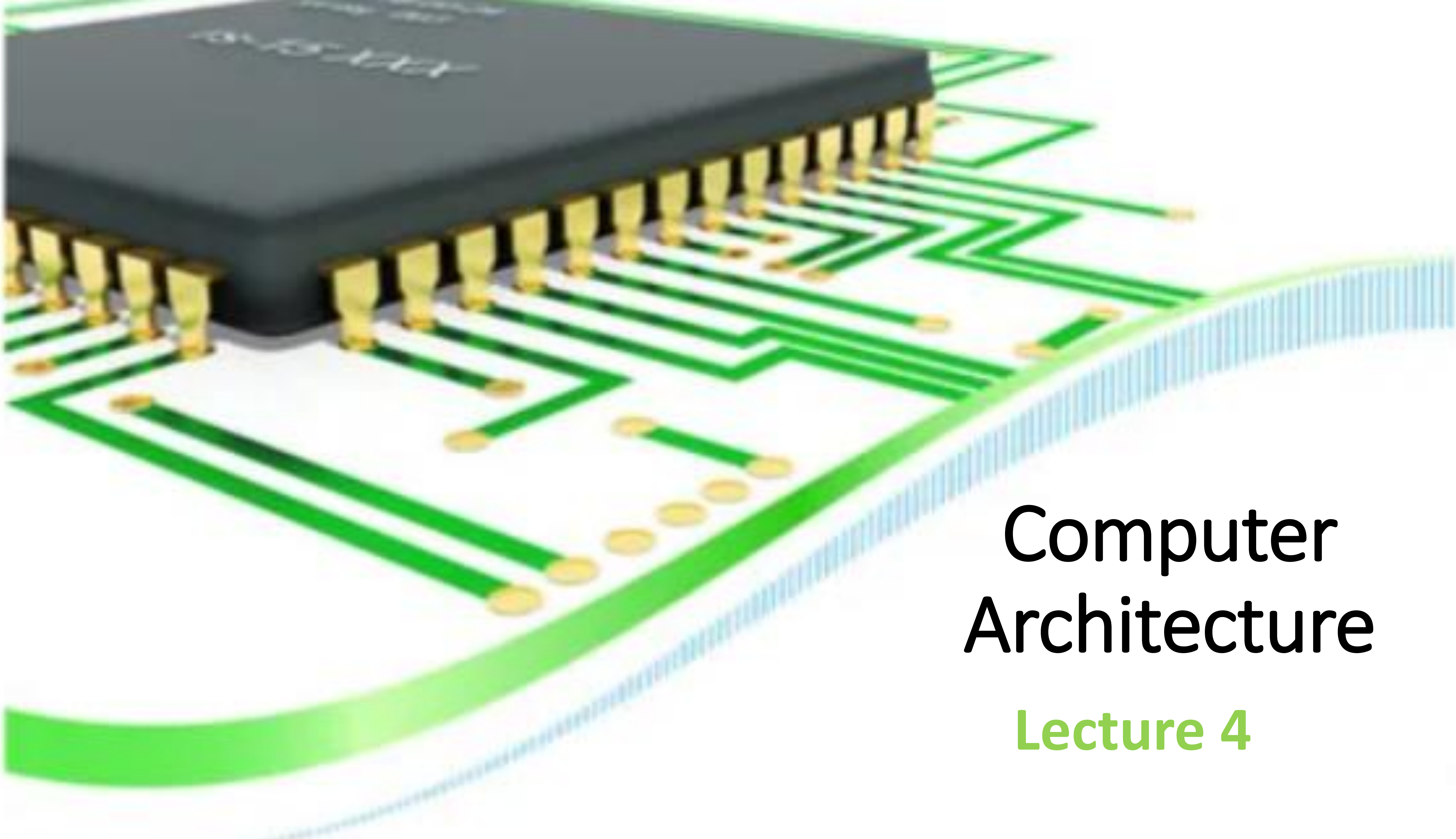


Computer Architecture

Lecture 4

Agenda

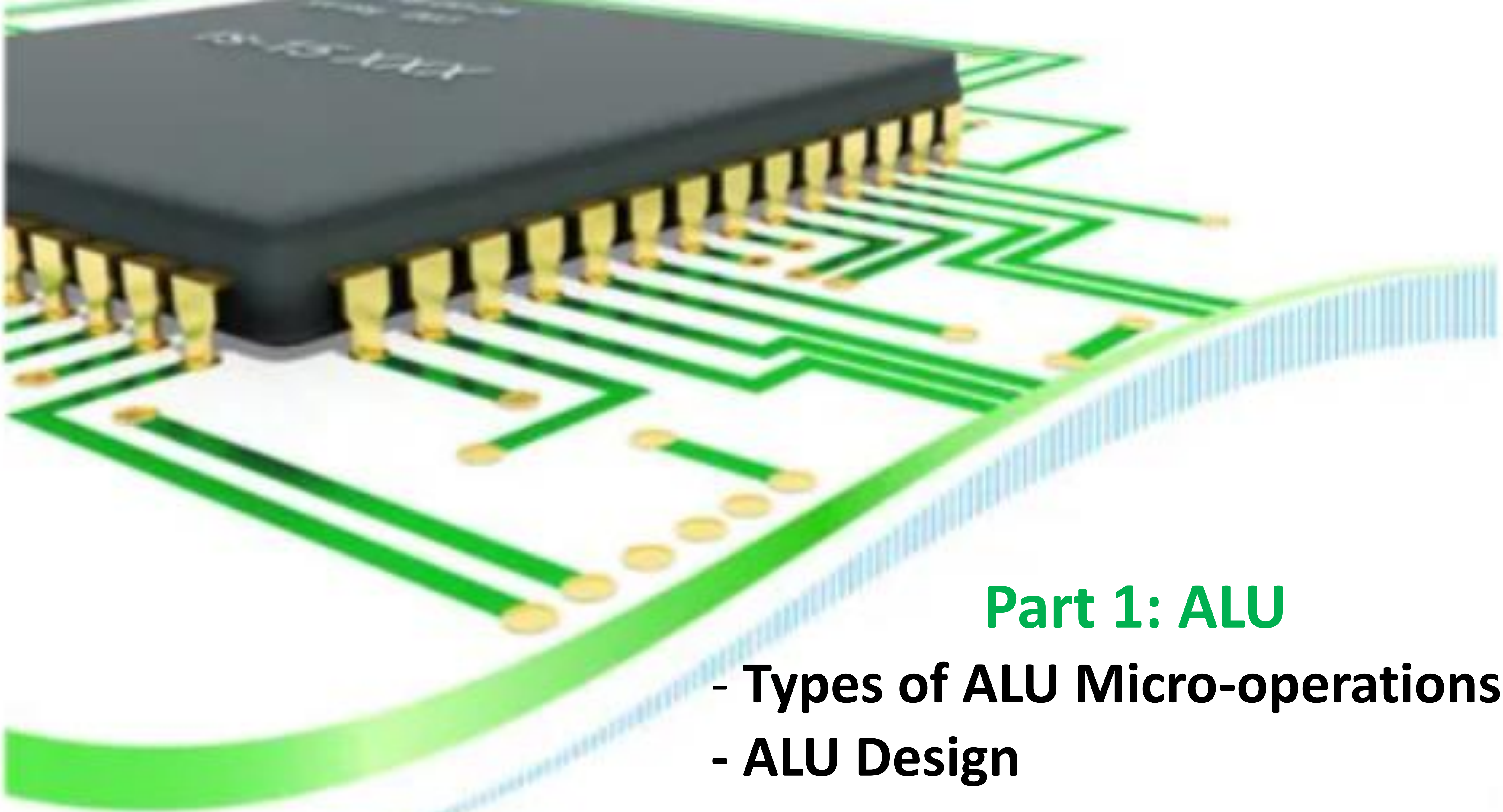
- **Part 1: ALU**

  - Types of ALU Micro-operations

  - ALU Design

- **Part 2: Instruction Set Architecture**

  - MIPS Instruction Set Architecture



Part 1: ALU

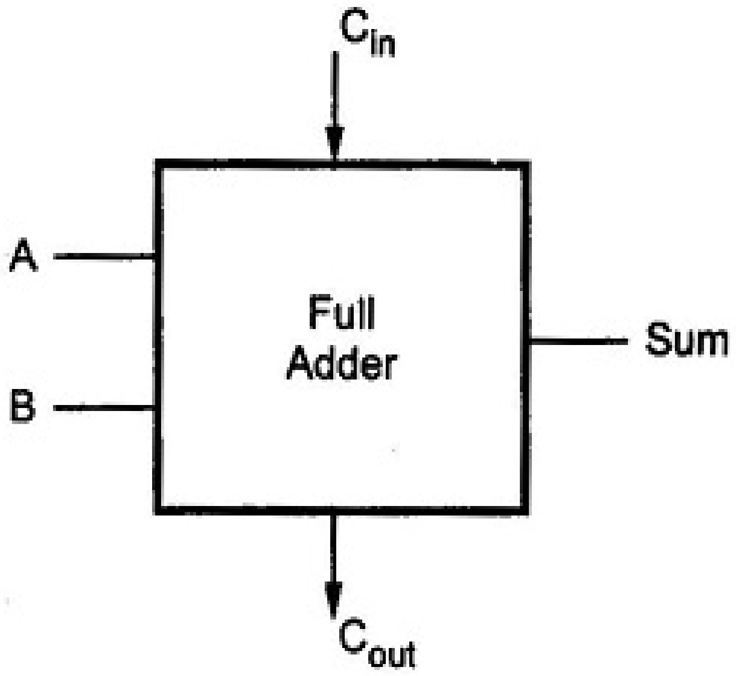
- Types of ALU Micro-operations
- ALU Design

Types of ALU Micro-operations

- Types of ALU micro-operations
  - Data Transfer
  - Arithmetic
  - Logical
  - Shift

D ←A

Full adder block diagram



Inputs			Outputs	
A	B	C_{in}	Carry	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

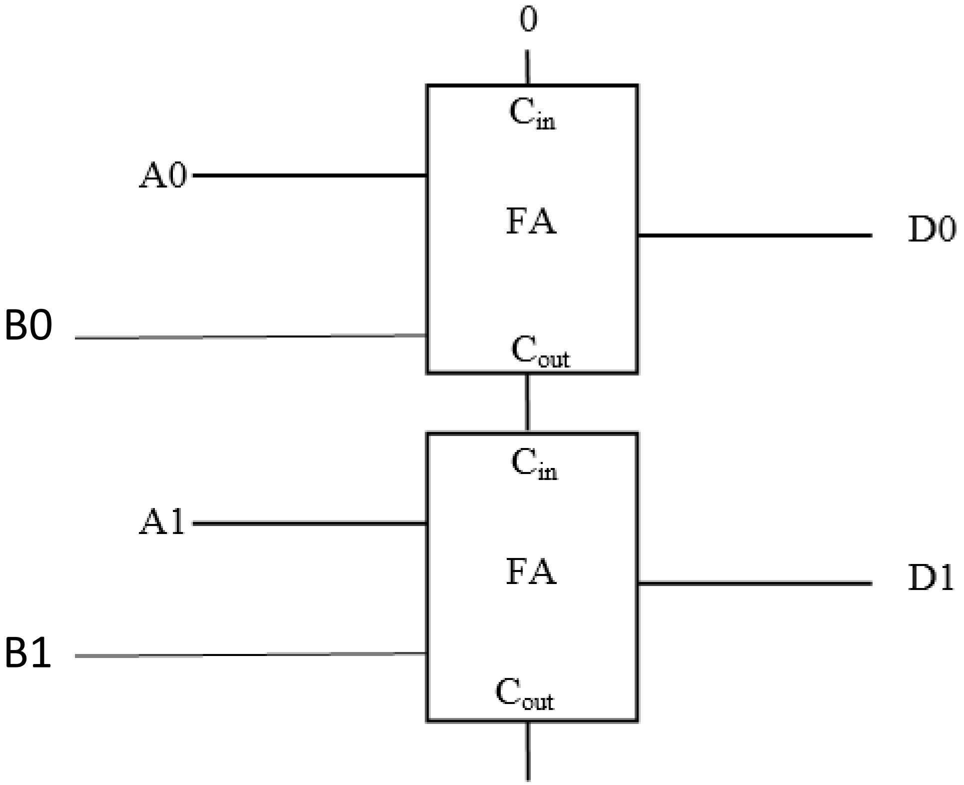
Truth Table of Full Adder

Types of ALU Micro-operations (Arithmetic micro-operations)

- Addition:

  D ← A+ B

```
          1
A      0011
+ B    0010
       --------
D      0101
```



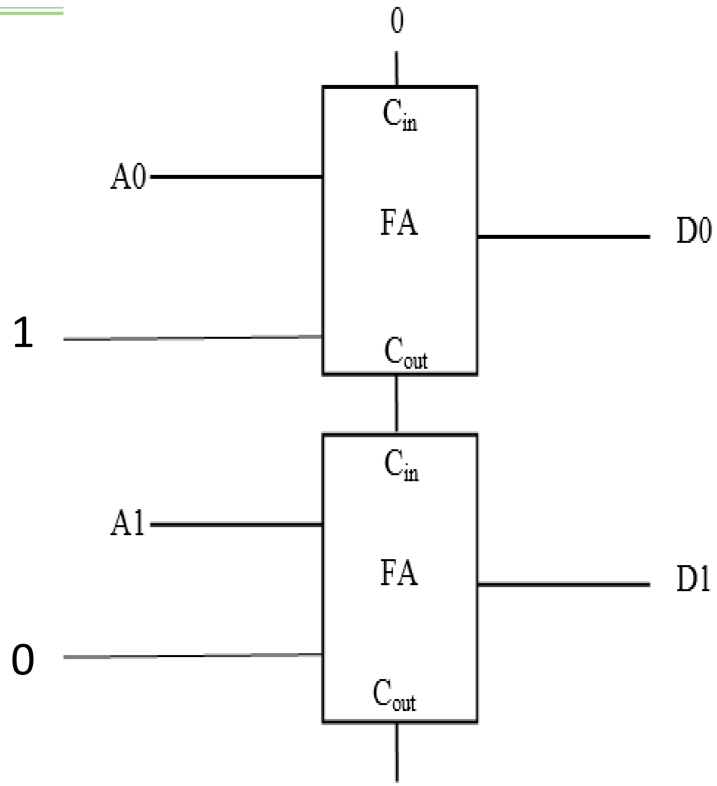
Part 3: Types of ALU Micro-operations (Arithmetic micro-operations)

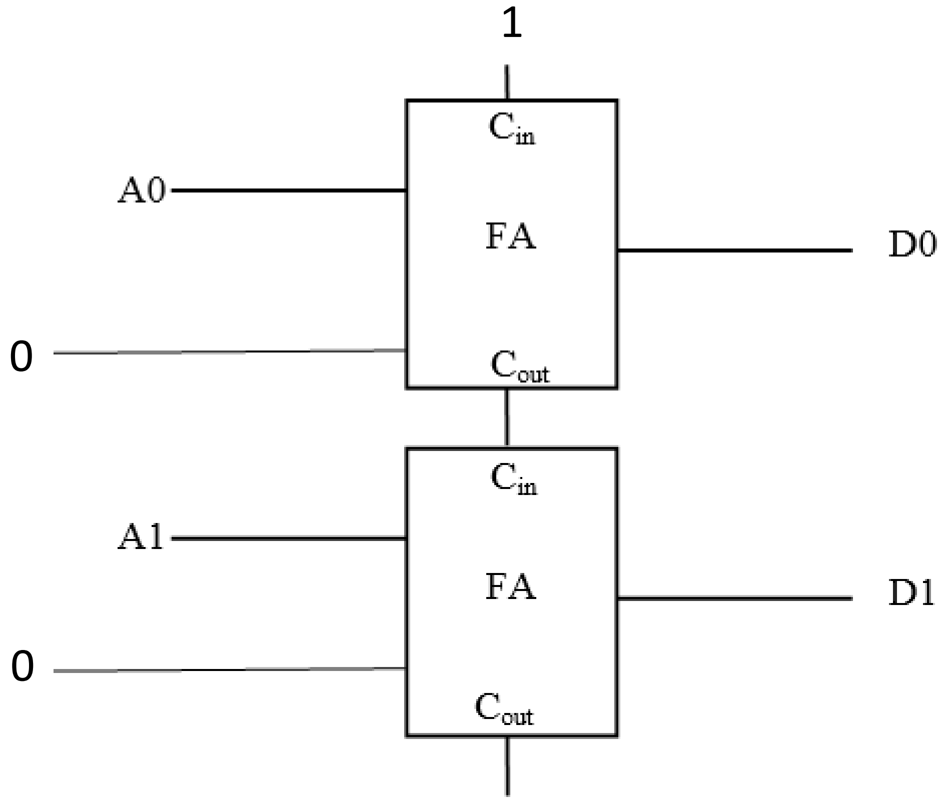
- Increment:

D ← A+ 1

A	0011
	0001

D	0100





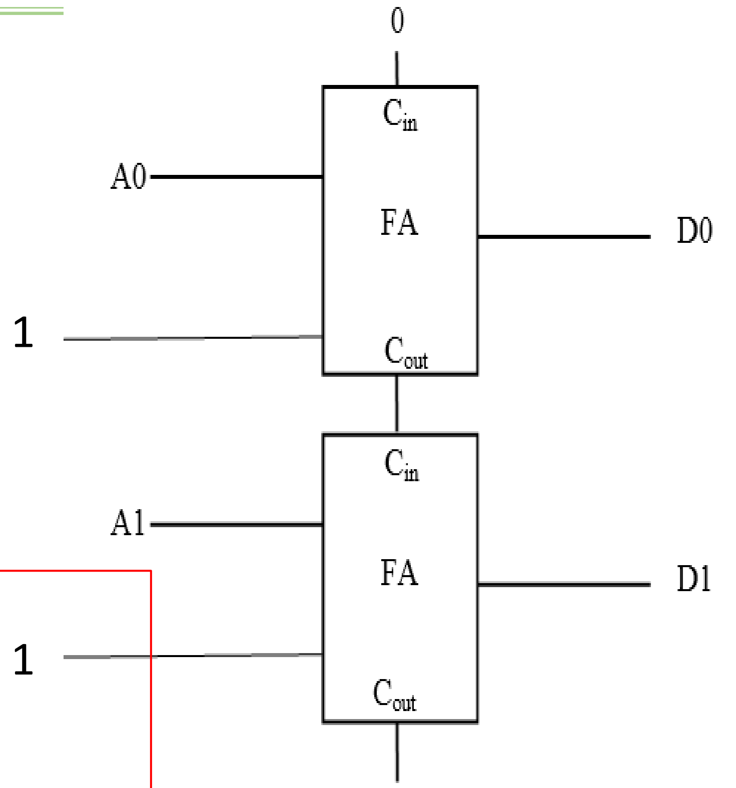
Part 3: Types of ALU Micro-operations (Arithmetic micro-operations)

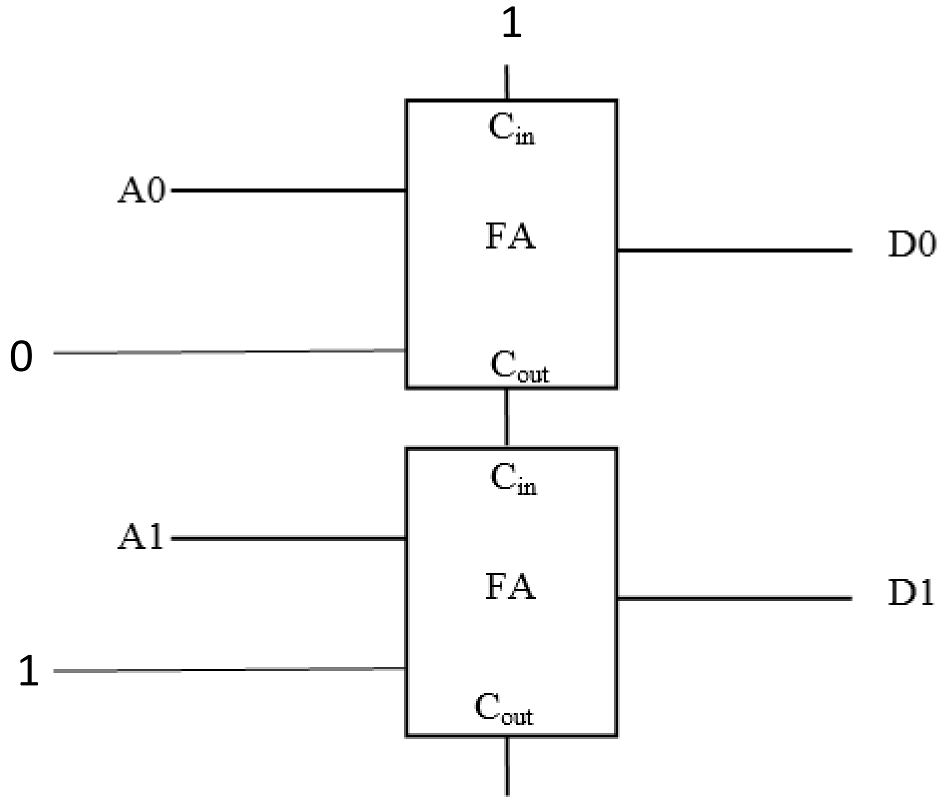
- Decrement:

D ← A - 1

	1
A	0101
	1111

D	0100





0001

1110
\+ 1

1111

- 2's complement OF 1 IS ALL ONES

Part 3: Types of ALU Micro-operations (Arithmetic micro-operations)

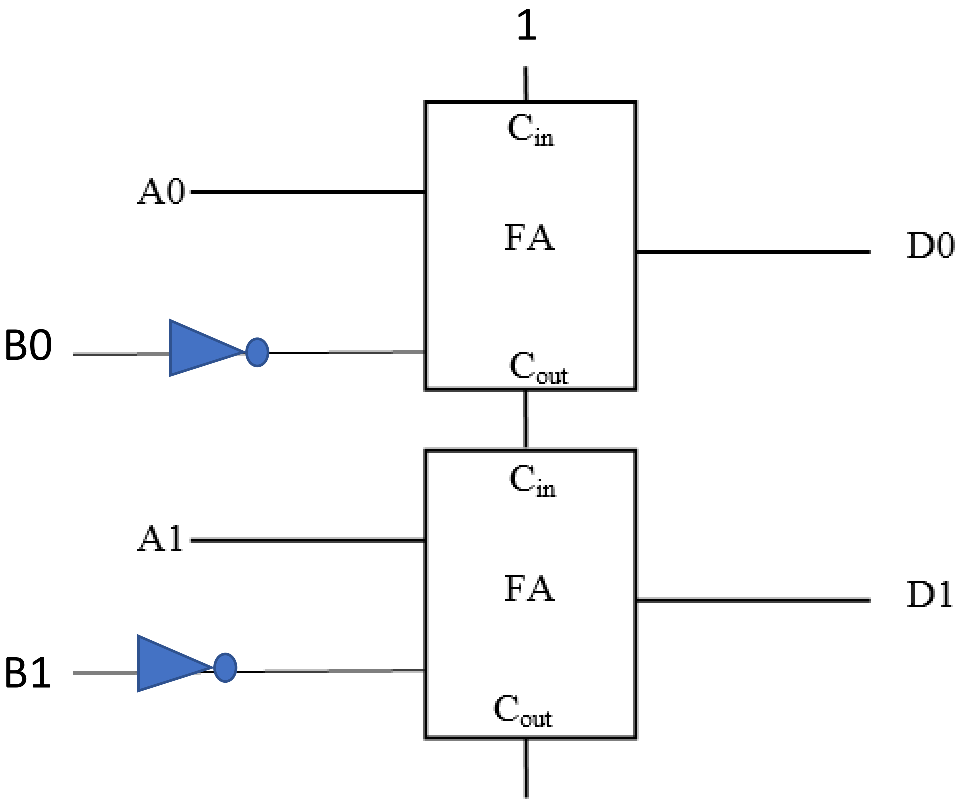
0011

- Subtraction:

$D \leftarrow A + \overline{B} + 1$

5	0101
- 3	0011

- Has to be converted to 5+(-3).
- We have to get negative representation of 3 by applying 2's complement



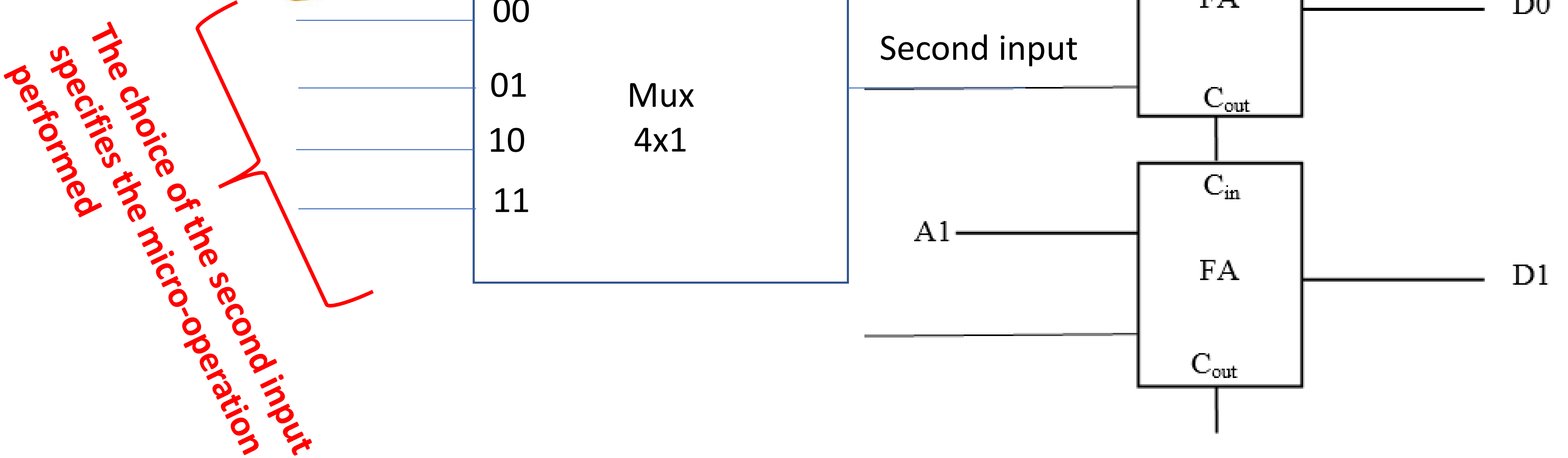
1's complement 1100
+ 1

1101

- 2's complement, this is -3

ALU design for Arithmetic micro-operations

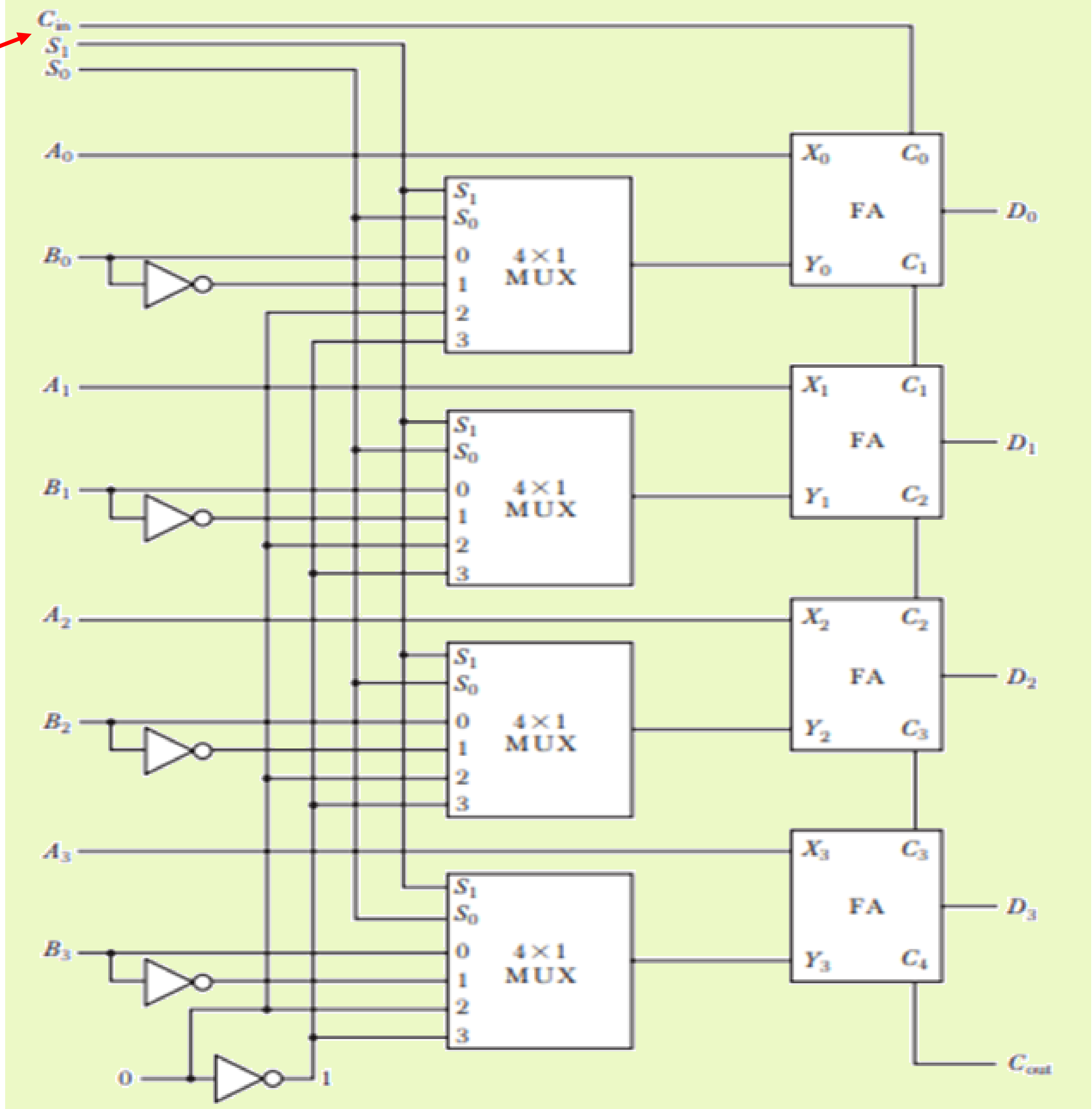
How the ALU is designed to allow all the arithmetic microoperations?



Note the C_{in} has also an impact on the micro-operation

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

4-bit arithmetic circuit



4-bit arithmetic circuit

Types of ALU micro-operations

Types of ALU micro-operations

- Data Transfer
- Arithmetic
- Logical
- Shift

D ←A

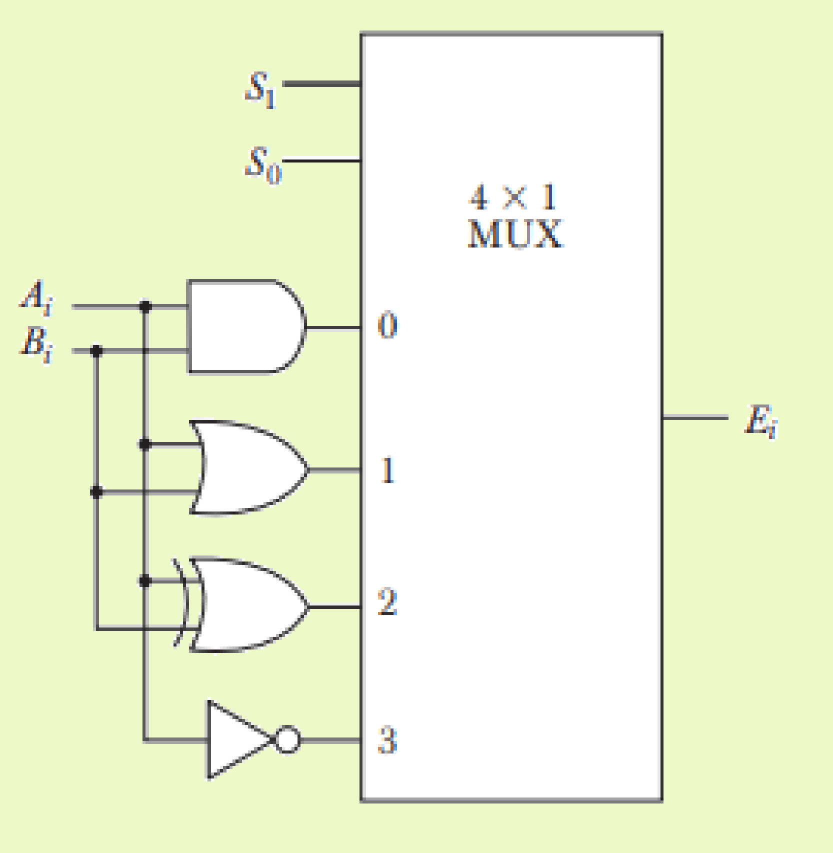
D ←A+B
D ←A+B’+1
D ← A+1
D ← A-1

Logical Micro-operations

Four Logical micro-operations

S1 S0	
0 0	E ← A ∧ B (And)
0 1	E ← A ∨ B (OR)
1 0	E ← A ⊕ B (XOR)
1 1	E ← A' (NOT)

$$\begin{array}{rll} & A & 0110 \\ \wedge & B & 0101 \\ \hline & E & 0100 \end{array} \qquad \begin{array}{rll} & A & 0110 \\ \vee & B & 0101 \\ \hline & E & 0111 \end{array} \qquad \begin{array}{rll} & A & 0110 \\ \oplus & B & 0101 \\ \hline & E & 0011 \end{array} \qquad \begin{array}{ll} A & 0110 \\ \hline E & 1001 \end{array}$$



One stage of the logical circuit

Shift micro-operations

Shift Micro-operations

- Logical Shift left
- Logical Shift right
- Circulate left
- Circulate right
- Arithmetic Shift left
- Arithmetic Shift right

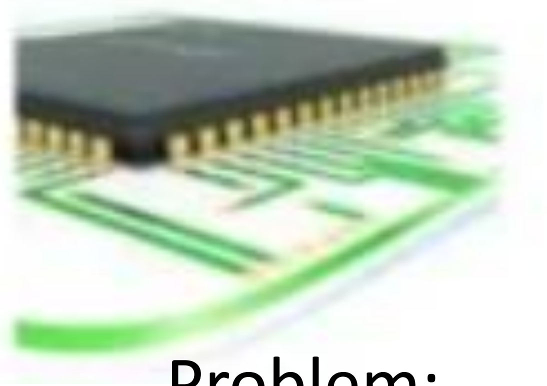
$V_s = R_{n-1} \oplus R_{n-2}$

	Logical Shift left	Logical Shift right	Circulate left	Circulate right	Arithmetic Shift left	Arithmetic Shift right
Example1	0010 → 0100	0100 → 0010	0101 → 1010	0101 → 1010	0010 → 0100, Vs =0	0100 → 0010
Example2	1010 → 0100	1100 → 0110	1100 → 1001	1100 → 0110	1010 → 0100, Vs =1	1100 → 1110

1 Shift left → multiply by 2 with unsigned numbers

1 Shift right → divide by 2 with unsigned numbers

Can not solve the problem, but indicates overflow



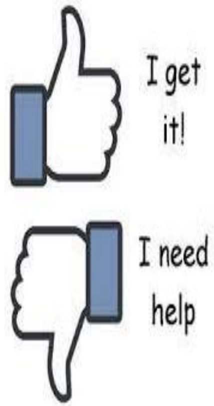
Check your Understanding

Problem:

Starting from an initial value of R = 01101001, determine the sequence of binary values of R after a logical shift left, followed by a circular shift right, followed by an arithmetic shift right, followed by arithmetic shift left. State whether there is an overflow.

Solution:

R = 01101001
After logical Shift left: 11010010
After circular shift right: 01101001
After arithmetic Shift right: 00110100
Arithmetic shift left: 01101000
After arithmetic shift left, no overflow occurred.



Shift micro-operations

RTL	Description
R ← shl R	Shift Left register R
R ← shr R	Shift right register R
R ← cil R	Circular shift Left register R
R ← cir R	Circular shift right register R
R ← ashl R	Arithmetic shift Left register R
R← ashr R	Arithmetic shift right register R

Shift micro-operations

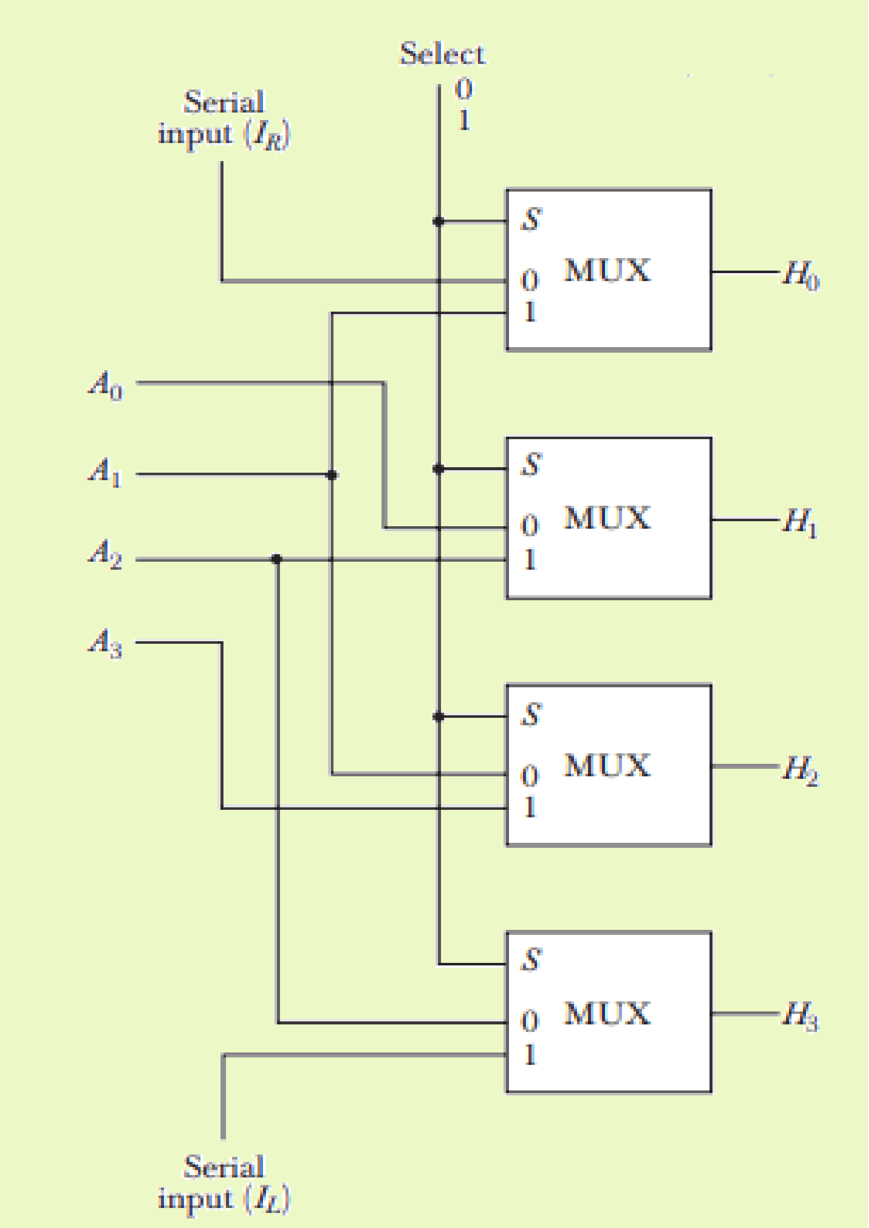
Shifting left

A3	A2	A1	A0	Before Shifting left
A2	A1	A0	Input	After Shifting left

Shifting right

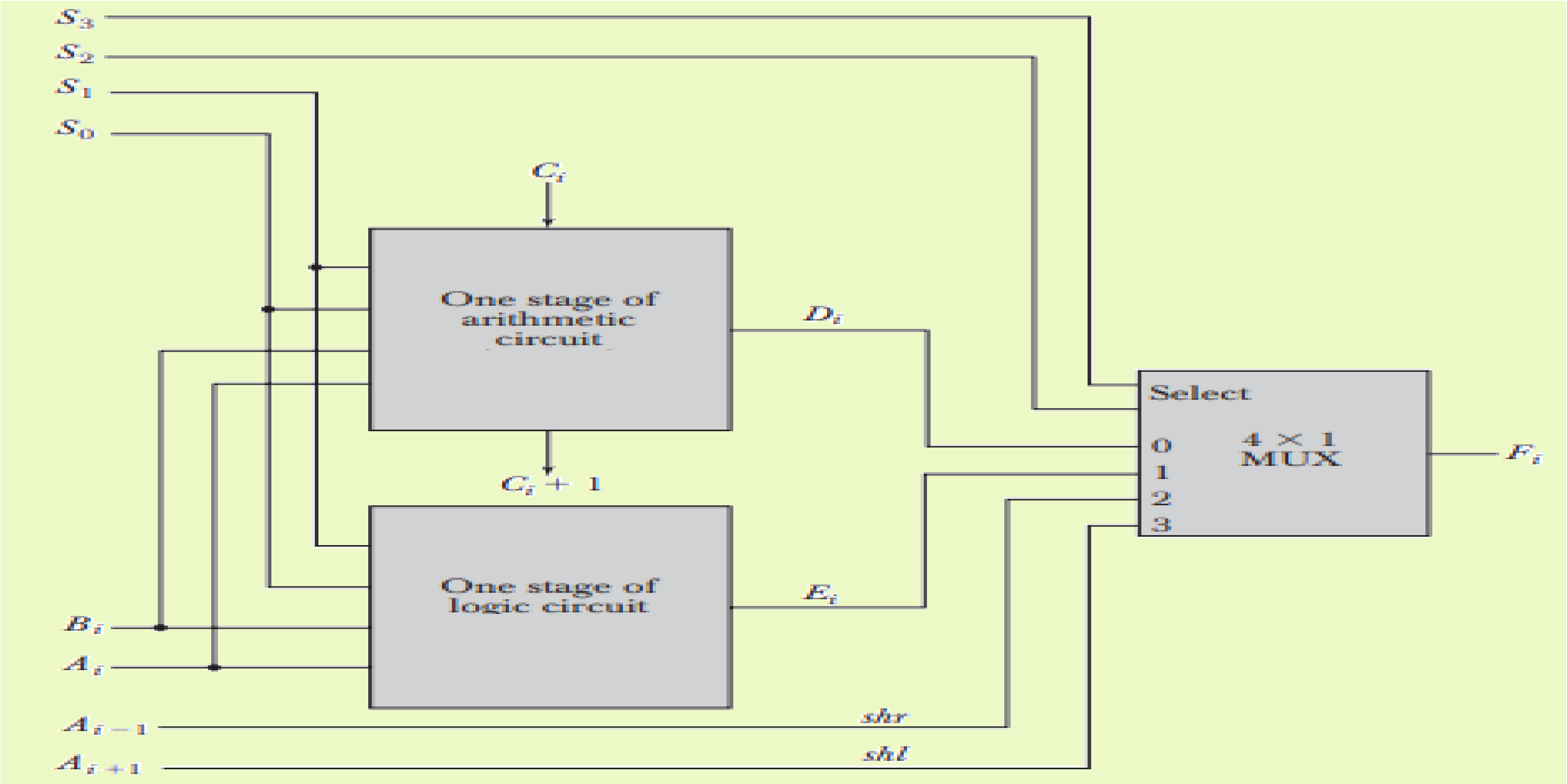
A3	A2	A1	A0	Before Shifting right
Input	A3	A2	A1	After Shifting right

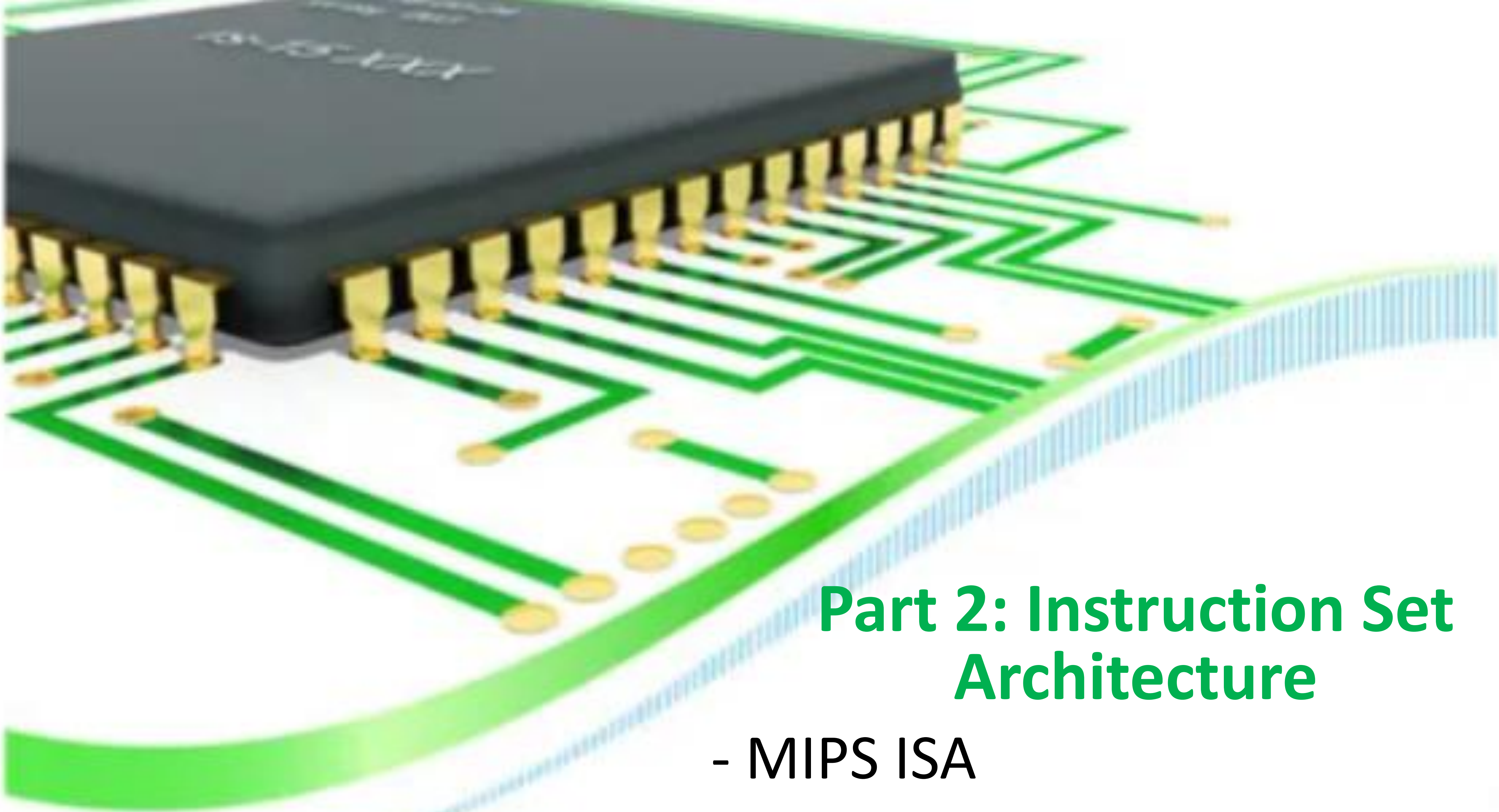
S	H3	H2	H1	H0	Micro-operation
0	A3	A1	A0	I_R	Shift left
1	I_L	A3	A2	A1	Shift right



4-bit circuit shifter

One Stage of ALU





Part 2: Instruction Set Architecture

- MIPS ISA

Instruction Set Architecture

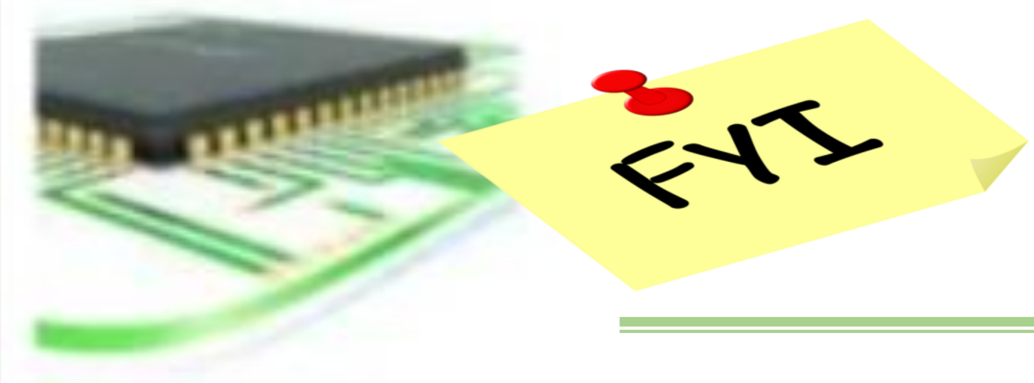
- An Instruction Set Architecture (ISA) is part of the abstract model of a computer that defines how the CPU is controlled by the software.

- The ISA acts as an interface *between the hardware and the software*, specifying both what the processor is capable of doing as well as how it gets done.

Instruction Set

- An instruction set is a group of commands for a central processing unit (CPU) in machine language. The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations.

We will study
MIPS microprocessor
Its Instruction Set Architecture and
Its Instruction Set



MIPS microprocessor

The five most iconic devices using MIPS CPUs

Sony PlayStation (MIPS R3000 CPU)

Nintendo 64 (MIPS R4300i CPU)

Tesla Model S (MIPS I-class CPU)

NEC Cenju-4 (MIPS R10000 CPU) supercomputer

SGI Indigo (MIPS R3000) workstation

MIPS-32 ISA

- MIPS-32 uses 32 general purpose registers, each 32 bits wide
- The MIPS architecture can support up to 32 address lines.
- MIPS is a byte-addressable architecture
- The word size is 32-bits

MIPS commonly used registers

Register Number	Register Name	Description
0	$zero	The value 0
2-3	$v0 - $v1	(values) from expression evaluation and function results
4-7	$a0 - $a3	(arguments) First four parameters for subroutine
8-15, 24-25	$t0 - $t9	Temporary variables
16-23	$s0 - $s7	Saved values representing final computed results
31	$ra	Return address

Categories of MIPS instructions

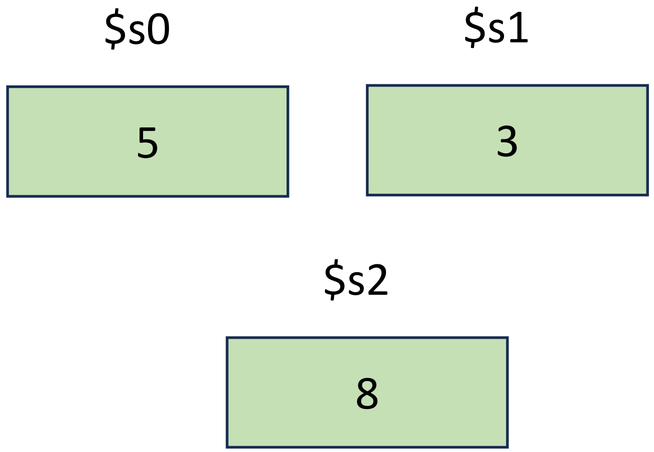
- Categories of MIPS instructions
  - Arithmetic
  - Logical
  - Data Transfer
  - Branch/jump
  - Comparison

Arithmetic instructions

add \$s2, \$s1, \$s0

destination register

Source operand registers



sub \$s2, \$s0, \$s1 # \$s2 = 2

mul \$s2, \$s1, \$s0 # \$s2 = 15

div \$s2, \$s0, \$s1 # \$s2 = 1

Arithmetic instruction

addi $s2, $s1, 3

Immediate value

destination register

Source operand register

li $s4, 50

Data transfer instruction

$s1

5

$s2

8

$s4

50

MIPS Assembly Example

```
a = 5
b = 3
c = 10
x = a+b
y = x+7
x = y- a
z= c  * x
result = z/a
```

```
li $s0, 5
li $s1, 3
li $s2, 10
add $t0, $s0, $s1
addi $t1, $t0, 7
sub $t0, $t1, $s0
mul $t3, $t0, $s2
div $s3, $t3, $s0
```

```
# Declare main as a global function
.globl main
# All program code is placed after the
# .text assembler directive
.text

# The label 'main' represents the starting point
main:
 li $s0, 5
 li $s1, 3
 li $s2, 10
 add $t0, $s0, $s1
 addi $t1, $t0, 7
 sub $t0, $t1, $s0
 mul $t3, $t0, $s2
 div $s3, $t3, $s0
 li $v0, 10 # Sets $v0 to "10" to select exit syscall
 syscall # Exit
```

R0	[r0]	=	0
R1	[at]	=	0
R2	[v0]	=	10
R3	[v1]	=	0
R4	[a0]	=	7
R5	[a1]	=	2147481548
R6	[a2]	=	2147481580
R7	[a3]	=	0
R8	[t0]	=	10
R9	[t1]	=	15
R10	[t2]	=	0
R11	[t3]	=	100
R12	[t4]	=	0
R13	[t5]	=	0
R14	[t6]	=	0
R15	[t7]	=	0
R16	[s0]	=	5
R17	[s1]	=	3
R18	[s2]	=	10
R19	[s3]	=	20

Thank You