Week 4

- Tri-State
- Memories
  - ROM
  - RAM

Tri-State

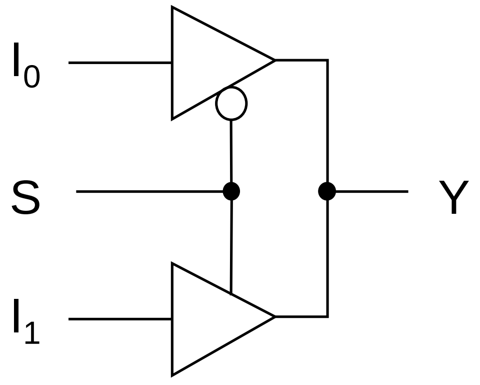
- Logic States: "0", "1"
- Don't Care/Don't Know State: "X" (must be some value in real circuit!)
- Third State: "Z" — high impedance — infinite resistance, no connection
- Tri-state gates:
  - output values are "0", "1", and "Z“
  - additional input: output enable (OE)

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

- Can tie multiple outputs together, if at most one at a time is driven.

Tri-State

Using tri-state gates to implement an economical multiplexer:

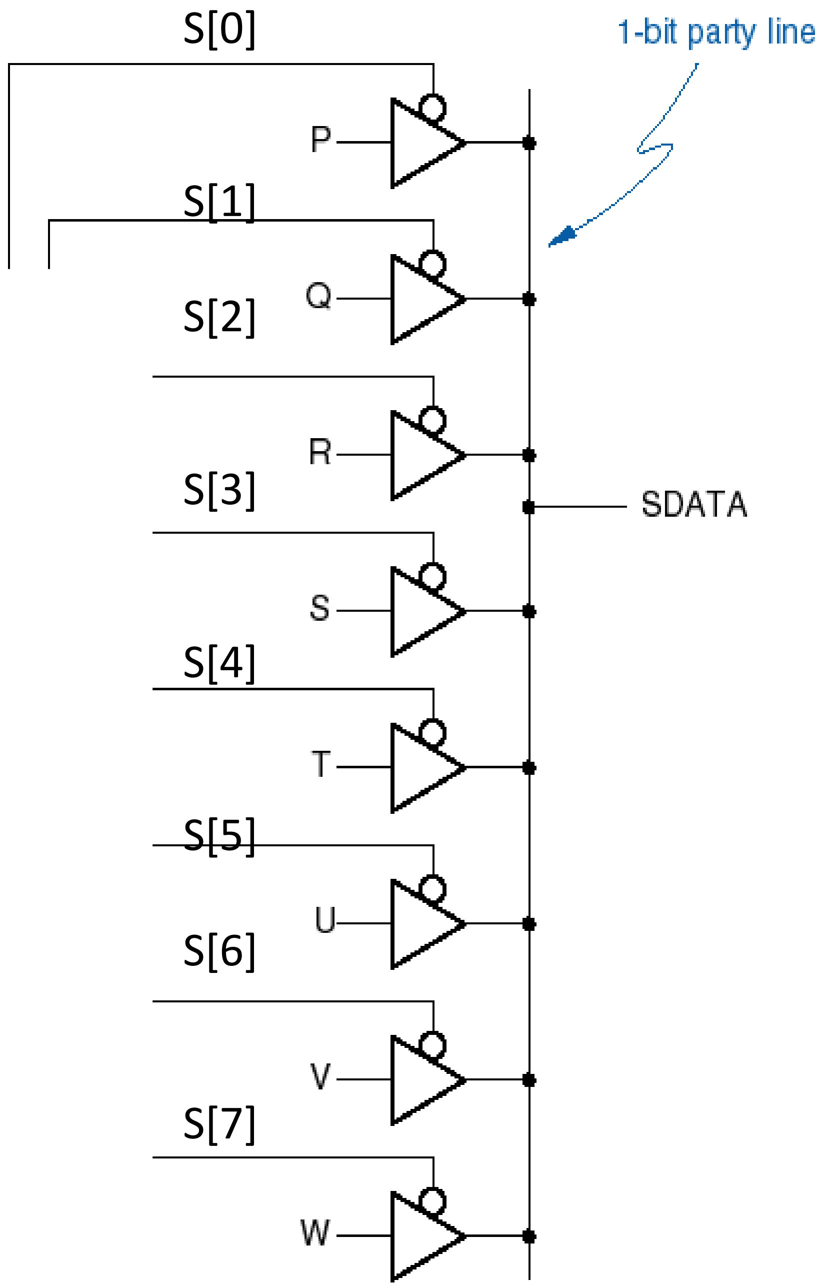


- When S is asserted high
  - I_1 is connected to Y
- When S is driven low
  - I_0 is connected to Y
- This is essentially a 2:1 Mux

Tri-state in Verilog

```
module muxtri (s, in0, in1, y);
input s, in0, in1;
output y;

assign y= s ? 1'bz : in0;
assign y= s ? in1 : 1'bz;

endmodule
```

8:1 Multiplexer Using Tri-States



Tri-state in verilog

```
module mux8to1 (s, p, q, r, s, t, u, v, w, sdata);
input p, q, r, s, t, u, v, w;
input [7:0] s;
output sdata;

assign sdata = s[7] ? w :1'bz;
assign sdata = s[6] ? v :1'bz;
assign sdata = s[5] ? u :1'bz;
assign sdata = s[4] ? t :1'bz;
assign sdata = s[3] ? s :1'bz;
assign sdata = s[2] ? r :1'bz;
assign sdata = s[1] ? q :1'bz;
assign sdata = s[0] ? p :1'bz;

endmodule
```

Tri-state verilog

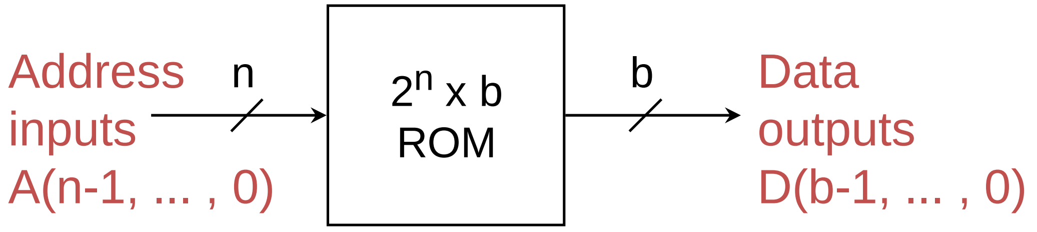
```
module mux8to1 (codeword, p, q, r, s, t,
u, v, w, sdata);
input p, q, r, s, t, u, v, w;
input [2:0] codeword;
reg [7:0] sel;
output sdata;

assign sdata = sel[7] ? w :1'bz;
assign sdata = sel[6] ? v :1'bz;
assign sdata = sel[5] ? u :1'bz;
assign sdata = sel[4] ? t :1'bz;
assign sdata = sel[3] ? s :1'bz;
assign sdata = sel[2] ? r :1'bz;
assign sdata = sel[1] ? q :1'bz;
assign sdata = sel[0] ? p :1'bz;

always @(codeword)
begin
sel=0;

case (codeword)
7 : sel = 8'h80;
6 : sel = 8'h40;
5 : sel = 8'h20;
4 : sel = 8'h10;
3 : sel = 8'h08;
2 : sel = 8'h04;
1 : sel = 8'h02;
0: sel = 8'h01;
endcase

end

endmodule
```

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 2^n words of b bits each, or
  - ROM stores an n-input, b-output truth table

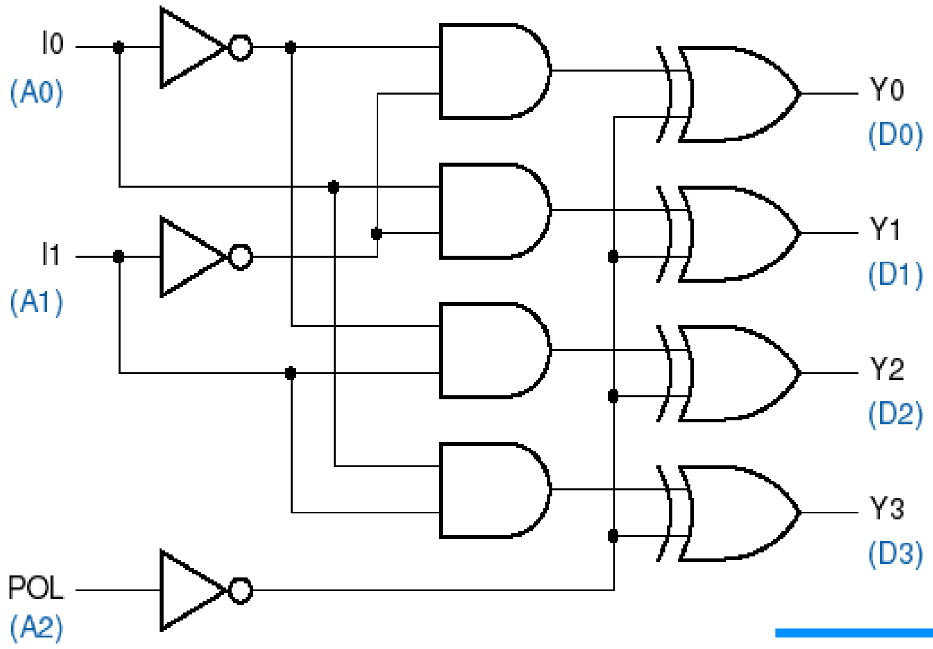


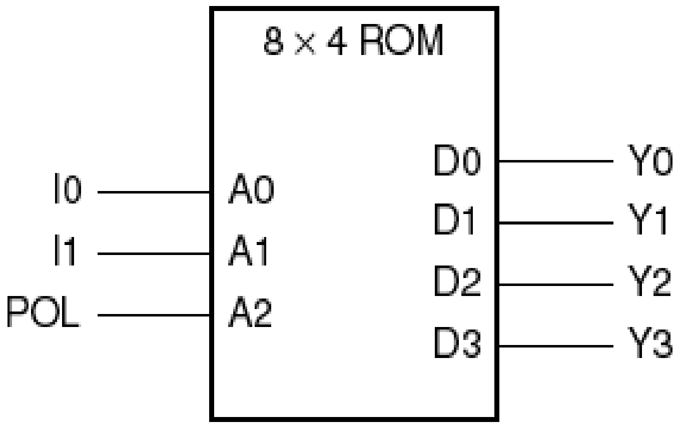
Example: n = 2, b = 4

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



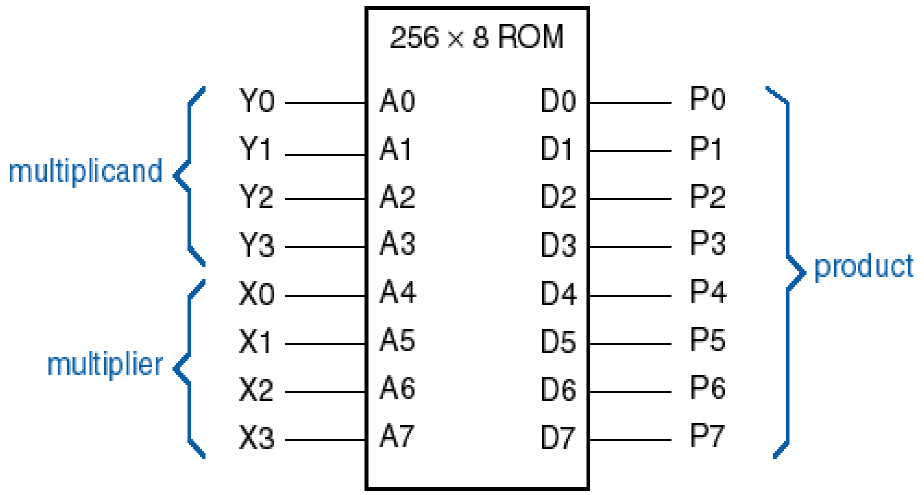


Inputs			*Outputs*			
A2	*A1*	*A0*	*D3*	*D2*	*D1*	*D0*
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Function: 2-to-4 Decoder with Polarity Control

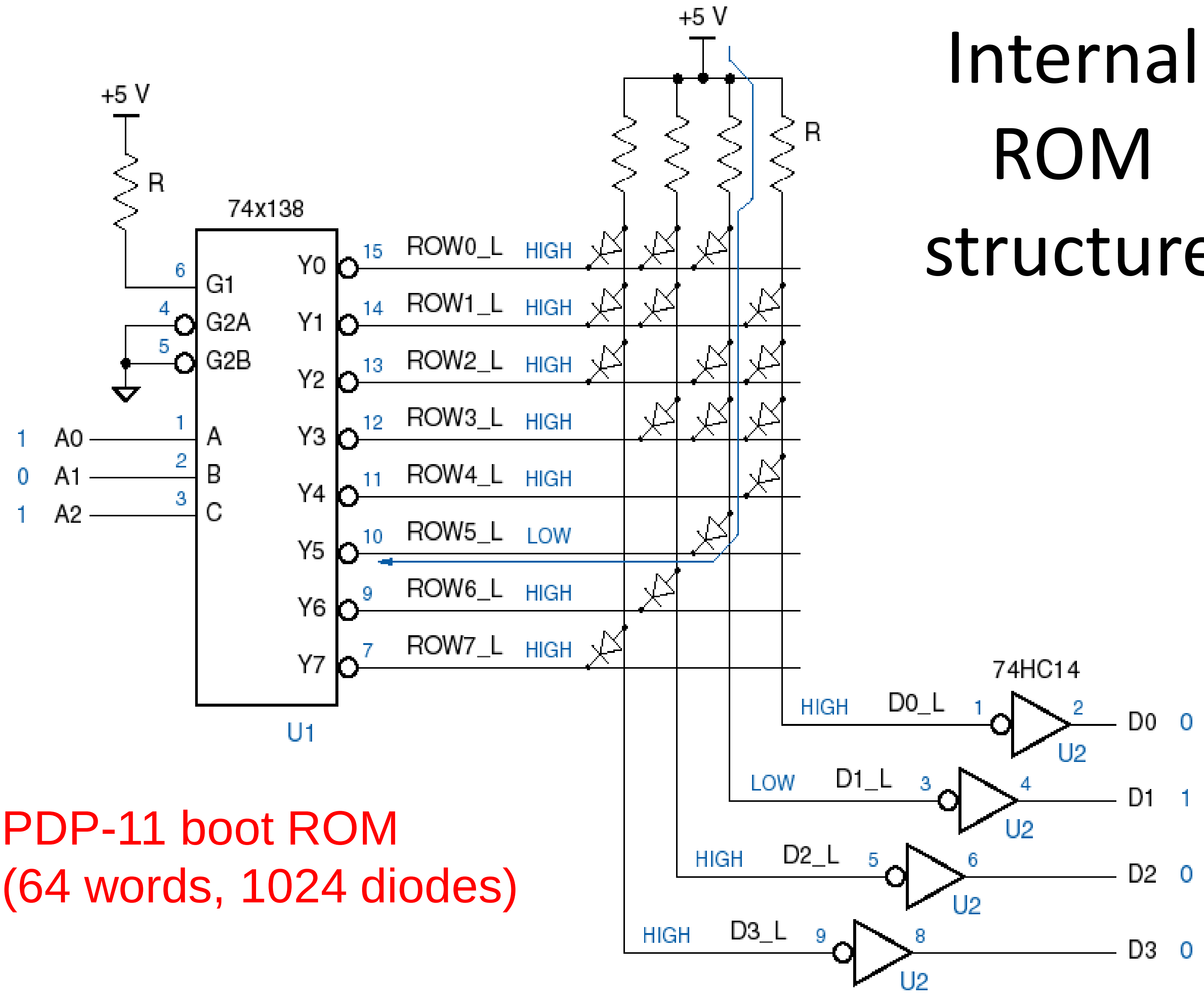
A2 = Polarity	(0 = active Low, 1= active High)
A1, A0 = I1, I0	(2-bit input)
D3...D0 = Y3...Y0	(4-bit decoded output)

4x4 multiplier example



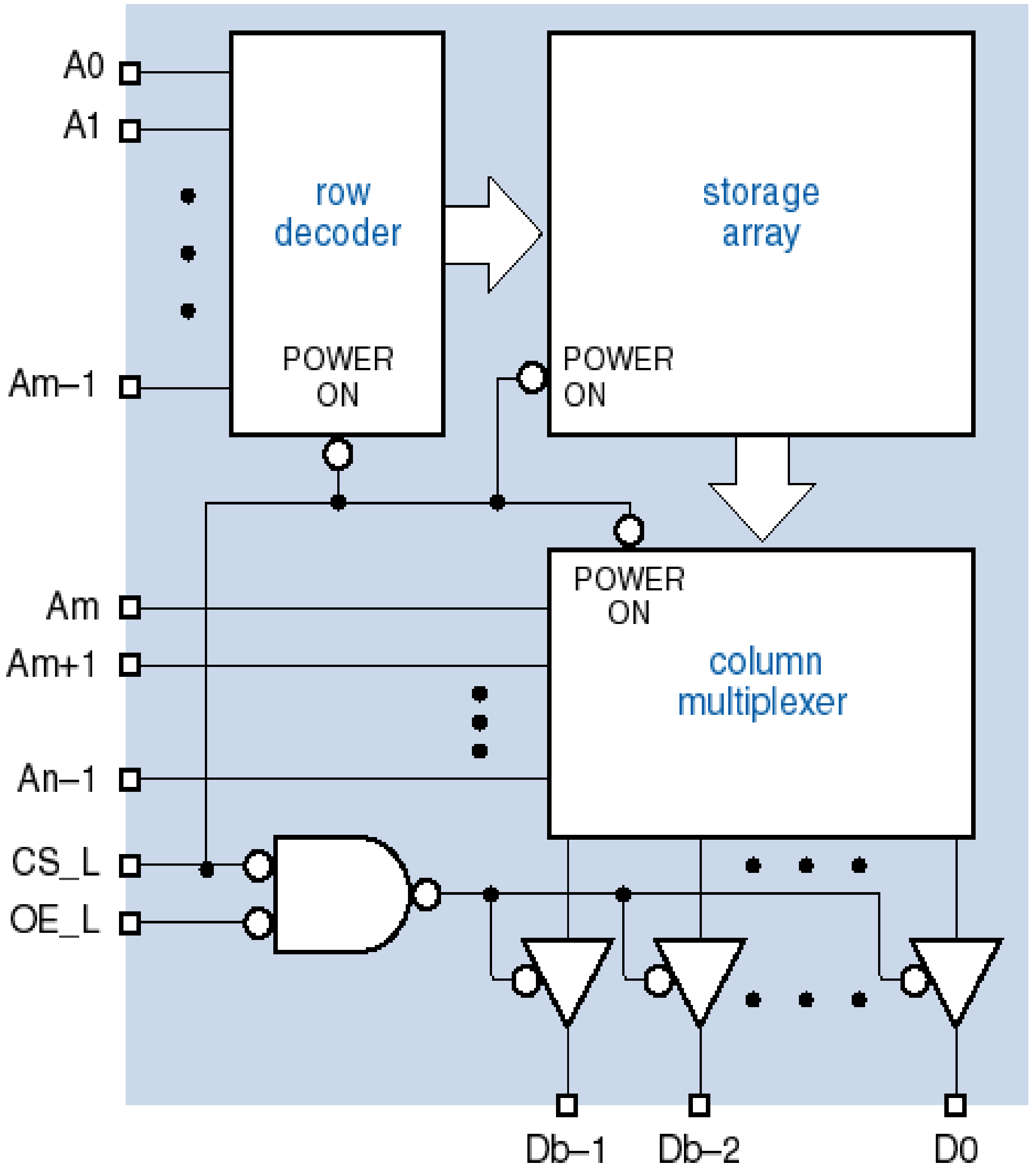
X \ Y	Decimal	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Decimal	Binary	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
1	0001	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	0010	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	0011	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	0100	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0101	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0110	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0111	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	1000	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	1001	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
10	1010	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
11	1011	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
12	1100	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
13	1101	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
14	1110	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
15	1111	0F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Internal ROM structure



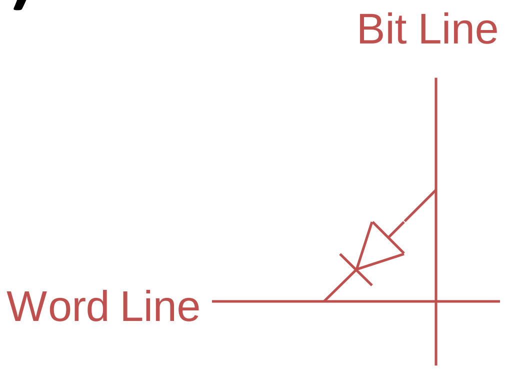
PDP-11 boot ROM
(64 words, 1024 diodes)

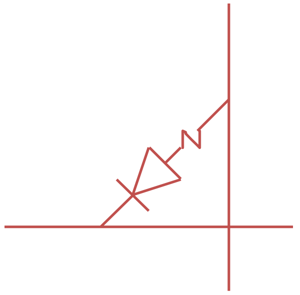
ROM control and I/O signals

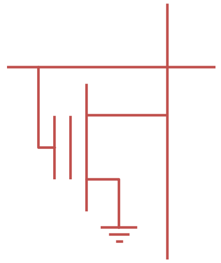


Types Of ROMs (1)

- Mask ROM
  - Programming using **mask** by semiconductor 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)

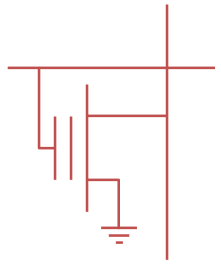






Types of ROMs (3)

- 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^{2}PROM)
  - 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



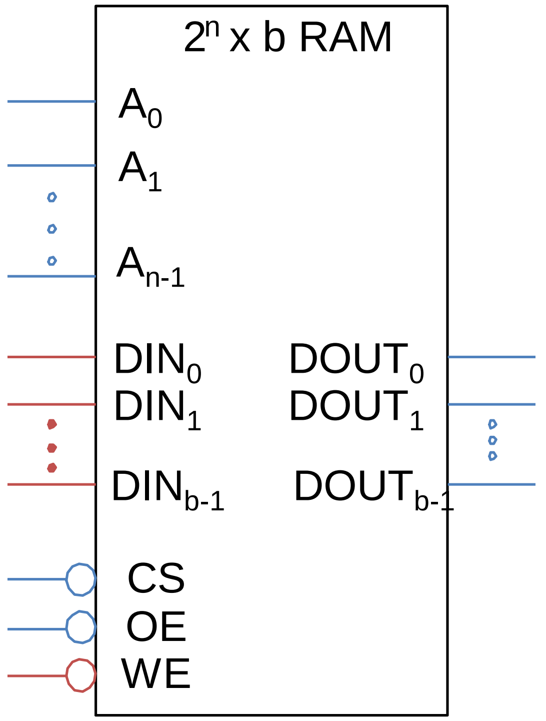
ROM Type Summary

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)

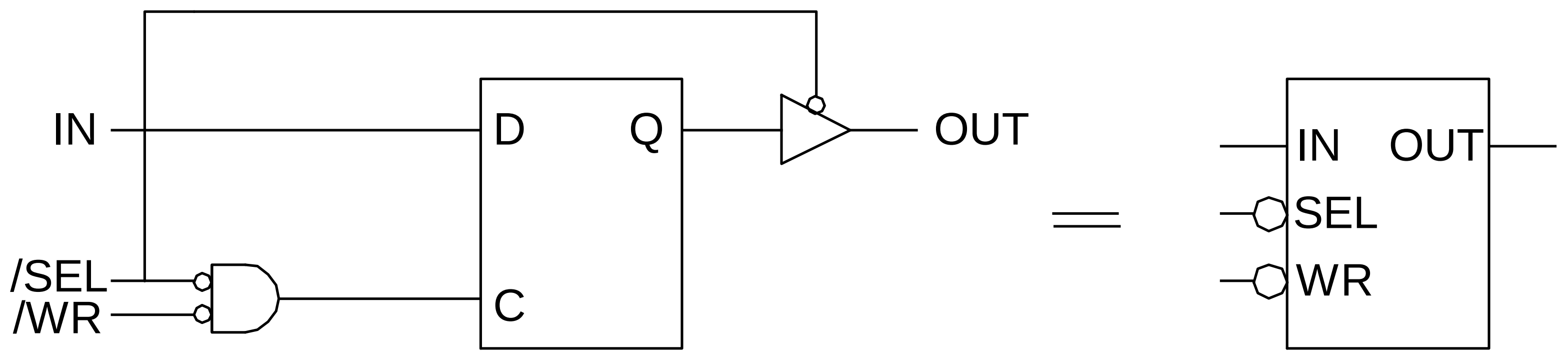
- Better name should be Read/Write Memory (RWM)
- Can store and retrieve data at the same speed
- Two types:
  - Static RAM (SRAM) retains data in latches (while powered)
  - Dynamic RAM (DRAM) stores data as capacitor charge; all capacitors must be recharged periodically.
- Static RAM:
  - Fast
  - 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)

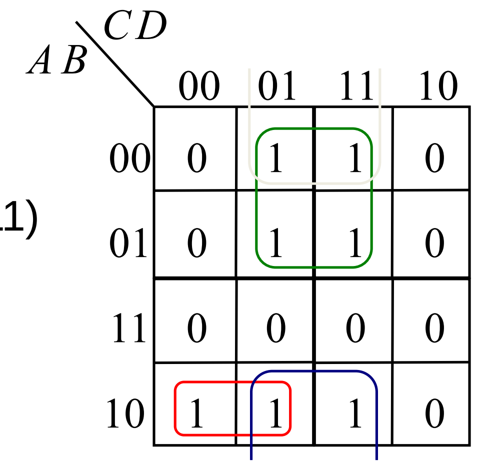
  Z(A,B,C,D) = m(1,2,3,5,7,10,12,13,14,15)

  – 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

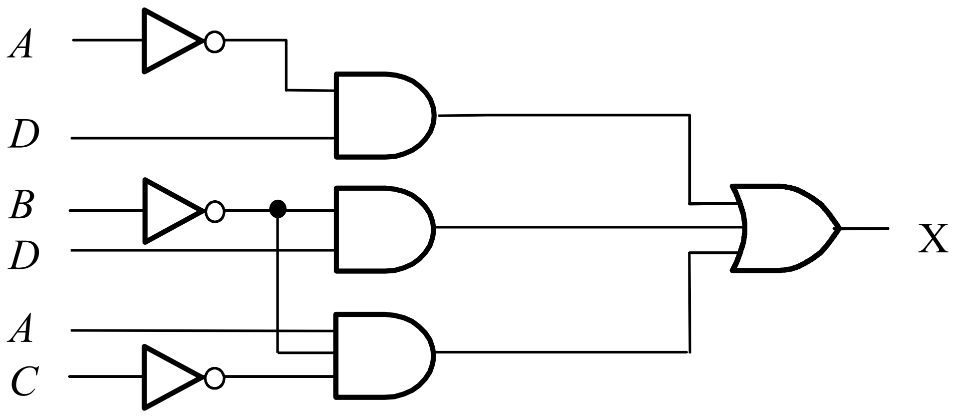
X(A,B,C,D) = m(1,3,5,7,8,9,11)

Using AND/OR gates



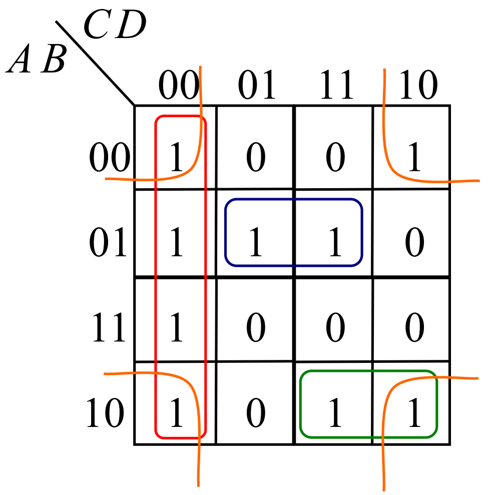
AB \ CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	0	0
10	1	1	1	0

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



Implement using NAND gates

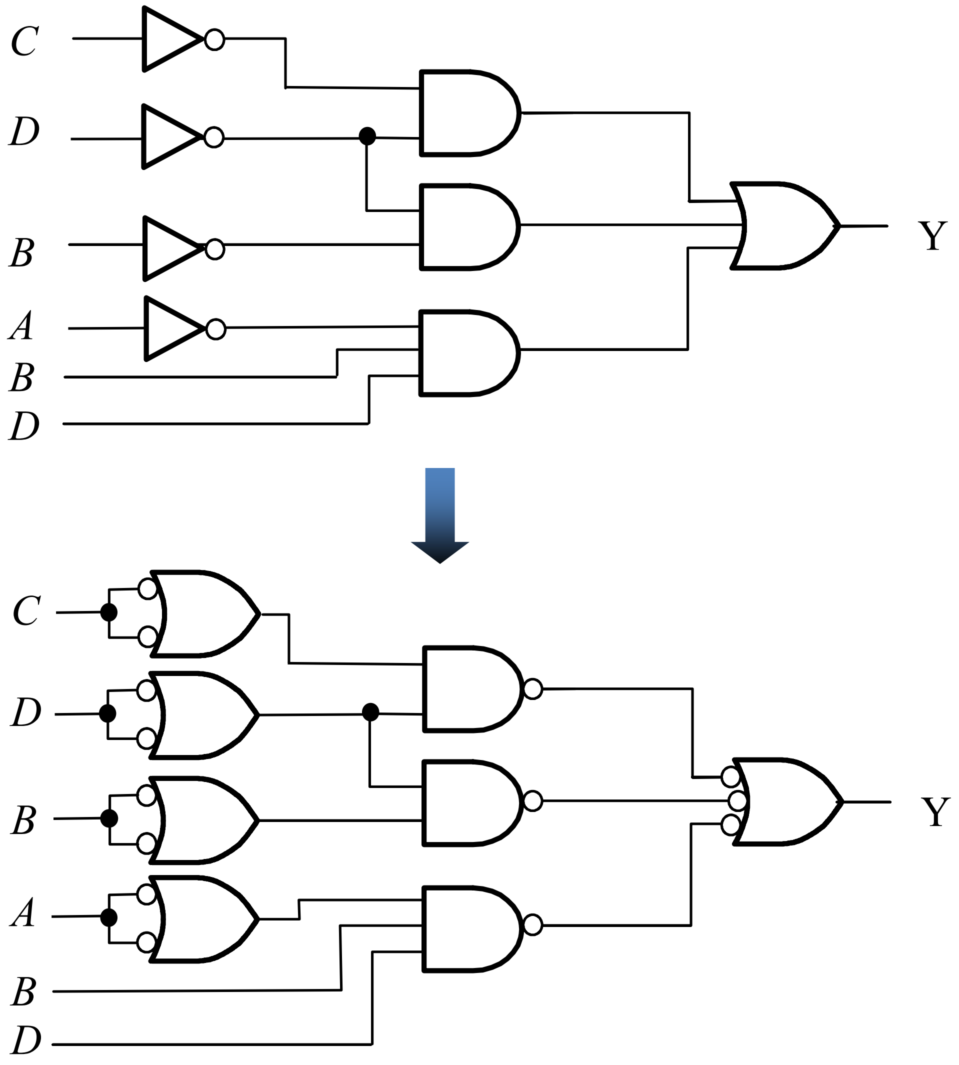
Y(A,B,C,D)Σ = m(0,2,4,5,7,8,10,11,12)



AB \ CD	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

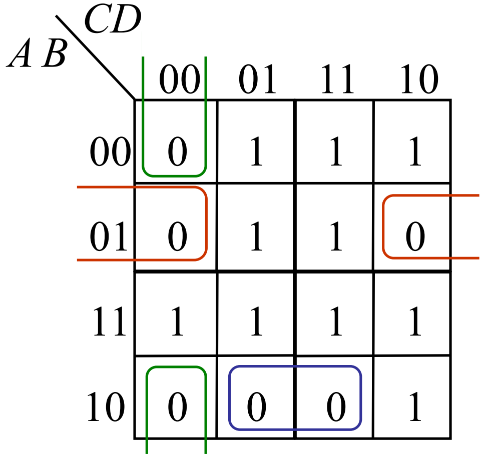


Exercise:

Implement Y using NOR gates

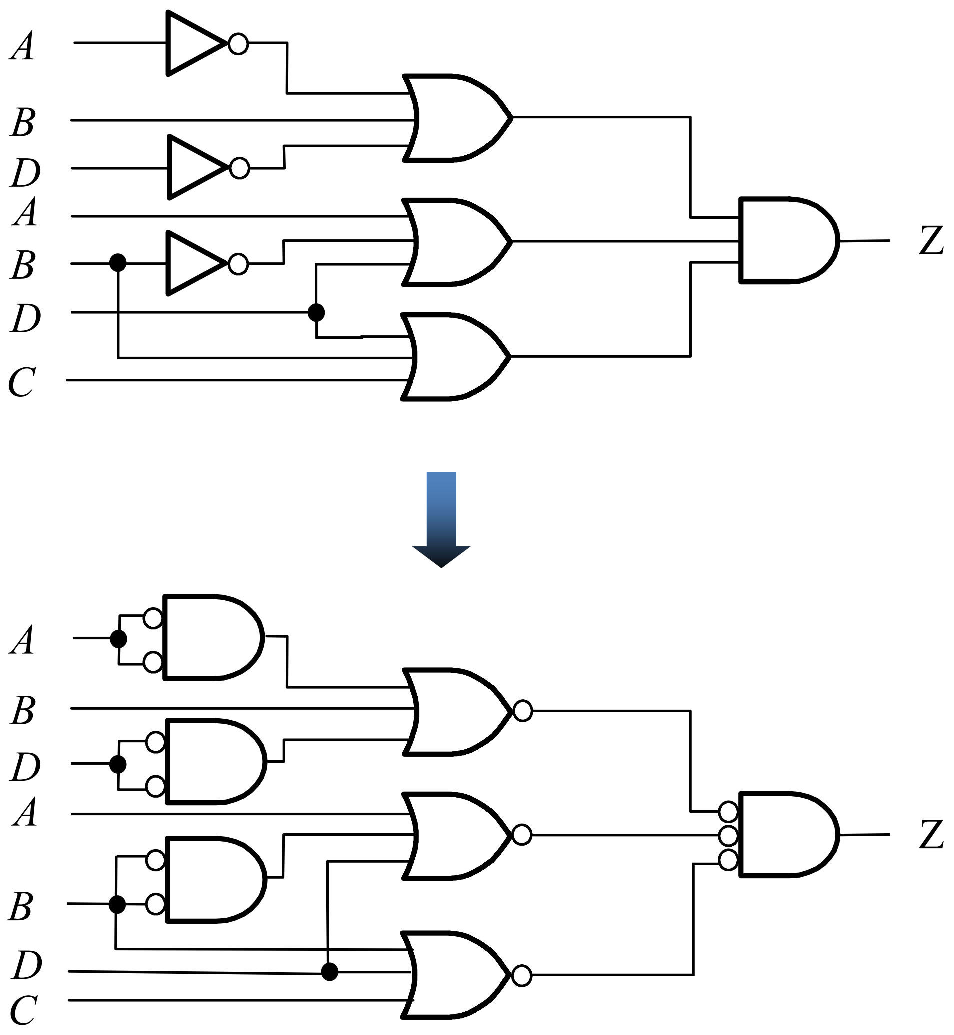
Implement using NOR gates

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



AB \ 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)

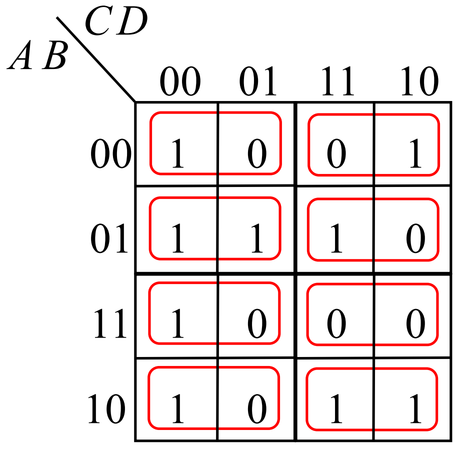


Exercise:

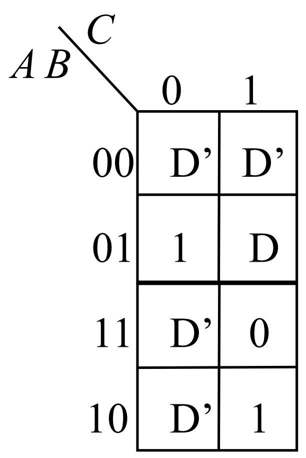
Implement Z using NAND gates

Implement using 8:1 MUX

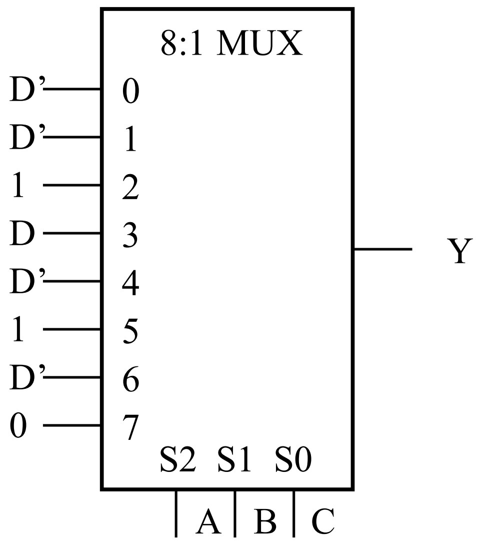
Y(A,B,C,D)Σ = m(0,2,4,5,7,8,10,11,12)



AB \ CD	00	01	11	10
00	1	0	0	1
01	1	1	1	0
11	1	0	0	0
10	1	0	1	1



AB \ 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:

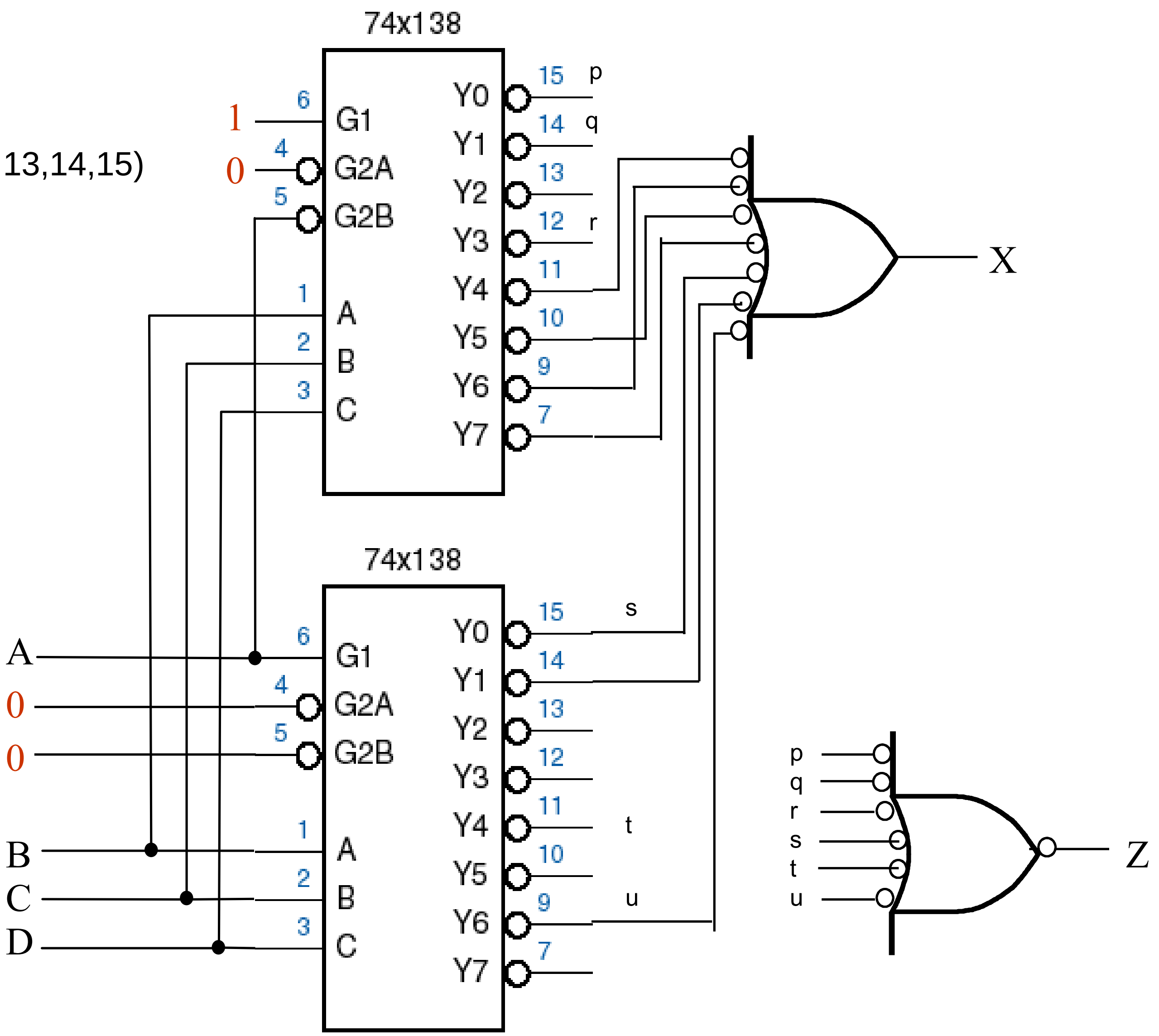
Implement X but with input B, C, D connected to decoder selectors C, B, A respectively.

Exercise 2:

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

Exercise 3:

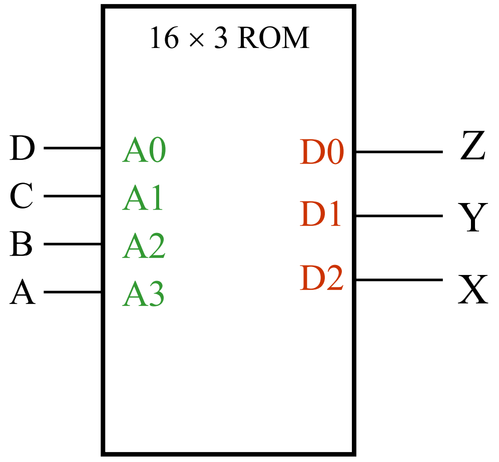
Implement Z with input B, C, D connected to decoder selectors C, B, A respectively, but you must use a 10-input NAND gate.



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