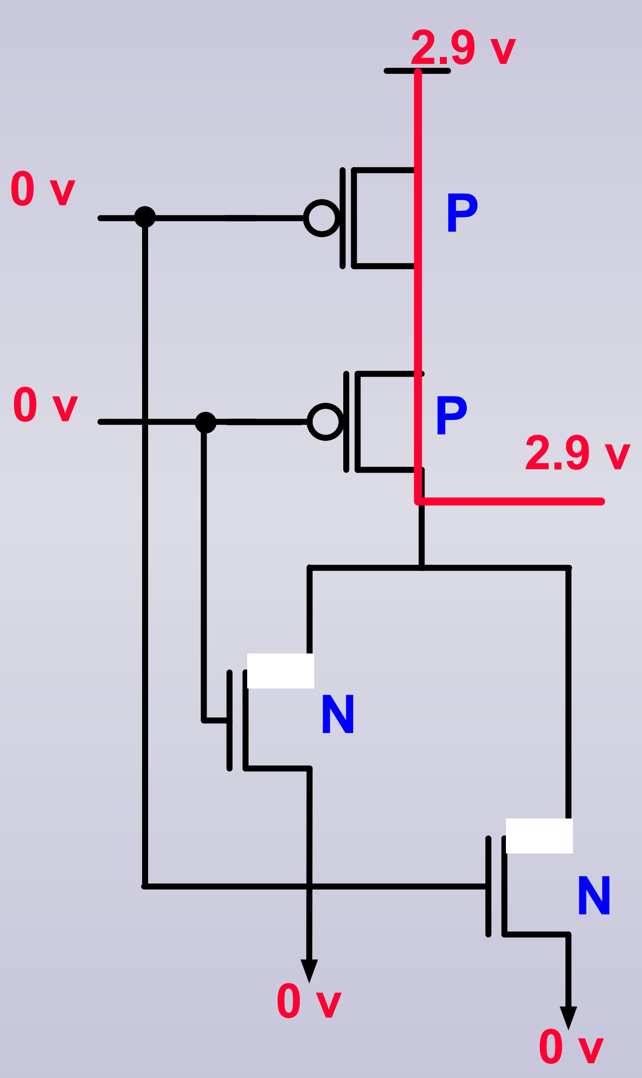
- When the input is on (in = high voltage), the P-type transistor is *open* and the N-type is *closed*, so the output is off (out = low voltage).
- Vice-versa: when the Input is off (in = low voltage), the output is connected to the high voltage.

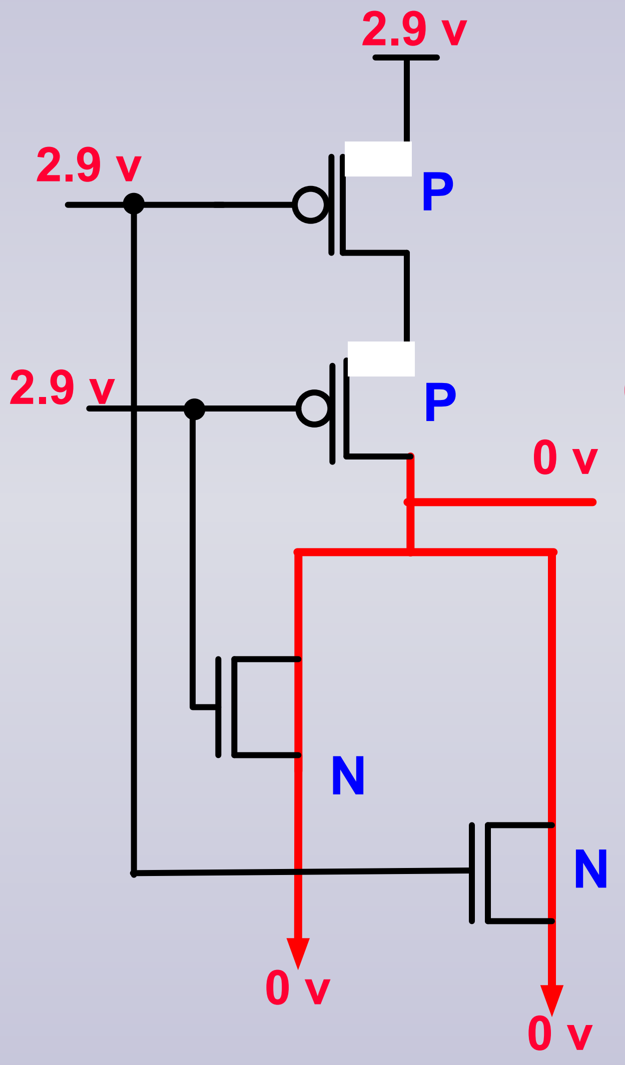
NOR Gate

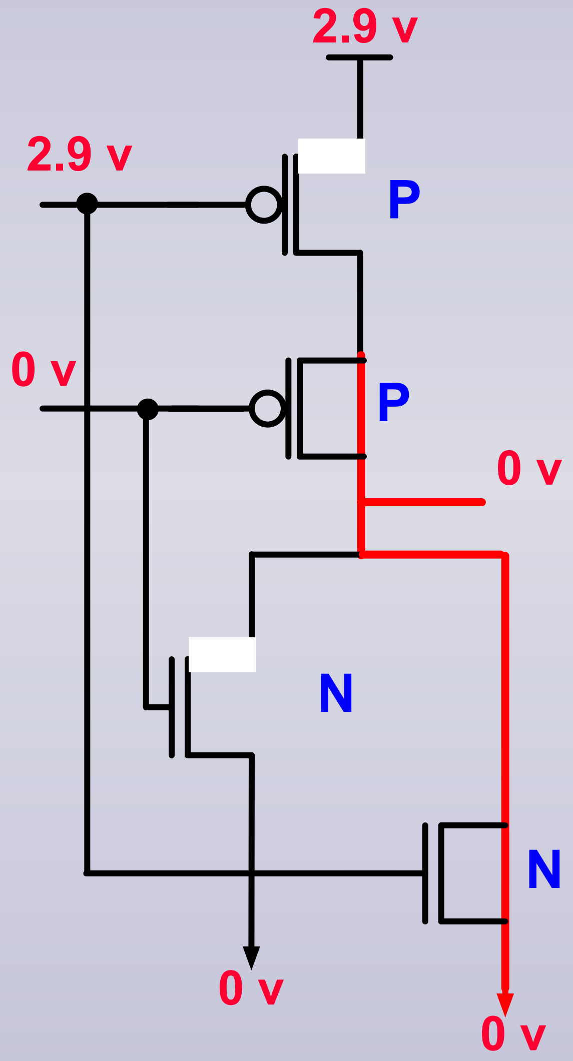


A	B	C
0	0	*1*
0	1	*0*
1	0	*0*
1	1	*0*

NOR Gate - Operation

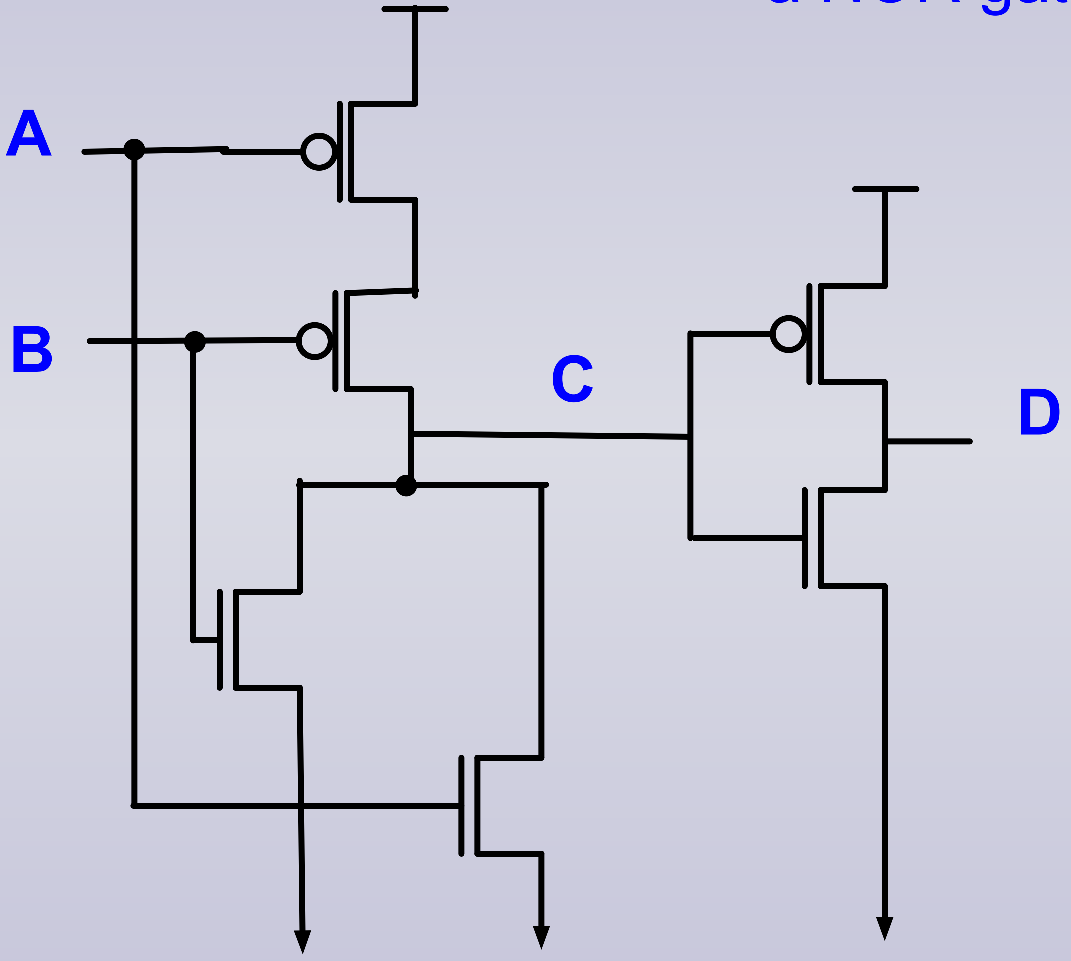






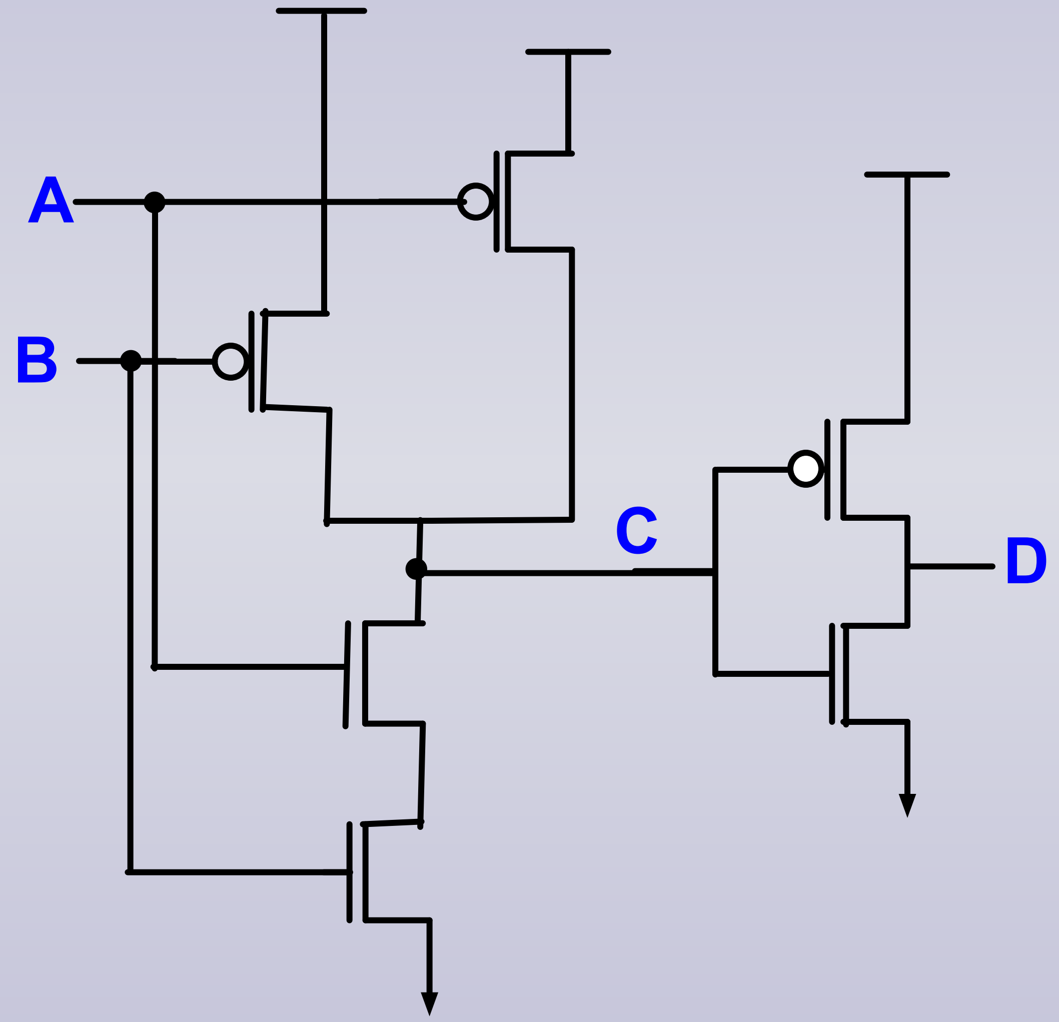
OR Gate

= a NOR gate followed by an inverter



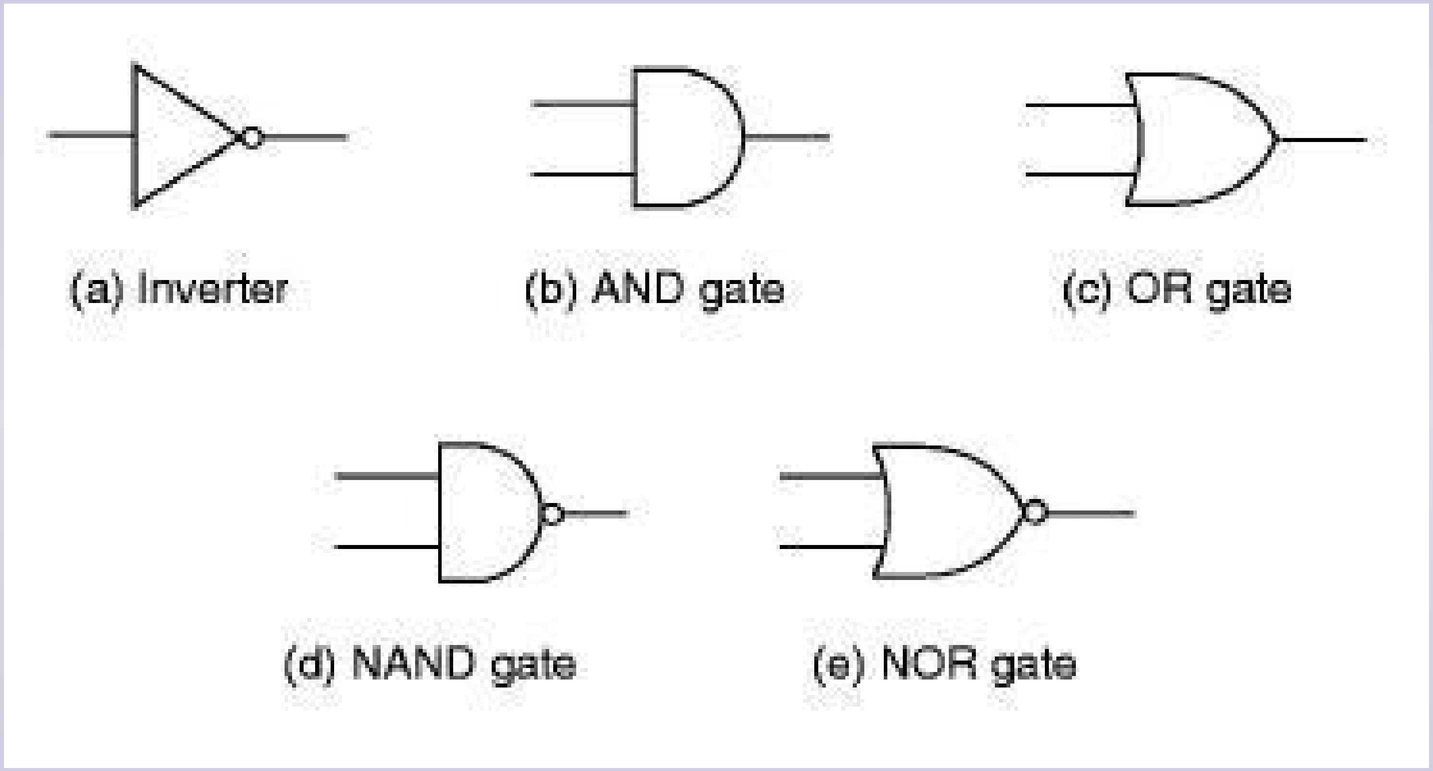
A	B	C	D
0	0	*1*	*0*
0	1	*0*	*1*
1	0	*0*	*1*
1	1	*0*	*1*

NAND & AND Gates



A	B	C	D
0	0	*1*	*0*
0	1	*1*	*0*
1	0	*1*	*0*
1	1	*0*	*1*

Logic Gates & Symbols

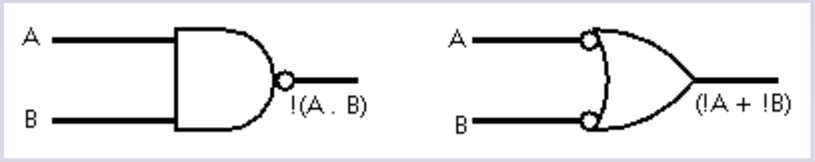


Note that gates can have more than 2 inputs.

De Morgan's Law

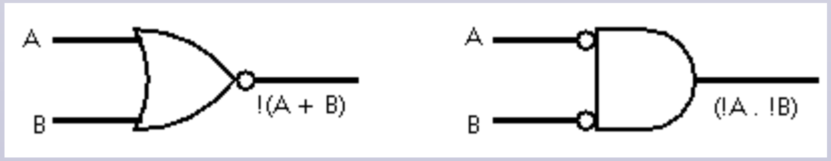
- not(A and B) = (not A) or (not B)

$$\overline{A \text{ and } B} = \overline{A} \text{ or } \overline{B}$$



- not(A or B) = (not A) and (not B)

$$\overline{A \text{ or } B} = \overline{A} \text{ and } \overline{B}$$



Completeness

- It can be shown that any truth table (i.e. any binary function of two variables) can be reduced to combinations of the AND & NOT functions, or of the OR & NOT functions.
    - *This result extends also to functions of more than two variables*

- In fact, it turns out to be convenient to use a basic set of three logic gates:
    - AND, OR & NOT

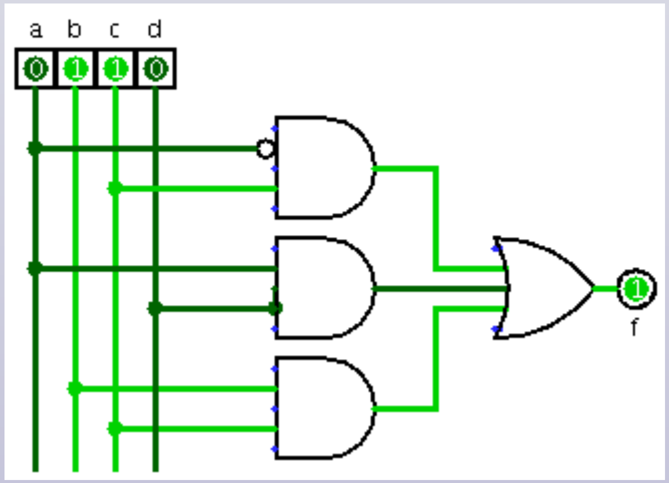
      or
    - NAND, NOR & NOT

Representation of Logic Functions

- A logic function can be represented as
  - a truth table
  - a logic expression
  - a logic circuit

- Example

a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



Types of Logic Structures

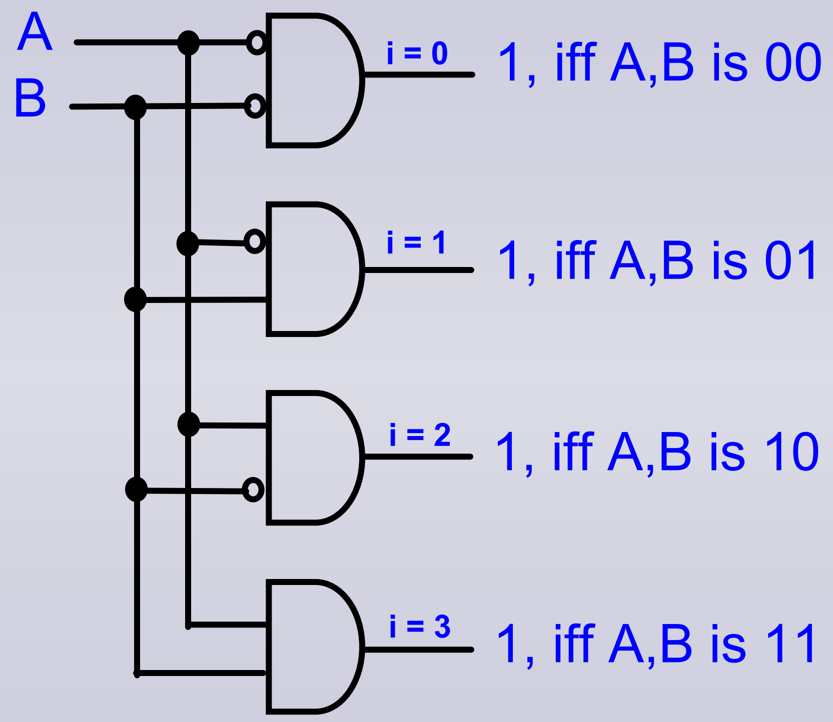
- Two types of logic structures ==> two types of logic circuits
  - *__Decision structures__: can make a decision based only on the current inputs: gates belong to this category.*
  - *__Storage structures__: permit the storage of information (as bits).*

- Combinational logic circuits
  - *a combinational logic structure is constructed from decision elements only: i.e. simple gates or other combinational logic circuits.*
  - *its output depends solely on its current input.*

- Sequential logic circuits
  - *combine combinational circuits & storage devices - we'll deal with these shortly.*

- Four examples of combinational logic circuits
  - *Decoder*
  - *Multiplexer (MUX)*
  - *Full adder*
  - *Programmable Logic Array*

Decoder

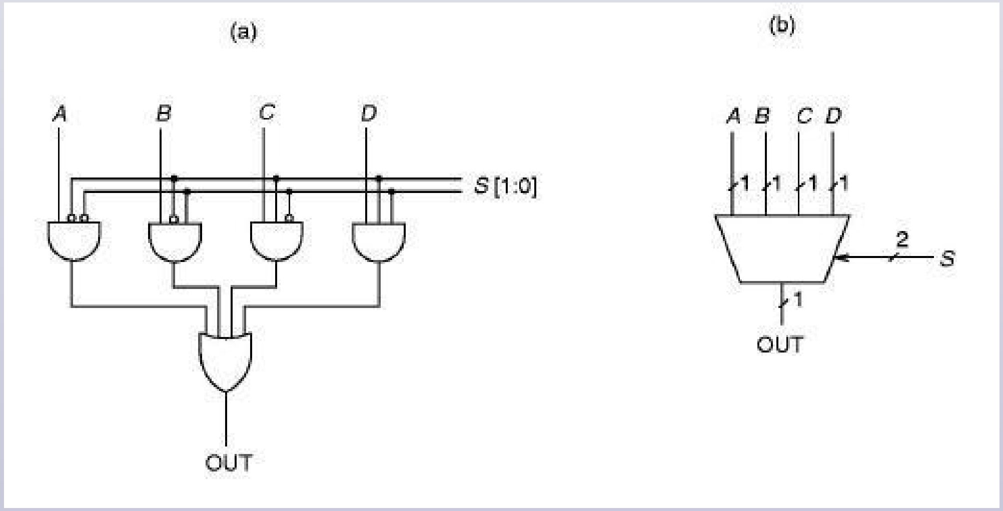


- An n input decoder has 2^n outputs.
- Output$_i$ is 1 iff the binary value of the n-bit input is i.
- At any time, exactly one output is 1, all others are 0.

Multiplexer (MUX)

- **In general, a MUX has**
  - ***2^n data inputs***
  - ***n select (or control) lines***
  - ***and 1 output.***
- **It behaves like a channel selector.**

$$Out = A.\overline{S_0}.\overline{S_1} + B.\overline{S_0}.S_1 + C.S_0.\overline{S_1} + D.S_0.S_1$$



A 4-to-1 MUX: Out takes the value of A,B, C or D, depending on the value of S (00, 01, 10, 11)

Adder

- **Half Adder**
  - 2 inputs
  - 2 outputs: sum and carry

a_i	b_i	c_{i+1}	s_i
0	0	*0*	*0*
0	1	*0*	*1*
1	0	*0*	*1*
1	1	*1*	*0*

Half-adder truth table

- **Full Adder**
  - performs the addition in column i
  - 3 inputs: a_i, b_i and c_i
  - 2 outputs: s_i and c_{i+1}
  - c_i is the carry in from bit position i-1
  - c_{i+1} is the carry out to bit position i+1

$$\begin{array}{rcccc} & a_{n-1} & a_{n-2} & \ldots & a_1 & a_0 \\ + & b_{n-1} & b_{n-2} & \ldots & b_1 & b_0 \\ + & c_{n-1} & c_{n-2} & \ldots & c_1 & 0 \\ \hline & s_{n-1} & s_{n-2} & \ldots & s_1 & s_0 \end{array}$$

Gate Level Full Adder

A_i	B_i	C_i	C_{i+1}	S_i
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

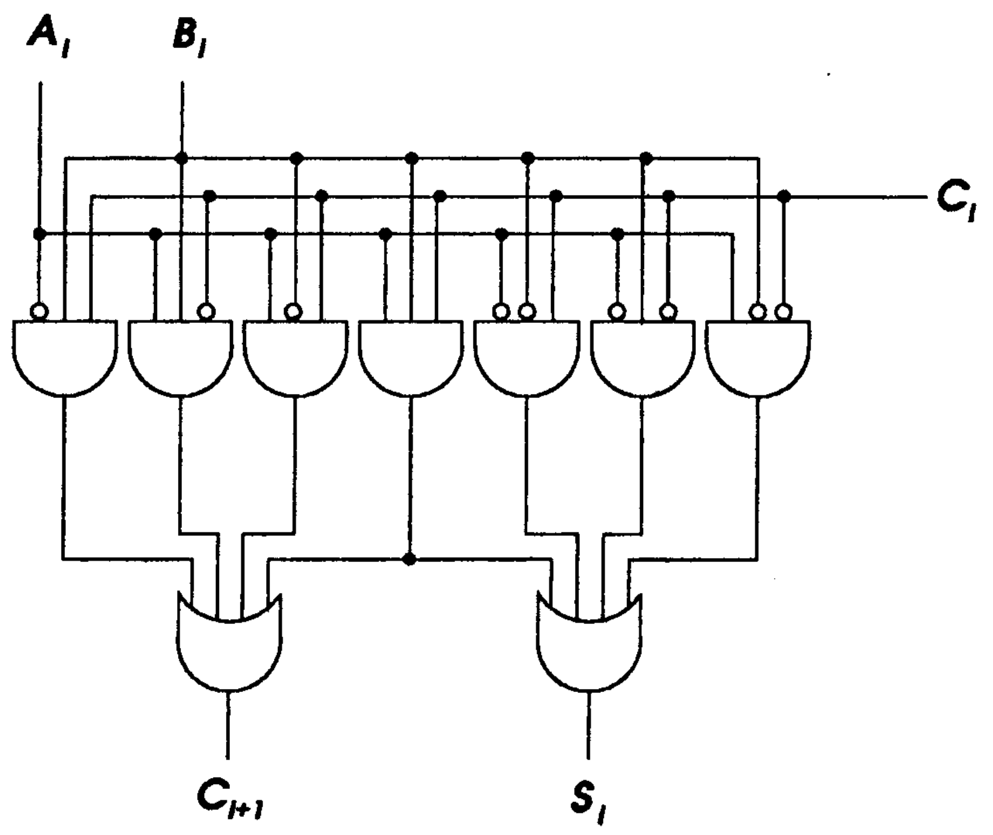


Figure 3.15: Gate Level Description of a Full Adder.

Full Adder - Expressions

$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i . b_i + c_i . (a_i + b_i)$$

where

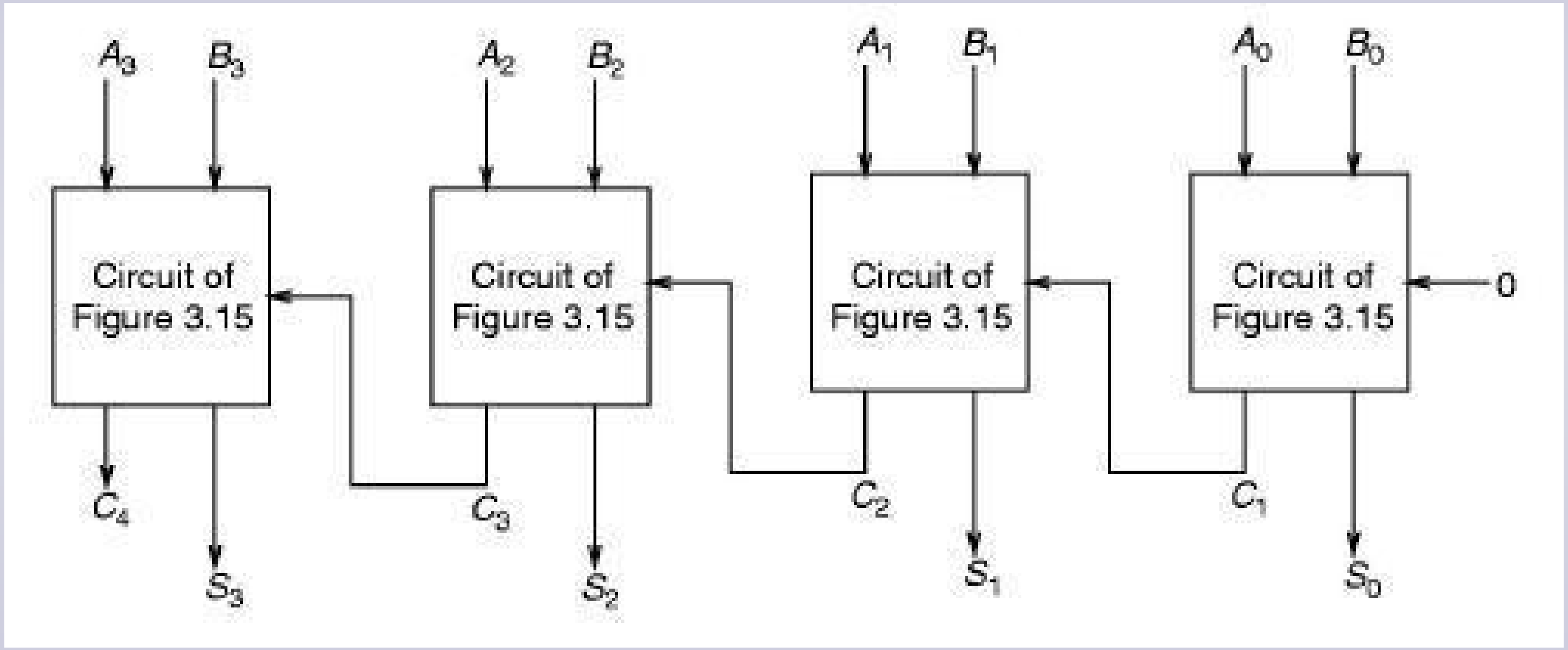
$\oplus$ is exclusive OR

. is the AND operation

$+$ is the OR operation

- verify that this corresponds to the gate-level implementation.

A 4-bit Ripple-Carry Adder



Carry Lookahead Addition

- **We can pre-compute the carry**
  - The carry in bit 4 (C_4) is 1 if any two of A_3, B_3 or C_3 are 1.

$$C_4 = A_3B_3 + C_3A_3 + C_3B_3$$
$$= A_3.B_3 + C_3(A_3 + B_3)$$
$$= G_3 + P_3C_3$$

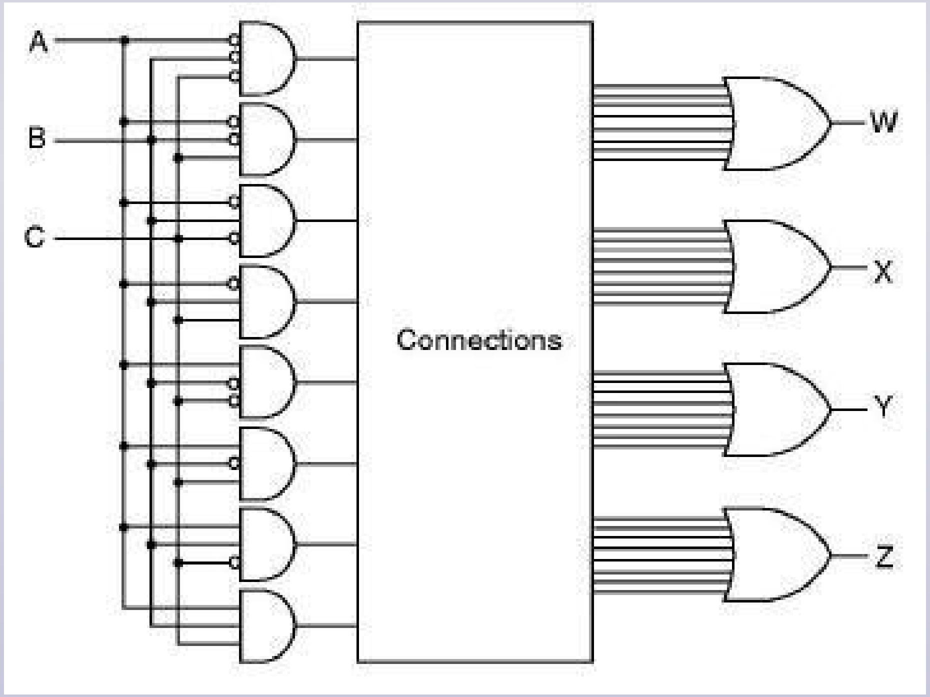
  - P_3 is called the propagate bit, and G_3 the generate bit

$$C_4 = G_3 + P_3C_3 = G_3 + P_3(G_2 + P_2C_2) = G_3 + P_3(G_2 + P_2(G_1 + P_1C_1))$$
$$= \cdots = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_1 + P_3P_2P_1C_0$$

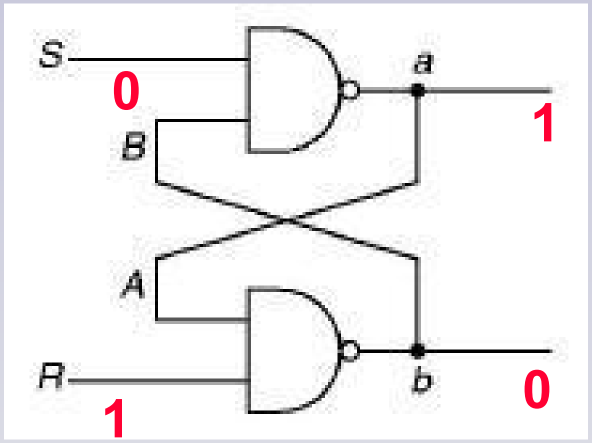
  - So every carry bit can be pre-computed using all the previous inputs.
  - Pre-computation can be done in 2 gate delays.

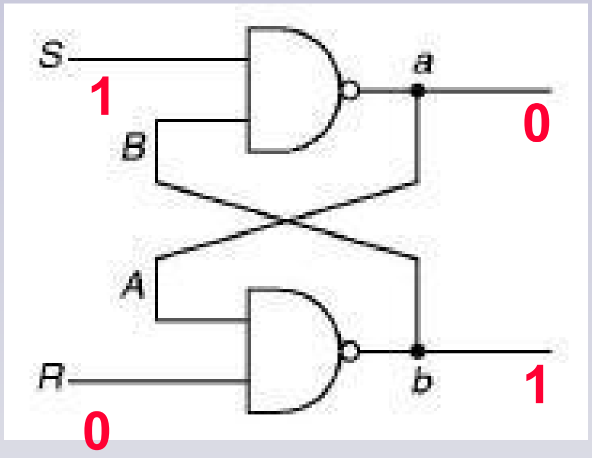
Programmable Logic Array

- It is possible to build a logic circuit that uses logic circuits to decide what logic circuits to implement!



Storage Elements: R-S Latch

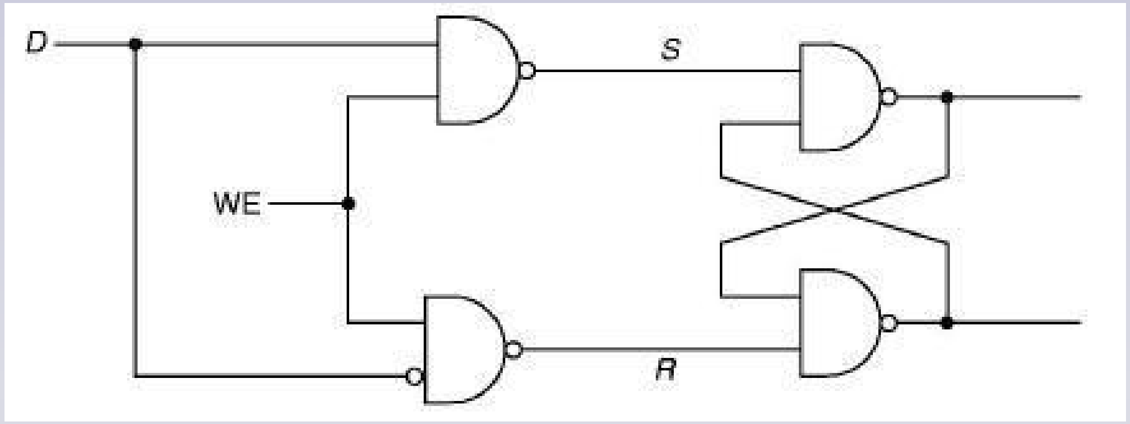




- The output *a* of the R-S latch can be set to 1 by momentarily setting S to 0 while keeping R at 1.
- When S is set back to 1 the output *a* stays at 1.
- Conversely, the output *a* can be set to 0 by keeping S at 1 and momentarily setting R to 0.
- When R is set back to 1, the output *a* stays at 0.

The flip-flop (R-S latch) is a bi-stable element

Storage Elements: Gated D Latch



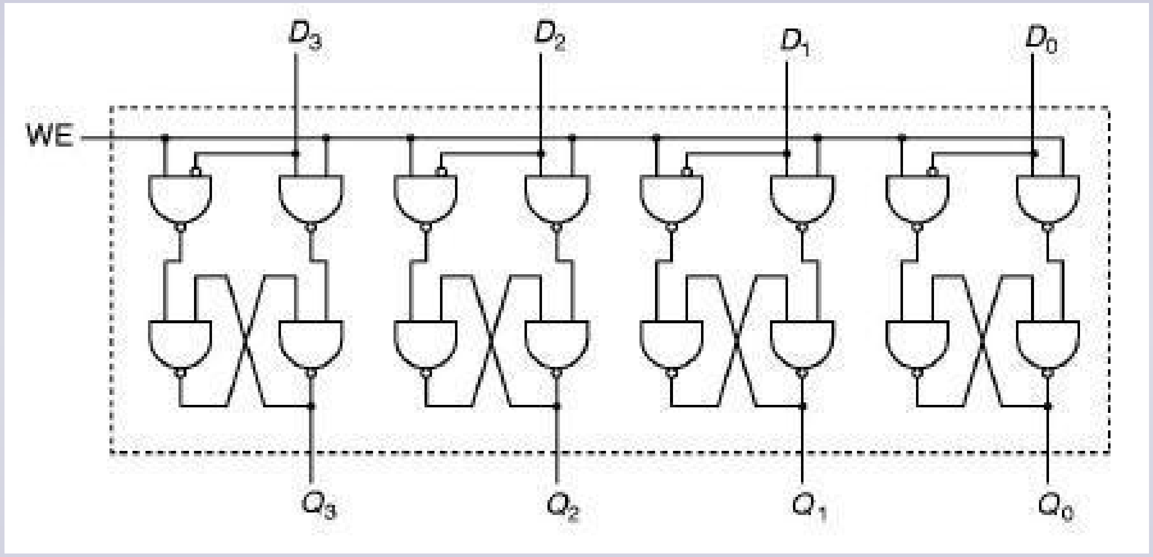
•The gated D latch is an extension of the R-S latch

•Two inputs: data (D) and write enable (WE)

•When the WE (write enable) is set to 1, the output of the latch
is set to the value of D.

•The output is held until WE is “asserted” (set to 1) again.

Registers



A 4-bit register made of four D latches

Memory - 1

A large number of addressable fixed size locations

- **Address Space**

n bits allow the addressing of 2^n memory locations.

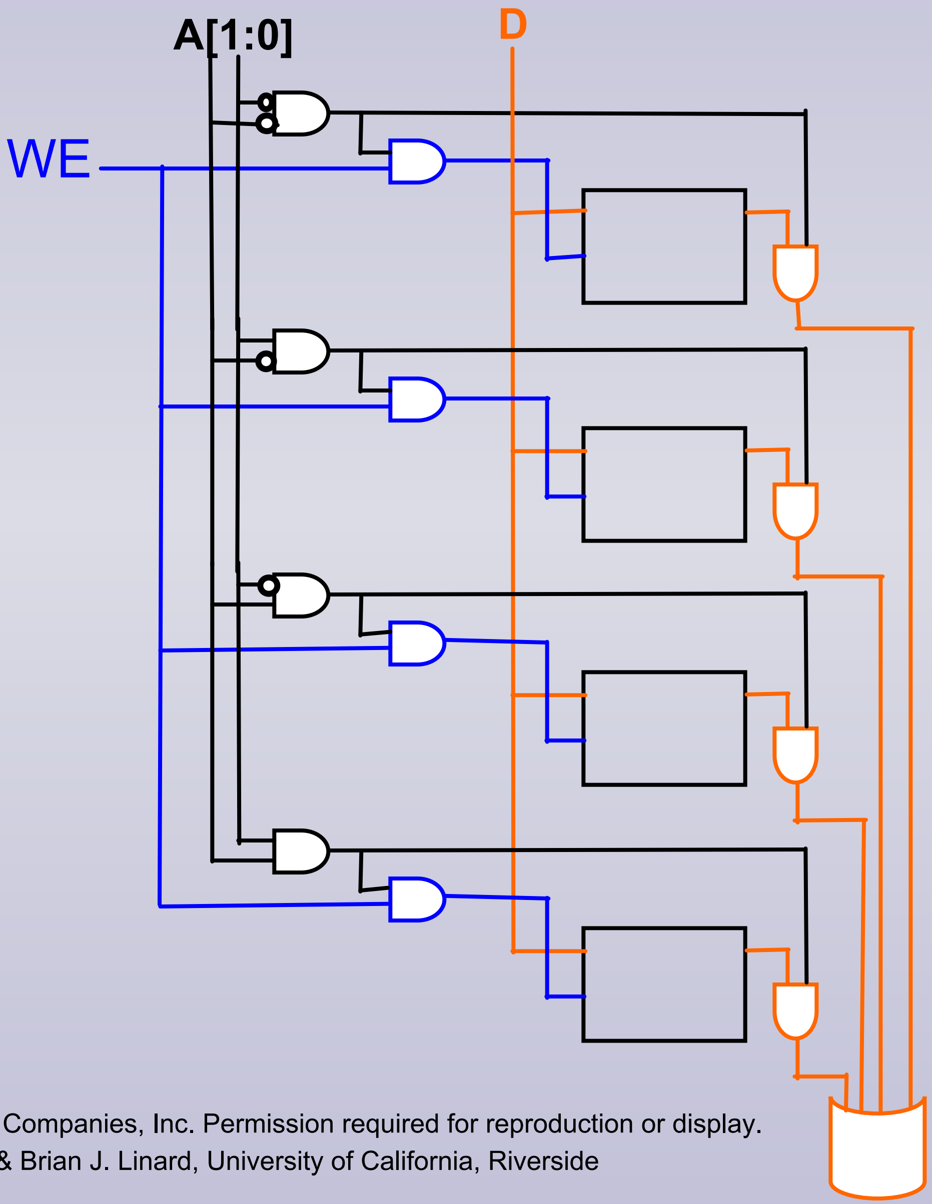
- **Example: 24 bits can address 2^{24} = 16,777,216 locations (i.e. 16M locations).**
- **If each location holds 1 byte (= 8 bits) then the memory is 16MB.**
- **If each location holds one word (32 bits = 4 bytes) then it is 64 MB.**

Memory - 2

- Addressability
  - Computers are either byte or word addressable - i.e. each memory location holds either 8 bits (1 byte), or a full standard word for that computer (16 bits for the LC-3, more typically 32 bits, though now many machines use 64 bit words).
  - Normally, a whole word is written and read at a time:
    - *If the computer is word addressable, this is simply a single address location.*
    - *If the computer is byte addressable, and uses a multi-byte word, then the word address is conventionally either that of its most significant byte (big endian machines) or of its least significant byte (little endian machines).*

Building a Memory

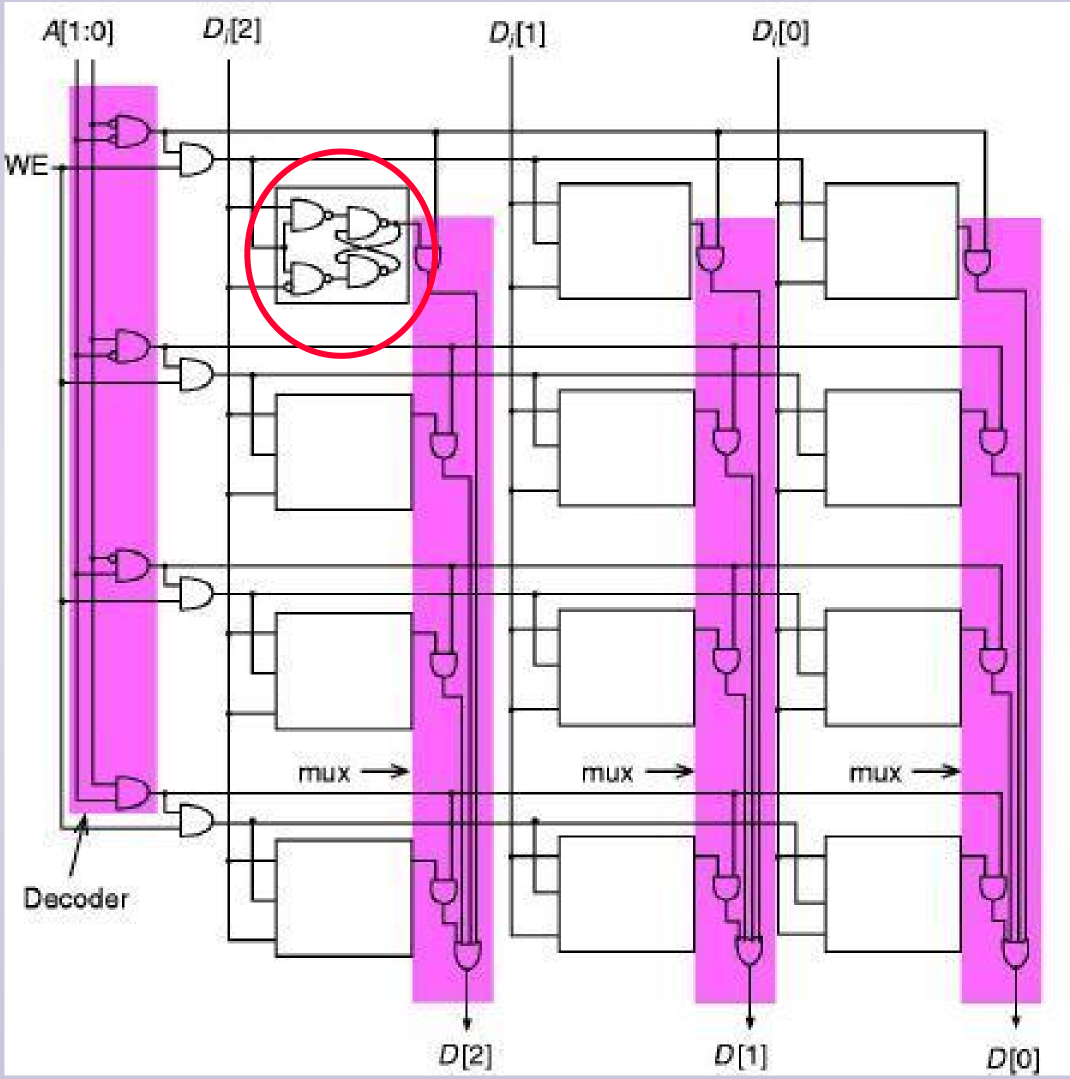
- **Each bit**
  - **is a gated D-latch**
- **Each location**
  - **consists of w bits (here w = 1)**
  - **w = 8 if the memory is byte addressable**
- **Addressing**
  - **n locations means $\log_2 n$ address bits (here 2 bits => 4 locations)**
  - **decoder circuit translates address into 1 of n locations**



Memory Example

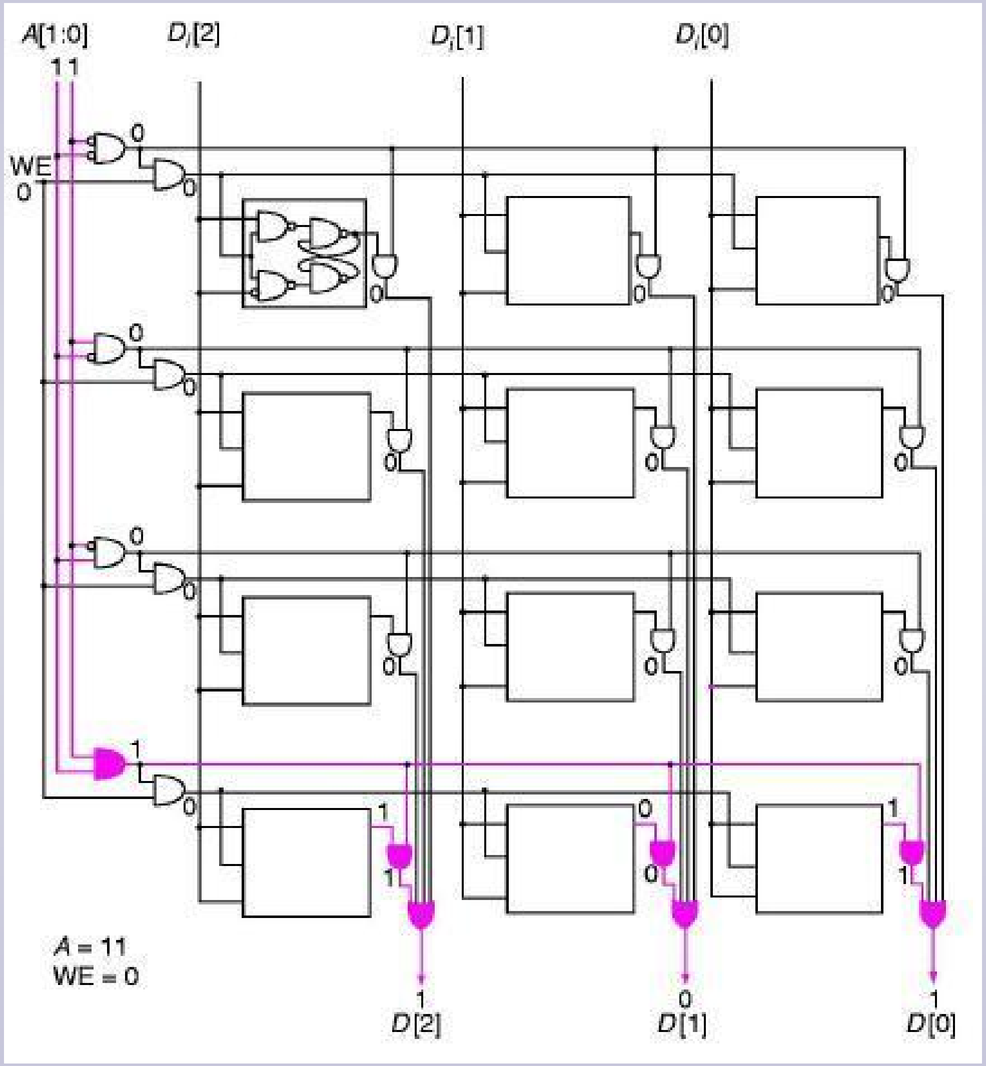
A 2^2 by 3 bits memory:

- two address lines: A[1:0]
- three data lines: D[2:0]
- one control line: WE



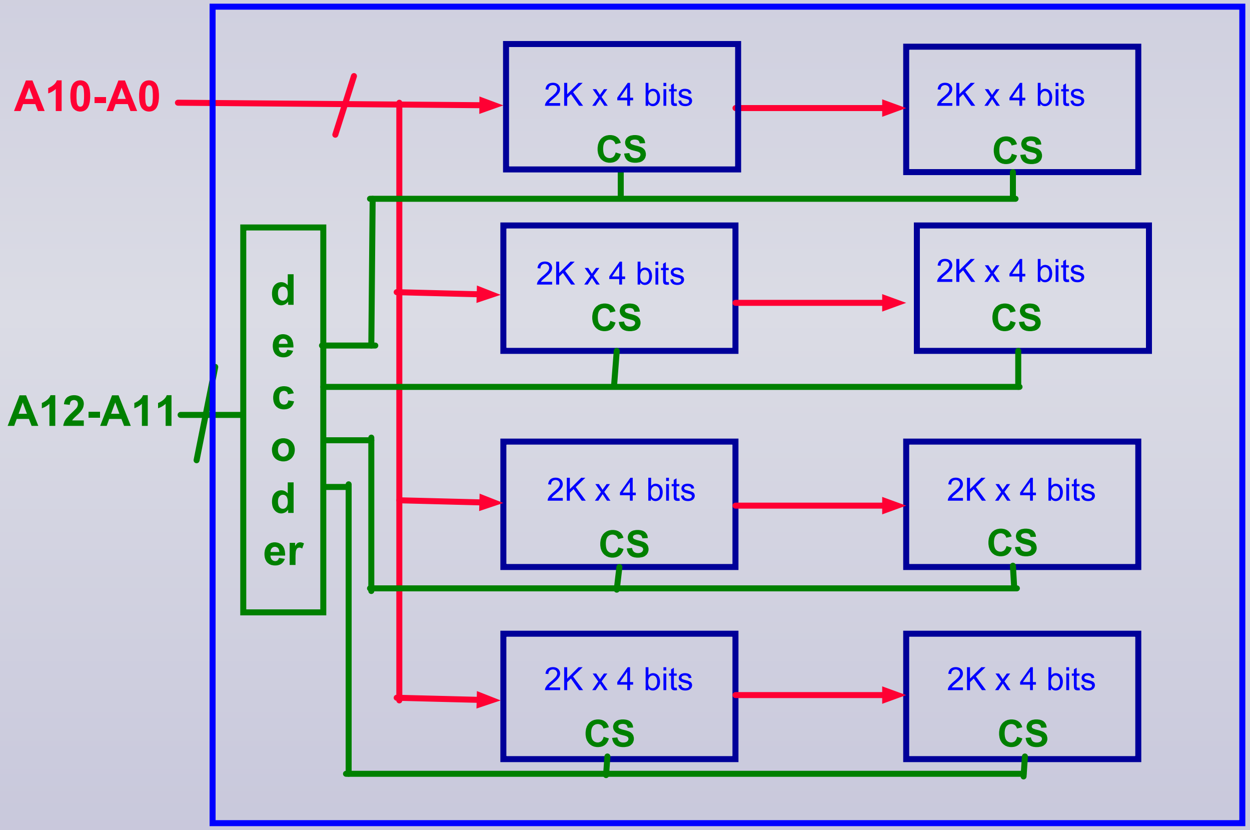
One gated D-latch

Reading a location in memory



Using Memory Building Blocks

- Building an 8K byte memory using chips that are 2K by 4 bits.

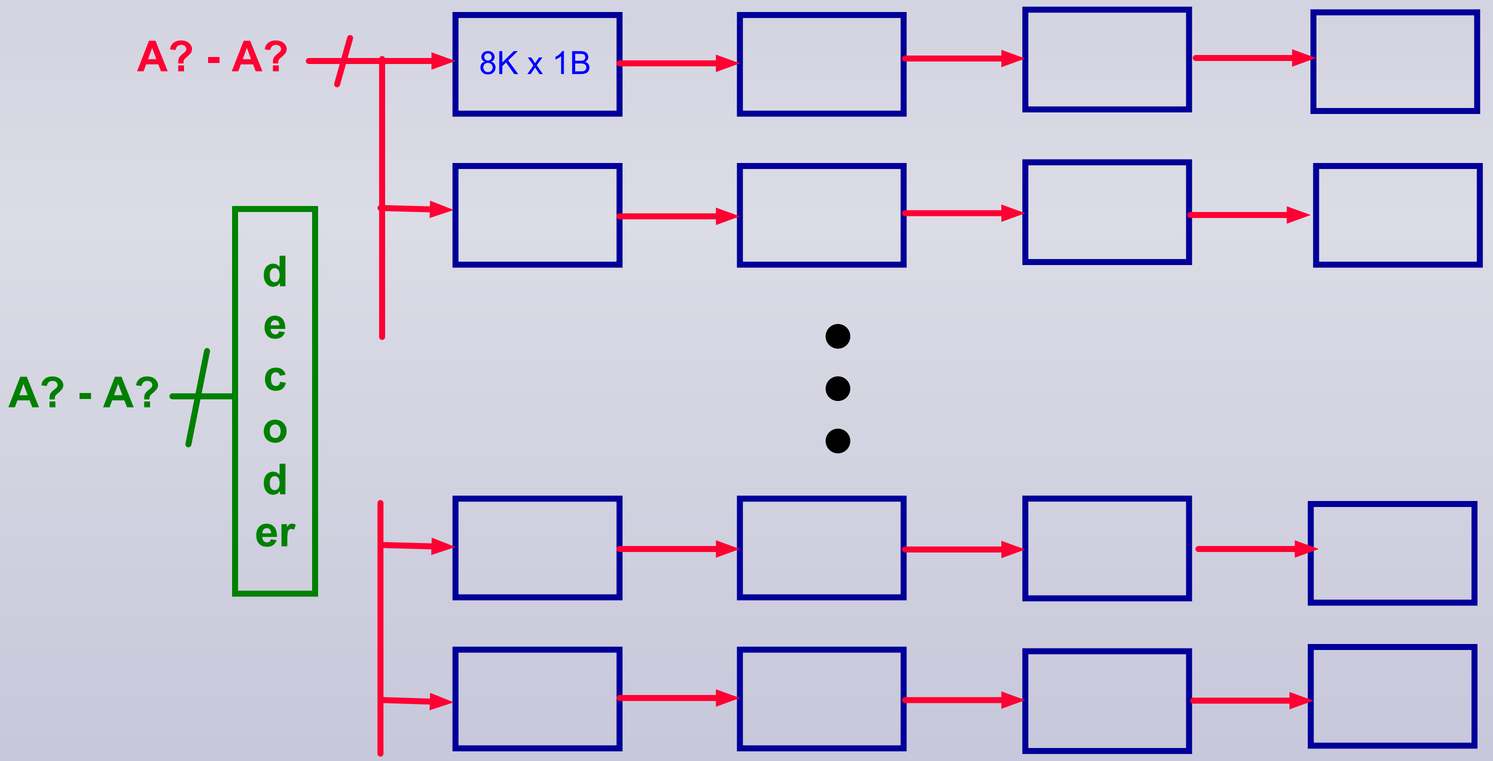


- CS = chip select: when set, it enables the addressing, reading and writing of that chip.

This is an 8KB byte addressable memory

Memory One Word Wide

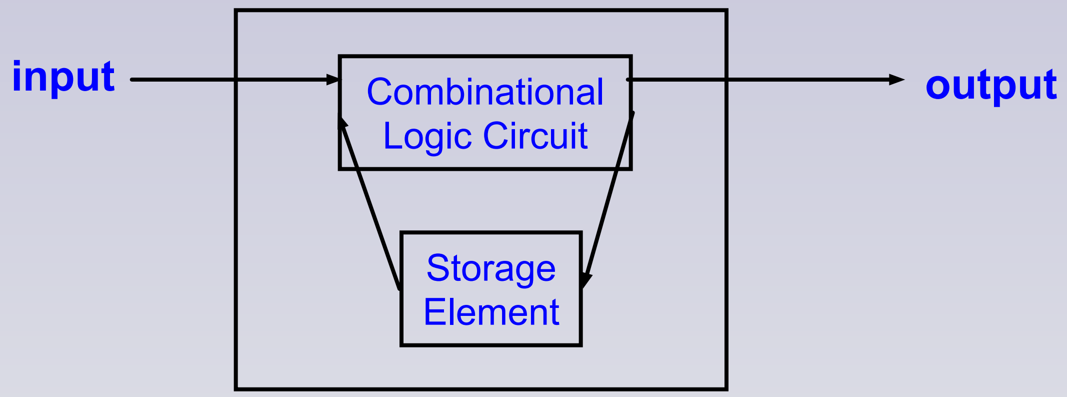
- Use the previous memory block of 8K x 1 byte to build a memory that is 64K words, with each location one word of 32 bits.
  - **what are the address lines if the memory is word addressed? or byte addressed?**



Sequential Logic Circuits - 1

- The concept of state
  - the state of a system is a “snapshot” of all relevant elements at a moment in time.
  - a given system will often have only a finite number of possible states.
  - e.g. the game of tic-tac-toe has only a certain number of possible dispositions of Xs and Os on the 3x3 grid.
  - A given game of tic-tac-toe will progress through a subset of these possible states (until someone wins) - i.e. it traverses a specific path through “state space”, one move at a time.
  - For many systems, we can define the rule which determine under what conditions a system can move from one state to another.

Sequential Logic Circuits - 2

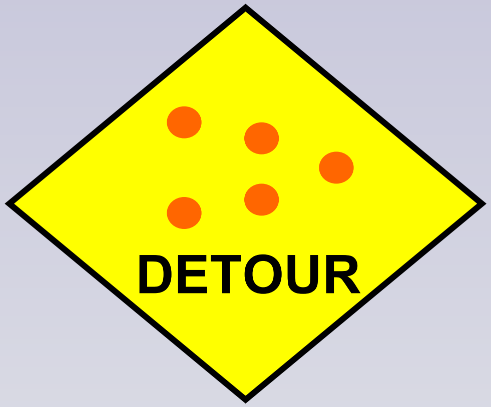


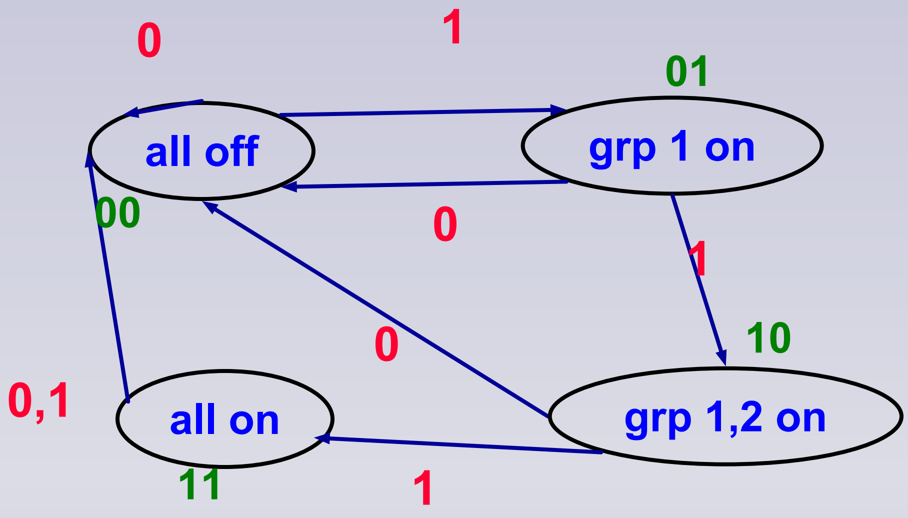
- The output is a function of the current input and the previous state
- It is computed by the *combinational logic circuit*
- The state is stored in the *storage element*
- The new state is also a function of the previous state and the current input
- This can work only if we make transitions from one state to another at well-defined times - this is why they are called sequential circuits.

Finite State Machines

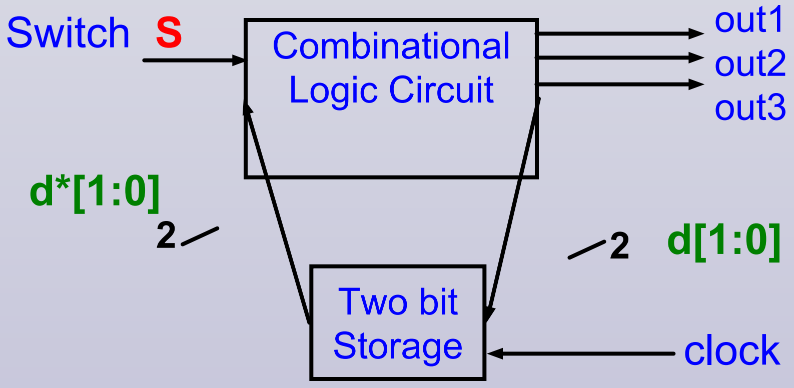
- Many systems meet the following five conditions:
  - A finite number of states
  - A finite number of external inputs
  - A finite number of external outputs
  - An explicit specification of all allowed state transitions
  - An explicit specification of the rules for each external output value
- In fact, as we will see, a microprocessor is a perfect candidate for description as a FSM.

Finite State Machine Example - 1





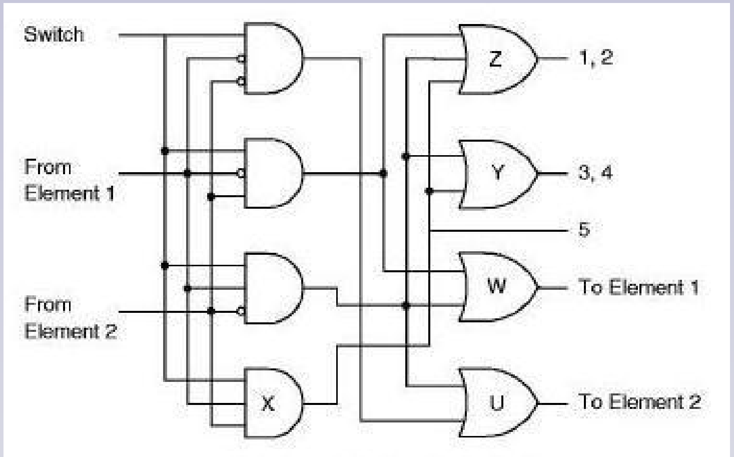
- Three groups of lights to be lit in a sequence: group 1 on, groups 1 & 2 on, all groups on, all off.
- The lights are on only if the main switch is on.
- <u>Four states:</u> so we need two bits to identify each state.



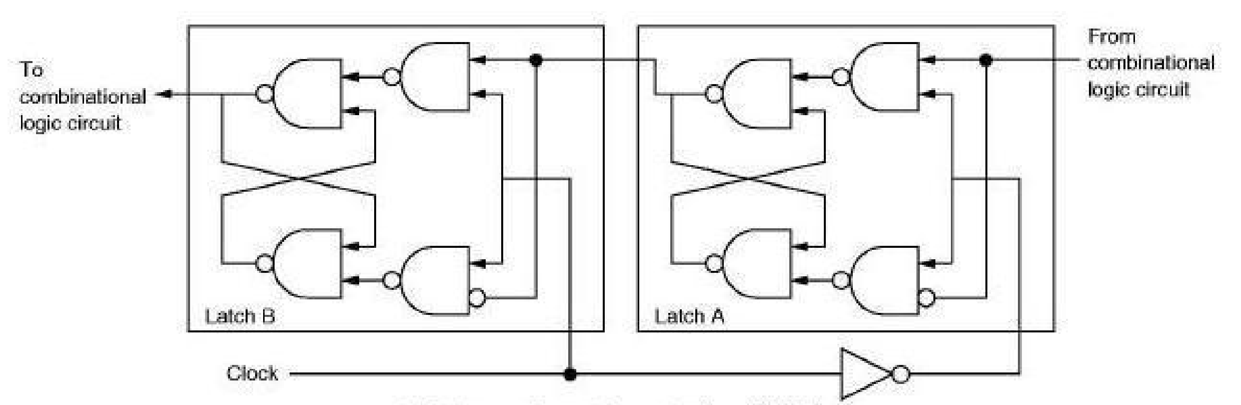
Finite State Machine Example - 2

- When is group 1 on?
  - in states 01, 10 and 11 - but only when the switch is on!
  - can you come up with a logic expression for d_0 and d_1?
- When do we switch to the next state?
  - the two bits of d[1:0] are updated at every clock cycle
  - we have to make sure that the new state does not propagate to the combinational circuit input until the next clock cycle.

Finite State Machine Example - 3



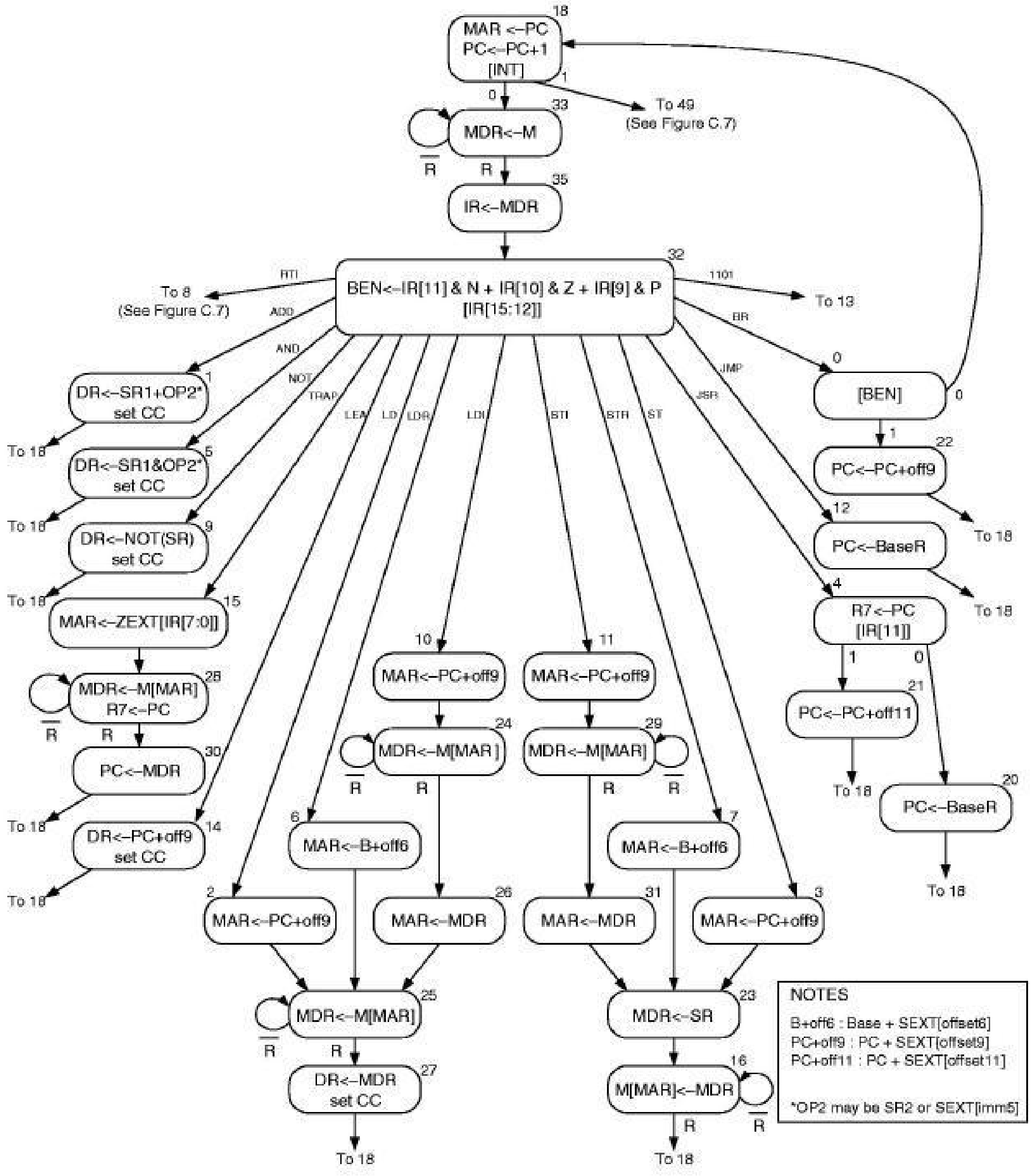
(b) The combinational logic circuit


To combinational logic circuit
From combinational logic circuit
Latch B
Latch A
Clock

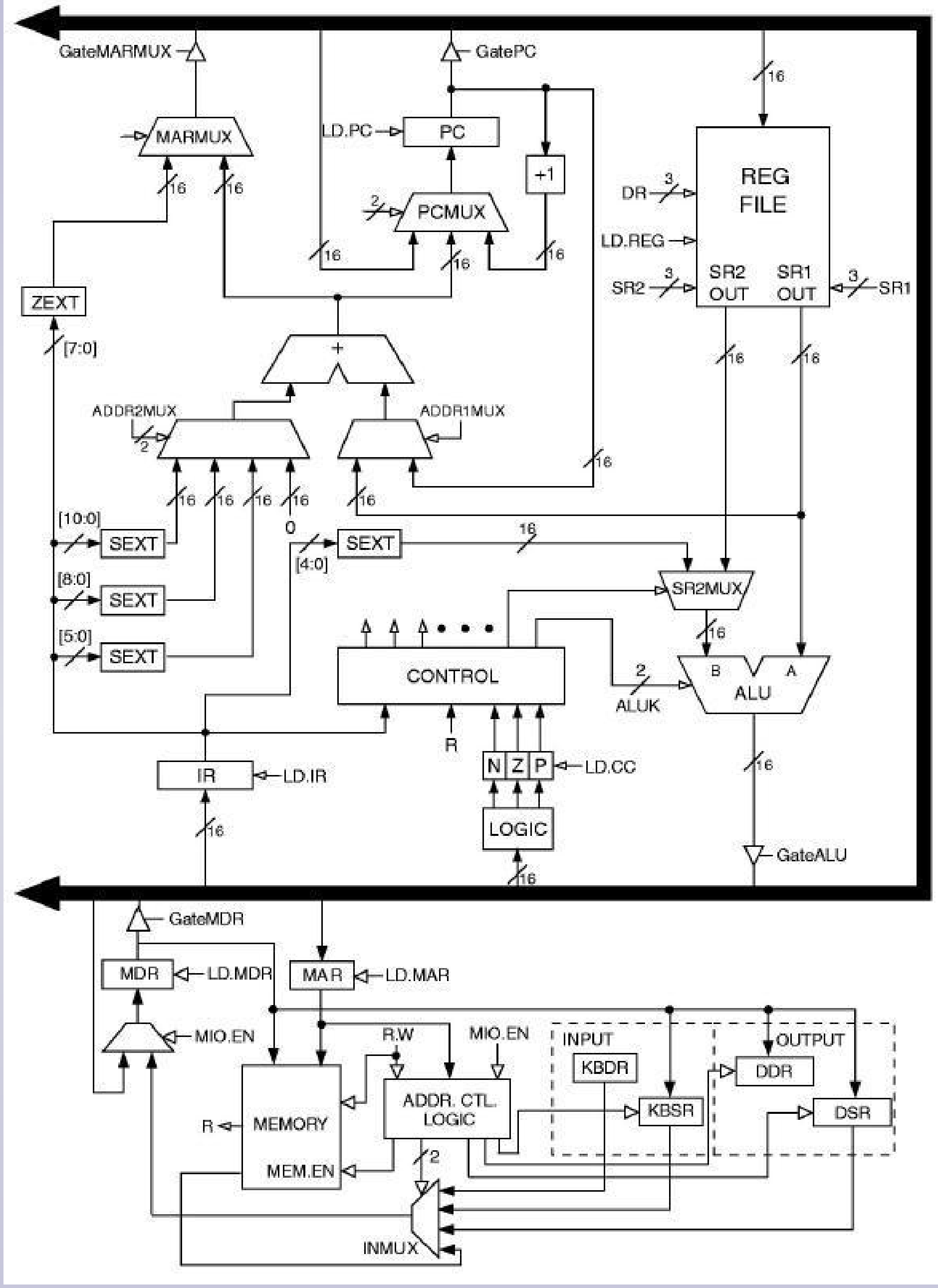
(c) A storage element (a master-slave flip-flop)

Note that signal "1,2" corresponds to "out1"; "3,4" to out2; "5" to out3; and Element 1 to d_1, Element 2 to d_0

The LC-3 as a Finite State Machine



Data Path of the LC-3



GateMARMUX
GatePC
MARMUX
LD.PC
PC
+1
REG FILE
DR
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
PCMUX
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
[10:0]
SEXT
[8:0]
[5:0]
[4:0]
SR2MUX
CONTROL
ALUK
ALU
R
N Z P
LD.CC
IR
LD.IR
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
MIO.EN
R.W
ADDR. CTL. LOGIC
INPUT
KBDR
KBSR
OUTPUT
DDR
DSR
MEMORY
MEM.EN
INMUX