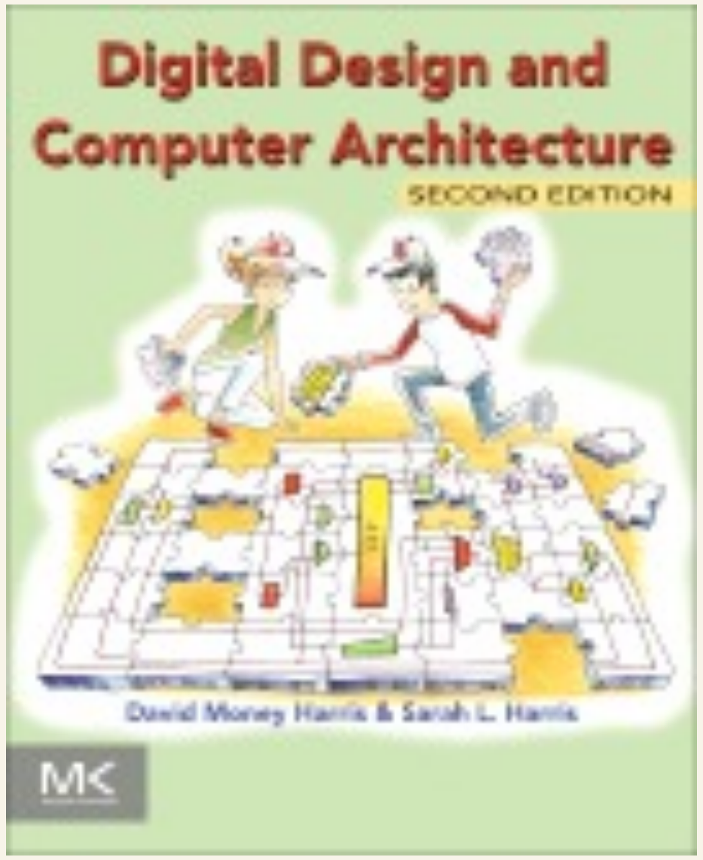
Digital Building Blocks

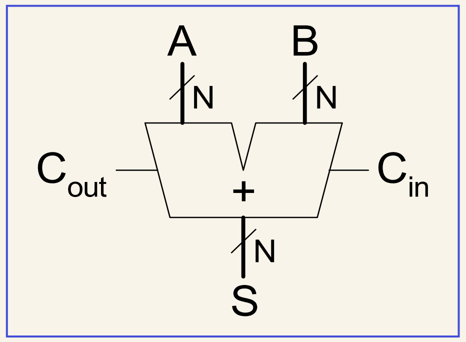
- Arithmetic Circuits
- Sequential Building Blocks
- Memory Arrays
- Logic Arrays
- Cyclone IV



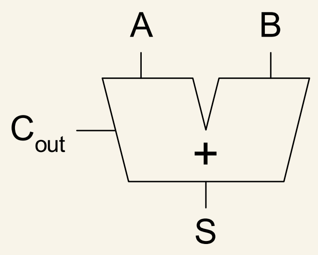
Chap 5

Arithmetic Circuits

- Adder
  - Several types of carry propagate adders (CPAs) are:
    - » Ripple-carry adders (slow)
    - » Carry-lookahead adders (fast)
    - » Prefix adders (faster)



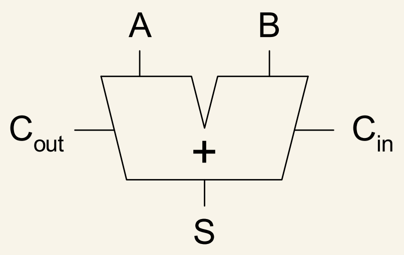
Half Adder



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$
$C_{out} = AB$

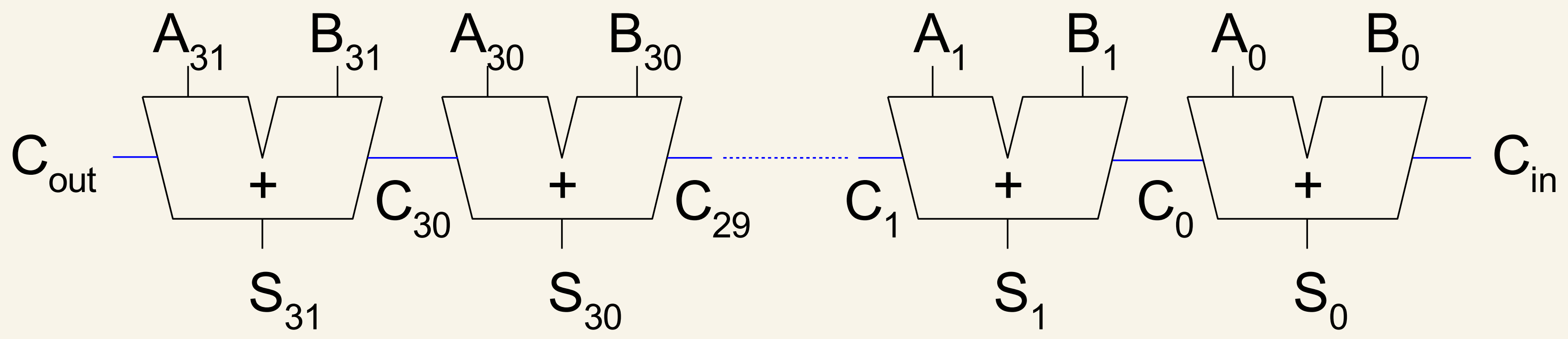
Full Adder



C_{in}	A	B	C_{out}	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

$S = A \oplus B \oplus C_{in}$
$C_{out} = AB + AC_{in} + BC_{in}$

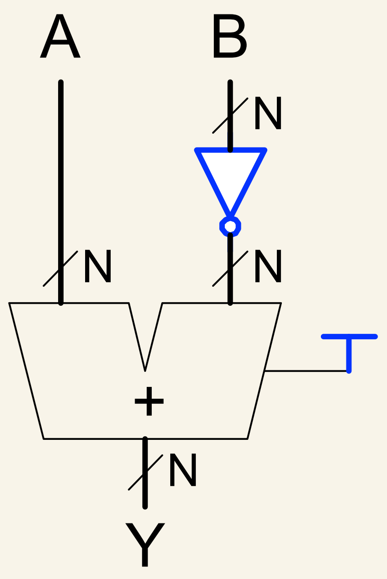
- Ripple-Carry Adder
  - Chain 1-bit adders together
  - Carry ripples through entire chain
  - Disadvantage: slow



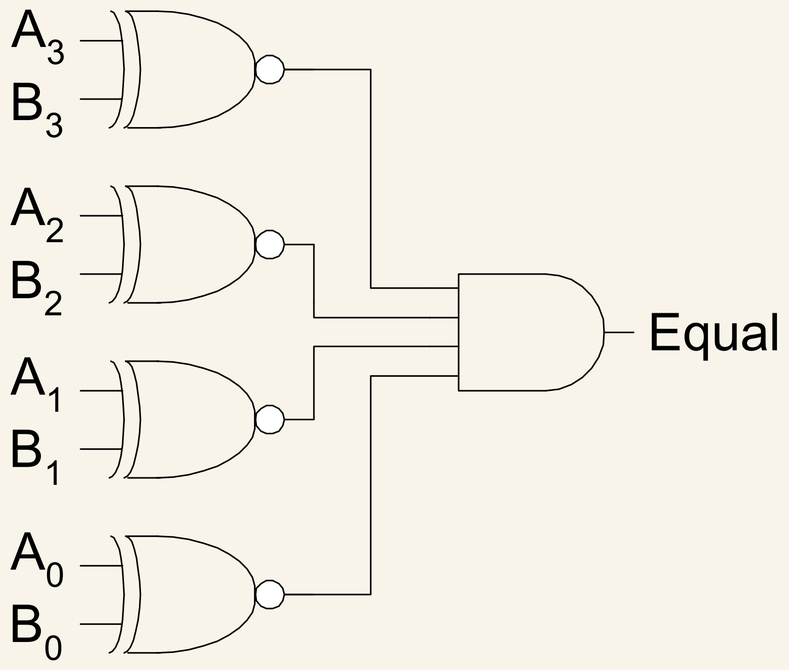
$$t_{ripple} = N\, t_{FA}$$

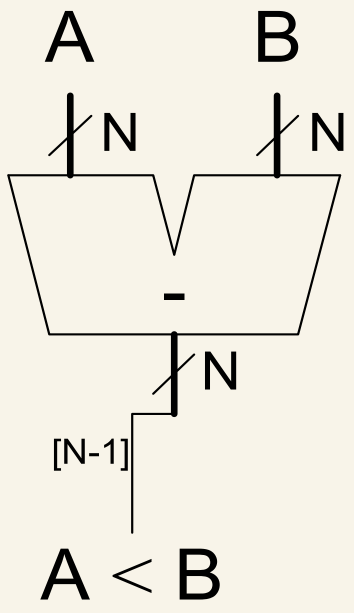
where t_{FA} is the delay of a 1-bit full adder

- Subtraction

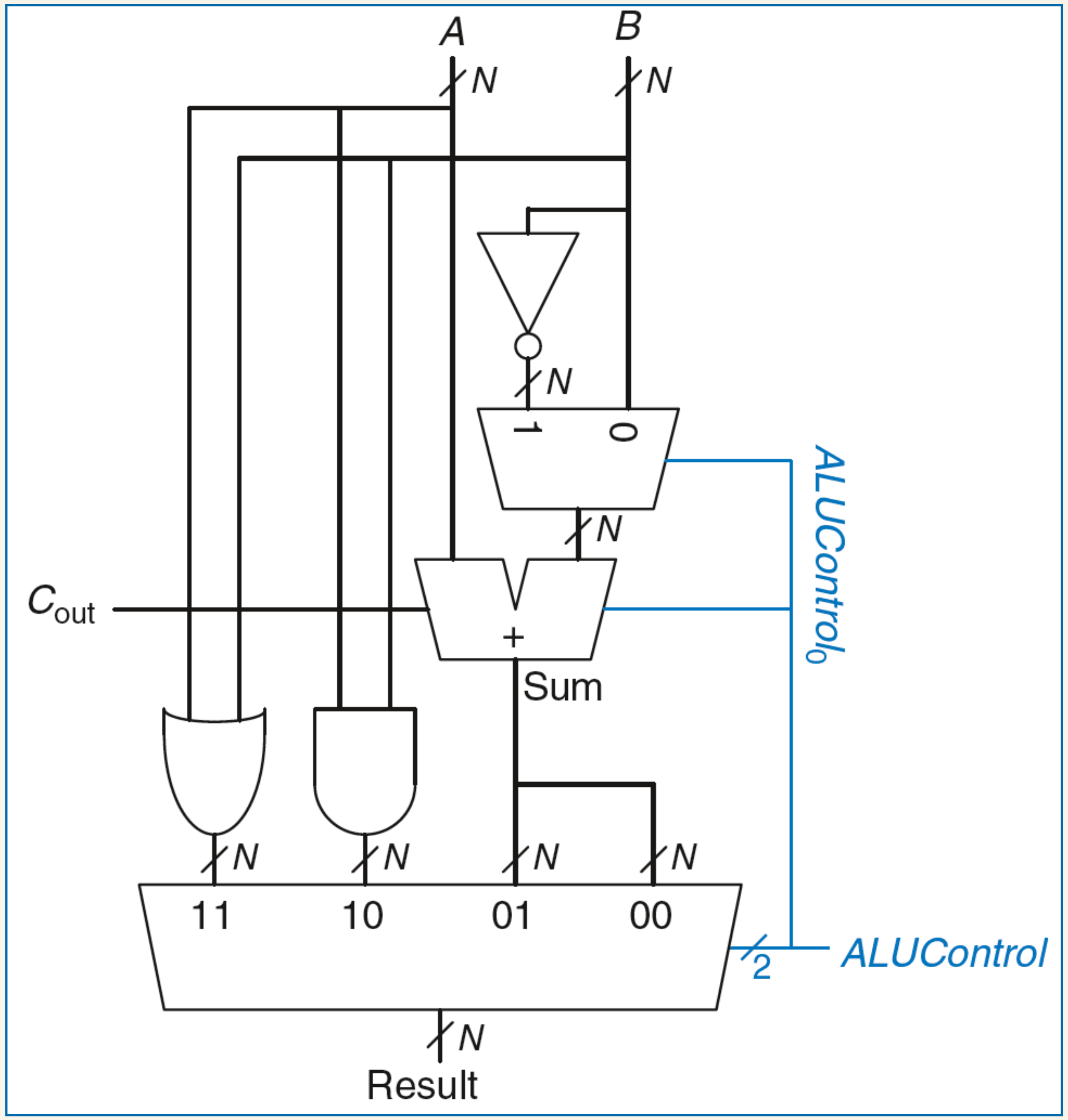


- Comparators



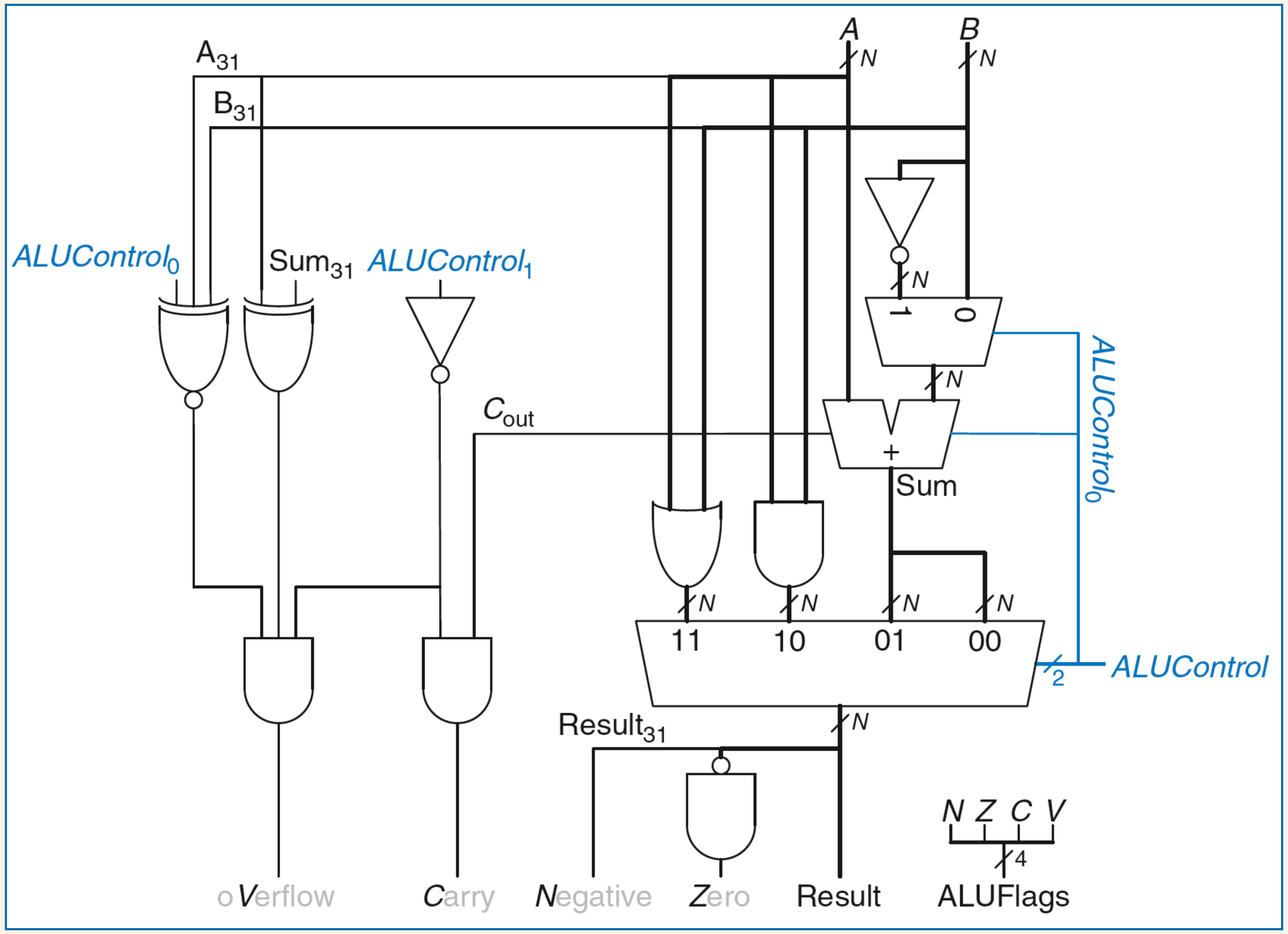


- ALU (Arithmetic Logic Unit)

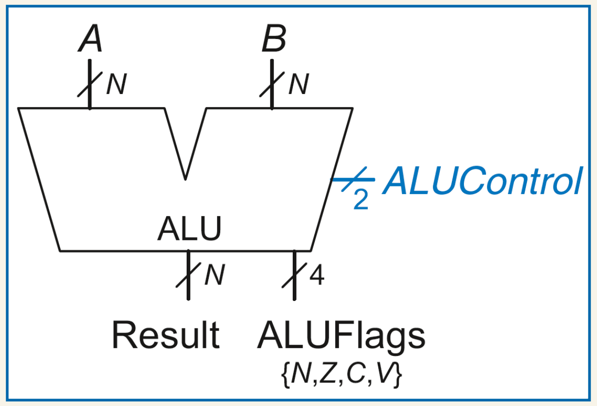


$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR

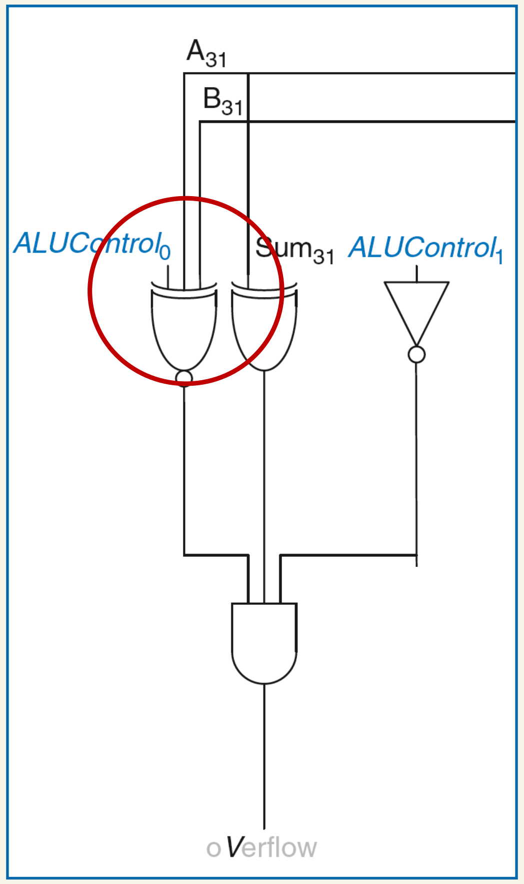
- ALU with Status Flags



Flag	Description
N	*Result* is **N**egative
Z	*Result* is **Z**ero
C	Adder produces **C**arry out
V	Adder o**V**erflowed



- C = 1 if
  - » Cout of Adder is 1 **AND**
  - » ALU is adding or subtracting (ALUControl is 00 or 01
- V = 1 if
  - » ALU is performing addition or subtraction (ALUControl1 = 0) **AND**
  - » A and Sum have opposite signs **AND**
  - » A and B have same signs upon addition (ALUControl0 = 0) **OR**
  - » A and B have different signs upon subtraction (ALUControl0 = 1)

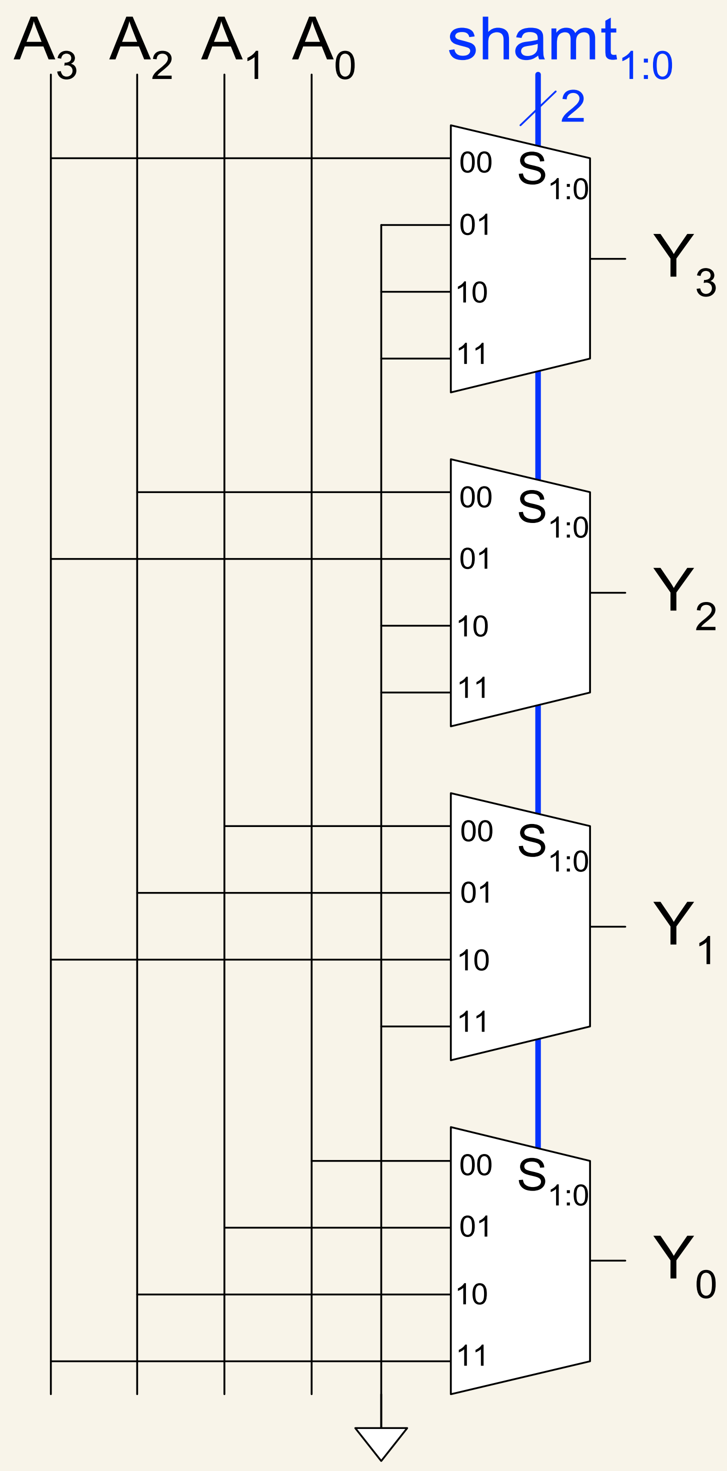


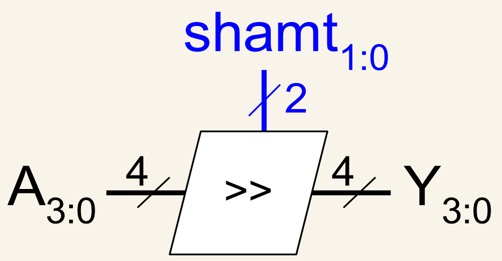
- Shifters and Rotators

  - **Logical shifter:** shifts value to left or right and fills empty spaces with 0's
    - » Ex: 11001 >> 2 = 00110
    - » Ex: 11001 << 2 = 00100

  - **Arithmetic shifter:** same as logical shifter, but on right shift, fills empty spaces with the old most significant bit (msb).
    - » Ex: 11001 >>> 2 = **11**110
    - » Ex: 11001 <<< 2 = 00100

  - **Rotator:** rotates bits in a circle, such that bits shifted off one end are shifted into the other end
    - » Ex: 11001 ROR 2 = 01110
    - » Ex: 11001 ROL 2 = 00111

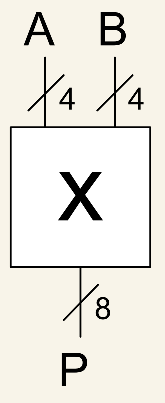




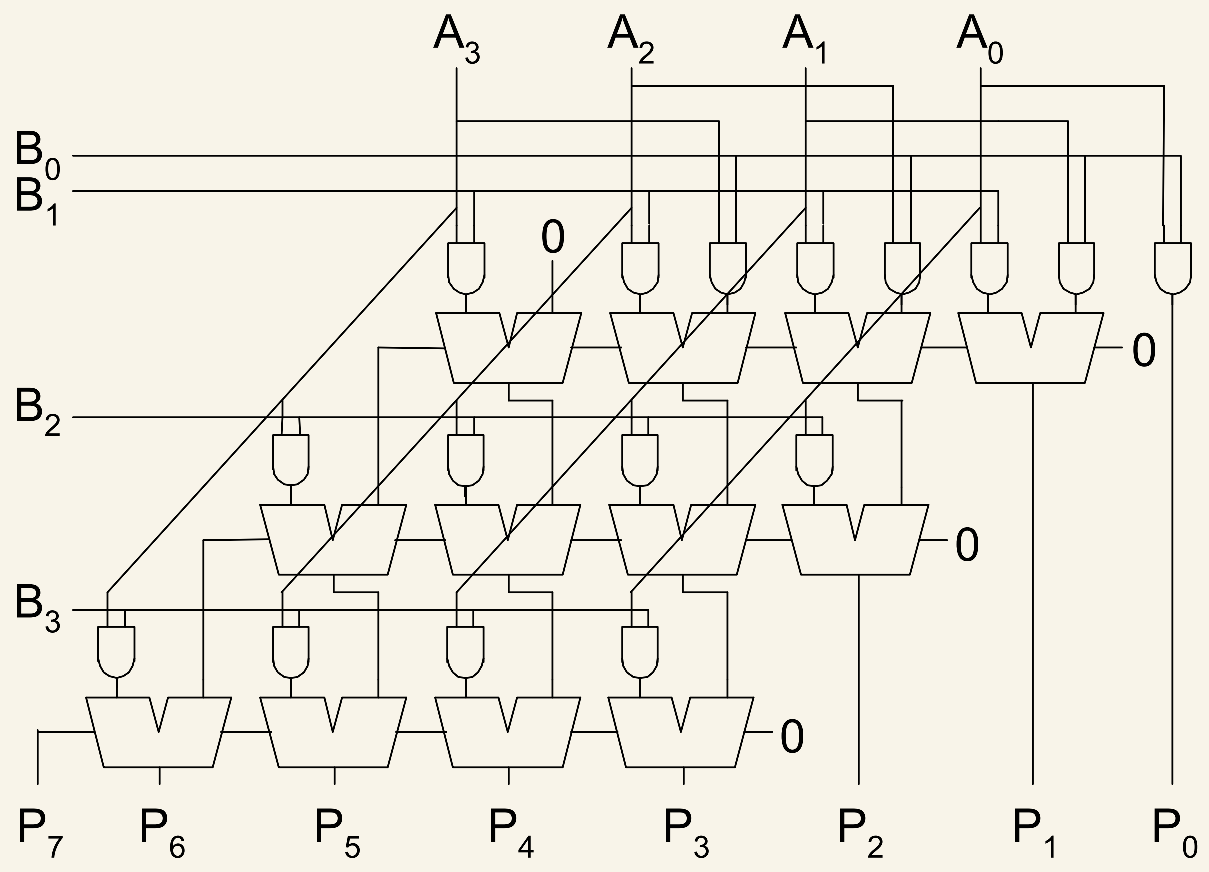
- Shifters as Multipliers, Dividers
  - $A << N = A \times 2^N$
    - » Example : 00001 << 2 = 00100 ($1 \times 2^2 = 4$)
    - » Example : 11101 << 2 = 10100 ($-3 \times 2^2 = -12$)
  - $A >>> N = A \div 2^N$
    - » Example : 01000 >>> 2 = 00010 ($8 \div 2^2 = 2$)
    - » Example : 10000 >>> 2 = 11100 ($-16 \div 2^2 = -4$)

- Multiplication
  - Partial products are formed by multiplying a single digit of the multiplier with the entire multiplicand
  - Shifted partial products are summed to form the result

Decimal		Binary
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial	0101
+ 920	products	0101
9660		0101
		+ 0000
	result	0100011
230 x 42 = 9660		5 x 7 = 35



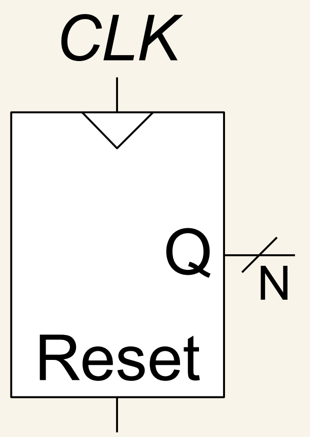
$$
\begin{array}{rcccccccc}
 & & & & & A_3 & A_2 & A_1 & A_0 \\
 & & & & \times & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & & & & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 \\
 & & & & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & \\
 & & & A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & & \\
+ & & A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 & & & \\
\hline
 & P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
$$

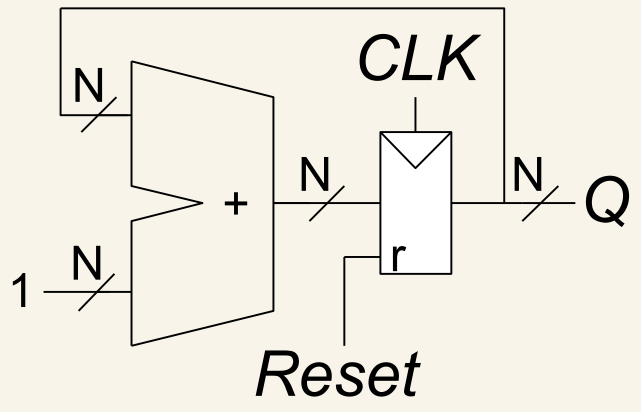


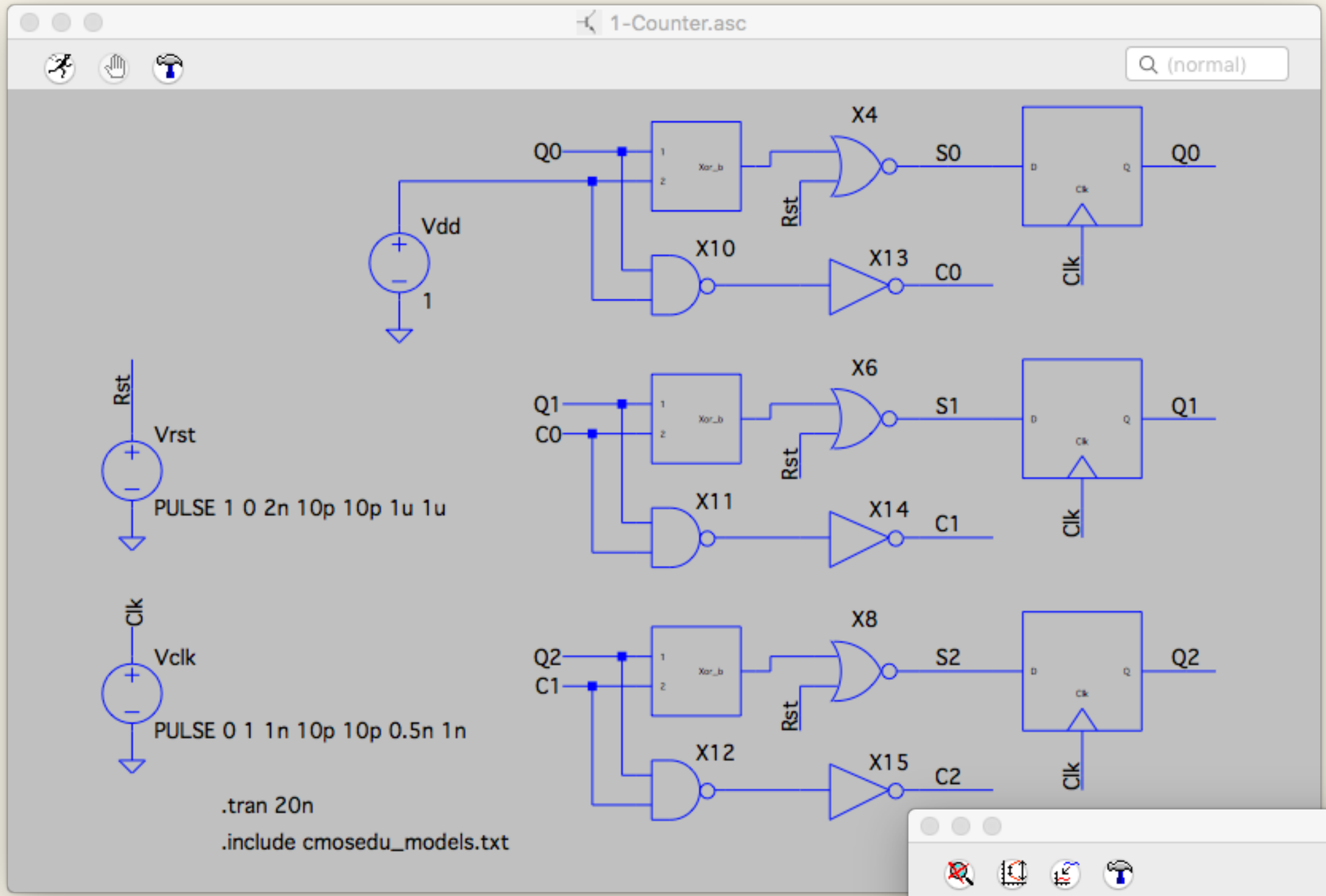
Sequential Building Blocks

- Counters
  - Increments on each clock edge
  - Used to cycle through numbers.
    - » For example
      - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001

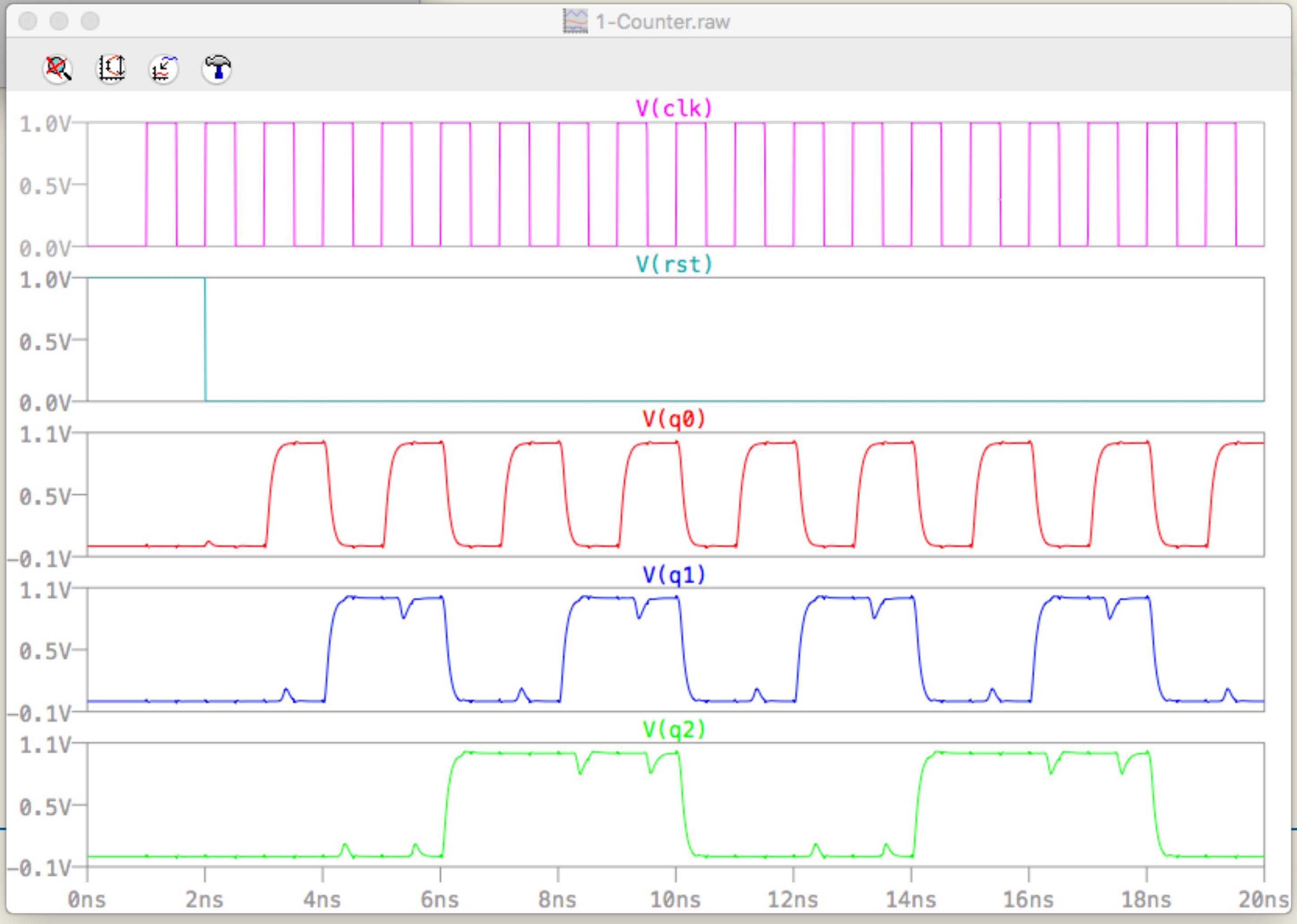
Symbol **Implementation**





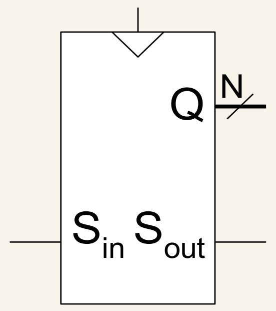


```
PULSE 1 0 2n 10p 10p 1u 1u
PULSE 0 1 1n 10p 10p 0.5n 1n
.tran 20n
.include cmosedu_models.txt
```

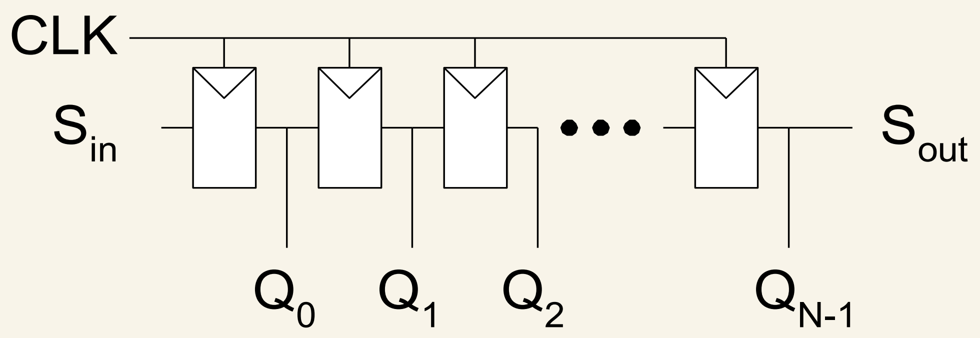


- Shift Registers
  - » Shift a new bit in on each clock edge
  - » Shift a bit out on each clock edge
  - » Serial-to-parallel converter: converts serial input (S_{in}) to parallel output ($Q_{0:N-1}$)

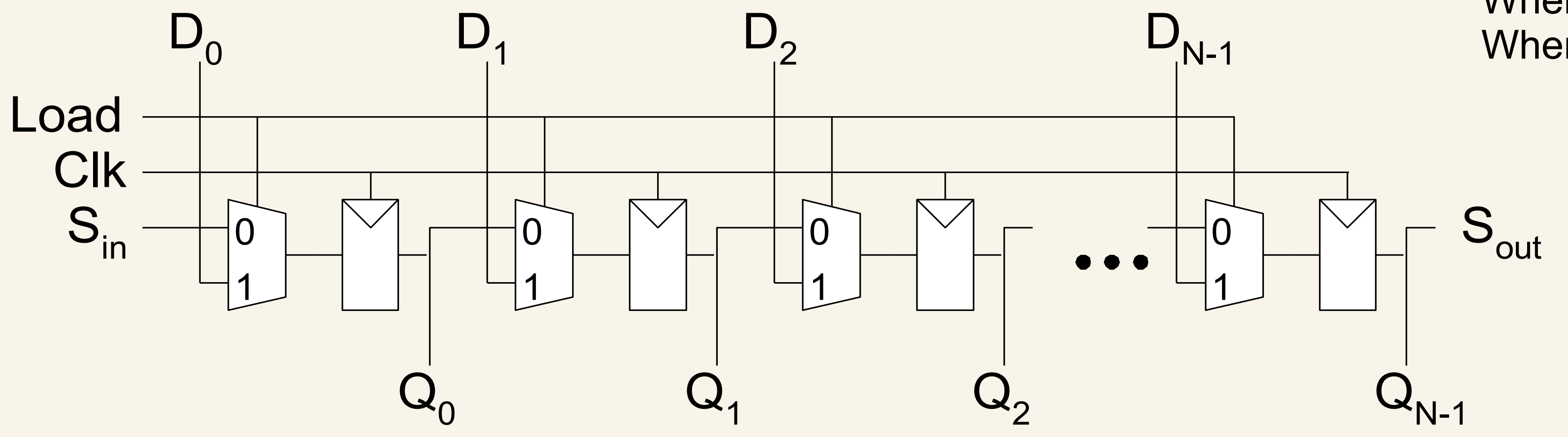
Symbol:



Implementation:

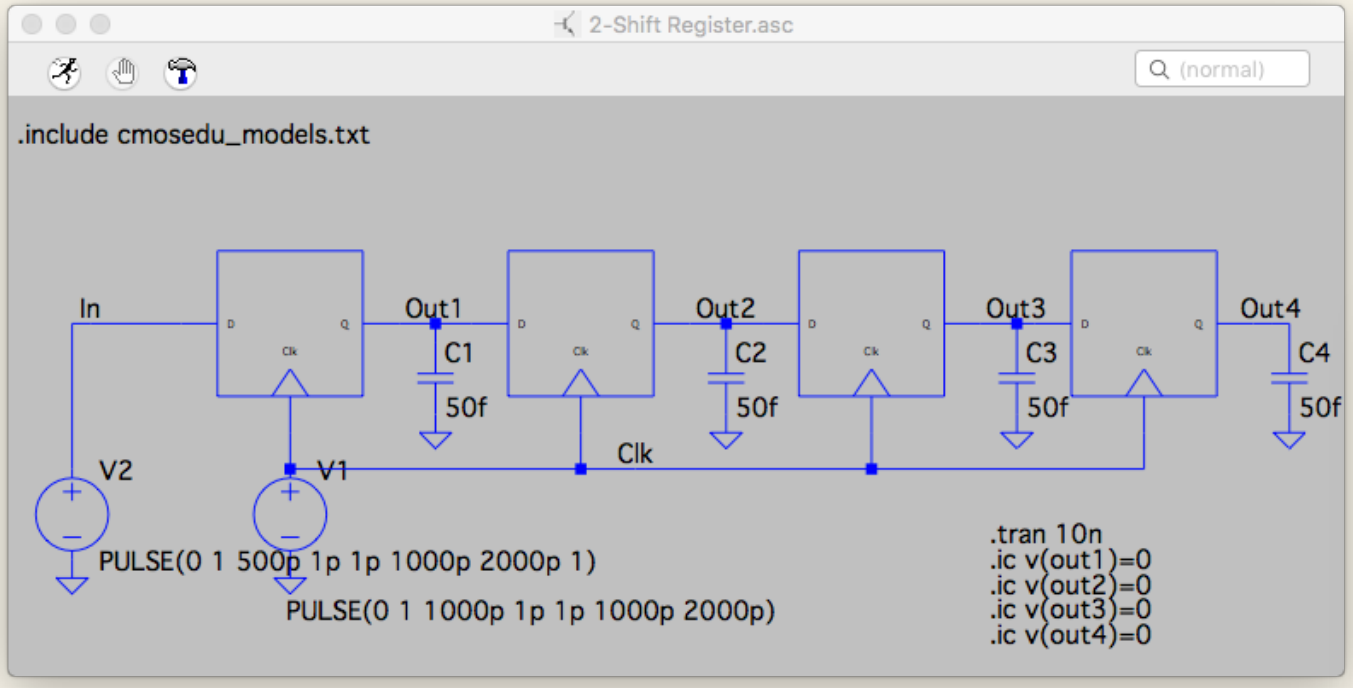


- Shift Register with Parallel Load



When Load = 1, acts as a normal N-bit register
When Load = 0, acts as a shift register

Application : Scannable Flip-Flop

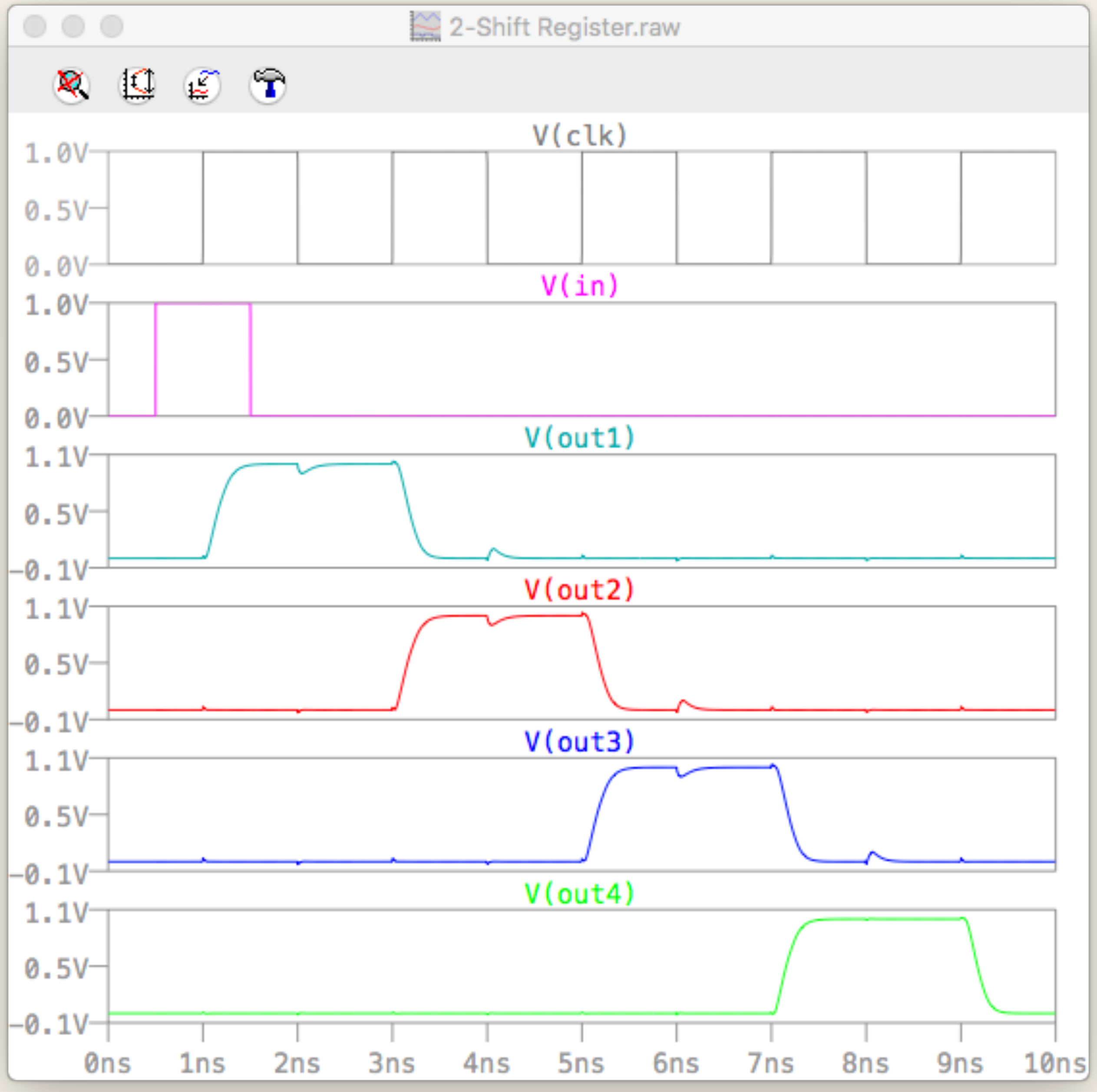


.include cmosedu_models.txt

PULSE(0 1 500p 1p 1p 1000p 2000p 1)

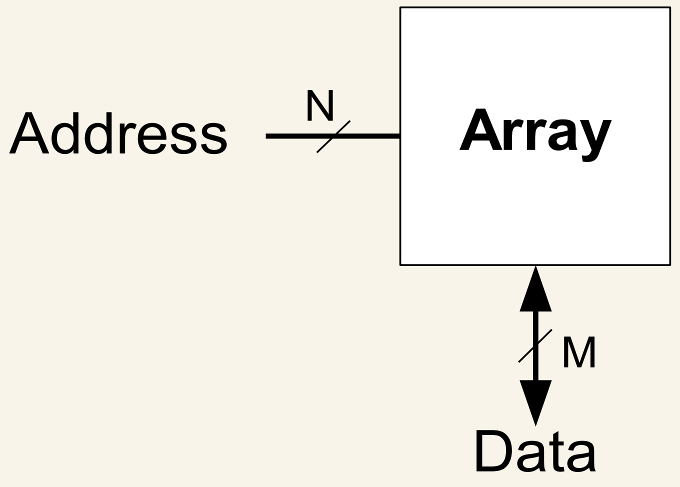
PULSE(0 1 1000p 1p 1p 1000p 2000p)

```
.tran 10n
.ic v(out1)=0
.ic v(out2)=0
.ic v(out3)=0
.ic v(out4)=0
```

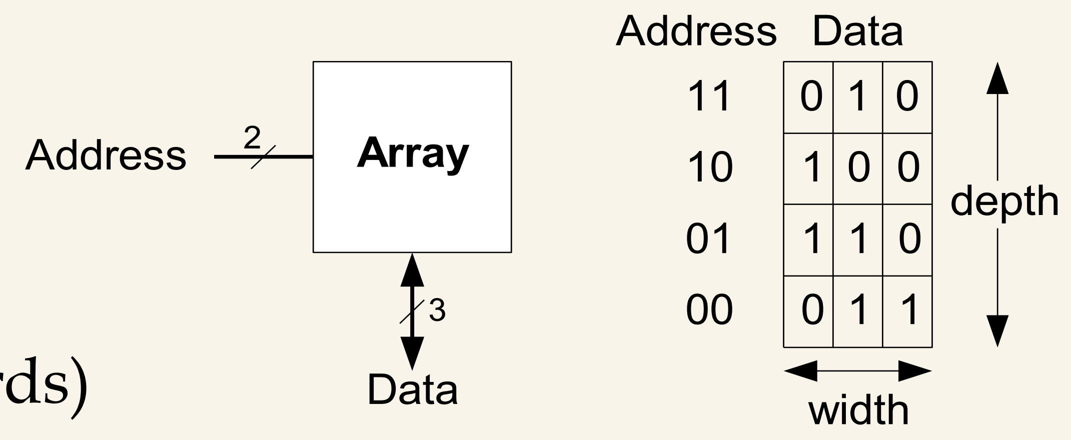


Memory Arrays

- Efficiently store large amounts of data
- 3 common types:
  - Dynamic random access memory (DRAM)
  - Static random access memory (SRAM)
  - Read only memory (ROM)
- M-bit data value read/written at each unique N-bit address

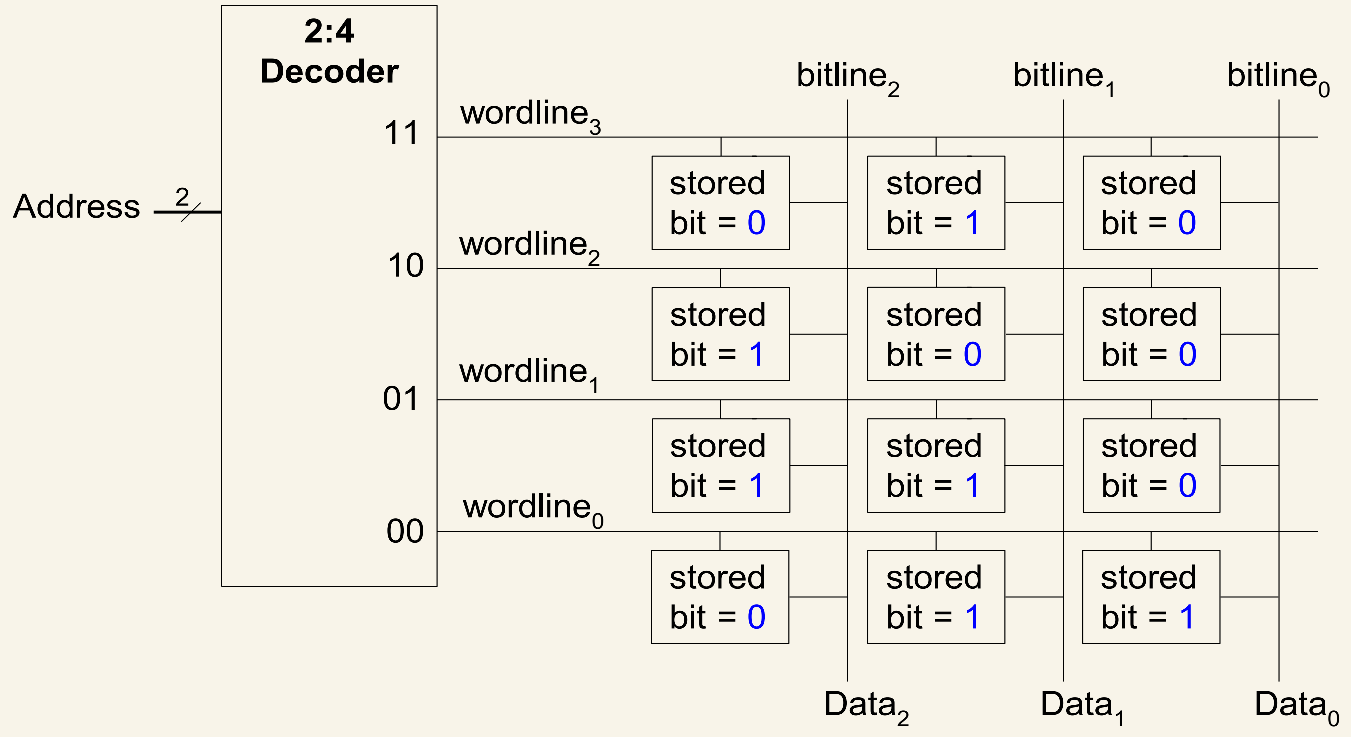


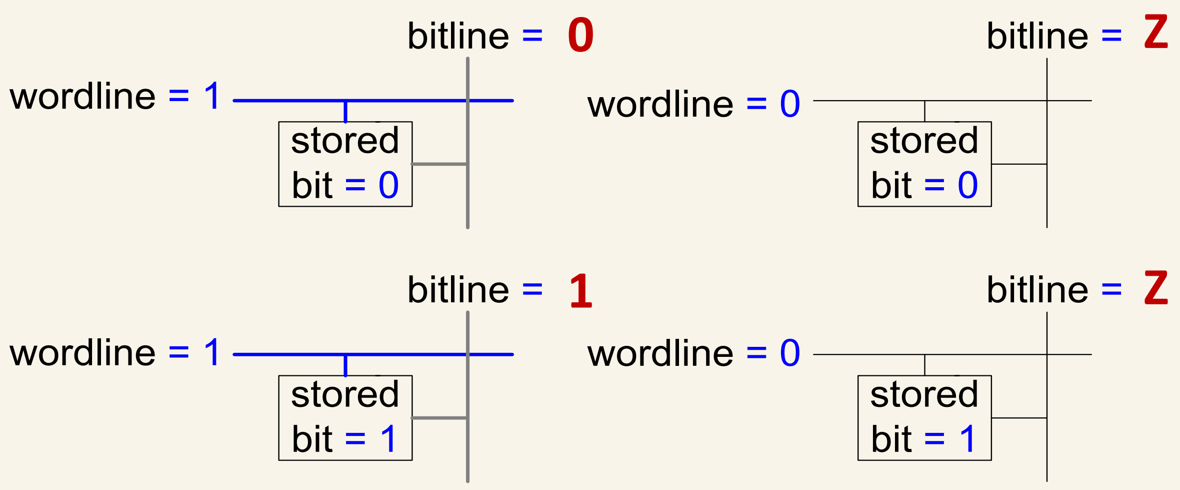
- 2-dimensional array of bit cells
- Each bit cell stores one bit
- N address bits and M data bits:
  - » 2^N rows and M columns
  - » Depth : number of rows (number of words)
  - » Width : number of columns (size of word)
  - » Array size : depth × width = $2^N \times M$



Address	Data
11	0 1 0
10	1 0 0
01	1 1 0
00	0 1 1

- Wordline :
  - » like an enable
  - » single row in memory array read/written
  - » corresponds to unique address
  - » only one wordline HIGH at once





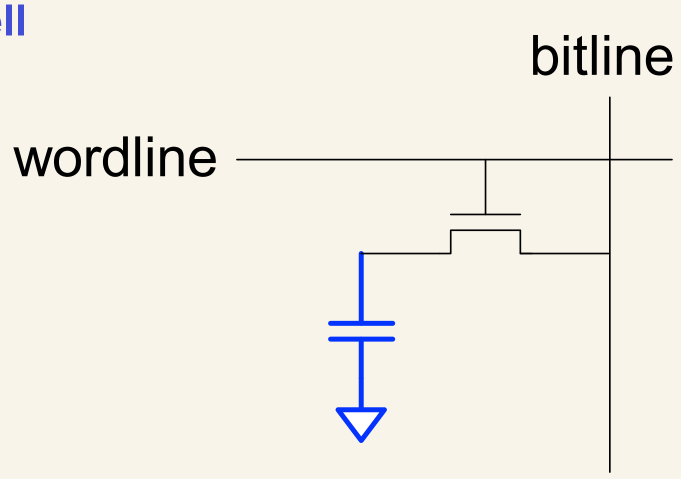
(a) (b)

RAM: Random Access Memory

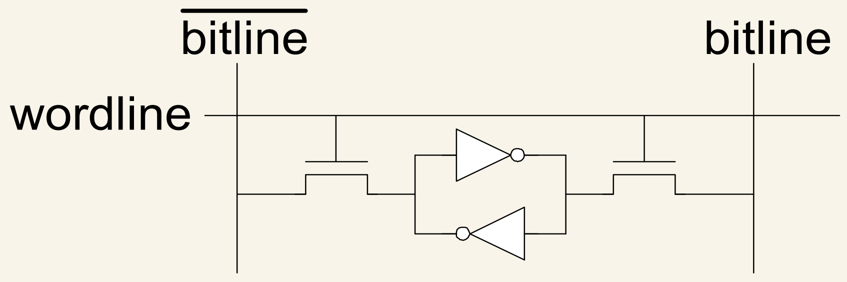
- Volatile: loses its data when power off
- Read and written quickly
- 2 types
  - DRAM (Dynamic random access memory)
    - » uses a capacitor
    - » Main memory in your computer is DRAM
  - SRAM (Static random access memory)
    - » uses cross-coupled inverters

Historically called random access memory because any data word accessed as easily as any other (in contrast to sequential access memories such as a tape recorder)

DRAM bit cell



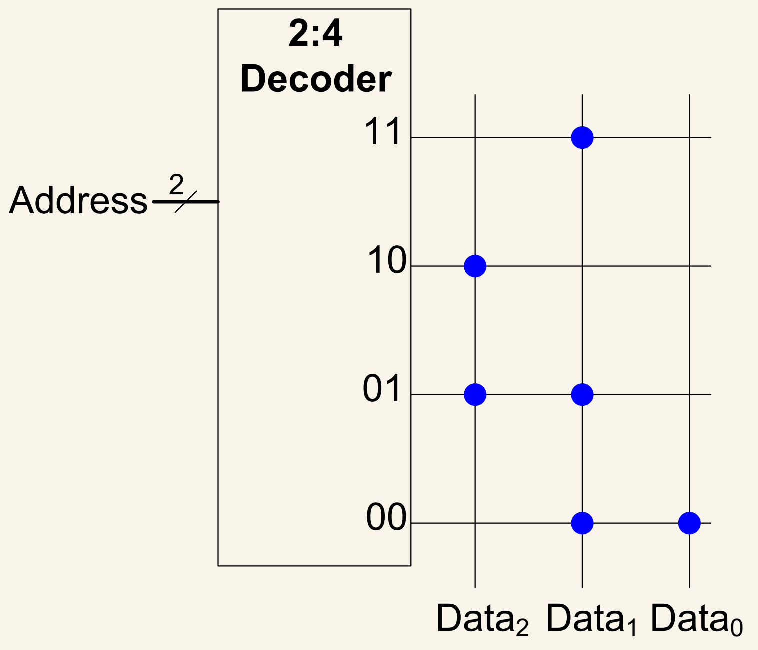
SRAM bit cell

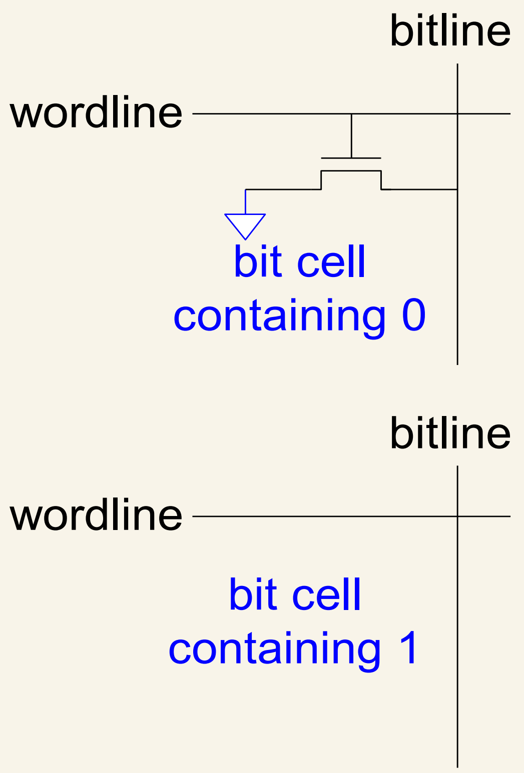


ROM: Read Only Memory

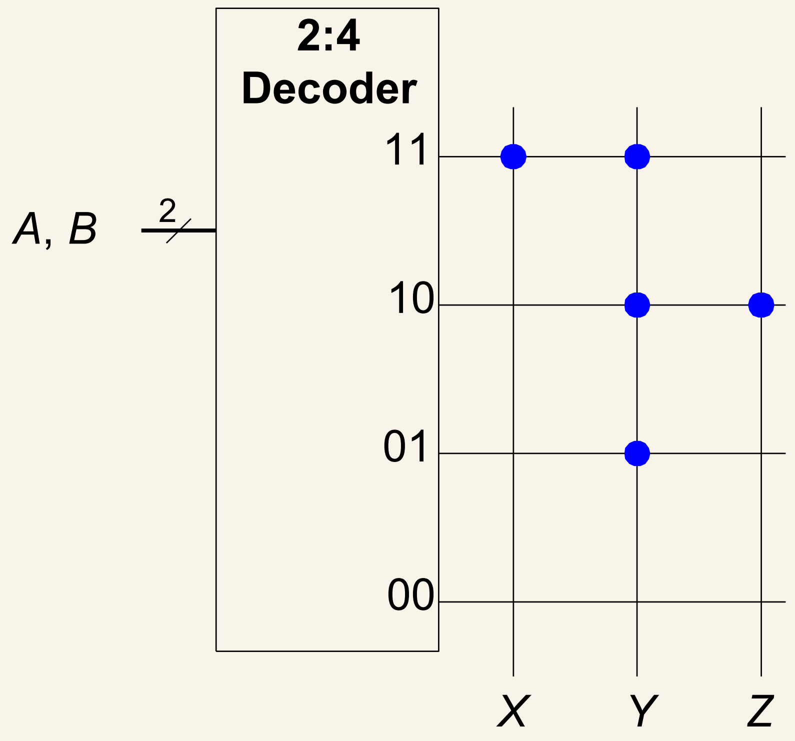
- Nonvolatile : retains data when power off
- Read quickly, but writing is impossible or slow
- Flash memory in cameras, thumb drives, and digital cameras are all ROMs

Historically called read only memory because ROMs were written at manufacturing time or by burning fuses. Once ROM was configured, it could not be written again. This is no longer the case for Flash memory and other types of ROMs.





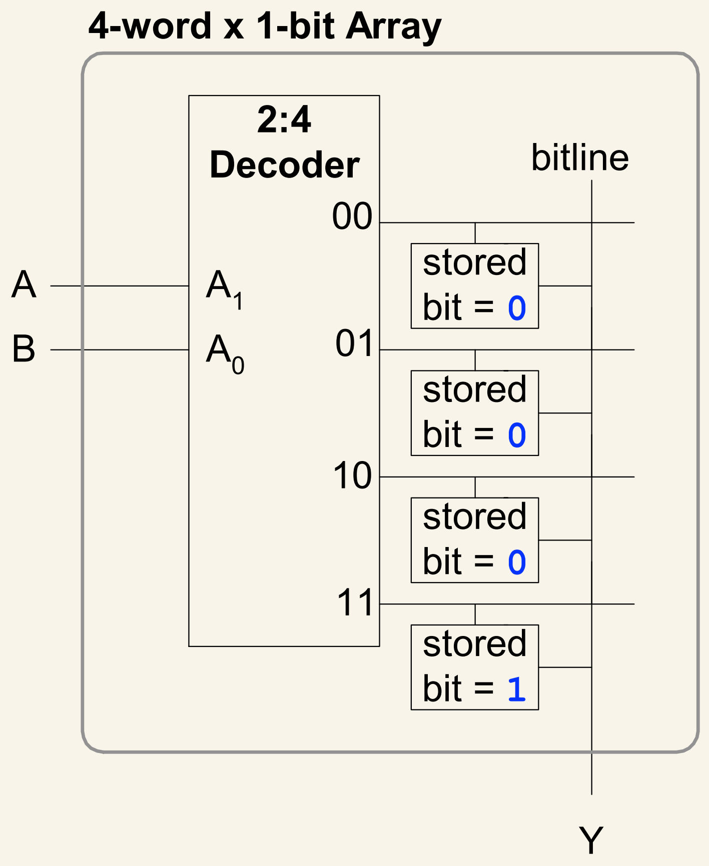
- Logic Using Memory Arrays
  - Implement the following logic functions using a $2^2 \times 3$-bit ROM
    - » $X = AB$
    - » $Y = A + B$
    - » $Z = A\overline{B}$



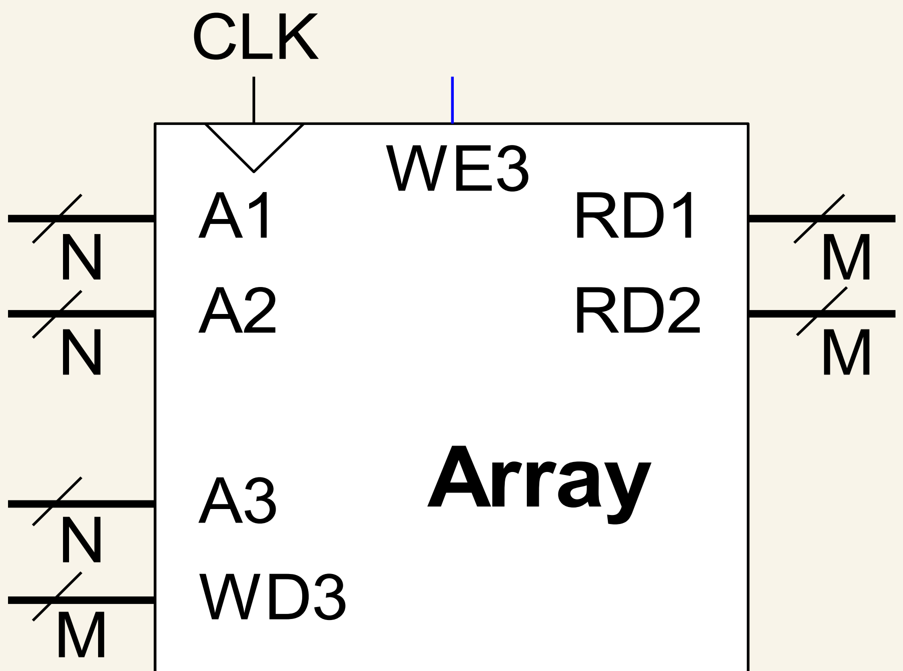
Called *lookup tables* (LUTs): look up output at each input combination (address)

Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



- Multi-ported Memories
  - Port : address/data pair
  - 3-ported memory
    - » 2 read ports (A1/RD1, A2/RD2)
    - » 1 write port (A3/WD3, WE3 enables writing)
  - Register file : small multi-ported memory


CLK
WE3
A1
A2
A3
WD3
RD1
RD2
Array
N
N
N
M
M
M