UCCD1133
Introduction to Computer Organisation and Architecture

Chapter 3

Basic Concept of Logic

Disclaimer

- This slide may contain copyrighted material of which has not been specifically authorized by the copyright owner. The use of copyrighted materials are solely for educational purpose. If you wish to use this copyrighted material for other purposes, you must first obtain permission from the original copyright owner.

Chapter 3-5

Combinational Circuits Building Blocks

Outcome

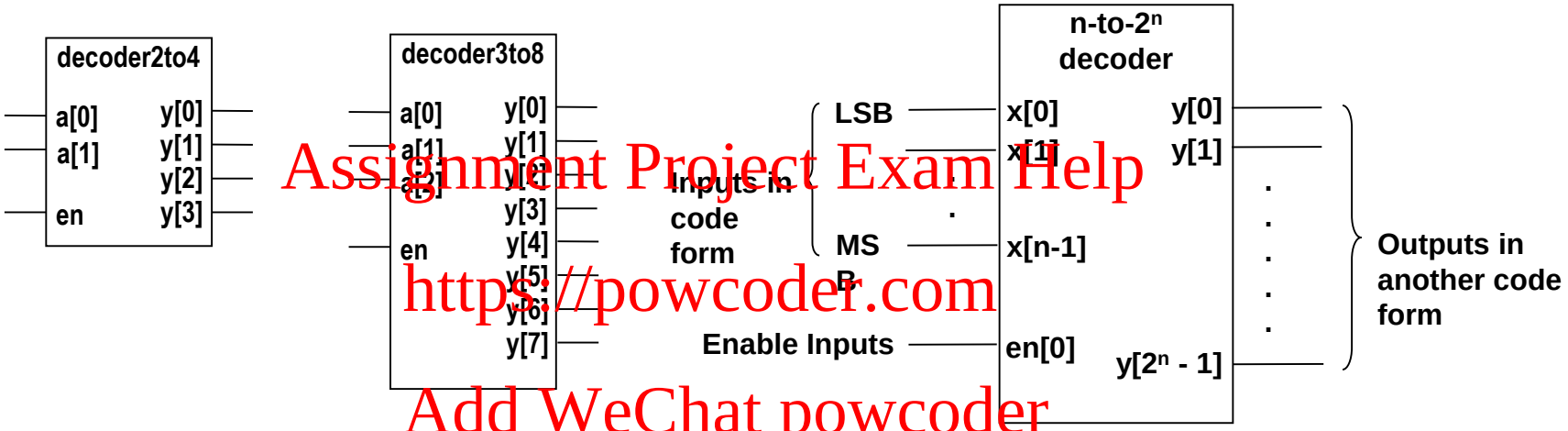
1. Describe the commonly used computational circuits building blocks:
   - Decoder
   - Multiplexer
   - Adder
2. Define and detect overflow

Combinational Circuit

- Example of non-arithmetic building blocks:
  - Decoder
  - Encoder and priority encoder
  - Demultiplexer
  - Multiplexer
- Example of arithmetic building blocks:
  - Adder
  - Subtractor
  - ALU (Arithmetic and Logic)
  - Multiplier
  - Diviser
  - Comparator
  - Shifter
- What is standard circuits?
  - Pre-designed circuits with at least correct functionalities
    - They speed up design work (modelling based on schematic, verification and implementation)
    - Collected as part of library

Decoder

- A decoder:
  - Converts one type of code, x[n-1:0], into another type, y[2^n-1:0].
  - The enable inputs, en - to enable/ disable the outputs.



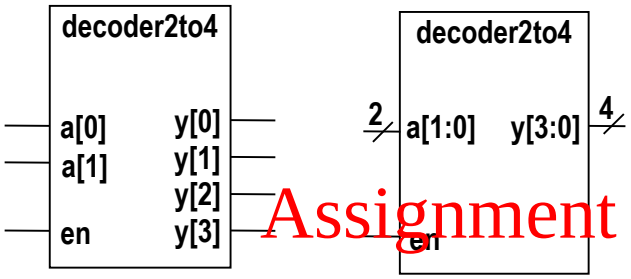
- A binary decoder
  - Most basic type of decoder
  - Also known as n-to-2^n decoder – e.g. 3-to-8 decoder.
    - Has n inputs and 2^n outputs.
  - Only one output (1-out-of-2^n) can be asserted
    - To indicate/ identify the code that has been placed at input.

- Other types of decoder: display decoder

Example: 2-to-4 Binary Decoder

- Block diagram

  First, determine the interfaces : 2 inputs and 4 outputs, plus one enable input.



- Functional behaviour

  The functional behaviour (the in-out relationship) for binary decoder:

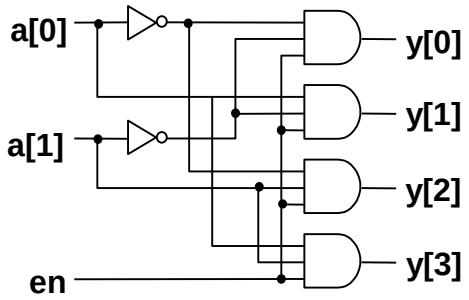
  - only an output (1-out-of-2^n) can be asserted when a code is applied to the inputs.

Inputs			Outputs			
en	a[1]	a[0]	y[3]	y[2]	y[1]	y[0]
0	d	d	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

- Logic function
  - From the truth table:

    y[0] = en (a[1]'a[0]')
    y[1] = en (a[1]'a[0])
    y[2] = en (a[1] a[0]')
    y[3] = en (a[1] a[0])

- Implementation using basic logic gates

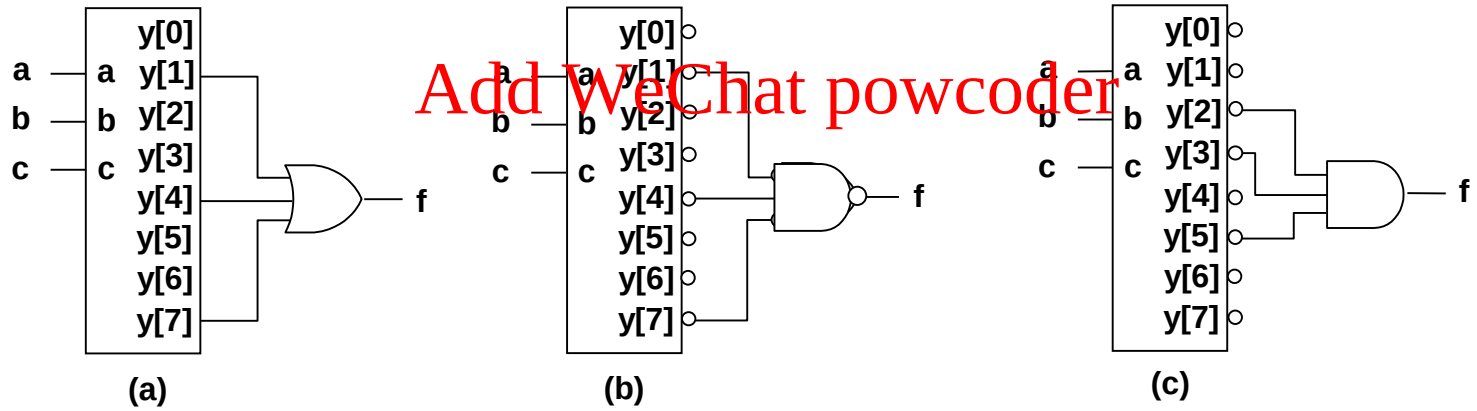


Decoder Application 1: Minterm Generation

- Notice the outputs of binary decoder represent all possible minterms.
  - Can be used as a minterm generator.
- Example

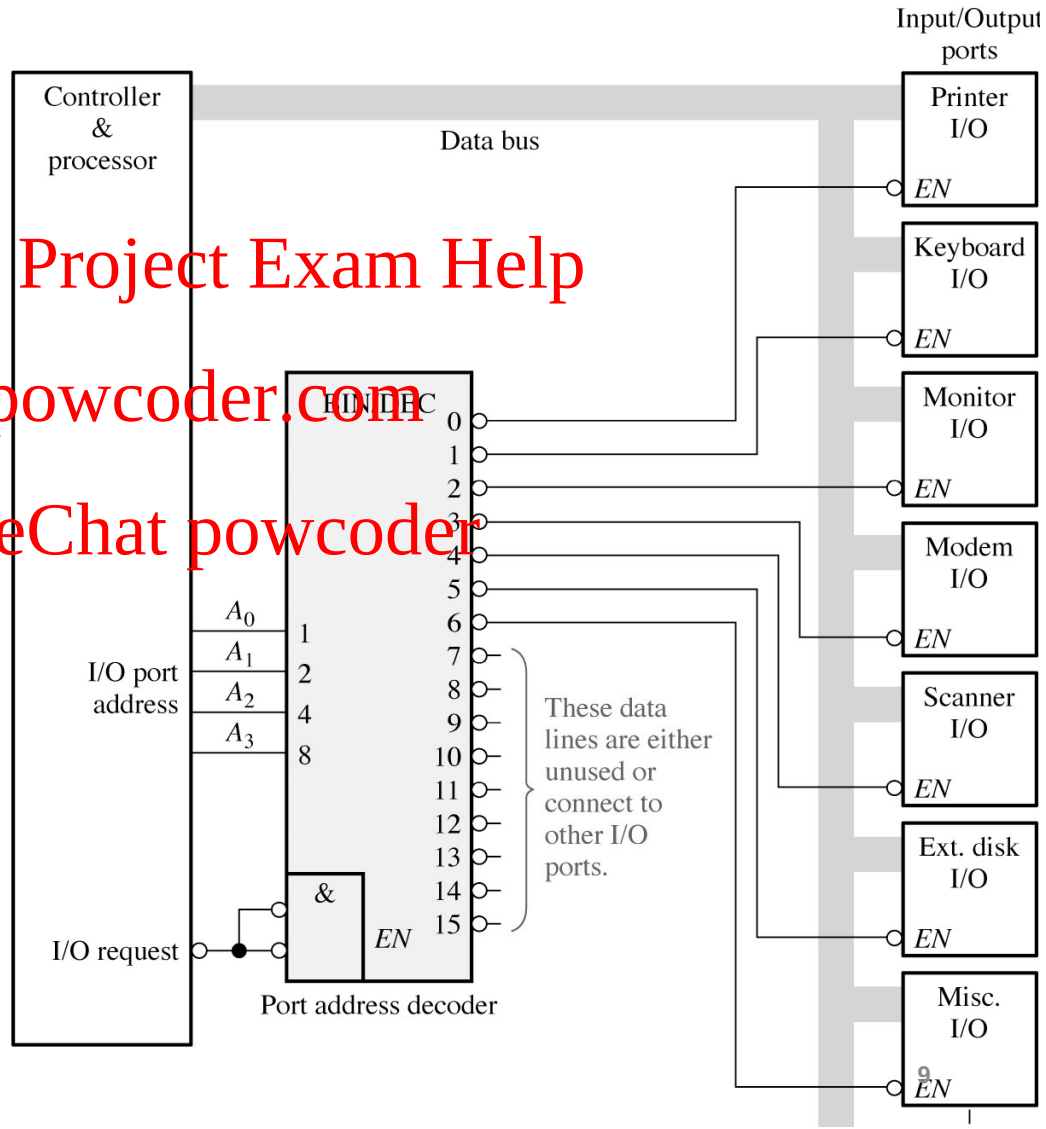
  Implement the following functions.
  - f(A,B,C) = m1 + m4 +m7: use a decoder (with active-high outputs) with an OR gate.
  - f(A,B,C) = (m1' m4' m7')': use a decoder (with active-low outputs) with a NAND gate.
  - f(A,B,C) = m2' m3' m5': use a decoder (with active-low outputs) with an AND gate.

- Solution



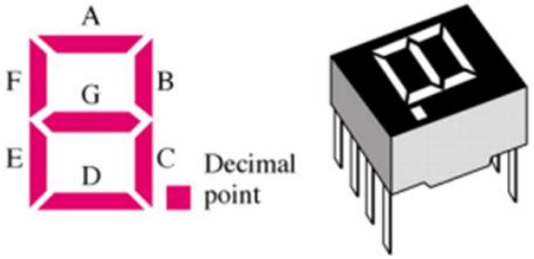
Decoder Application 2: Device Enable/Control

- Example 1
  - A 4-to-16 binary decoder to selectively enable one out of 7 I/O devices.
  - 9 outputs are not used.
  - Only one device is allowed to be active at any time.

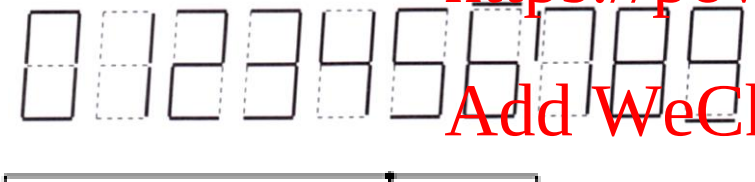


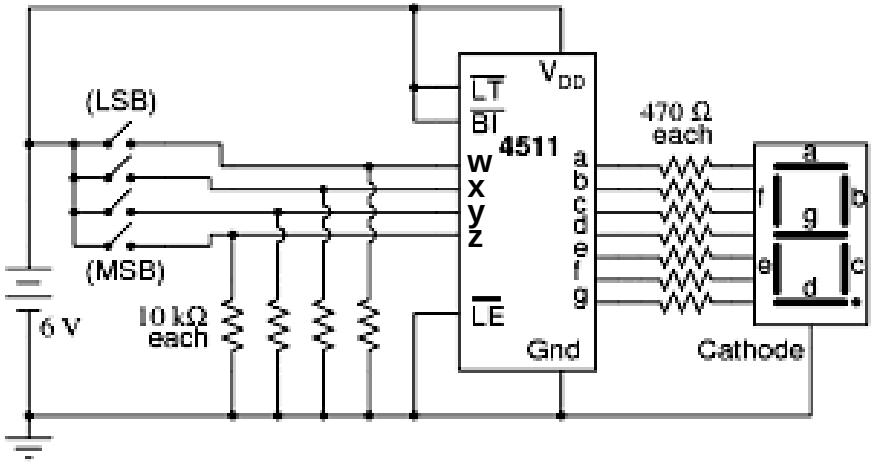
Display Decoder

- A display decoder converts a code into a suitable format to drive a numeric display (LCD, 7-segment display, dot matrix display, etc)



- Example
  - 7-segment display decoders (BCD-to-7 segment decoders)
  - Available in the market as standard components 7447 and 74X49.

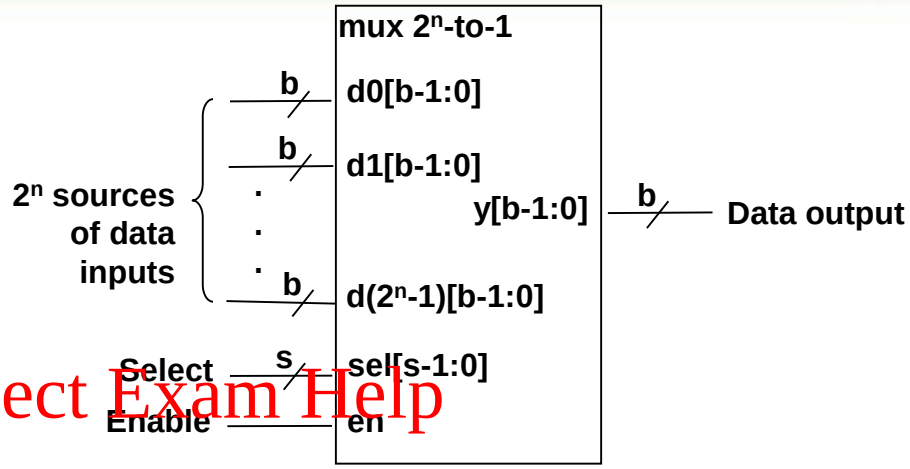




Inputs	Outputs						
w x y z	a	b	c	d	e	f	g
0 0 0 0	1	1	1	1	1	1	0
0 0 0 1	0	1	1	0	0	0	0
0 0 1 0	1	1	0	1	1	0	1
0 0 1 1	1	1	1	1	0	0	1
0 1 0 0	0	1	1	0	0	1	1
0 1 0 1	1	0	1	1	0	1	1
0 1 1 0	1	0	1	1	1	1	1
0 1 1 1	1	1	1	0	0	1	0
1 0 0 0	1	1	1	1	1	1	1
1 0 0 1	1	1	1	1	0	1	1
1 0 1 0	d	d	d	d	d	d	d
1 0 1 1	d	d	d	d	d	d	d
1 1 0 0	d	d	d	d	d	d	d
1 1 0 1	d	d	d	d	d	d	d
1 1 1 0	d	d	d	d	d	d	d
1 1 1 1	d	d	d	d	d	d	d

Multiplexer (Data Selector)

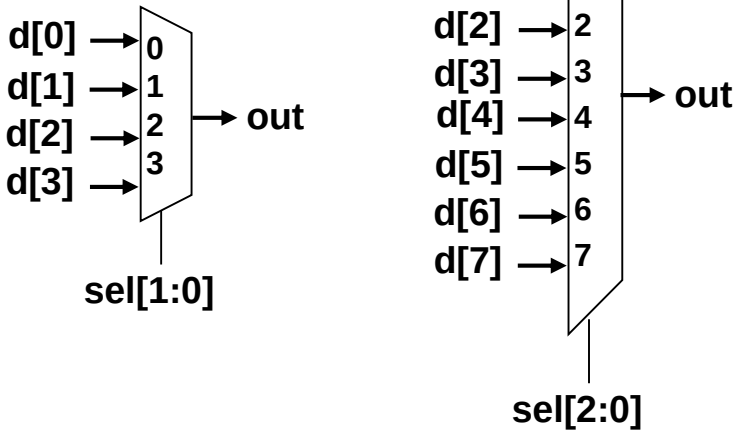
- A multiplexer (mux) is a circuit that has:
  - 2^n data inputs
  - s data selectors (control inputs)
  - An output



Block diagram of 2^n-to-1 mux

- It is used as a digital switch
  - Connects data from one of 2^n sources to the output, based on the select signals, sel[s-1:0].
  - The s-bit must meet the condition $s \geq n$

- Like many other circuits, the mux can have additional enable input signal, en
  - To enable or disable the operation of the mux
  - Set the output to z



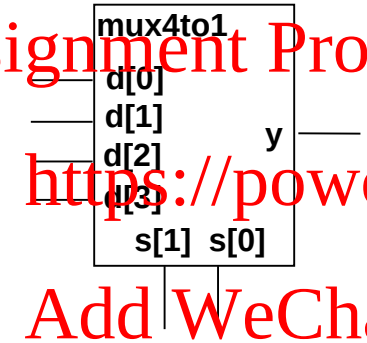
Graphical symbol of 4-to-1 mux and 8-to-1 mux

Example: 4-to-1 Multiplexer

- Block diagram

  A basic 4-to-1 multiplexer has the following signals.
  - The output signal y.
  - The input signals, d[3:0].

  Since the multiplexer has 4 data inputs, it required 2 bits data-select ($4 = 2^2$).



- Functional behaviour

s[1]	s[0]	y
0	0	d[0]
0	1	d[1]
1	0	d[2]
1	1	d[3]

- Logic function

  The logic function of the multiplexer:

  $y = s[1]'s[0]'d[0] + s[1]'s[0]\ d[1] + s[1]s[0]'d[2] + s[1]s[0]d[3]$

Application: Minterm Generation

- Select inputs -> generates minterms, while data lines are used to enable the minterms

- Example 1

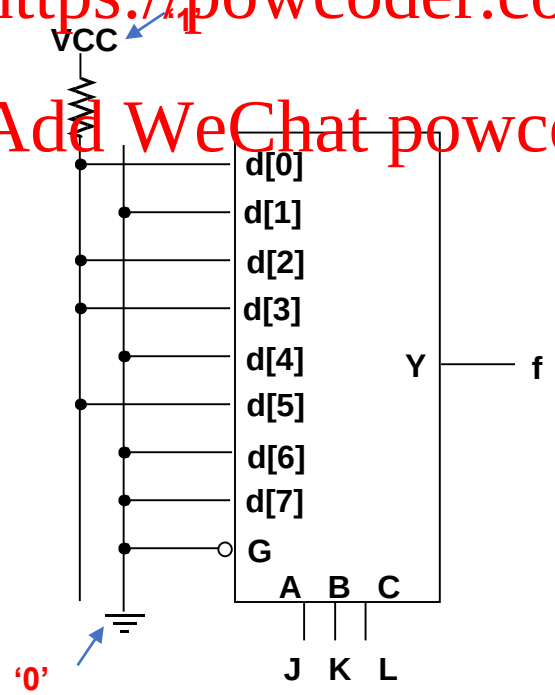
  Use a 8-to-1 multiplexer to realize f(J,K,L) = $\sum$ m(0, 2, 3, 5).

- Solution

  List the truth table for the function.

  Then map the content of the truth table to the mux.

J K L	f
0 0 0	1
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	0



Application: Minterm Generation

- Example 2

(a) Implement f(a,b,c) = ab + b'c using 4-to-1 mux if a and b are used as the select signal of the mux.

(b) Implement the above function using one 2-to-1 mux and some basic gates if a is used as the select signal of the mux.

Derive the reduced inputs (Mux inputs)

a	b	c	f	Mux inputs
0	0	0	0	d[0] = c
0	0	1	1	
0	1	0	0	d[1] = 0
0	1	1	0	
1	0	0	0	d[2] = c
1	0	1	1	
1	1	0	1	d[3] = 1
1	1	1	1	

- Solution

Part (a)

Express the logic function in canonical form.

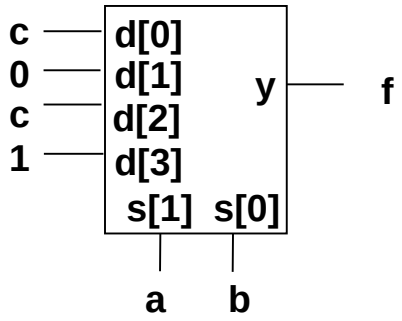
f = ab + b'c

= ab(c' + c) + b'c(a' + a)

= abc' + abc + a'b'c + ab'c

Construct its truth table.

The final result:



If not specified, other variables can be used as the select input

Application: Minterm Generation

- Solution

  Part (b)

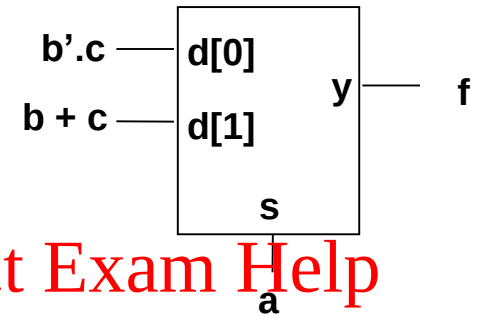
  From(a), we know that

  f = abc' + abc + a'b'c + ab'c

Construct its truth table and derive the mux inputs.

a	b	c	f	Mux inputs
0	0	0	0	
0	0	1	1	d[0] = b'.c
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	d[1] = b + c
1	1	0	1	
1	1	1	1	

The final result:

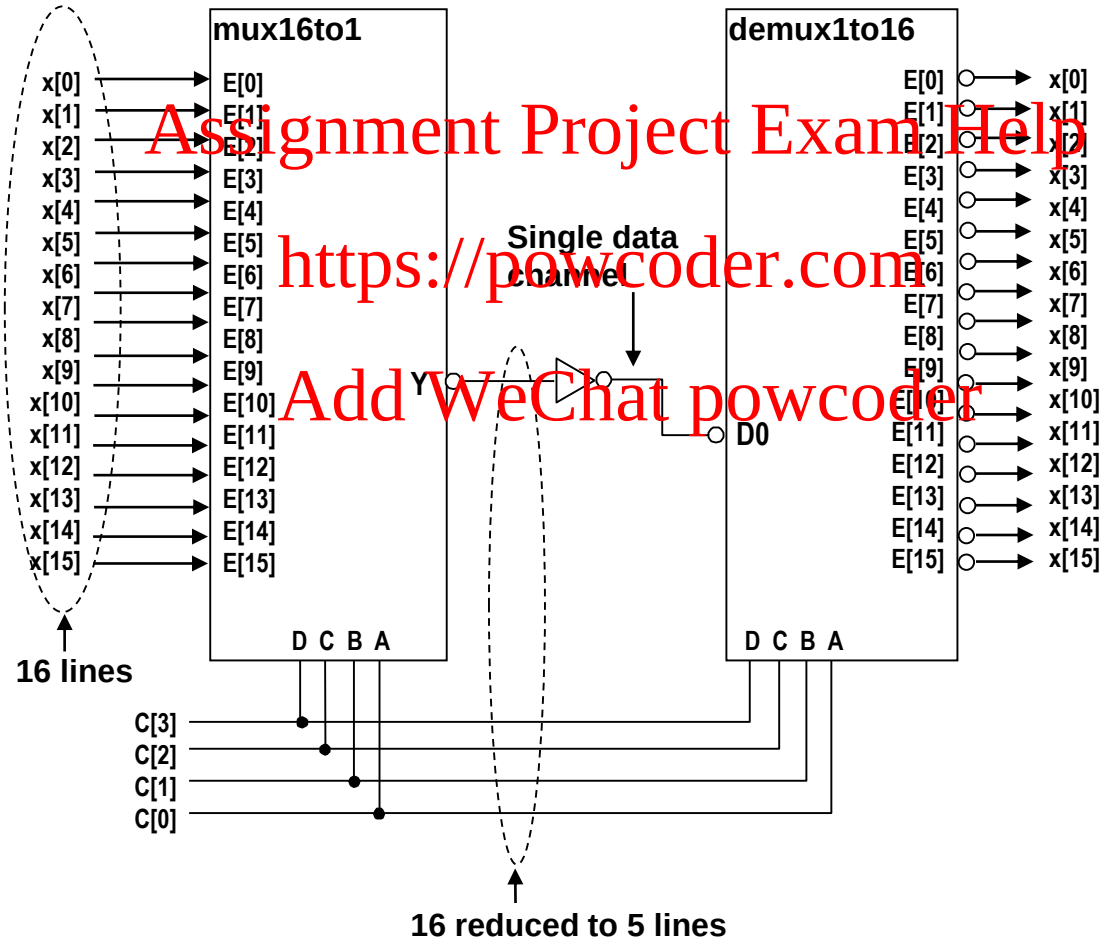


If not specified, other variables can be used as the select input

Application of Mux/Demux: Data Routing/Transmission*

- Data must be switched from multiple sources to a destination.
- Example

  Design a 16-line to 16-line mux/demux system.
  - The purpose of the system is to reduce the number of wires used for transmission



Adder

- Addition is one of the most common operation in digital systems.
- Adder is an arithmetic circuit that adds two n-bit binary numbers.
- Can be implemented in combinational or sequential logic.
- Terminology for addition operation:

$1111_2 + 01100_2 = 11010_2$

Augend **Addend** **Sum**

The carry bit 1 is called the carry output, C_{out} of the third column and the carry in, C_{in} to the fourth column of addition.

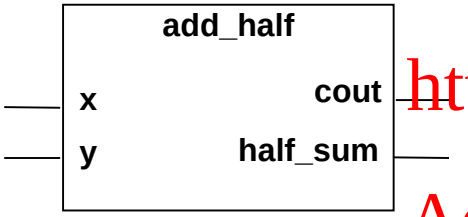
	1	1	1	1	0	Augend
	1←	1←	1←	0←	0←	0← Carries
+	0	1	1	0	0	Addend
	0	1	0	1	0	Sum

Half-Adder

- A half-adder is a logic circuit that performs a single bit addition. It adds two bits, x and y, and produces two outputs, a sum bit and a carry out bit.

- There is no carry-in input
  - carry-in is treated as 0.

- Block diagram:



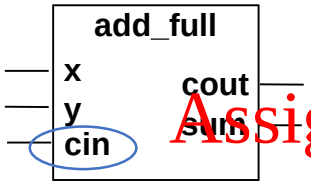
- Truth table:

x	y	cout	half_sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full-Adder

- The main difference between a full adder and half-adder is that a full adder accepts the carry in c_{in}.

- Block diagram:



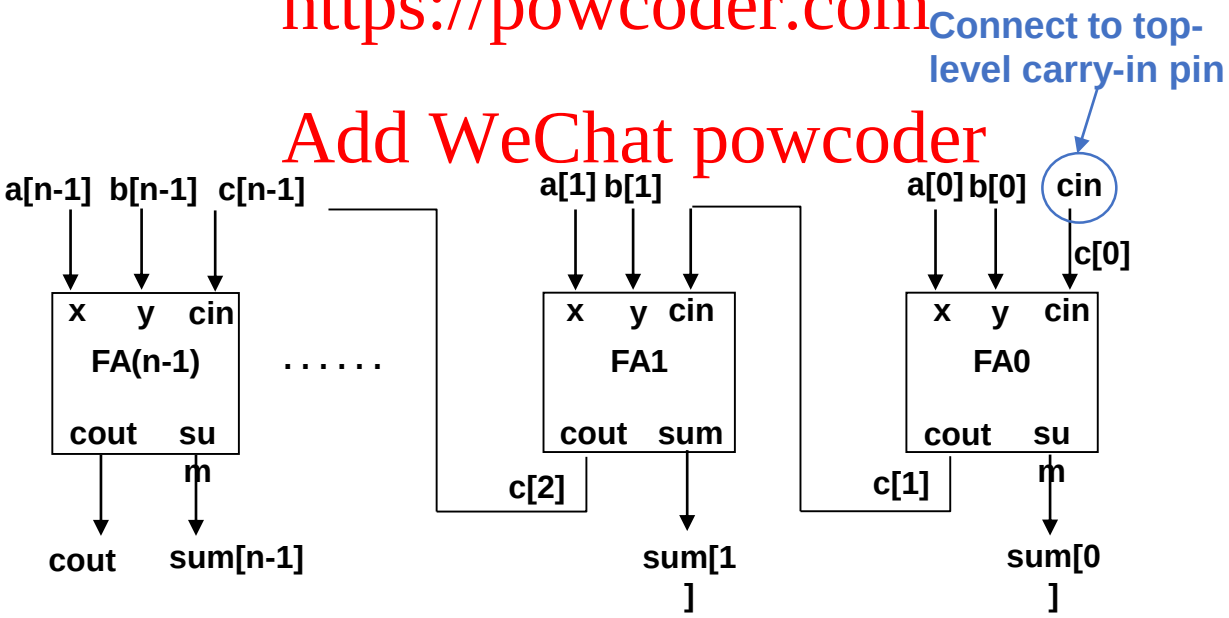
- Truth table:

cin	x	y	cout	sum
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

n-bit Adder: Carry-Ripple Adder (cra)

- n-bit carry-ripple adder adds 2 n-bit binary numbers (signed and unsigned)
  - Result is (n+1)-bit: n-bit sum and a carry-out.
  - Each bit position requires a full-adder

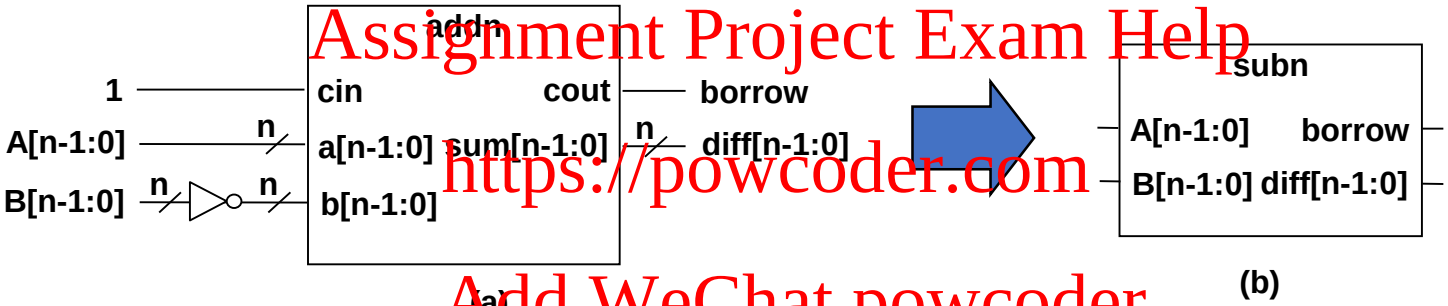
- Principle in constructing an n-bit carry-ripple adder
  - Cascading n full-adders (FA) through a chain of carry signals.
  - FA(i).cout => FA(i+1).cin where i = 0, 1, ... n-1



Subtracter Constructed from Adder

- Subtraction can be carried out using the adder circuit
  - Based on the 2's complement arithmetic

$$D_2 = A_2 - B_2$$
$$= A_2 + [B]_2$$
$$= A_2 + (B_2)' + 1$$

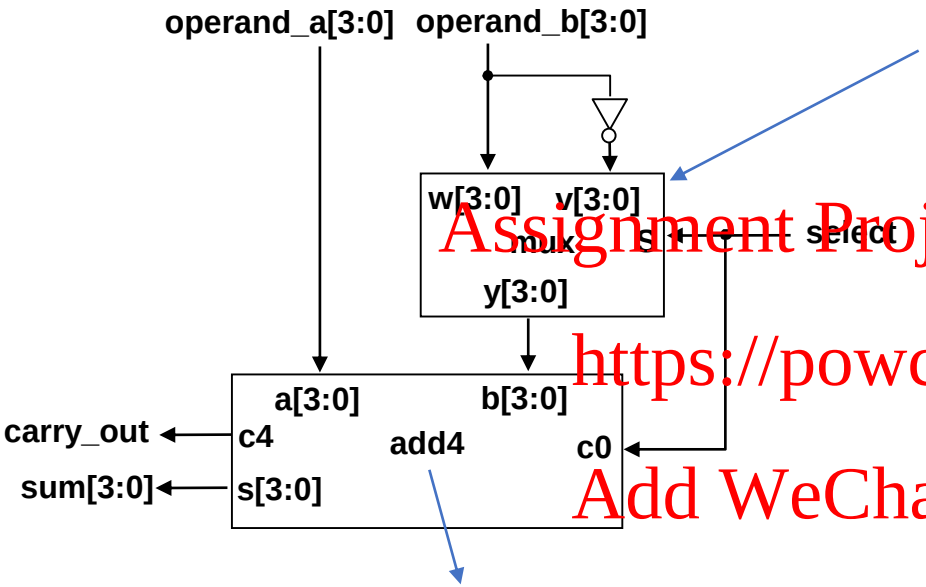


- However, the circuit can only perform one function i.e. subtraction.
  - Cannot perform addition.

- To perform both add and subtract functions, combine the adder and subtracter circuits into one.

Subtraction/Addition Implementation Based on 2's Complement Arithmetic

- Construction of binary adder used to perform both addition and subtraction.



Signals	Addition Operation	Subtraction Operation
select	select = 0 Mux output, y[3:0] = w[3:0] = operand_b[3:0]	select = 1 Mux output, y[3:0] = v[3:0] = operand_b[3:0]'
c0	c0 = select = 0	c0 = select = 1
sum[3:0]	sum[3:0] = a[3:0] + b[3:0] + c0 = operand_a[3:0] + y[3:0] + 0 = operand_a[3:0] + w[3:0] = operand_a[3:0] + operand_b[3:0]	sum[3:0] = a[3:0] + b[3:0] + c0 = operand_a[3:0] + y[3:0] + 1 = operand_a[3:0] + v[3:0] + 1 = operand_a[3:0] + operand_b[3:0]' + 1

Overflow

- Two's complement numbers (2cns) uses complement arithmetic.
  - E.g. (A – B) can be performed by computing A + (-B) = A + B_{2c}
    - Only need a binary adder and complementing circuits to handle both add and subtract.
  - Easy hardware implementation.

- One problem of digital systems is that it have limited bits for data representation (limited range of signed values).
  - The decimal range for an n-bit digital system using 2cns is $(-2^{n-1}) \leq N \leq (2^{n-1} - 1)$.
    E.g. an 8-bit digital system can only operate on values between -128 to 127.

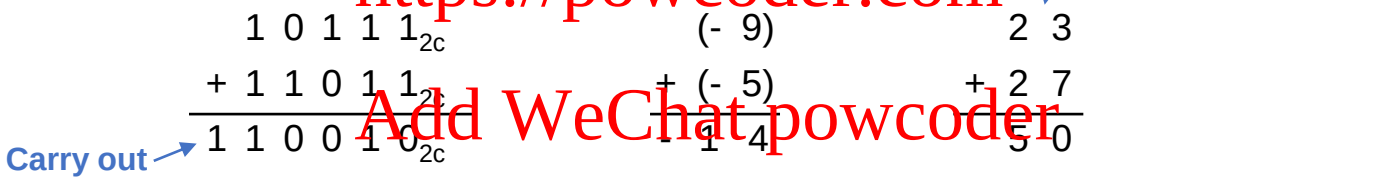
- If the result is out of the range, signal overflow occurs.
  - The out-of-range result cannot be used.
    - Should design digital system that generates a warning signal
    - So that invalid numbers are not mistaken for correct results.

Overflow: Signed and Unsigned Numbers

- The computation for signed and unsigned overflows are not the same.
- Example

Compute (– 9 – 5).

- 9 - 5 = (-9) + (-5)
= [01001] + [00101]
= 10111_{2c} + 11011_{2c}



- For unsigned arithmetic, carry out indicates an overflow.
- For signed numbers, the carry out from the sign-bit is not sufficient to determine the overflow.
  - It is meant for extending the adder circuit to higher order bits.
  - However, overflow can be detected by observing the carry into the sign-bit position and the carry out of the sign-bit position. If these two carries are not equal, an overflow has occurred.

Overflow: 2's Complement Arithmetic for Signed Numbers

- The following 3 case studies are useful for detecting and analysing overflow.
- Assume B and C are positive in a 5-bit system (range represented: -16 to 15).

Case study 1: A = B + C

Compute 12 + 7.

$$\begin{array}{r} 0\ 1\ 1\ 0\ 0_{2c} \\ +\ 0\ 0\ 1\ 1\ 1_{2c} \\ \hline 1\ 0\ 0\ 1\ 1_{2c} \end{array} \qquad \begin{array}{r} 1\ 2 \\ +\ \ 7 \\ \hline +\ 1\ 9 \end{array}$$

Result shows an incorrect sign bit (indicates overflow)

- Addition of two positive numbers produce a negative result
- $[10011]_{2c}$ = -01101_2 = -13
- -13 is incorrectly interpreted due to overflow
- Sum requires > 5 bits to represent it

Case study 2: A = - B - C

Compute (–12 – 5).

- 12 - 5 = (-12) + (-5)
  = [01100] + [00101]
  = 10100_{2c} + 11011_{2c}

$$\begin{array}{r} 1\ 0\ 1\ 0\ 0_{2c} \\ +\ 1\ 1\ 0\ 1\ 1_{2c} \\ \hline 1\ 0\ 1\ 1\ 1\ 1_{2c} \end{array} \qquad \begin{array}{r} (-1\ 2) \\ +\ (-\ \ 5) \\ \hline -\ \ 1\ \ 7 \end{array}$$

Carry discarded

Result shows an incorrect sign bit (indicates overflow)

- Addition of two negative numbers produce a positive result
- Incorrectly interpreted as 15 due to overflow

Overflow rule for addition:
If two 2's complement numbers with the same sign are added, then overflow occurs if:
- adding two positive numbers give a NEGATIVE result.
- adding two negative numbers give a POSITIVE result.

Overflow: 2's Complement Arithmetic for Signed Numbers

- **Case study 3: A = B - C**
  - The magnitude of (B - C) will always be less than either of the two numbers.
  - Overflow can never occur

Compute 12 - 5.

12 - 5 = 12 + (-5)

= 01100 + [00101]

= 01100_{2c} + 11011_{2c}

$$\begin{array}{r} 0\ 1\ 1\ 0\ 0_{2c} \\ +\ 1\ 1\ 0\ 1\ 1_{2c} \\ \hline 1\ 0\ 0\ 1\ 1\ 1_{2c} \end{array} \qquad \begin{array}{r} 12 \\ +\ (-5) \\ \hline 7 \end{array}$$

Carry discarded

The carry generated can be discarded

- It's not a valid carry
  - B > C shouldn't generate a carry