University of British Columbia
Electrical and Computer Engineering
Digital Design and Microcomputers CPEN312

# Lecture 8: Encoders, Decoders, Multiplexers, and the ALU.

Dr. Jesús Calviño-Fraga. P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

January 26, 2017



## Objectives

- Decoders.
- Encoders.
- Priority Encoders.
- Multiplexers.
- Three State Outputs.
- The Arithmetic Logic Unit (ALU).

# Readings…

- Please read Chapter 4 of Digital Design by M. Mano & M. Ciletti, Fifth Edition, Pearson 2013.
- Posted in Connect. It complies with UBC's copyright fair dealing guidelines: http://copyright.ubc.ca/requirements/fair-dealing/
- Read about binary "Carry Look-head" and "Binary Multipliers".



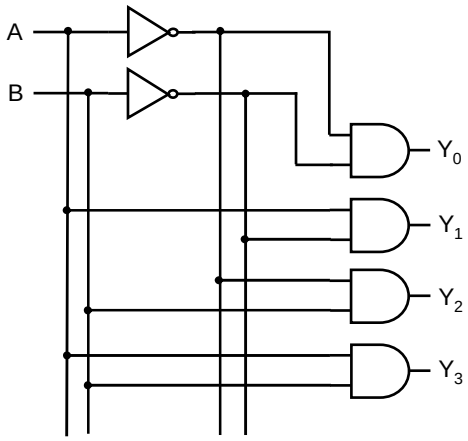


# Decoders

- A decoder is a combinational circuit that converts n binary inputs into $2^n$ unique binary outputs.
- For example, the truth table of a 2 input decoder may look like this:

| Inputs | | Outputs | | | |
|---|---|---|---|---|---|
| B | A | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
| 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-to-4 Decoder





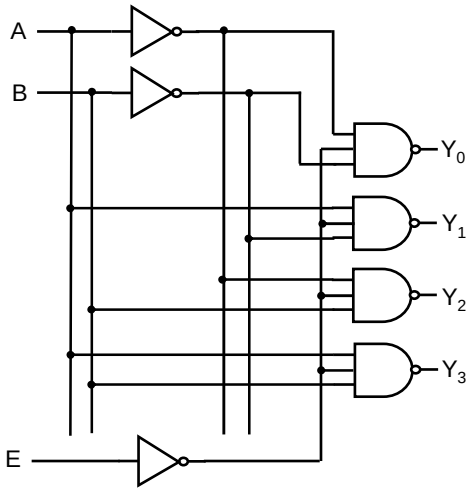



# 2-to-4 Decoder

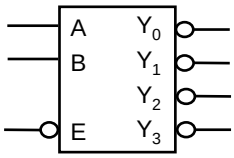
- It is more common to encounter this kind of decoder: the active output is zero, and it has a global enable pin:

| Inputs | | | Outputs | | | |
|---|---|---|---|---|---|---|
| **B** | **A** | **E** | **$Y_0$** | **$Y_1$** | **$Y_2$** | **$Y_3$** |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| X | X | 1 | 1 | 1 | 1 | 1 |

# 2-to-4 Decoder




- It is so commonly used that it has his own symbol:

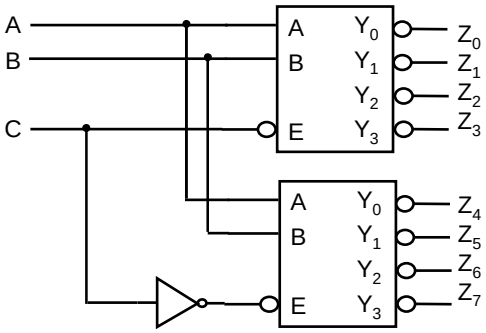







# Example 1

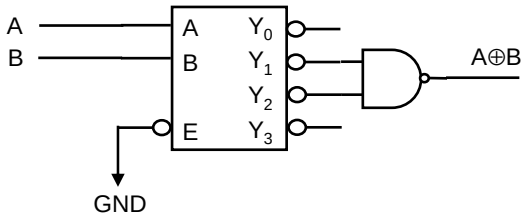
- Use two 2-to-4 decoders (with enable) and a not gate to make a 3-to-8 decoder. The outputs should be active low.



# Example 2

- Use a 2-to-4 decoder and a NAND gate to build an XOR gate.

f=A'.B+A.B'



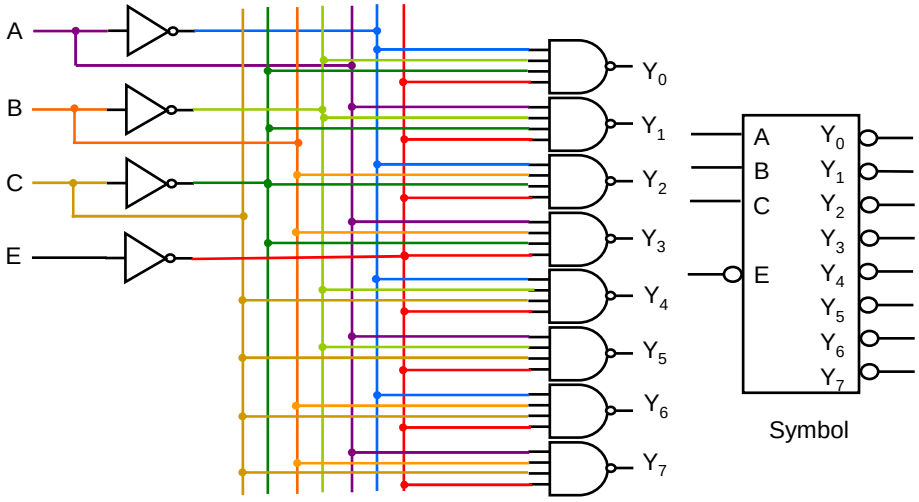





# 3-to-8 Decoder

| Inputs | | | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C | B | A | E | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ | $Y_6$ | $Y_7$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| X | X | X | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

# 3-to-8 Decoder








# Decoder IC's: 74LS139 and 74LS138

**Function Tables**

DM74LS138

| Inputs | | | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Enable | | Select | | | | | | | | | |
| G1 | G2 (Note 1) | 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 |

DM74LS139

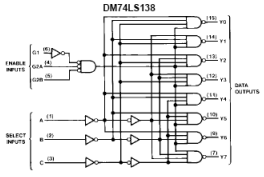
| Inputs | | | Outputs | | | |
|---|---|---|---|---|---|---|
| Enable | Select | | | | | |
| G | B | A | Y0 | Y1 | Y2 | Y3 |
| H | X | X | H | H | H | H |
| L | L | L | L | H | H | H |
| L | L | H | H | L | H | H |
| L | H | L | H | H | L | H |
| L | H | H | H | H | H | L |

H = HIGH Level
L = LOW Level
X = Don't Care

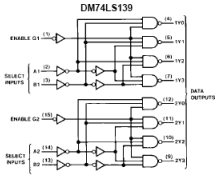
Note 1: G2 = G2A + G2B

**Logic Diagrams**

DM74LS138



DM74LS139



From Farichild's DM74LS138 • DM74LS139 Decoder/Demultiplexer Datasheet

# Boolean Functions with Decoders

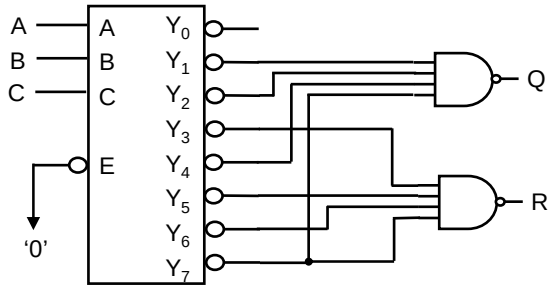
- Since we have all the minterms available in the decoder it is quite simple to combine then for any Boolean function we need.
- For example:
  - Q=C'.B'.A+C'.B.A'+C.B'.A'+C.B.A
  - R=C'.B.A+C.B'.A+C.B.A'+C.B.A





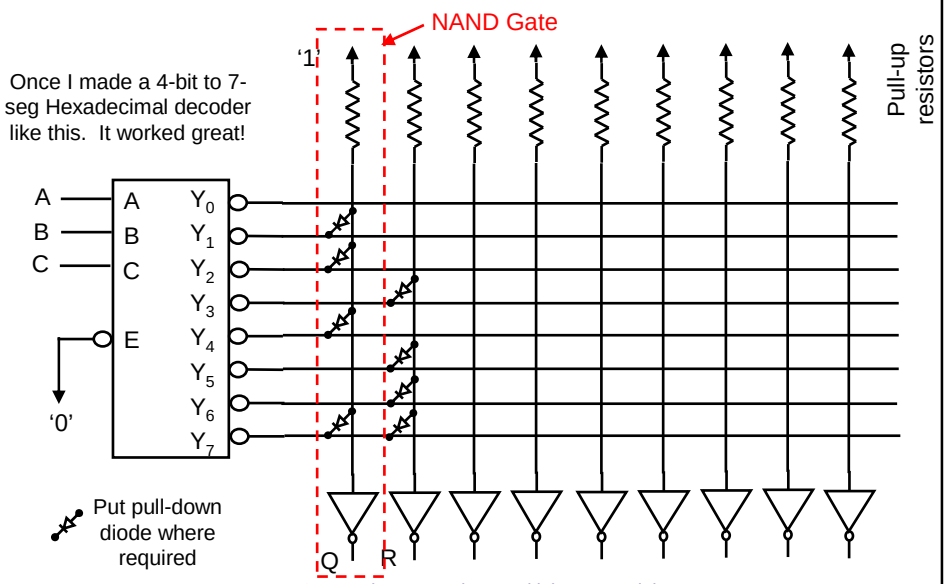
# Example 3: Boolean Functions With Decoders

Q=C'.B'.A+C'.B.A'+C.B'.A'+C.B.A
R=C'.B.A+C.B'.A+C.B.A'+C.B.A




Instead of using gates, we can make sort of a programmable array, see next slide:

## Example 3:Boolean Functions With Decoders








# Encoders

- An encoder is a digital circuit that performs the inverse operation of a decoder. It has up to $2^n$ inputs and n outputs.
- For example, consider the Octal-to-Binary encoder:

| Inputs | | | | | | | | Outputs | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | C | B | A |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

# Octal to Binary Encoder

- The outputs can be obtained with these equations:

  $C=D_0.D_1.D_2.D_3$

  $B=D_0.D_1.D_4.D_5$

  $A=D_0.D_2.D_4.D_6$
- There are two problems with this encoder:
  - Only one input can be zero at a time.
  - If no input is zero, the output is incorrect.
- To solve these problems we can use a priority encoder.





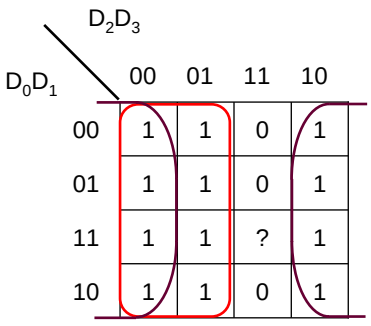
# Priority Encoder

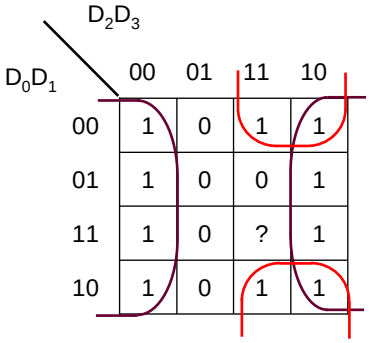
- A priority encoder gives preference to one input over another. Also, has an extra output to indicate a valid input. For example, consider the 4-to-2 priority encoder:

| Inputs | | | | Out | |
|---|---|---|---|---|---|
| $D_3$ | $D_2$ | $D_1$ | $D_0$ | B | A |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |

| Inputs | | | | Out | |
|---|---|---|---|---|---|
| $D_3$ | $D_2$ | $D_1$ | $D_0$ | B | A |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | ? | ? |

# Priority Encoder




$B=D_2'+D_3'$

$A=D_1'.D_2+D_3'$

$V=(D_0.D_1.D_2.D_3)'$

If V=1, Encoder has valid output.

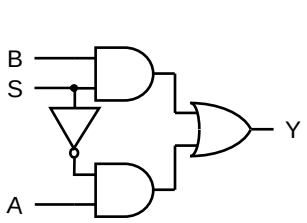


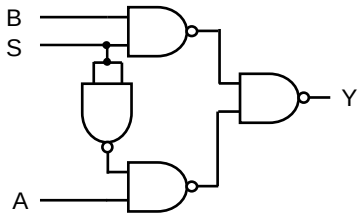


# Multiplexers

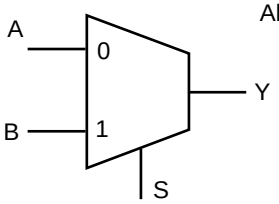
- A multiplexer is a circuit that selects one of many inputs and connects it to an output.
- Two kinds of multiplexers you are likely to use:
  - Digital. We already used one as a data selector in the 9's complement for the BCD Adder/Subtractor.
  - Analog. Very useful to measure different voltages within the same Analog to Digital Converter (ADC), change the gain of amplifiers, sample signals, etc. Will not be cover in this course, but it worth checking them out. See for example the 74HC4051, 74HC4052, and 74HC4053 ICs.

# 2-to-1 Multiplexer




All NAND version.

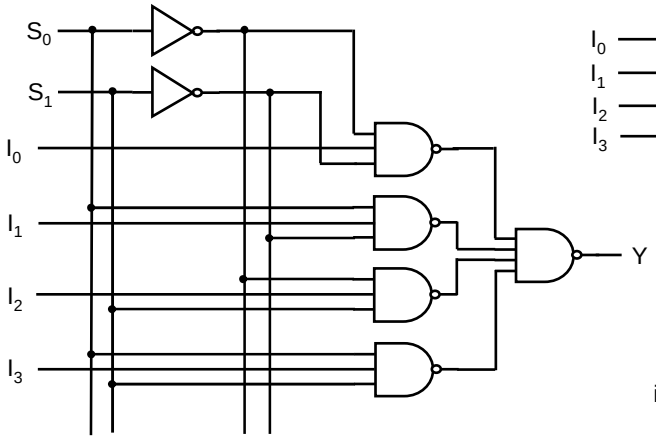



Symbol

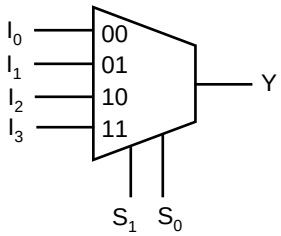




# 4-to-1 Multiplexer

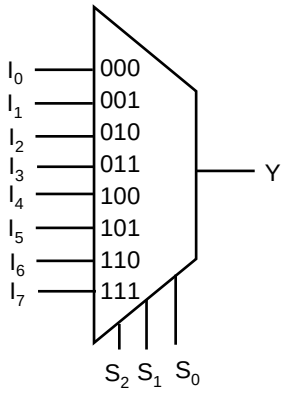



Symbol




With a 4-to-1 multiplexer we can implement ANY two input logic function!

# 8-to-1 Multiplexer



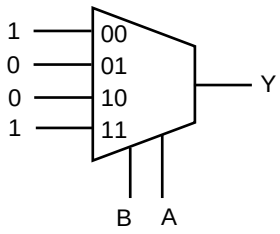





# Example 4

- Implement a two input XNOR function using a 4-to-1 multiplexer.

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



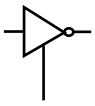
# Three-State Output Gates

- Three state gates have an extra input that disables the output of the gate.
- The three possible states of these gates are:
  - 0: Usually around 0V.
  - 1: Usually around 3.3V or 5V.
  - High Impedance: The gate does not drive a voltage output.
- Three-state gates are very useful to handle data buses in microcomputer circuits.



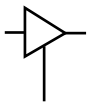


# Three State Gates



3-state NOT gate with active high output enable

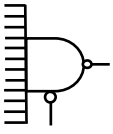
3-state NOT gate with active low output enable



3-state BUFFER with active high output enable

3-state BUFFER with active low output enable

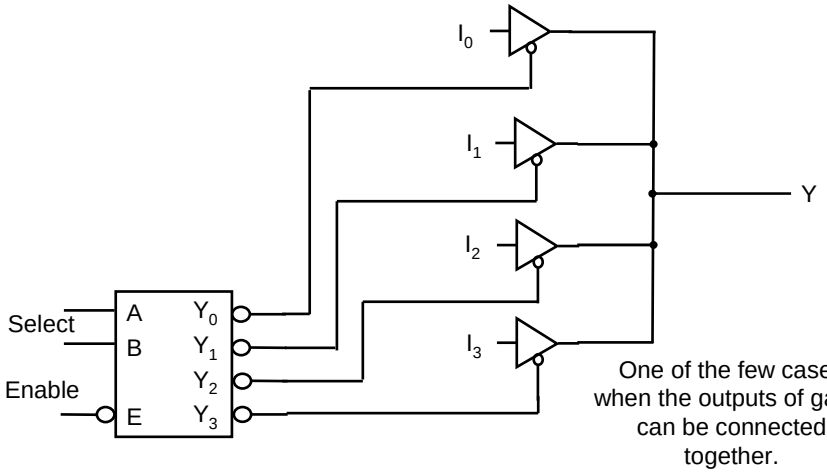
Draw the symbol for a 12-input NAND gate with active low output enable.



74S134

# Example 5

- 4-input multiplexer with 3-state gates.

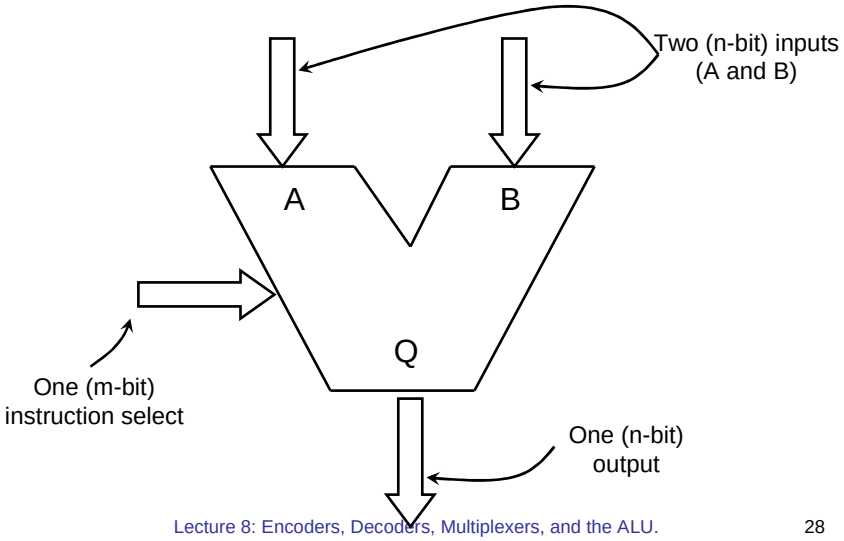



One of the few cases when the outputs of gates can be connected together.





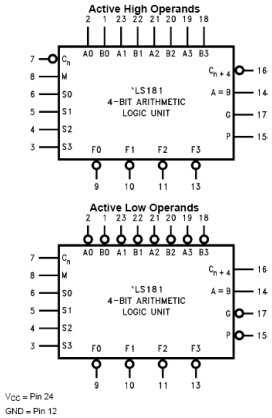
# ALU



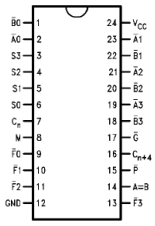
# 74LS181 ALU



**Logic Symbols**




**Connection Diagram**



**Pin Descriptions**

| Pin Names | Description |
|---|---|
| $\overline{A}0$–$\overline{A}3$ | Operand Inputs (Active LOW) |
| $\overline{B}0$–$\overline{B}3$ | Operand Inputs (Active LOW) |
| S0–S3 | Function Select Inputs |
| M | Mode Control Input |
| $C_n$ | Carry Input |
| $\overline{F}0$–$\overline{F}3$ | Function Outputs (Active LOW) |
| A = B | Comparator Output |
| $\overline{G}$ | Carry Generate Output (Active LOW) |
| $\overline{P}$ | Carry Propagate Output (Active LOW) |
| $C_{n+4}$ | Carry Output |





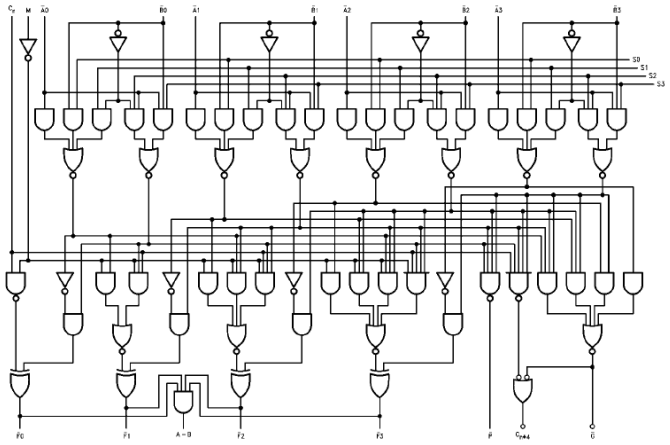
# 74LS181 ALU

**Function Table**

| Mode Select Inputs | | | | Active LOW Operands & $F_n$ Outputs | | Active HIGH Operands & $F_n$ Outputs | |
|---|---|---|---|---|---|---|---|
| | | | | Logic | Arithmetic (Note 2) | Logic | Arithmetic (Note 2) |
| S3 | S2 | S1 | S0 | (M = H) | (M = L) ($C_n$ = L) | (M = H) | (M = L) ($C_n$ = H) |
| L | L | L | L | $\overline{A}$ | A minus 1 | $\overline{A}$ | A |
| L | L | L | H | $\overline{AB}$ | AB minus 1 | $\overline{A}+B$ | A + B |
| L | L | H | L | $\overline{A}+\overline{B}$ | $A\overline{B}$ minus 1 | $\overline{A}B$ | $A+\overline{B}$ |
| L | L | H | H | Logic 1 | minus 1 | Logic 0 | minus 1 |
| L | H | L | L | $\overline{A}+\overline{B}$ | A plus $(A+\overline{B})$ | $\overline{AB}$ | A plus $A\overline{B}$ |
| L | H | L | H | $\overline{B}$ | AB plus $(A+\overline{B})$ | $\overline{B}$ | $(A+B)$ plus $A\overline{B}$ |
| L | H | H | L | $\overline{A}\oplus\overline{B}$ | A minus B minus 1 | $A\oplus B$ | A minus B minus 1 |
| L | H | H | H | $A+\overline{B}$ | $A+\overline{B}$ | $A\overline{B}$ | AB minus 1 |
| H | L | L | L | $\overline{A}B$ | A plus (A + B) | $\overline{A}+B$ | A plus AB |
| H | L | L | H | $A\oplus B$ | A plus B | $\overline{A\oplus B}$ | A plus B |
| H | L | H | L | B | $A\overline{B}$ plus (A + B) | B | $(A+\overline{B})$ plus AB |
| H | L | H | H | A + B | A + B | AB | AB minus 1 |
| H | H | L | L | Logic 0 | A plus A (Note 1) | Logic 1 | A plus A (Note 1) |
| H | H | L | H | $A\overline{B}$ | AB plus A | $A+\overline{B}$ | $(A+B)$ plus A |
| H | H | H | L | AB | $A\overline{B}$ minus A | $A+B$ | $(A+\overline{B})$ plus A |
| H | H | H | H | A | A | A | A minus 1 |

# 74LS181 ALU

**Logic Diagram**

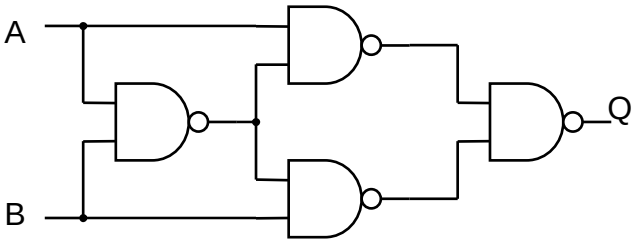






# EXOR gate

- For my design I need an EXOR implemented only with NAND gates:




| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Full adder

- Now, design and build a full adder:

| $C_i$ | $A$ | $B$ | $C_o$ | $S$ |
|---|---|---|---|---|
| 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 |

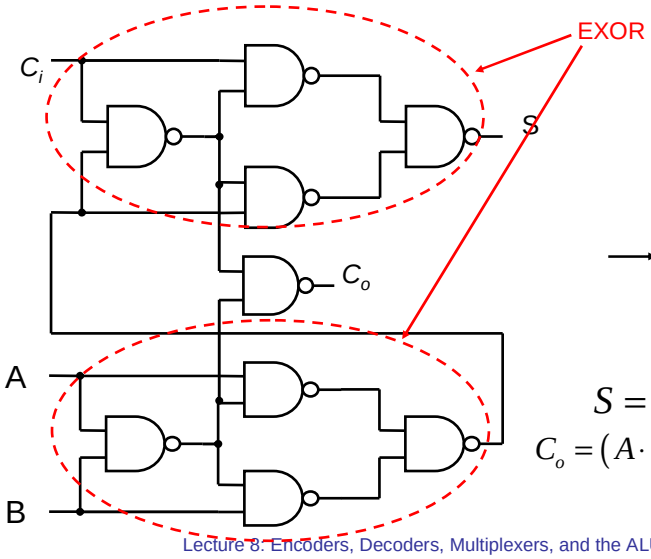
$$S = A \oplus B \oplus C_i$$

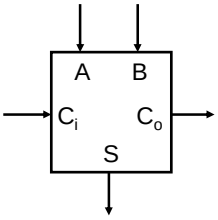
$$C_o = (A \cdot B) + (C_i \cdot (A \oplus B))$$





# Full adder circuit



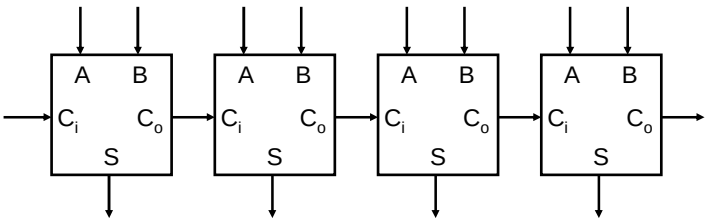




$$S = A \oplus B \oplus C_i$$

$$C_o = (A \cdot B) + (C_i \cdot (A \oplus B))$$

# Multi-bit Full Adder

- Put several simple adders together to make a multi-bit adder:

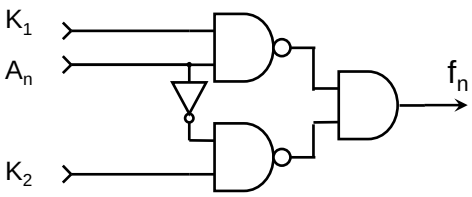



- Finally, to modify the adder into an ALU we need one more circuit:





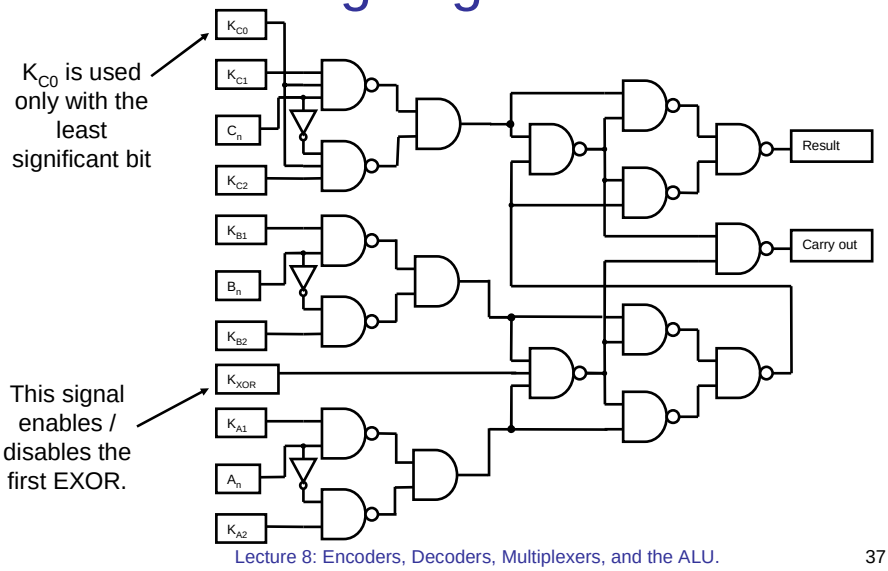
# Data selector




Put this circuit before the inputs of the simple adder, and we get a very powerful ALU!

| $K_1$ | $K_2$ | $f_n$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | $A_n$ |
| 1 | 0 | $A_n$' |
| 1 | 1 | 0 |

# Designing an ALU



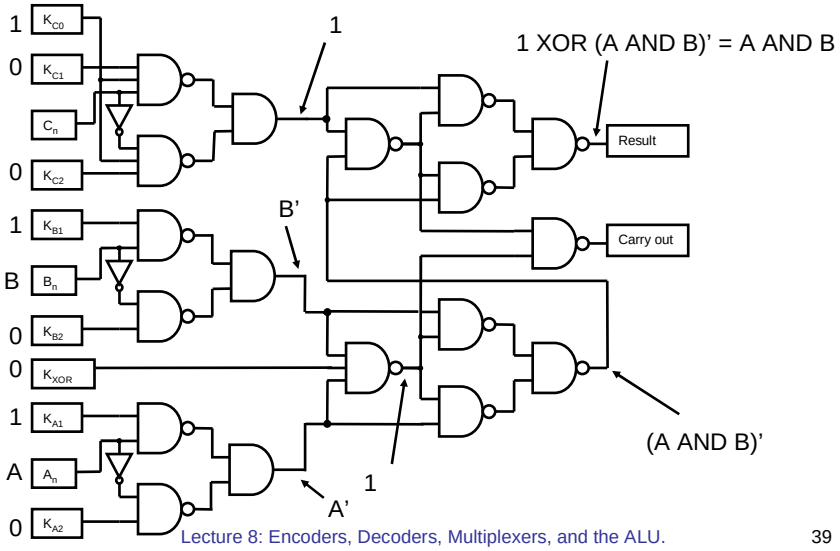





# Some Supported Operation Codes

| f | Functions | $K_{C0}$ | $K_{C1}$ | $K_{C2}$ | $K_{B1}$ | $K_{B2}$ | $K_{A1}$ | $K_{A2}$ | $K_{EXOR}$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | **A AND B** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | **A OR B** | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 2 | **A** | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 3 | **A'** | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 4 | **B** | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 5 | **B'** | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 6 | **All zeroes** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 7 | **All ones** | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 8 | **A XOR B** | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 9 | **A + 1 count** | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 10 | **A – 1 decrement** | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 11 | **B + 1 count** | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 12 | **B – 1 decrement** | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 13 | **A + B addition** | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 14 | **A – B subtraction** | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 15 | **B – A subtraction** | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |

# Testing the ALU: A AND B







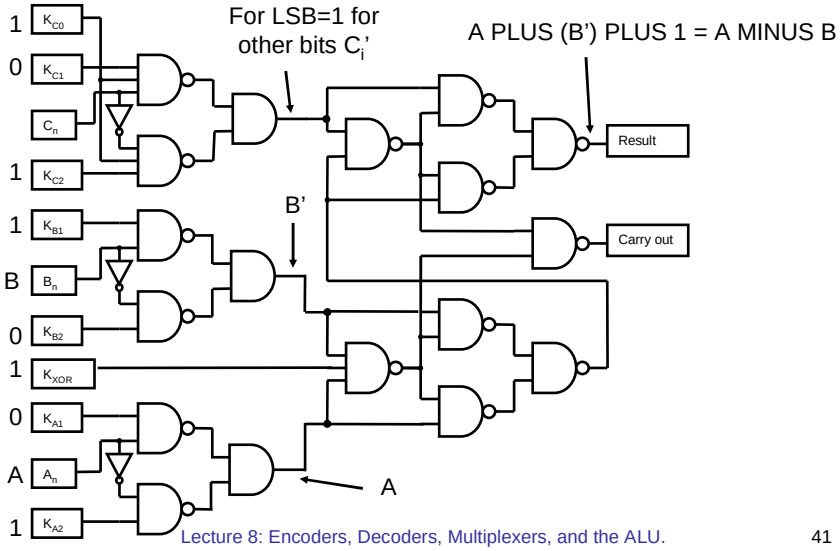

# Perform subtraction using the full adder (two's complement)

- A-B=(A+B′+1). For example:

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| ? | ? | ? | ? |

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| | | | 1 |
| 0 | 1 | 1 | 0 |

# Testing the ALU: A MINUS B








# Simple ALU

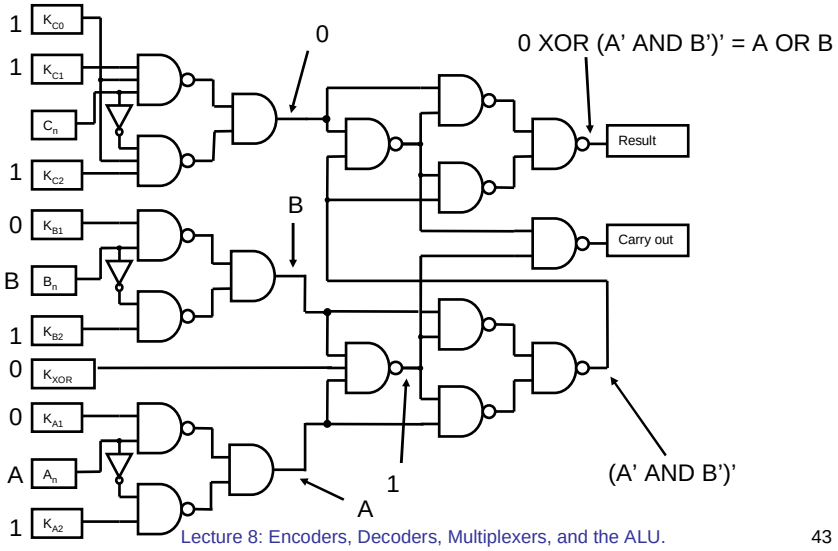
- Other operations supported by the ALU:

| | | | |
|---|---|---|---|
| $\overline{A}\ AND\ B$ | $\overline{A}\ OR\ B$ | $\overline{A}\ PLUS\ B$ | $\overline{A}\ MINUS\ B$ |
| $A\ AND\ \overline{B}$ | $A\ OR\ \overline{B}$ | $A\ PLUS\ \overline{B}$ | $A\ MINUS\ \overline{B}$ |
| $\overline{A}\ AND\ \overline{B}$ | $\overline{A}\ OR\ \overline{B}$ | $\overline{A}\ PLUS\ \overline{B}$ | $\overline{A}\ MINUS\ \overline{B}$ |

- Also notice that the OR function is implemented by using De Morgan's theorem:

$$A\ OR\ B = \overline{\overline{A}\ AND\ \overline{B}}$$

# Testing the ALU: A OR B








# Exercises

- Use a 2-to-4 decoder and a NAND gate to build an XNOR gate.
- Use a 3-to-8 decoder and NAND gates to implement a 3-bit binary to Gray code converter. In gray code only one bit changes at a time: http://en.wikipedia.org/wiki/Gray_code
- Use two 8-to-1 multiplexers to implement a full adder.
- Show how to obtain (A+B)' using the ALU described in class.