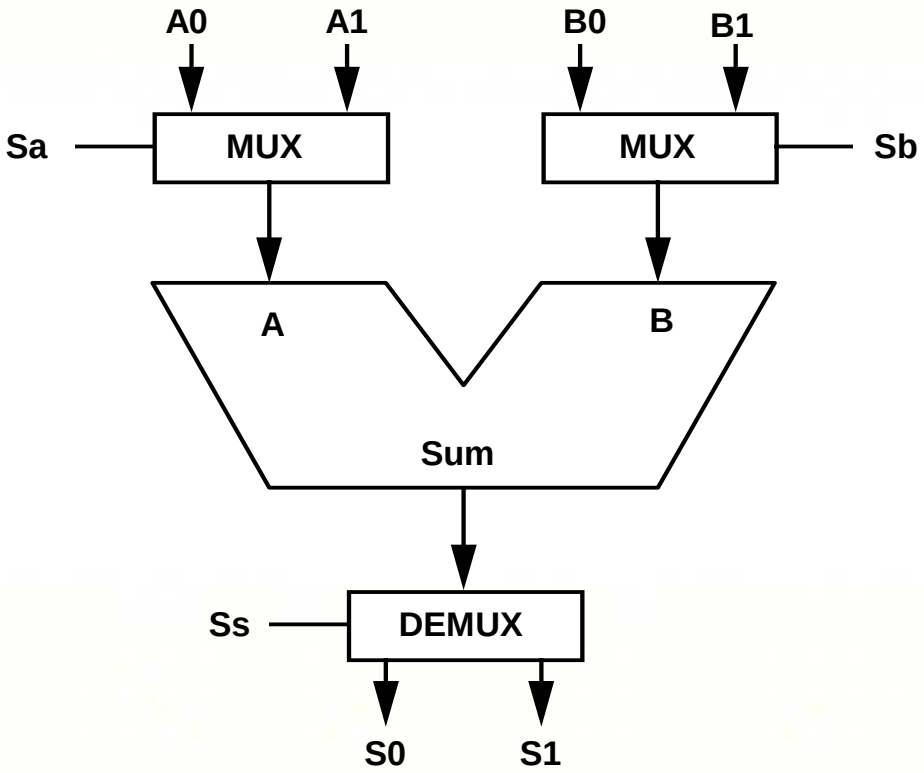
SEE 3243/4243
Digital System

Lecturers :
Muhammad Mun'im Ahmad Zabidi
Muhammad Nadzir Marsono
Kamal Khalil

Week 3: Logic Design Using MSI Components & Memory

Multiplexers/Selectors



Multiple input sources

Multiple output destinations

Multiplexers/Selectors: General Concept

- 2^n data inputs, n control inputs, 1 output
- used to connect 2^n points to a single point
- control signal pattern form binary index of input connected to output

$$Z = A' I_0 + A I_1$$

Functional form

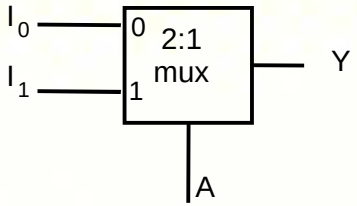
A	Z
0	I_0
1	I_1

Logical form

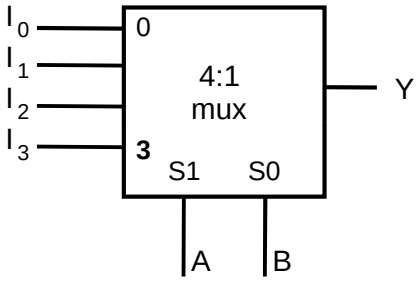
I_1	I_0	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Two alternative forms for a 2:1 Mux Truth Table

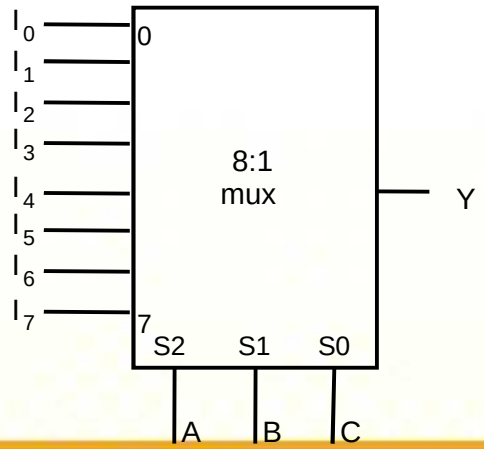
Multiplexers/Selectors



$$Y = A' I_0 + A I_1$$

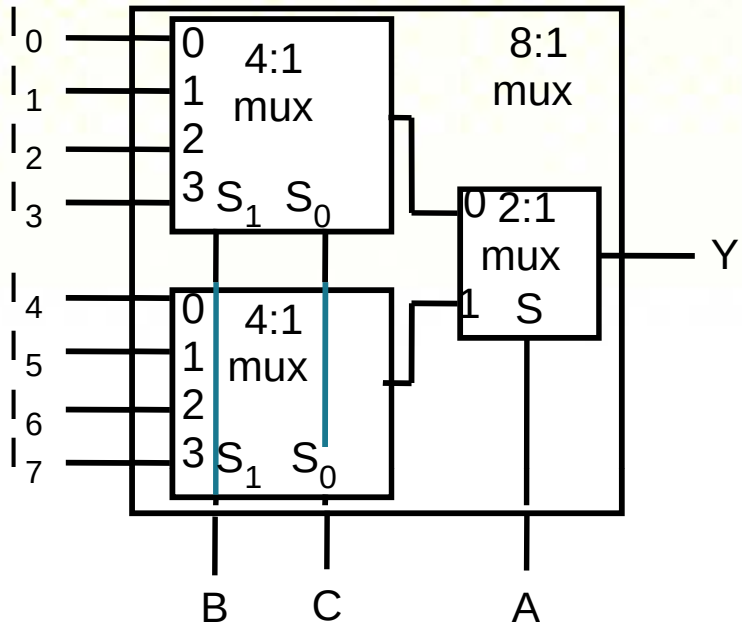


$$Y = A' B' I_0 + A' B I_1 + A B' I_2 + A B I_3$$



$$Y = A' B' C' I_0 + A' B' C I_1 + A' B C' I_2 + A' B C I_3 + A B' C' I_4 + A B' C I_5 + A B C' I_6 + A B C I_7$$

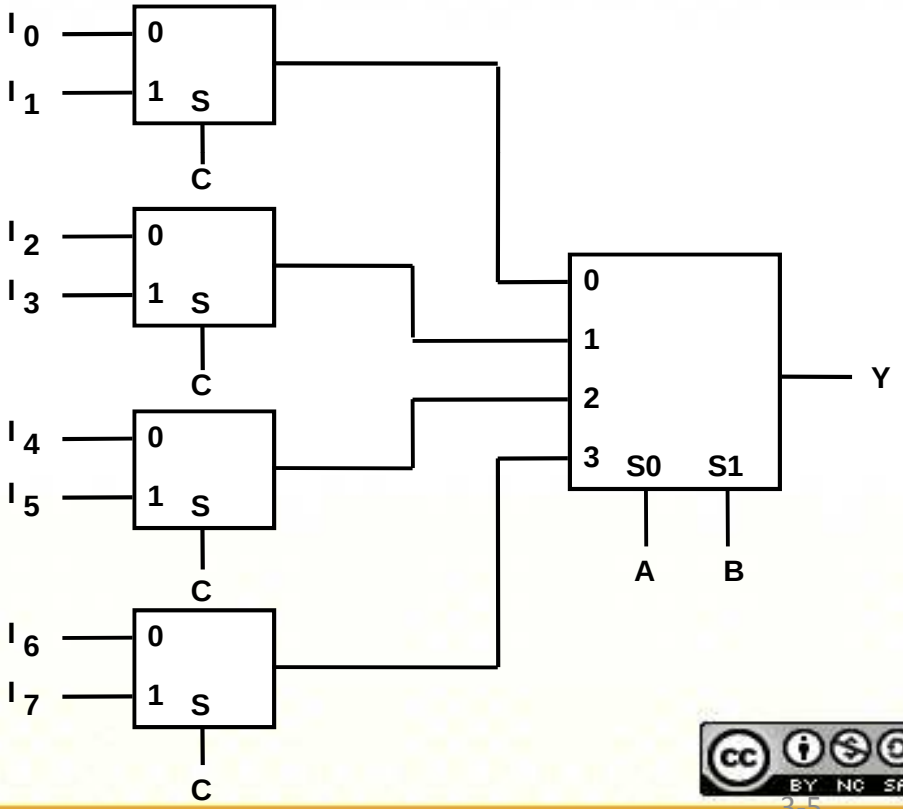
Multiplexer/Selector: Expansion



Alternative 8:1 Mux Implementation

Control signals B and C simultaneously choose one of I_0-I_3 and I_4-I_7

Control signal A chooses which of the upper or lower MUX's output to gate to Y



Multiplexer/Selector

2^{n-1} :1 multiplexer can implement any function of n variables

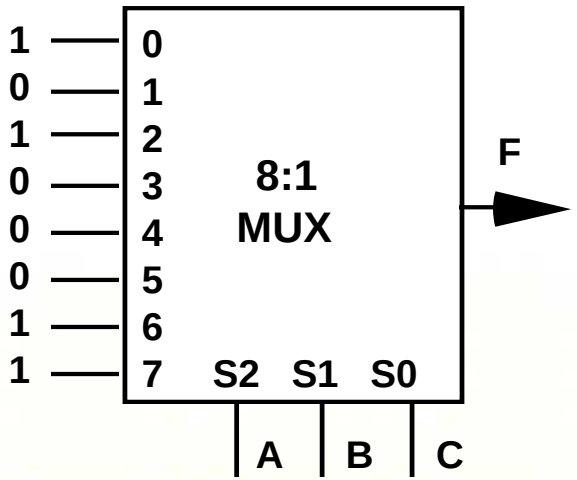
n-1 control variables; remaining variable is a data input to the mux

Example:

$F(A,B,C) = m_0 + m_2 + m_6 + m_7$

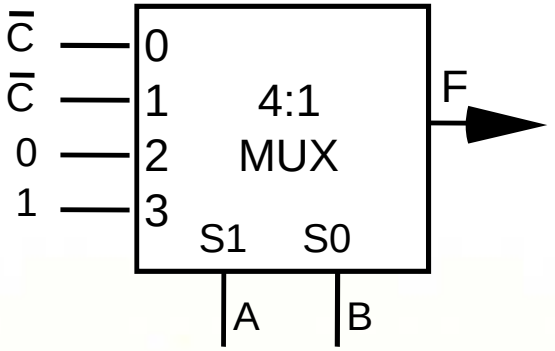
$= A' B' C' + A' B C' + A B C' + A B C$

$= A' B' (C') + A' B (C') + A B' (0) + A B (1)$



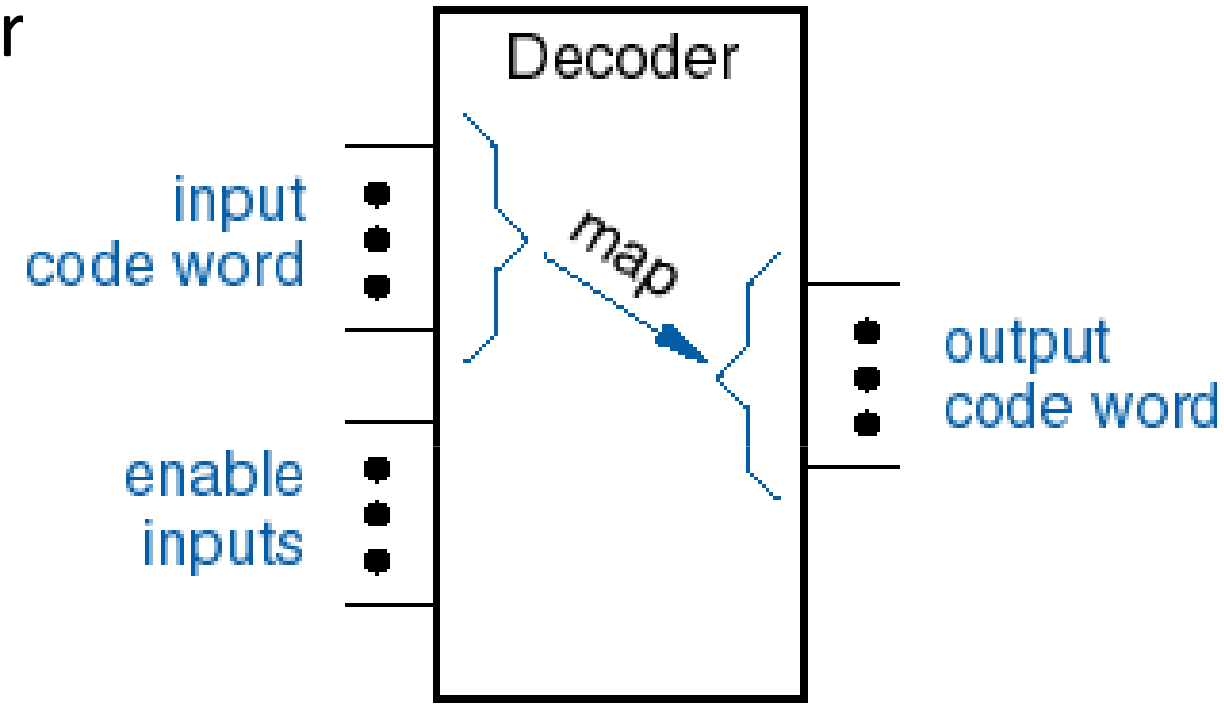
"Lookup Table"

A	B	C	F	
0	0	0	1	$\overline{C}$
0	0	1	0	
0	1	0	1	$\overline{C}$
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



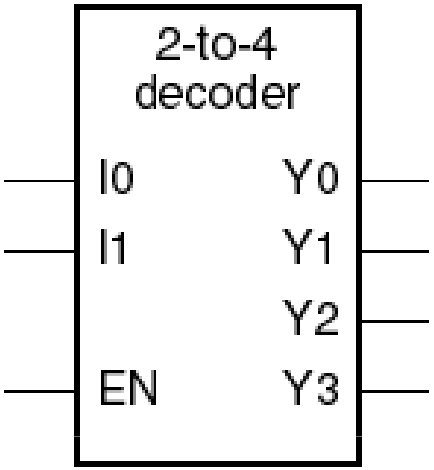
Decoders

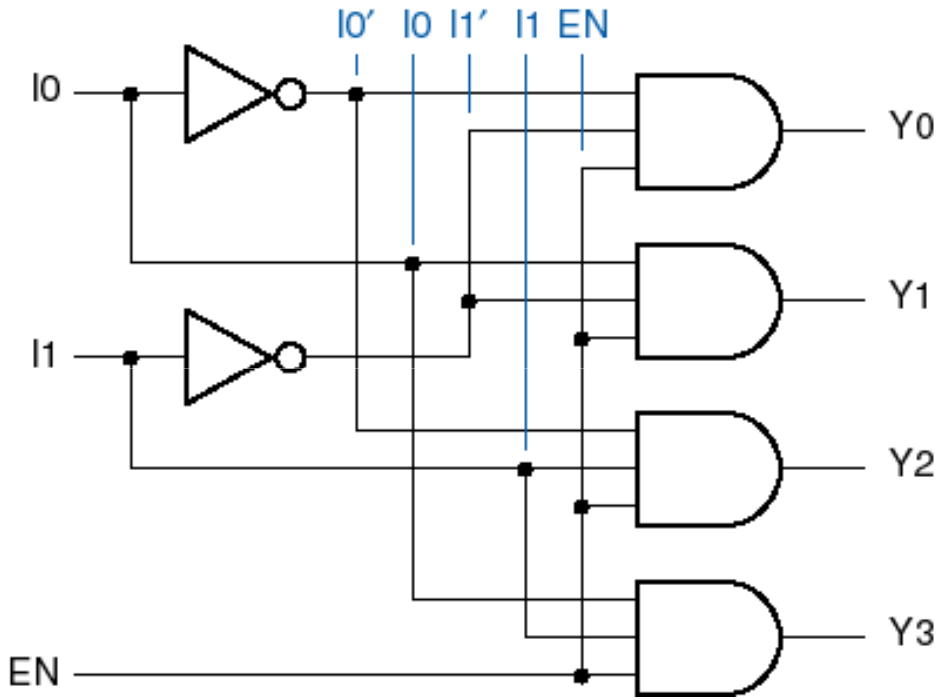
- General decoder



- Typically *n* inputs, 2^n outputs
  - 2-to-4, 3-to-8, 4-to-16, etc
- Control inputs (called select S) represent Binary index of output to which the input is connected
- Data input usually called "enable" (G)

Binary 2-to-4 decoder

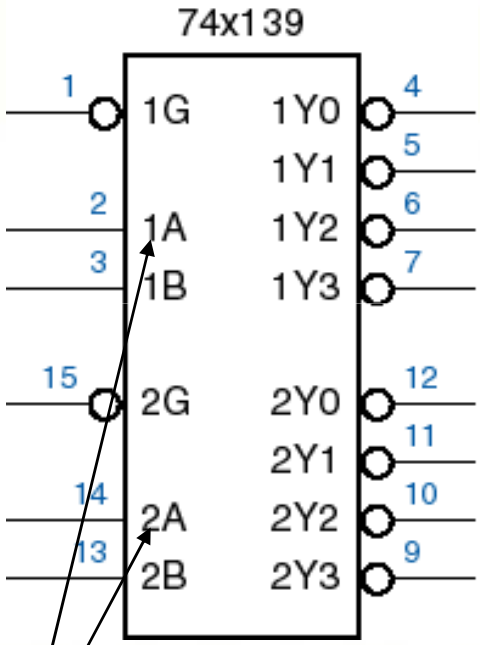




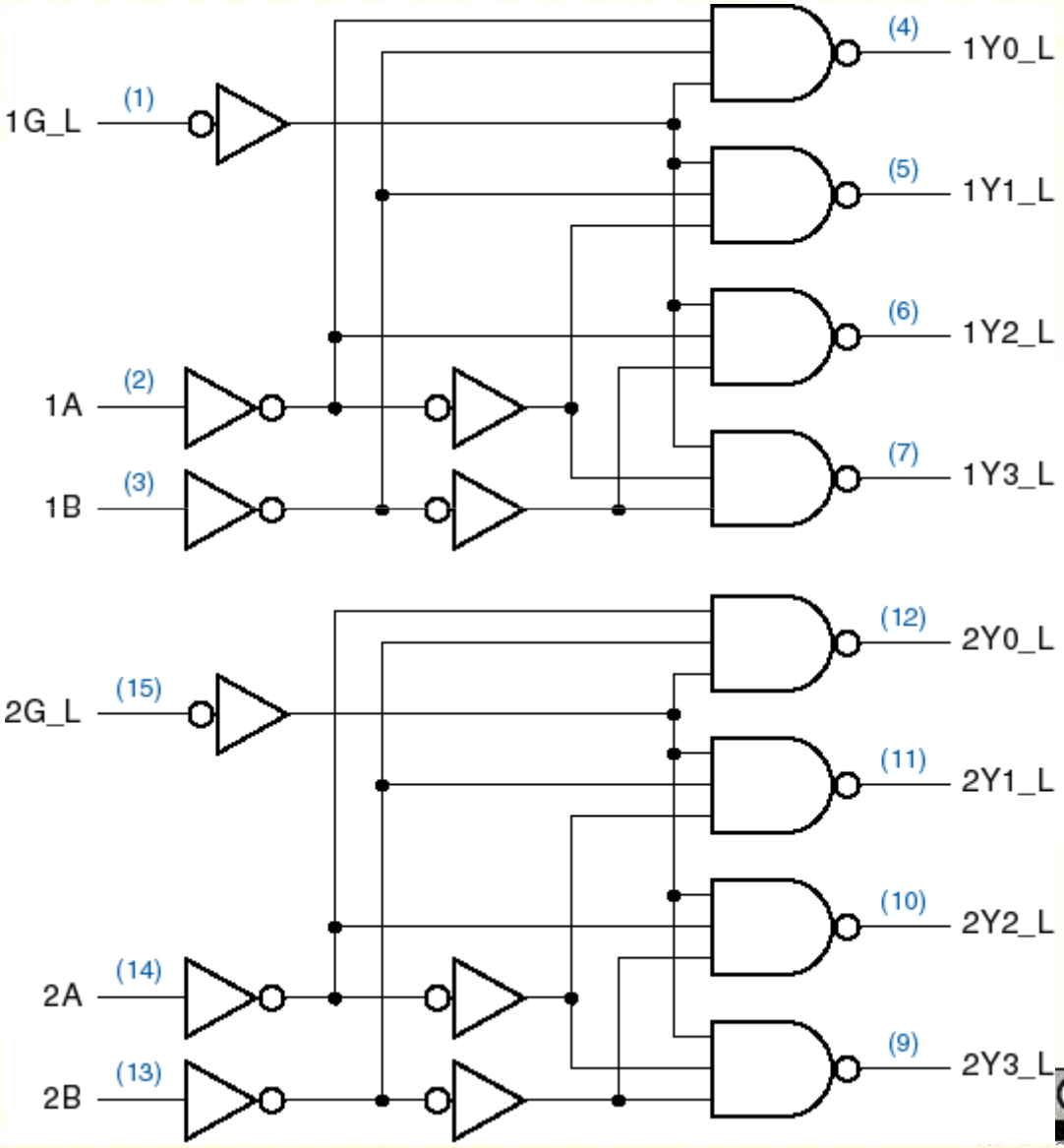
Inputs			*Outputs*			
EN	I1	I0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Note “x” (don’t care) notation.

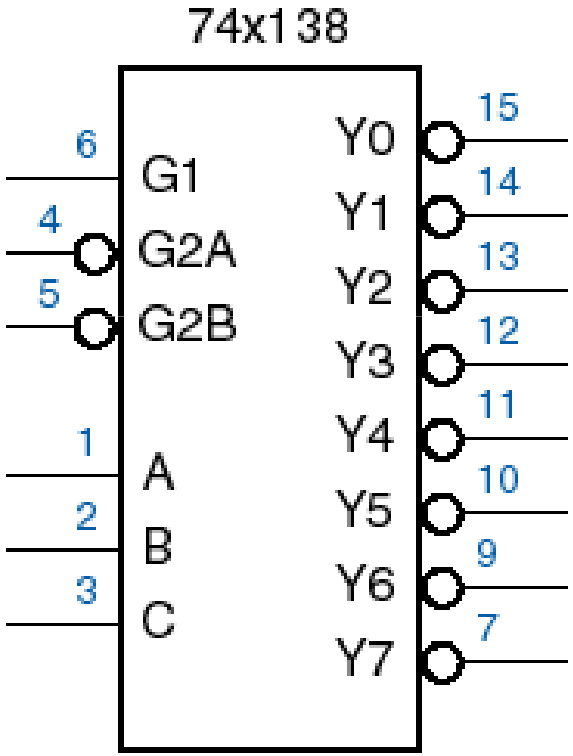
Complete 74x139 Decoder



A is LSB selector

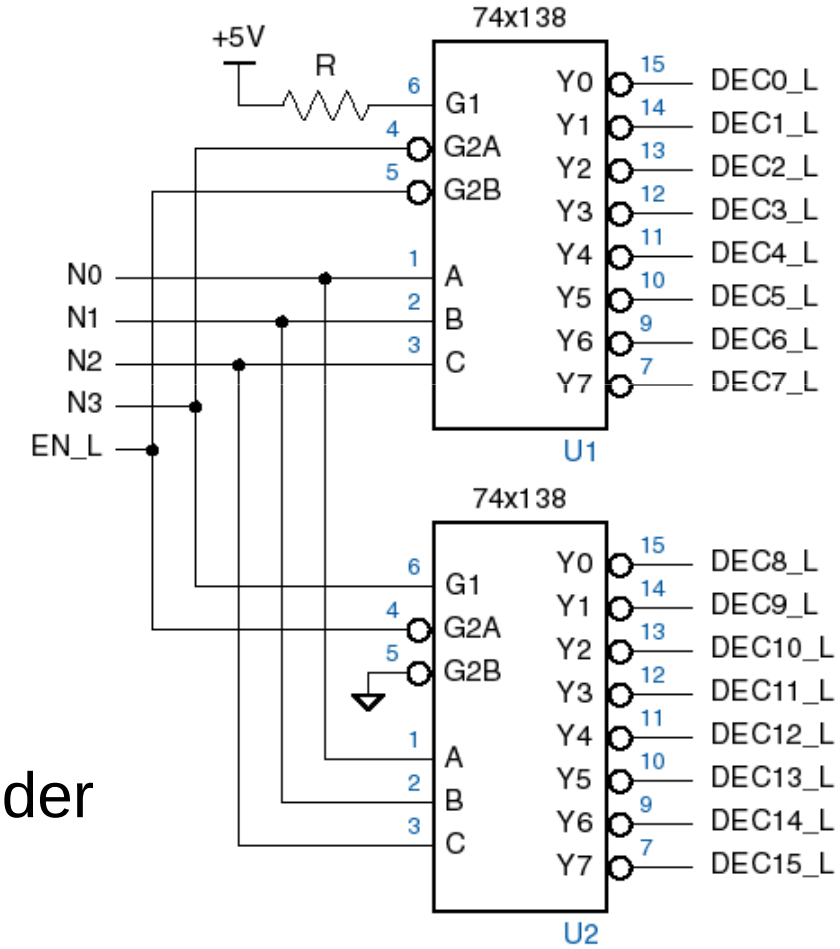


74x138 3-to-8-decoder symbol



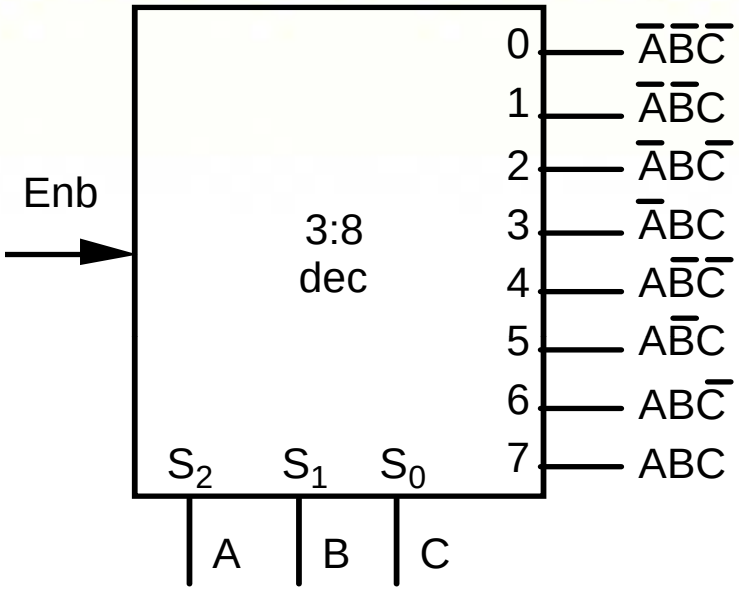
A is LSB selector

Decoder cascading



4-to-16 decoder

Decoder/Demultiplexer: Logic Building Block

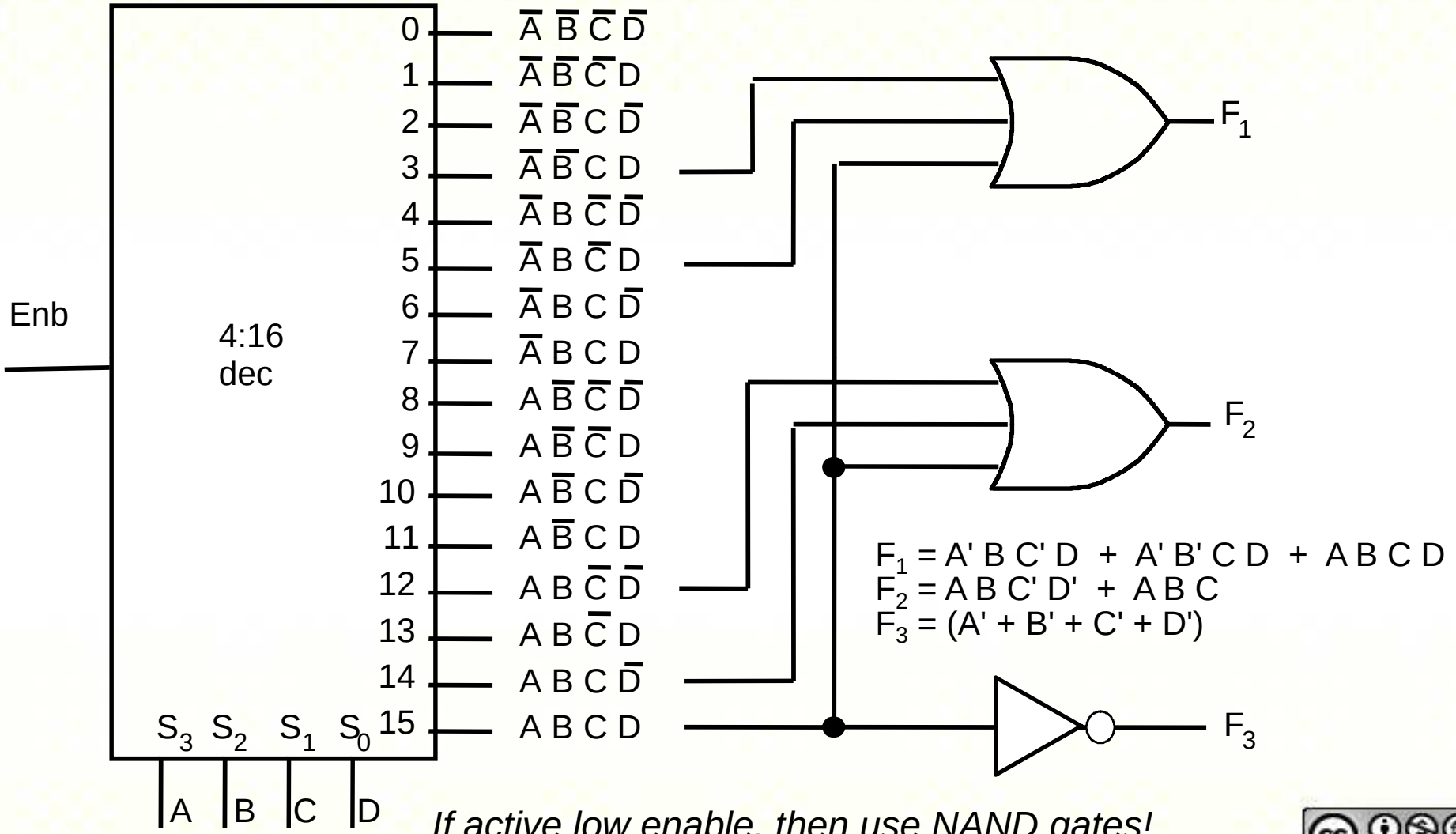


Decoder Generates Appropriate Minterm based on Control Signals

Example Function:

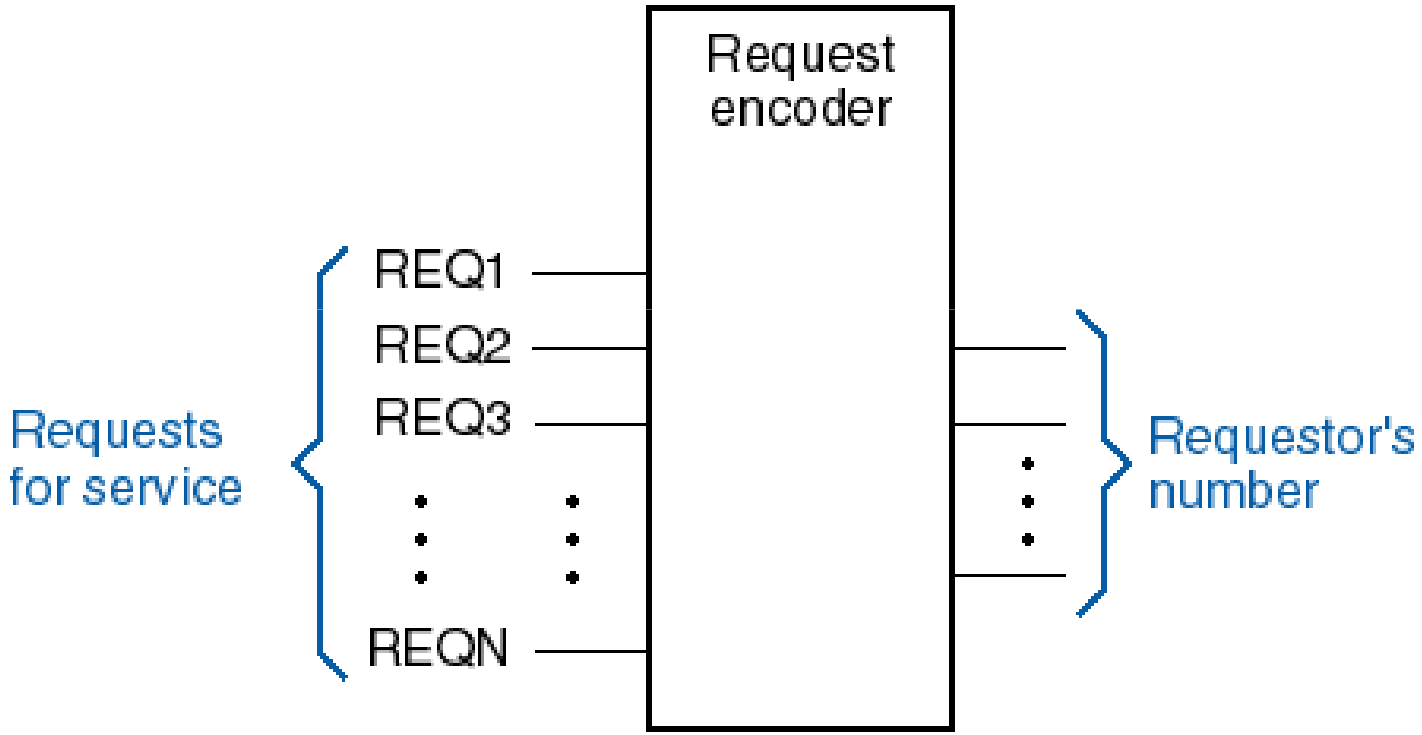
F1 = A' B C' D + A' B' C D + A B C D
F2 = A B C' D' + A B C
F3 = (A' + B' + C' + D')

Decoder/Demultiplexer:

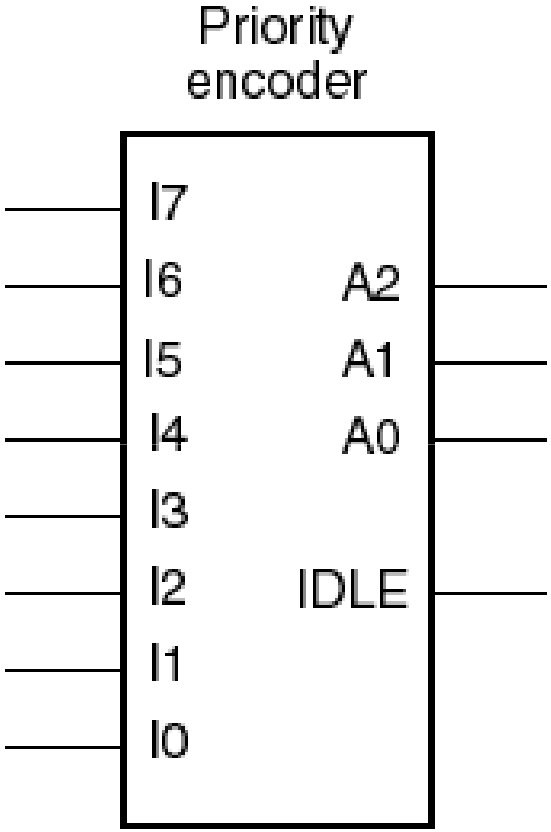


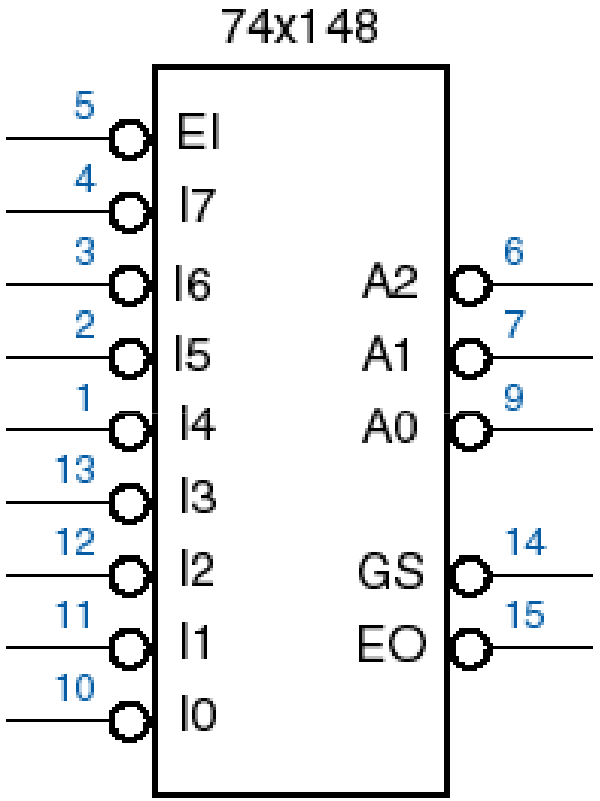
If active low enable, then use NAND gates!

Need priority in most applications



74x148 8-input priority encoder





- Active-low I/O
- Enable Input
- "Got Something"
- Enable Output

Priority-encoder logic equations

$$H7 = I7$$

$$H6 = I6 \cdot I7'$$

$$H5 = I5 \cdot I6' \cdot I7'$$

$$\cdots$$

$$H0 = I0 \cdot I1' \cdot I2' \cdot I3' \cdot I4' \cdot I5' \cdot I6' \cdot I7'$$

$$A2 = H4 + H5 + H6 + H7$$

$$A1 = H2 + H3 + H6 + H7$$

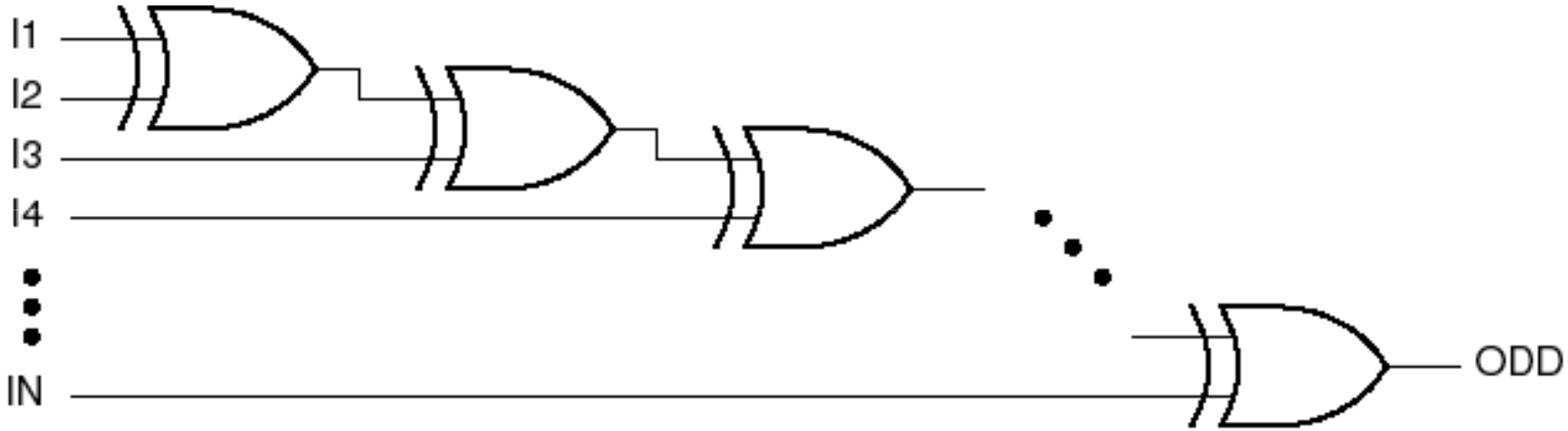
$$A0 = H1 + H3 + H5 + H7$$

$$IDLE = (I0 + I1 + I2 + I3 + I4 + I5 + I6 + I7)'$$

$$= I0' \cdot I1' \cdot I2' \cdot I3' \cdot I4' \cdot I5' \cdot I6' \cdot I7'$$

Multi-input XOR

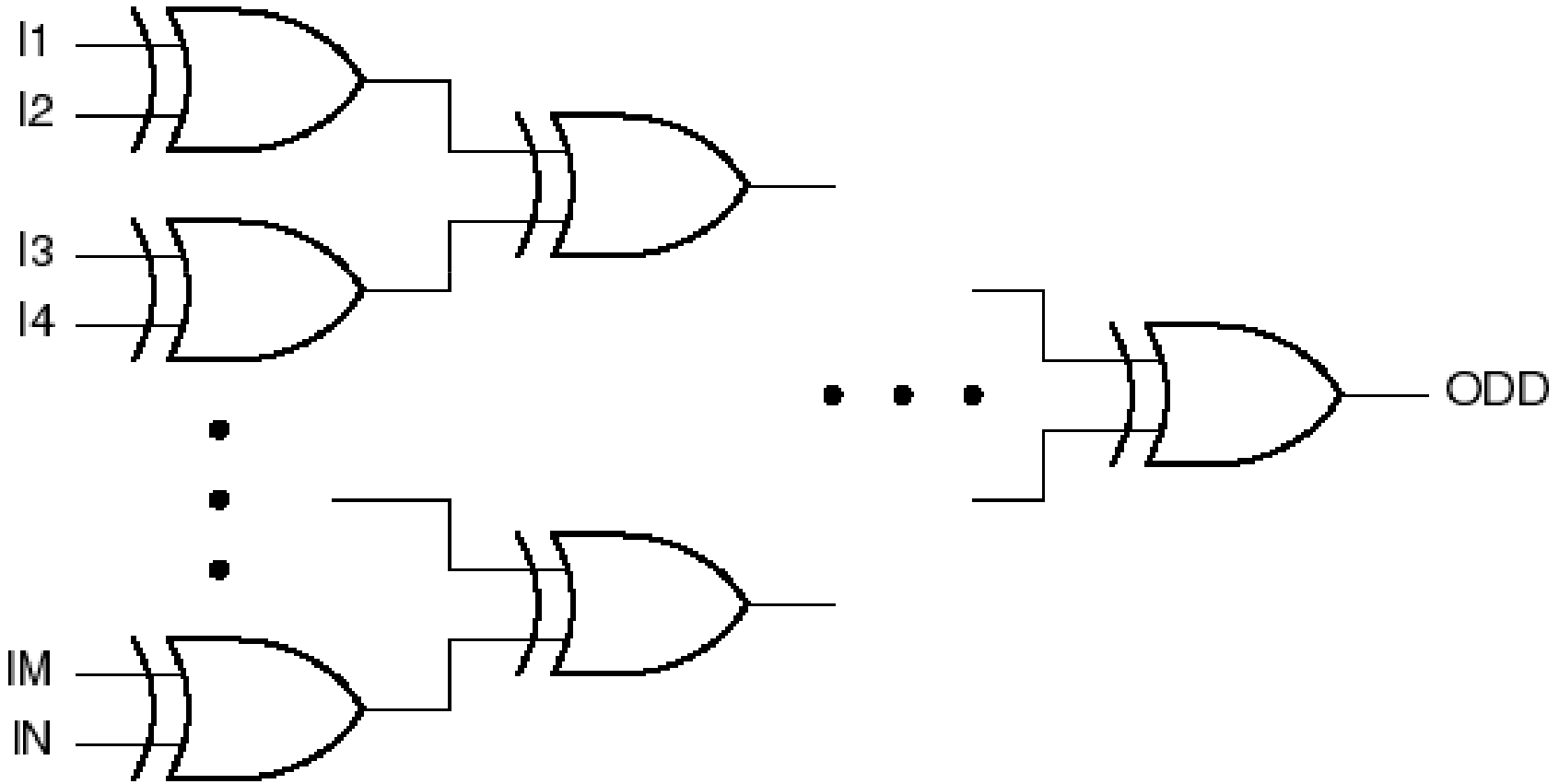
- Sum modulo 2
- Parity computation



- Used to generate and check parity bits in computer systems.
  - Detects any single-bit error

Parity tree

- Faster with balanced tree structure



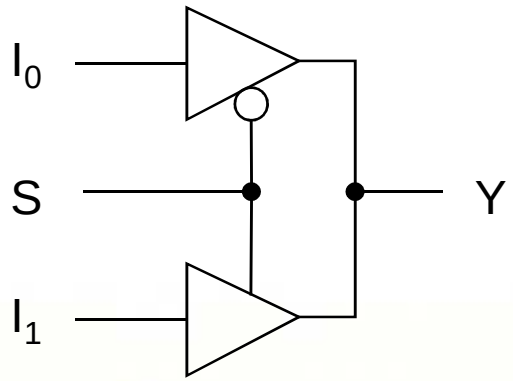
Tri-State

- Logic States: "0", "1"
- Don't Care/Don't Know State: "X" (must be some value in real circuit!)
- Third State: "Z" — high impedance — infinite resistance, no connection
- Tri-state gates:
  - output values are "0", "1", and "Z“
  - additional input: output enable (OE)
- Can tie multiple outputs together, if at most one at a time is driven.

A	OE	F
X	0	Z
0	1	0
1	1	1

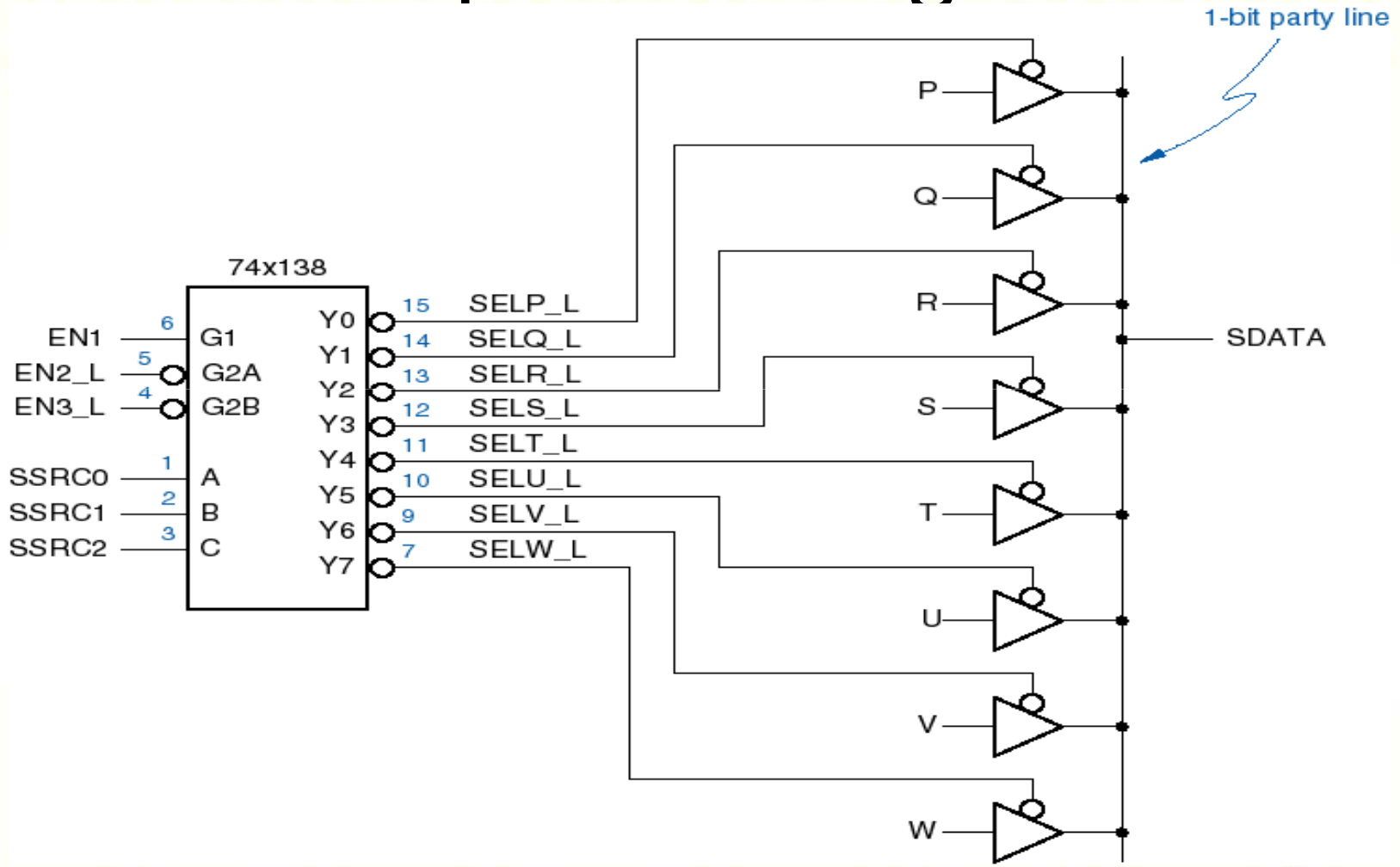
Tri-State

Using tri-state gates to implement an economical multiplexer:



- When S is asserted high
  - I_1 is connected to F
- When S is driven low
  - I_0 is connected to F
- This is essentially a 2:1 Mux

8:1 Multiplexer Using Tri-States

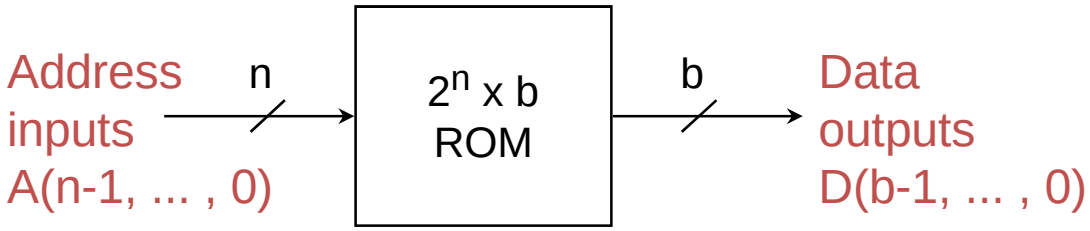


Memory Technology

- Memory chips can store information
- Memory terminologies:
  - Programmable : values determined by user
  - Nonvolatile : contents retained without power
  - Read-only memory: user can read, cannot modify
  - Random-access Memory: user can read and write the memory

Read-Only Memory (ROM)

- A combinational circuit with n inputs and b outputs:
- Two views:
  - ROM stores 2n words of b bits each, or
  - ROM stores an n-input, b-output truth table

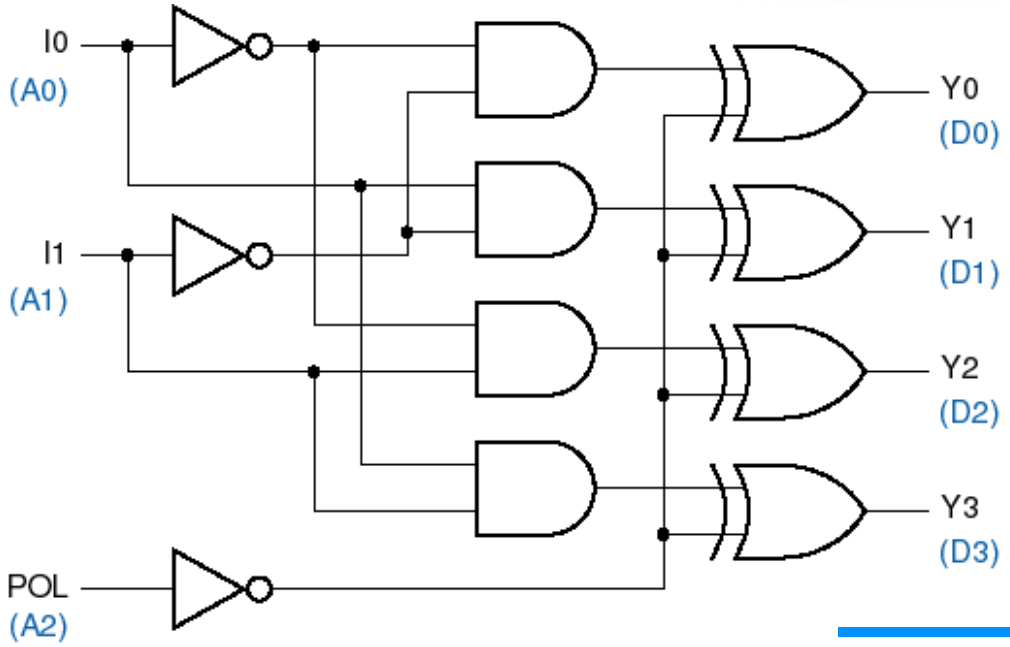


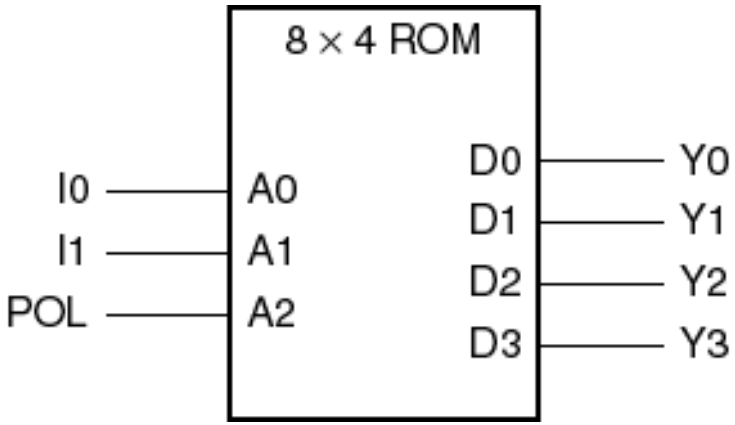
Example:

n = 2		b = 4			
A1	A0	D3	D2	D1	D0
0	0	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	1	1	0	0	0

⇦ Stores 4 4-bit words, or stores 4 functions of 2 input variables

Logic Implementation Using ROM





Inputs			*Outputs*			
A2	*A1*	*A0*	*D3*	*D2*	*D1*	*D0*
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

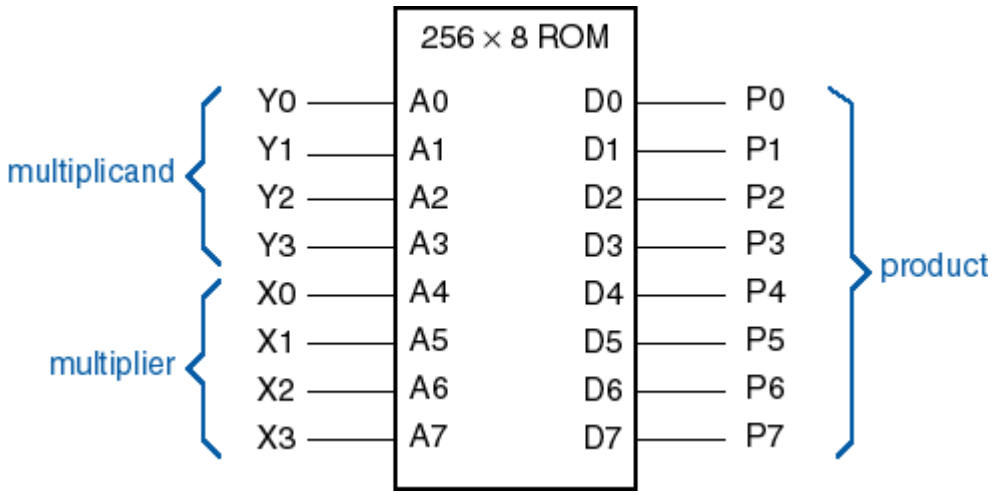
Function: 2-to-4 Decoder with Polarity Control

A2 = Polarity (0 = active Low, 1= active High)

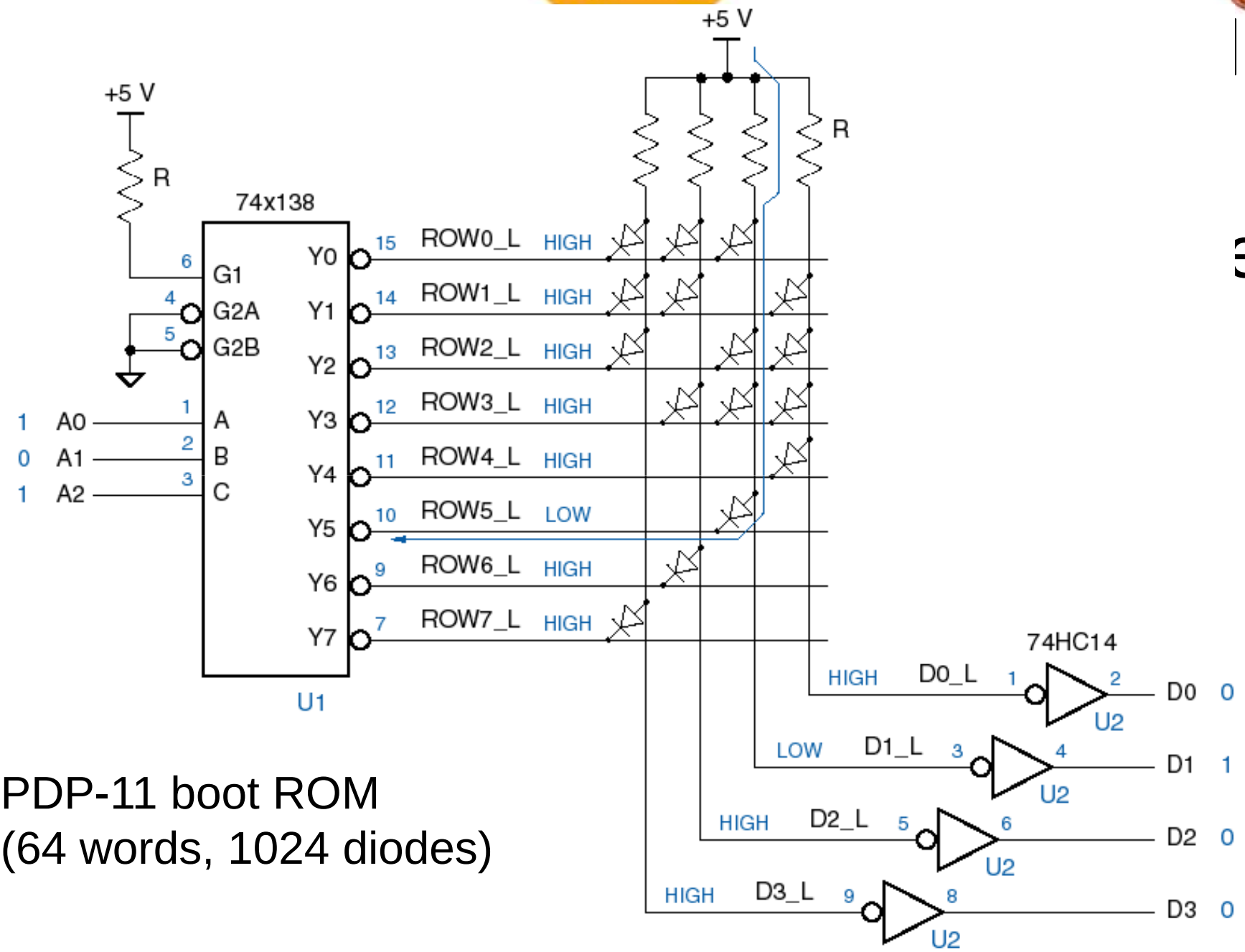
A1, A0 = I1, I0 (2-bit input)

D3...D0 = Y3...Y0 (4-bit decoded output)

4x4 multiplier example

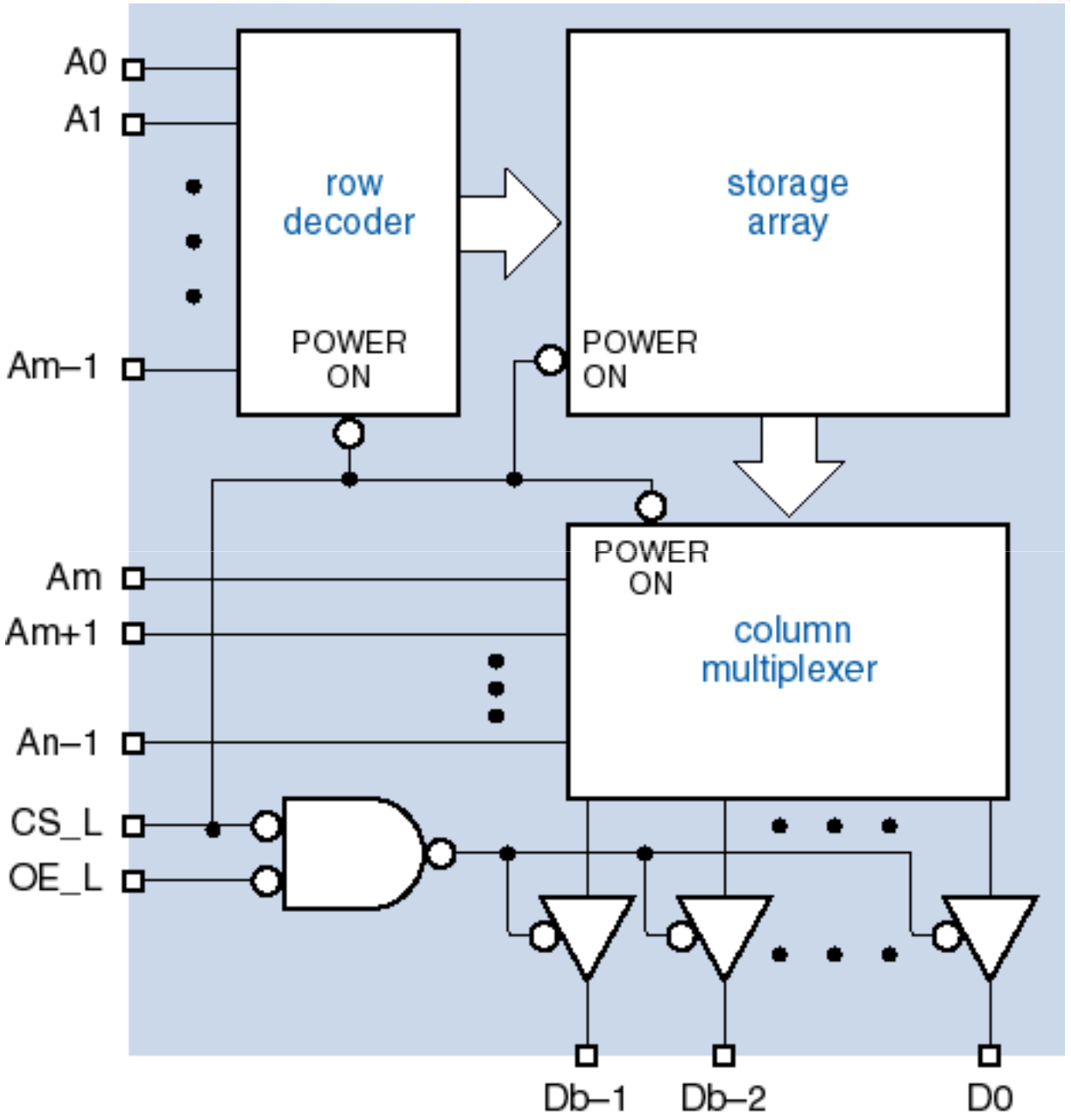



```
00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
20: 00 02 04 06 08 0A 0C 0E 10 12 14 16 18 1A 1C 1E
30: 00 03 06 09 0C 0F 12 15 18 1B 1E 21 24 27 2A 2D
40: 00 04 08 0C 10 14 18 1C 20 24 28 2C 30 34 38 3C
50: 00 05 0A 0F 14 19 1E 23 28 2D 32 37 3C 41 46 4B
60: 00 06 0C 12 18 1E 24 2A 30 36 3C 42 48 4E 54 5A
70: 00 07 0E 15 1C 23 2A 31 38 3F 46 4D 54 5B 62 69
80: 00 08 10 18 20 28 30 38 40 48 50 58 60 68 70 78
90: 00 09 12 1B 24 2D 36 3F 48 51 5A 63 6C 75 7E 87
A0: 00 0A 14 1E 28 32 3C 46 50 5A 64 6E 78 82 8C 96
B0: 00 0B 16 21 2C 37 42 4D 58 63 6E 79 84 8F 9A A5
C0: 00 0C 18 24 30 3C 48 54 60 6C 78 84 90 9C A8 B4
D0: 00 0D 1A 27 34 41 4E 5B 68 75 82 8F 9C A9 B6 C3
E0: 00 0E 1C 2A 38 46 54 62 70 7E 8C 9A A8 B6 C4 D2
F0: 00 0F 1E 2D 3C 4B 5A 69 78 87 96 A5 B4 C3 D2 E1
```



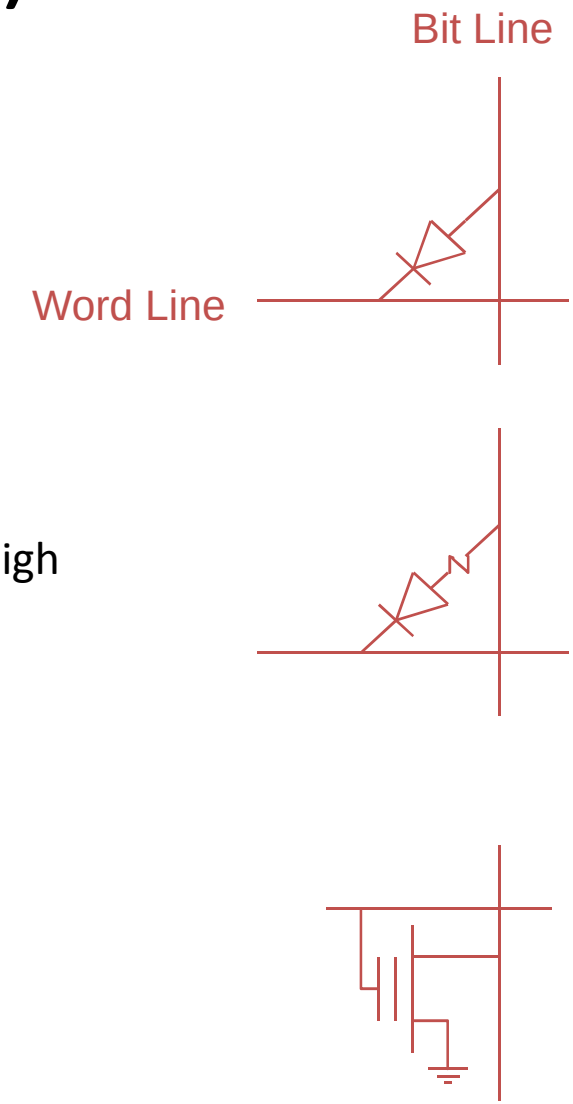
PDP-11 boot ROM
(64 words, 1024 diodes)

ROM control and I/O signals



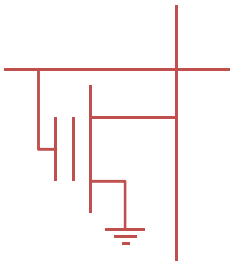
Types Of ROMs (1)

- Mask ROM
  - Programming using **mask** by memiconductor vendor
    - The only type that cannot be programmed by user
  - Expensive setup cost
  - Several weeks for delivery
  - Needs high volume to be cost-effective
- PROM
  - Programmable ROM
  - Vaporize (blow) fusible links with PROM programmer using high voltage/current pulses
  - Once blown, fuse cannot revert to original state
- EPROM
  - Erasable Programmable ROM
  - Charge trapped on extra “floating gate” of MOS transistors
  - Exposure to UV light removes charge
    - 10-20 minutes
    - Quartz Lid = expensive package
  - Limited number of erasures (10-100)



Types of ROMs (3)

- OTP ROM
  - One Time Programmable ROM
  - EPROM packaged in cheaper plastic packaging
  - Cannot erase once programmed
  - Used together with EPROM
    - Use regular EPROM for R&D, engineers need to reprogram it
    - Use OTP for shipping product, consumers do not reprogram chips
- EEPROM (E^2PROM)
  - Electrically Erasable PROM
  - Floating gates charged/discharged electrically
  - Cannot replace RAM! (slow write)
  - Limited number of charge/discharge cycles (10,000)
- Flash Memory
  - Electronically erasable in blocks
  - 100,000 erase cycles
  - Simpler and denser than EEPROM
  - Very popular in digital cameras, handphones, thumb drives, etc



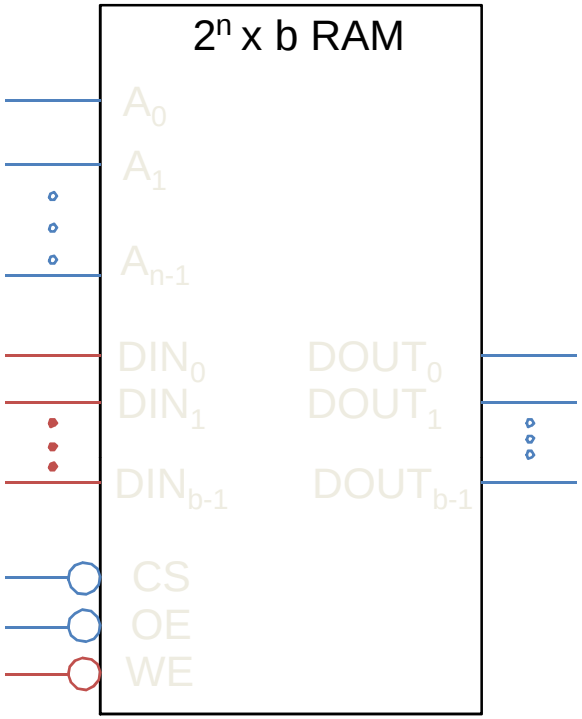
ROM Type Summary

Type	Technology	Read Cycle	Write Cycle	Comments
Mask ROM	NMOS, CMOS	20-200 ns	4 weeks	Write once; low power
Mask ROM	Bipolar	<100 ns	4 Weeks	Write once; high power, low density
PROM	Bipolar	<100 ns	5 minutes	Write once; high power; no mask charge
EPROM	NMOS, CMOS	52-200 ns	5 minutes	Reusable; low power; no mask charge
EEPROM	NMOS	50-200 ns	10 ms/byte	10,000 writes/location limit
FLASH	CMOS	25-200 ns	10 ms/block	100,000 erase cycles

Random Access Memory (RAM)

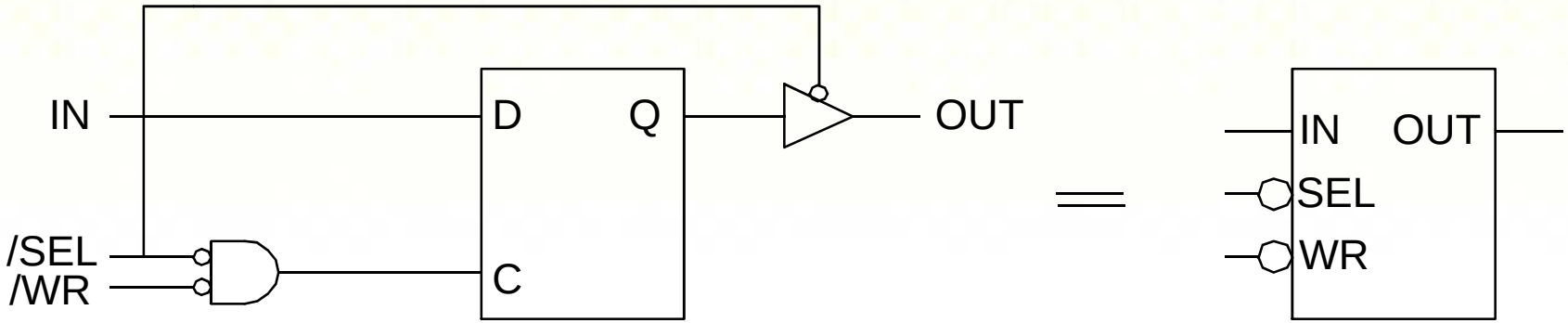
- Better name should be Read/Write Memory (RWM)
- Can store and retrieve data at the same speed
- Two types:
  - Static RAM (SRAM) retains data in latches (while powered)
  - Dynamic RAM (DRAM) stores data as capacitor charge; all capacitors must be recharged periodically.
- Static RAM:
  - Fast
  - Simple interface
  - Moderate bit density (4 to 6 transistors per bit)
  - Moderate cost/bit
  - For small systems or very fast requirements (eg cache memory)
- Dynamic RAM:
  - Moderate speed
  - Complex interface
  - High bit density (1 transistor cell per bit)
  - Low cost/bit
  - For large memories (eg main memory in PC, servers)

Basic Structure of SRAM



- Address/Control/Data Out lines like a ROM (Reading)
  + Write Enable (WE) and Data In (DIN) (Writing)

One Bit of SRAM



Control Input	Chip Function
SEL and WR asserted	IN data stored in D-latch (Write)
SEL only asserted	D-latch output enabled (Read)
SEL not asserted	No operation

A worked Example

- Given these functions

  X(A,B,C,D) = m(1,3,5,7,8,9,11)

  Y(A,B,C,D) = m(0,2,4,5,7,8,10,11,12)

  Z(A,B,C,D) = m(1,2,3,5,7,10,12,13,14,15)

  - Implement X using AND/OR
  - Implement Y using NAND/NAND
  - Implement Z using NOR/NOR
  - Implement Y using 8:1 multiplexer
  - Implement XYZ using 74x138 decoder and NAND gates
  - Implement XYZ using ROM

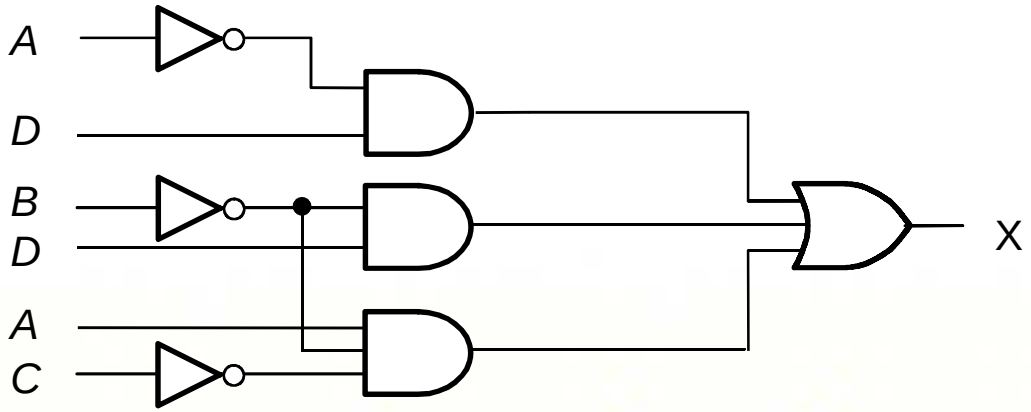
Implement

X(A,B,C,D) = m(1,3,5,7,8,9,11)

Using AND/OR gates

A B \ C D	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	0	0
10	1	1	1	0

Solving the K-map we get X = A'D + B'D + AB'C'



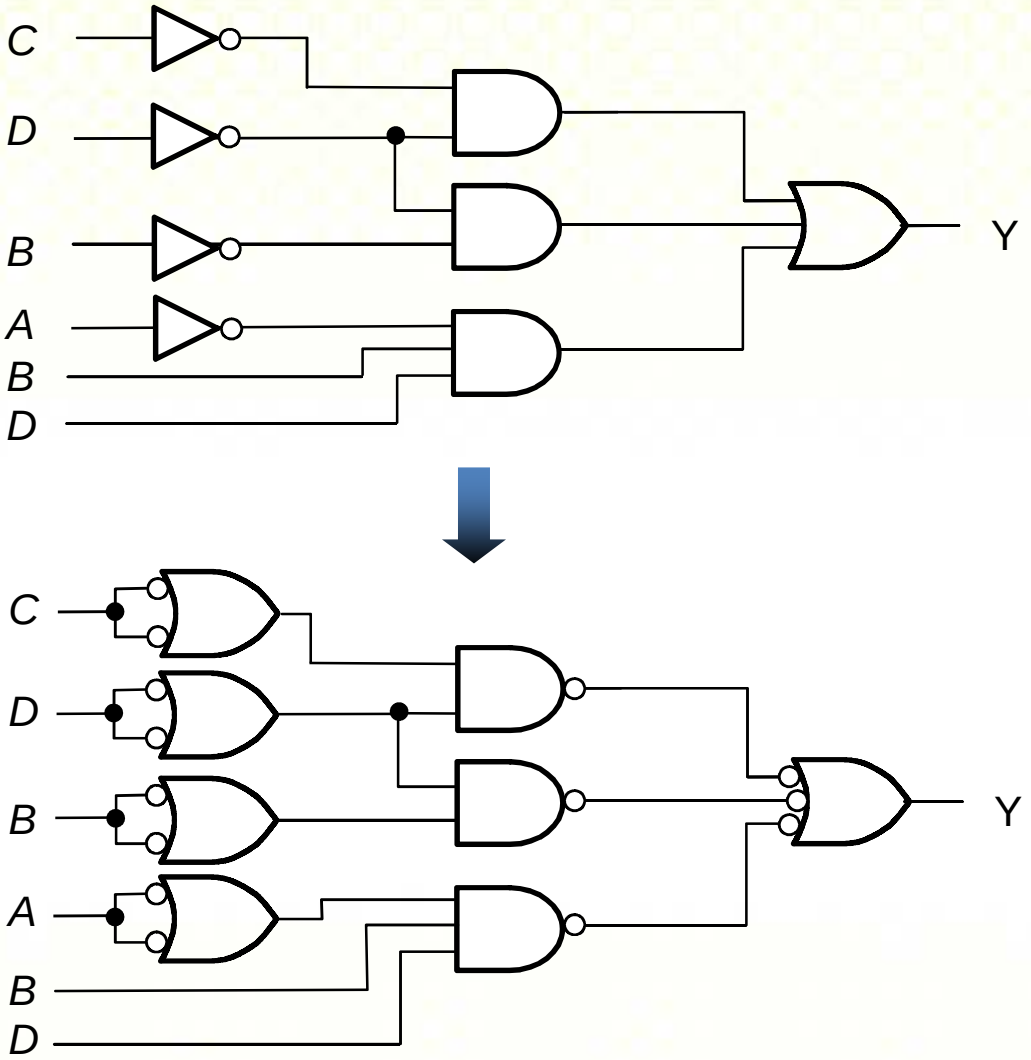
Implement using NAND gates

$Y(A,B,C,D) = \Sigma m(0,2,4,5,7,8,10,11,12)$

A B \ C D	00	01	11	10
00	1	0	0	1
01	1	1	1	0
11	1	0	0	0
10	1	0	1	1

Solving the K-map we get

$Y = C'D' + B'D' + A'BD$



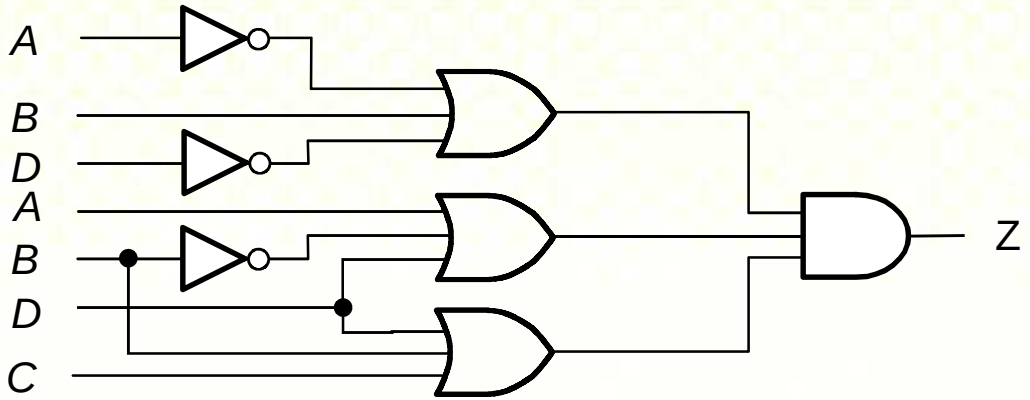
Exercise:

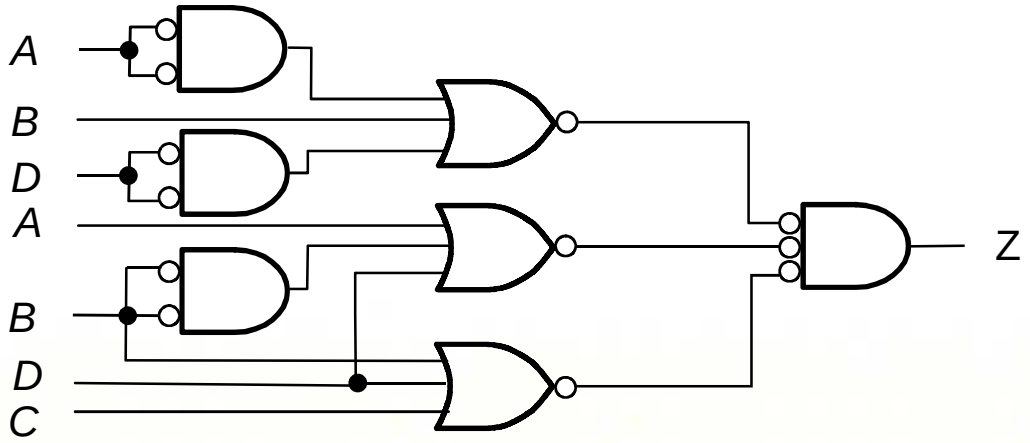
Implement Y using NOR gates

Implement using NOR gates

Z(A,B,C,D) = Σm(1,2,3,5,7,10,12,13,14,15)

A B \ CD	00	01	11	10
00	0	1	1	1
01	0	1	1	0
11	1	1	1	1
10	0	0	0	1





Solving the K-map using POS gives

Z = (A' + B + D')(A + B' + D)(B + C + D)

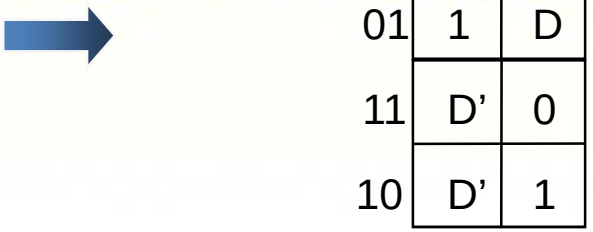
Exercise:

Implement Z using NAND gates

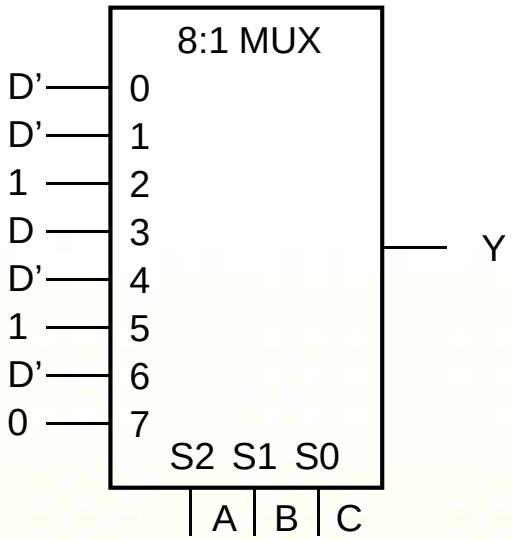
Implement using 8:1 MUX

$Y(A,B,C,D) = \Sigma m(0,2,4,5,7,8,10,11,12)$

A B \ C D	00	01	11	10
00	1	0	0	1
01	1	1	1	0
11	1	0	0	0
10	1	0	1	1



A B \ C	0	1
00	D’	D’
01	1	D
11	D’	0
10	D’	1



Exercise 1:

Implement Y using 8:1 MUX but with C connected to S2, B to S1, and A connected to S0.

Exercise 2:

Implement Y using 8:1 MUX but with B connected to S2, C to S1, and D connected to S0.

Implement using 74x138

X(A,B,C,D) = m(1,3,5,7,8,9,11)
Z(A,B,C,D) = m(1,2,3,5,7,10,12,13,14,15)

Exercise 1:

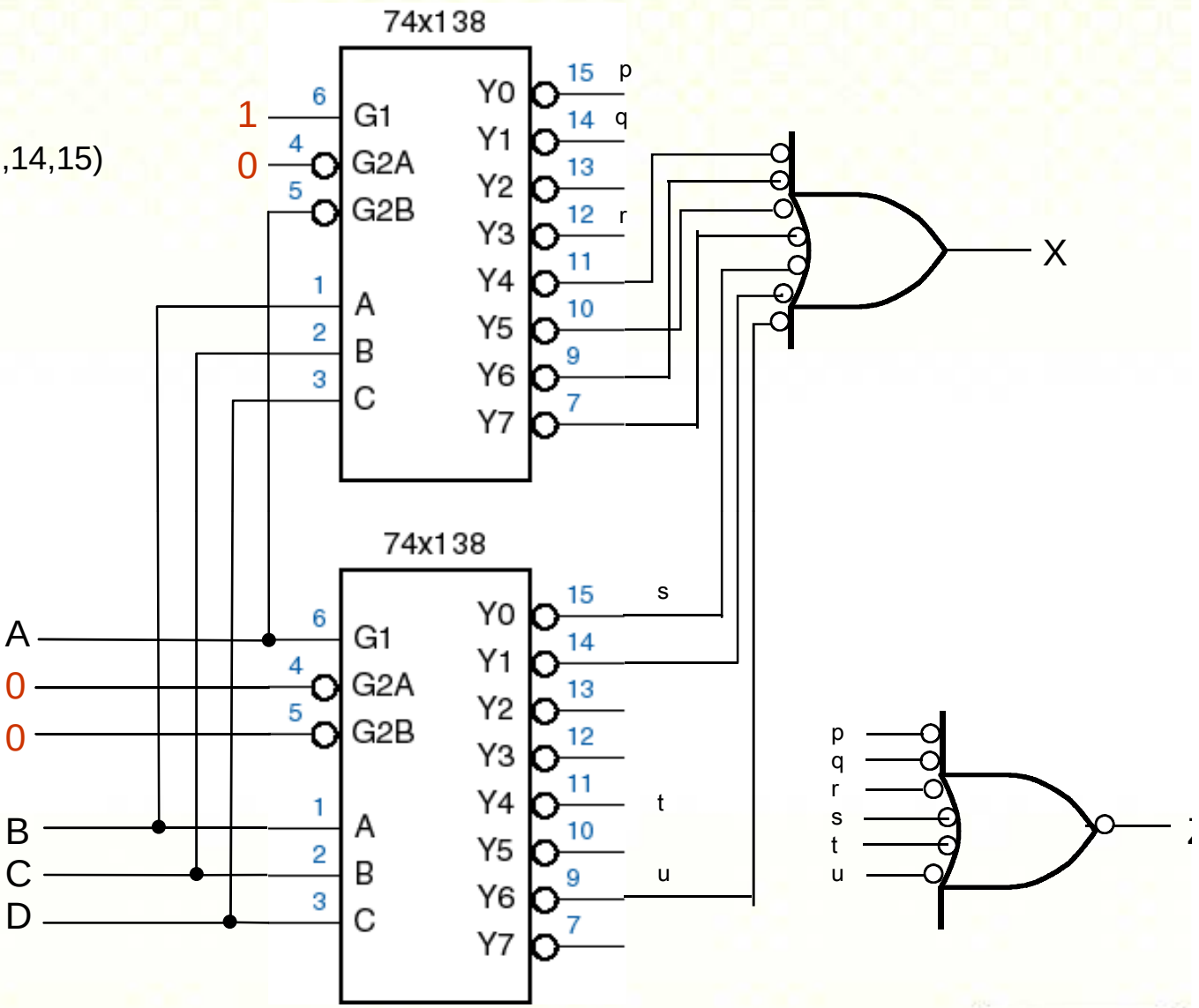
Implement X but with input B, C, D connected to decoder selectors C, B, A respectively.

Exercise 2:

Implement Z but with input B, C, D connected to decoder selectors C, B, A respectively.

Exercise 3:

Implement Z with input B, C, D connected to decoder selectors C, B, A respectively, but you must use a 10-input NAND gate.



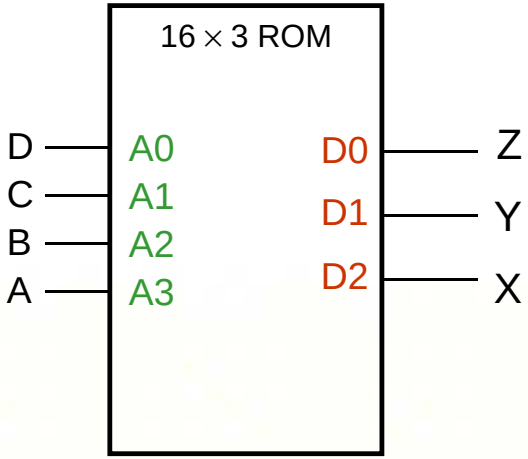
Implement

X(A,B,C,D) = m(1,3,5,7,8,9,11)

Y(A,B,C,D) = m(0,2,4,5,7,8,10,11,12)

Z(A,B,C,D) = m(1,2,3,5,7,10,12,13,14,15)

Using ROM



Input				Output		
A3	A2	A1	A0	D2	D1	D0
0	0	0	0	0	1	0
0	0	0	1	1	0	1
0	0	1	0	0	1	1
0	0	1	1	1	0	1
0	1	0	0	0	1	0
0	1	0	1	1	1	1
0	1	1	0	0	0	0
0	1	1	1	1	1	1
1	0	0	0	1	1	0
1	0	0	1	1	0	0
1	0	1	0	0	1	1
1	0	1	1	1	1	0
1	1	0	0	0	1	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	0	1