

Computer Organization and Architecture

Prepared by **Dr. Eng. Gamal Selim**

Lecture 10 : Revision



Computer Abstractions and Technology

Classes of Computers

- **Personal computers**
 - General purpose, variety of software
 - Subject to cost/performance tradeoff
- **Server computers**
 - Network based
 - High capacity, performance, reliability
 - Range from small servers to building sized

Classes of Computers

- **Supercomputers**
 - High-end scientific and engineering calculations
 - Highest capability but represent a small fraction of the overall computer market
- **Embedded computers**
 - Hidden as components of systems
 - Stringent power/performance/cost constraints

Classes of Computers

- **Personal Mobile Device (PMD)**
 - Battery operated
 - Connects to the Internet
 - Hundreds of dollars
 - Smart phones, tablets, electronic glasses
- **Cloud computing**
 - Warehouse Scale Computers (WSC)
 - Software as a Service (SaaS)
 - Portion of software run on a PMD and a portion run in the Cloud
 - Amazon and Google

Instructions: Language of the Computer

Instruction Set

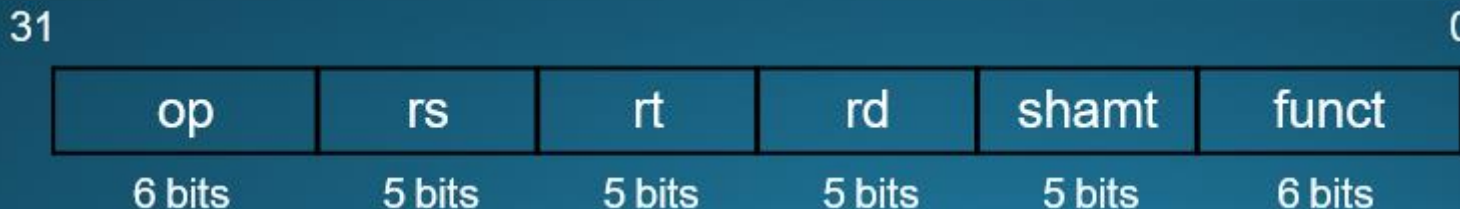
- The collection of instructions of a computer
- Different computers have different instruction sets
 - But with many aspects in common
- Early computers had very simple instruction sets
 - Simplified implementation
- Many modern computers also have simple instruction sets

Instruction Set Architecture

- Anything **programmers** need to know to make a binary machine language program work correctly, including:
 - Registers
 - Organization of programmable storage
 - Data types and data structures: encoding and representations
 - Instruction set
 - Instruction formats
 - Modes of addressing: accessing data items and instructions
 - Exceptional conditions

Arithmetic Operations

- Add and subtract, three **operands**
- Two sources and one destination
- add a, b, c # a gets $b + c$
- Operation, operator, operand
- All arithmetic operations have this form
- *Design Principle 1: Simplicity favors regularity*
 - **Regularity** makes implementation simpler
 - **Simplicity** enables higher performance at lower cost



Arithmetic Example

- **C code:**
 - $f = (g + h) - (i + j);$
- **Compiled MIPS code:**
 - add **t0**, g, h # temp t0 = g + h
 - add **t1**, i, j # temp t1 = i + j
 - sub f, t0, t1 # f = t0 - t1

Register Operands

- Arithmetic instructions use operands from **registers**
- The size of a register in MIPS architecture is 32 bits, which is a **word**
- The **word** is the natural **unit of access** in a computer, usually a group of 32 bits; **corresponds to the size of a register**

Register Operands

- MIPS has a 32×32 -bit **register file**
 - Use for frequently accessed data
 - Cache of **cache**?
- Numbered 0 to 31
- Assembler names
 - \$t0, \$t1, ..., \$t9 for **temporary** values
 - \$s0, \$s1, ..., \$s7 for **saved** variables
 - **\$zero**
- *Design Principle 2: Smaller is faster Cf. main memory: millions of locations*

Register Operand Example

- C code:
 - $f = (g + h) - (i + j);$
 - $f, \dots, j$ in $\$s0, \dots, \$s4$
- Compiled MIPS code:

```
add    $t0,    $s1,    $s2
add    $t1,    $s3,    $s4
sub    $s0,    $t0,    $t1
```

Memory Operands

- **Data transfer instruction:** before arithmetic operations can be done
 - **Load** values from memory into registers
 - **Store** result from register to memory
 - **RISC** (Reduced instruction set computing) vs. CISC
 - Only *load* and *store* instructions access memory
- Memory is **byte addressed**
 - Each address identifies an 8-bit byte
- **Alignment restriction:** words are aligned in memory
 - For lw/sw, the address must be a multiple of 4
 - If either of the **two least-significant bits** of the address are non-zero, an **Address Error exception** occurs

Memory Operand Example 1

- **C code:**
 - $g = h + A[8];$
 - g in $\$s1$, h in $\$s2$, base address of A in $\$s3$
- **Compiled MIPS code:**
 - Index **8** requires offset of 32
 - 4 bytes per word

```
lw    $t0, 32($s3)    # load word
add   $s1, $s2, $t0
```

offset

base register

Memory Operand Example 2

- **C code:**
 - $A[12] = h + A[8];$
 - h in \$s2, base address of A in \$s3

- **Compiled MIPS code:**

- Index 8 requires offset of 32

lw	\$t0,	32(\$s3)	# load	word
add	\$t0,	\$s2, \$t0		
sw	\$t0,	48(\$s3)	# store	word

Arithmetic for Computers

- Operations on integers
 - Addition and subtraction
 - Multiplication and division
- **ALU: Arithmetic Logic Unit**
 - HW performs addition, subtraction, and usually logical operations such as AND and OR

Addition in Binary

Addition in Binary

- In decimal if you add two digits and the sum is more than the base (10) you add all the 10s in the result to the next higher digit and leave the remainder in the current digit.
- The rule is the same for integer and decimal fractions.
- In binary this rule applies as :
 - $1+0 = 1$
 - $1+1 =$
 - 10 (as no 2 is allowed in binary)

Problem 1

Add the following in binary and check results by doing addition in decimal:

(b) $1111 + 0011$

Solution:

$$\begin{array}{rcl}
 (15)_{10}: & (1111)_2 & \\
 (3)_{10}: & (0011)_2 & + \\
 \hline
 (18)_{10}: & (10010)_2 &
 \end{array}$$

(c) $1011.1101 + 11.1$

Solution:

$$\begin{array}{rcl}
 (11.8125)_{10}: & (1011.1101)_2 & \\
 (3.5)_{10}: & (0011.1000)_2 & + \\
 \hline
 (15.3125)_{10}: & (1111.0101)_2 &
 \end{array}$$

(e) $1111000 + 00010010$

Solution:

$$\begin{array}{rcl}
 (120)_{10}: & (1111000)_2 & \\
 (18)_{10}: & (0010010)_2 & + \\
 \hline
 (138)_{10}: & (10001010)_2 &
 \end{array}$$

Signed Numbers

Signed number representation in binary

- We have two ways of representing signed numbers:
 1. Sign-magnitude system.
 2. 2's complement system

Problem 2

Repeat each of the following signed decimal numbers in 2's-complement system. Use a total of 8 bits including sign bit

(a)+32

Solution:

The representation of 32 in binary is 100000

In 8 bits = **00**100000

Problem 3

Each of the following numbers represents a signed decimal number in the 2's-complement system. determine the decimal value in each case. (Hint: Use negative to convert negative numbers to positive)

(a) 01101

Solution:

The sign bit = 0

Then the number is positive and the value =
 $8+4+1 = 13$

Problem 5

Represent each of the following decimal values as 5-bit signed magnitude binary values. Then negate each one.

(b) -12

Solution:

-12 =

11100

The negative of -12 = 01100

Subtraction in Binary

Problem 6

Perform the following operation in 2's-complement system. Use 8-bit (including the sign bit) for each number. Check your results by converting the binary result back to decimal.

(a) Add +19 to -24

$$(24)_{10} = (00011000)_2$$

$$(19)_{10}: \quad (00010011)_2$$

$$(-24)_{10}: \quad (11101000)_2$$

$$(-5)_{10}: \quad 11111011$$

$$(5)_{10}: \quad (00000101)_2$$

+

(b) Add -48 to -80

$$(48)_{10} = (00110000)_2$$

$$(80)_{10} = (01010000)_2$$

$$(-48)_{10}: \quad \overset{1}{(11010000)_2}$$

$$(-80)_{10}: \quad (10110000)_2$$

+

$$(-128)_{10}: \quad 1 \ 10000000$$

$$(128)_{10}: \quad (10000000)_2$$

Multiplication in Binary

Problem 7

Multiply the following pairs of binary numbers and check by doing multiplication in decimal.

(b) 1011×1011

Solution:

$$(11)_{10}: (1011)_2$$

$$(11)_{10}: (1011)_2$$

×

$$\begin{array}{r} 1011 \\ 1011 \\ 0000 \\ 1011 \\ \hline \end{array}$$

$$(121)_{10}: 111\underline{1001}$$

(d) 11111×101

Solution:

$$(31)_{10}: (11111)_2$$

$$(5)_{10}: (101)_2$$

×

$$\begin{array}{r} 11111 \\ 00000 \\ 11111 \\ \hline \end{array}$$

$$(155)_{10}: 1001\underline{1011}$$

Division in Binary

Problem 9

Perform the following divisions. Check the results by doing division in decimal.

(a) $111111/1001$

Solution

$$\begin{array}{r}
 000111 \text{ (7)}_{10} \\
 (1001)_2 \overline{) (111111)_2} \\
 \underline{1001} \\
 01101 \\
 \underline{1001} \\
 01001 \\
 \underline{1001} \\
 0000
 \end{array}$$

Basic Theorem of Boolean Algebra

Basic Theorem of Boolean Algebra

T₁ : Properties of 0

$$(a) 0 + A = A$$

$$(b) 0 A = 0$$

T₂ : Properties of 1

$$(a) 1 + A = 1$$

$$(b) 1 A = A$$

T₃ : Commutative Law

$$(a) A + B = B + A$$

$$(b) A B = B A$$

T₄ : Associate Law

$$(a) (A + B) + C = A + (B + C)$$

$$(b) (A B) C = A (B C)$$

T₅ : Distributive Law

$$(a) A (B + C) = A B + A C$$

$$(b) A + (B C) = (A + B) (A + C)$$

$$(c) A + A'B = A + B$$

T₆ : Indempotence (Identity) Law

$$(a) A + A = A$$

$$(b) A A = A$$

T₇ : Absorption (Redundance) Law

$$(a) A + A B = A$$

$$(b) A (A + B) = A$$

T₈ : Complementary Law

$$(a) X + X' = 1$$

$$(b) X.X' = 0$$

T₉ : Involution

$$(a) x'' = x$$

Problem 1

- Simplify the following expression using Boolean Algebra:

(a) $\overline{ABC} + \overline{A}BC + ABC + A\overline{B}\overline{C} + A\overline{B}C$

Solution:

$$= \overline{A} + \overline{B} + \overline{C} + BC(\overline{A} + A) + A(\overline{B} + \overline{C}) + A\overline{B}C$$

$$= \overline{A} + \overline{B} + \overline{C} + BC + A\overline{B} + A\overline{C} + A\overline{B}C$$

$$= \overline{A} + (\overline{B} + A\overline{B} + A\overline{B}C) + (\overline{C} + A\overline{C}) + BC$$

$$= \overline{A} + \overline{B}(1 + A + AC) + \overline{C}(1 + A) + BC$$

$$= \overline{A} + \overline{B} + (\overline{C} + BC)$$

$$= \overline{A} + \overline{B} + \overline{C} + B$$

$$= \overline{A} + 1 + \overline{C} = 1$$

Another solution :

Replacing ABC by X

$$\overline{X} + \overline{A}BC + X + A\overline{B}\overline{C} + A\overline{B}C$$

$$= 1 + \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C = 1$$

Adders

Half Adder

- Adding two single-bit binary values, X, Y produces a sum S bit and a carry out C-out bit.
- This operation is called half addition and the circuit to realize it is called a half adder.

Half Adder Truth Table

Inputs		Outputs	
X	Y	S	C-out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S(X, Y) = \Sigma (1, 2)$$

$$S = X'Y + XY'$$

$$S = X \oplus Y$$

$$C\text{-out}(x, y, C\text{-in}) = \Sigma (3)$$

$$C\text{-out} = XY$$



Full Adder

- A combinational circuit that adds 3 input bits to generate a Sum bit and a Carry bit

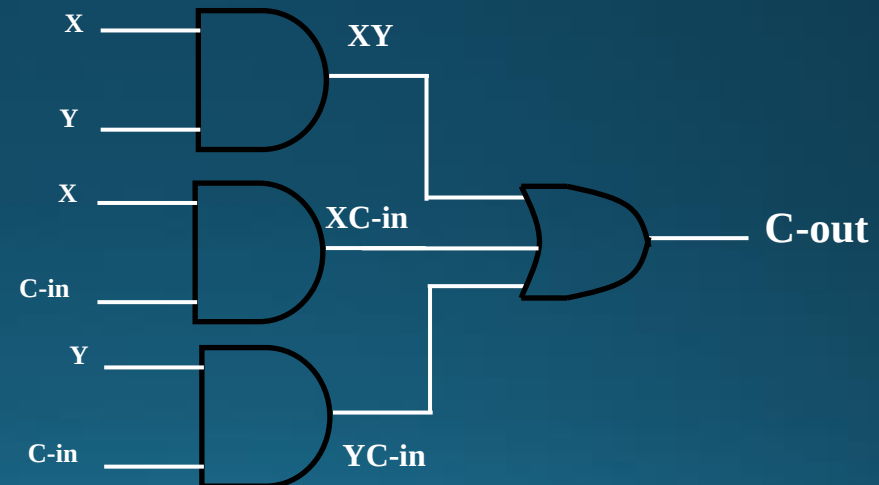
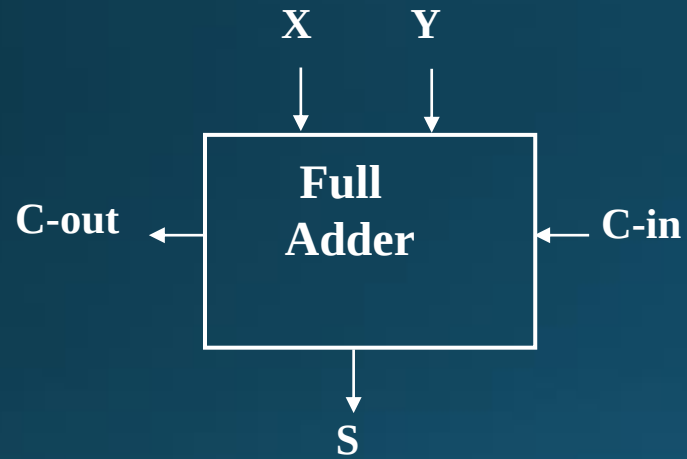
- A truth table and sum of minterm equations for C and S are shown below.

	X	Y	Z	C	S
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
$0 + 1 + 1 = 10 \rightarrow$	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0
$1 + 1 + 1 = 11 \rightarrow$	1	1	1	1	1

$$C(X,Y,Z) = \sum m(3,5,6,7)$$

$$S(X,Y,Z) = \sum m(1,2,4,7)$$

Full Adder Circuit Using XOR



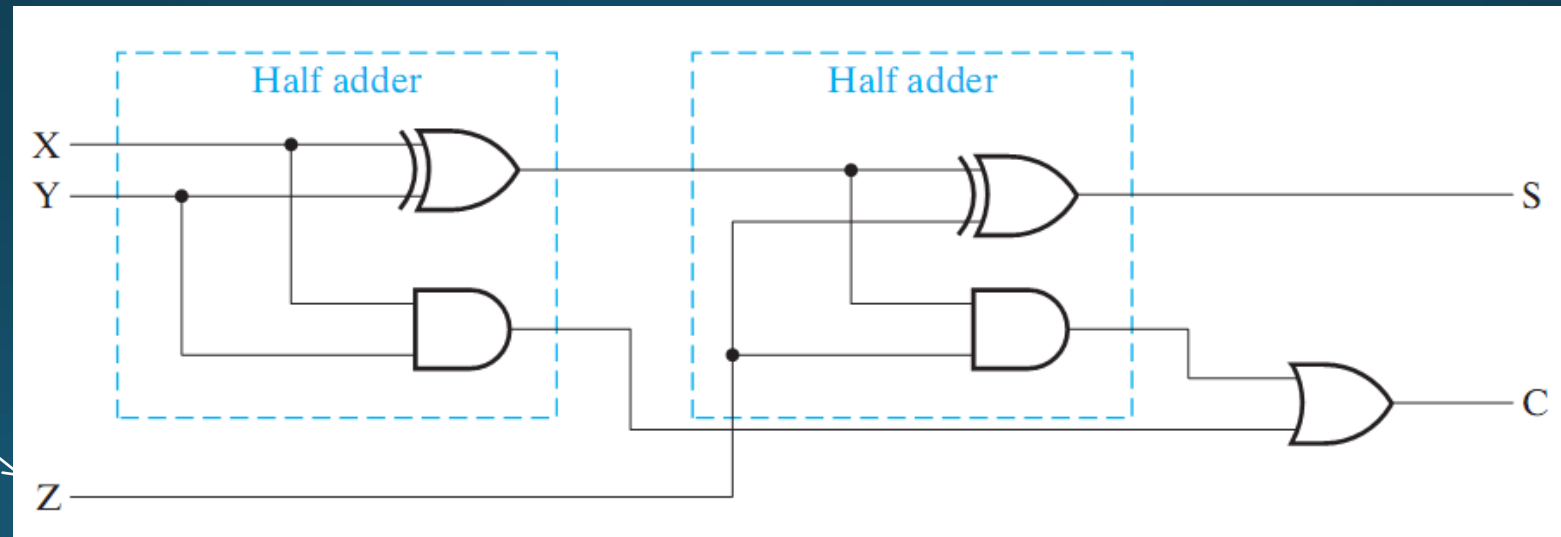
Full Adder = 2 Half Adders

Manipulating the Equations:

$$S = (X \oplus Y) \oplus Z$$

$$C = XY + XZ + YZ = XY + Z(X \oplus Y)$$

Think of
Z as a
carry in



SRC: Mano's BOOK

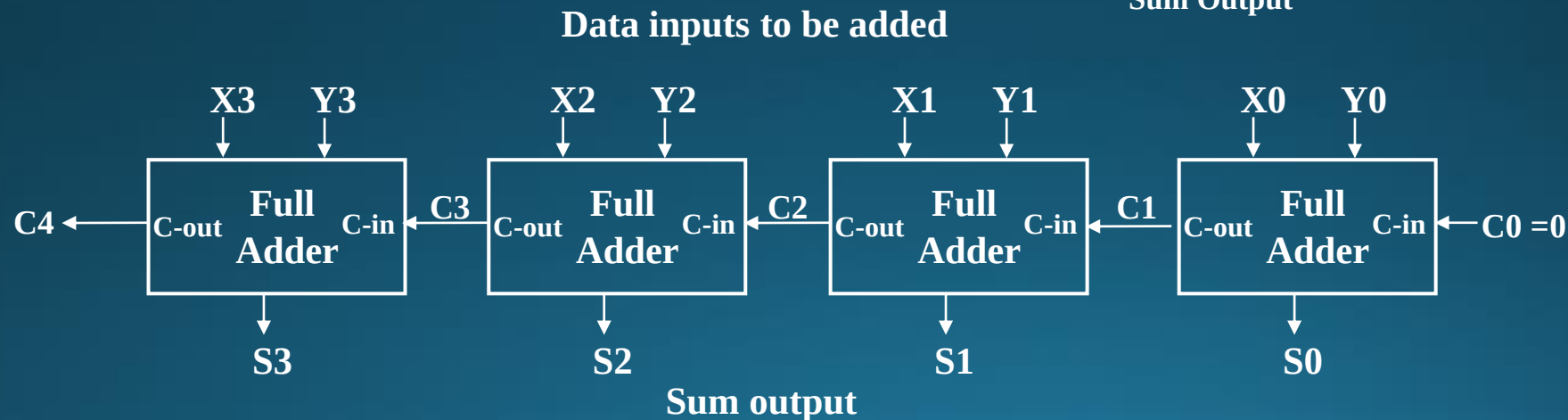
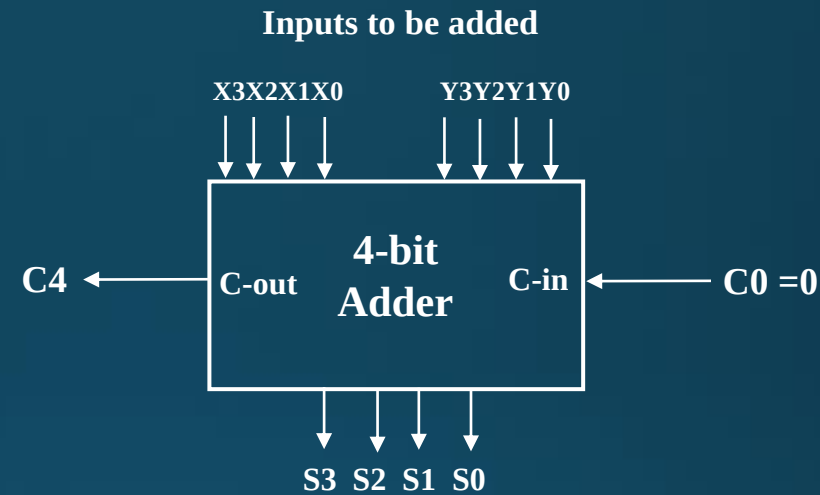
4-bit Carry Ripple Adder

Adds two 4-bit numbers:

$X = X_3 \ X_2 \ X_1 \ X_0$

$Y = Y_3 \ Y_2 \ Y_1 \ Y_0$

producing the sum $S = S_3 \ S_2 \ S_1 \ S_0$,
 $C\text{-out} = C_4$ from the most significant position $j=3$



Subtractors

Half Subtractor

- Subtracting a single-bit binary value Y from another X (I.e. $X - Y$) produces a difference bit D and a borrow out bit $B\text{-out}$.
- This operation is called half subtraction and the circuit to realize it is called a half subtractor.

Half Subtractor Truth Table

Inputs		Outputs	
X	Y	D	$B\text{-out}$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



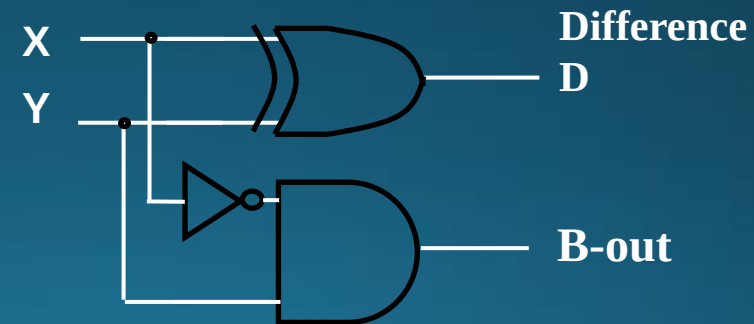
$$D(X, Y) = \Sigma (1, 2)$$

$$D = X'Y + XY'$$

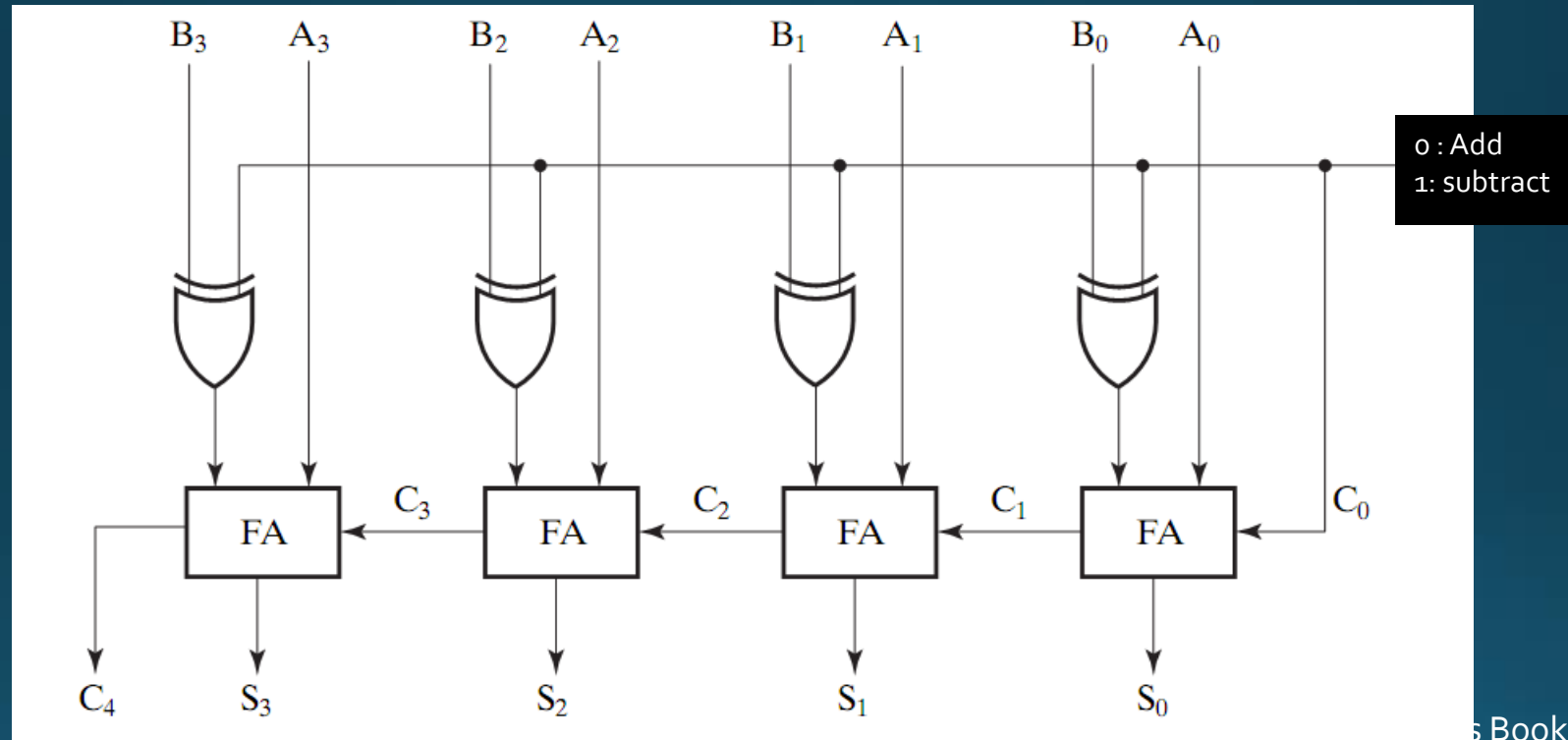
$$D = X \oplus Y$$

$$B\text{-out}(x, y, C\text{-in}) = \Sigma (1)$$

$$B\text{-out} = X'Y$$



Adder/Subtractor



Using full adders and XOR we can build an Adder/Subtractor!

Decoders

Decoder

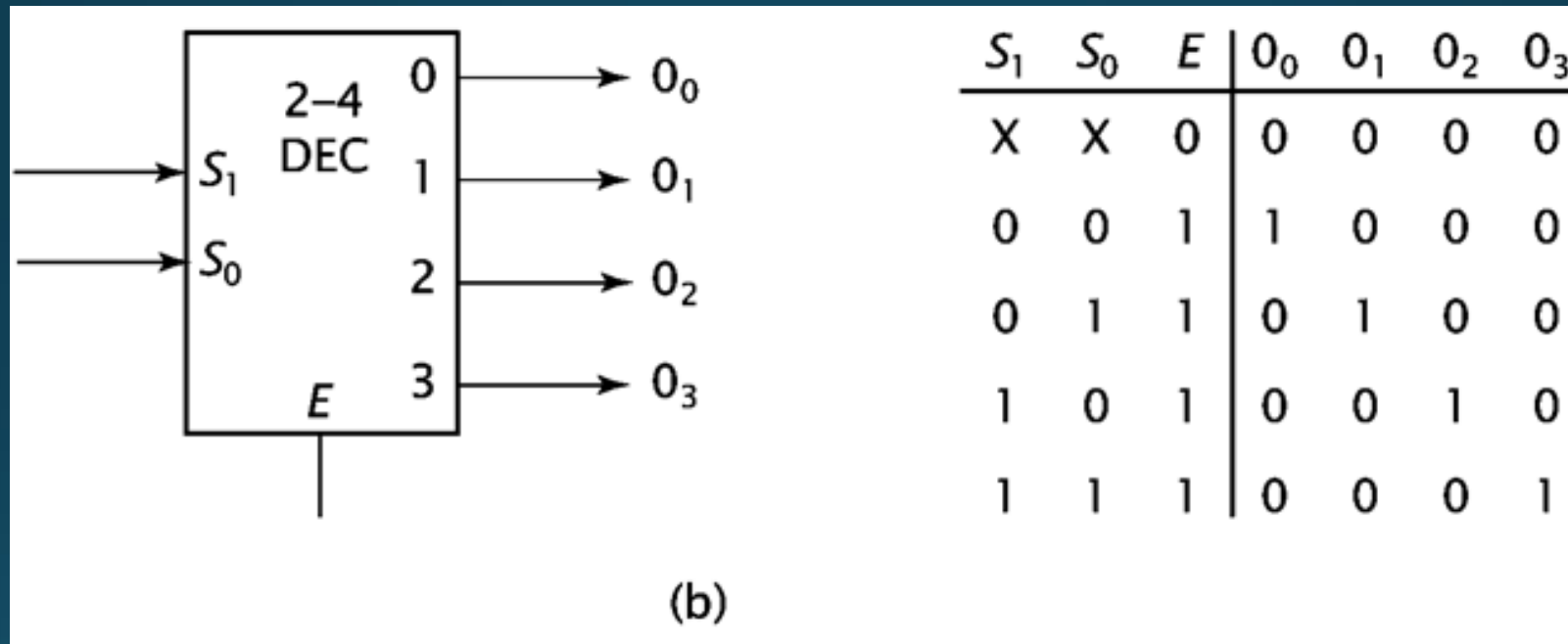
Accepts a value and decodes it

- Output corresponds to value of n inputs

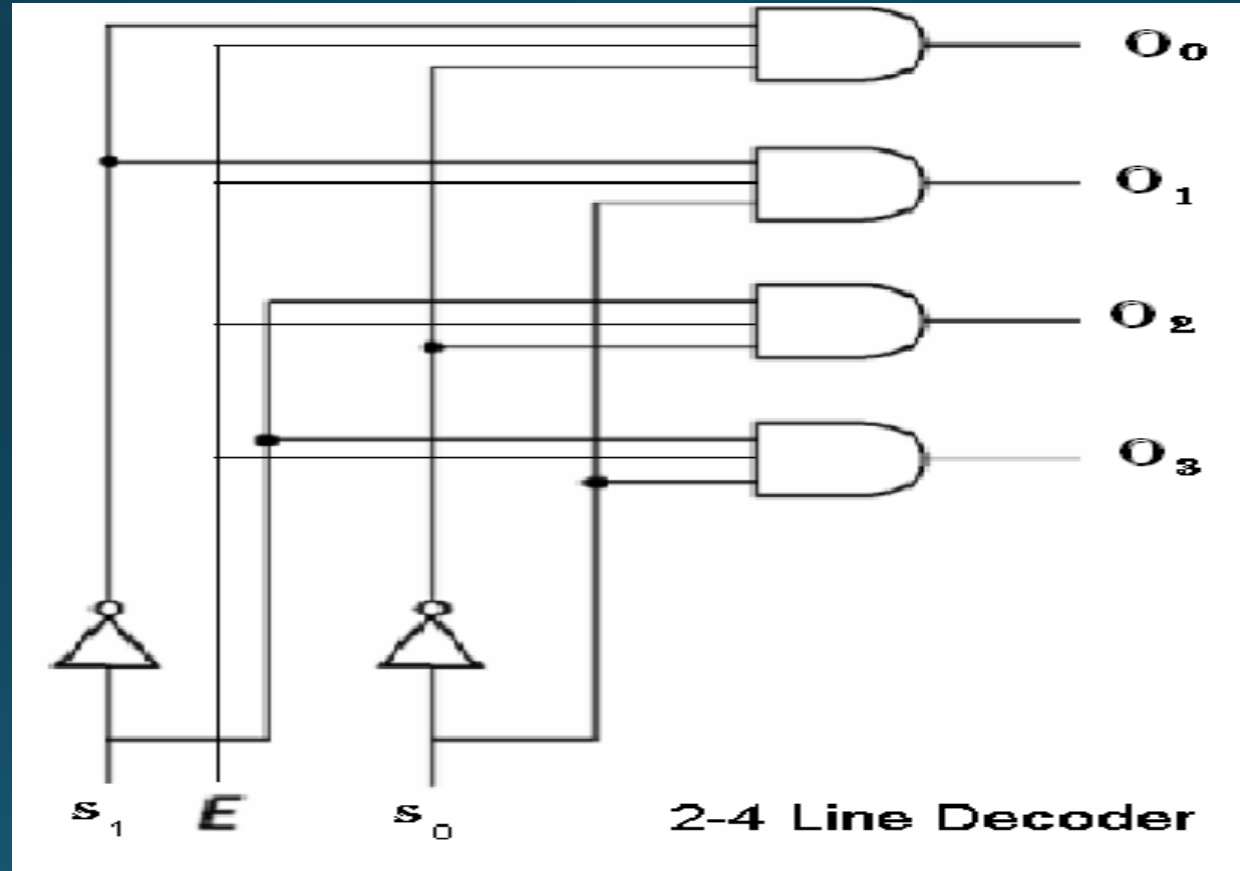
Consists of:

- Inputs (n)
- Outputs (2^n , numbered from $0 \rightarrow 2^n - 1$)
- Selectors / Enable (active high or active low)

2-to-4 Decoder

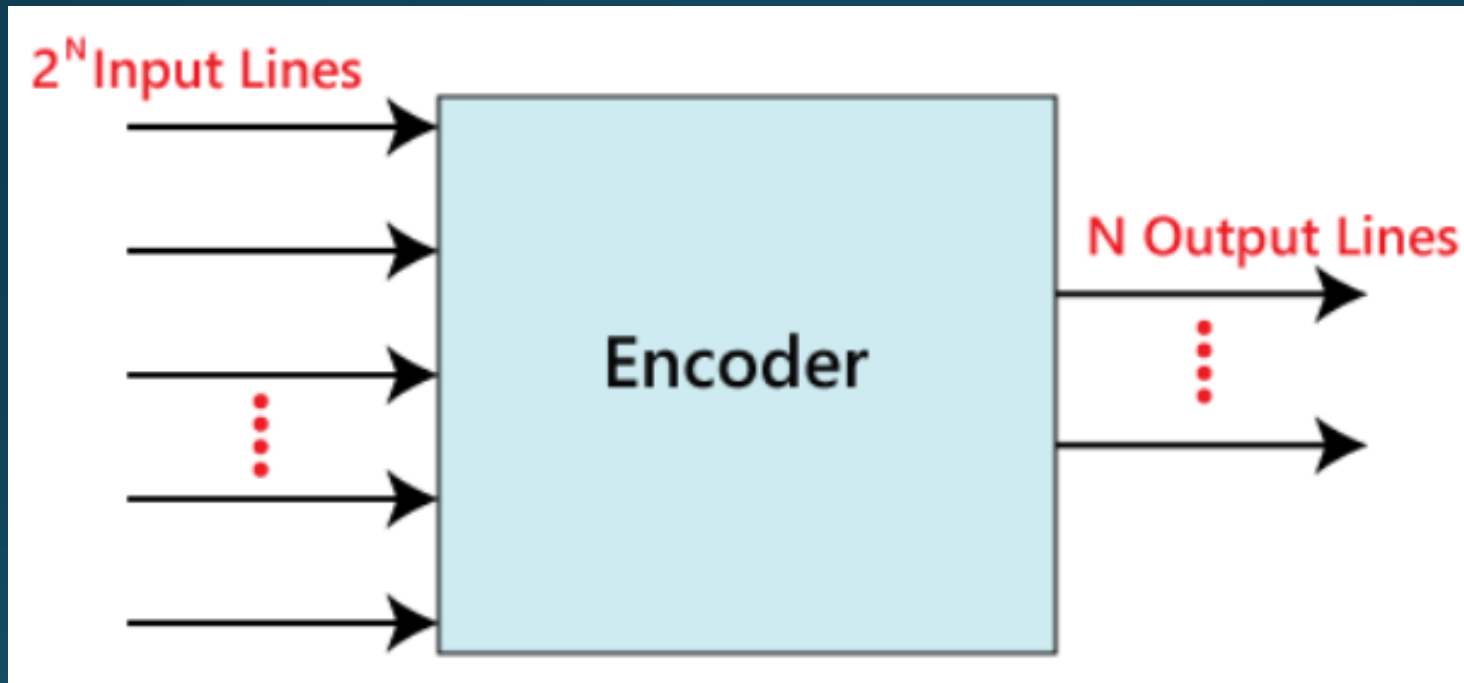


2-to-4 Decoder



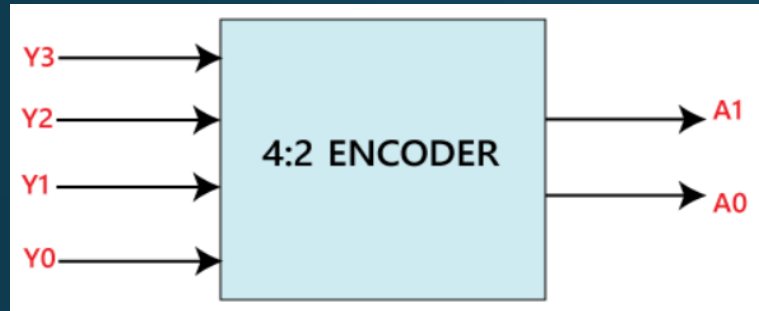
Encoders

Encoders



4-to-2 line Encoder

4 to 2 line Encoder

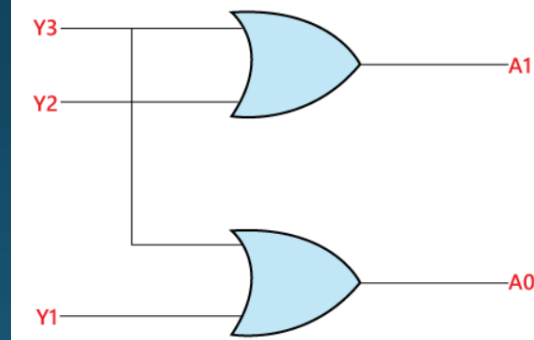


The logical expression of the term A_0 and A_1 is as follows:

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

Logical circuit of the above expressions is given below:



Truth Table:

INPUTS				OUTPUTS	
Y_3	Y_2	Y_1	Y_0	A_1	A_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



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY



Thank You

