CS2100
COMPUTER ORGANISATION

http://www.comp.nus.edu.sg/~cs2100/

CS2100
Recitation 9

MSI Components

24 March 2025

Aaron Tan

NUS National University of Singapore | School of Computing

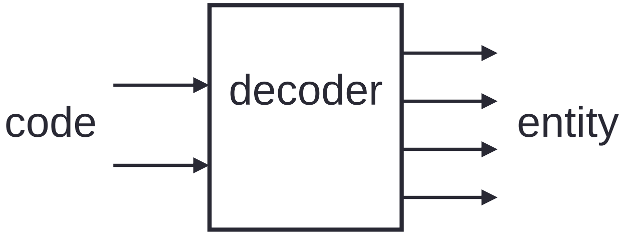
Contents

- Decoders
  - Implementing functions using decoders
  - Decoder with enable
  - Standard MSI decoder
- Encoders
  - Priority encoders
- Multiplexers
  - Standard MSI multiplexer
  - Implementing functions using multiplexers
  - Using smaller multiplexers

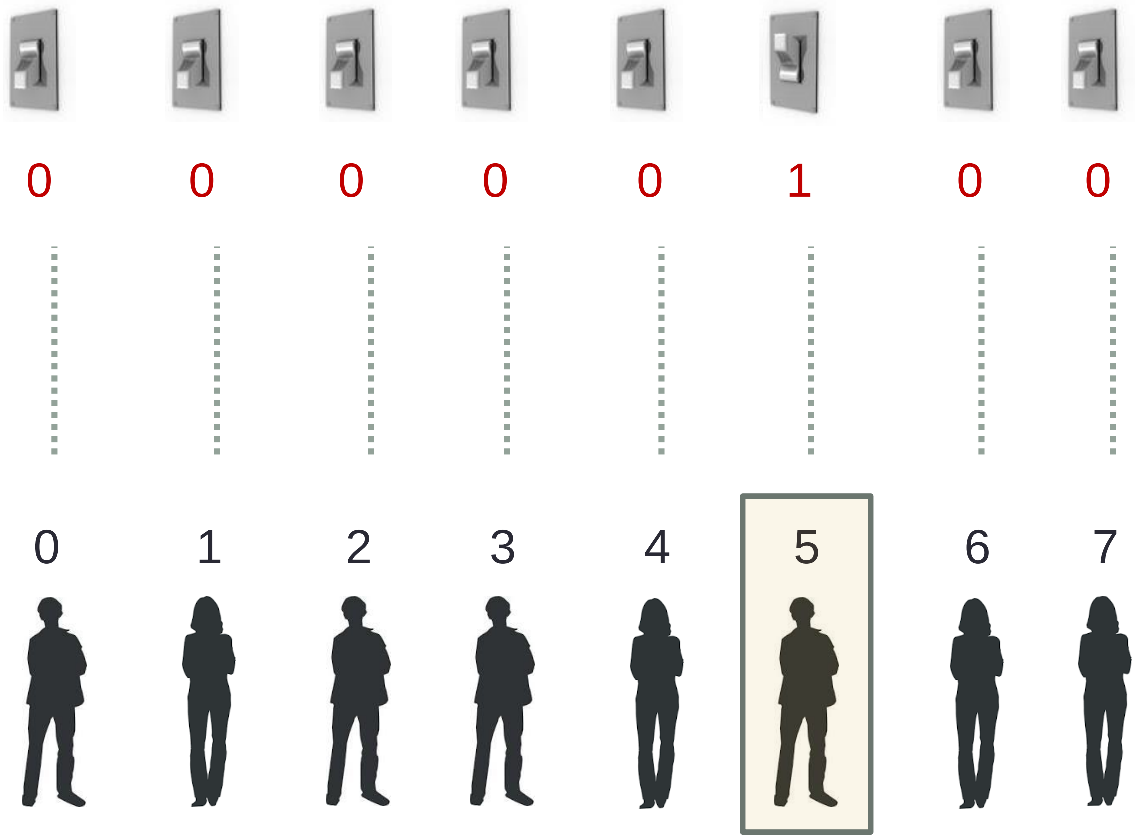
- Selected Past Years' Exam Questions
  - AY2016/17 Sem2 Q2
  - AY2017/18 Sem2 Q3(a)(b)(c)
  - AY2018/19 Sem2 Q5(c)
  - AY2019/20 Sem2 Q5(a)(b)

SUMMARY

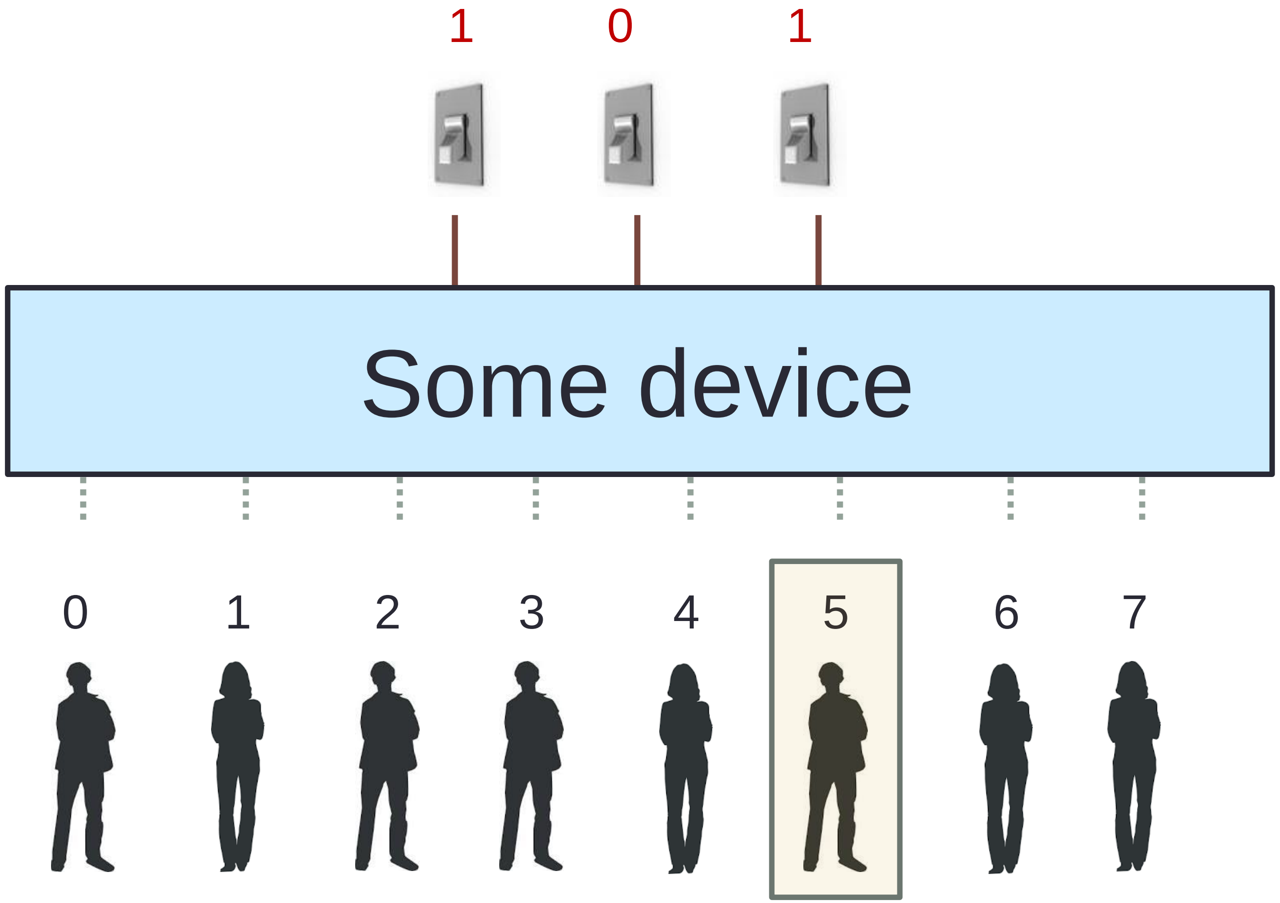
Decoders



Decoding – the inefficient way

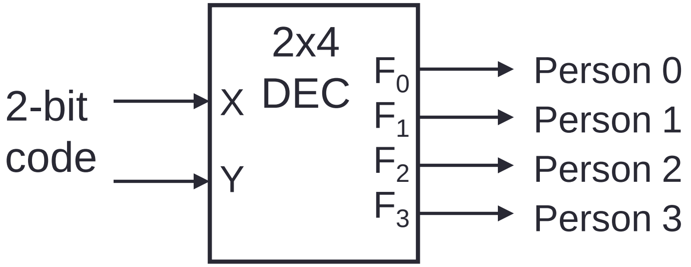


Decoding – the better way



2. Decoders (2/5)

- Example: If codes 00, 01, 10, 11 are used to identify four persons, we may use a 2-bit decoder.



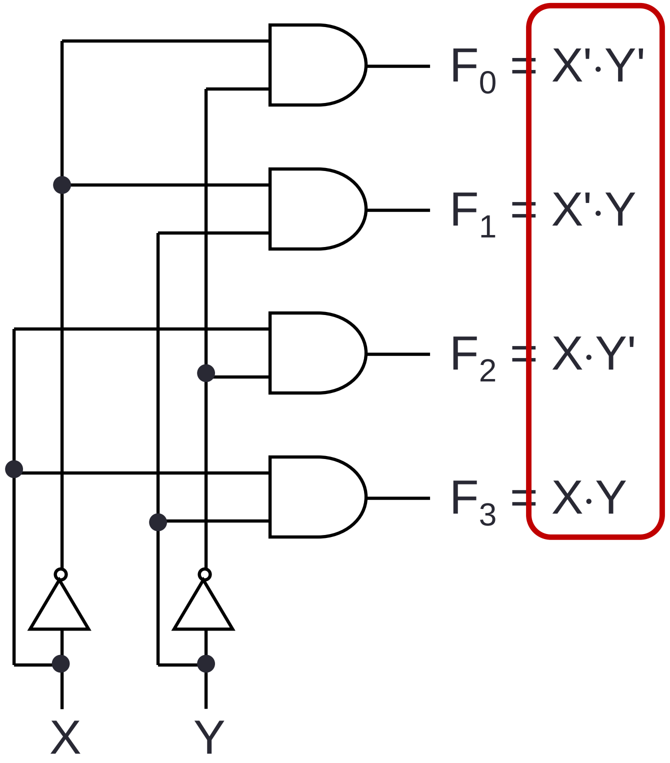
- This is a 2×4 decoder which selects an output line based on the 2-bit code supplied.
- Truth table:

X	Y	F_0	F_1	F_2	F_3
0	0	**1**	0	0	0
0	1	0	**1**	0	0
1	0	0	0	**1**	0
1	1	0	0	0	**1**

2. Decoders (3/5)

- From truth table, circuit for 2×4 decoder is:

X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

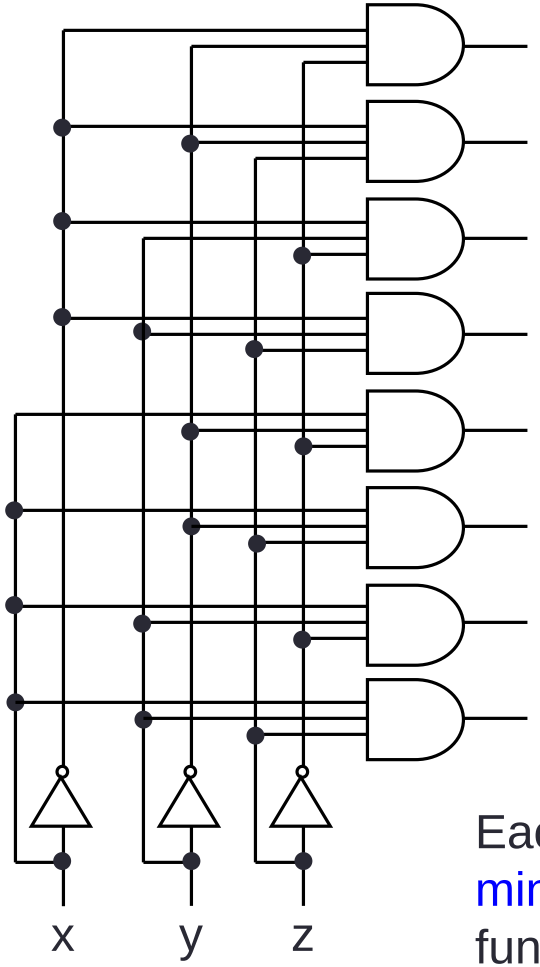


Each output is a minterm (X'·Y', X'·Y, X·Y' or X·Y) of a 2-variable function.

2. Decoders (4/5)

- Design a 3×8 decoder.

x	y	z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



$F_0 = x'\cdot y'\cdot z'$

$F_1 = x'\cdot y'\cdot z$

$F_2 = x'\cdot y\cdot z'$

$F_3 = x'\cdot y\cdot z$

$F_4 = x\cdot y'\cdot z'$

$F_5 = x\cdot y'\cdot z$

$F_6 = x\cdot y\cdot z'$

$F_7 = x\cdot y\cdot z$

Each output is a minterm of a 3-variable function.

2. Decoders: Implementing Functions (1/3)

- A Boolean function, in sum-of-minterms form ⇨
  - decoder to generate the minterms, and
  - an OR gate to form the sum.
- Any combinational circuit with *n* inputs and *m* outputs can be implemented with an *n*:2^n decoder with *m* OR gates.
- Good when circuit has many outputs, and each function is expressed with a few minterms.

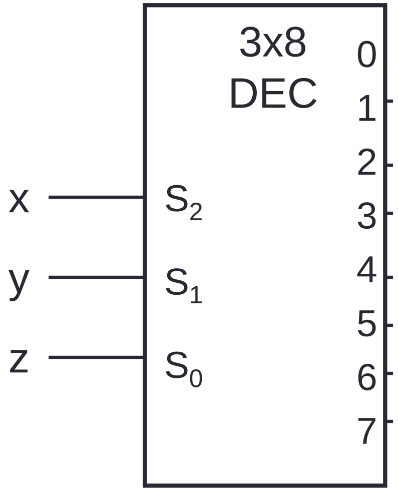
2. Decoders: Implementing Functions (2/3)

- Example: Full adder

  S(x, y, z) = Σ m(1,2,4,7)

  C(x, y, z) = Σ m(3,5,6,7)

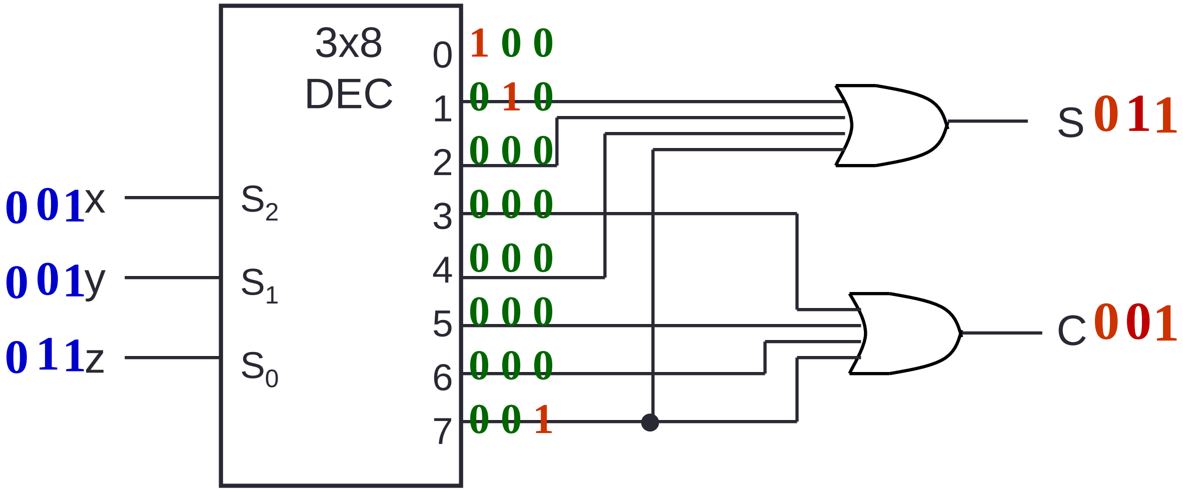
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



2. Decoders: Implementing Functions (3/3)

$S(x, y, z) = \Sigma\ m(1,2,4,7)$

$C(x, y, z) = \Sigma\ m(3,5,6,7)$



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

BRAVO!!!

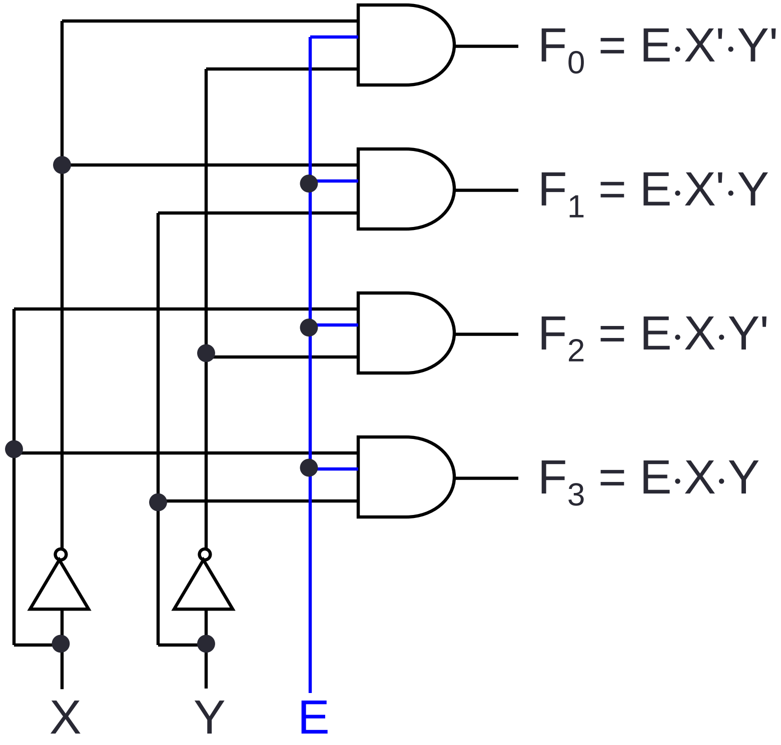
2. Decoders with Enable (1/2)

- Decoders often come with an *enable* control signal, so that the device is only activated when the enable, E = 1.

- Truth table:

E	X	Y	F_0	F_1	F_2	F_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	d	d	0	0	0	0

- Circuit of a 2×4 decoder with enable:


$F_0 = E{\cdot}X'{\cdot}Y'$
$F_1 = E{\cdot}X'{\cdot}Y$
$F_2 = E{\cdot}X{\cdot}Y'$
$F_3 = E{\cdot}X{\cdot}Y$
X
Y
E

2. Decoders with Enable (2/2)

- In the previous slide, the decoder has a one-enable control signal, i.e. the decoder is enabled with E=1.
- In most MSI decoders, enable signal is zero-enable, usually denoted by E' or Ē. The decoder is enabled when the signal is zero (low).

E	X	Y	F_0	F_1	F_2	F_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	d	d	0	0	0	0

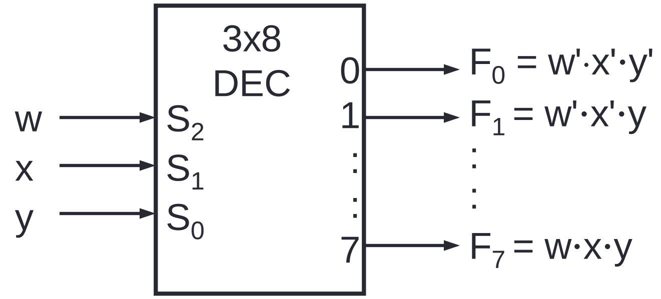
Decoder with 1-enable

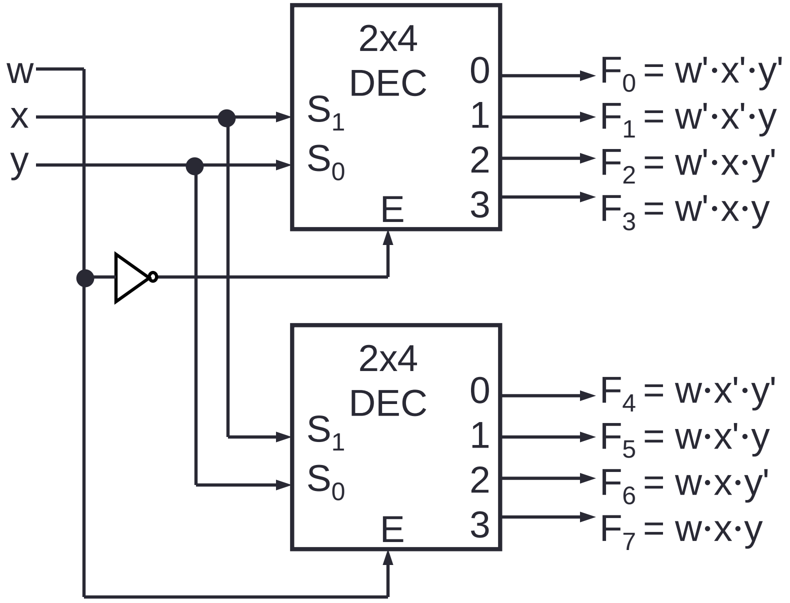
E'	X	Y	F_0	F_1	F_2	F_3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	d	d	0	0	0	0

Decoder with 0-enable

2. Constructing Larger Decoders (1/4)

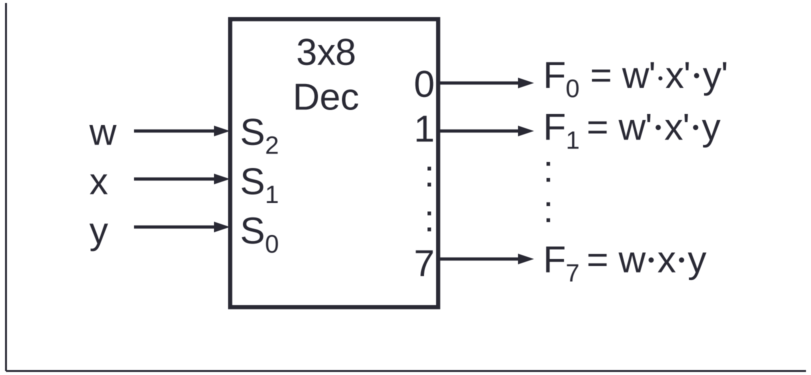
- Larger decoders can be constructed from smaller ones.
- Example: A 3×8 decoder can be built from two 2×4 decoders (with one-enable) and an inverter.

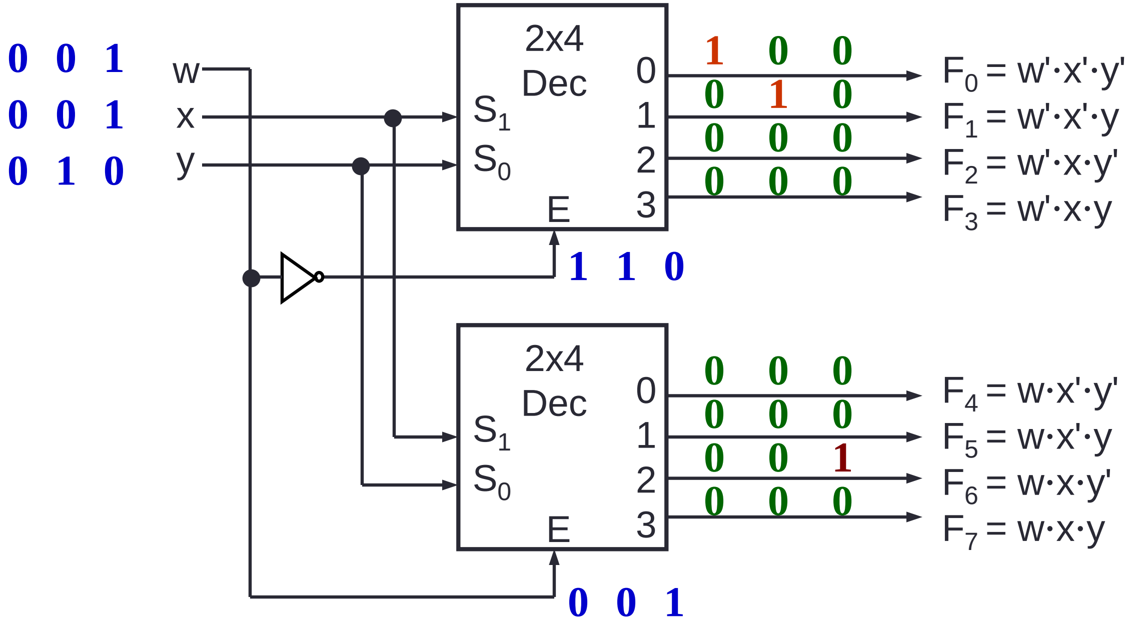

3x8
DEC
w
x
y
S_2
S_1
S_0
0
1
7
$F_0 = w' \cdot x' \cdot y'$
$F_1 = w' \cdot x' \cdot y$
$F_7 = w \cdot x \cdot y$


w
x
y
2x4
DEC
S_1
S_0
E
0
1
2
3
$F_0 = w' \cdot x' \cdot y'$
$F_1 = w' \cdot x' \cdot y$
$F_2 = w' \cdot x \cdot y'$
$F_3 = w' \cdot x \cdot y$
2x4
DEC
S_1
S_0
E
0
1
2
3
$F_4 = w \cdot x' \cdot y'$
$F_5 = w \cdot x' \cdot y$
$F_6 = w \cdot x \cdot y'$
$F_7 = w \cdot x \cdot y$

2. Constructing Larger Decoders (2/4)

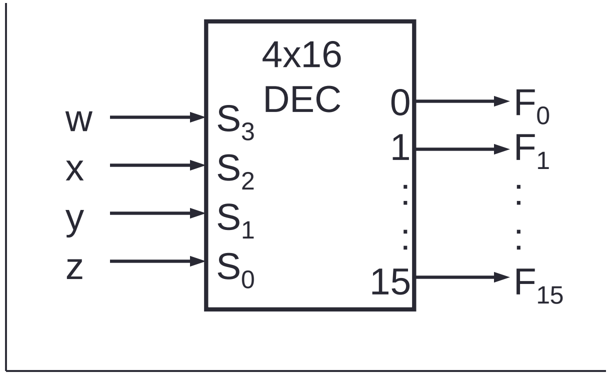
BRAVO!!!

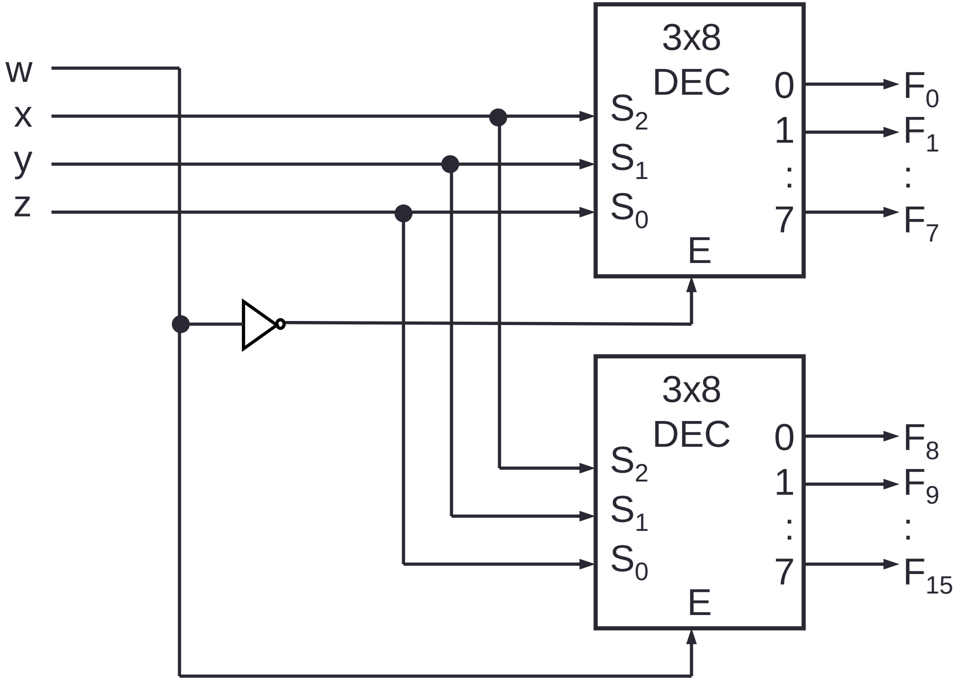

3x8
Dec
w
x
y
S_2
S_1
S_0
0
1
7
$F_0 = w'{\cdot}x'{\cdot}y'$
$F_1 = w'{\cdot}x'{\cdot}y$
$F_7 = w{\cdot}x{\cdot}y$


0 0 1
0 0 1
0 1 0
w
x
y
2x4
Dec
S_1
S_0
E
0
1
2
3
1 0 0
0 1 0
0 0 0
0 0 0
1 1 0
$F_0 = w'{\cdot}x'{\cdot}y'$
$F_1 = w'{\cdot}x'{\cdot}y$
$F_2 = w'{\cdot}x{\cdot}y'$
$F_3 = w'{\cdot}x{\cdot}y$
2x4
Dec
S_1
S_0
E
0
1
2
3
0 0 0
0 0 0
0 0 1
0 0 0
0 0 1
$F_4 = w{\cdot}x'{\cdot}y'$
$F_5 = w{\cdot}x'{\cdot}y$
$F_6 = w{\cdot}x{\cdot}y'$
$F_7 = w{\cdot}x{\cdot}y$

2. Constructing Larger Decoders (3/4)

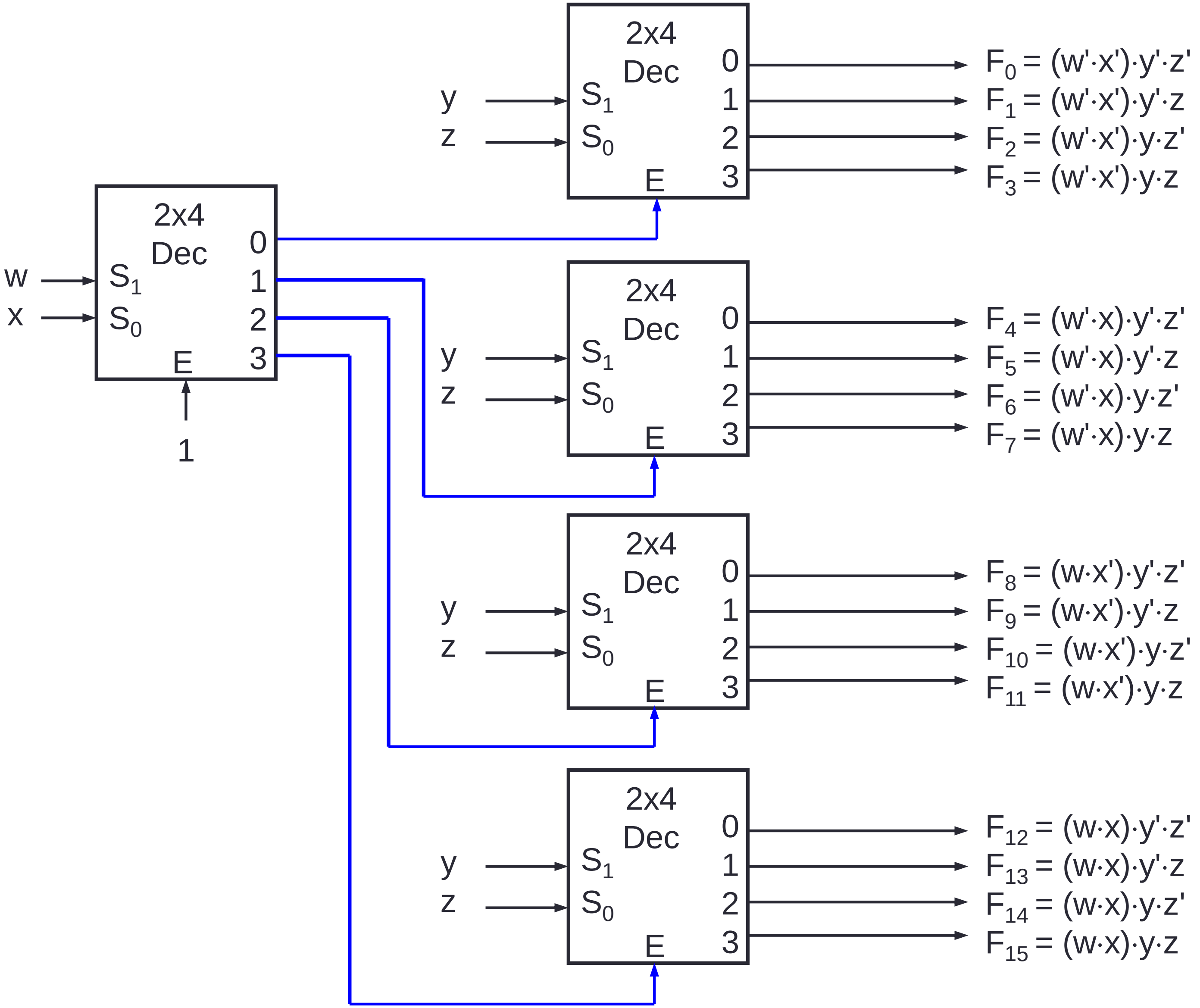
- Construct a 4×16 decoder from two 3×8 decoders with one-enable and an inverter.





- Note: The input w and its complement w' are used to select either one of the two smaller decoders.

Exercise: How to construct a 4×16 decoder using five 2×4 decoders with enable?


2x4 Dec
w
x
S_1
S_0
E
1
0
1
2
3
y
z
$F_0 = (w'\cdot x')\cdot y'\cdot z'$
$F_1 = (w'\cdot x')\cdot y'\cdot z$
$F_2 = (w'\cdot x')\cdot y\cdot z'$
$F_3 = (w'\cdot x')\cdot y\cdot z$
$F_4 = (w'\cdot x)\cdot y'\cdot z'$
$F_5 = (w'\cdot x)\cdot y'\cdot z$
$F_6 = (w'\cdot x)\cdot y\cdot z'$
$F_7 = (w'\cdot x)\cdot y\cdot z$
$F_8 = (w\cdot x')\cdot y'\cdot z'$
$F_9 = (w\cdot x')\cdot y'\cdot z$
$F_{10} = (w\cdot x')\cdot y\cdot z'$
$F_{11} = (w\cdot x')\cdot y\cdot z$
$F_{12} = (w\cdot x)\cdot y'\cdot z'$
$F_{13} = (w\cdot x)\cdot y'\cdot z$
$F_{14} = (w\cdot x)\cdot y\cdot z'$
$F_{15} = (w\cdot x)\cdot y\cdot z$

Negated outputs (active low outputs) decoders

- A decoder may be 1-enabled or 0-enabled.
- A decoder may also normal output or negated outputs. (See next two slides.)
    - Normal outputs = active high outputs
    - Negated outputs = active low outputs

E	X	Y	F_0	F_1	F_2	F_3
1	0	0	**1**	0	0	0
1	0	1	0	**1**	0	0
1	1	0	0	0	**1**	0
1	1	1	0	0	0	**1**
0	d	d	0	0	0	0

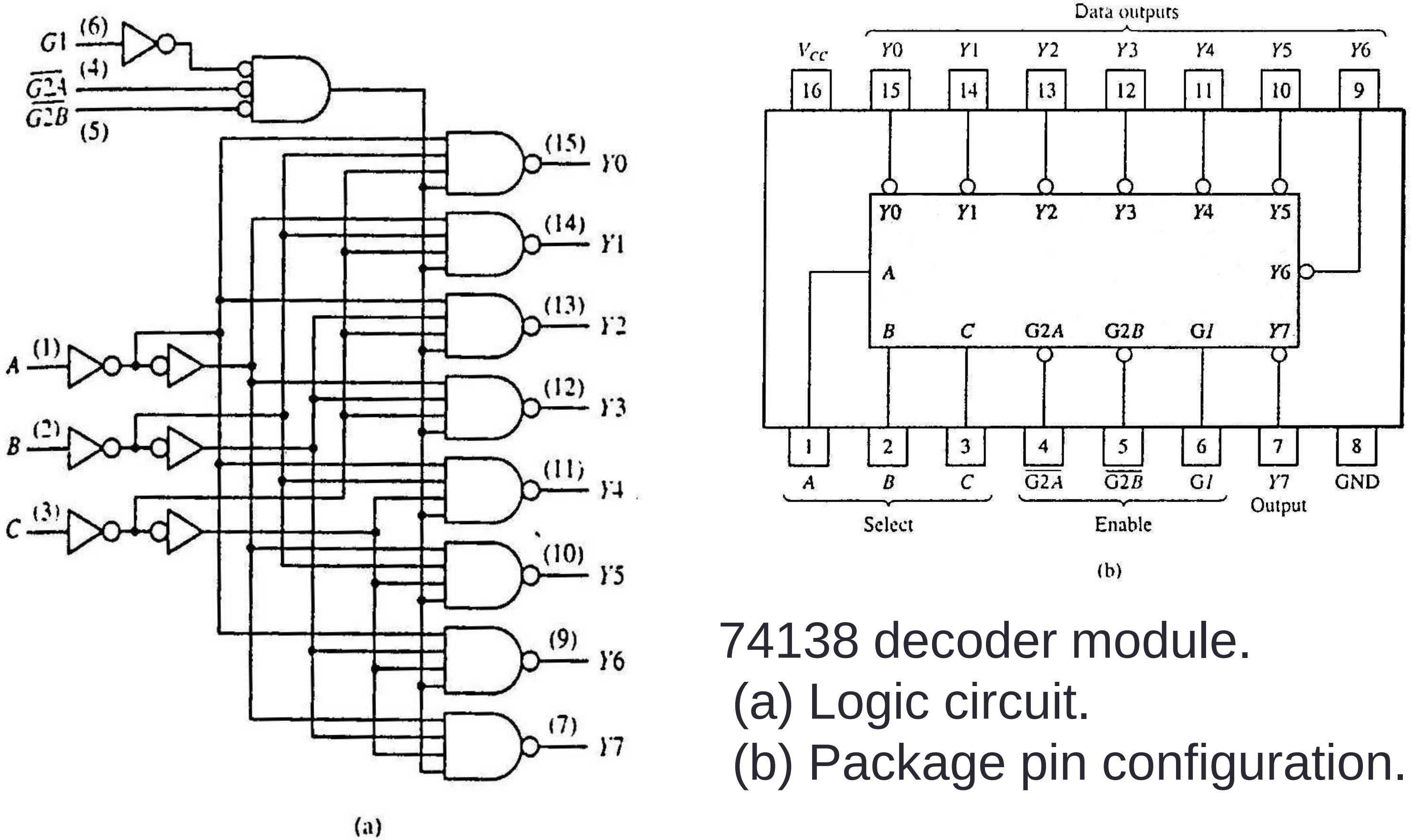
1-enabled, active high outputs 2x4 decoder.

E	X	Y	F_0	F_1	F_2	F_3
1	0	0	**0**	1	1	1
1	0	1	1	**0**	1	1
1	1	0	1	1	**0**	1
1	1	1	1	1	1	**0**
0	d	d	1	1	1	1

1-enabled, active low outputs 2x4 decoder.

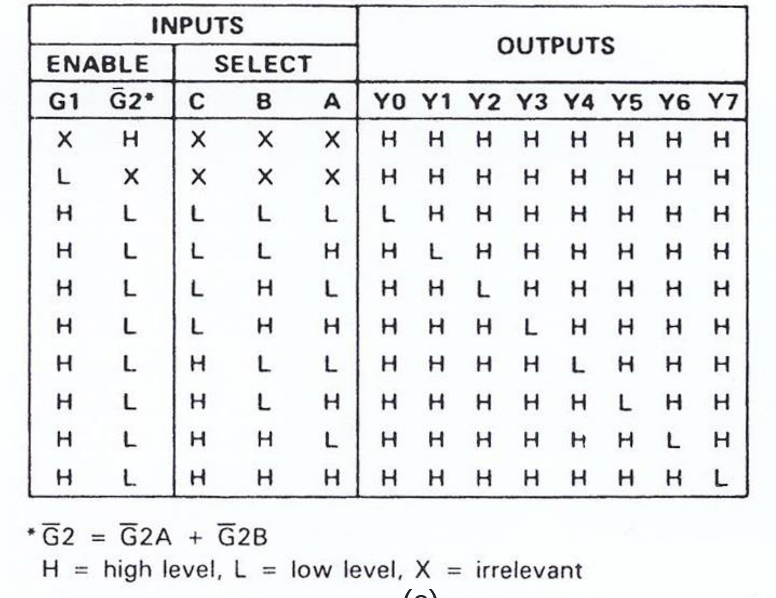
2. Standard MSI Decoder (1/2)

- 74138 (3-to-8 decoder)



74138 decoder module.
(a) Logic circuit.
(b) Package pin configuration.

2. Standard MSI Decoder (2/2)



INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	$\overline{G}2$*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

*$\overline{G}2 = \overline{G}2A + \overline{G}2B$

H = high level, L = low level, X = irrelevant

(c)

74138 decoder module.

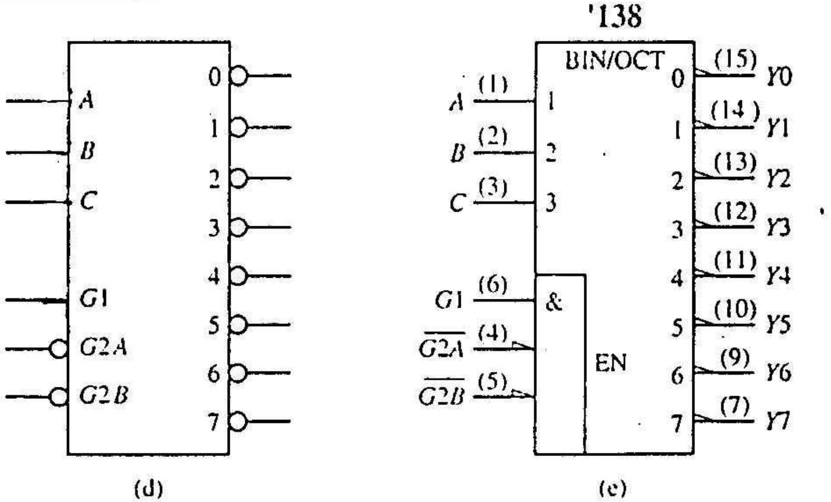
(c) Function table.

74138 decoder module.

(d) Generic symbol.

(e) IEEE standard logic symbol.

Source:The Data Book Volume 2, Texas Instruments Inc.,1985



(d) (e)

2. Decoders: Implementing Functions Revisit (1/2)

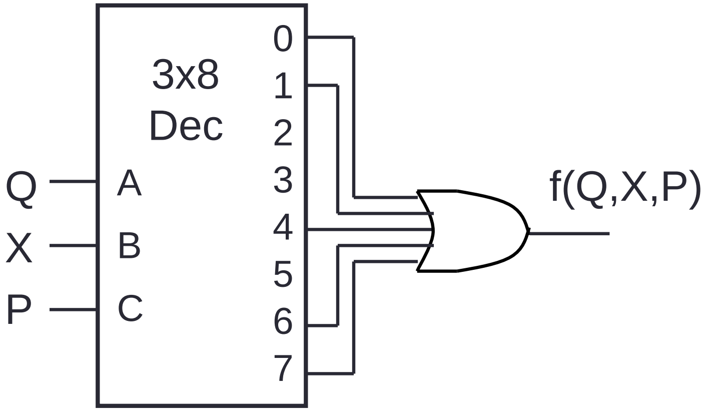
- Example: Implement the following function using a 3×8 decoder and an appropriate logic gate

$$f(Q,X,P) = \sum m(0,1,4,6,7) = \prod M(2,3,5)$$

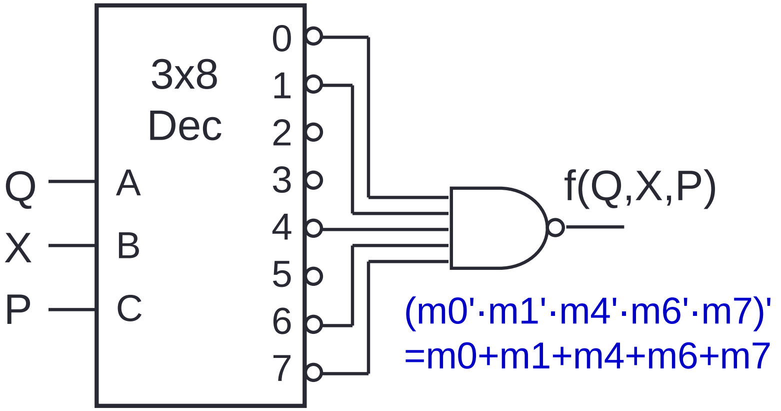
- We may implement the function in several ways:
  - Using a decoder with active-high outputs with an OR gate:
    $f(Q,X,P) = m_0 + m_1 + m_4 + m_6 + m_7$
  - Using a decoder with active-low outputs with a NAND gate:
    $f(Q,X,P) = (m_0' \cdot m_1' \cdot m_4' \cdot m_6' \cdot m_7' \ )'$
  - Using a decoder with active-high outputs with a NOR gate:
    $f(Q,X,P) = (m_2 + m_3 + m_5)' \ [= M_2 \cdot M_3 \cdot M_5]$
  - Using a decoder with active-low outputs with an AND gate:
    $f(Q,X,P) = m_2' \cdot m_3' \cdot m_5'$

2. Decoders: Implementing Functions Revisit (2/2)

$f(Q,X,P) = \Sigma m(0,1,4,6,7) = \prod M(2,3,5)$

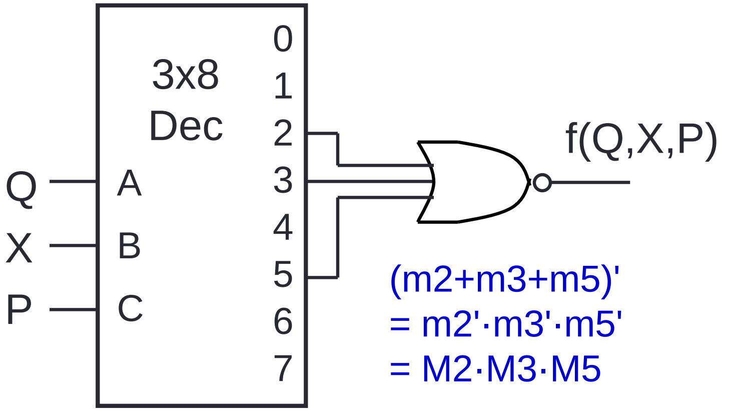


(a) Active-high decoder with OR gate.



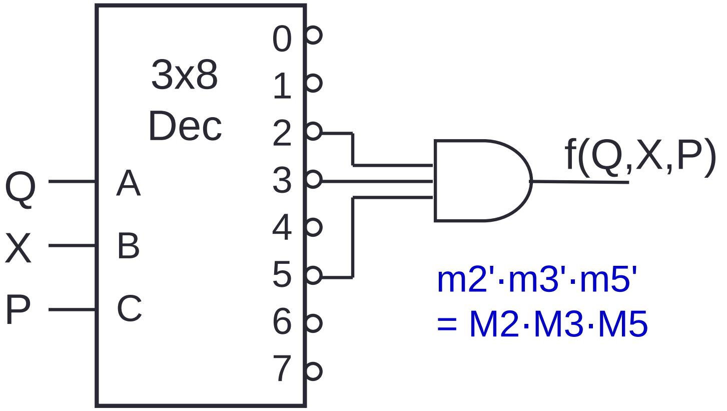
(m0'·m1'·m4'·m6'·m7)'
=m0+m1+m4+m6+m7

(b) Active-low decoder with NAND gate.



(m2+m3+m5)'
= m2'·m3'·m5'
= M2·M3·M5

(c) Active-high decoder with NOR gate.

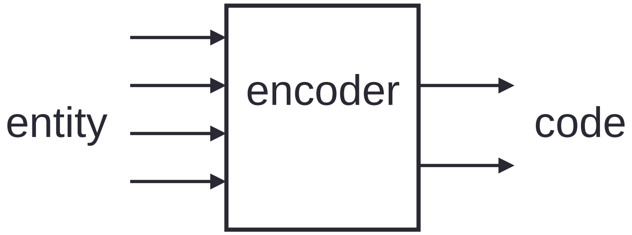


m2'·m3'·m5'
= M2·M3·M5

(d) Active-low decoder with AND gate.

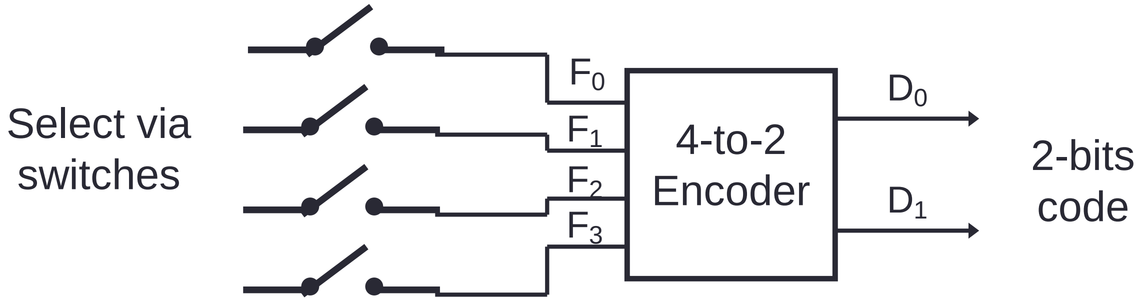
SUMMARY

Encoders

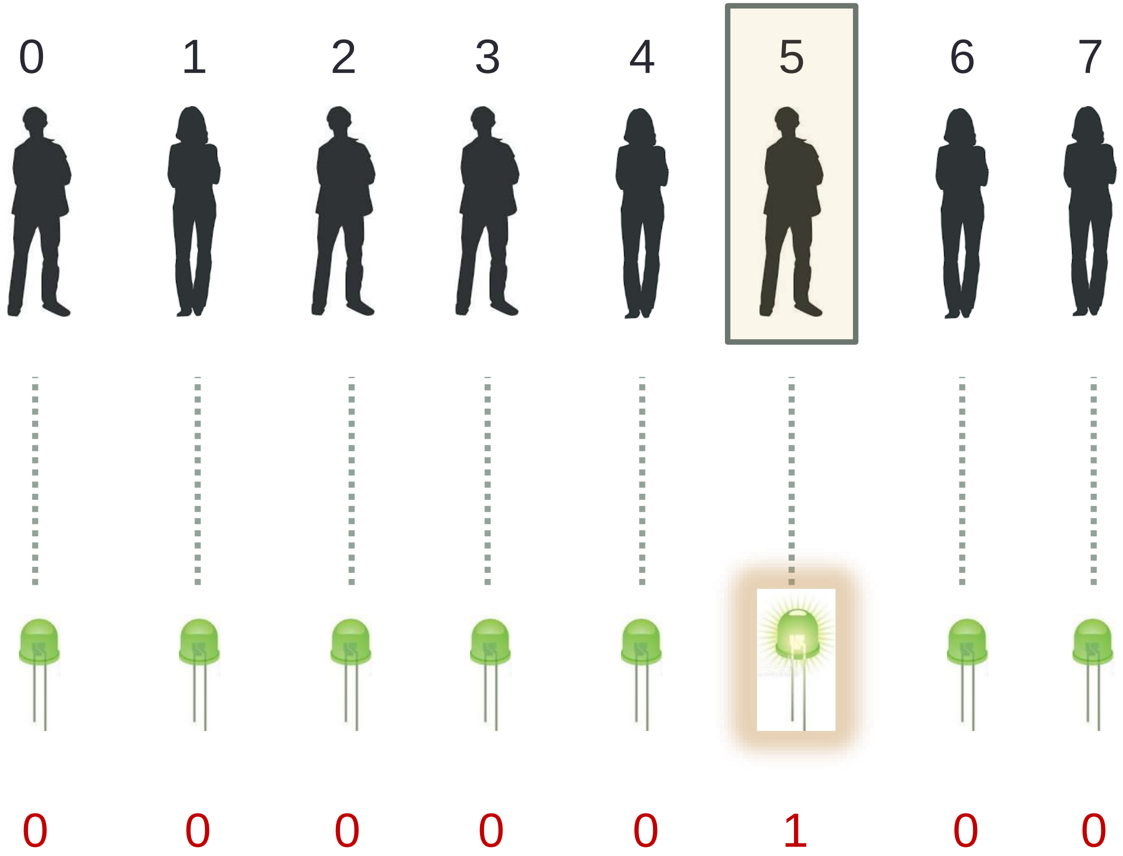


3. Encoders (1/4)

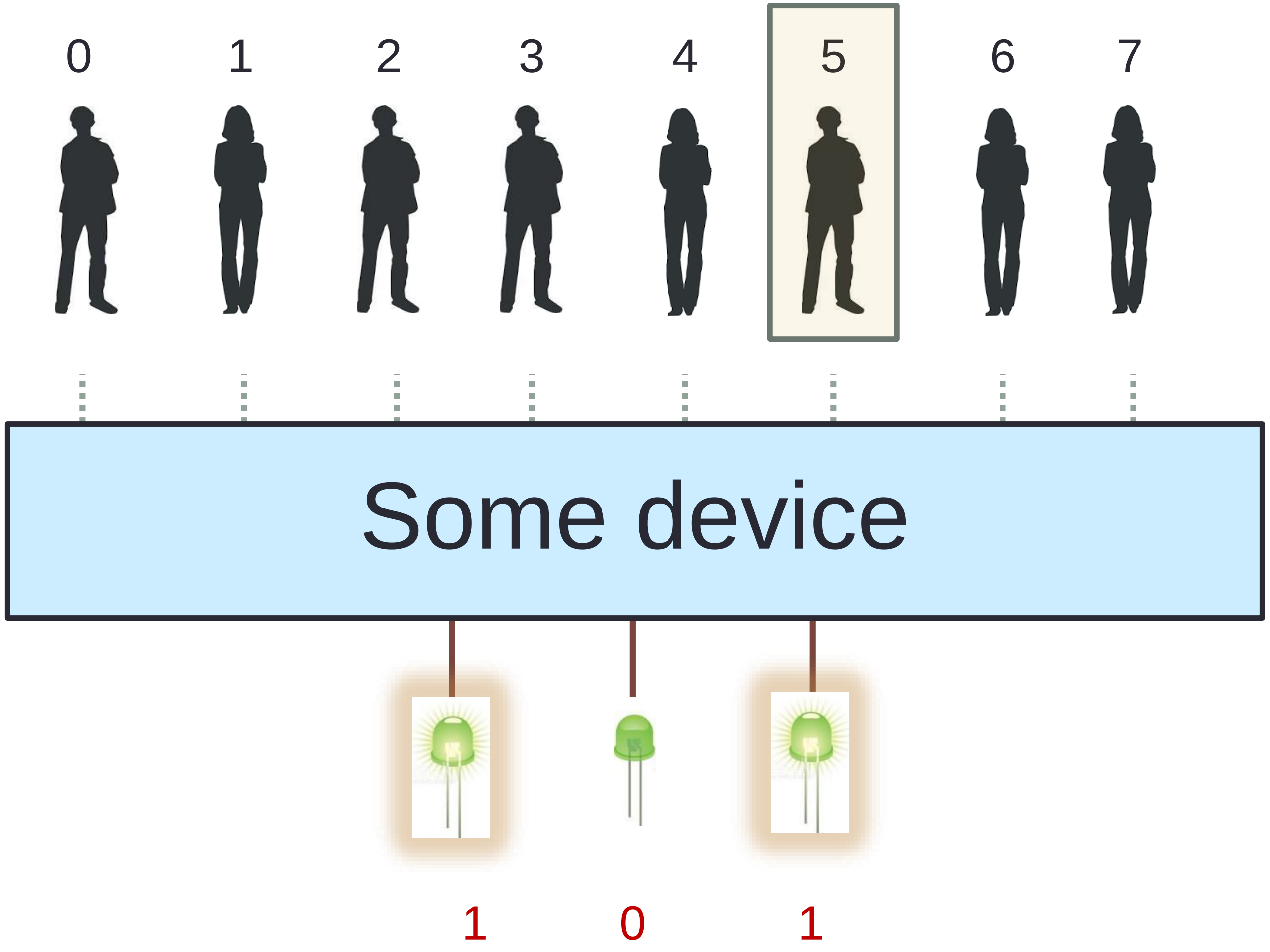
- Encoding is the converse of decoding.
- Given a set of input lines, of which exactly one is high and the rest are low, the encoder provides a code that corresponds to that high input line.
- Contains 2^n (or fewer) input lines and n output lines.
- Implemented with OR gates.
- Example:



Encoding – the inefficient way



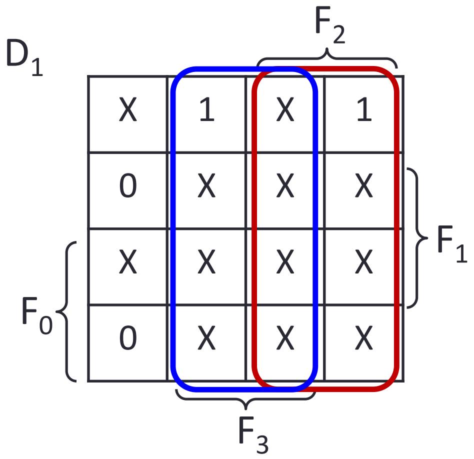
Encoding – the better way

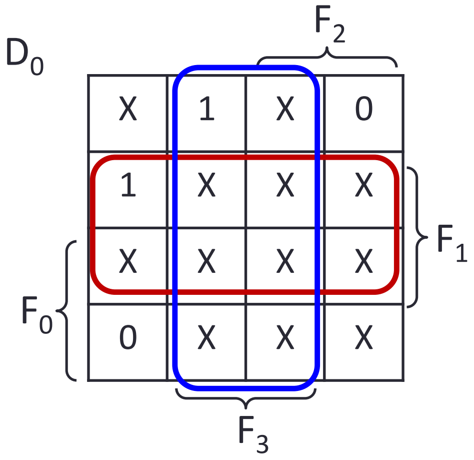


3. Encoders (2/4)

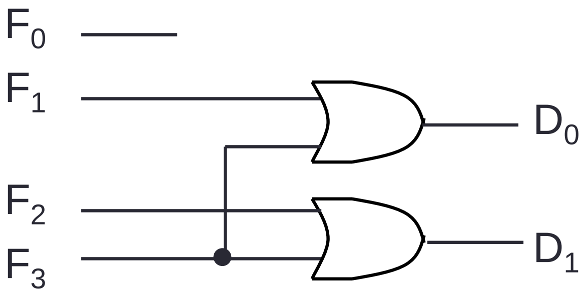
- Truth table:

F_0	F_1	F_2	F_3	D_1	D_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1
0	0	0	0	X	X
0	0	1	1	X	X
0	1	0	1	X	X
0	1	1	0	X	X
0	1	1	1	X	X
1	0	0	1	X	X
1	0	1	0	X	X
1	0	1	1	X	X
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	X





- With K-map, we obtain:
  - $D_1 = F_2 + F_3$
  - $D_0 = F_1 + F_3$
- Circuit:



Simple 4-to-2 encoder

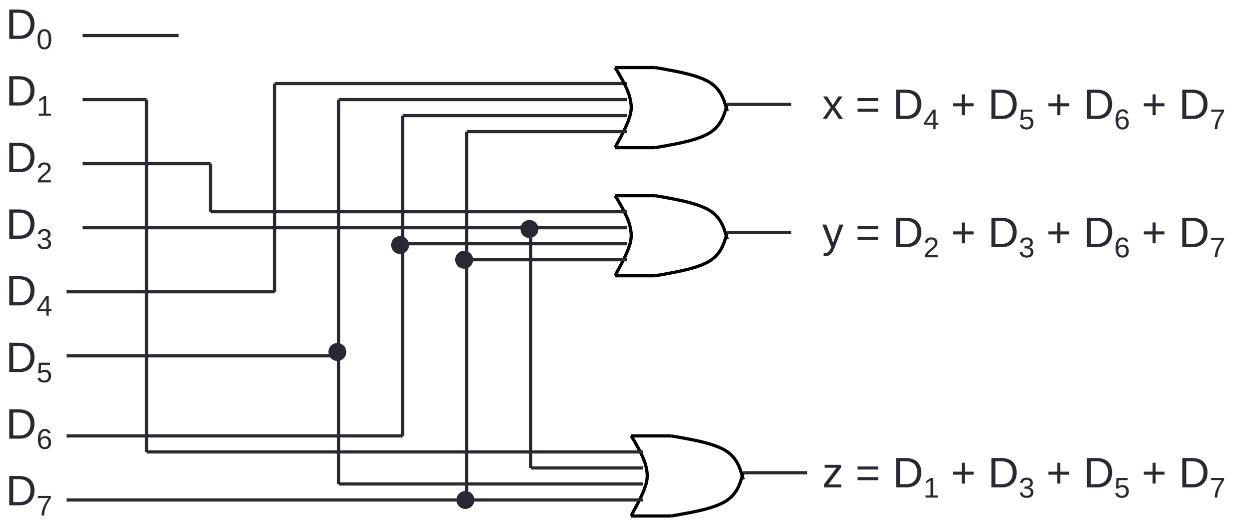
3. Encoders (3/4)

- Example: 8-to-3 encoder.
  - At any one time, only one input line of an encoder has a value of 1 (high), the rest are zeroes (low).

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

3. Encoders (4/4)

- Example: 8-to-3 encoder.


D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
$x = D_4 + D_5 + D_6 + D_7$
$y = D_2 + D_3 + D_6 + D_7$
$z = D_1 + D_3 + D_5 + D_7$

An 8-to-3 encoder

- Exercise: Can you design a 2^n-to-n encoder without using K-map?

3. Priority Encoders (1/2)

- A priority encoder is one with priority
  - To allow for more than one input line to carry a 1.
  - If two or more inputs or equal to 1, the input with the highest priority takes precedence.
  - If all inputs are 0, this input combination is considered invalid.
- Example of a 4-to-2 priority encoder:

Inputs				Outputs		
D_0	D_1	D_2	D_3	f	g	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

3. Priority Encoders (2/2)

- Understanding “compact” function table

Inputs				Outputs		
D_0	D_1	D_2	D_3	*f*	*g*	*V*
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

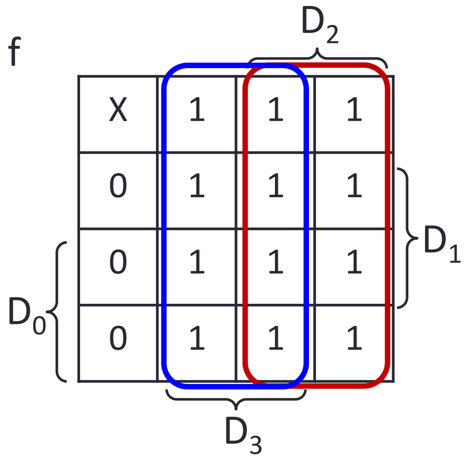
Inputs				Outputs		
D_0	D_1	D_2	D_3	*f*	*g*	*V*
0	0	0	0	X	X	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0	0	1	1
0	0	1	0	1	0	1
0	1	1	0	1	0	1
1	0	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

- Exercise: Obtain the simplified expressions for *f*, *g* and *V*.

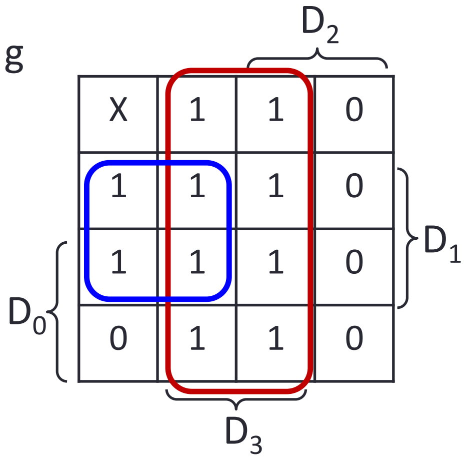
Hi Prof, please go through priority encoder and how to map out the expression using kmap. Slide 34 lect 18.

- Exercise: Obtain the simplified expressions for *f*, *g* and *V*.

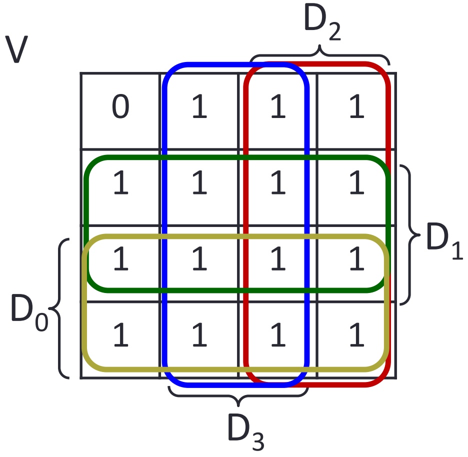
Inputs				Outputs		
D_0	D_1	D_2	D_3	*f*	*g*	*V*
0	0	0	0	X	X	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0	0	1	1
0	0	1	0	1	0	1
0	1	1	0	1	0	1
1	0	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1



X	1	1	1
0	1	1	1
0	1	1	1
0	1	1	1



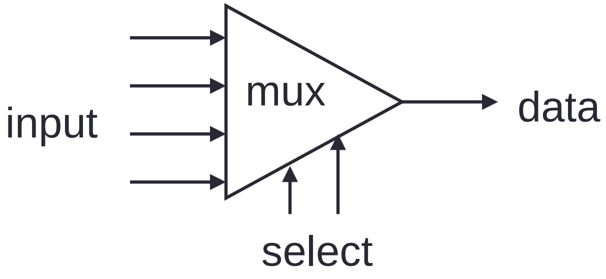
X	1	1	0
1	1	1	0
1	1	1	0
0	1	1	0

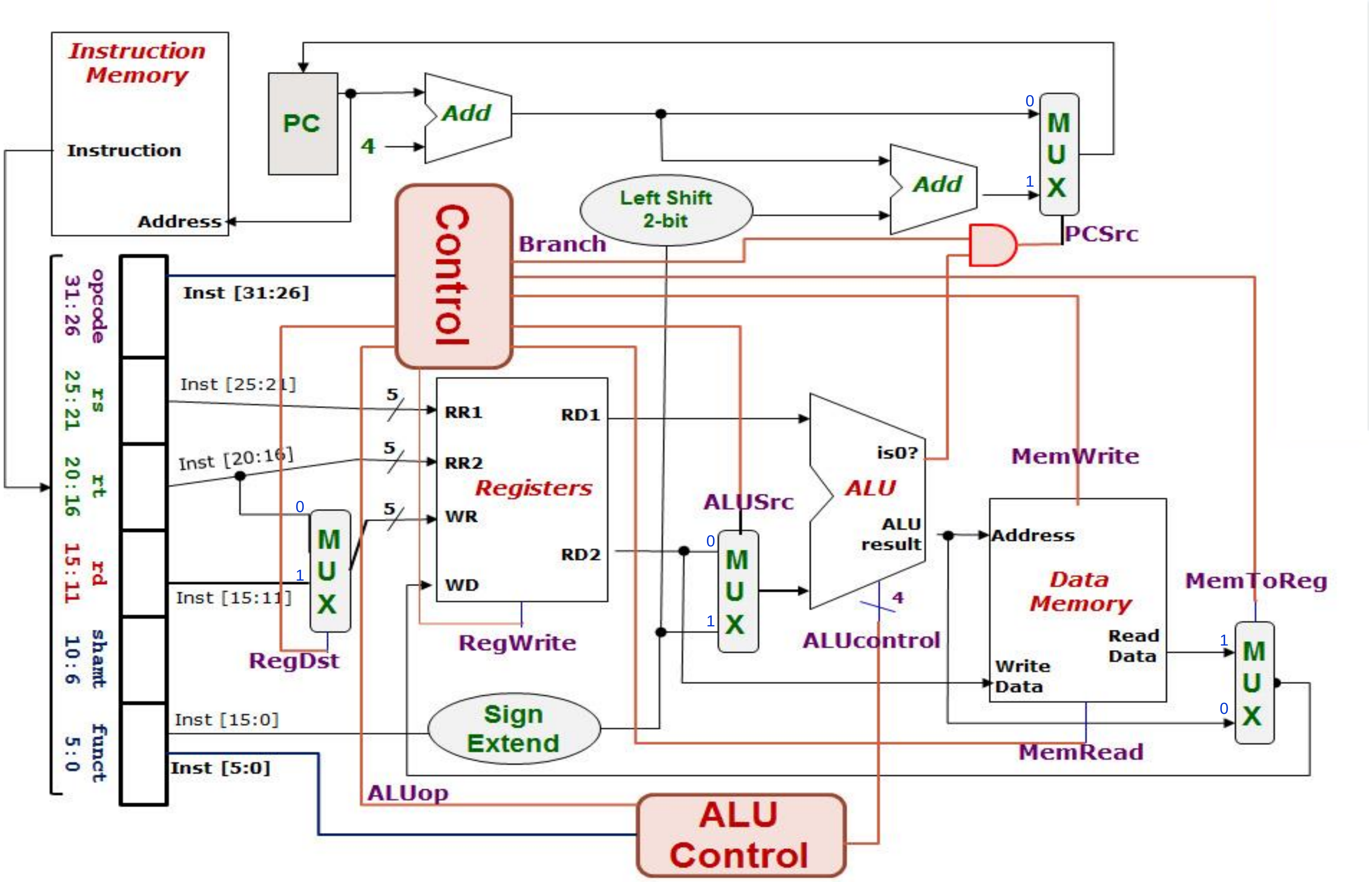


0	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

SUMMARY

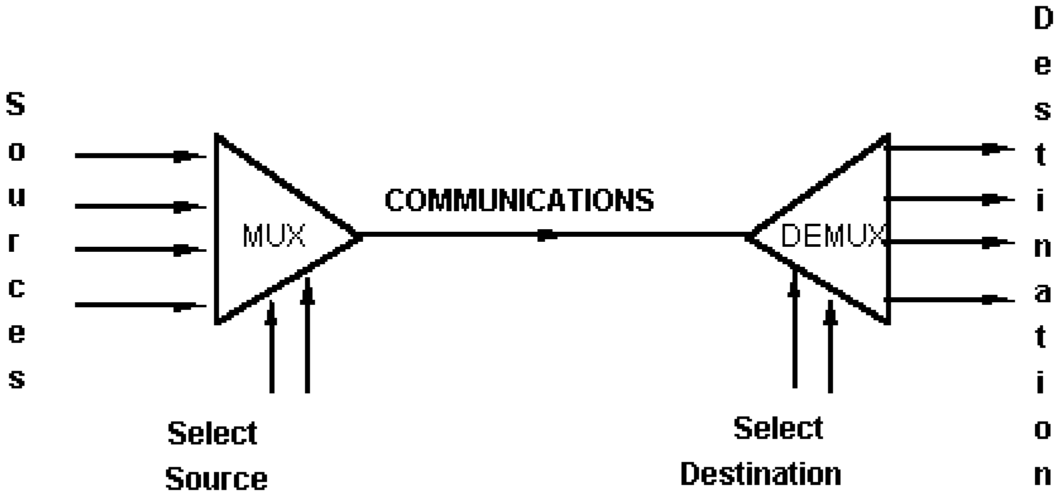
Multiplexers




Instruction Memory
Instruction
Address
PC
4
Add
Add
Left Shift 2-bit
MUX
0
1
PCSrc
Control
Branch
opcode 31:26
rs 25:21
rt 20:16
rd 15:11
shamt 10:6
funct 5:0
Inst [31:26]
Inst [25:21]
Inst [20:16]
Inst [15:11]
Inst [15:0]
Inst [5:0]
5
5
5
RR1
RR2
WR
WD
RD1
RD2
Registers
RegWrite
RegDst
ALUSrc
ALU
is0?
ALU result
4
ALUcontrol
MemWrite
Address
Data Memory
Write Data
Read Data
MemRead
MemToReg
Sign Extend
ALUop
ALU Control

Multiplexers and Demultiplexers

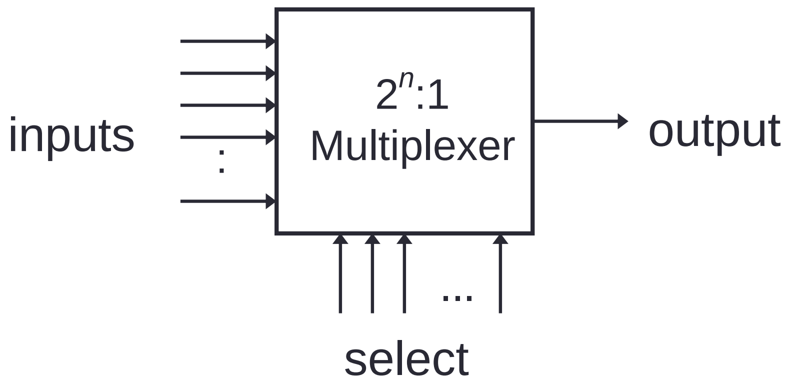
- An application:



- Helps share a *single communication line* among a number of devices.
- At any time, only one source and one destination can use the communication line.

5. Multiplexers (1/4)

- A multiplexer is a device that has
  - A number of input lines
  - A number of selection lines
  - One output line
- It steers one of 2^n inputs to a single output line, using *n* selection lines. Also known as a *data selector*.

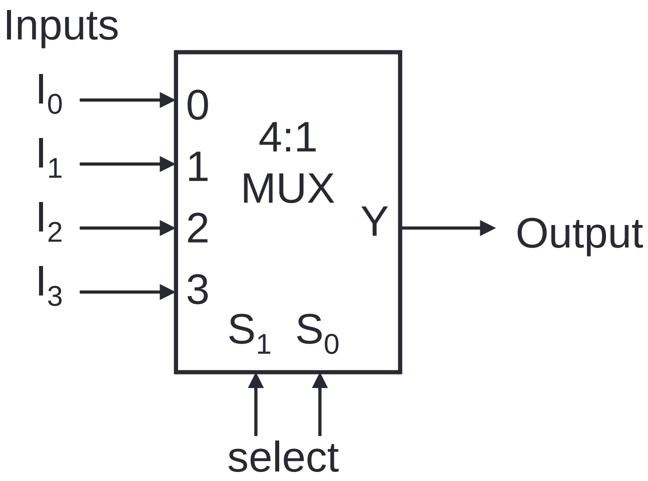


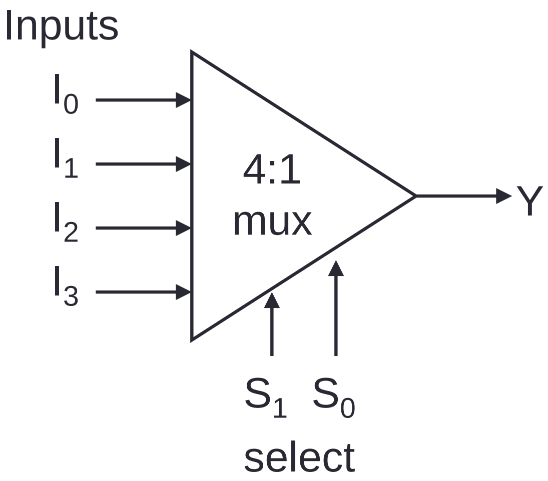
5. Multiplexers (2/4)

- Truth table for a 4-to-1 multiplexer:

I_0	I_1	I_2	I_3	S_1	S_0	Y
d_0	d_1	d_2	d_3	0	0	d_0
d_0	d_1	d_2	d_3	0	1	d_1
d_0	d_1	d_2	d_3	1	0	d_2
d_0	d_1	d_2	d_3	1	1	d_3

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3





5. Multiplexers (3/4)

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- Output of multiplexer is

  "sum of the (product of *data lines* and *selection lines*)"

- Example: Output of a 4-to-1 multiplexer is:

$$Y = I_0\cdot(S_1'\cdot S_0') + I_1\cdot(S_1'\cdot S_0) + I_2\cdot(S_1\cdot S_0') + I_3\cdot(S_1\cdot S_0)$$

Note:

Expressing

$$I_0\cdot(S_1'\cdot S_0') + I_1\cdot(S_1'\cdot S_0) + I_2\cdot(S_1\cdot S_0') + I_3\cdot(S_1\cdot S_0)$$

in minterms notation, it is equal to

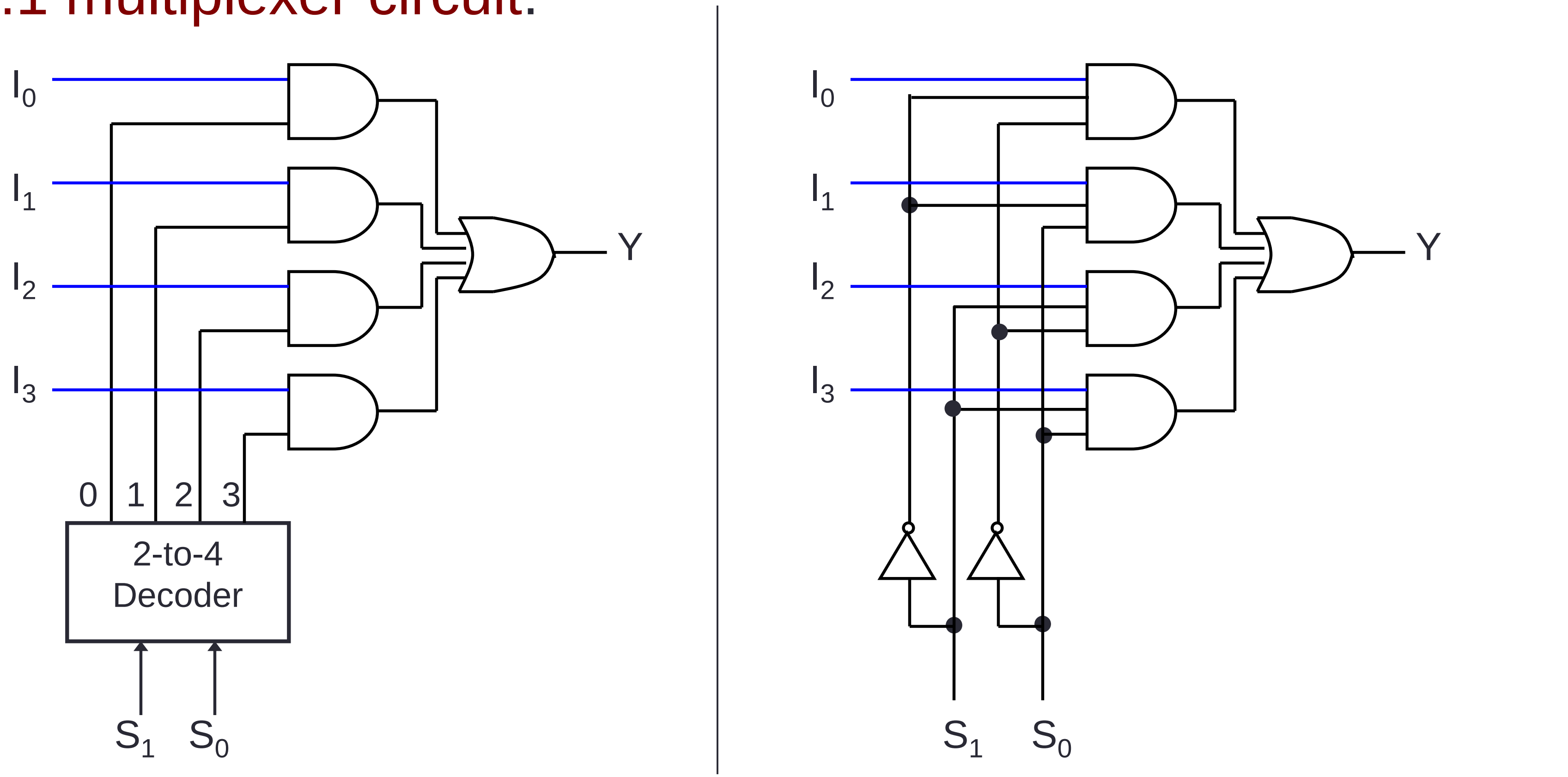
$$I_0\cdot m_0 + I_1\cdot m_1 + I_2\cdot m_2 + I_3\cdot m_3$$

This is useful later (eg: slide 45).

5. Multiplexers (4/4)

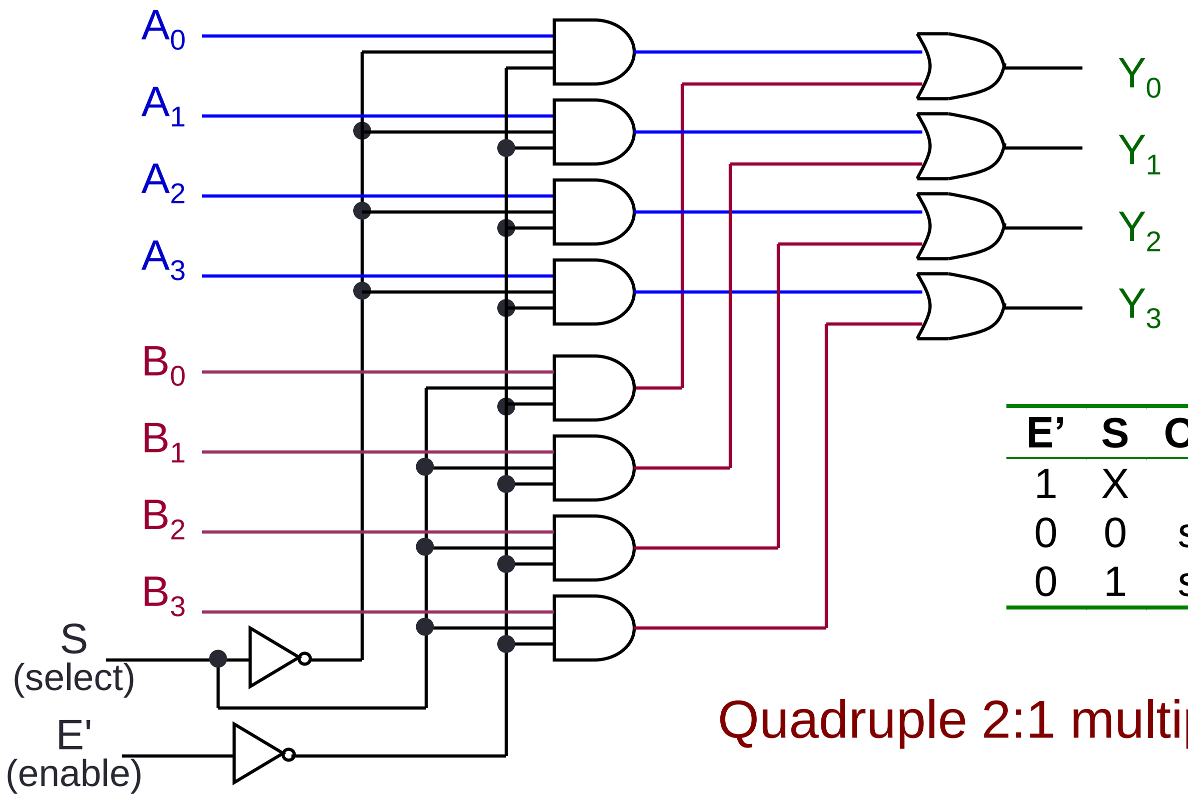
- A 2^n-to-1-line multiplexer, or simply 2^n:1 MUX, is made from an n:2^n decoder by adding to it 2^n input lines, one to each AND gate.

- A 4:1 multiplexer circuit:



5. Multiplexer IC Package

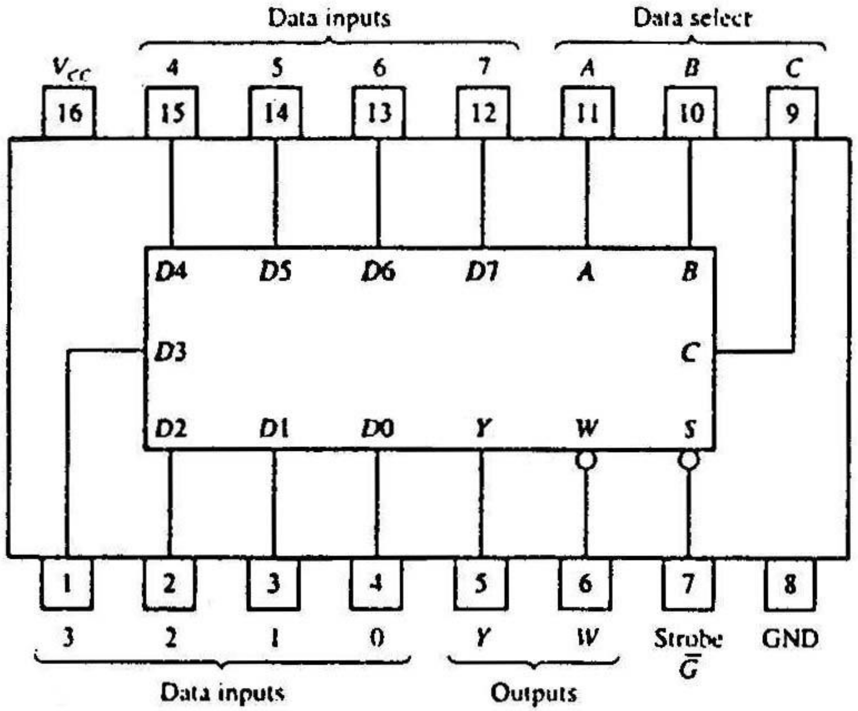
- Some IC packages have a few multiplexers in each package (chip). The selection and enable inputs are common to all multiplexers within the package.



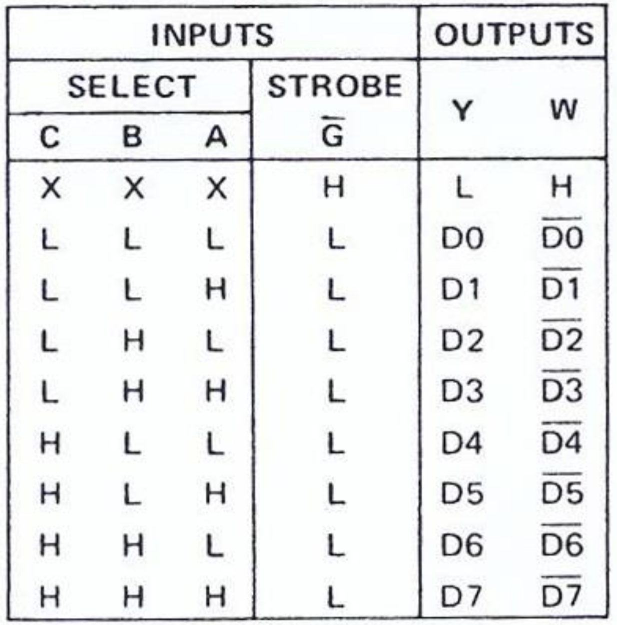
E’	S	Output Y
1	X	all 0’s
0	0	select A
0	1	select B

Quadruple 2:1 multiplexer

5. Standard MSI Multiplexer (1/2)



(a)

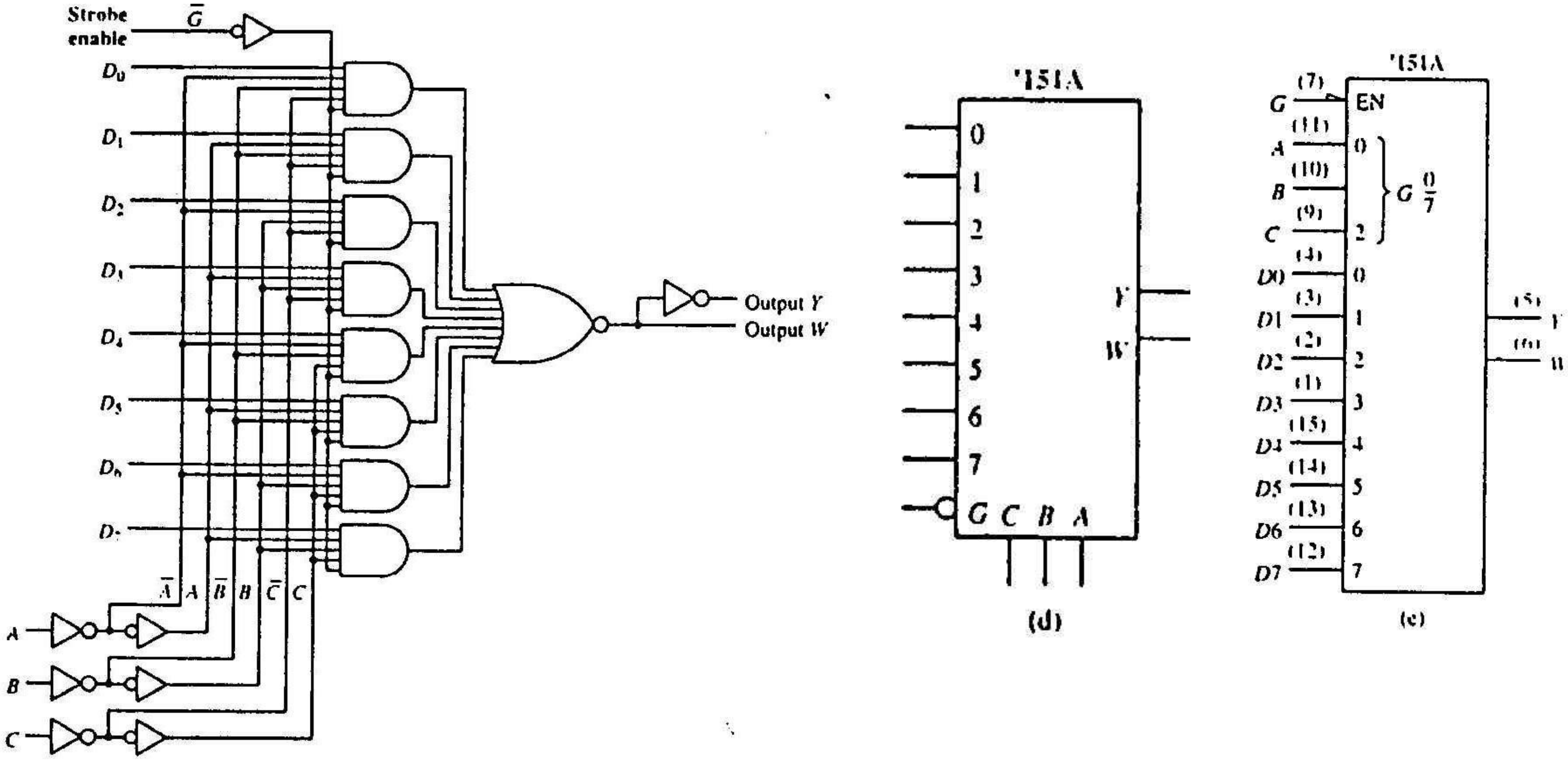


INPUTS				OUTPUTS	
SELECT			STROBE	Y	W
C	B	A	$\overline{G}$		
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

(b)

74151A 8-to-1 multiplexer. (a) Package configuration. (b) Function table.

5. Standard MSI Multiplexer (2/2)



(c)

74151A 8-to-1 multiplexer. (c) Logic diagram. (d) Generic logic symbol. (e) IEEE standard logic symbol.

Source: The TTL Data Book Volume 2. Texas Instruments Inc.,1985.

5. Multiplexers: Implementing Functions (1/3)

- Boolean functions can be implemented using multiplexers.
- A 2^n-to-1 multiplexer can implement a Boolean function of *n* input variables, as follows:

1. Express in sum-of-minterms form.

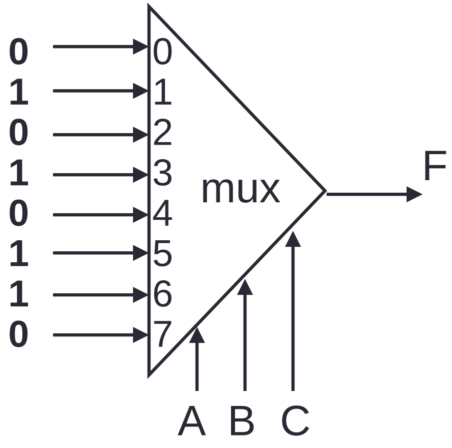
   Example:

   $$\begin{aligned} F(A,B,C) &= A' \cdot B' \cdot C + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C' \\ &= \Sigma\, m(1,3,5,6) \end{aligned}$$

2. Connect *n* variables to the *n* selection lines.
3. Put a ‘1’ on a data line if it is a minterm of the function, or ‘0’ otherwise.

5. Multiplexers: Implementing Functions (2/3)

- $F(A,B,C) = \Sigma\, m(1,3,5,6)$



This method works because:

$$\text{Output} = I_0 \cdot m_0 + I_1 \cdot m_1 + I_2 \cdot m_2 + I_3 \cdot m_3 + I_4 \cdot m_4 + I_5 \cdot m_5 + I_6 \cdot m_6 + I_7 \cdot m_7$$

Supplying ‘1’ to I_1, I_3, I_5, I_6 , and ‘0’ to the rest:

$$\text{Output} = m_1 + m_3 + m_5 + m_6$$

From slide 34 (4:1 mux)

Expressing

$$I_0 \cdot (S_1' \cdot S_0') + I_1 \cdot (S_1' \cdot S_0) + I_2 \cdot (S_1 \cdot S_0') + I_3 \cdot (S_1 \cdot S_0)$$

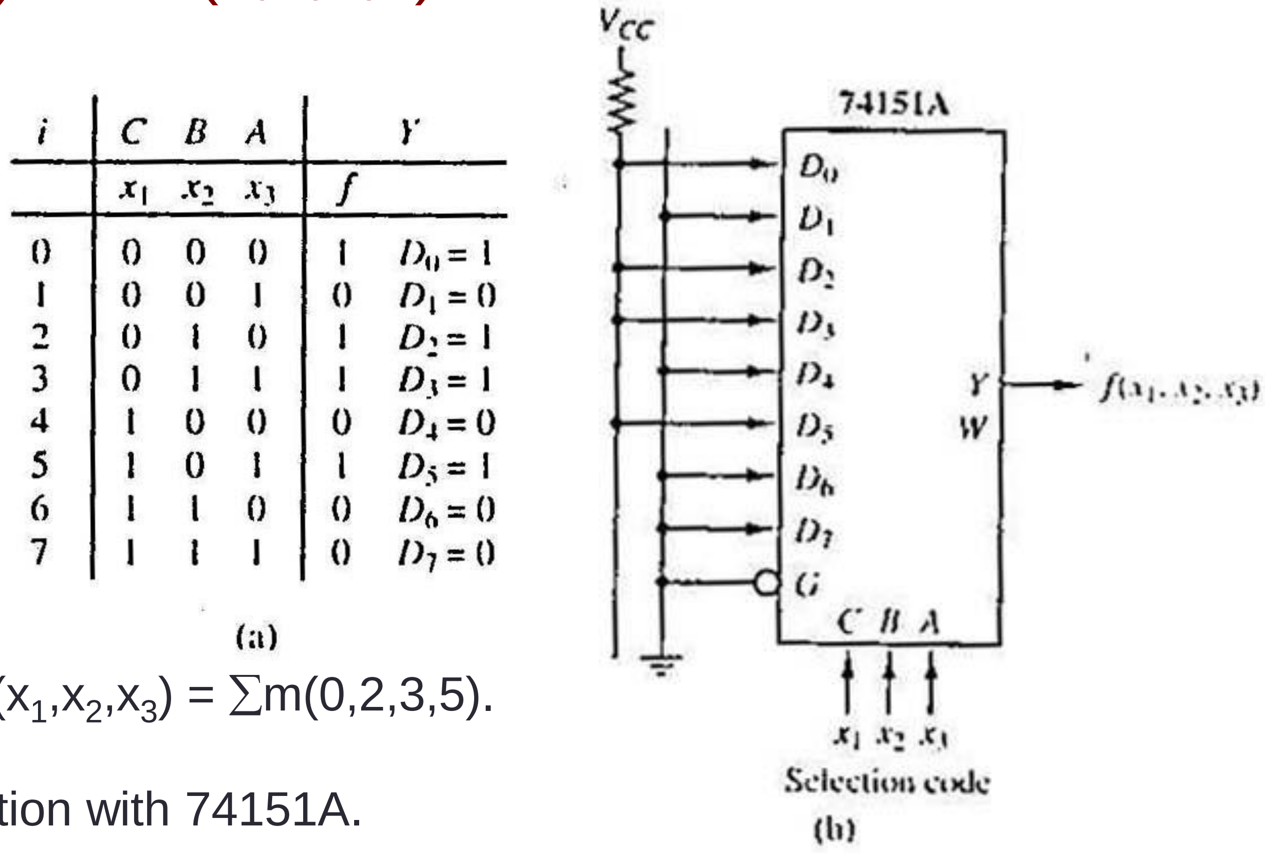
in minterms notation, it is equal to

$$I_0 \cdot m_0 + I_1 \cdot m_1 + I_2 \cdot m_2 + I_3 \cdot m_3$$

5. Multiplexers: Implementing Functions (3/3)

- Example: Use a 74151A to implement

  $f(x_1,x_2,x_3) = \Sigma\, m(0,2,3,5)$



i	C	B	A	Y	
	x_1	x_2	x_3	f	
0	0	0	0	1	$D_0 = 1$
1	0	0	1	0	$D_1 = 0$
2	0	1	0	1	$D_2 = 1$
3	0	1	1	1	$D_3 = 1$
4	1	0	0	0	$D_4 = 0$
5	1	0	1	1	$D_5 = 1$
6	1	1	0	0	$D_6 = 0$
7	1	1	1	0	$D_7 = 0$

(a)

Realization of $f(x_1,x_2,x_3) = \Sigma m(0,2,3,5)$.
(a) Truth table.
(b) Implementation with 74151A.

5. Using Smaller Multiplexers (1/6)

- Earlier, we saw how a 2^n-to-1 multiplexer can be used to implement a Boolean function of *n* (input) variables.
- However, we can use a single smaller $2^{(n-1)}$-to-1 multiplexer to implement a Boolean function of *n* (input) variables.
- Example: The function

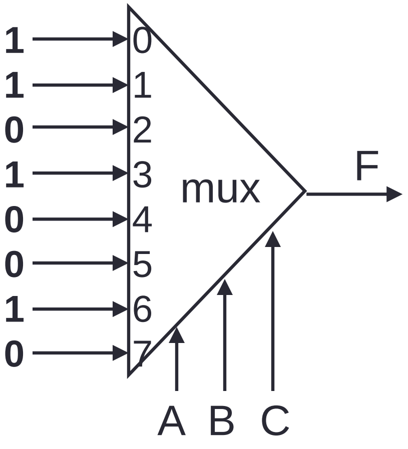
  $$F(A,B,C) = \Sigma\ m(1,3,5,6)$$

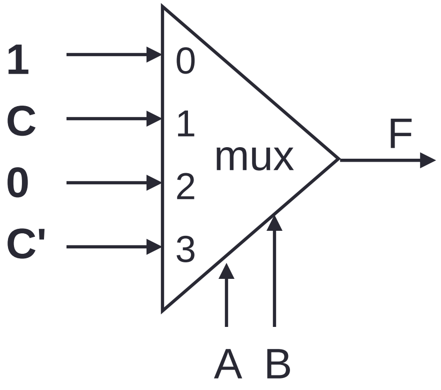
  can be implemented using a 4-to-1 multiplexer (rather than an 8-to-1 multiplexer).

5. Using Smaller Multiplexers (2/6)

- Let’s look at this example:

  $F(A,B,C) = \Sigma\, m(0,1,3,6) = A'{\cdot}B'{\cdot}C' + A'{\cdot}B'{\cdot}C + A'{\cdot}B{\cdot}C + A{\cdot}B{\cdot}C'$





- Note: Two of the variables, A and B, are applied as selection lines of the multiplexer, while the inputs of the multiplexer contain 1, C, 0 and C'.

5. Using Smaller Multiplexers (3/6)

- Procedure

1. Express Boolean function in sum-of-minterms form.

   Example: $F(A,B,C) = \Sigma\, m(0,1,3,6)$

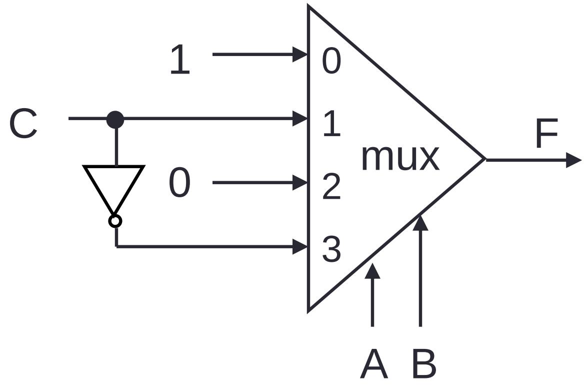
2. Reserve one variable (in our example, we take the least significant one) for input lines of multiplexer, and use the rest for selection lines.

   Example: C is for input lines; A and B for selection lines.

5. Using Smaller Multiplexers (4/6)

3. Draw the truth table for function, by grouping inputs by selection line values, then determine multiplexer inputs by comparing input line (C) and function (F) for corresponding selection line values.

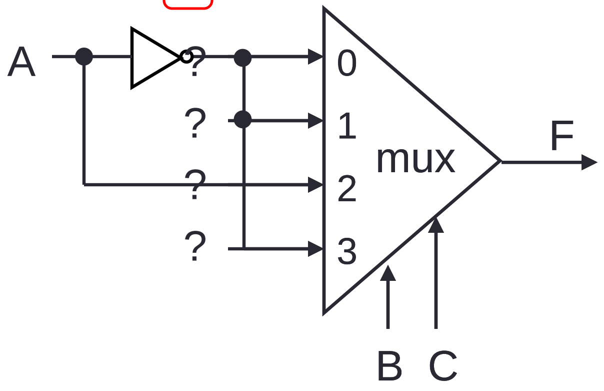
A	B	C	F	MUX input
0	0	0	1	**1**
0	0	1	1	
0	1	0	0	**C**
0	1	1	1	
1	0	0	0	**0**
1	0	1	0	
1	1	0	1	**C'**
1	1	1	0	



5. Using Smaller Multiplexers (5/6)

- Alternative: What if we use A for input lines, and B, C for selector lines?

A	B	C	F	Mux Input
0	0	0	1	1
0	0	1	1	
0	1	0	0	C
0	1	1	1	
1	0	0	0	0
1	0	1	0	
1	1	0	1	C’
1	1	1	0	



A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

A' (when BC = 00)

A' (when BC = 01)

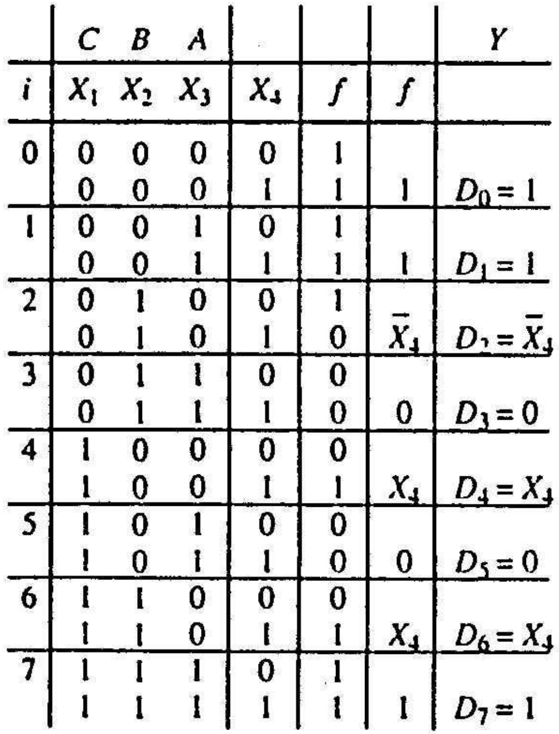
A (when BC = 10)

A' (when BC = 11)

5. Using Smaller Multiplexers (6/6)

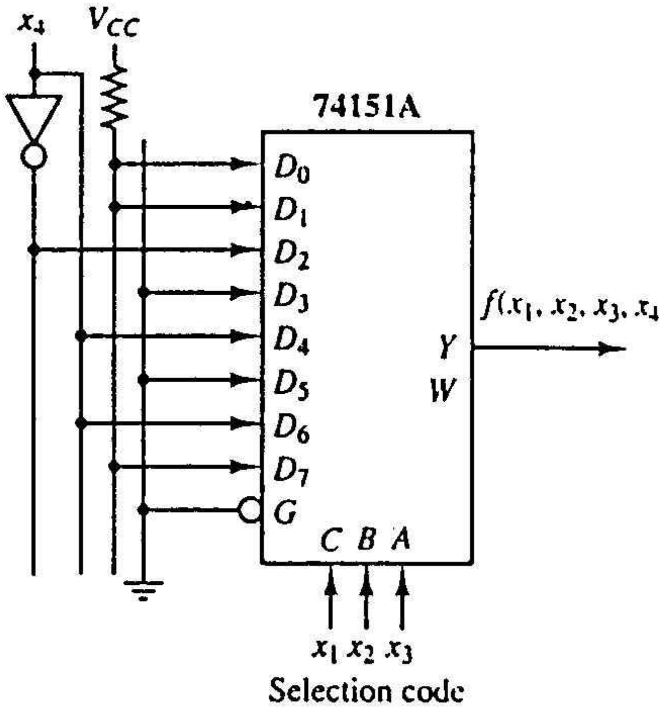
- Example: Implement the function below with 74151A:

$$f(x_1,x_2,x_3,x_4) = \Sigma\ m(0,1,2,3,4,9,13,14,15)$$



	C	B	A				Y
i	X_1	X_2	X_3	X_4	f	f	
0	0	0	0	0	1		
	0	0	0	1	1	1	$D_0 = 1$
1	0	0	1	0	1		
	0	0	1	1	1	1	$D_1 = 1$
2	0	1	0	0	1		
	0	1	0	1	0	$\bar{X}_4$	$D_2 = \bar{X}_4$
3	0	1	1	0	0		
	0	1	1	1	0	0	$D_3 = 0$
4	1	0	0	0	0		
	1	0	0	1	1	X_4	$D_4 = X_4$
5	1	0	1	0	0		
	1	0	1	1	0	0	$D_5 = 0$
6	1	1	0	0	0		
	1	1	0	1	1	X_4	$D_6 = X_4$
7	1	1	1	0	1		
	1	1	1	1	1	1	$D_7 = 1$

(a)

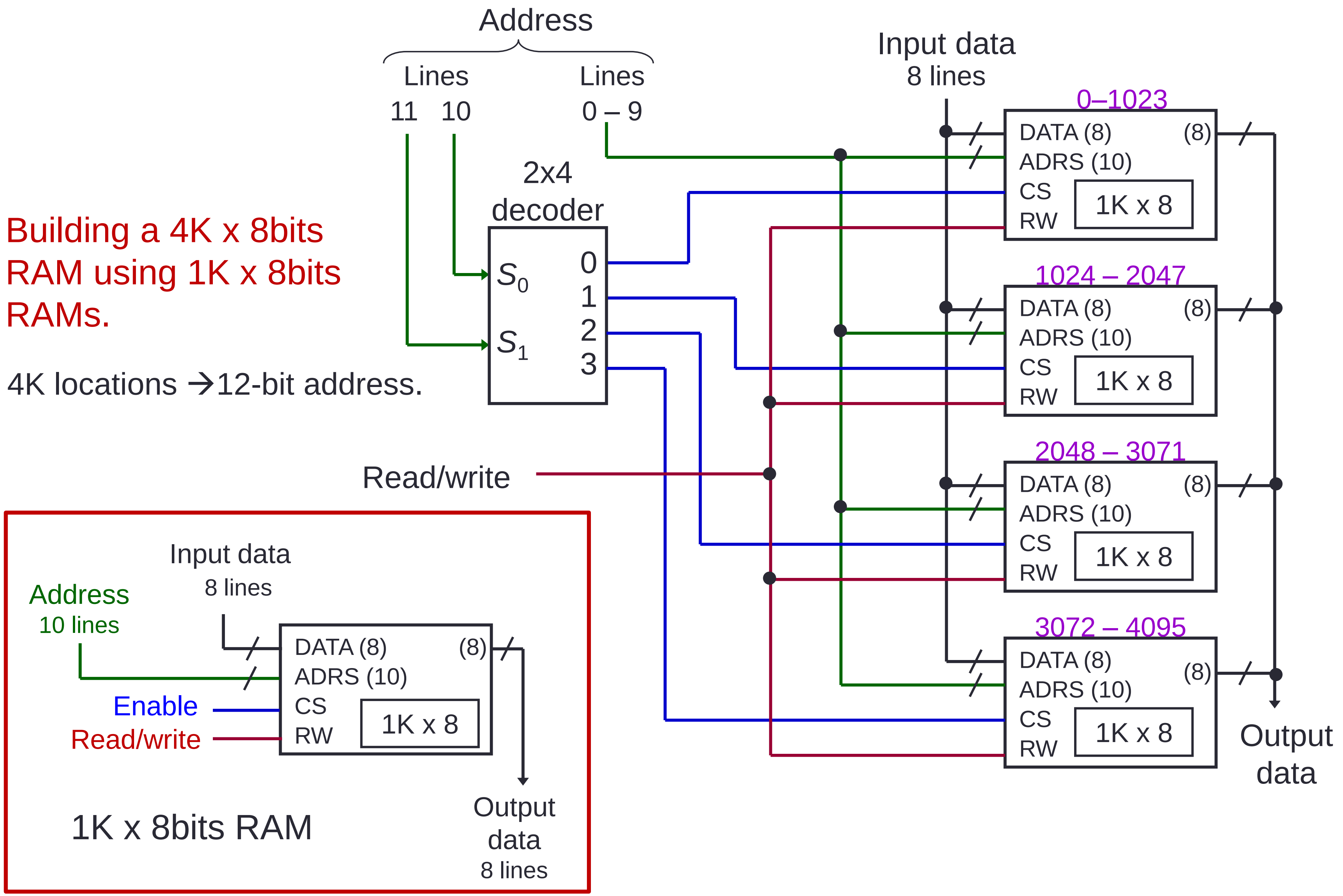


(b)

Peeking Ahead

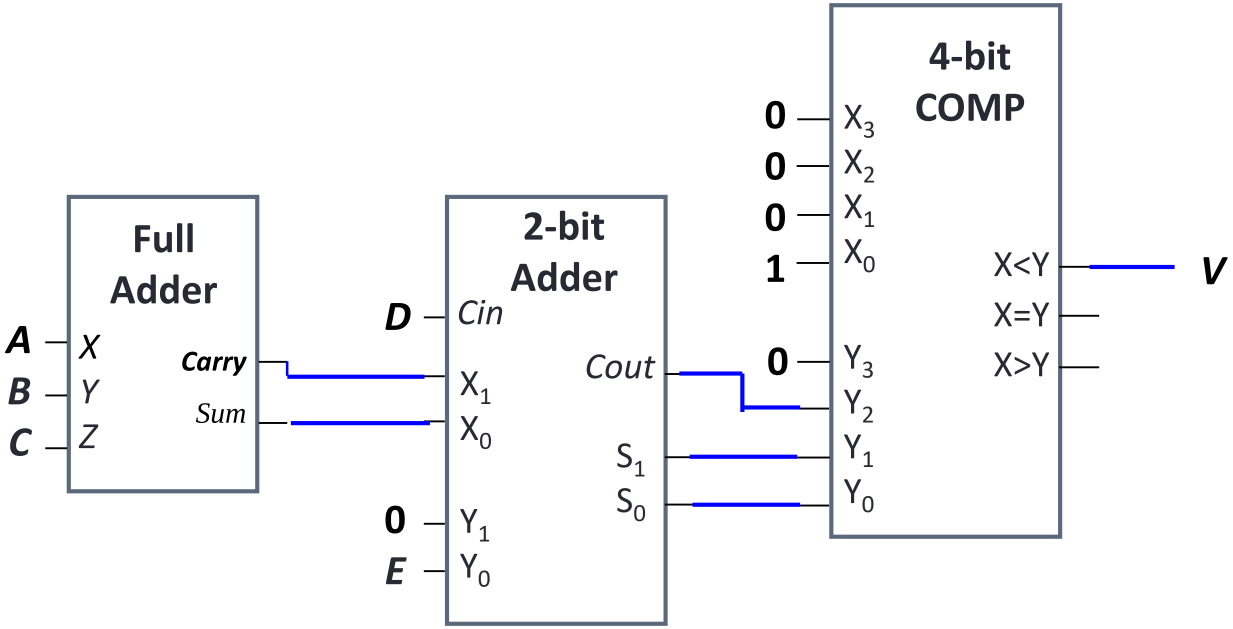
Building a 4K x 8bits RAM using 1K x 8bits RAMs.

4K locations →12-bit address.



QNA QUERIES
(PAST SEMESTER’S)

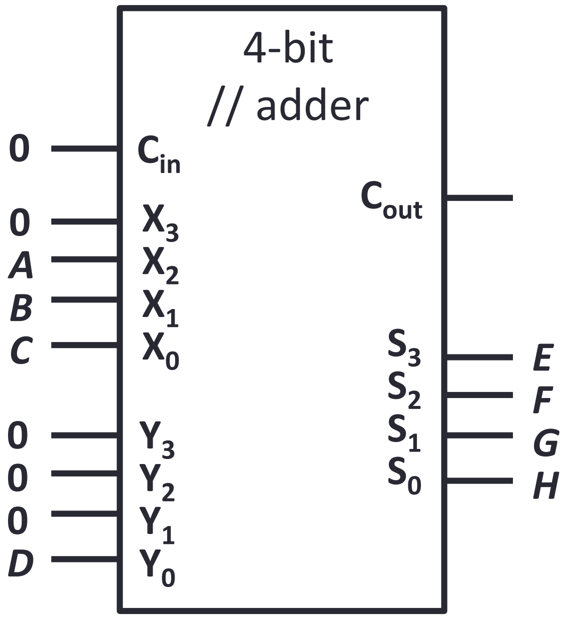
For tutorial 7 q1 why is the 2nd input 0 and E why it cannot be D and E?



Can you go through tutorial 7 q2?

Q2(a) *EFGH = (ABCD+1)/2*

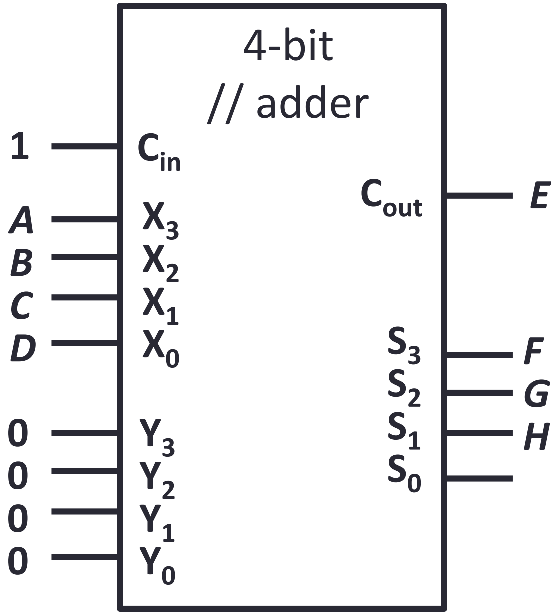
A	*B*	*C*	*D*	*E*	*F*	*G*	*H*
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1
0	0	1	1	0	0	1	0
0	1	0	0	0	0	1	0
0	1	0	1	0	0	1	1
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	0
1	0	0	1	0	1	0	1
1	0	1	0	0	1	0	1
1	0	1	1	0	1	1	0
1	1	0	0	0	1	1	0
1	1	0	1	0	1	1	1
1	1	1	0	0	1	1	1
1	1	1	1	1	0	0	0



Can you go through tutorial 7 q2?

Q2(a) *EFGH* = (*ABCD*+1)/2

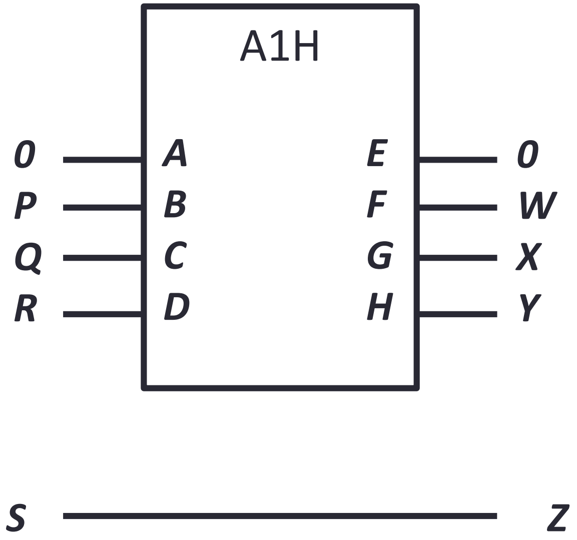
A	*B*	*C*	*D*	*E*	*F*	*G*	*H*
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1
0	0	1	1	0	0	1	0
0	1	0	0	0	0	1	0
0	1	0	1	0	0	1	1
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	0
1	0	0	1	0	1	0	1
1	0	1	0	0	1	0	1
1	0	1	1	0	1	1	0
1	1	0	0	0	1	1	0
1	1	0	1	0	1	1	1
1	1	1	0	0	1	1	1
1	1	1	1	1	0	0	0



Can you go through tutorial 7 q2?

Q2(b) 4221-to-8421 decimal code converter

P	Q	R	S	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	0
0	0	1	1	0	0	1	1
0	1	1	0	0	1	0	0
1	0	0	1	0	1	0	1
1	1	0	0	0	1	1	0
1	1	0	1	0	1	1	1
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	1



PAST YEARS’ QUESTIONS

MSI Components

Past years’ exam questions

- AY2016/17 Sem2 Q2
- AY2017/18 Sem2 Q3(a)(b)(c)
- AY2018/19 Sem2 Q5(c)
- AY2019/20 Sem2 Q5(a)(b)

Assumption

- Logical constants 0 and 1 are available, but not complemented literals.

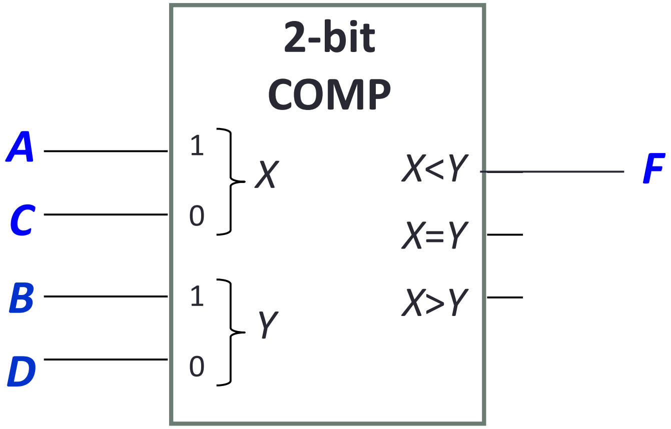
AY2016/17 Sem2 Q2(a)

2. **[10 marks]**

a. Given the following Boolean function:

$$F(A,B,C,D) = \Sigma m(1, 4, 5, 6, 7, 13)$$

You are to implement F using a single **2-bit magnitude comparator** with no additional logic gates. Note that complemented literals are not available. [5 marks]



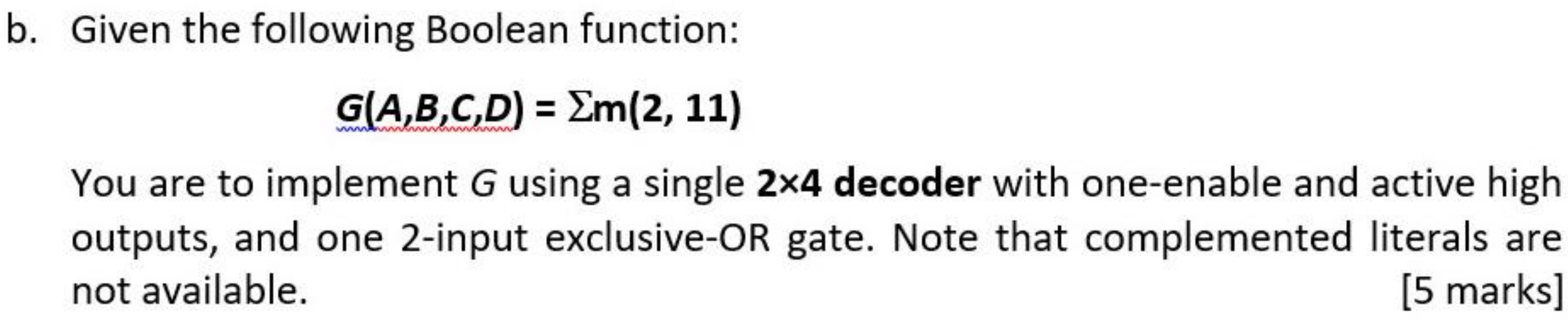
Observe that

$$F = AC < BD$$

Other answers possible.

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

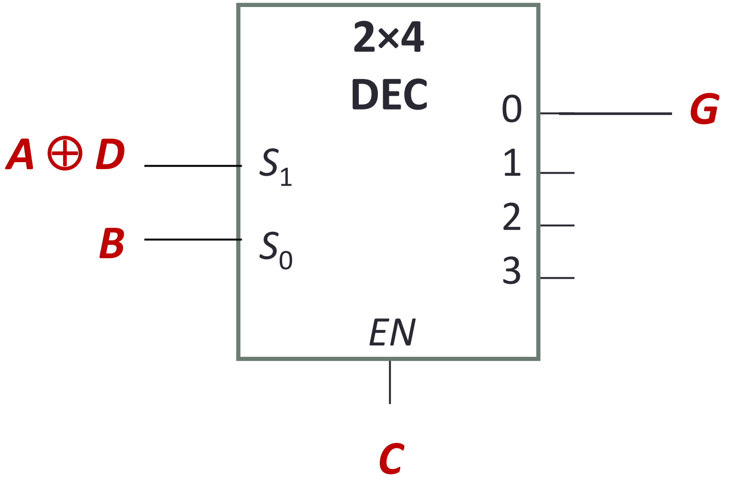
AY2016/17 Sem2 Q2(b)



b. Given the following Boolean function:

$$G(A,B,C,D) = \Sigma m(2, 11)$$

You are to implement *G* using a single **2×4 decoder** with one-enable and active high outputs, and one 2-input exclusive-OR gate. Note that complemented literals are not available. [5 marks]



Observe that

$$G(A,B,C,D) = \Sigma m(2, 11)$$

$$= A' \cdot B' \cdot C \cdot D' + A \cdot B' \cdot C \cdot D$$

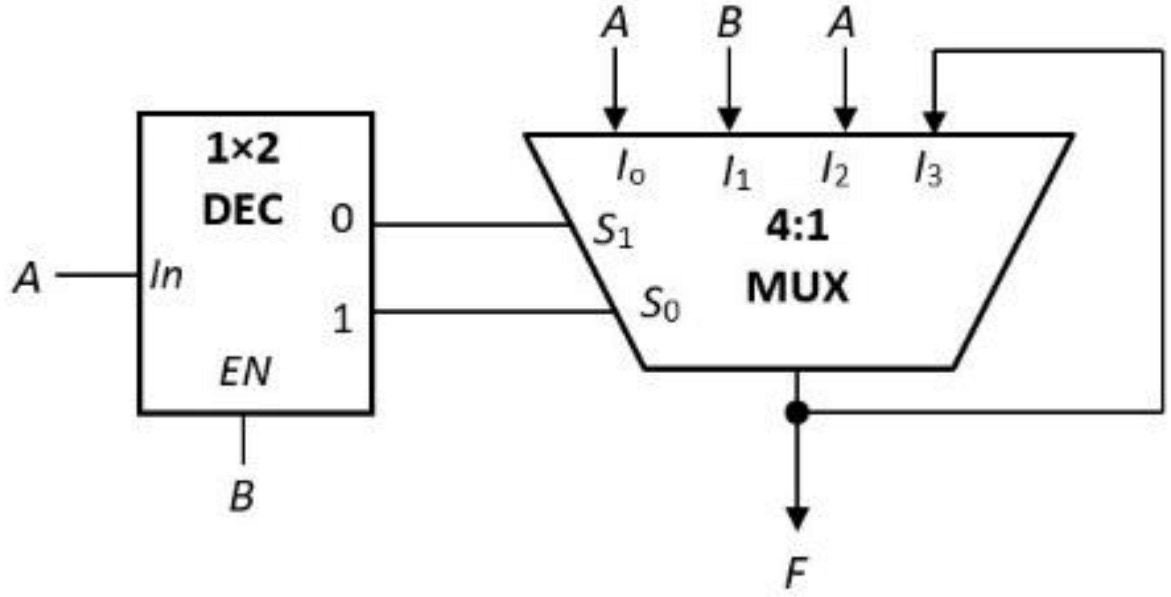
$$= C \cdot ((A' \cdot D' + A \cdot D) \cdot B')$$

$$= C \cdot ((A \oplus D)' \cdot B')$$

AY2017/18 Sem2 Q3(a)

3. **[20 marks]**

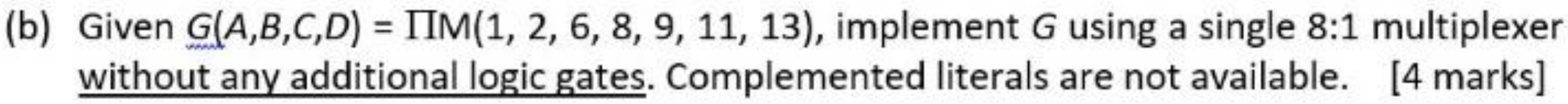
(a) Given the following circuit, what is *F*? [4 marks]



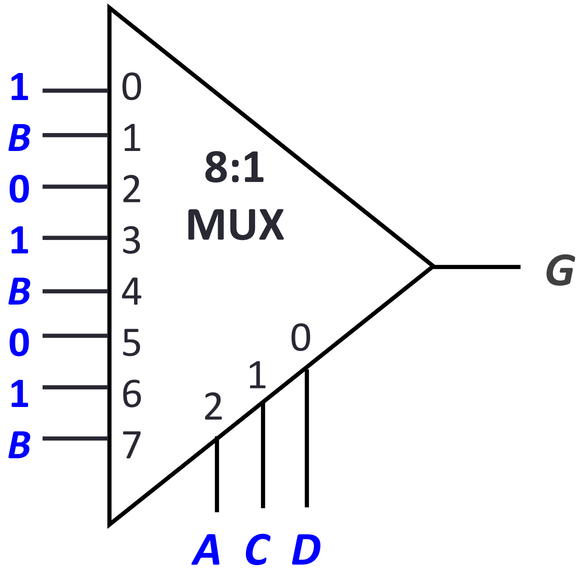
A	*B*	*F*
0	0	**0**
0	1	**0**
1	0	**1**
1	1	**1**

$F = A$

AY2017/18 Sem2 Q3(b)

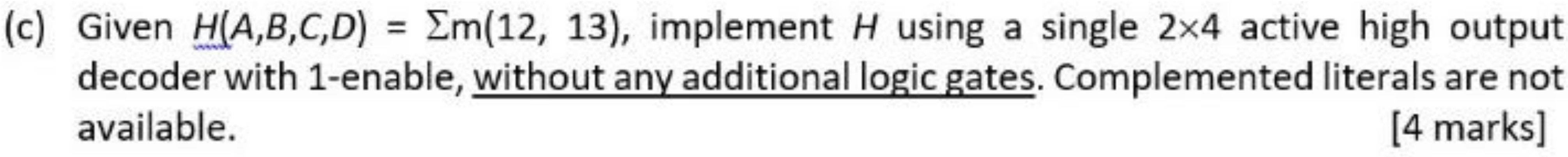


(b) Given *G*(*A*,*B*,*C*,*D*) = ΠM(1, 2, 6, 8, 9, 11, 13), implement *G* using a single 8:1 multiplexer without any additional logic gates. Complemented literals are not available. [4 marks]

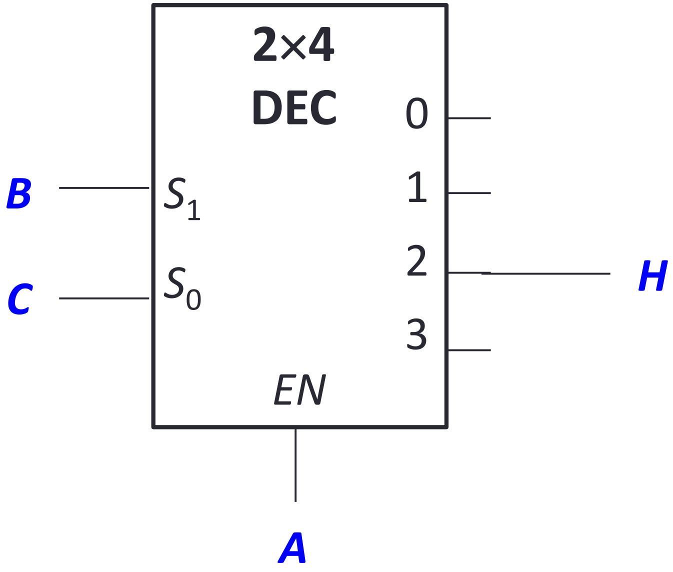


A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

AY2017/18 Sem2 Q3(c)



(c) Given $H(A,B,C,D) = \Sigma m(12, 13)$, implement H using a single 2×4 active high output decoder with 1-enable, without any additional logic gates. Complemented literals are not available. [4 marks]



$$H(A,B,C,D) = \Sigma m(12, 13)$$

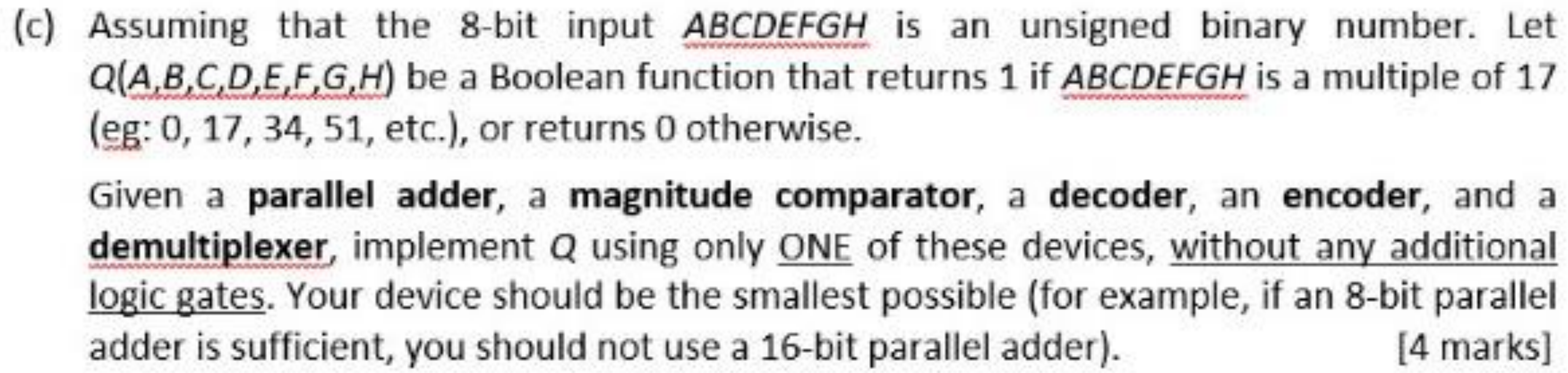
$$= A \cdot B \cdot C' \cdot D' + A \cdot B \cdot C' \cdot D$$

$$= A \cdot (B \cdot C' \cdot D' + B \cdot C' \cdot D)$$

$$= A \cdot (B \cdot C')$$

Other alternative answers possible.

AY2018/19 Sem2 Q5(c)



(c) Assuming that the 8-bit input *ABCDEFGH* is an unsigned binary number. Let *Q*(*A*,*B*,*C*,*D*,*E*,*F*,*G*,*H*) be a Boolean function that returns 1 if *ABCDEFGH* is a multiple of 17 (eg: 0, 17, 34, 51, etc.), or returns 0 otherwise.

Given a **parallel adder**, a **magnitude comparator**, a **decoder**, an **encoder**, and a **demultiplexer**, implement *Q* using only ONE of these devices, without any additional logic gates. Your device should be the smallest possible (for example, if an 8-bit parallel adder is sufficient, you should not use a 16-bit parallel adder). [4 marks]

$0_{10} = 0000\ \ 0000_2$

$17_{10} = 0001\ \ 0001_2$

$34_{10} = 0010\ \ 0010_2$

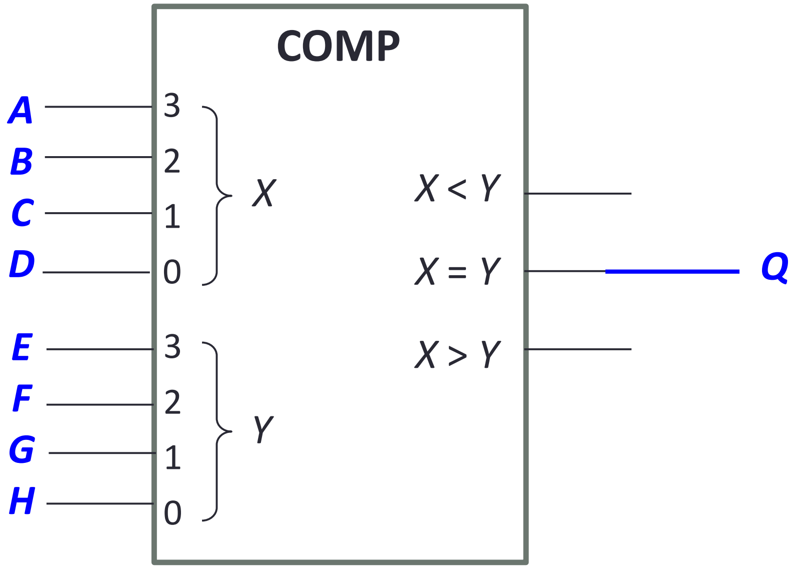
$51_{10} = 0011\ \ 0011_2$

$68_{10} = 0100\ \ 0100_2$

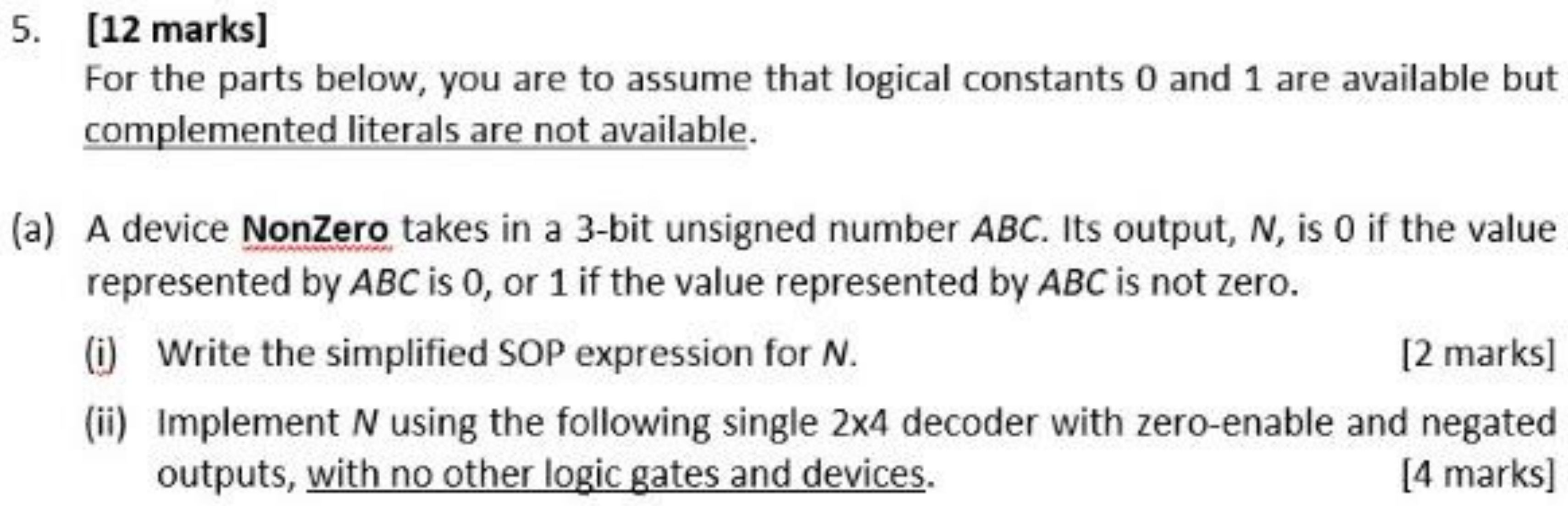
$85_{10} = 0101\ \ 0101_2$

:

$255_{10} = 1111\ \ 1111_2$



AY2019/20 Sem2 Q5(a)



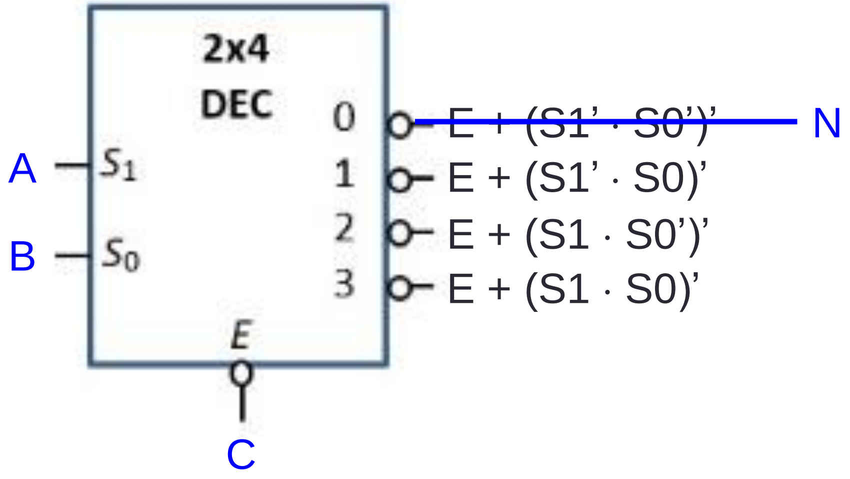
5. **[12 marks]**
For the parts below, you are to assume that logical constants 0 and 1 are available but complemented literals are not available.

(a) A device **NonZero** takes in a 3-bit unsigned number *ABC*. Its output, *N*, is 0 if the value represented by *ABC* is 0, or 1 if the value represented by *ABC* is not zero.

(i) Write the simplified SOP expression for *N*. [2 marks]

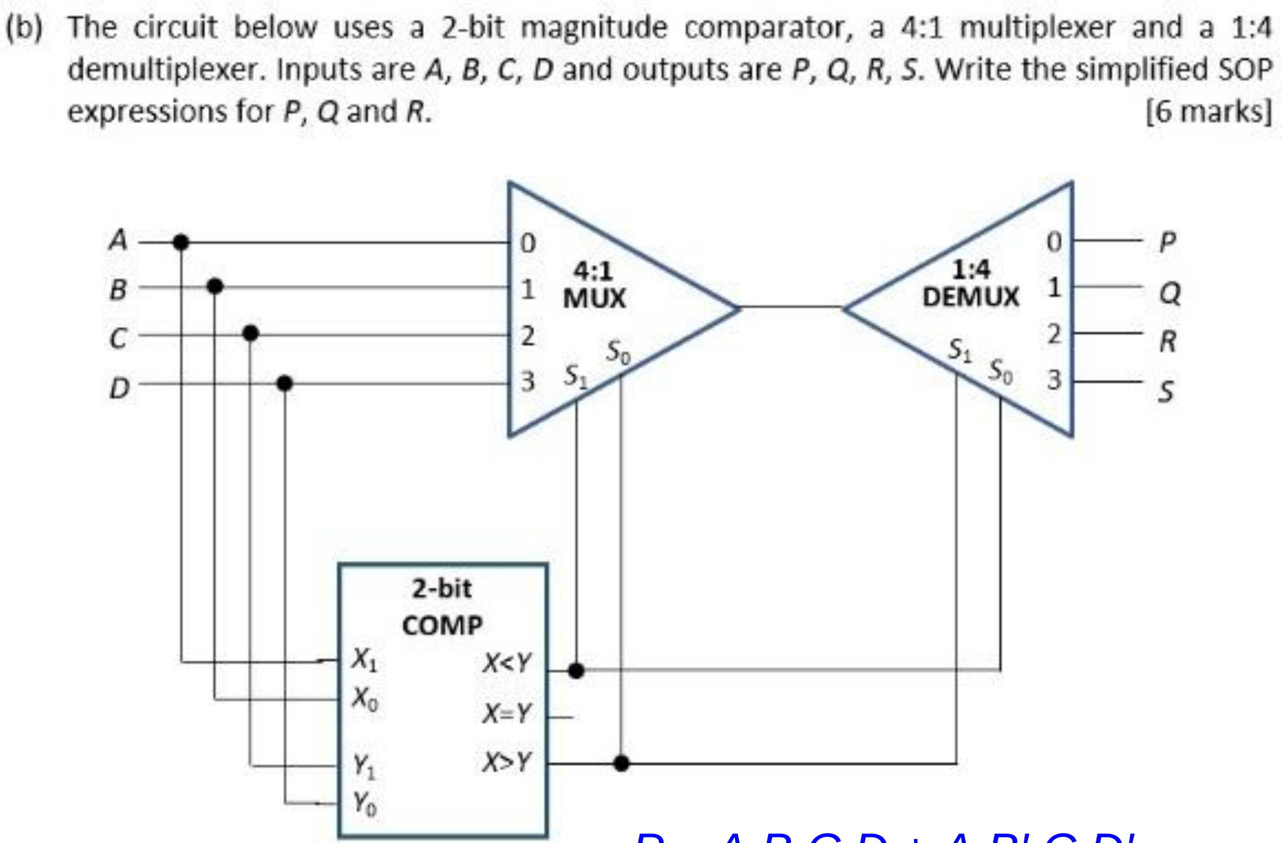
(ii) Implement *N* using the following single 2x4 decoder with zero-enable and negated outputs, with no other logic gates and devices. [4 marks]

A	B	C	N
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



N = A + B + C

AY2019/20 Sem2 Q5(b)



(b) The circuit below uses a 2-bit magnitude comparator, a 4:1 multiplexer and a 1:4 demultiplexer. Inputs are *A*, *B*, *C*, *D* and outputs are *P*, *Q*, *R*, *S*. Write the simplified SOP expressions for *P*, *Q* and *R*. [6 marks]

$P = A{\cdot}B{\cdot}C{\cdot}D + A{\cdot}B'{\cdot}C{\cdot}D'$

$Q = A'{\cdot}C + B'{\cdot}C{\cdot}D$

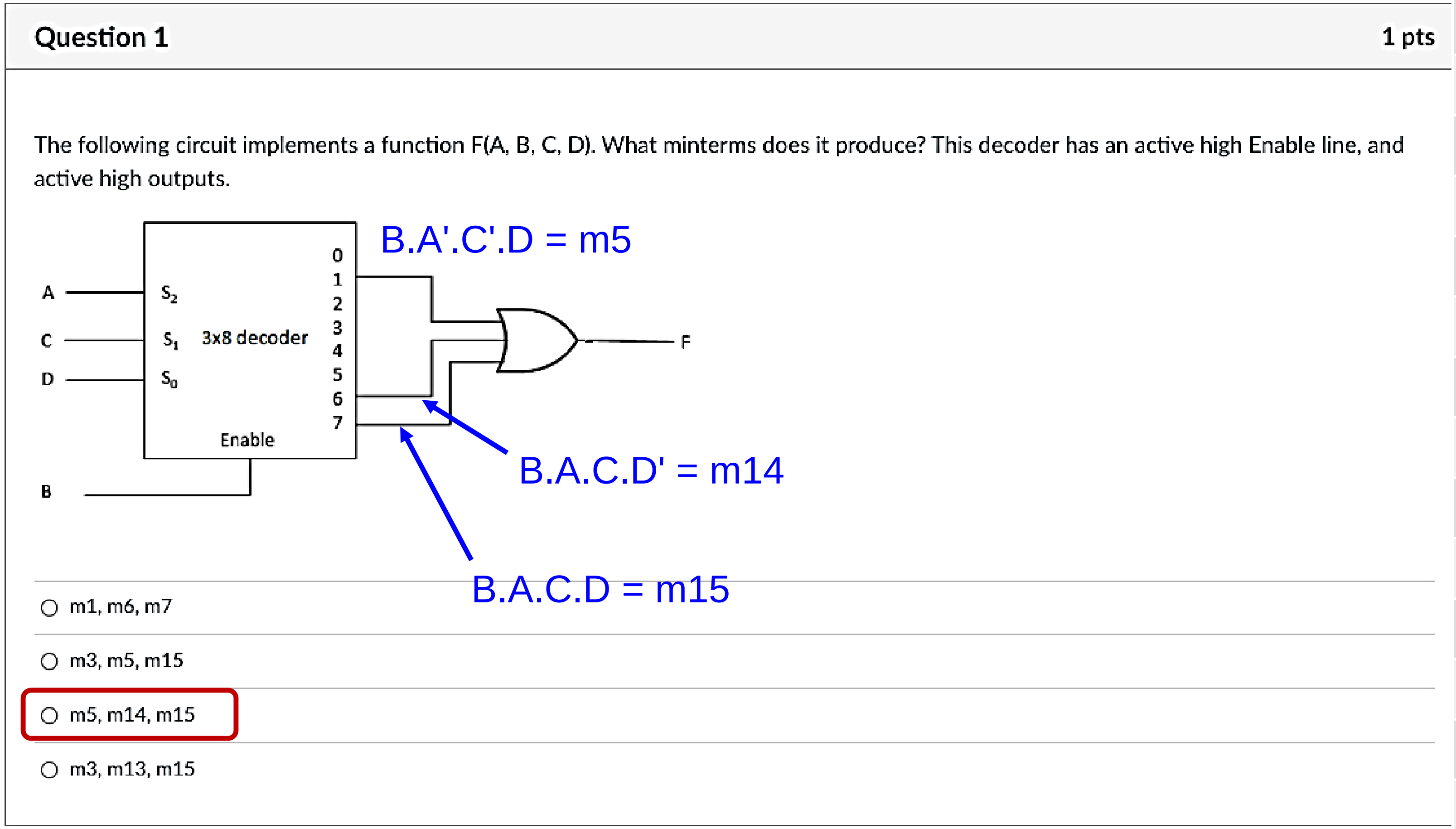
$R = B{\cdot}C'{\cdot}D' + A{\cdot}B{\cdot}C' + A{\cdot}B{\cdot}D'$

AB	*CD*	*AB<CD*	*AB>CD*	(MUX) S_1S_0	(DEMUX) S_1S_0	*PQRS*	*PQRS*
00	00	0	0	00	00	A000	0000
00	01	1	0	10	01	0C00	0000
00	10	1	0	10	01	0C00	0100
00	11	1	0	10	01	0C00	0100
01	00	0	1	01	10	00B0	0010
01	01	0	0	00	00	A000	0000
01	10	1	0	10	01	0C00	0100
01	11	1	0	10	01	0C00	0100
10	00	0	1	01	10	00B0	0000
10	01	0	1	01	10	00B0	0000
10	10	0	0	00	00	A000	1000
10	11	1	0	10	01	0C00	0100
11	00	0	1	01	10	00B0	0010
11	01	0	1	01	10	00B0	0010
11	10	0	1	01	10	00B0	0010
11	11	0	0	00	00	A000	1000

QUIZ

21. MSI Components Quiz

Question 1



Question 1 **1 pts**

The following circuit implements a function F(A, B, C, D). What minterms does it produce? This decoder has an active high Enable line, and active high outputs.

B.A'.C'.D = m5

B.A.C.D' = m14

B.A.C.D = m15

○ m1, m6, m7

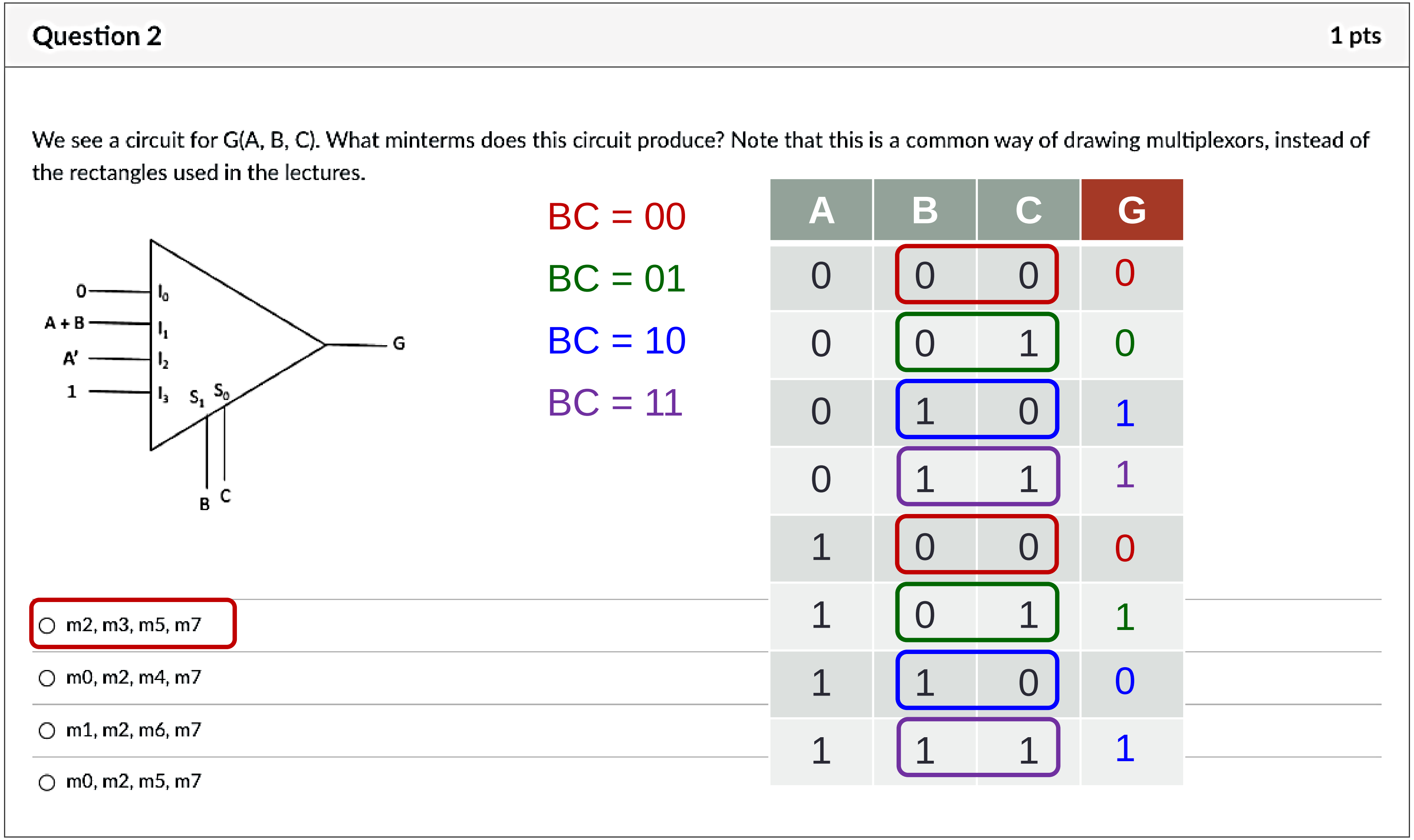
○ m3, m5, m15

○ [✓ m5, m14, m15]

○ m3, m13, m15

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Question 2



Question 2 **1 pts**

We see a circuit for G(A, B, C). What minterms does this circuit produce? Note that this is a common way of drawing multiplexors, instead of the rectangles used in the lectures.

BC = 00

BC = 01

BC = 10

BC = 11

A	B	C	G
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

○ [✓ m2, m3, m5, m7]

○ m0, m2, m4, m7

○ m1, m2, m6, m7

○ m0, m2, m5, m7

~~NEXT WEEK~~ THIS THURSDAY

Sequential Circuits

Online recitation on 27 March 2025, Thursday, 4pm

- Zoom link on Canvas > Zoom

Past years' exam questions:

- AY2015/16 Sem1 Q7
- AY2020/21 Sem2 Q12

End of File