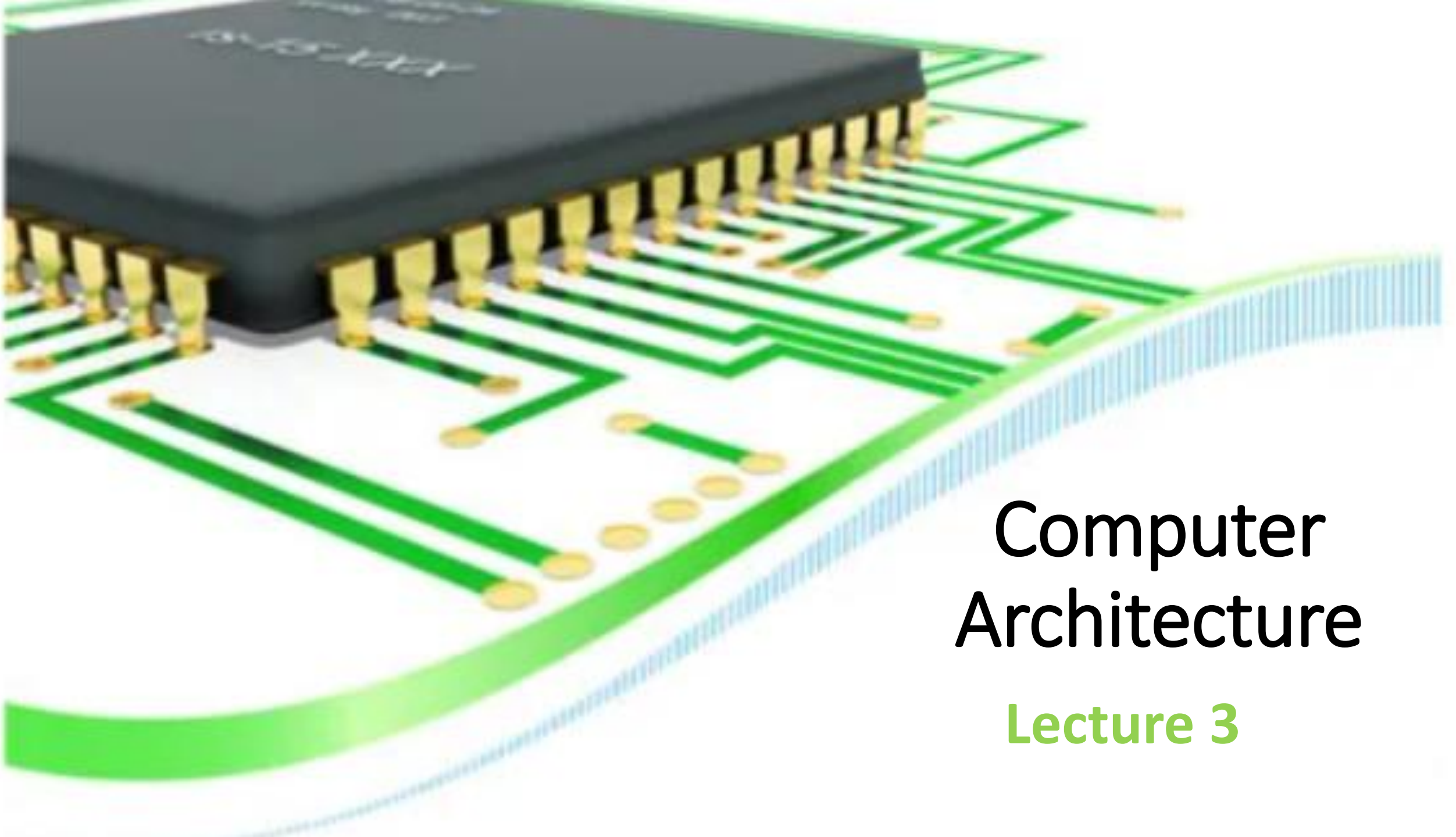
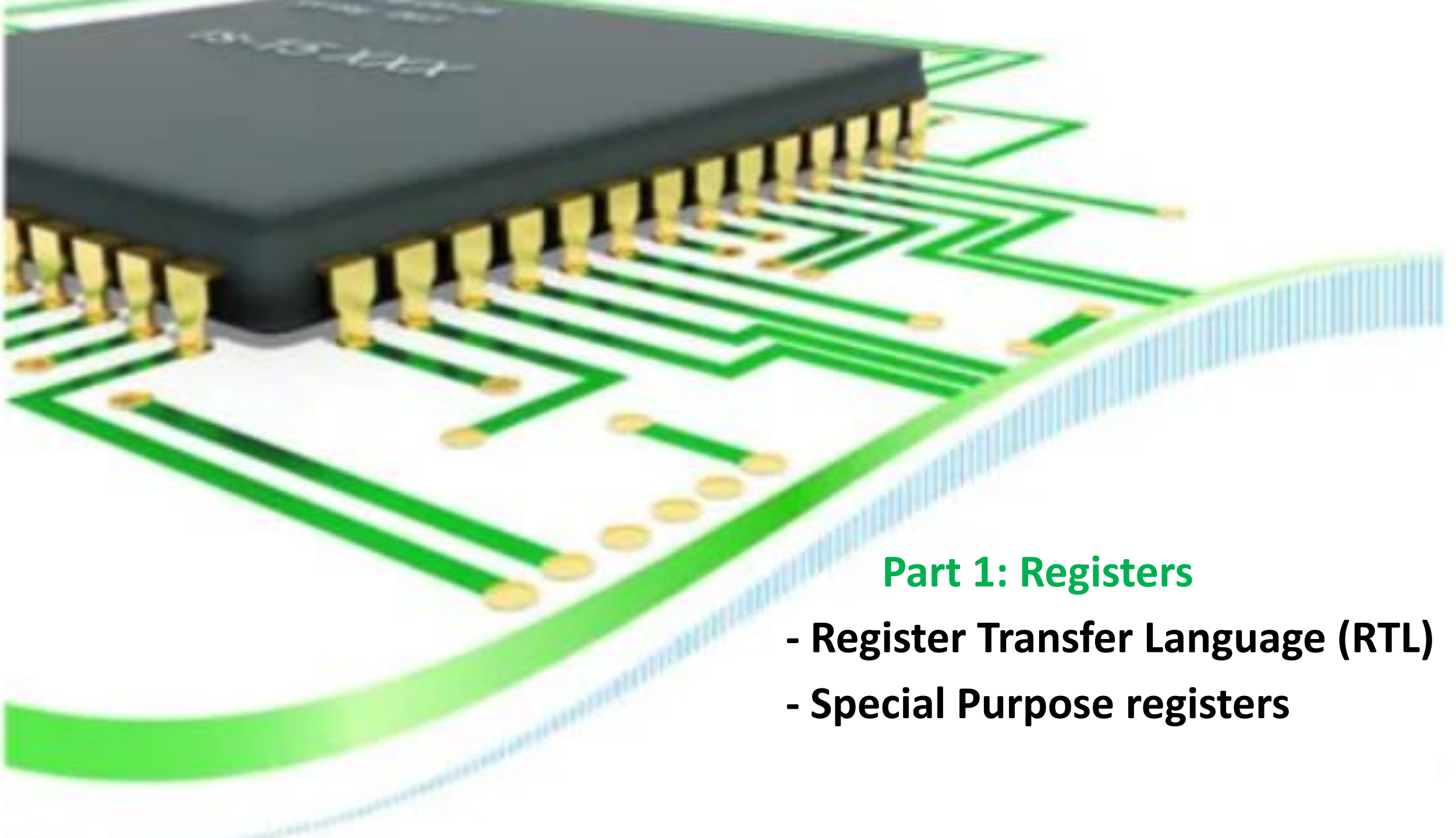


Computer Architecture

Lecture 3

Agenda

- **Part 1: Registers**
  - Register Transfer Language (RTL)
  - Special Purpose registers
- **Part 2: Buses**
  - Different types of Busses
  - Common Data Bus
- **Part 3: ALU**
  - Types of ALU Micro-operations
  - ALU Design

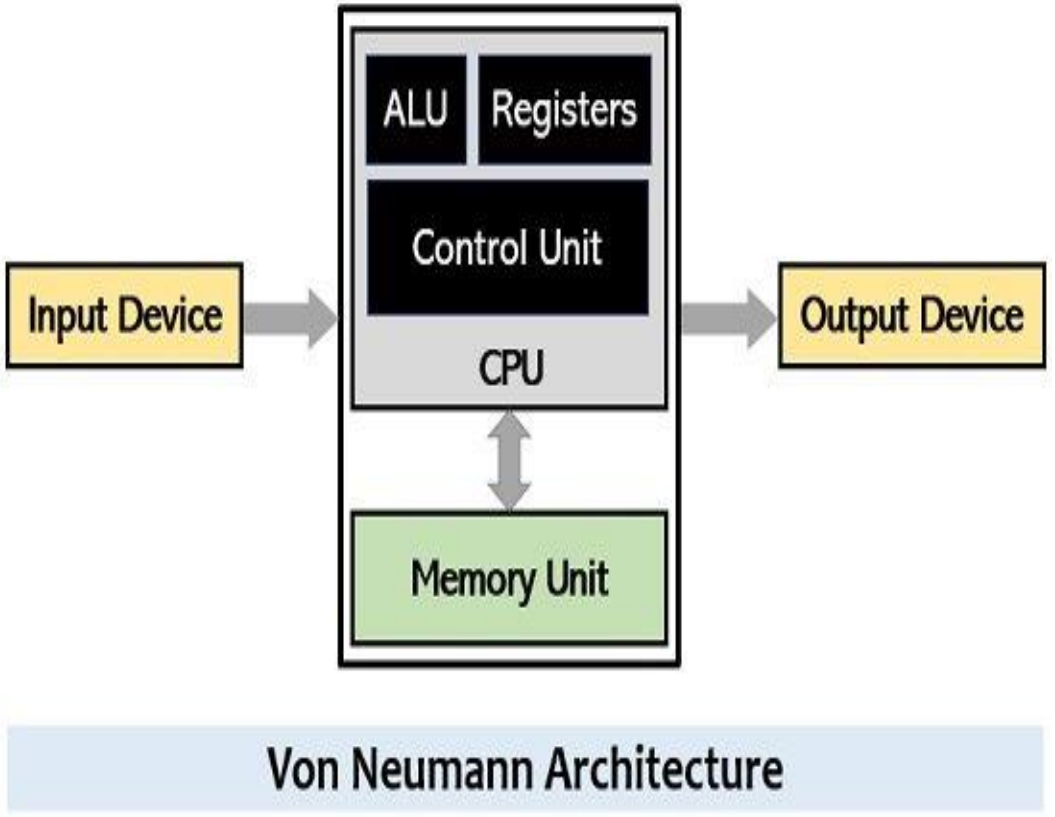


Part 1: Registers

- Register Transfer Language (RTL)
- Special Purpose registers

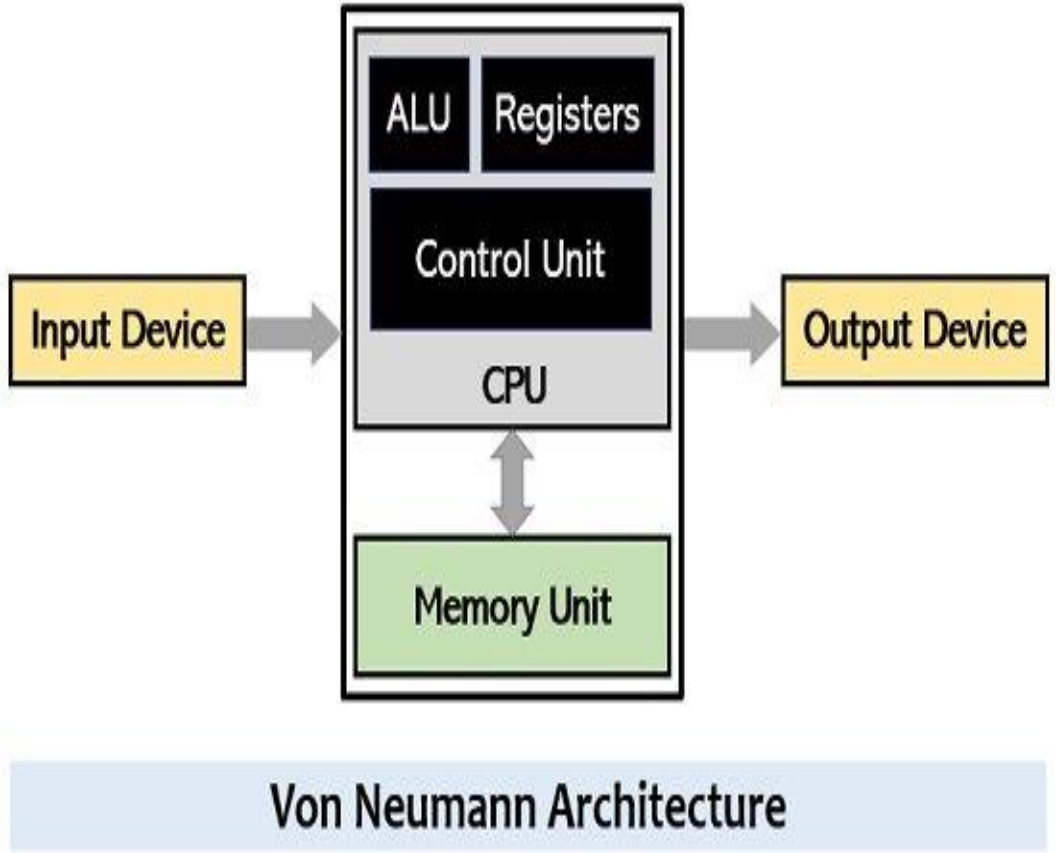
Von Neumann Architecture

- Von Neumann envisioned the structure of a computer system as being composed of three Subsystems:
- Central Processing unit (CPU) composed of:
  - Control Unit (CU)
  - Arithmetic Logic Unit (ALU)
  - Registers
- Memory
- Input and Output Subsystems



Subunits of CPU

- **Control Unit (CU):** The brain within the brain.
- **Arithmetic Logic Unit (ALU) :** responsible for performing arithmetic operations (add, sub,…).
- **Registers:** Temporarily storage area within the CPU.



General Scenario for an Instruction cycle

Control Unit

Control lines

=1

Control Signals

R1

1

R2

4

R3

5

ALU

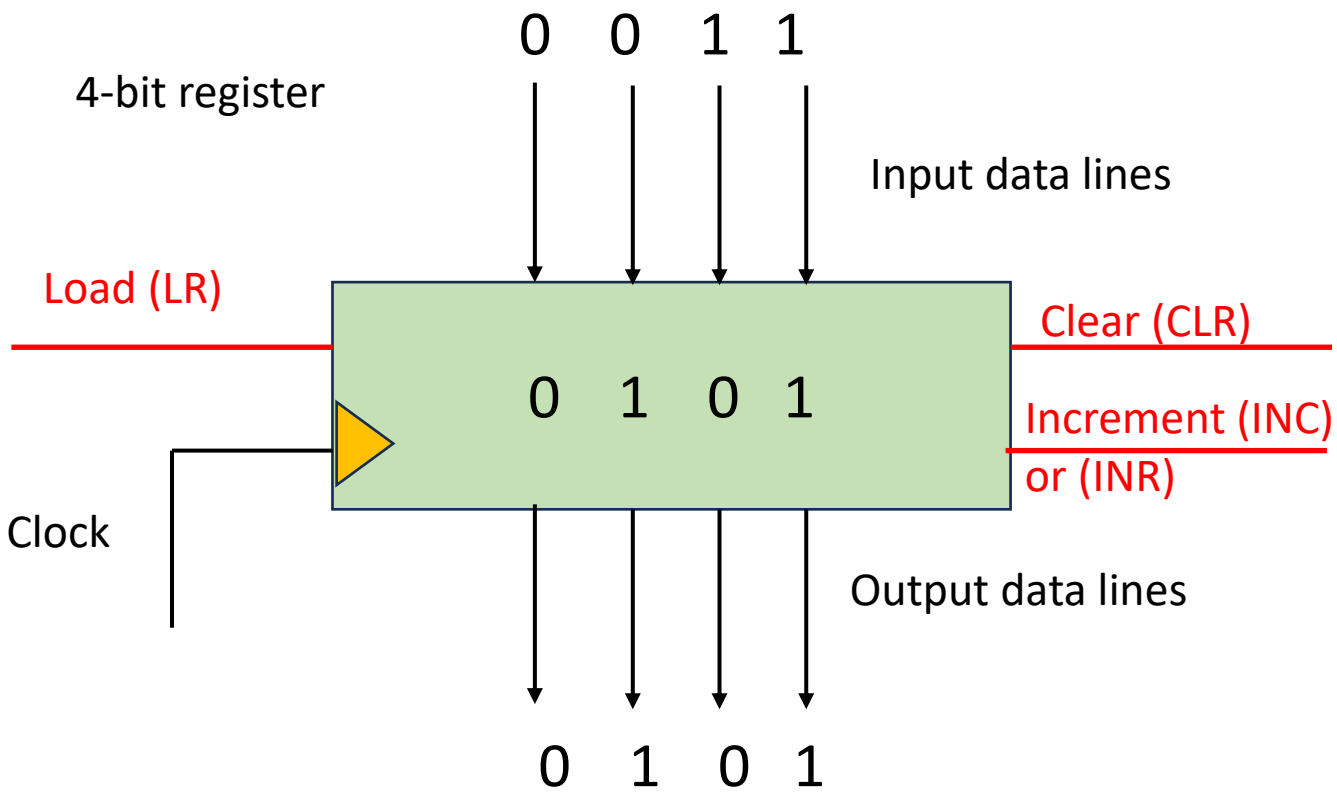
Memory

1+4

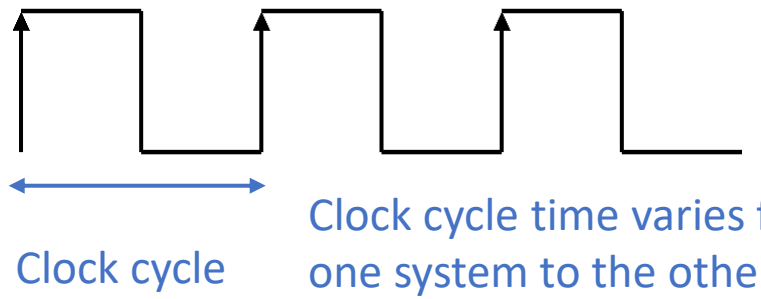
5

Register

- A special, high-speed temporary storage area within the CPU.
- There is a limited number of registers in each architecture.



Clear	Load	Increment	Operation
0	0	0	No change
0	0	1	Increment value by 1
0	1	X	Load new input
1	X	X	Clear value to Zero



Control Signals to Registers

- **Increment (INC or INR):**

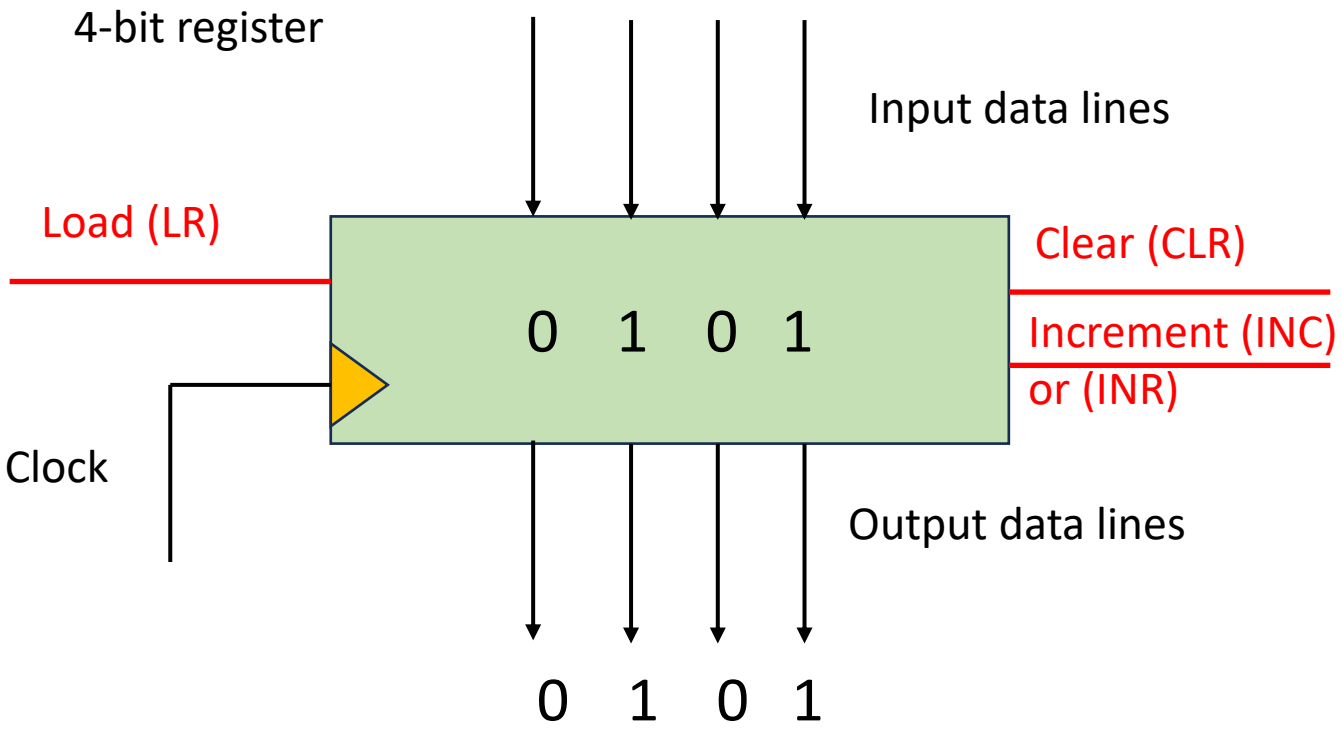
Increments the value inside the register.

- **Load (LD):**

Loads a new value inside the register.

- **Clear (CLR):**

Clears the value inside the register to be zero.

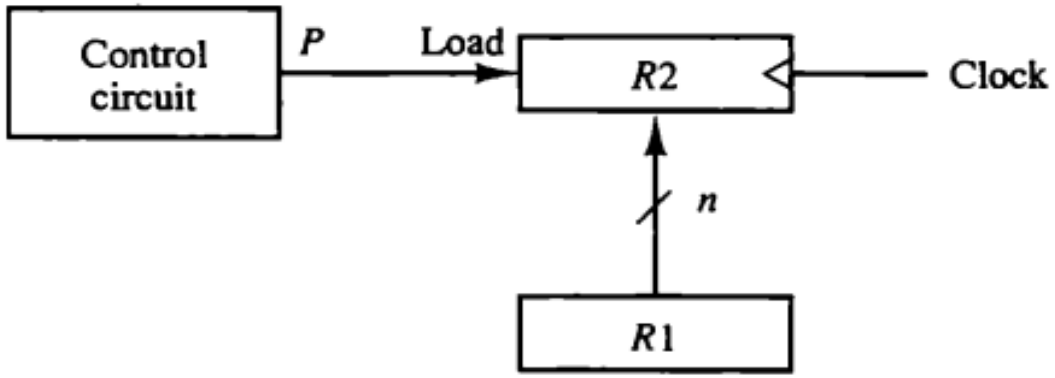


Register Transfer language (RTL)

- It is the language (symbolic notation)used to describe data flow at the **register-transfer** level of an architecture.
- For Example:
  - If Control line **P = 1** then a transfer from R1 to R2 will be done

This is written in RTL as:

P: R2 ← R1



Register Transfer language (RTL)

- If Control line **Q = 1** then a transfer from R1 to R2 and from R1 to R3 will be done

  This is written in RTL as:
  Q: R2 ← R1, R3 ← R1

- If both Control lines **P and Q must be equal to 1** then a transfer from R3 to R4 will be done

  This is written in RTL as:
  PQ: R4 ← R3

  P.Q: R4 ← R3

Register Transfer language (RTL)

- For Example:
  - If any of the Control lines **X or Y should be equal to 1** then an addition between R1 and R2 should be done and the result is stored in R3.

    This is written in RTL as:

    X+Y: R3 ← R1+R2
  - If Control line **S should be equal to 0** then R1 will be transferred to R2

    This is written in RTL as:

    $\overline{S}$: R2 ← R1

Register Transfer language (RTL)

- For Example:
  - If the Control line **X is equal to 1** then an **AND** micro-operation between R1 and R2 is performed and the result is stored in R3.

    This is written in RTL as:

    X: R3 ← R1 ∧ R2

  - If the Control line **Y is equal to 1** then an **OR** micro-operation between R1 and R2 is performed and the result is stored in R3.

    This is written in RTL as:

    X: R3 ← R1 ∨ R2

Summary of Basic Symbols for Register Transfer Language

Symbol	Description	Examples
Letters & numerals	Denotes a register	MAR, R2
Parenthesis ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Comma ,	Separates two micro operations	R2 ← R1, R1 ← R2

Register Transfer language (RTL)

- Represent the following conditional control statement by register transfer statements (RTL) :

```
If {(P = 1) then
      Transfer R1 + R2 to R1
      Transfer R1 ∧ R2 to R2
else
      if {( Q = 0) then
            (Transfer R1 − R2 to R1)
            Transfer R1 ∨ R2 to R2
      else
            if {(T = 1) then
                  (Transfer R2 + 1 to R1)
                  (Transfer NOT R1 to R2)}}}
```

Solution:

P: R1 ← R1+R2 , R2 ← R1 ∧ R2

$\overline{P}\overline{Q}$: R1 ← R1-R2, R2← R1 V R2

$\overline{P}$ Q T: R1 ← R2+1, R2← $\overline{R1}$

Types of Registers

- ***General purpose registers***

These are mainly used for operands for logical and arithmetic operations.

- ***Special purpose registers***

A special purpose register is one that has a specific task to carry out.

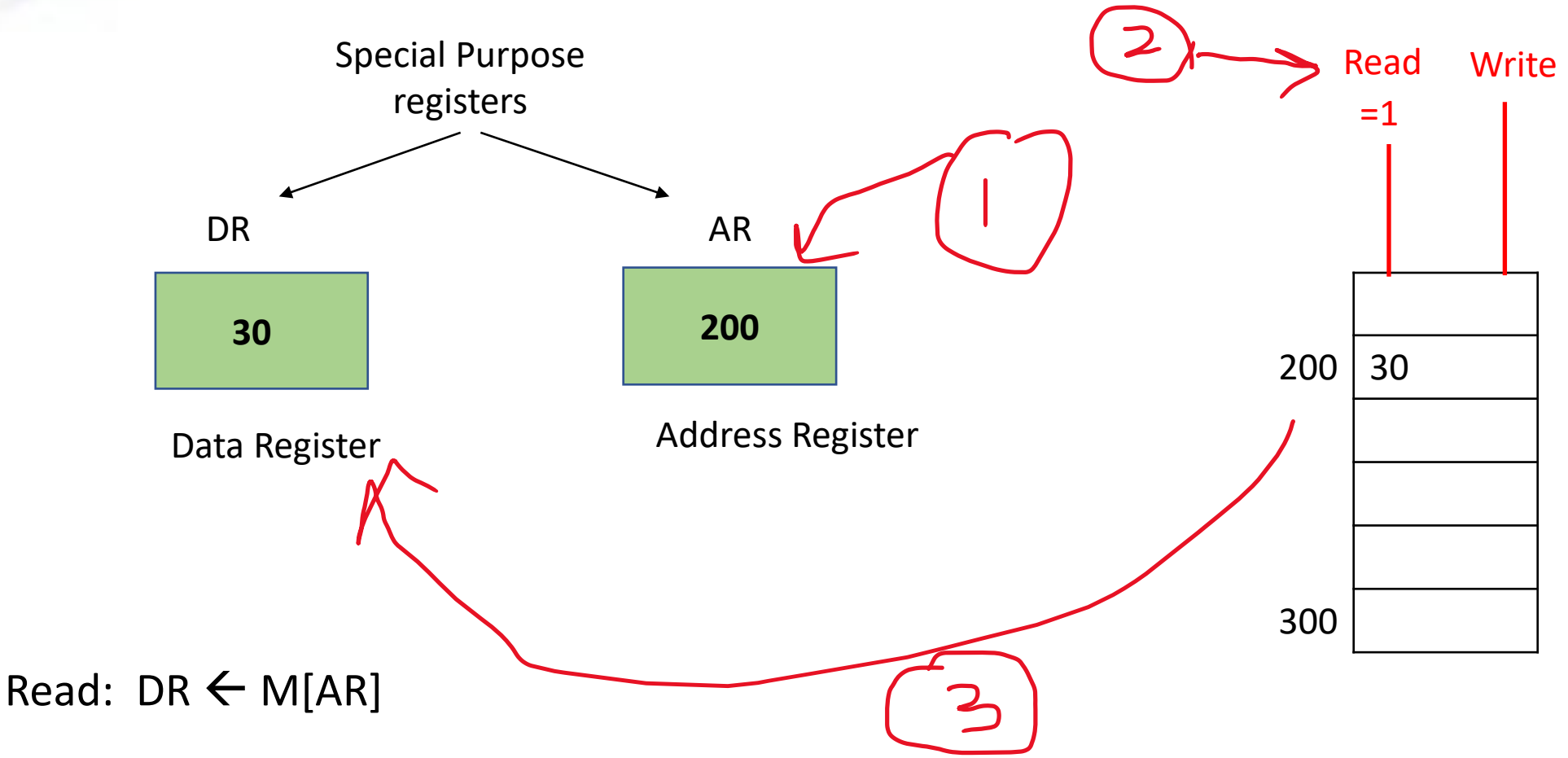
Two Special purpose registers

- **Memory data register (MDR)/(DR)** A register used for holding data transferred from the memory to the central processor, or vice versa. It is also called **Memory Buffer Register(MBR).**

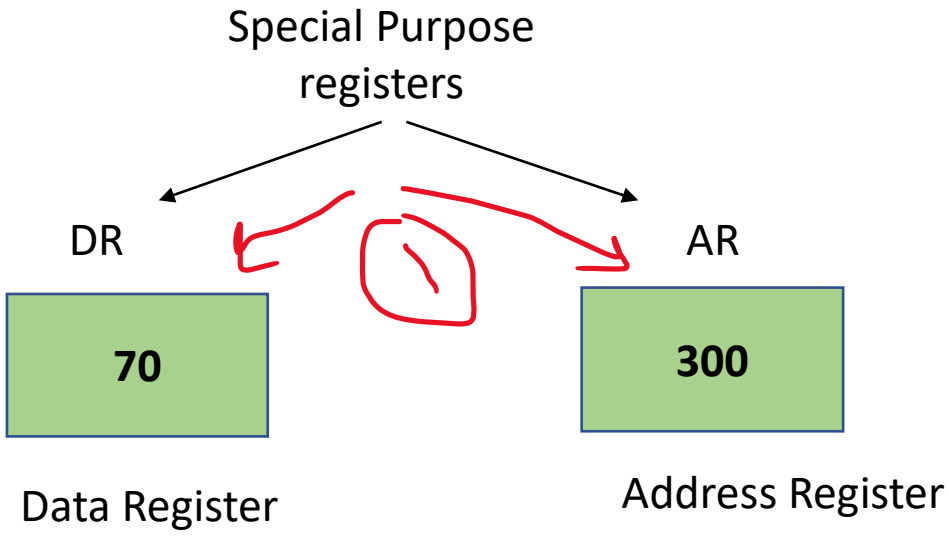
- **Memory address register (MAR)/(AR)** is the CPU register that either stores the memory address from which data will be fetched(read) to the CPU registers, or the address to which data will be sent and stored in the memory unit.

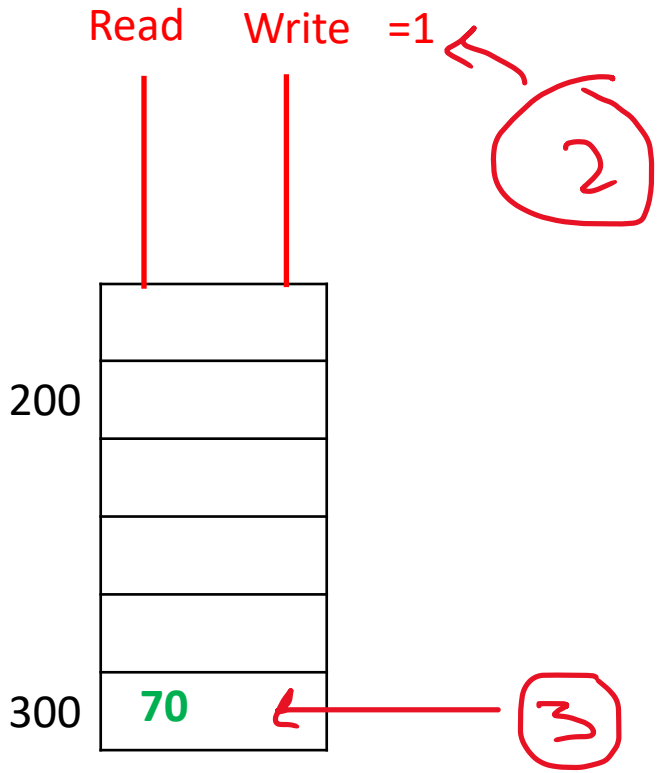
More Special purpose registers will be introduced later

General Scenario for reading from memory Unit



General Scenario for writing to memory Unit





Write: M[AR] ← DR

Special Purpose Registers

- These registers are designated for a special purpose:
- **MAR/AR**
- **MDR/DR/MBR**

discussed before

- **Program Counter (PC):** it holds the address of the memory location of the next instruction when the current instruction is executed by the microprocessor.
- **Instruction Register (IR):** This holds the current instruction to be executed after being read (fetched) from memory.
- **Accumulator (AC):** It is used for storing the results that are produced by the system. When the CPU gives the results after the execution, then all the results are stored into the AC Register.

Special Purpose Registers

IR

add R3, R1, R2

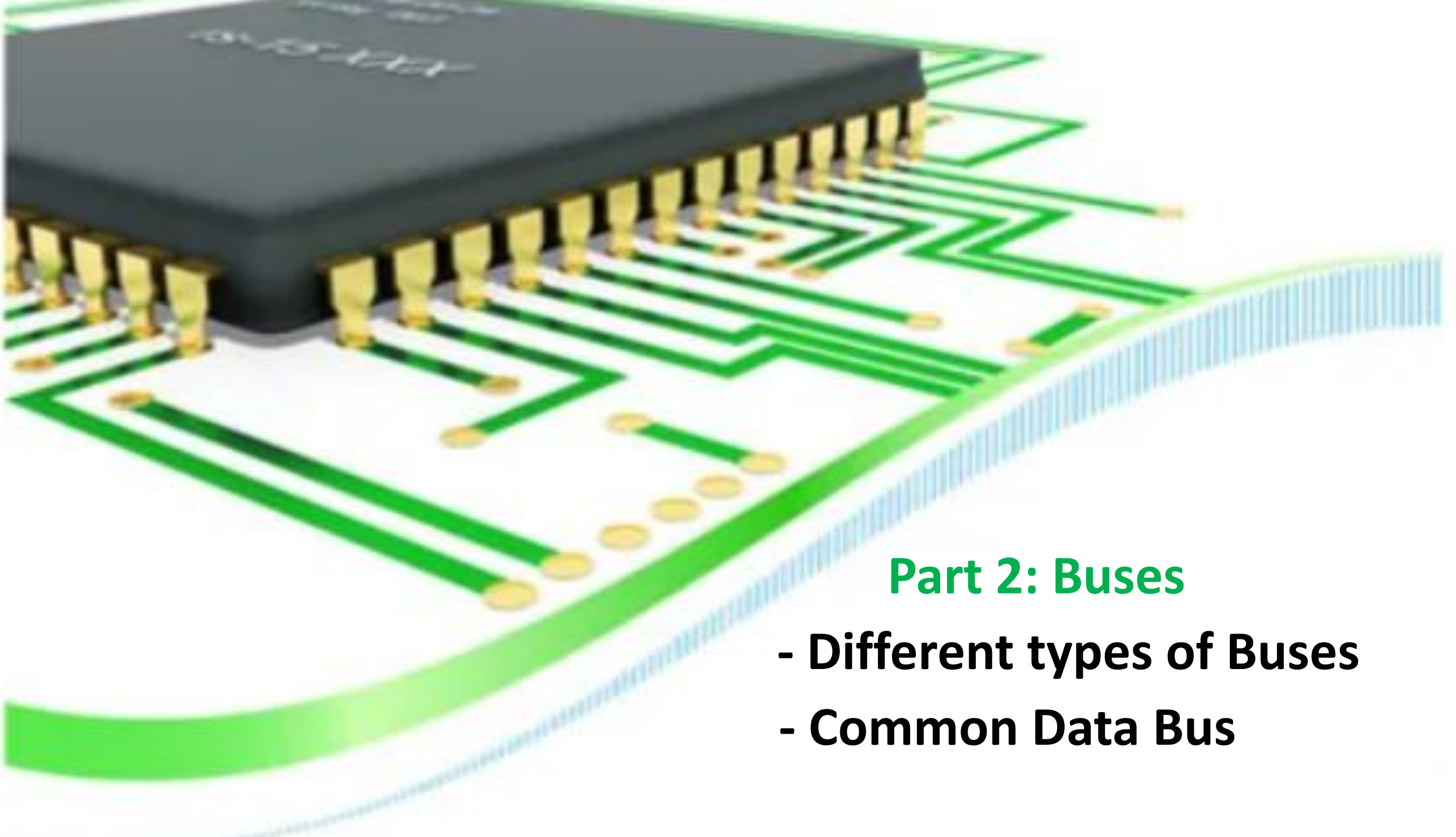
PC

204

Memory

200	add R3, R1, R2
204	sub R4, R1, R2

Assume a system where each instruction is stored in 4 cells of memory



Part 2: Buses

- Different types of Buses
- Common Data Bus

What is a bus?

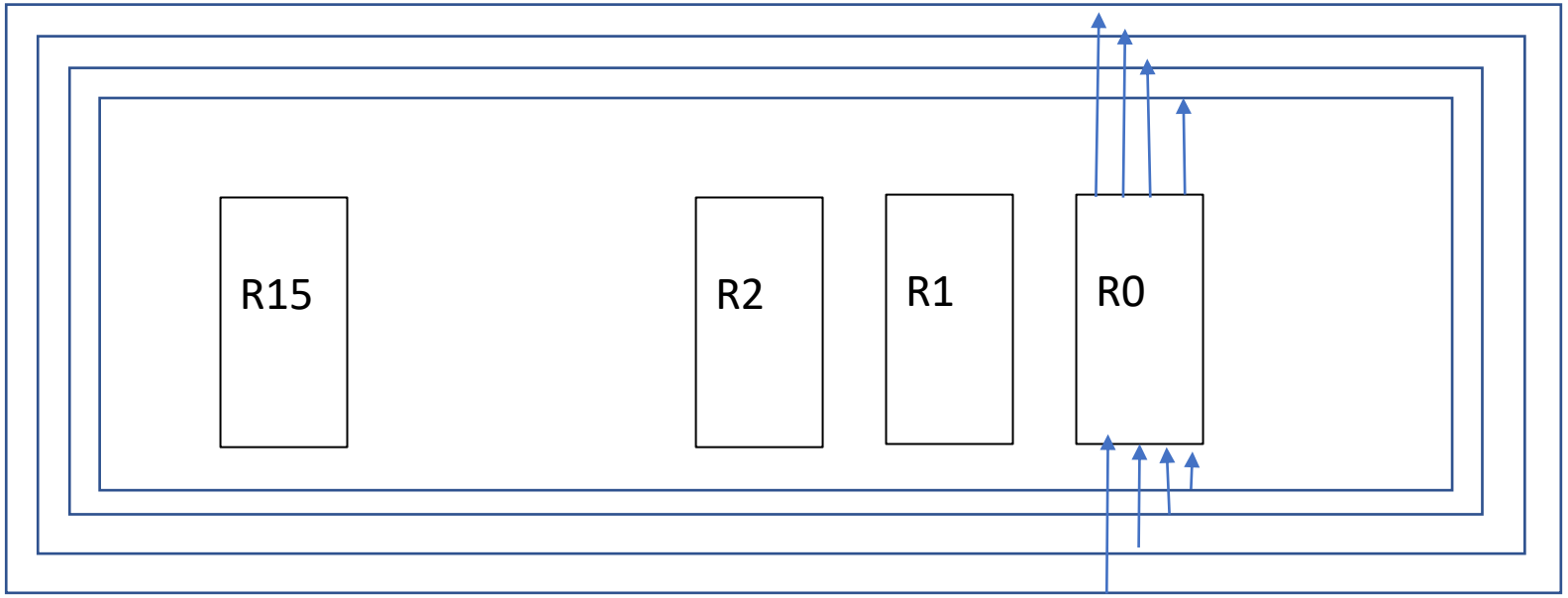
- A bus is a high-speed internal connection. Buses are used to send control signals and data between the processor and other components.

- A bus is a communication system that transfers data between components inside a computer, or between computers.

Different types of buses

- **Three types of bus are used:**
  - **Address bus** - carries memory addresses from the processor to other components such as primary storage and input/output devices.
  - **Data bus** - carries the data between the processor and other components.
  - **Control bus** - carries control signals from the processor to other components.

Common Data Bus



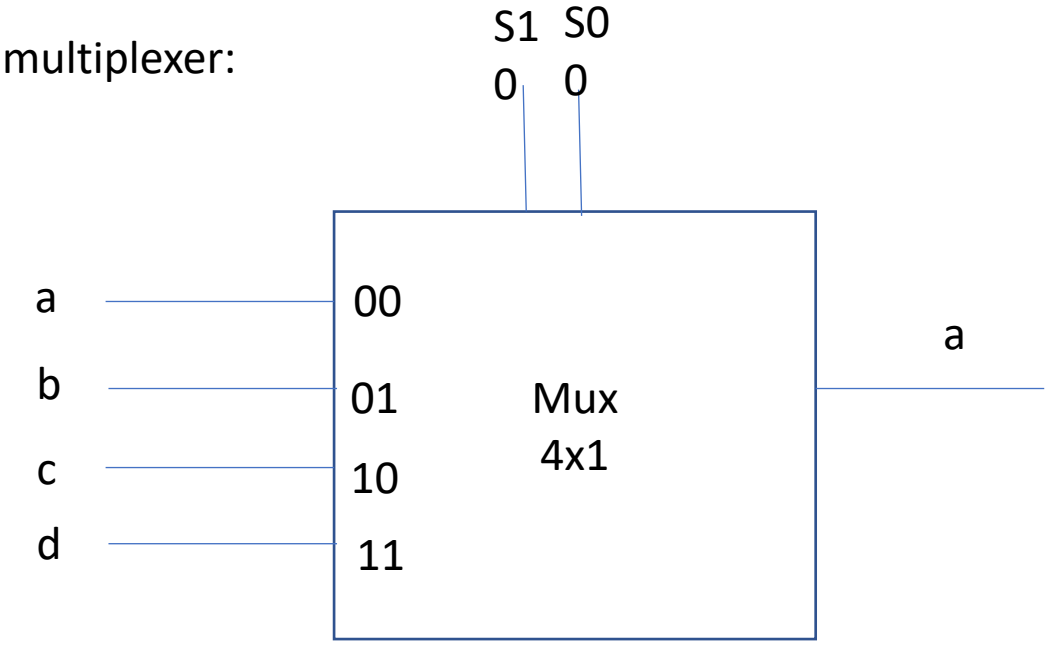
R15← R1

What really happens, the control unit orders R1 to give its content to the bus, then sets the load of R15 to 1 to load the content of R1

R15 ← bus, bus← R1

Common Data Bus, how the bus is designed

We need to revise a multiplexer:

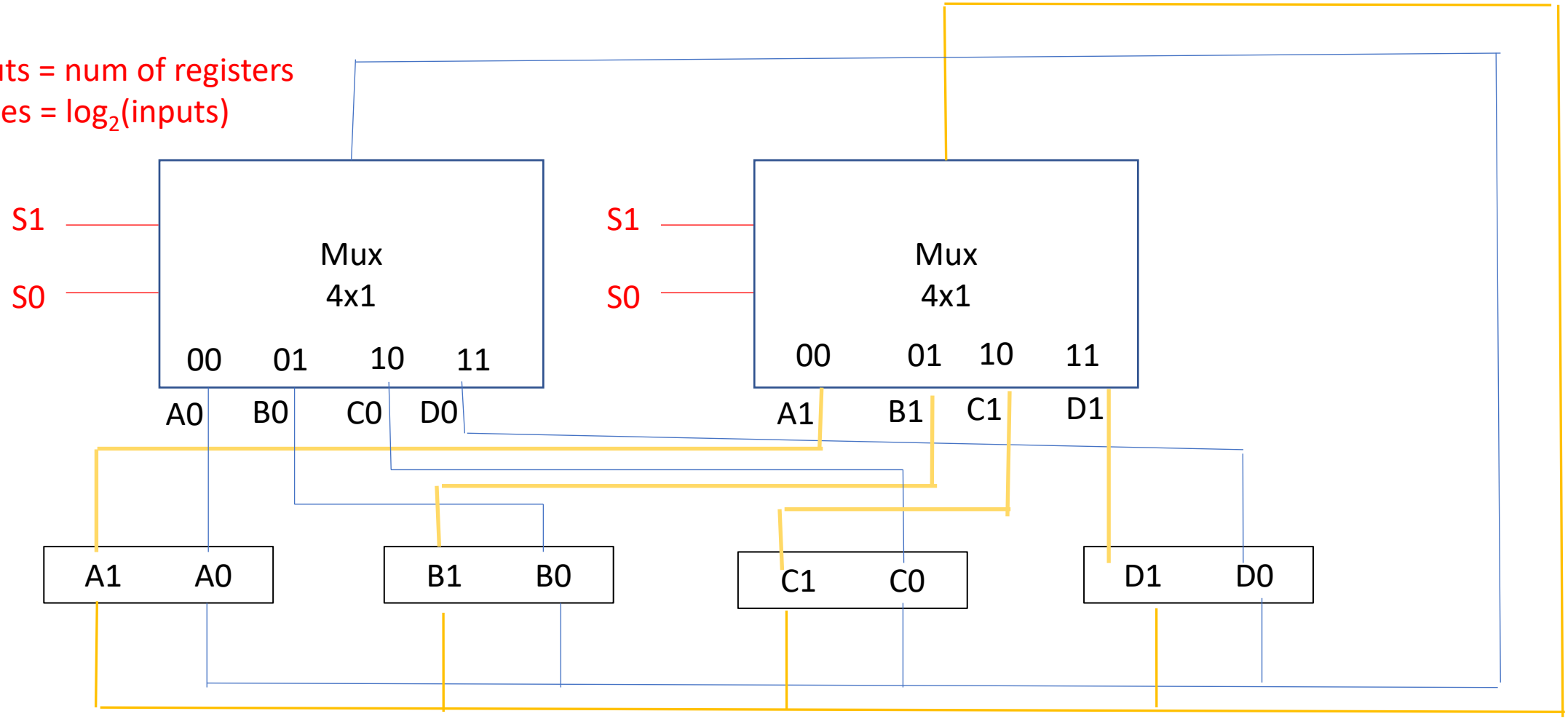


Number of selections = $\log_2$(inputs)

Part 2: Common Data Bus, how the bus is designed?

Num of Mux = num of bits in a register
Num of inputs = num of registers
Selection lines = $\log_2$(inputs)

A simple bus for 4 registers and each register is 2 bits only.



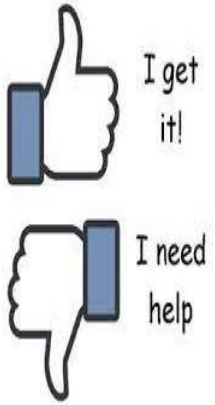
Check your understanding

- A digital computer has a common bus system for 32 registers of 16 bits each. The bus is constructed with multiplexers.

a) How many multiplexers are in the bus?
b) What is the size of each multiplexer?
c) How many selection inputs are there in each multiplexer?

Solution:

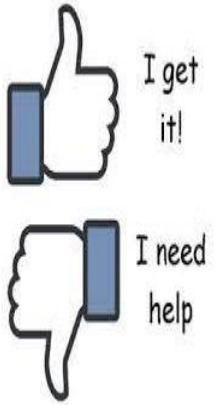
a. 16 Multiplexers
b. 32x1 Multiplexers
c. 5 selection lines



Check your understanding

- Design the common bus for 32 registers of 16 bits each, showing only the first two multiplexers and focusing on the first four inputs for each of these multiplexers.

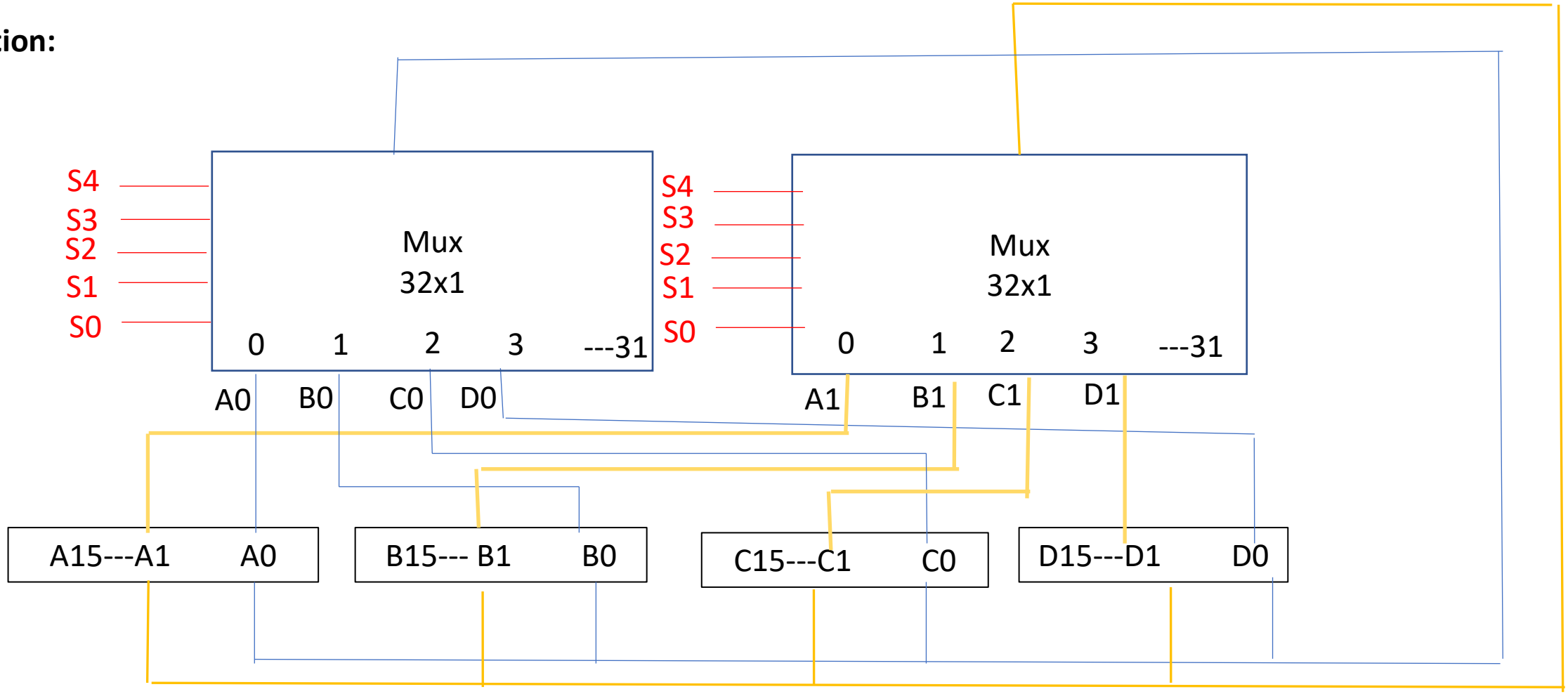
Solution next slide



Check your understanding

A simple bus for 32 registers and each register is 16 bits.

Solution:



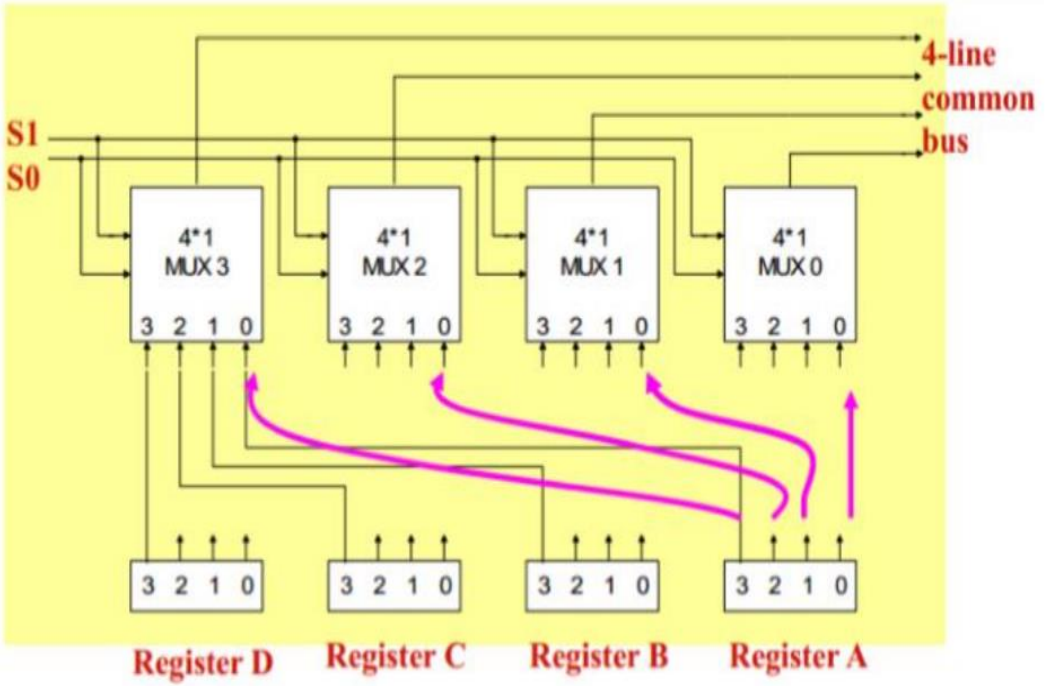
Check your understanding

- Given the common bus system for 4 registers where each register is 4-bits. State for each of the following Register transfer language statements, the selection lines values, and the load of which register should be set to 1.

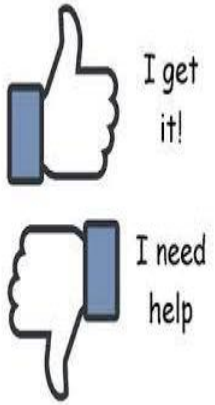
a) C ← A

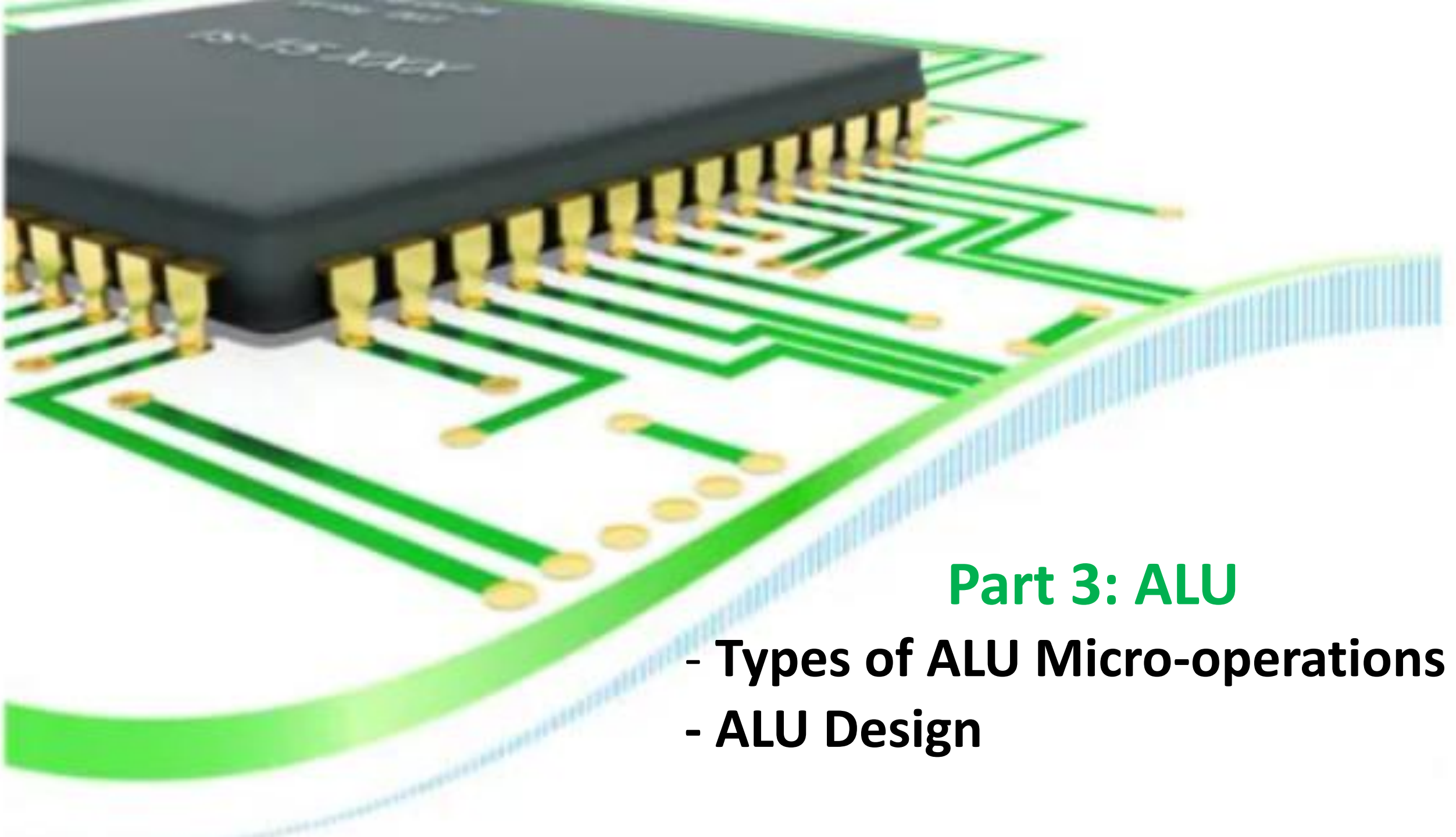
b) A ← B

c) A←D, C←D



	S1	S0	Load
a) C <- A	0	0	Register C
b) A <-B	0	1	Register A
d) A <- D, C<- D	1	1	Registers A, C

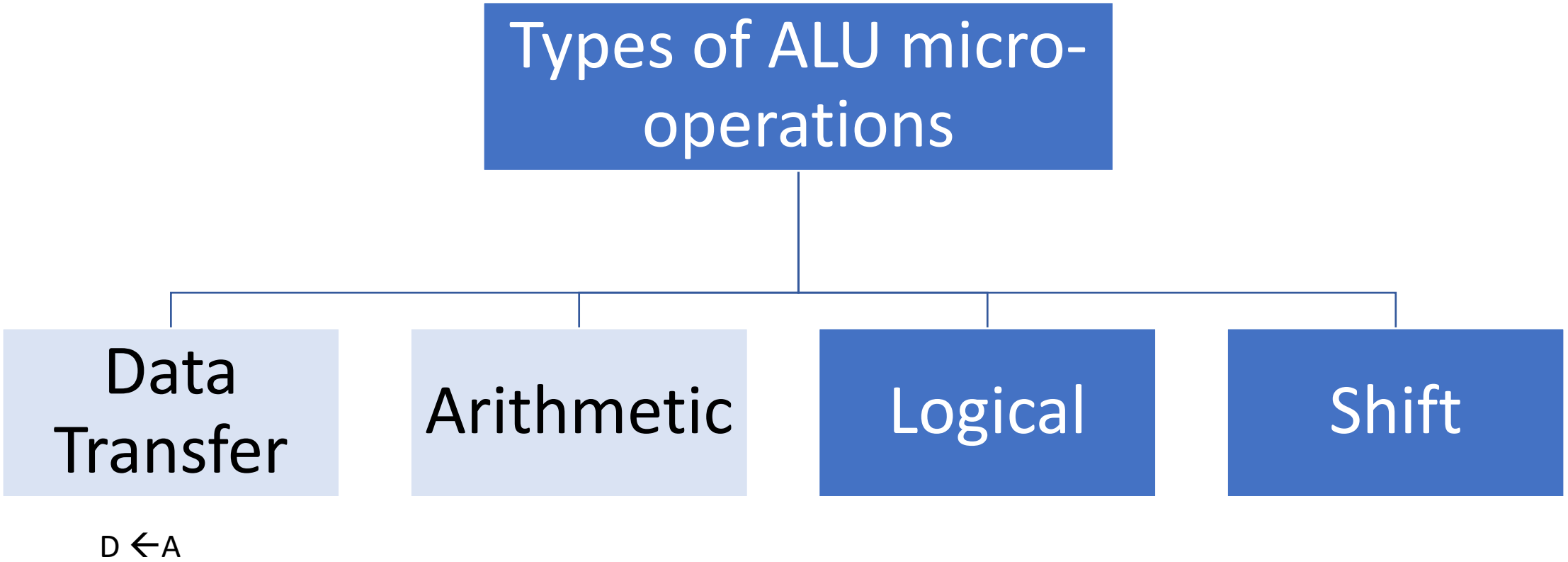




Part 3: ALU

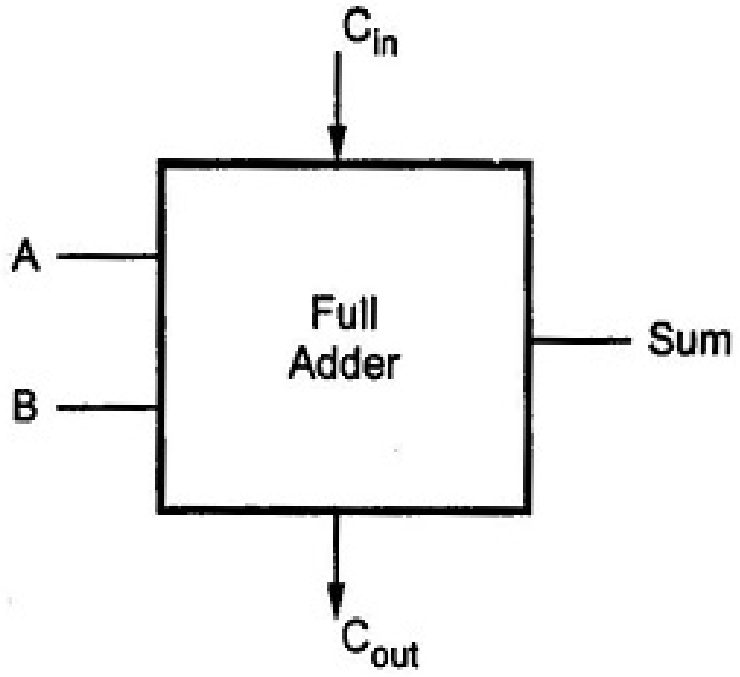
- Types of ALU Micro-operations
- ALU Design

Types of ALU Micro-operations



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

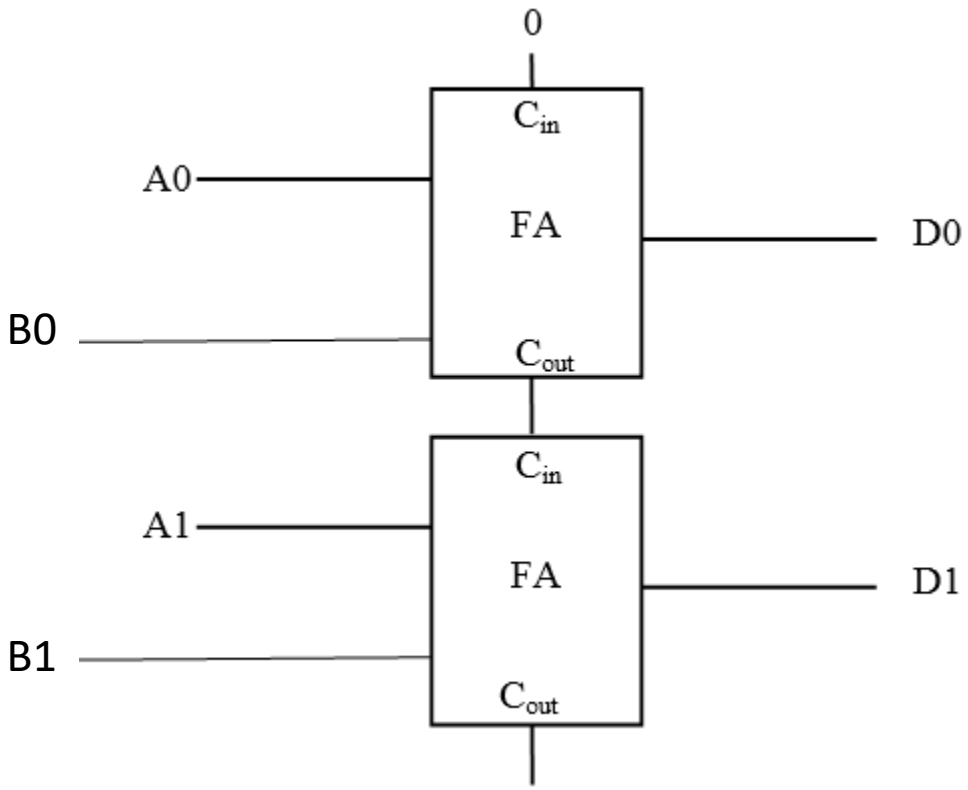
Truth Table of Full Adder

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

- Addition:

  D ← A+ B

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



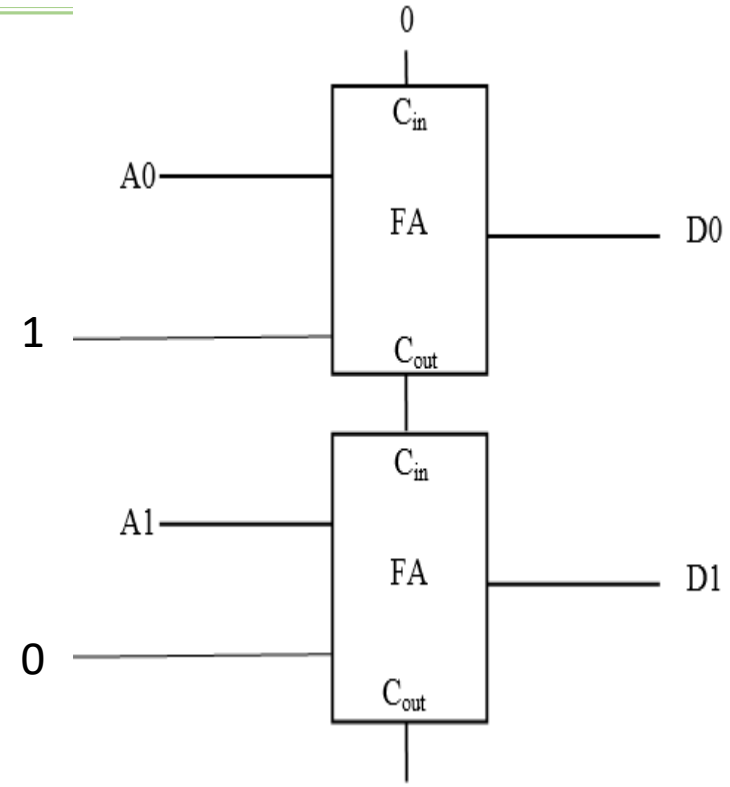
Part 3: Types of ALU Micro-operations (Arithmetic micro-operations)

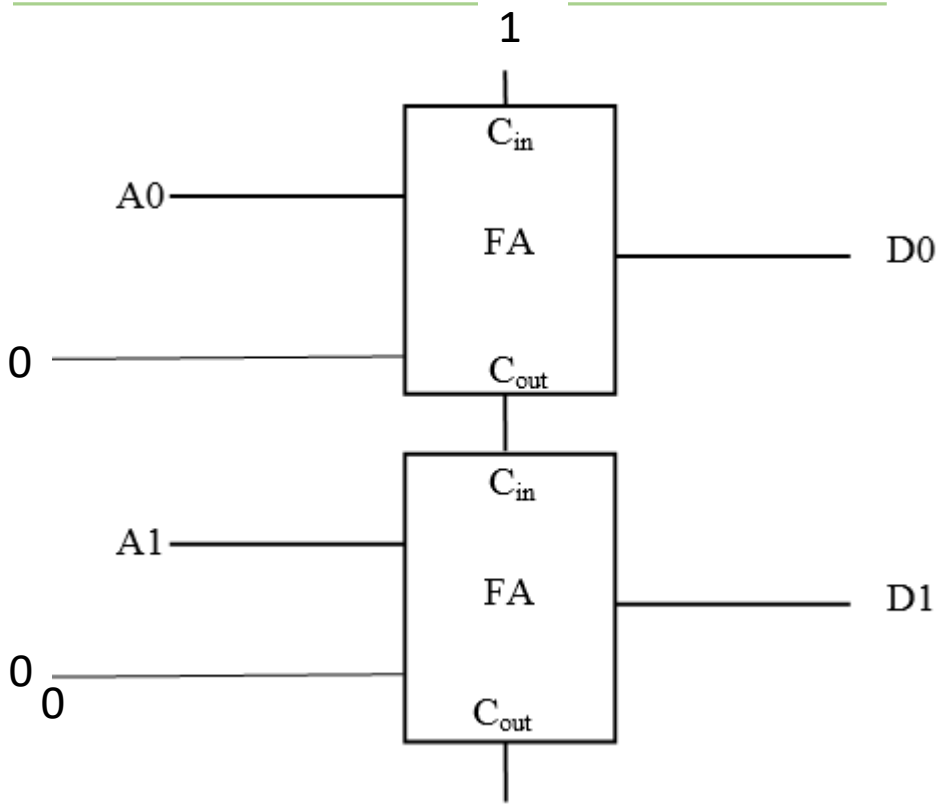
- Increment:

  D ← A+ 1

A	0011
	0001

D	0100





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

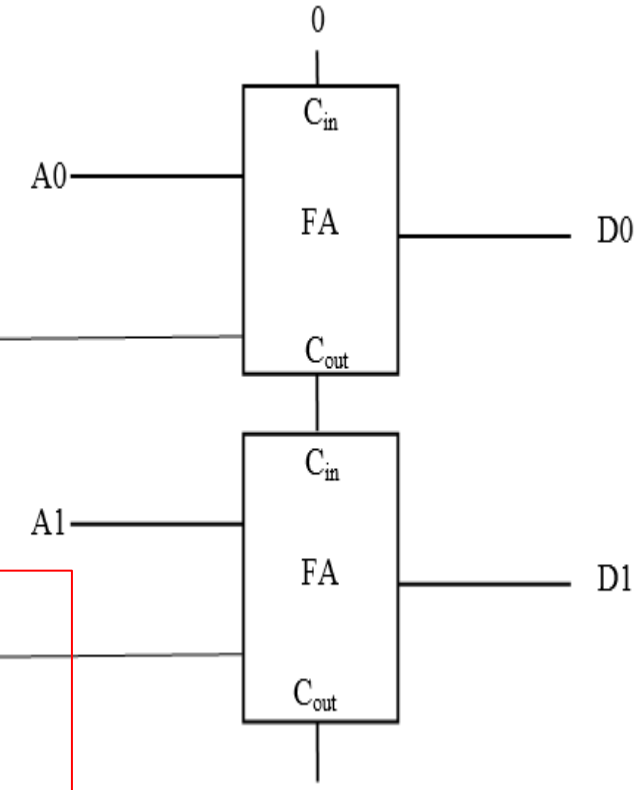
- Decrement:

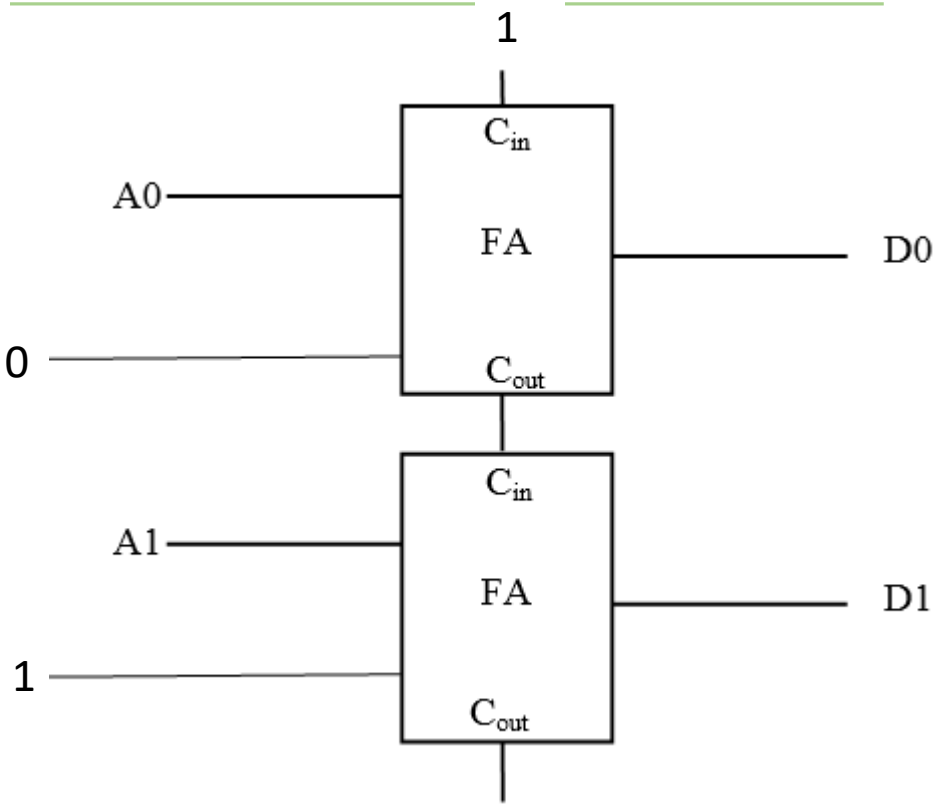
D ← A - 1

```
A      0101
       1111
       --------
D      0100
```

```
       0001
       1110
   +      1
   ----------
       1111
```

- 2's complement OF 1 IS ALL ONES





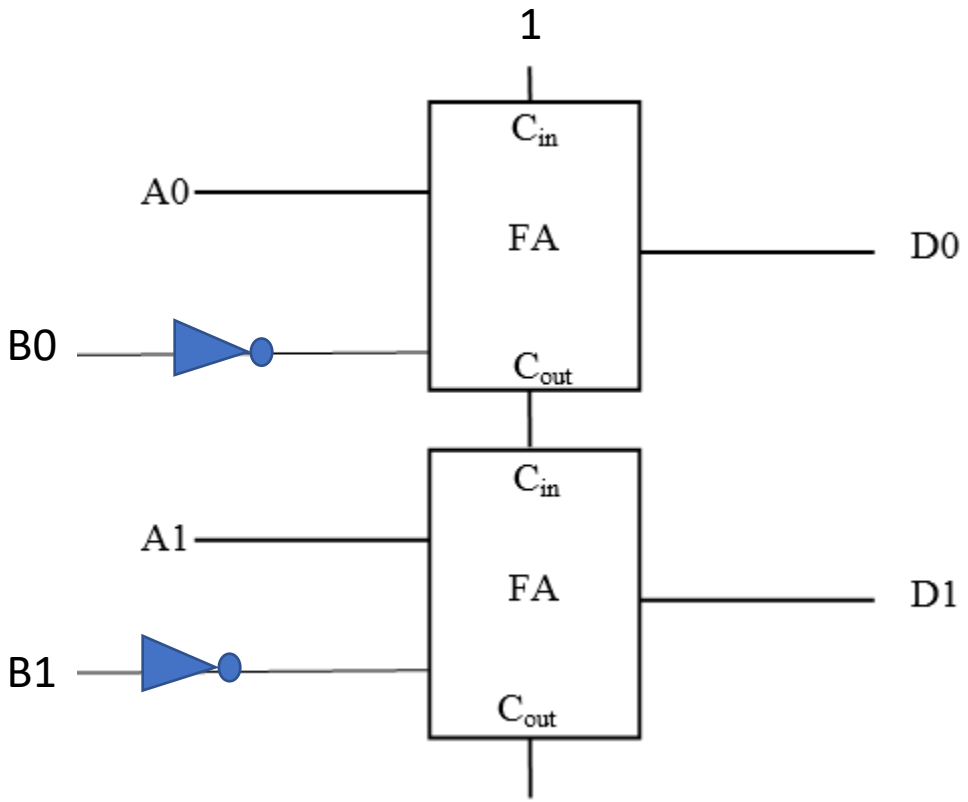
Part 3: Types of ALU Micro-operations (Arithmetic micro-operations)

- 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



0011

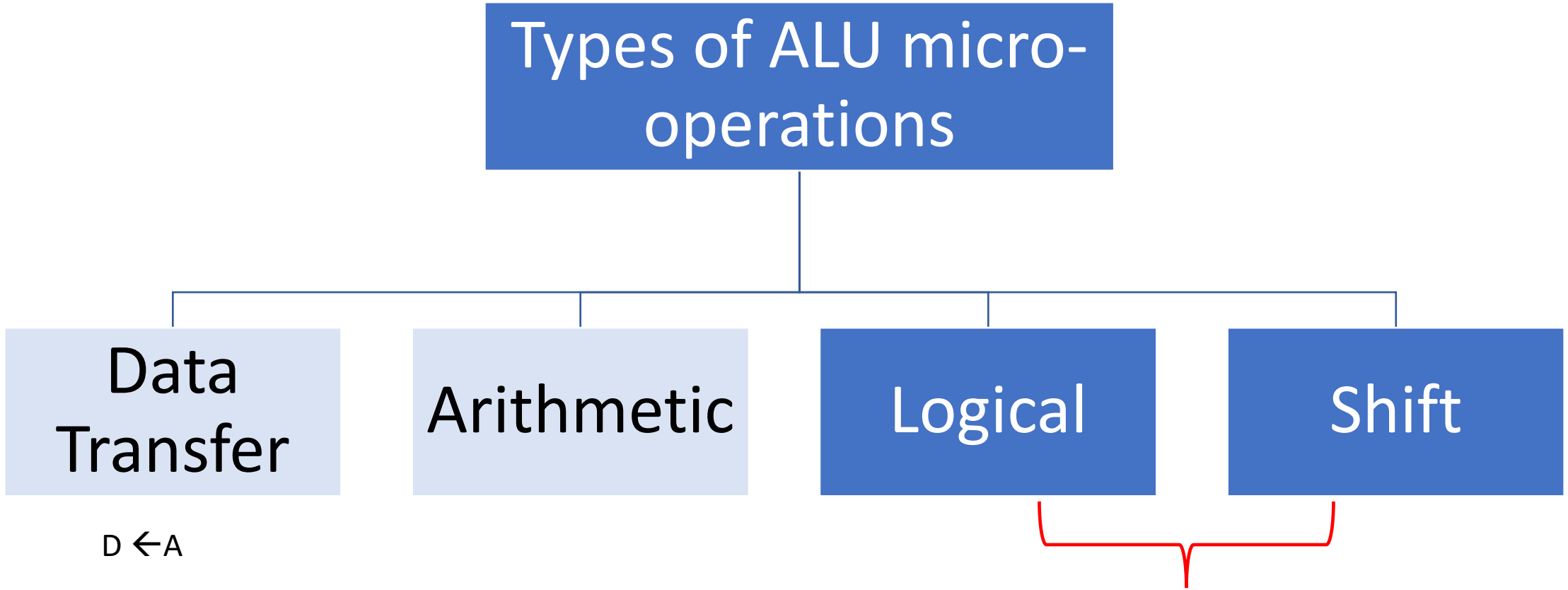
1's complement 1100

\+ 1

1101

- 2's complement, this is -3

Types of ALU Micro-operations



Thank You