Video chapters

Chapter-1 (Introduction): Boolean Algebra, Types of Computer, Functional units of digital system and their interconnections, buses, bus architecture, types of buses and bus arbitration. Register, bus and memory transfer. Processor organization, general registers organization, stack organization and addressing modes.

Chapter-2 (Arithmetic and logic unit) : Look ahead carries adders. Multiplication: Signed operand multiplication, Booth's algorithm and array multiplier. Division and logic operations. Floating point arithmetic operation, Arithmetic & logic unit design. IEEE Standard for Floating Point Numbers

Chapter-3 (Control Unit) : Instruction types, formats, instruction cycles and sub cycles (fetch and execute etc), micro-operations, execution of a complete instruction. Program Control, Reduced Instruction Set Computer,. Hardwire and micro programmed control: micro programme sequencing, concept of horizontal and vertical microprogramming.

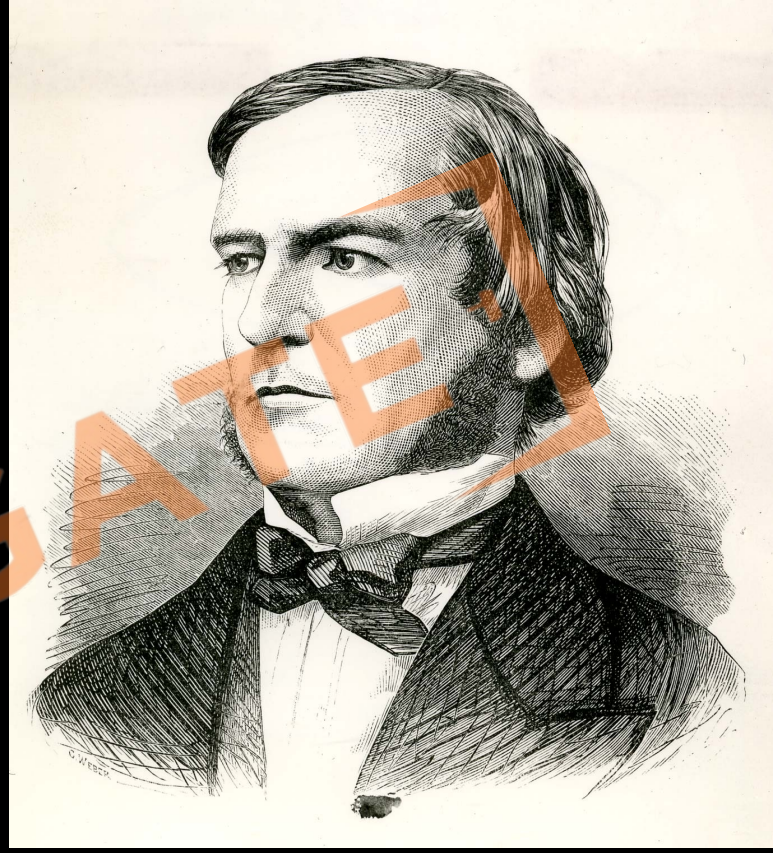
Chapter-4 (Memory) : Basic concept and hierarchy, semiconductor RAM **memories**, 2D & 2 1/2D memory organization. ROM memories. Cache memories: concept and design issues & performance, address mapping and replacement Auxiliary memories: magnetic disk, magnetic tape and optical disks Virtual memory: concept implementation.

Chapter-5 (Input / Output) : Peripheral devices, 1/0 interface, 1/0 ports, Interrupts: interrupt hardware, types of interrupts and exceptions. Modes of Data Transfer: Programmed 1/0, interrupt initiated 1/0 and Direct Memory Access., 1/0 channels and processors. Serial Communication: Synchronous & asynchronous communication, standard communication interfaces.

Chapter-6 (Pipelining): Uniprocessing, Multiprocessing, Pipelining, Speed UP, Structural hazards, Control hazards, Data hazards, Operand Forwarding.

Boolean algebra

- The signal in most present day electronic digital system uses just two discrete values and are therefore said to be binary.
- Boolean algebra was introduced by George Boole in his first book The Mathematical Analysis of Logic (1847), and set forth more fully in his An Investigation of the Laws of Thought (1854).
- George boole introduced the concept of binary number system in the studies of the mathematical theory of logic and developed its algebra known as Boolean algebra.



Boolean algebra

1. In mathematics and mathematical logic, Boolean algebra is the branch of algebra in which the values of the variables are the truth values true and false, usually denoted 1 and 0 respectively.

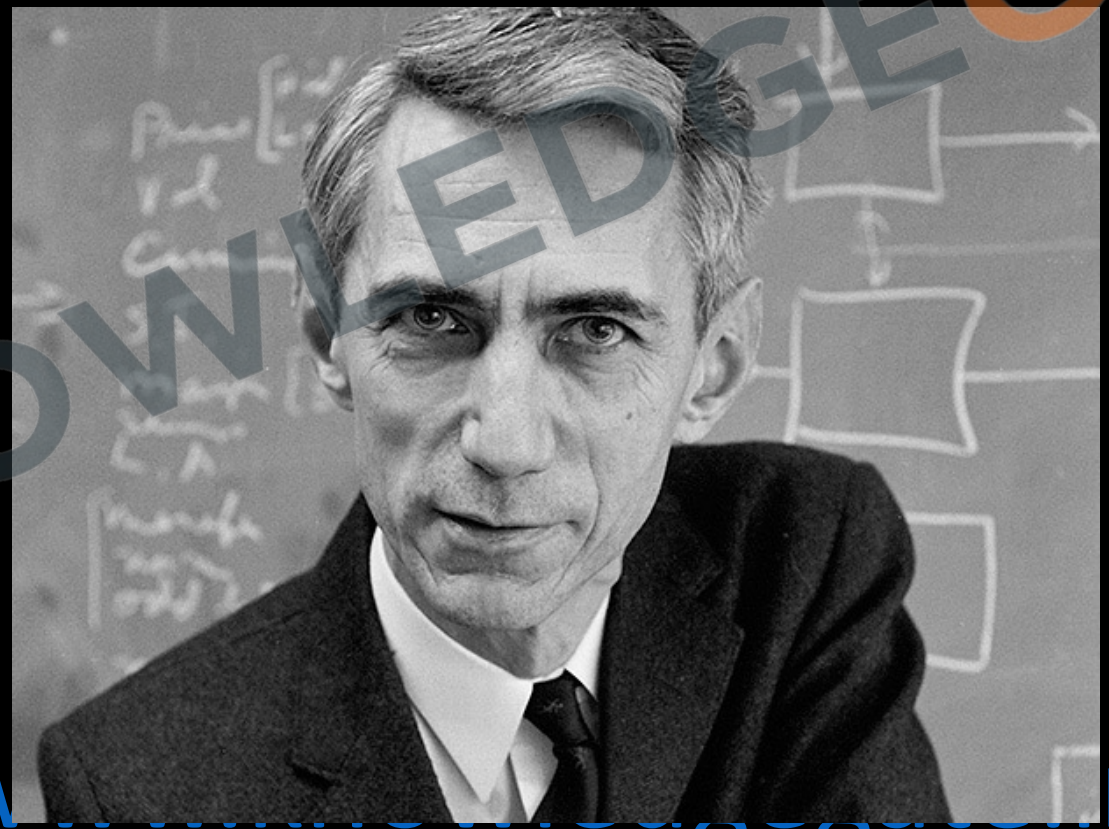
2. Instead of elementary algebra where the values of the variables are numbers, and the prime operations are addition and multiplication. The main operations of Boolean algebra are
   - The conjunction and denoted as ∧
   - The disjunction or denoted as ∨
   - The negation not denoted as ¬

Turing Machine

- The Church-Turing thesis states that any algorithmic procedure that can be carried out by human beings/computer can be carried out by a Turing machine.(1936)
- It has been universally accepted by computer scientists that the Turing machine provides an ideal theoretical model of a computer.

Digital System

- In the 1930s, while studying switching circuits, Claude Shannon observed that one could also apply the rules of Boole's algebra in this setting, and he introduced switching algebra as a way to analyze and design circuits by algebraic means in terms of logic gates.
- These logic concepts have been adapted for the design of digital hardware since 1938 Claude Shannon (father of information theory), organized and systematized Boole’s work.



- Shannon already had at his disposal the Boolean algebra, thus he cast his switching algebra as the two-element Boolean algebra. Efficient implementation of Boolean functions is a fundamental problem in the design of combinational logic circuits.

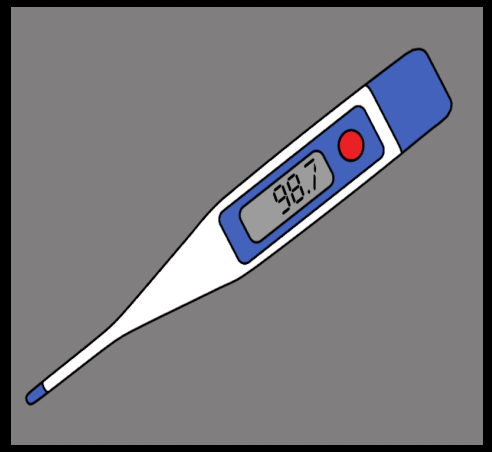
- Boolean constants are denoted by 0 or 1. Boolean variables are quantities that can take different values at different times. They may represent input, output or intermediate signals.

- Here we can have n number of variables usually written as a, b, c…. (lower case) and it satisfy all Boolean laws, which will be discussed later.

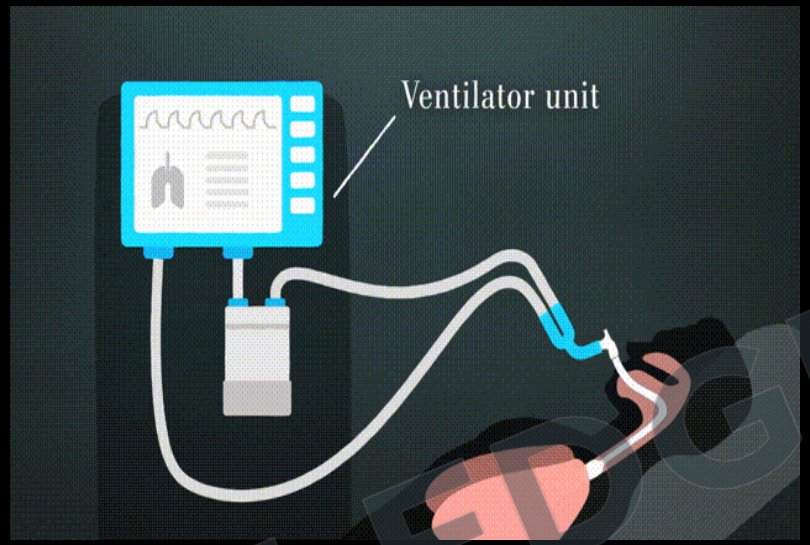
- Historically there have been 2 types of Computers:
  - **Fixed Program Computers / Dedicated device / Embedded system –** Their function is very specific and they couldn't be reprogrammed, e.g. Calculators washing machine.
  - **Stored Program Computers / General purpose computer / von Neumann architecture** These can be programmed to carry out many different tasks, applications are stored on them, hence the name.

Fixed Program Computers / Dedicated device / Embedded system

- They are designed to perform a specific task their functionality is written in terms of program which is permanently fused in a chipset. E.g. washing machine, microwave etc.

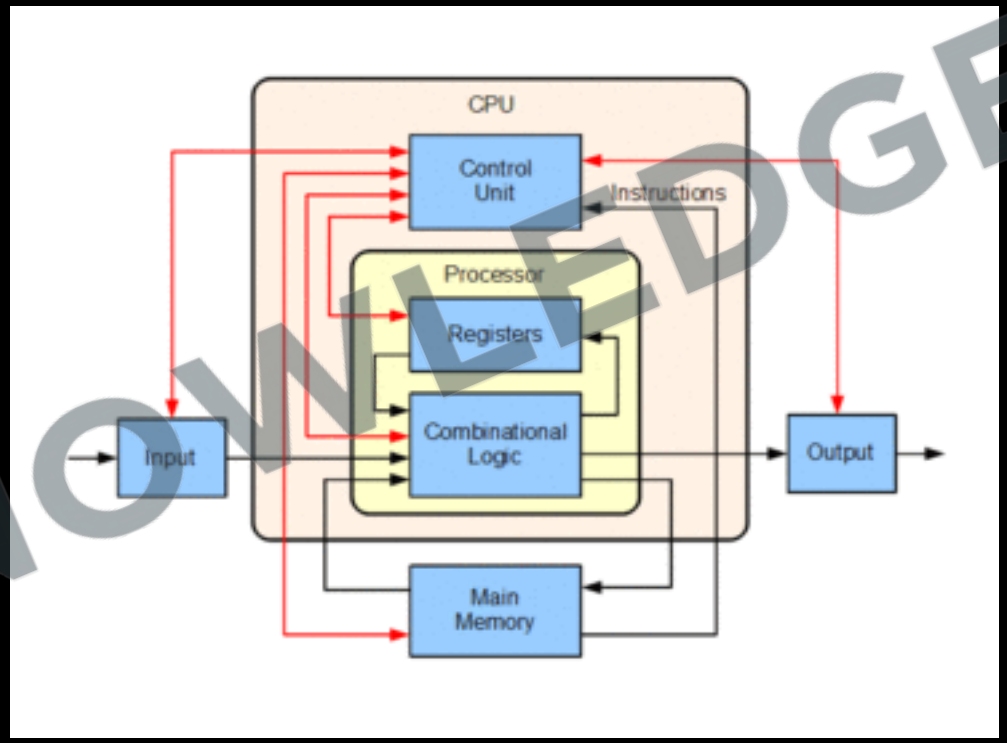


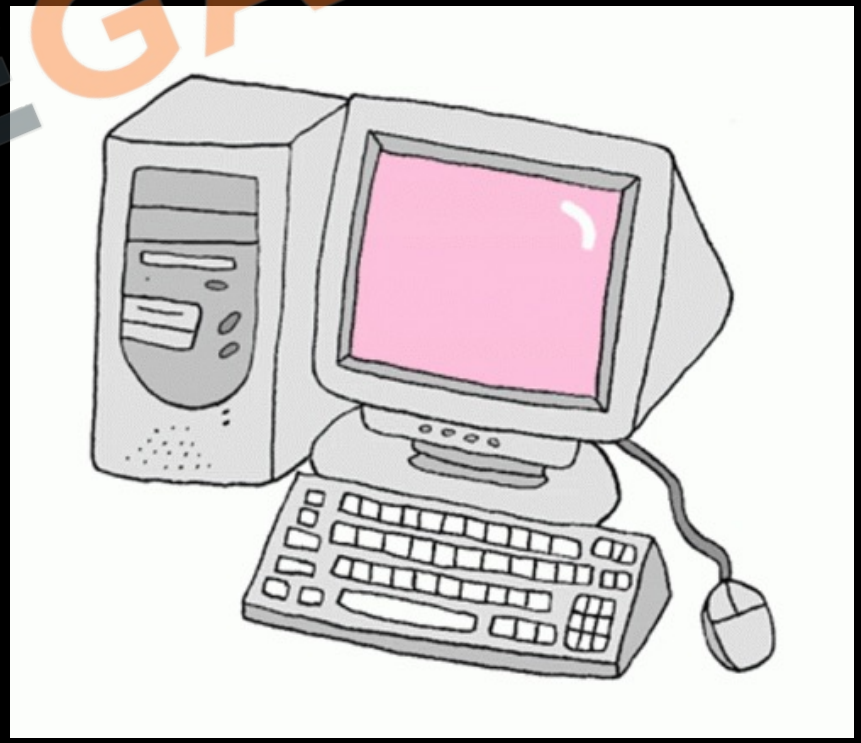
- Embedded system are kind of simple computers designed to do Specific task for e.g. microwave, washing machine etc., they have small microprocessors inside them. These microprocessors are hardwired and can not do everything, except specific task.



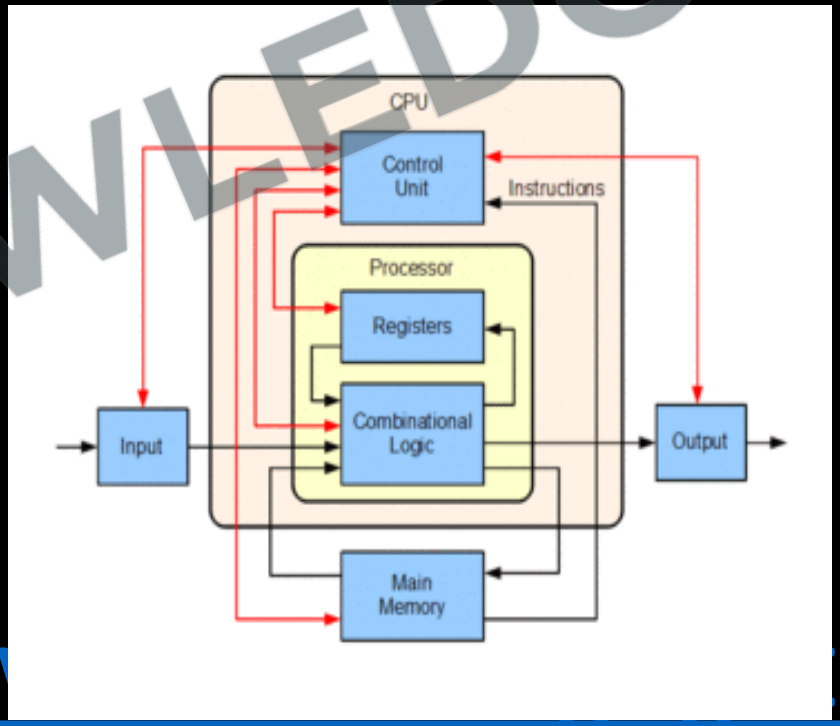
Stored Program Computers / General purpose computer / von Neumann architecture

- Modern computers are based on a stored-program concept introduced by John Von Neumann, Which can perform anything which can be theoretically done by a Turing machine. These computer works on stored program concept where a programmer, writes a program store it in the memory and then computer executes it, based on the requirement program can be changed and so does the functionality.

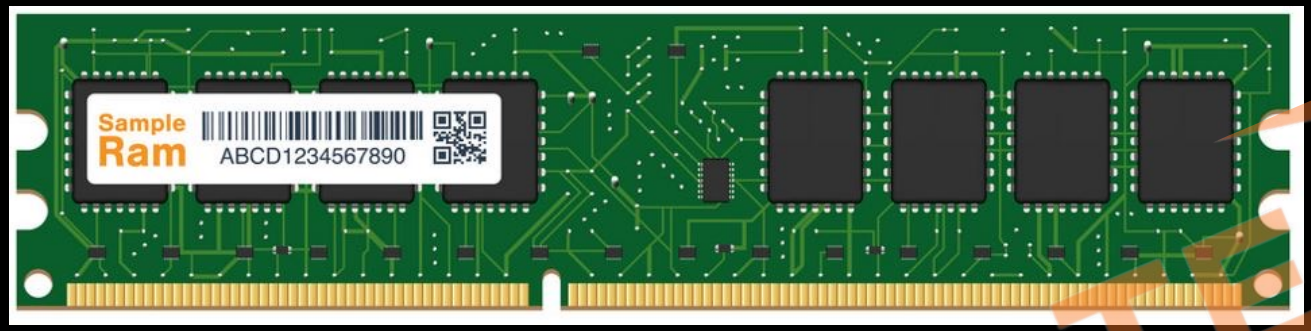




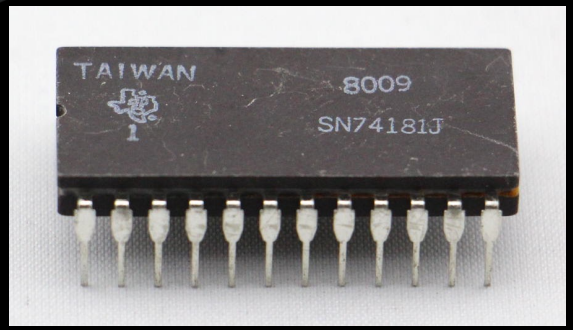
- **Machine / CPU architecture: -** In general a CPU contain 3 important components
  - **Control unit: -** Which generates control signal (master generator) to control every other part of the CPU. It directs all input and output flow, fetches code for instructions, and controls how data moves around the system.
  - **Registers: -** few important registers are in every processor like program counter which contains address of the next instruction then instruction register which contain address of the current register and base register which contain base address of program.
  - **ALU: -** ALU is a complex combination circuit which can perform arithmetic Operations, Bit Shifting Operations, and logical operation. e.g. Addition, Subtraction, Comparisons etc.



- Now the question is what are the main elements we need
  - Memory (store a program)



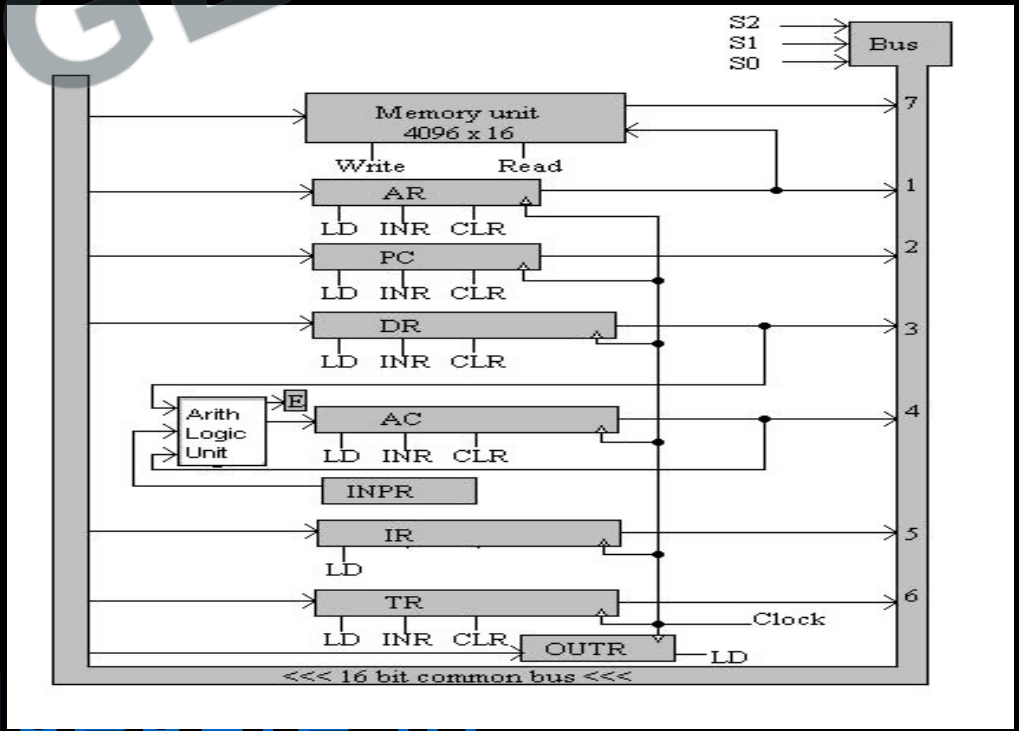
  - ALU (circuit which perform operations)



  - Register (fast memory, sequence of flip-flop) (load, clear, increment pin)

- Timing circuit (sequence counter) (to order certain operations, like fetch, decode, execute), will generate timing signals

- Control unit will generate control signals-to select registers, select other circuit, to give inputs to registers, So we need a special unit called control unit, which will give signals to all the components



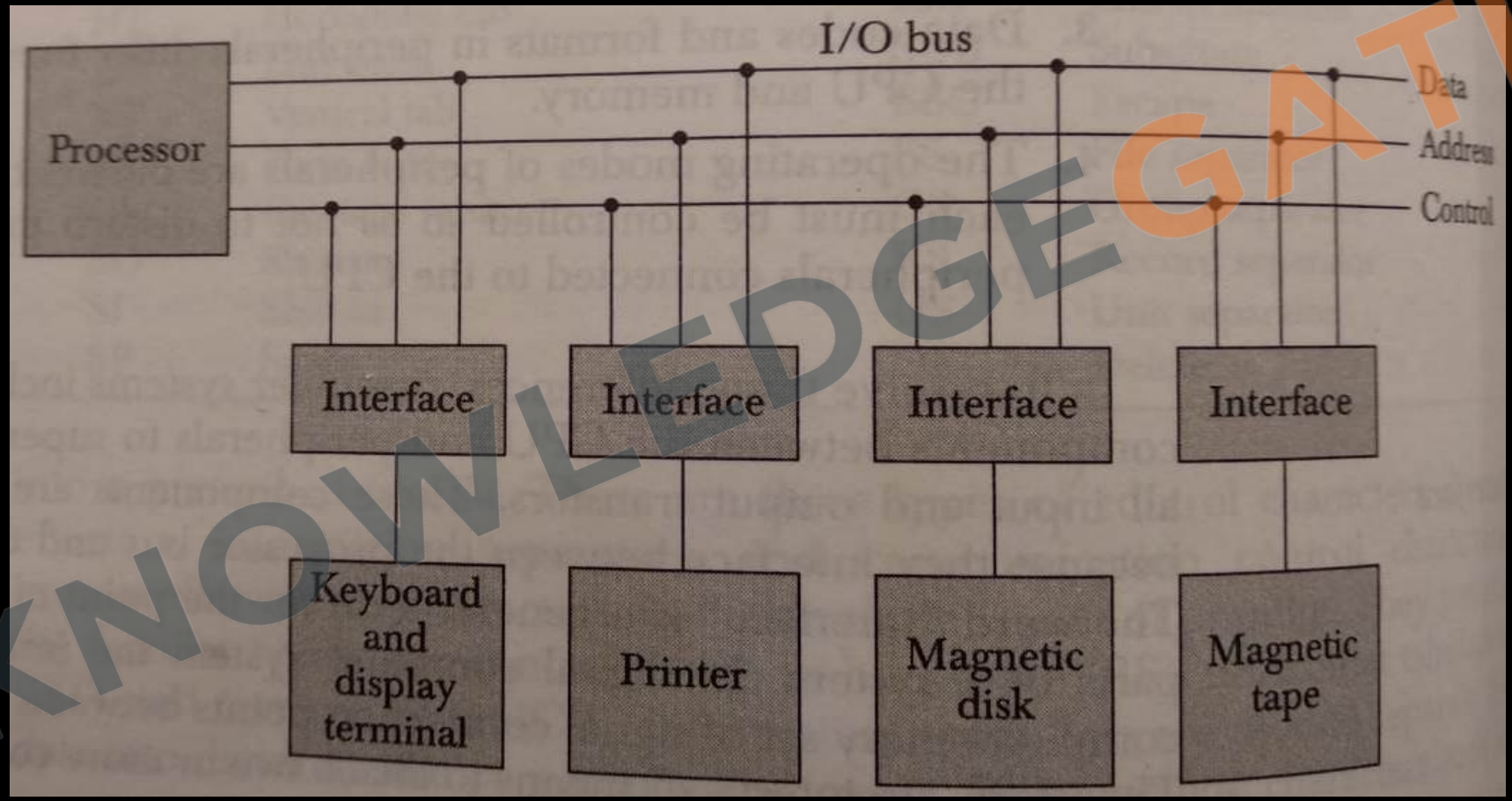
- Flags – one-bit information

- Bus – using which we will connect different component together, and perform data transfer using multiplexer

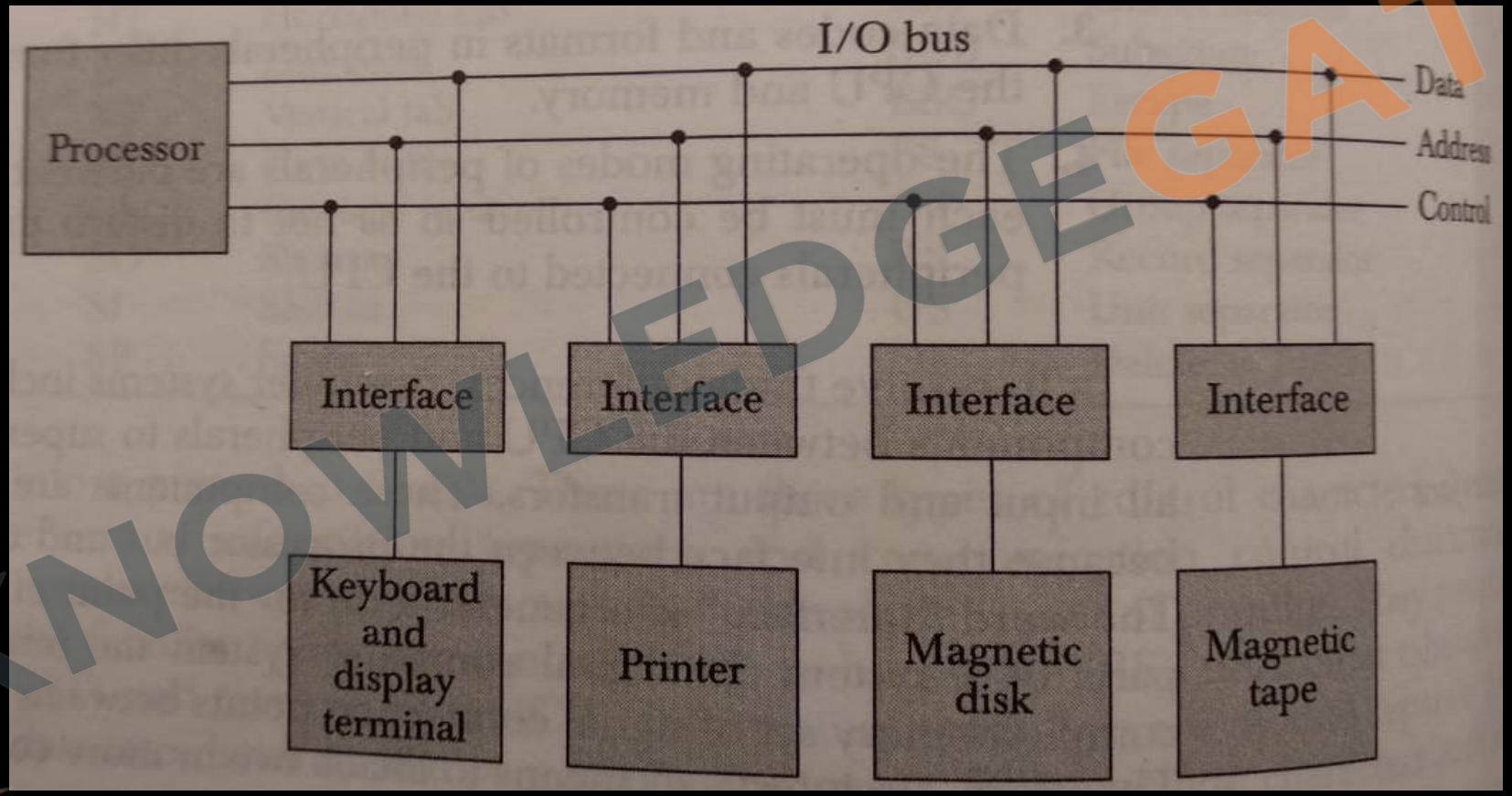
- how in general operation are performed?

  - ***memory -> register -> ALU (perform operation) register -> memory.***

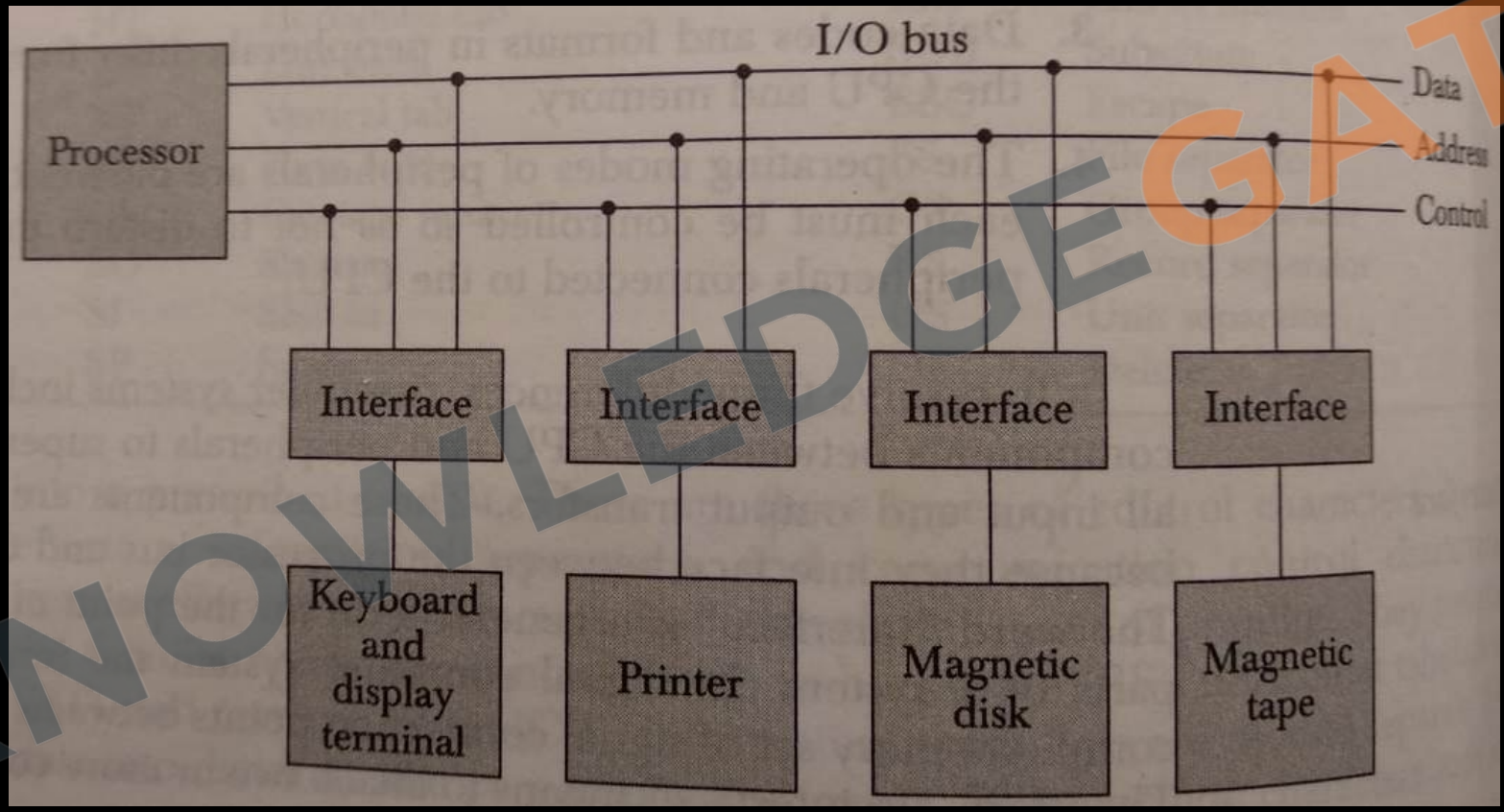
- **Address bus: -** Is used to identify the correct i/o devices among the number of i/o device , so CPU put an address of a specific i/o device on the address line, all devices keep monitoring this address bus and decode it, and if it is a match then it activates control and data lines.



- **Control bus: -** After selecting a specific i/o device CPU sends a functional code on the control line. The selected device(interface) reads that functional code and execute it. E.g. i/o command, control command, status command etc.



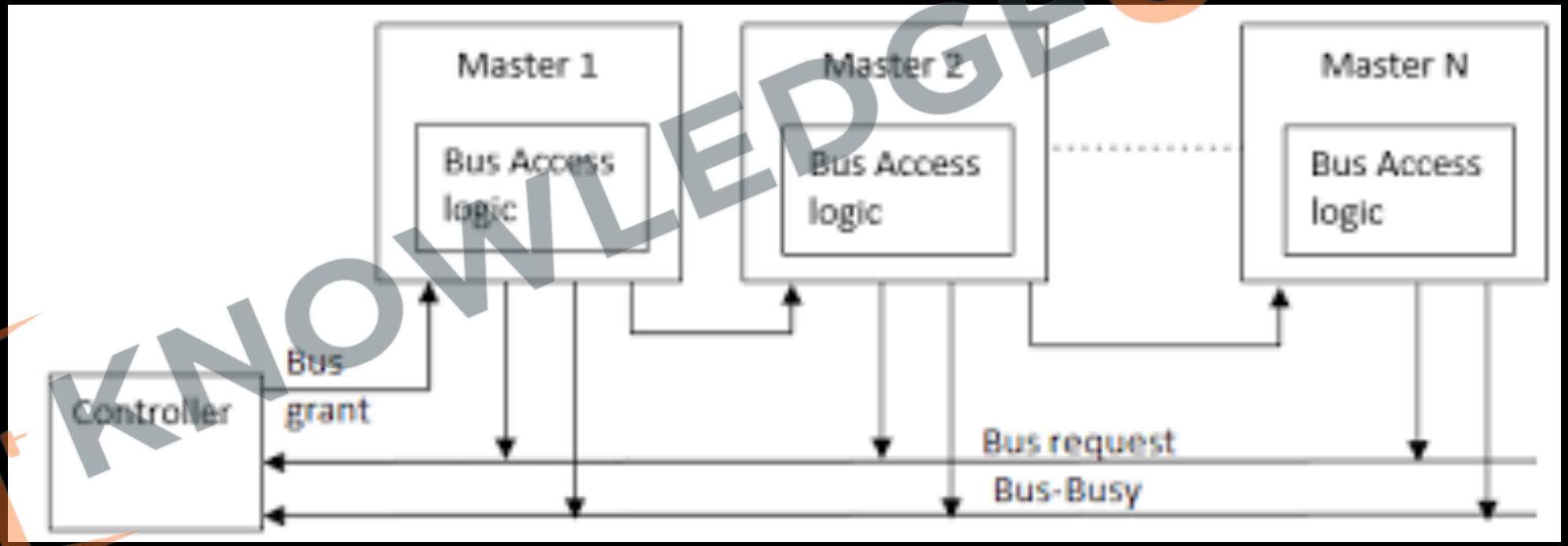
- **Data bus:** - In the final step depending on the operation either CPU will put data on the data line and device will store it or device will put data on the data line and CPU will store it.



Bus Arbitration

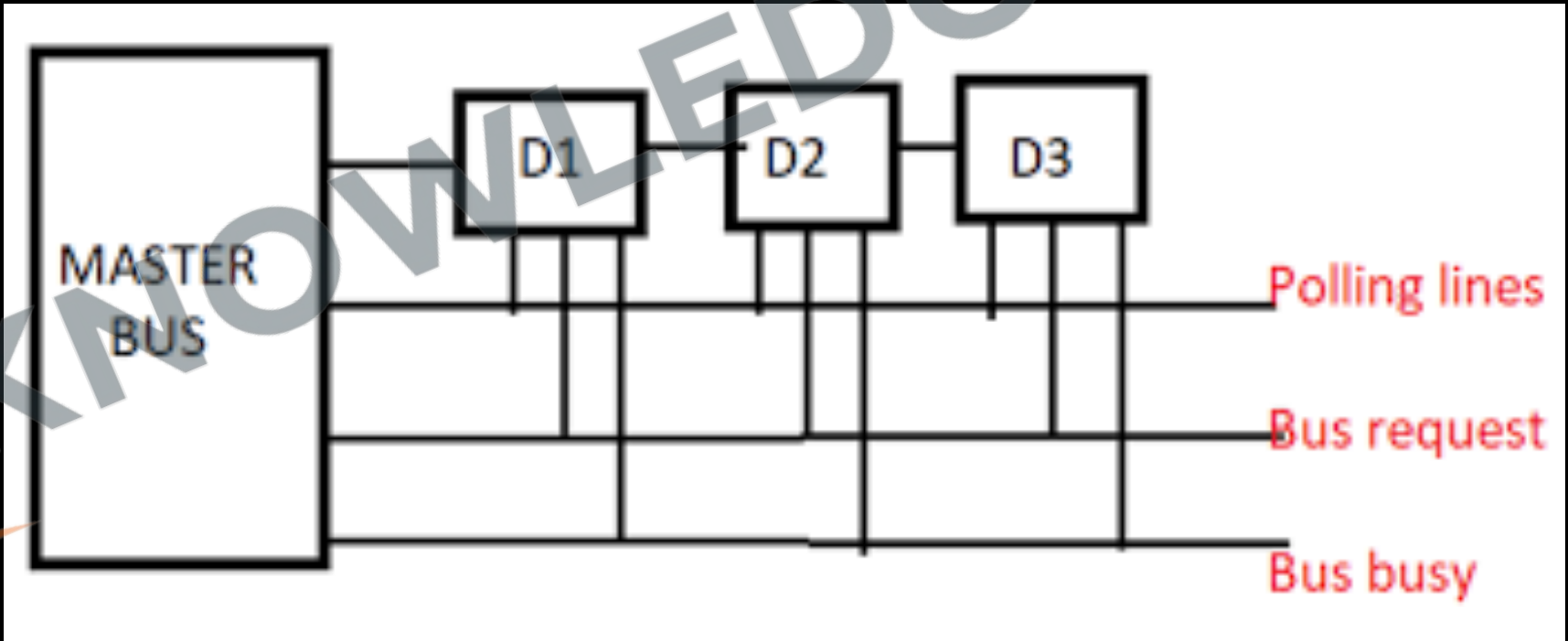
- **Bus Arbitration**: This is the method used to decide which device gets access to the common bus when multiple devices request it simultaneously, ensuring data integrity and system stability.
- **Conflict Resolution**: Without bus arbitration, simultaneous access could result in data corruption and system malfunctions, making this mechanism essential for orderly and reliable data transfer.

- **Daisy Chaining method:** It is a simple and cheaper method where all the bus masters use the same line for making bus requests. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, therefore any other requesting module will not receive the grant signal and hence cannot access the bus.



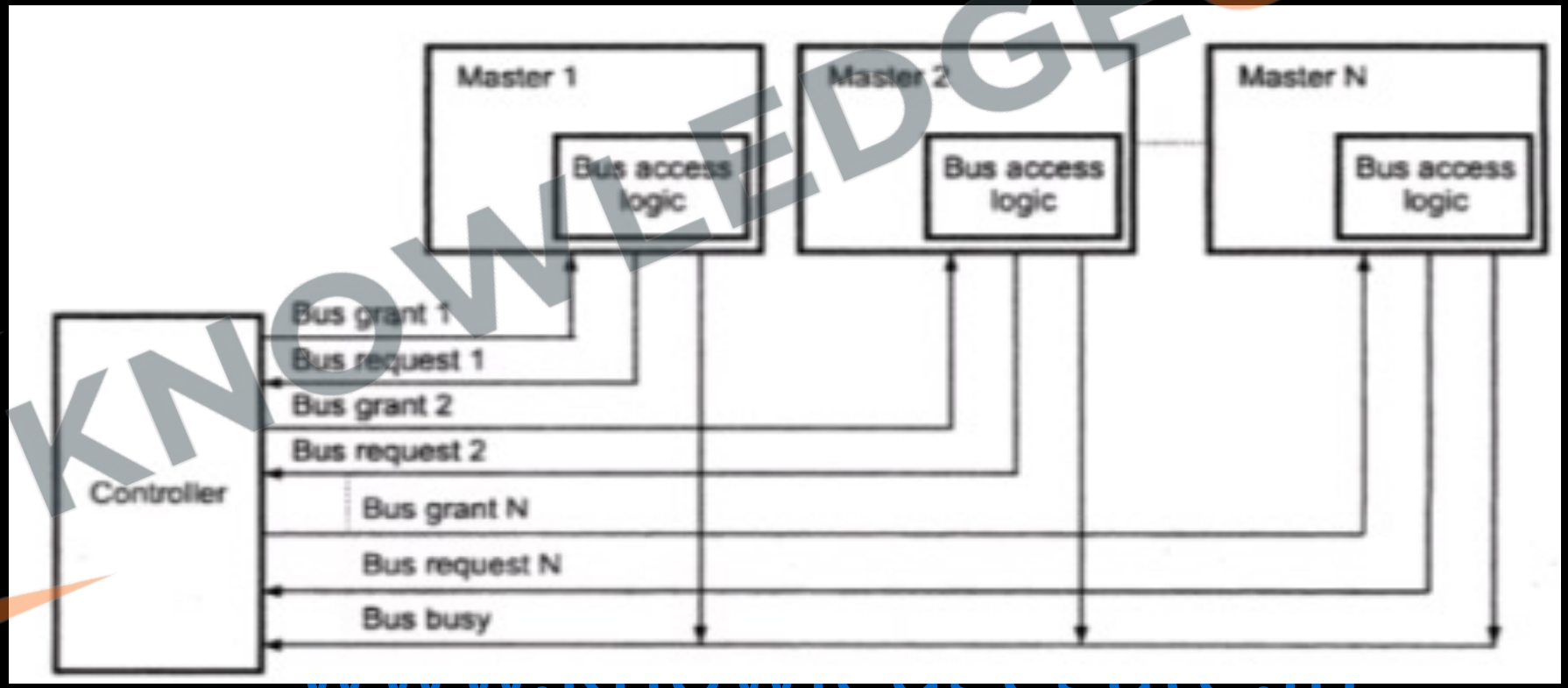
Polling

- **Address Generation:** In this method, a controller generates unique addresses for each master (device) based on their priority. The number of address lines correlates with the number of masters in the system.
- **Sequence & Activation**: The controller cycles through the generated addresses. When a master recognizes its own address, it activates a "busy" signal and gains access to the bus for data transfer.

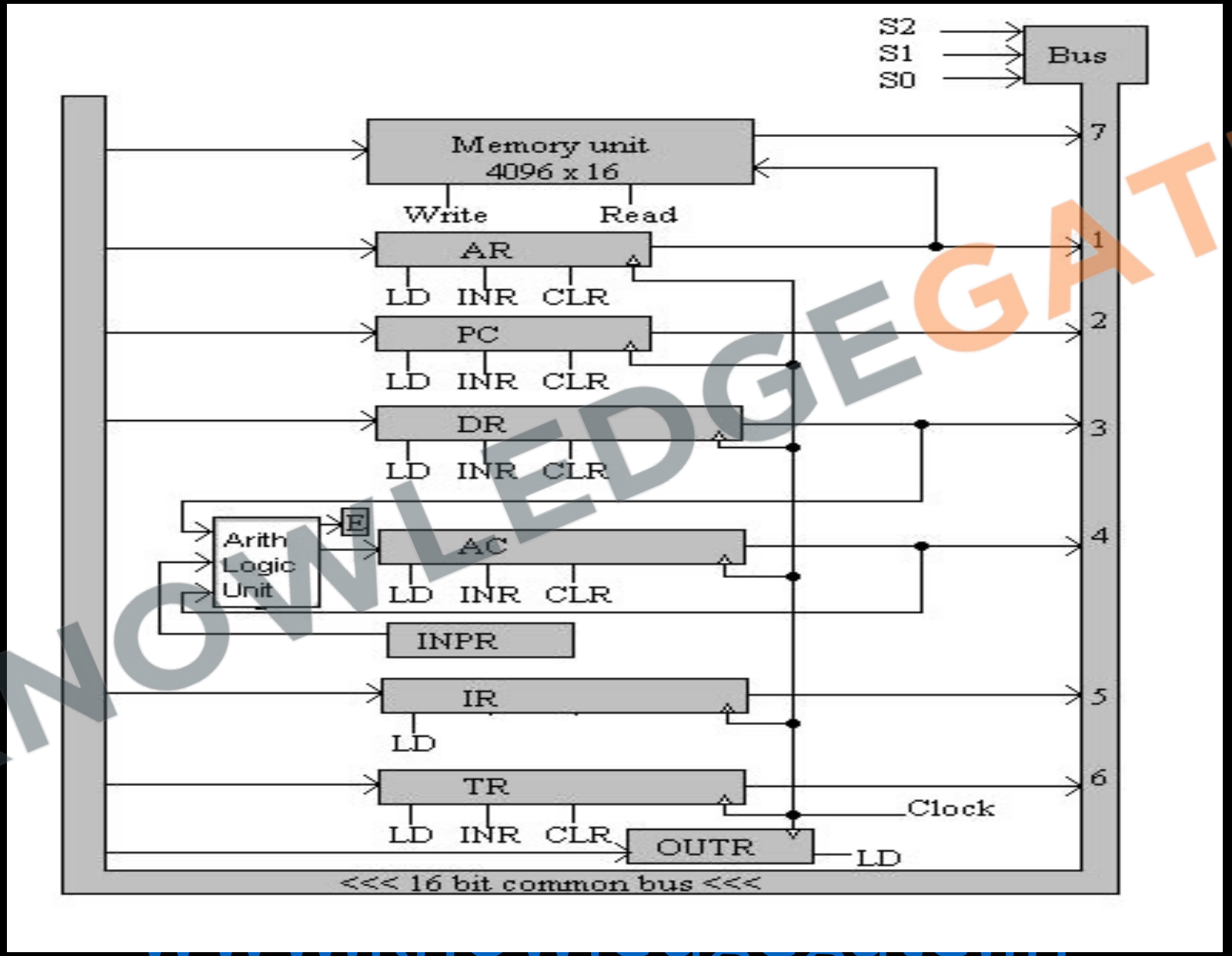


Fixed priority or Independent Request method

- In this, each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it.
- The built-in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.

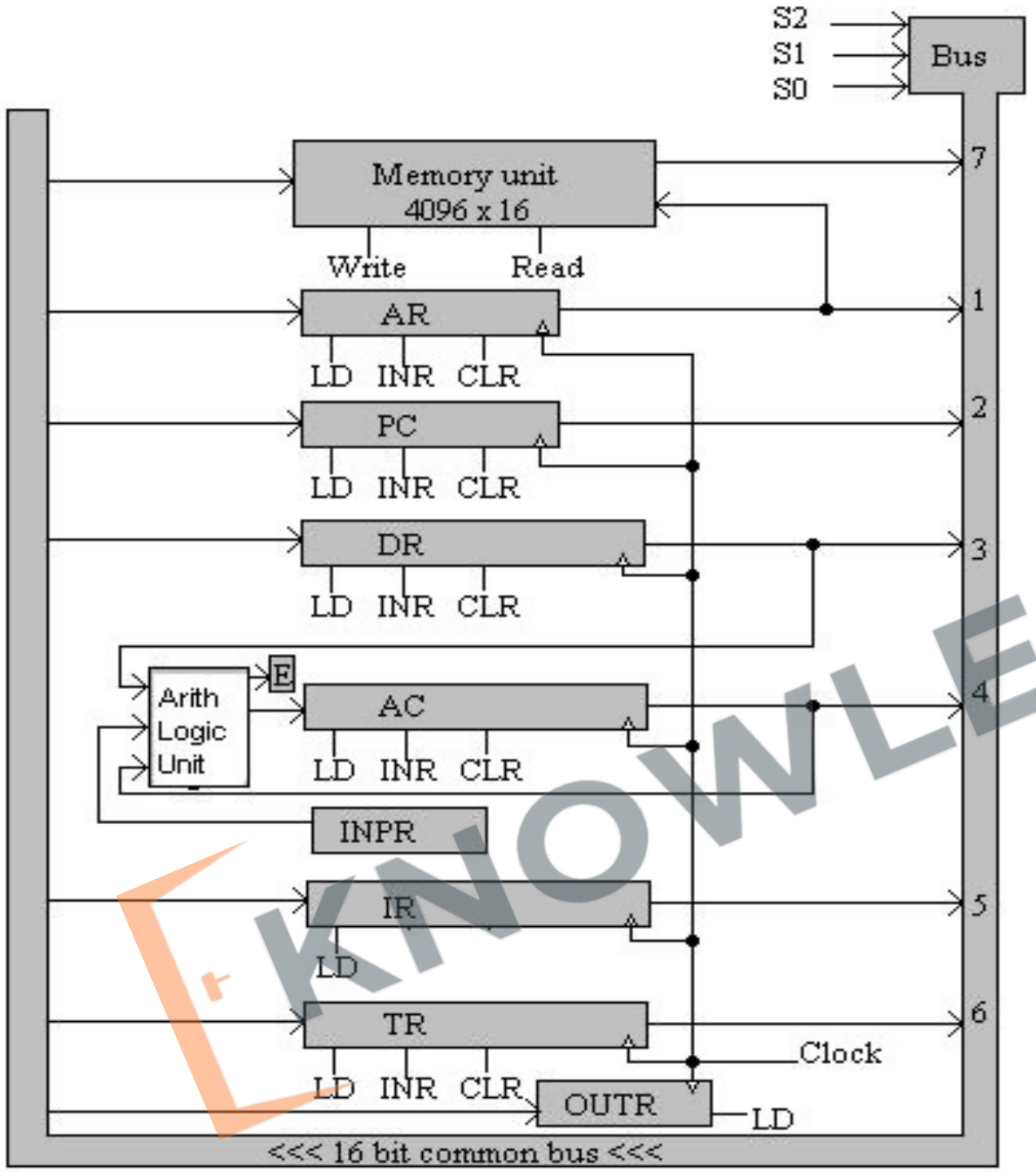


Processor Organization

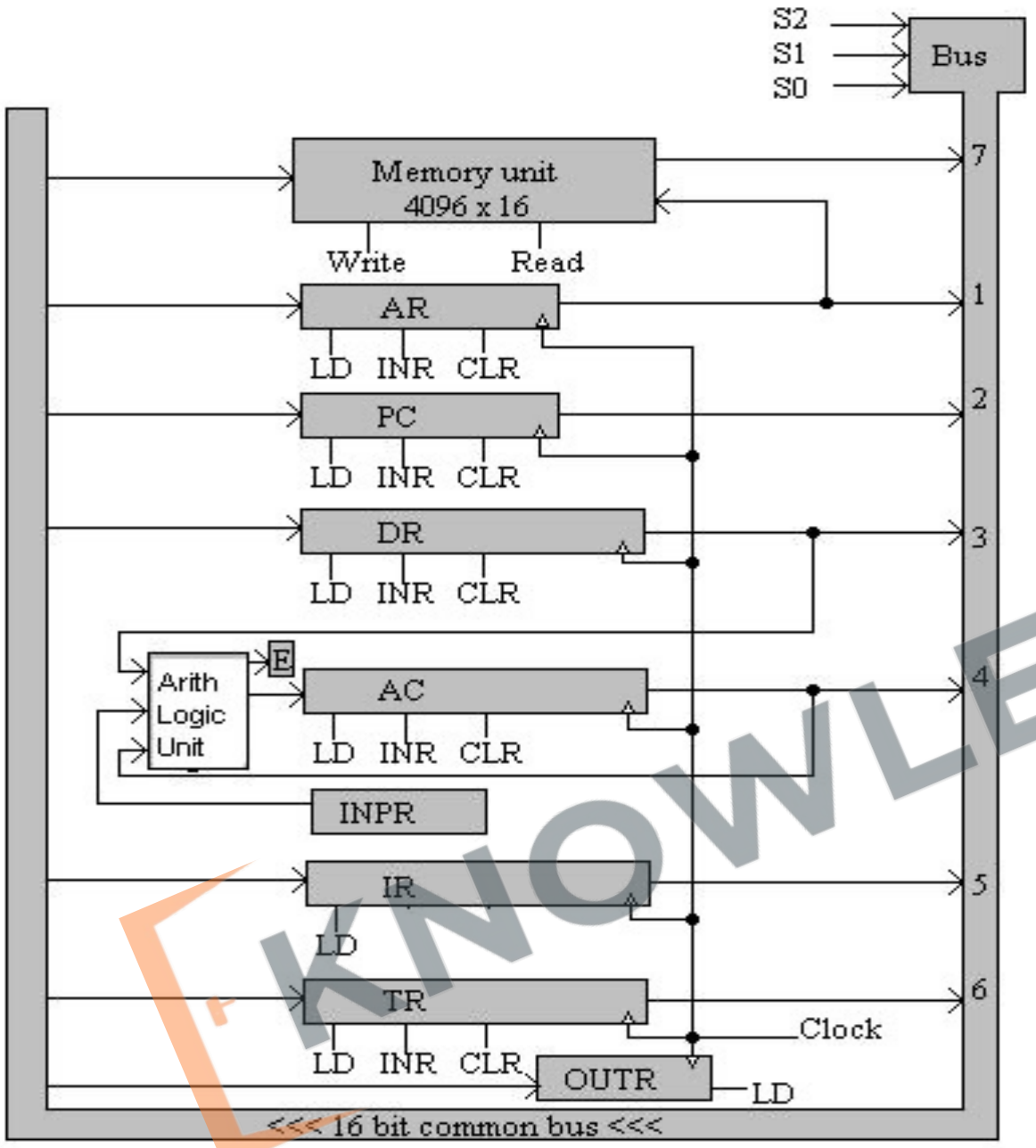


Registers - Registers refer to high-speed storage areas in the CPU. The data processed by the CPU are fetched from the registers. There are different types of registers used in architecture.

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

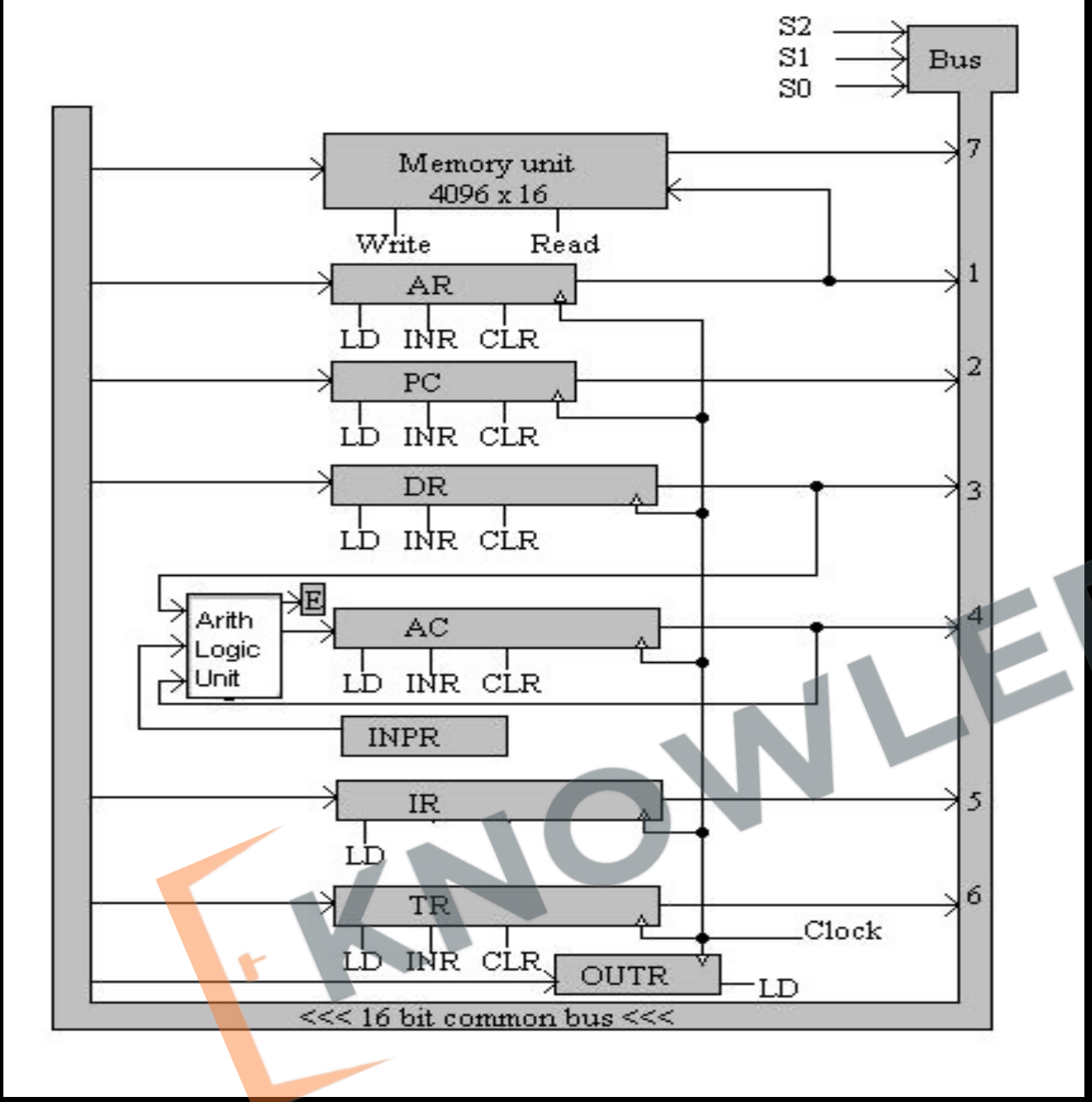


- The **data register (DR)** is a register which is used to store any data that is to be transferred to memory or which are fetched from memory.
- The **accumulator (AC)** register is a general-purpose processing register. Will hold the intermediate arithmetic and logic results of the operation performed in the ALU, as ALU is not directly connected to the memory.



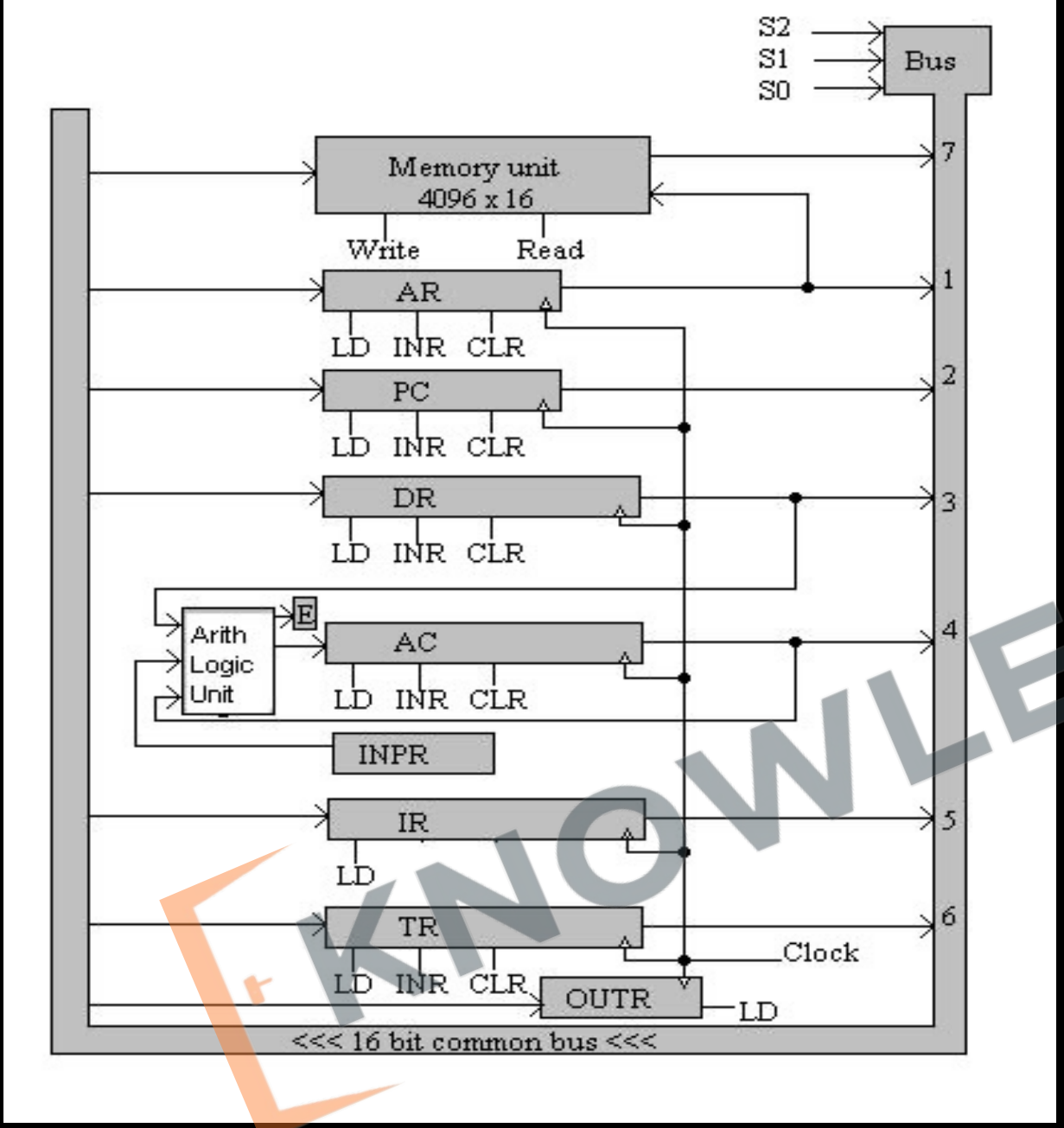
- **Instruction register**- is used to store instruction which we fetched from memory, so that we analyses the instruction using a decoder and can understand what instruction want to perform.

- The **temporary register (TR)** is used for holding temporary data during the processing.



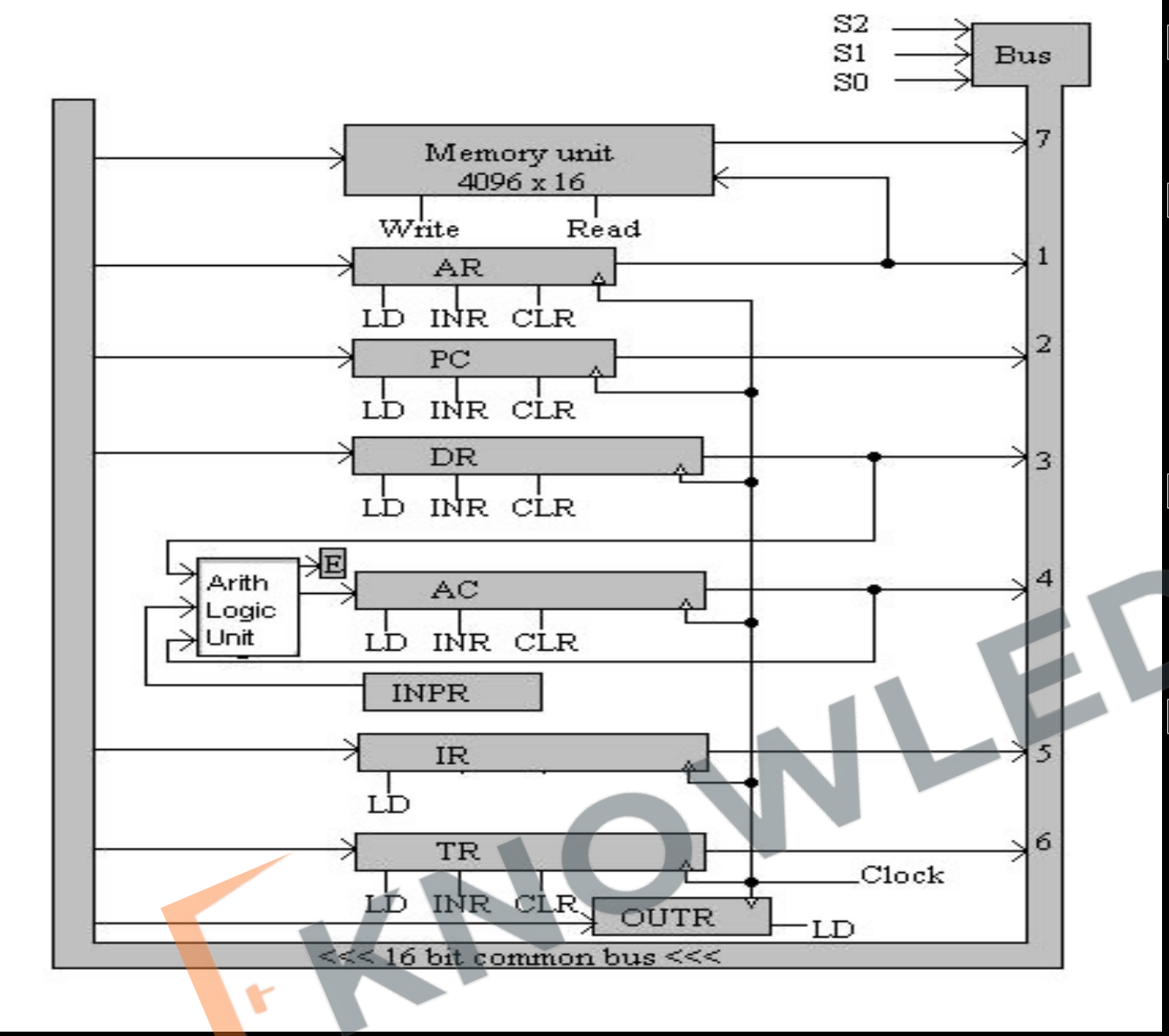
- The memory **address register (AR)** has 12 bits since this is the width of a memory address (always used least significant bits of bus). It stores the memory locations of instructions or data that need to be fetched from memory or stored in memory.

- The **program counter (PC)** also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed. The PC goes through a counting sequence and causes the computer to read sequential instructions previously stored in memory.

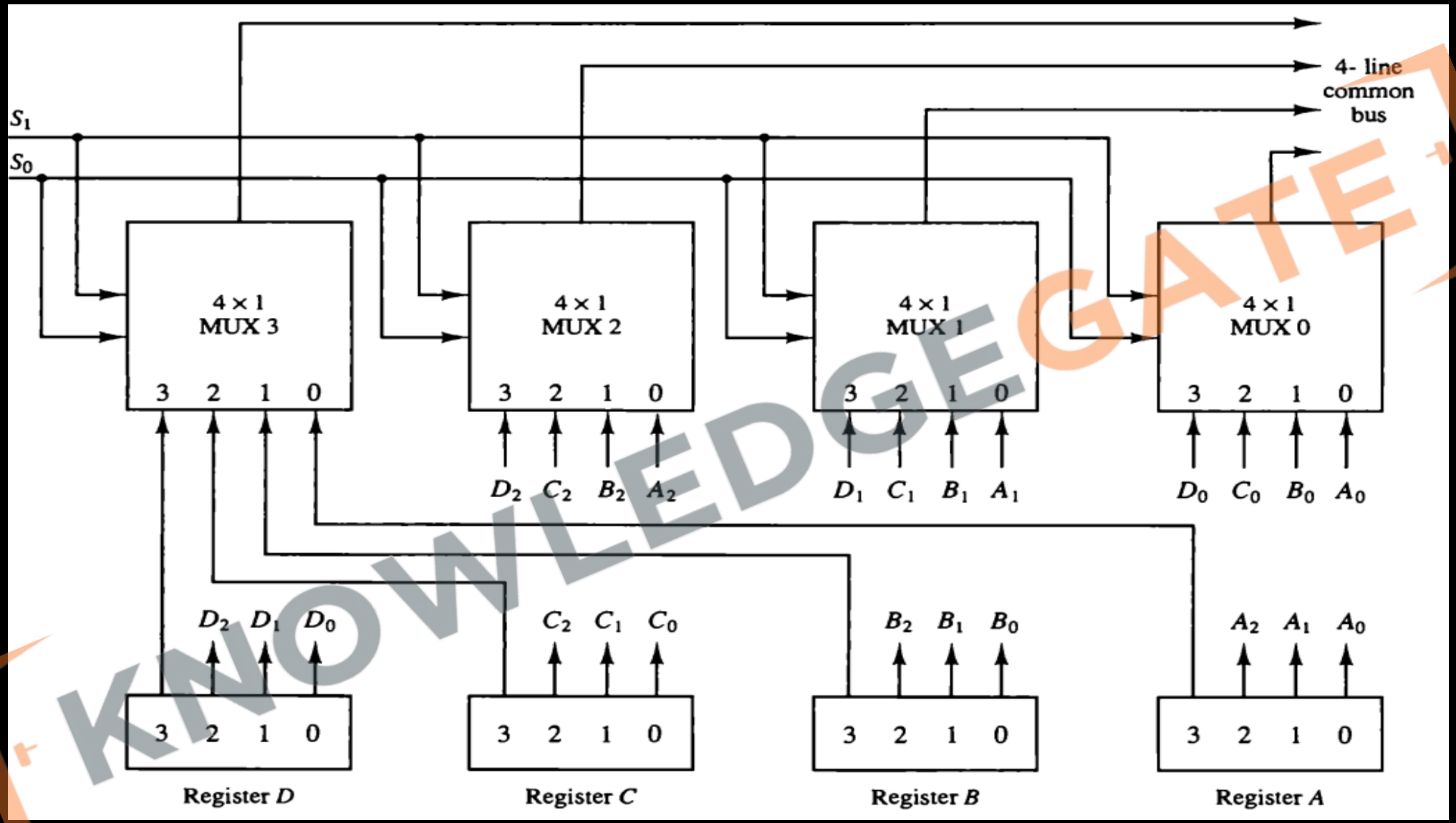


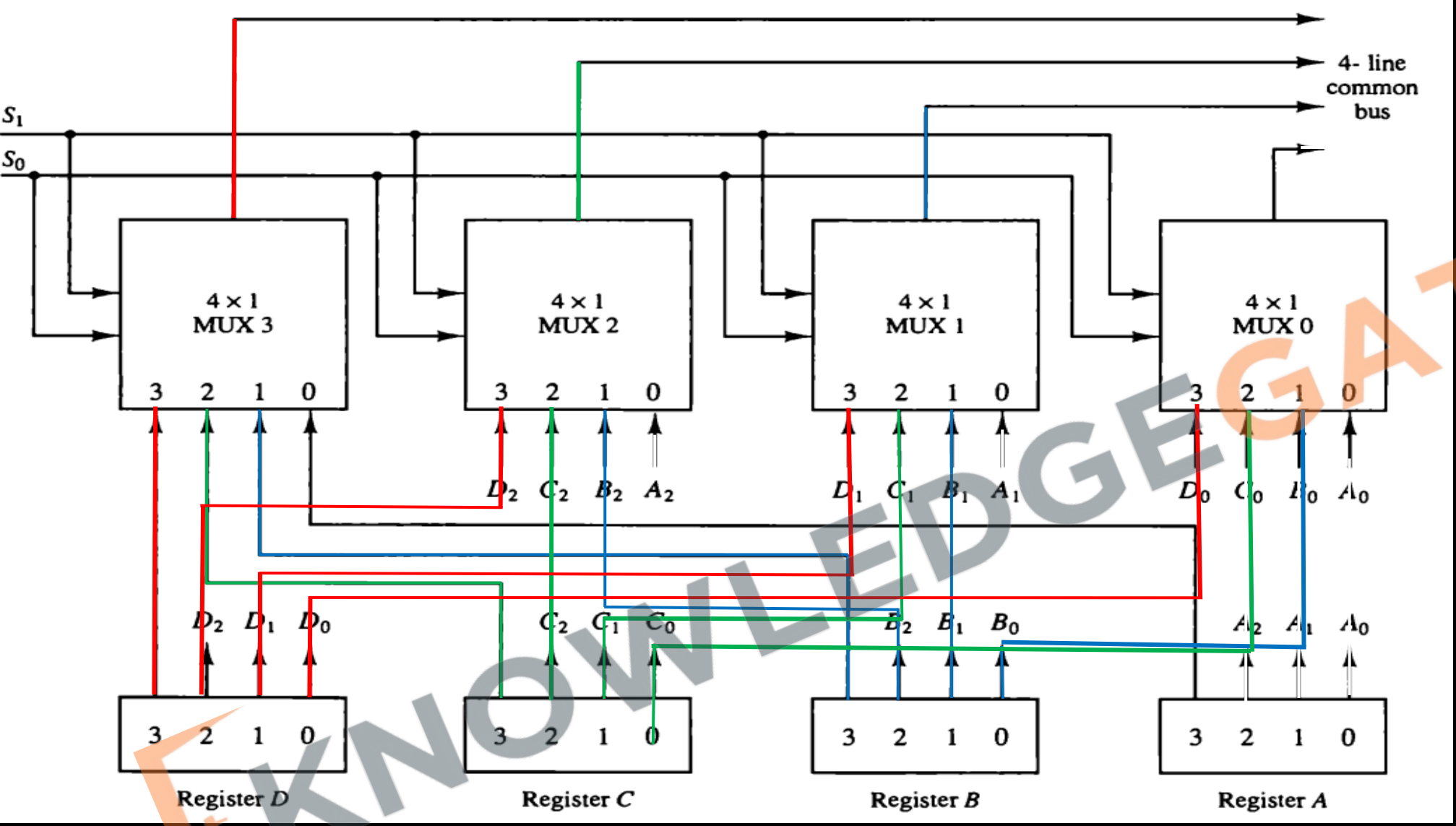
- Two registers are used for input and output. The **input register (INPR)** receives an 8-bit character from an input device, and pass it to ALU then accumulator and then to memory.

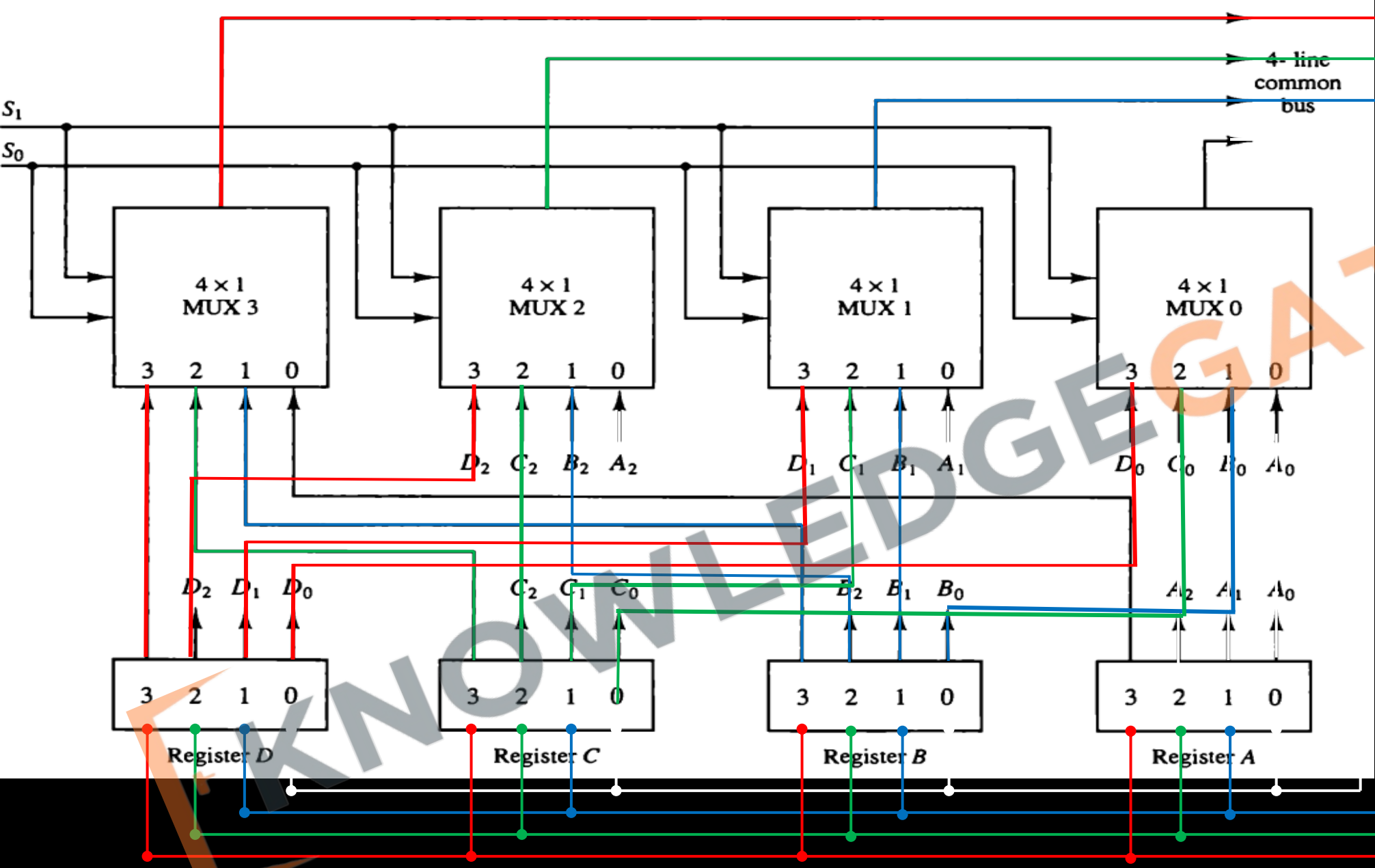
- The output **register (OUTR)** holds an 8-bit character for an output device, screen, printer etc.

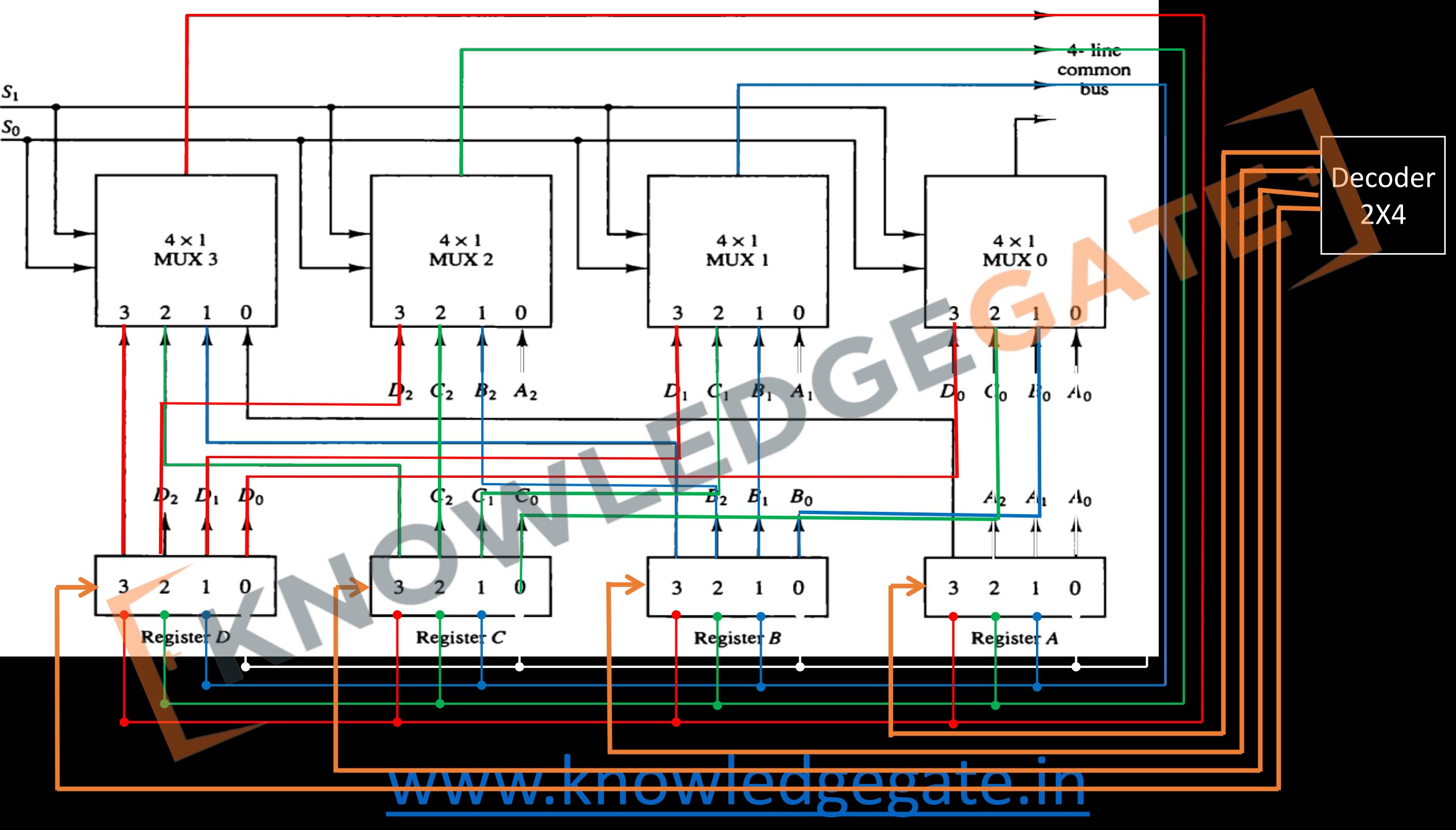


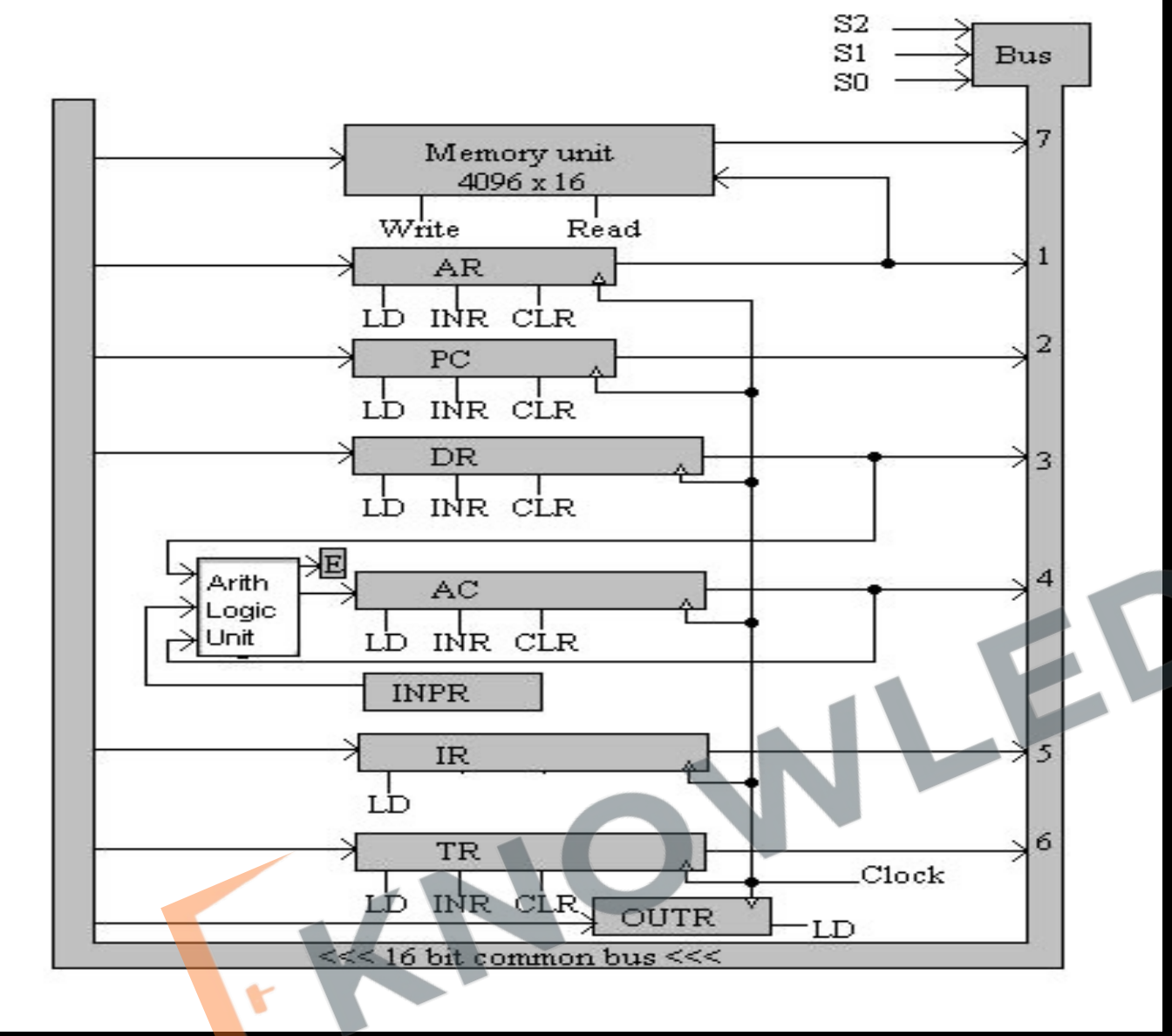
- The outputs of seven registers and memory are connected to the common bus.
- The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0.
- The number along each output shows the decimal equivalent of the required binary selection.
- Example: the number along the output of DR is 3. The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0 = 011$.



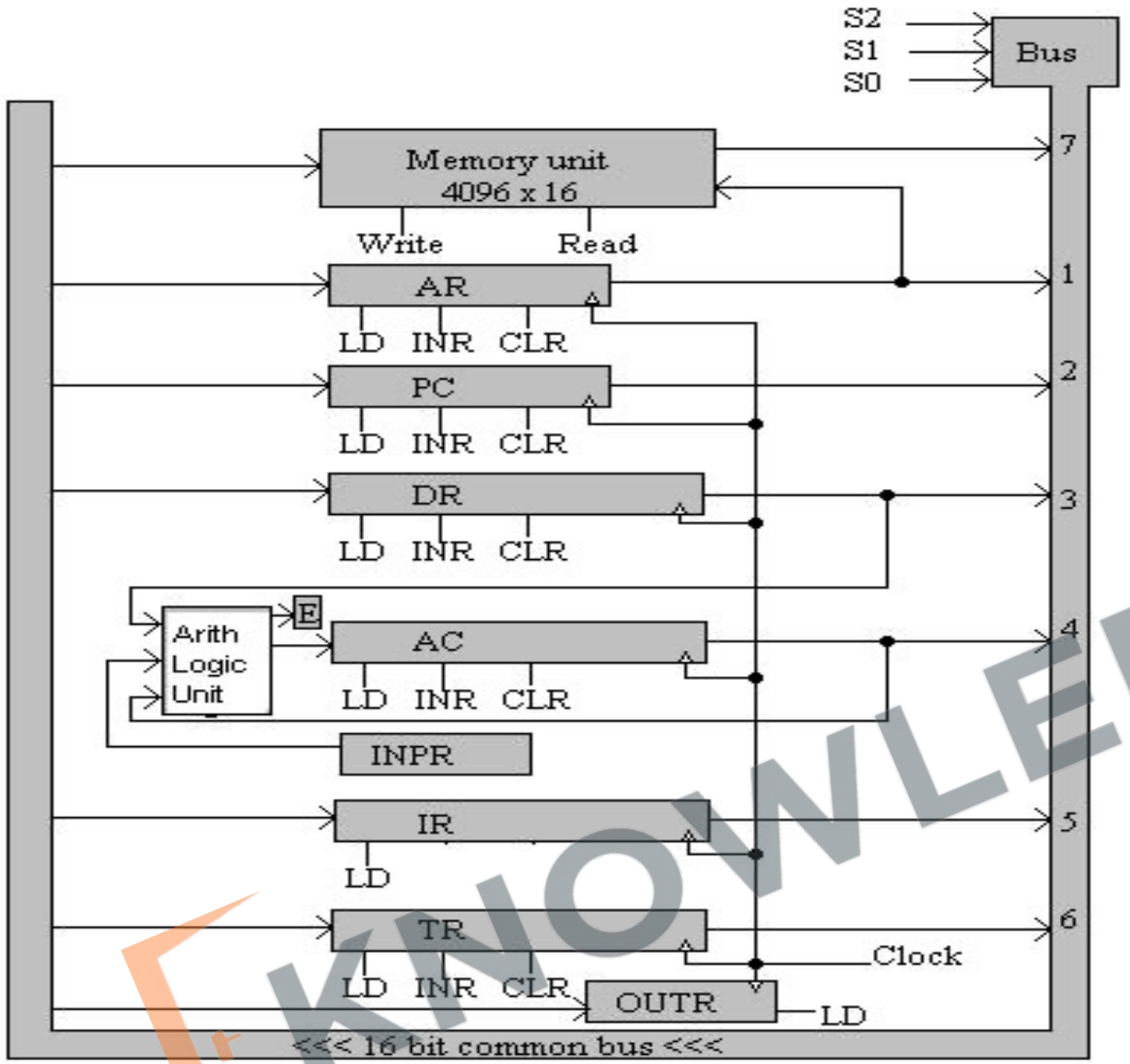




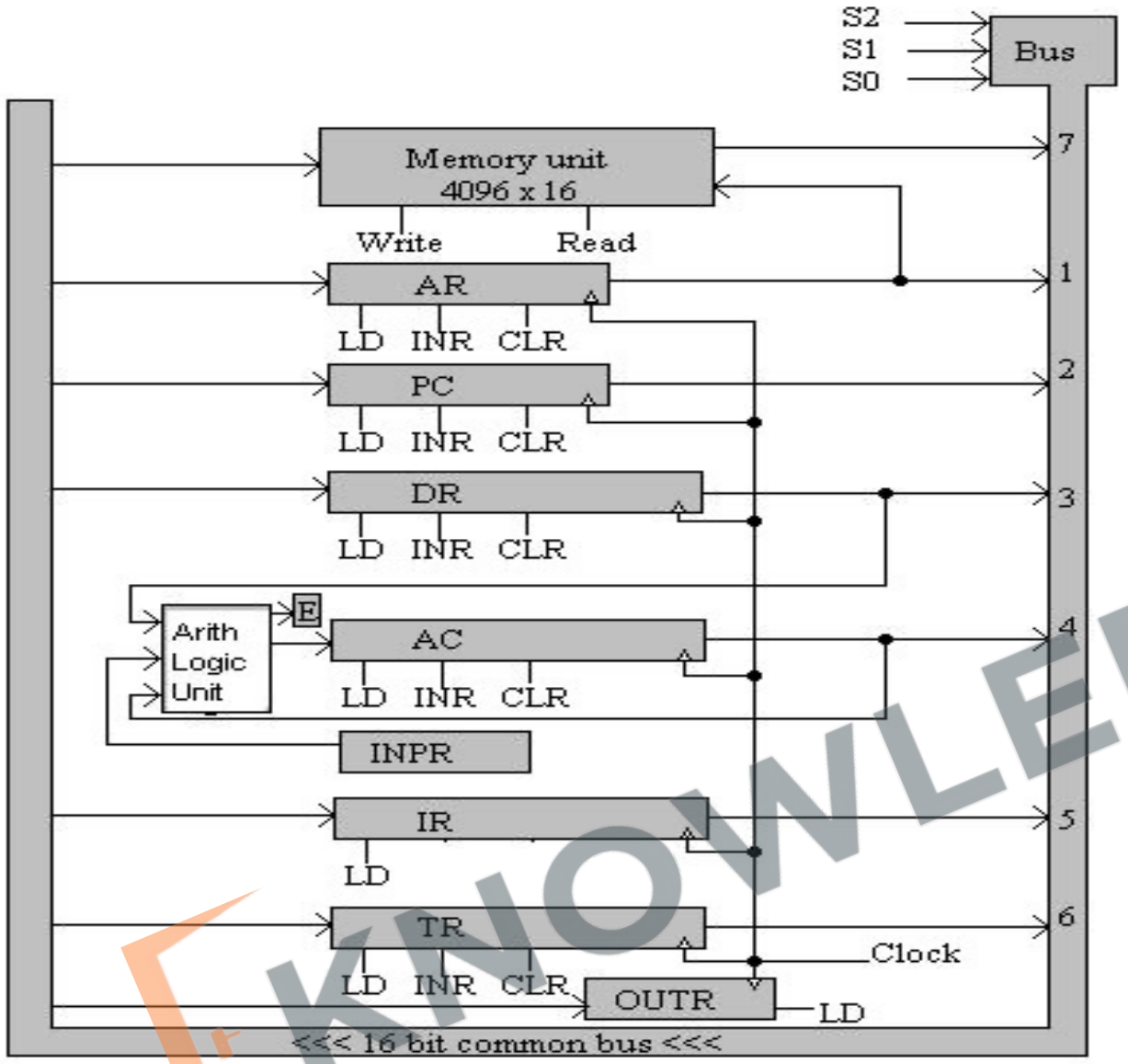




- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition.
- The memory receives the contents of the bus when its write input is activated.
- The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0 = 111$.



- The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus.
- INPR is connected to provide information to the bus but OUTR can only receive information from the bus. There is no transfer from OUTR to any of the other registers.



- Some additional points to notice
  - The 16 lines of the common bus receive information from six registers and the memory unit.
  - The bus lines are connected to the inputs of six registers and the memory.
  - Five registers have three control inputs: LD (load), INR (increment), and CLR (clear)

General registers organization

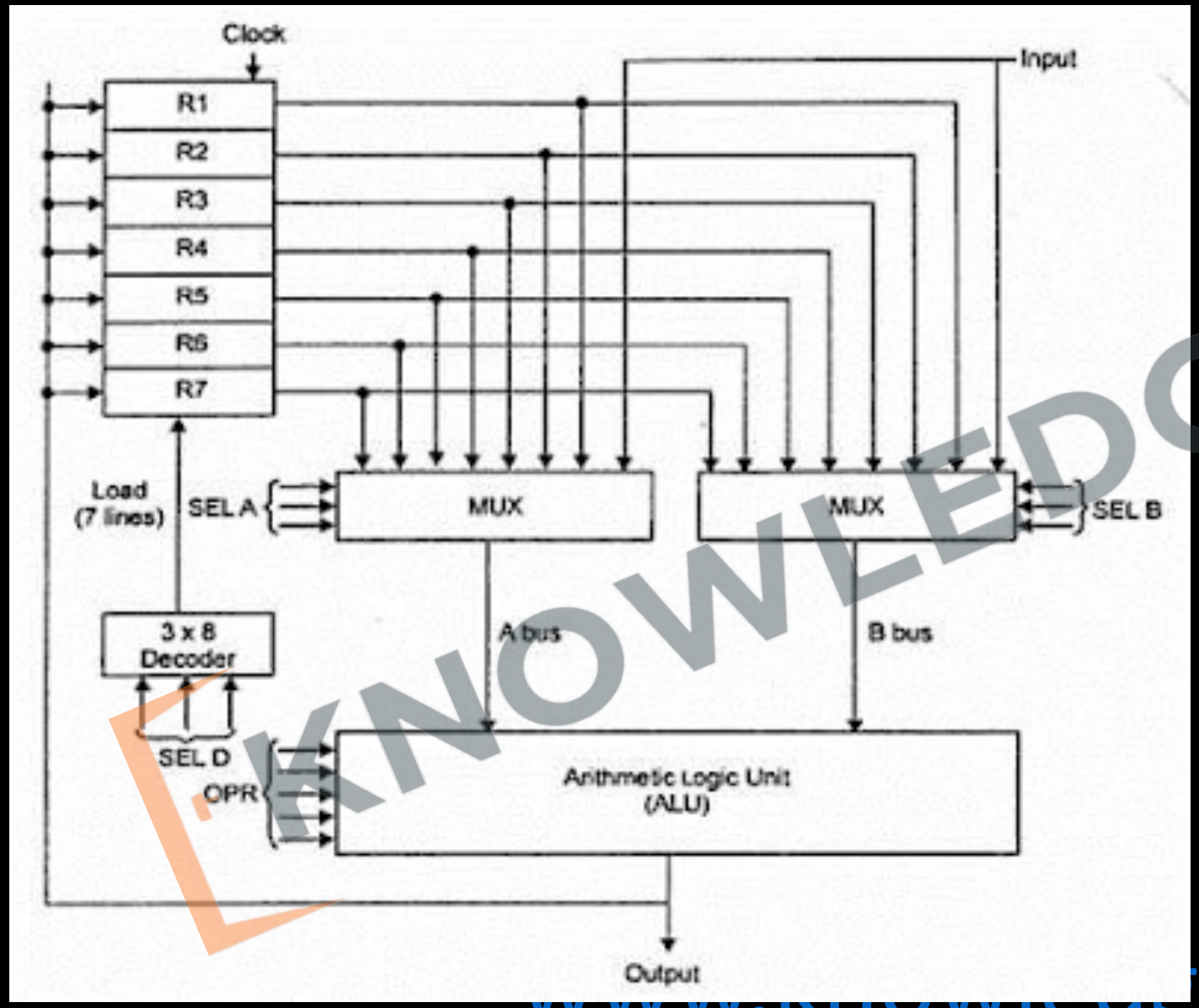


TABLE 8-1 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	*R1*	*R1*	*R1*
010	*R2*	*R2*	*R2*
011	*R3*	*R3*	*R3*
100	*R4*	*R4*	*R4*
101	*R5*	*R5*	*R5*
110	*R6*	*R6*	*R6*
111	*R7*	*R7*	*R7*

TABLE 8-2 Encoding of ALU Operations

OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add *A* + *B*	ADD
00101	Subtract *A* − *B*	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*	SHLA

General registers organization

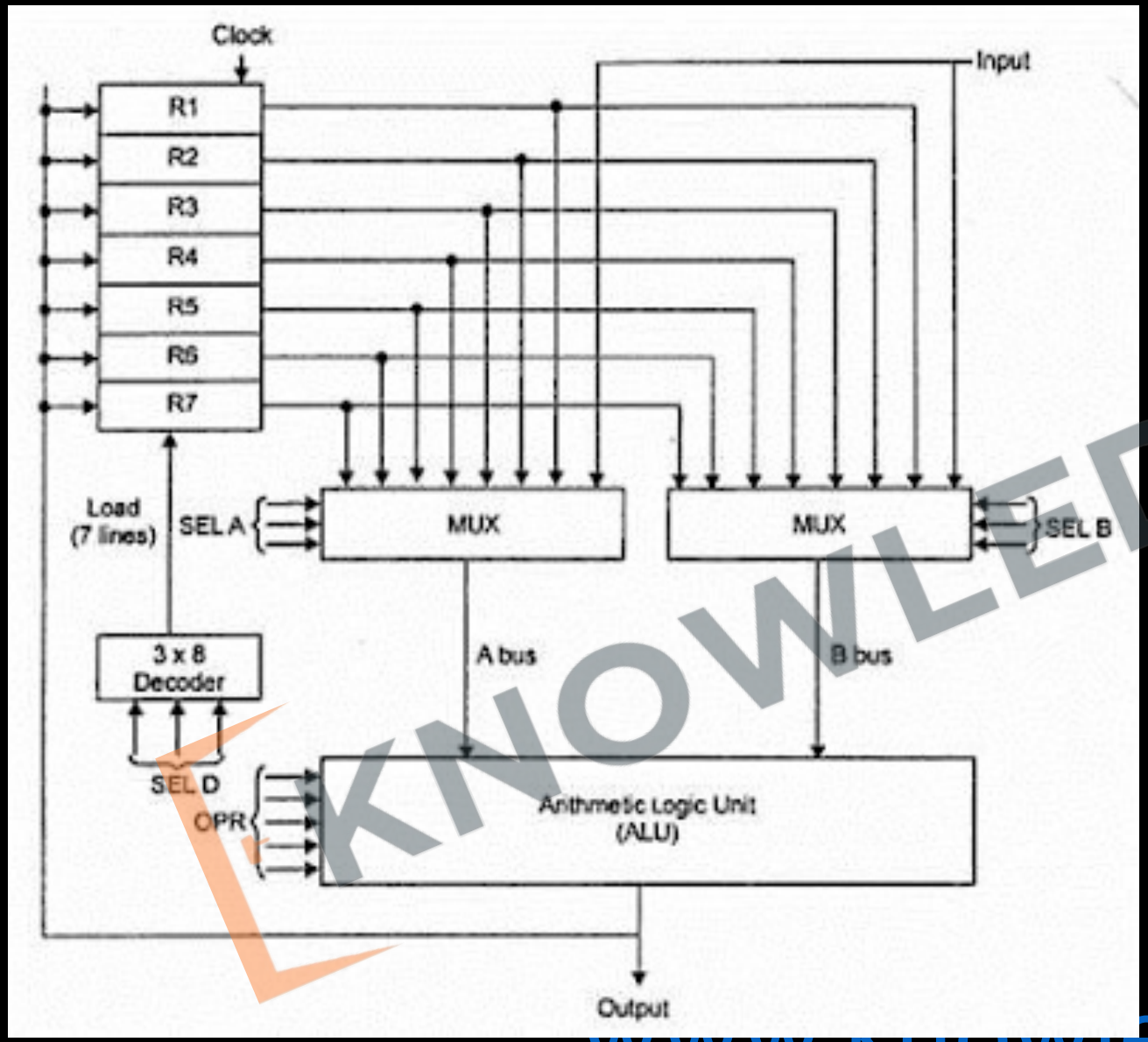
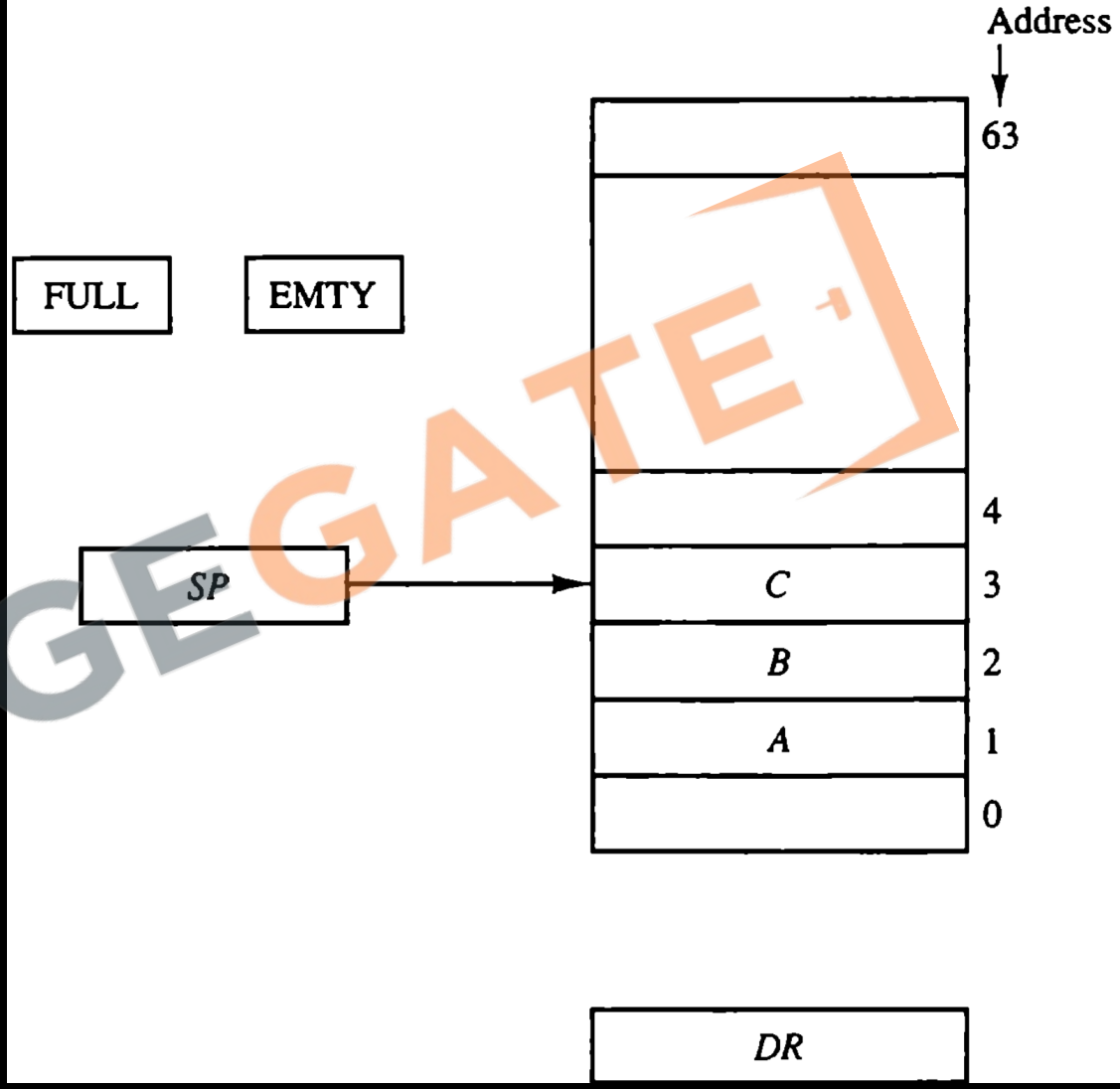


TABLE 8-3 Examples of Microoperations for the CPU

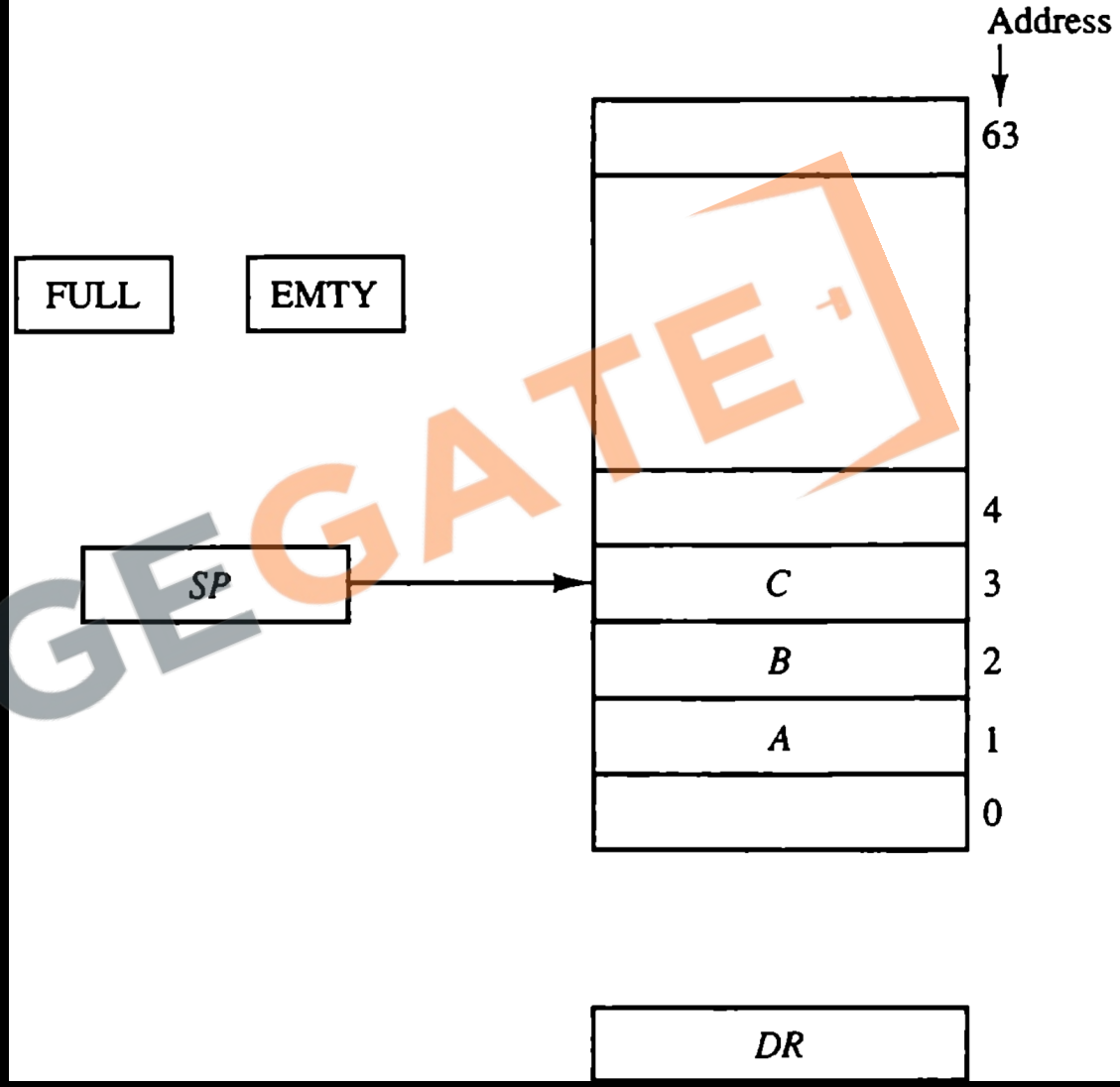
Microoperation	Symbolic Designation SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

Stack organization

- It is an ordered list in which addition of a new data item and deletion of already existing data item is done from only one end known as top of stack (TOS)
- The element which is added in last will be first to be removed and the element which is inserted first will be removed in last, so it is called last in first out (LIFO) or first in last out (FILO) type of list.
- Most frequently accessible element in the stack is the topmost element, whereas the least accessible element is the bottom of the stack.

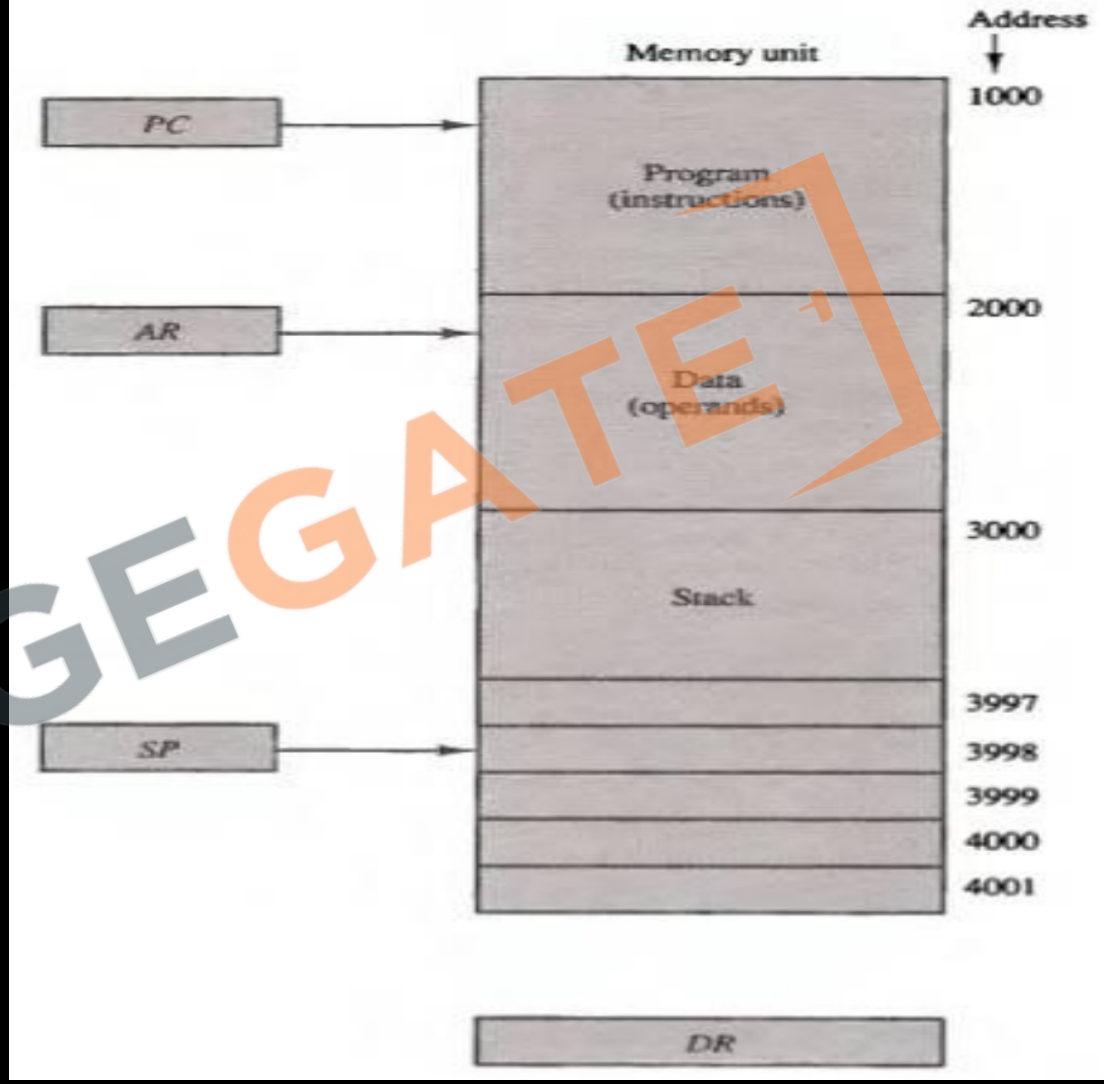


- Stack Pointer register (SP): It contains a value in binary each of 6 bits, which is the address of the top of the stack. Here, the stack pointer SP contains 6 bits and SP cannot contain a value greater than 111111 i.e. 63
- **Full Register**: it can store 1 bit of information, set to 1 when stack is full.
- **Empty Register**: it can store 1 bit of information, set to 1 when the stack is empty.
- **Data Register**: It hold the data to be written into into be read from the stack.



Memory Stack

- Memory stacks operate on a Last-In, First-Out (LIFO) principle, where the most recently added element is the first to be removed.
- A special register called the Stack Pointer (SP) points to the top of the stack, and it can contain binary values up to a limit, often determined by the architecture, such as 6 bits equating to 63 in decimal.
- To monitor the stack's status, Full and Empty Registers are used, each storing a single bit to indicate whether the stack is full or empty.
- A Data Register holds the data to be written into or read from the stack, serving as an intermediary between the CPU and the stack. These elements collectively form the essential organization of a memory stack.



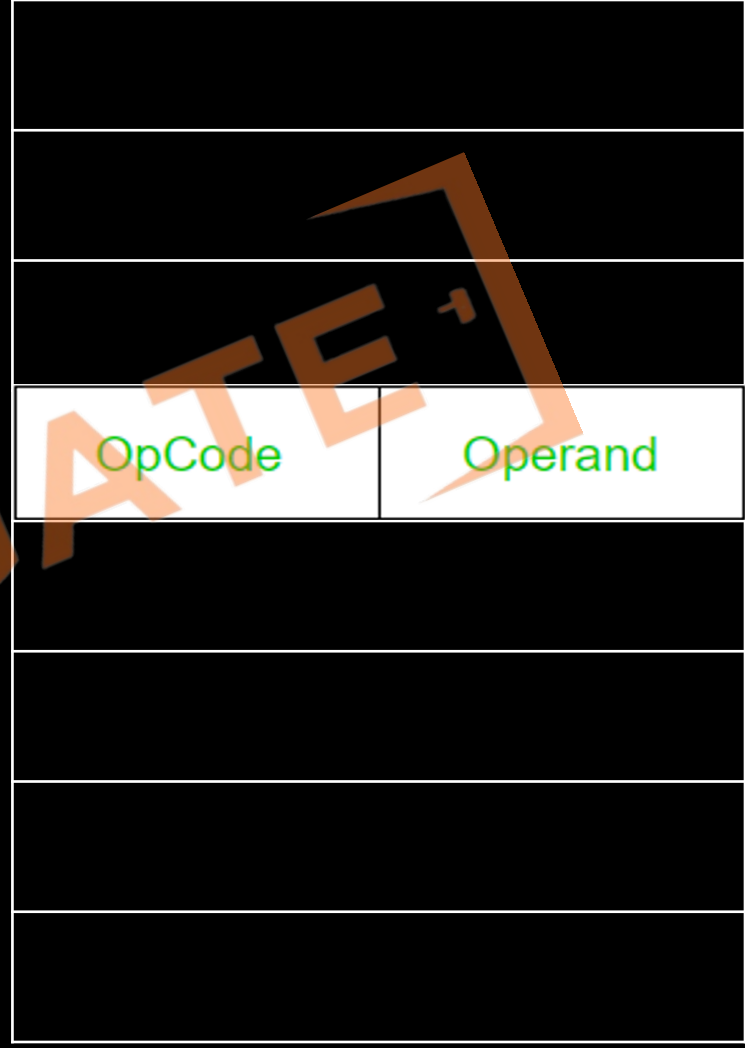
Addressing Mode

- It specifies the different ways possible in which reference to the operand can be made.
- **Effective address**: - It is the final address of the location where the operand is stored.
- Calculation of the effective address can be done in two ways
  - Non-computable addressing
  - Computable addressing (which involve arithmetic's)

- Criteria for different addressing mode
  - It should be fast
  - The length of the instruction must be small
  - They should support pointers
  - They should support looping constructs, indexing of data structure
  - Program relocation

Immediate mode addressing

- It means the operand is itself part of the instruction.
  - E.g. ADD 3
  - Means Add 3 to the accumulator



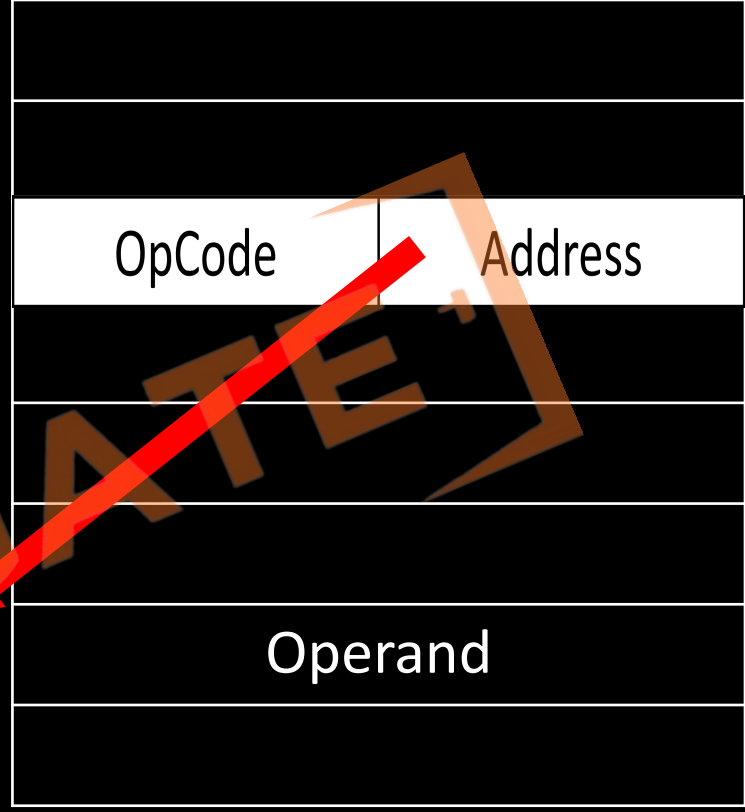
- Advantage
  - Can be used for constants, where values are already known.
  - Extremely fast, no memory reference is required.

- Disadvantage
  - Cannot be used with variables whose values are unknown at the time of program writing.
  - Cannot be used for large constant whose values cannot be stored in the small part of instruction.

- Application
  - Mostly used when required data is directly moved to required register or memory

Direct mode addressing (absolute address mode)

- It means instruction contain address of the memory location where data is present (effective address).
- Here only one memory reference operation is required to access the data.



- Advantage
  - With variable whose values are unknown direct addressing is the simplest one.
  - No restriction on the range of data values and largest value which system can hold, can be used in this mode.
  - Can be used to access global variables whose address is known at compile time.

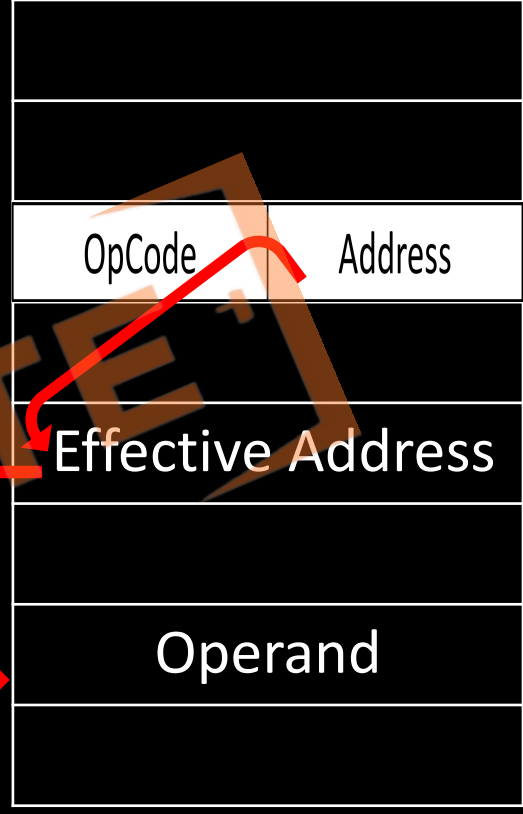
- Disadvantage
  - Relatively slow compare to immediate mode
  - No of variable used are limited
  - In large calculation it will fail

Indirect mode addressing

- Here in the instruction we store the address where the (address of the variable) effective address is stored using which we can access the actual data.

- Here two references are required.

- 1st reference to get effective address and 2nd reference to access the data.



- **Advantage**
  - No limitation on no of variable or size of variables.
  - Implementation of pointer are feasible and relatively more secure.

- **Disadvantage**
  - Relatively slow as memory must be referred more than one time.

Implied mode addressing

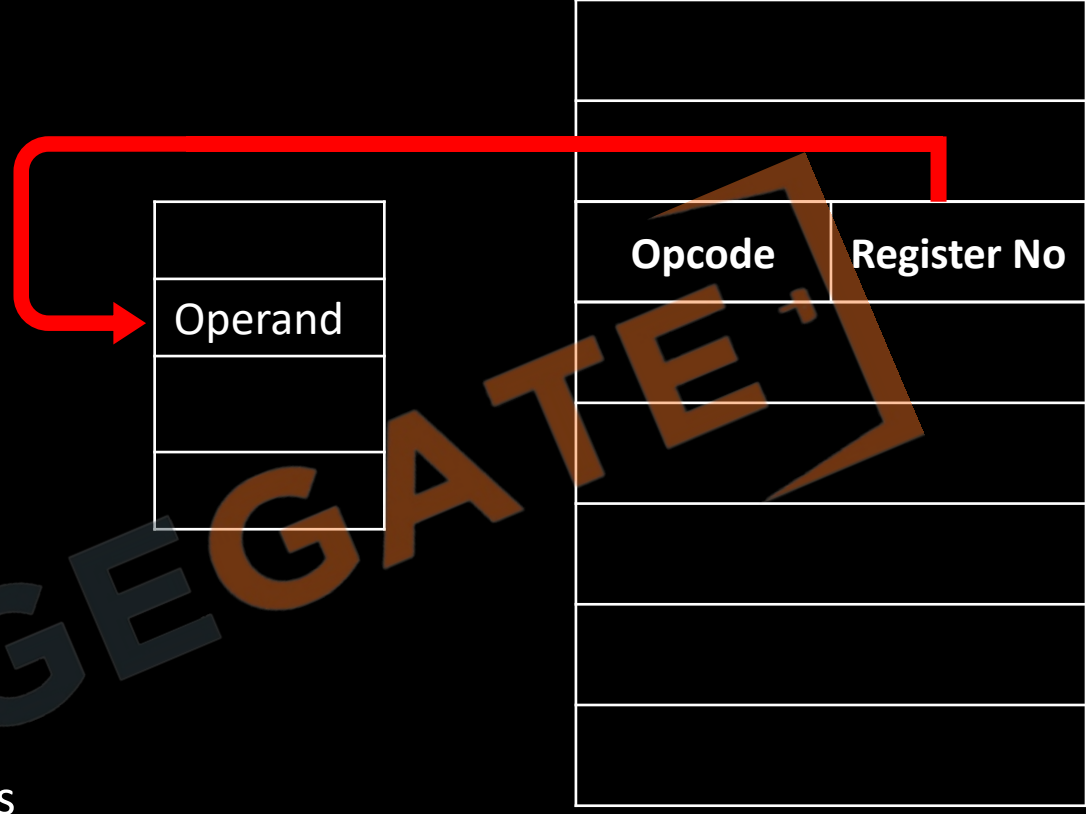
- In implied addressing mode, the operands are specified implicitly in the definition of the instruction.
- All the instructions which reference registers that use an accumulator are implied mode instructions.
- Zero address instructions in a stack organized computer are also implied mode instructions. Thus, it is also known as stack addressing mode.
- E.g. Increment accumulator, Complement accumulator

Register mode addressing

- It means variable are stored in register of the CPU instead of memory, in the instruction we will give the register no

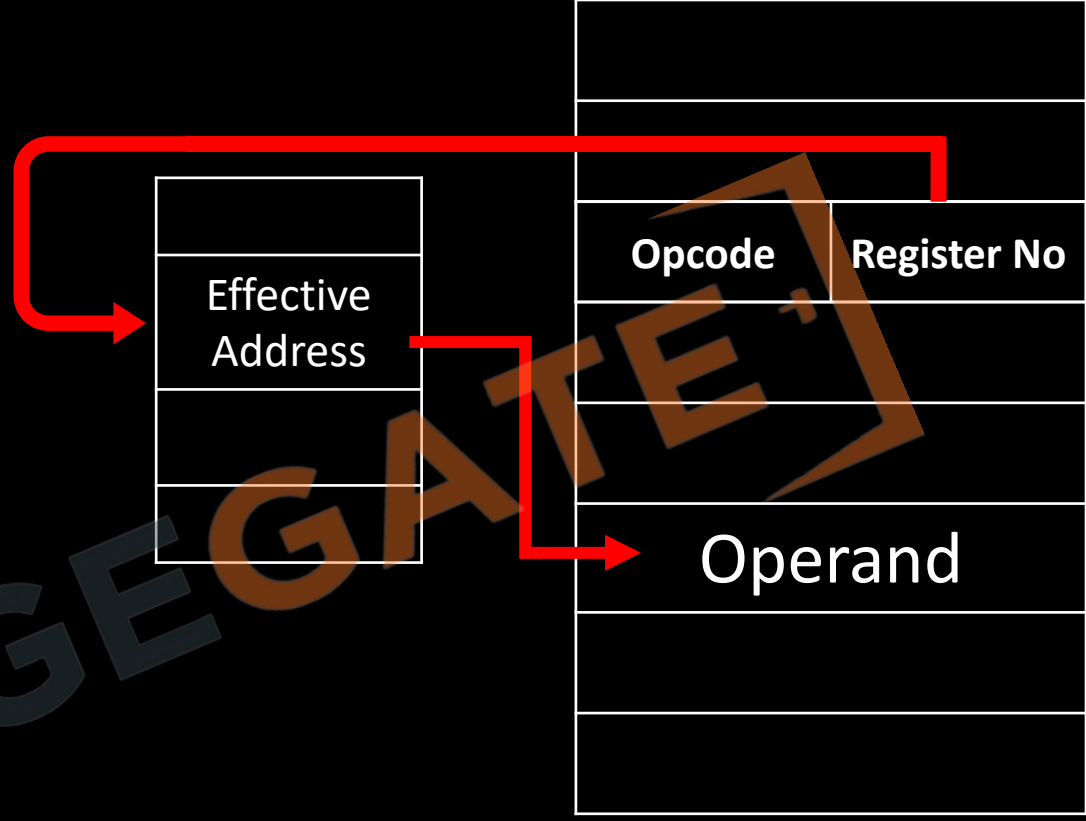
- Advantage
  - Will be extremely fast as register access time will be less then cache access time.
  - Because no of registers is less, so bits required to specify a register is also less.

- Disadvantage
  - Because no of registers is very less so only with few variables this method can be used



Register indirect mode addressing

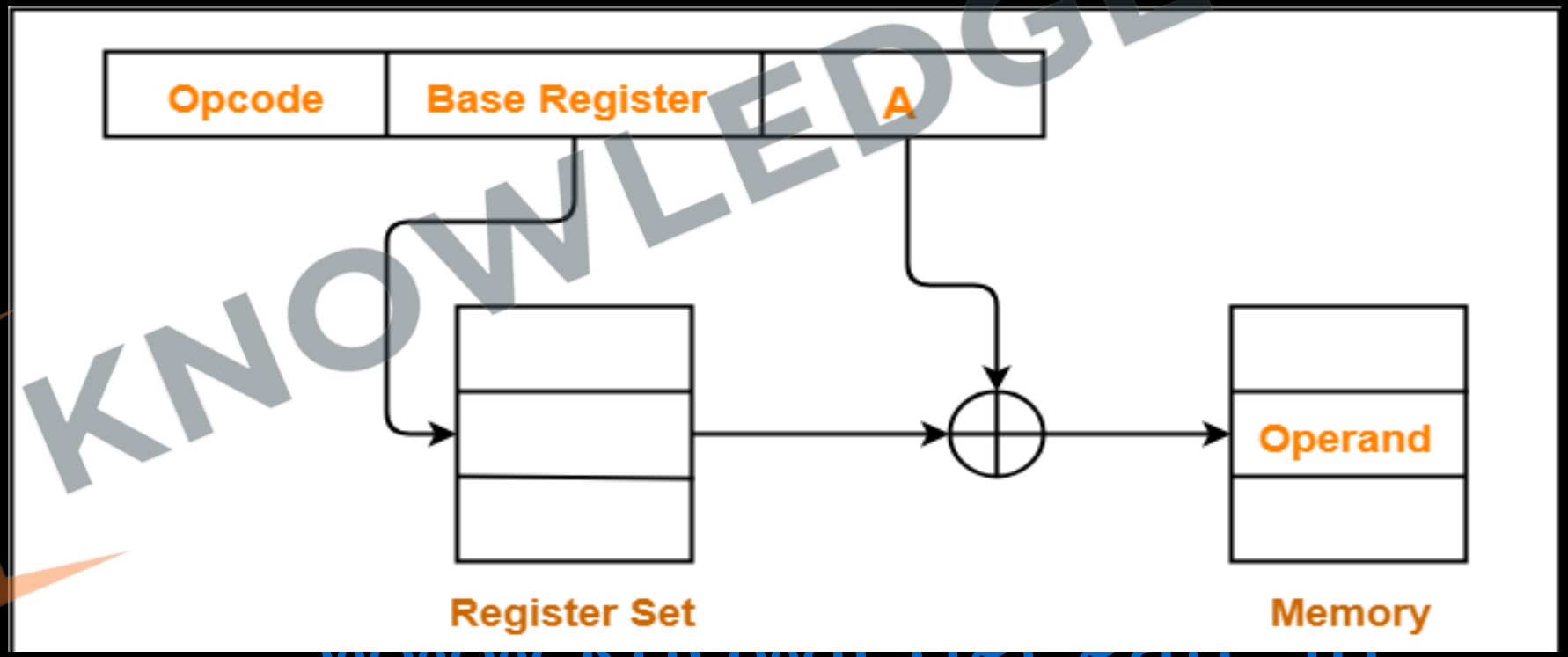
- In this mode in the instruction we will specify the address of the register where inside the register actual memory address of the variable is stored effective address.
- When same address is required again and again, this approach is very useful
- Here the same register can be used to provide different data by charging the value of register, i.e. it can reference memory without paying the price of having a full memory in the instruction.
- In pointer arithmetic, it provides more flexibility compare to register mode.
- Register indirect mode can further be improved by auto increment and auto decrement command where we read or work on some continuous data structure like array or matrix.



Base register (off set) mode

- In multiprogramming environment, the location of the process in the memory keeps on changing from one place to another.
- But if we are using direct address in the process then it will create a problem.
- So, to solve this problem we try to save the starting of the program address in a register called base register. It is a very popular approach in most of the computer.

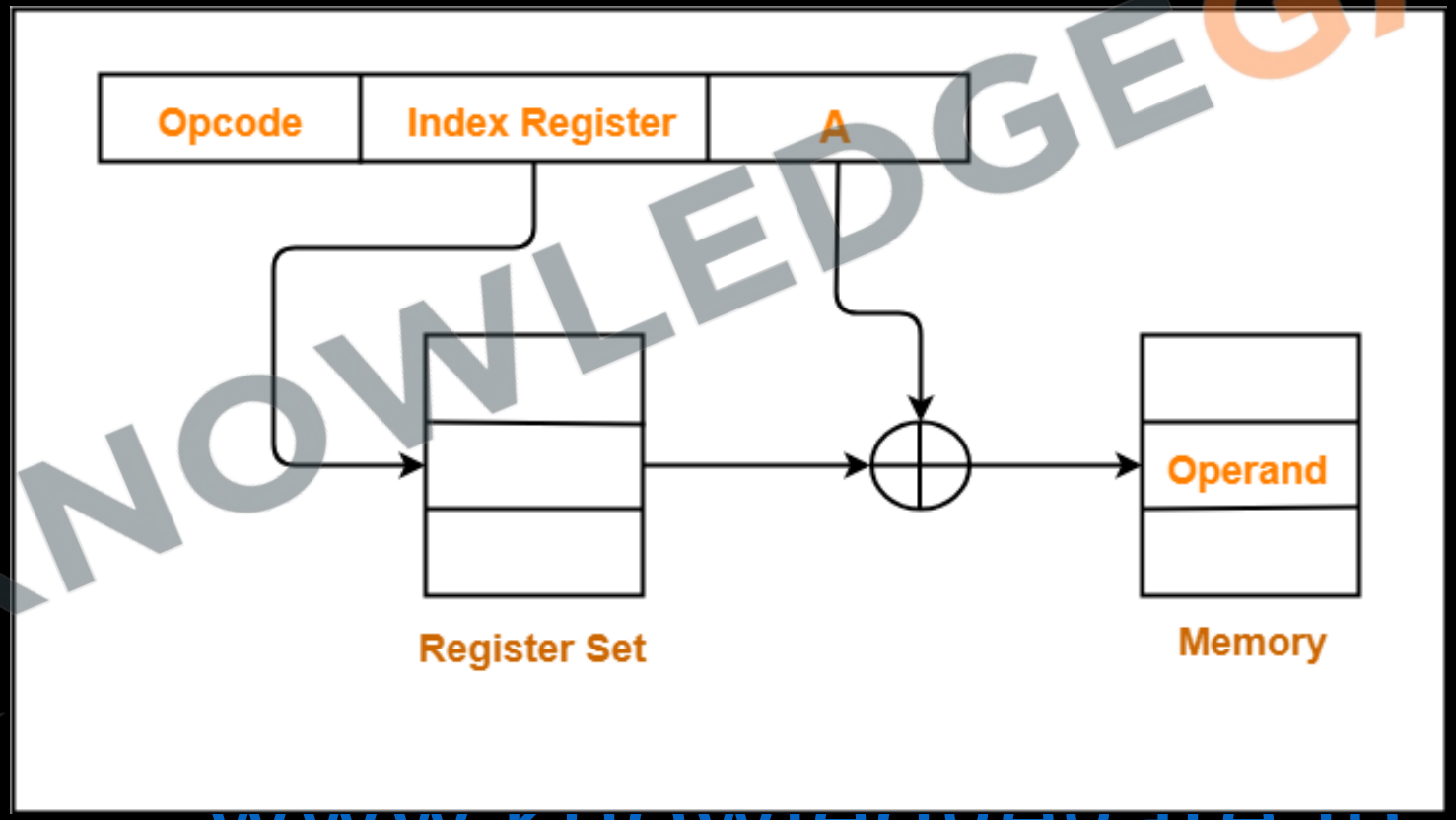
- Then instead to giving the direct branch address we give off set in the instruction
- Effective address = base address + off set (instruction)
- Now the advantage is even if we try to shift process in the memory, we only need to change the content of the base register.
- Final address will be the sum of what is given in instruction and given in base register.



Index addressing mode

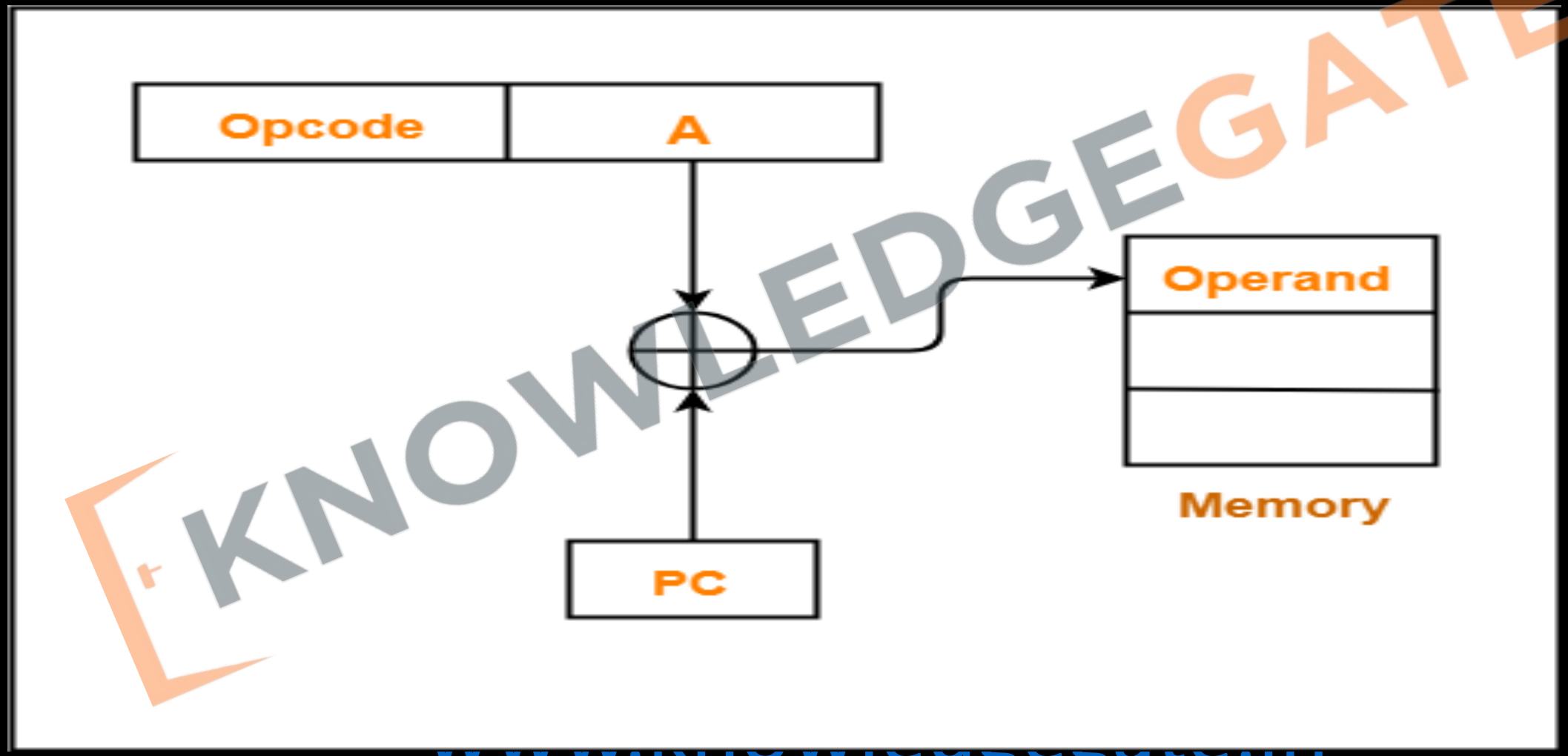
- This mode is generally used when CPU has a number of registers, and out of which one can be used an index register
- This mode is especially useful, when we try to access large sixed array elements.

- Idea is, give the base address of the array in the instruction and the index which we want to access will be there in the register.
- So by changing the index in the index register we can use the same instruction to access the different element in the array.
- Base in present inside the instruction and index is present inside the register



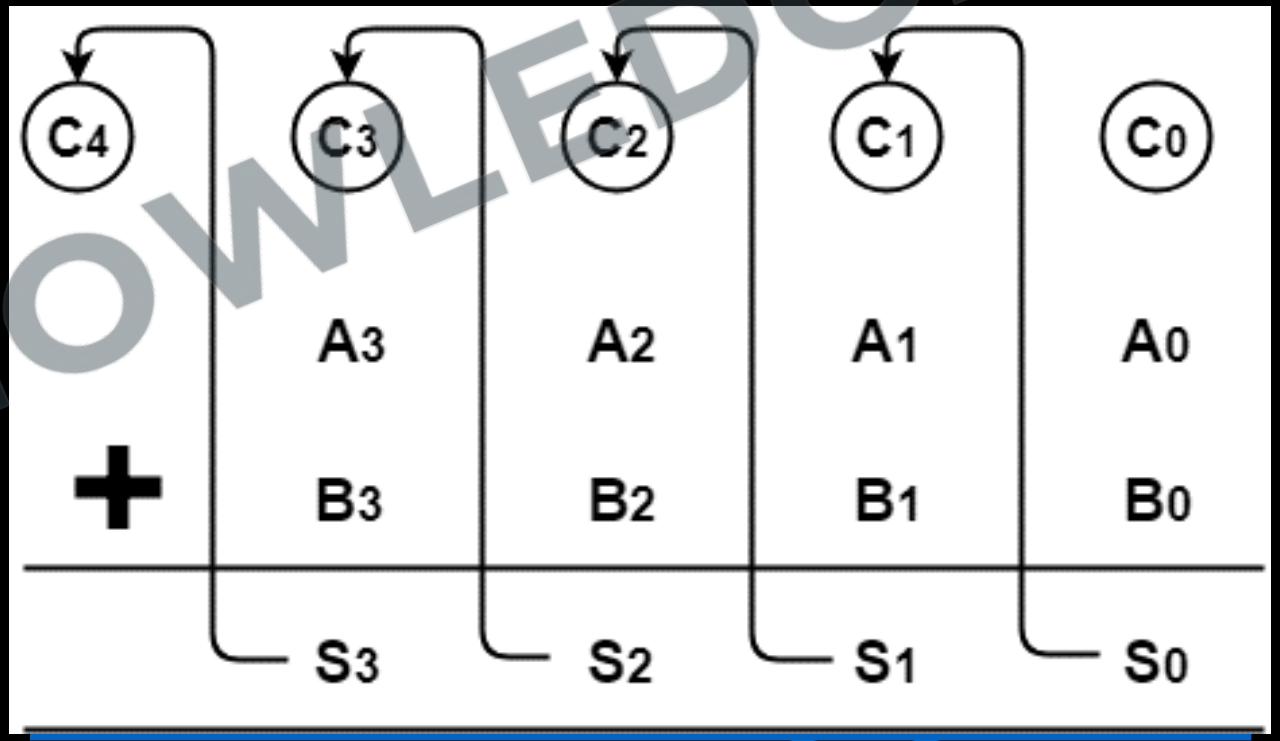
Relative Addressing Mode

- Effective address of the operand is obtained by adding the content of program counter with the address part of the instruction.

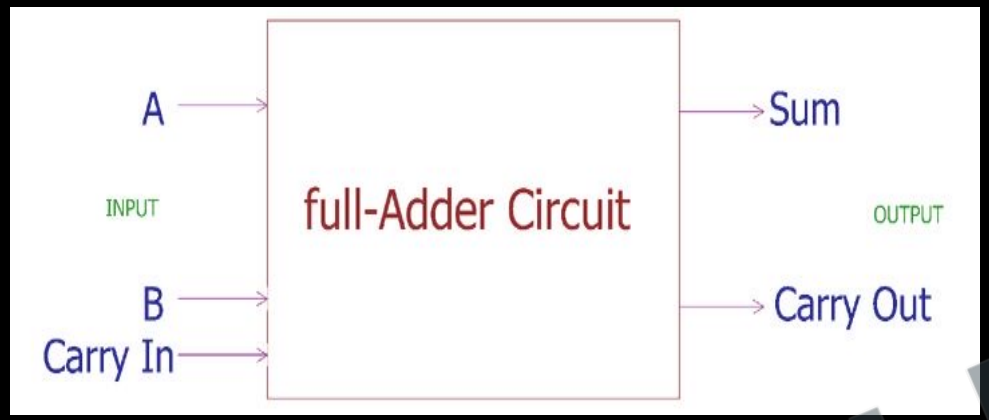


Full adder

1. A full adder is a combinational logic circuit that performs the arithmetic sum of three input bits.
2. Where A_n, B_n are the n^{th} order bits of the number A and B respectively and C_n is the carry generated from the addition of $(n-1)^{th}$ order bits.
3. It consists of three input bits, denoted by A (First operand), B (Second operand), C_{in} (Represents carry from the previous lower significant position).



- Two output bits are same as of half adder, which is Sum and $Carry_{out}$.
- When the augend and addend number contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.

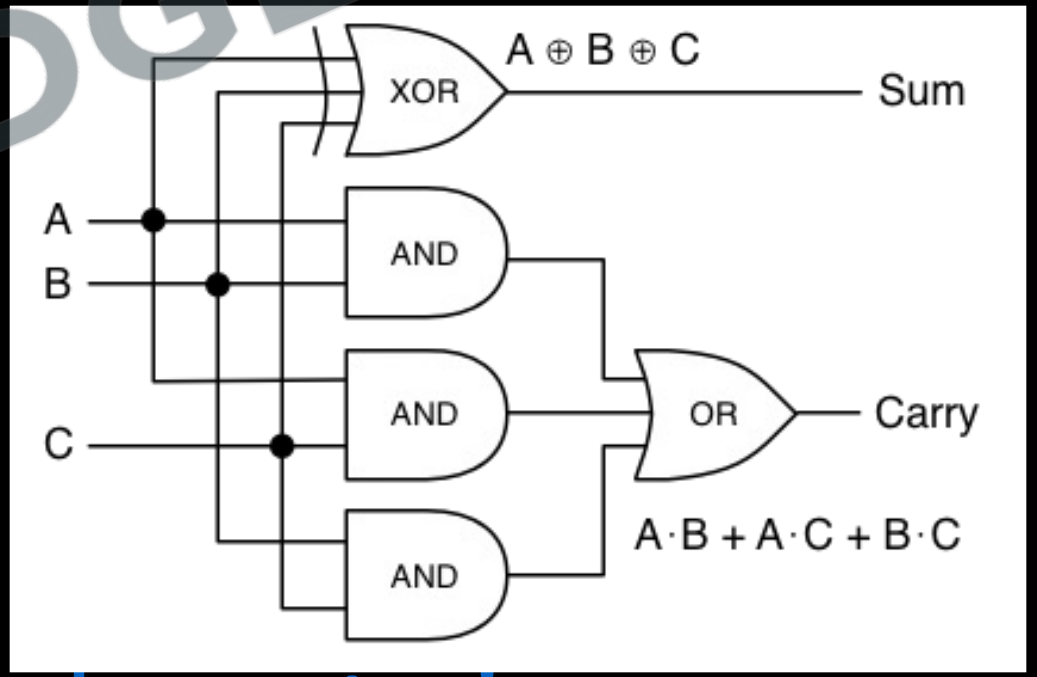


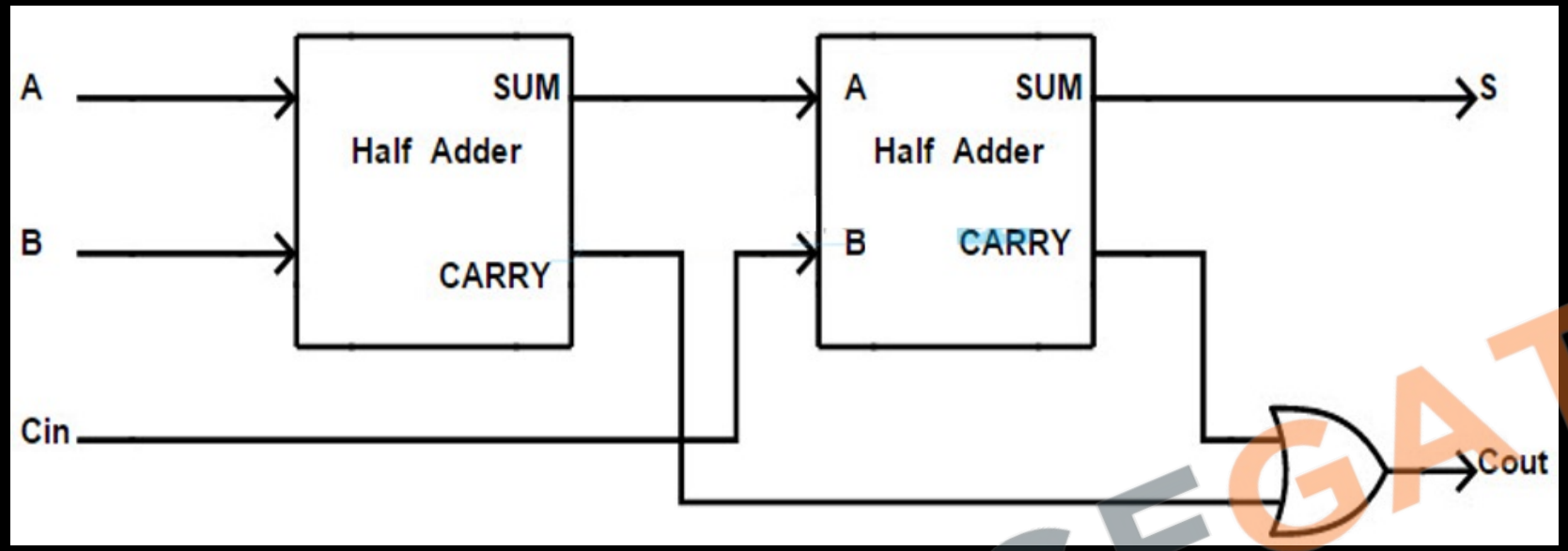
INPUTS			OUTPUTS	
A	B	C_{in}	C_{out}	Sum
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

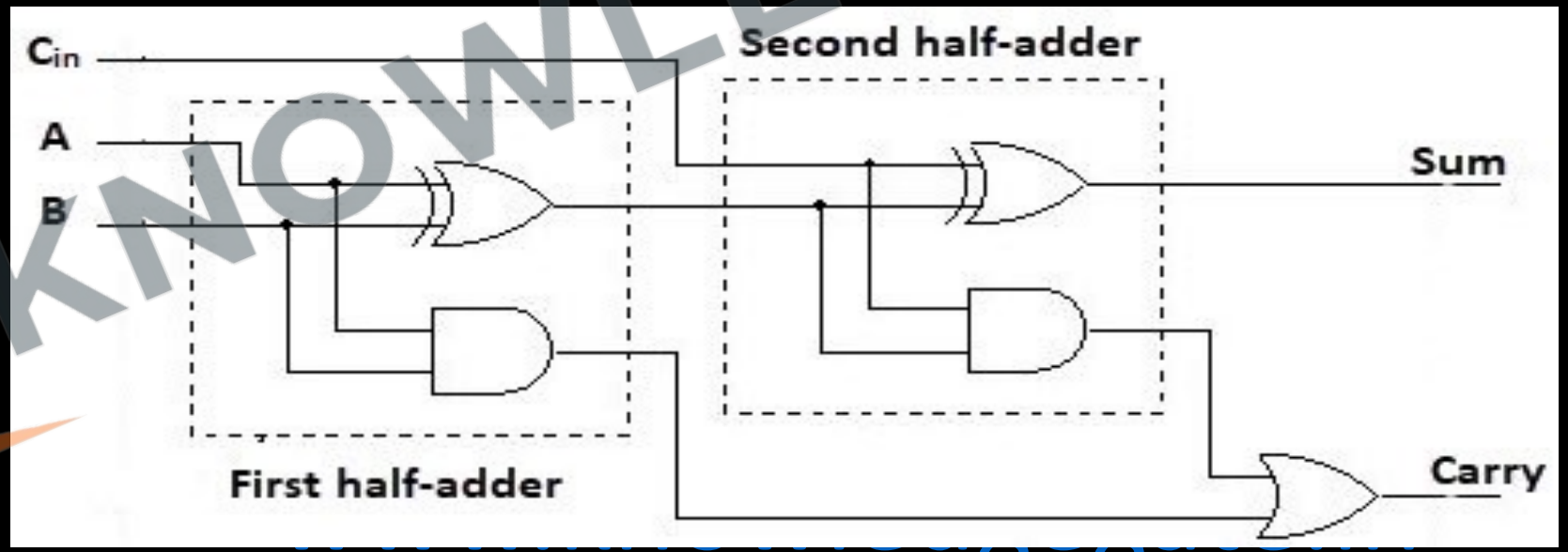
	ab	a’b’	a’b	ab	ab’
C_{in}		00	01	11	10
C_{in}’	0	0	2	6	4
C_{in}	1	1	3	7	5

	ab	a’b’	a’b	ab	ab’
C_{in}		00	01	11	10
C_{in}’	0	0	2	6	4
C_{in}	1	1	3	7	5

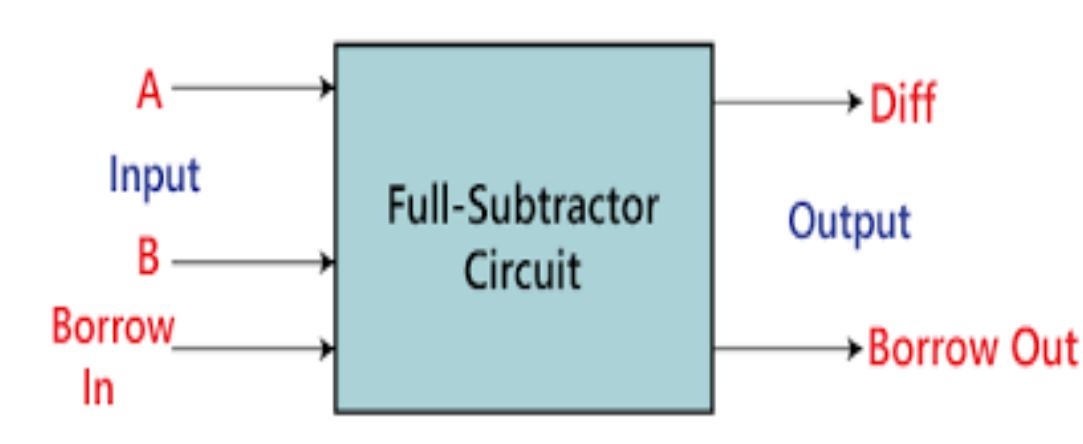
INPUTS			OUTPUTS	
A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

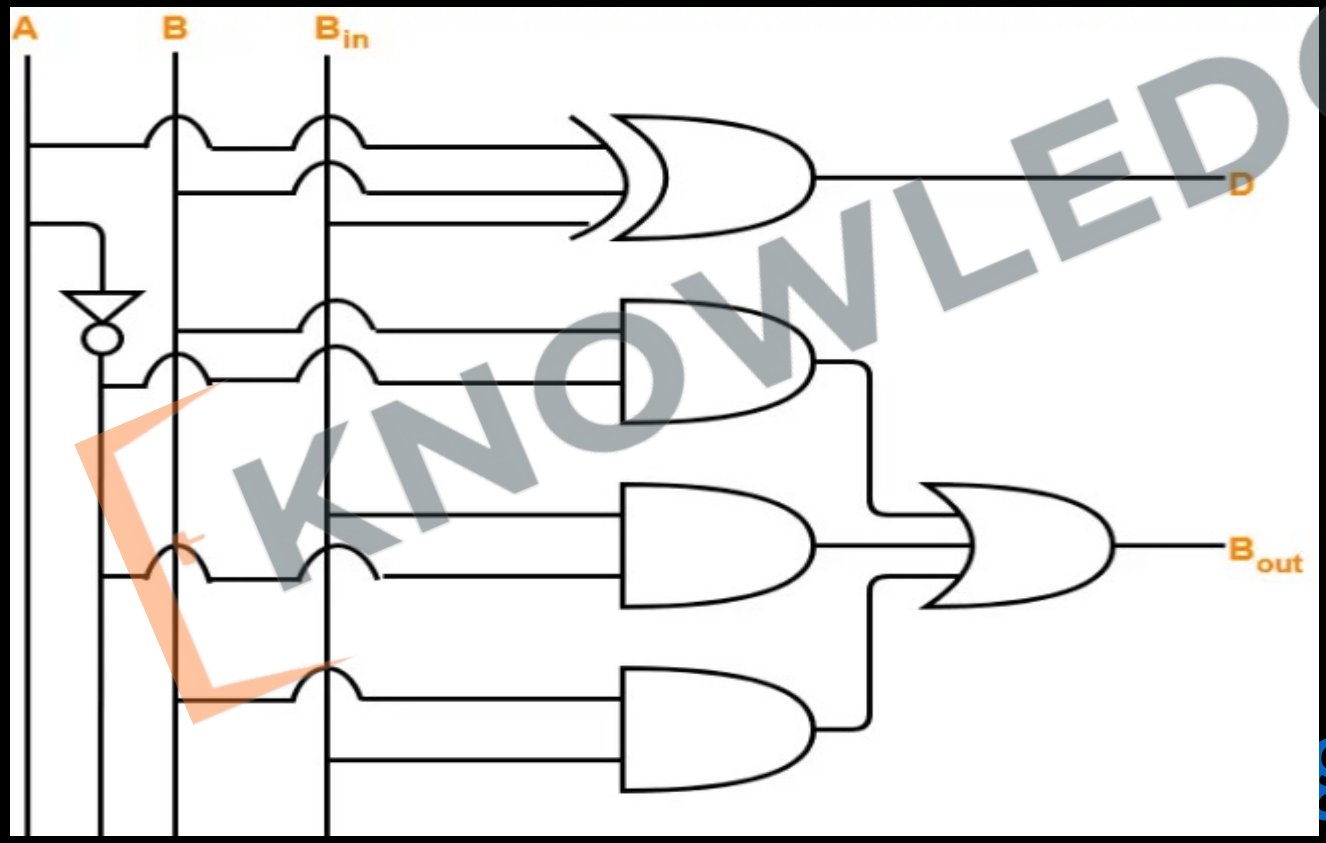




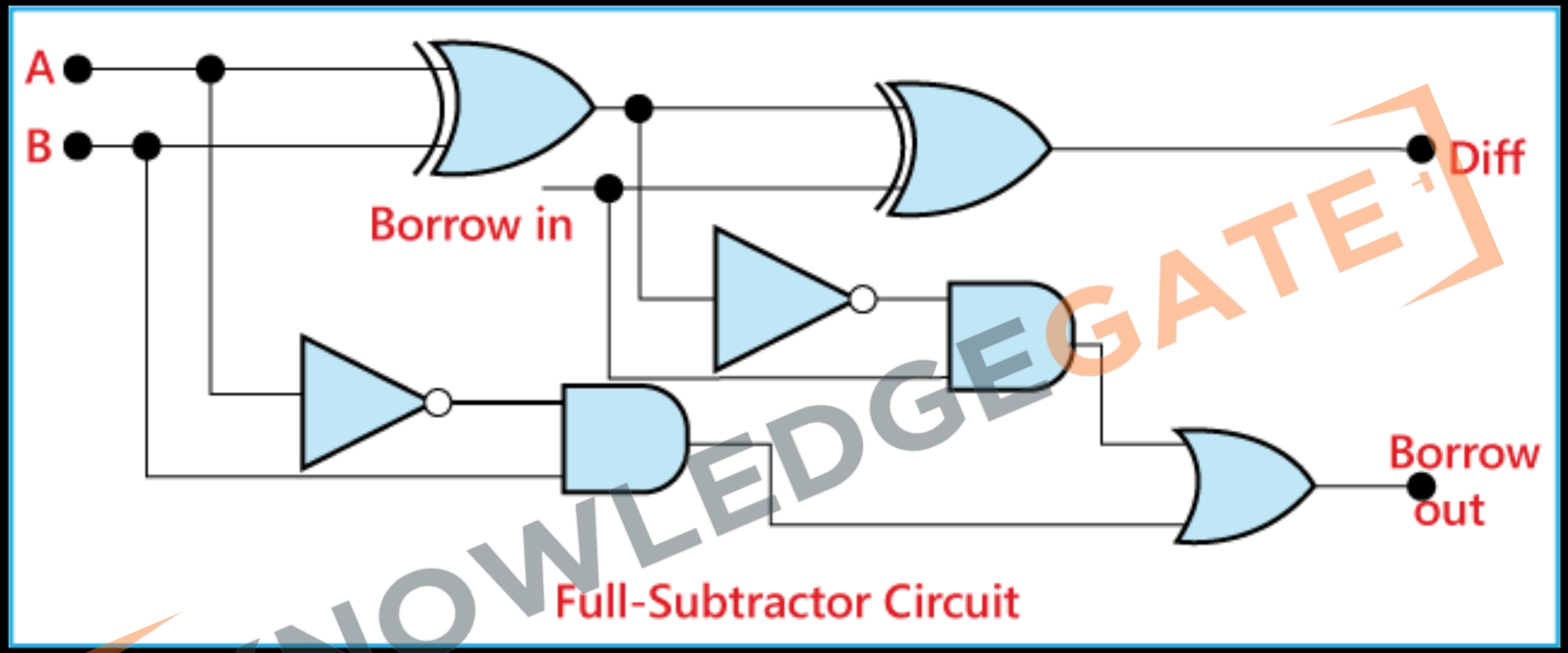


Full subtractor



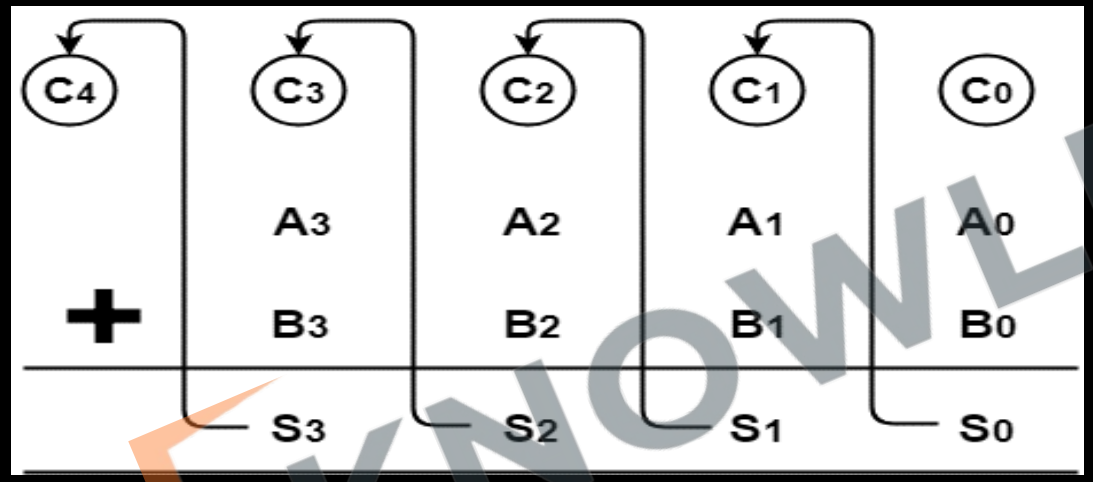


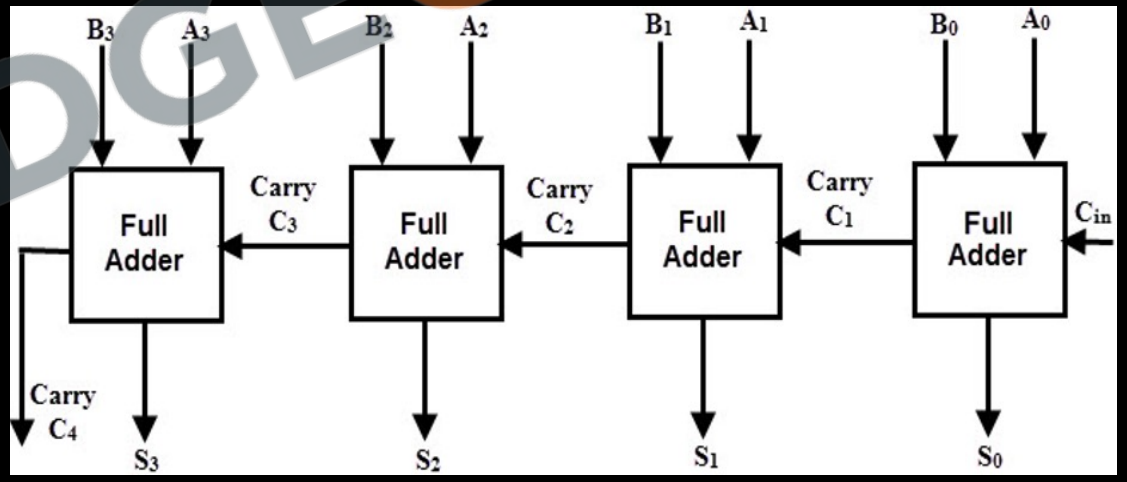
Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



Four-bit parallel binary adder / Ripple adder

- As we know that full adder is capable of adding two 1 bit number and 1 previous carry, but in order to add binary numbers with more than one bits, additional full adders must be employed. For e.g. a four bit binary adder can be constructed using four full adders.
- Theses four full adders are connected in cascade, carry output of each adder is connected to the carry input of the next higher-order adder. So a n-bit parallel adder is constructed using ‘n’ number of full adders.





- There are some scope of improvement in parallel binary adder / Ripple adder is it is very slow

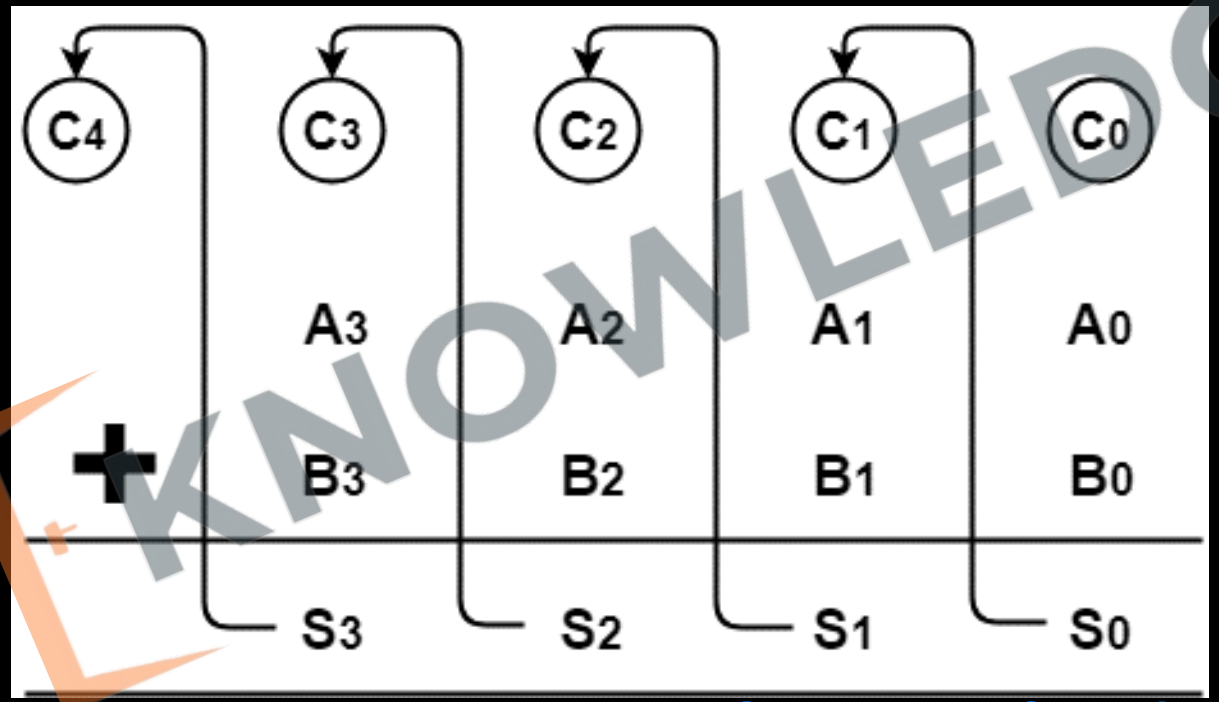
Carry propagation delay

Look ahead Carry Generator

Look ahead carry adder

- The carry propagation time is an important attribute of the adder because it limits the speed with which two numbers are added. The solution to delay is to increase the complexity of the equipment in such a way that the carry delay time is reduced.

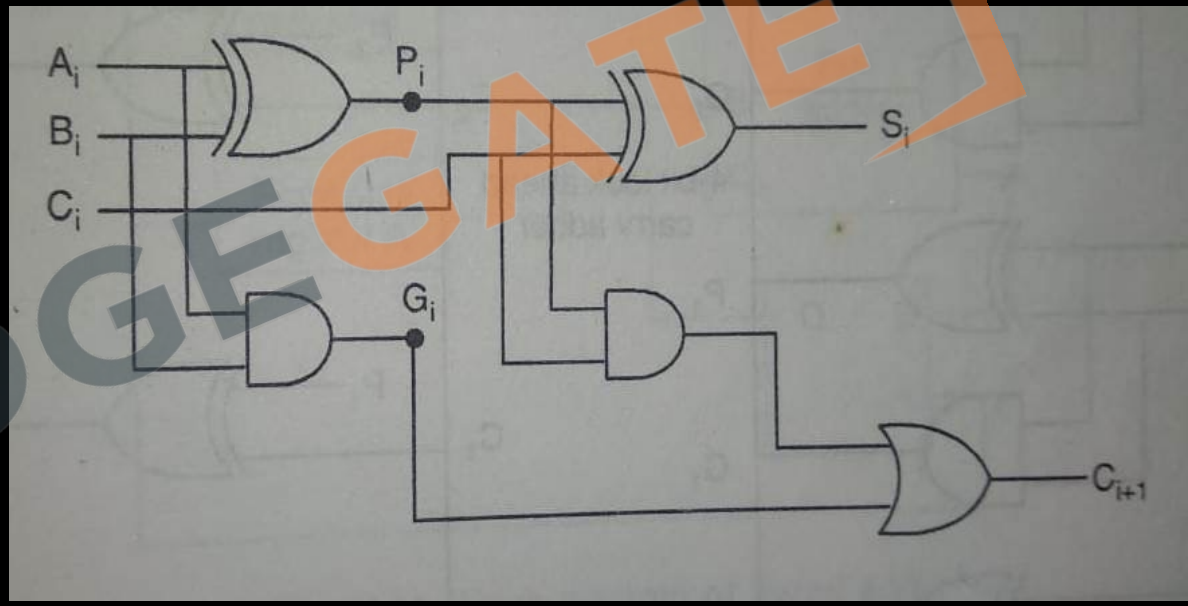
- To solve this problem most widely used technique employs the principle of 'look ahead carry'. This method utilizes logic gates to look at the lower order bits of the augend and addend to see if a higher order carry is to be generated. It uses two functions carry generate G_i and carry propagate P_i



$A(A_3\ A_2\ A_1\ A_0)$

$B(B_3\ B_2\ B_1\ B_0)$

- G_i is called a ***carry generate***, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i.
- P_i is called a ***carry propagate***, because it determines whether a carry into stage i will propagate into stage i + 1.
- We now write the Boolean functions for the carry outputs of each stage and substitute the value of each C_i from the previous equations:

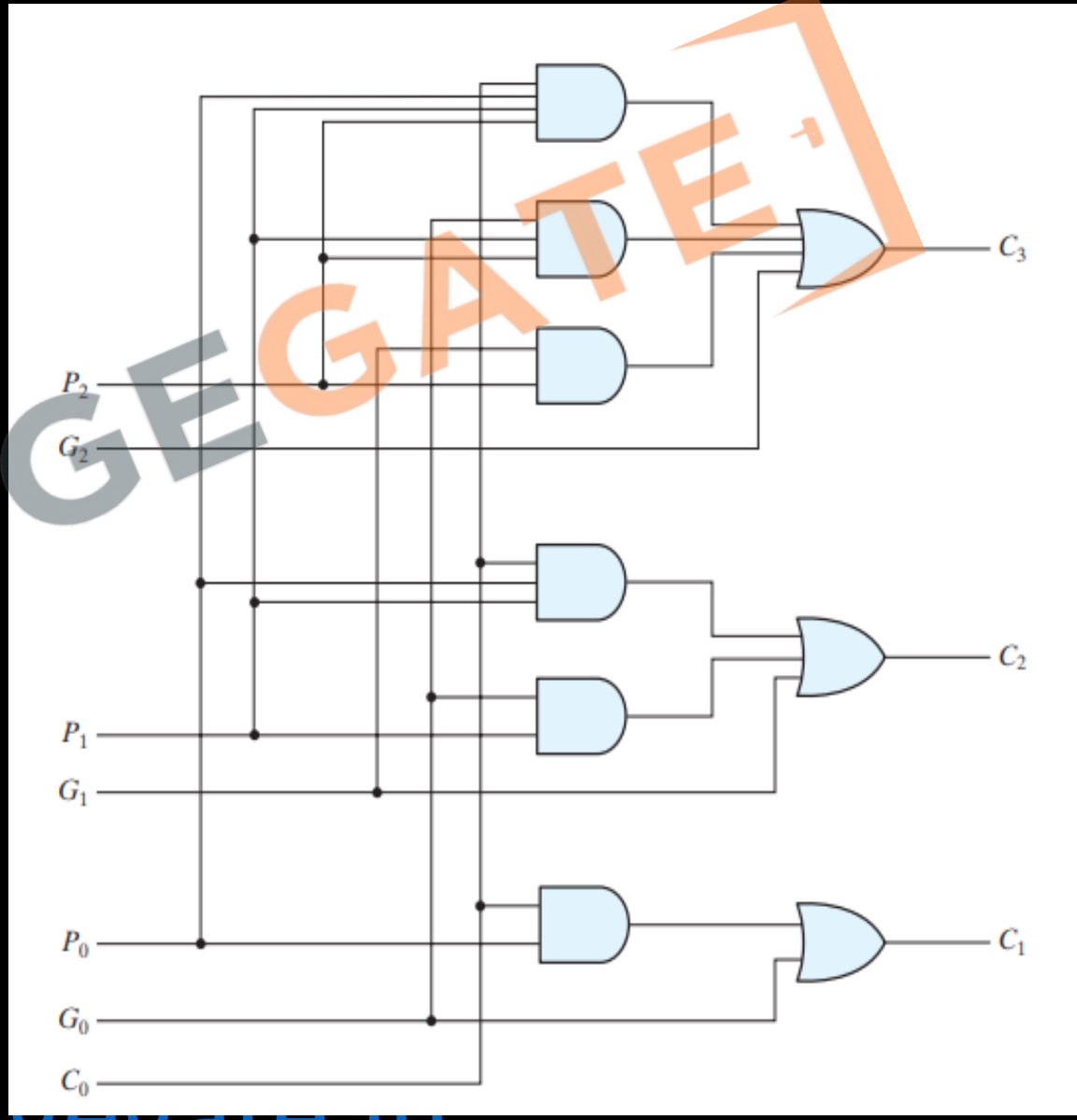


- $P_i = A_i \oplus B_i$
  $G_i = A_i \cdot B_i$
  $S_i = P_i \oplus C_i$
  $C_{i+1} = G_i + P_i C_i$

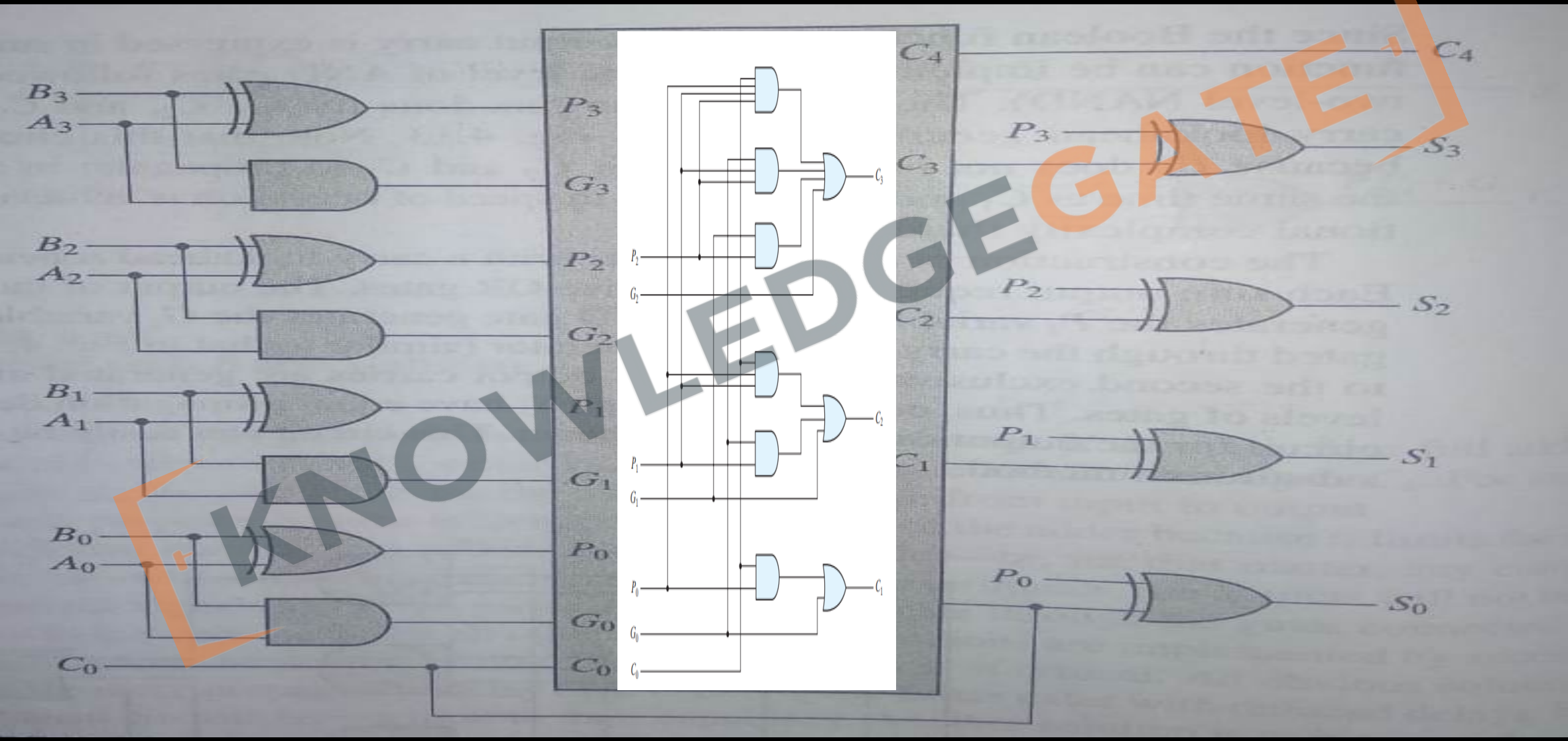
- $C_0 = 0$
- $C_1 = G_0 + P_0 C_0$
- $C_2 = G_1 + P_1G_0 + P_1P_0 C_0$
- $C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1 P_0 C_0$
- $C_4 = G_3 + G_2.P_3 + G_1.P_2.P_3 + G_0.P_1.P_2.P_3 + C_0.P_0.P_1.P_2.P_3$

- Since the Boolean function for each output carry is expressed in sum-of-products form. Each function can be implemented with one level of AND gates followed by an OR gate (or by a two-level NAND).

- $C_0 = 0$
- $C_1 = G_0 + P_0 C_0$
- $C_2 = G_1 + P_1G_0 + P_1P_0 C_0$
- $C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1 P_0 C_0$
- $C_4 = G_3 + G_2.P_3 + G_1.P_2.P_3 + G_0.P_1.P_2.P_3 + C_0.P_0.P_1.P_2.P_3$

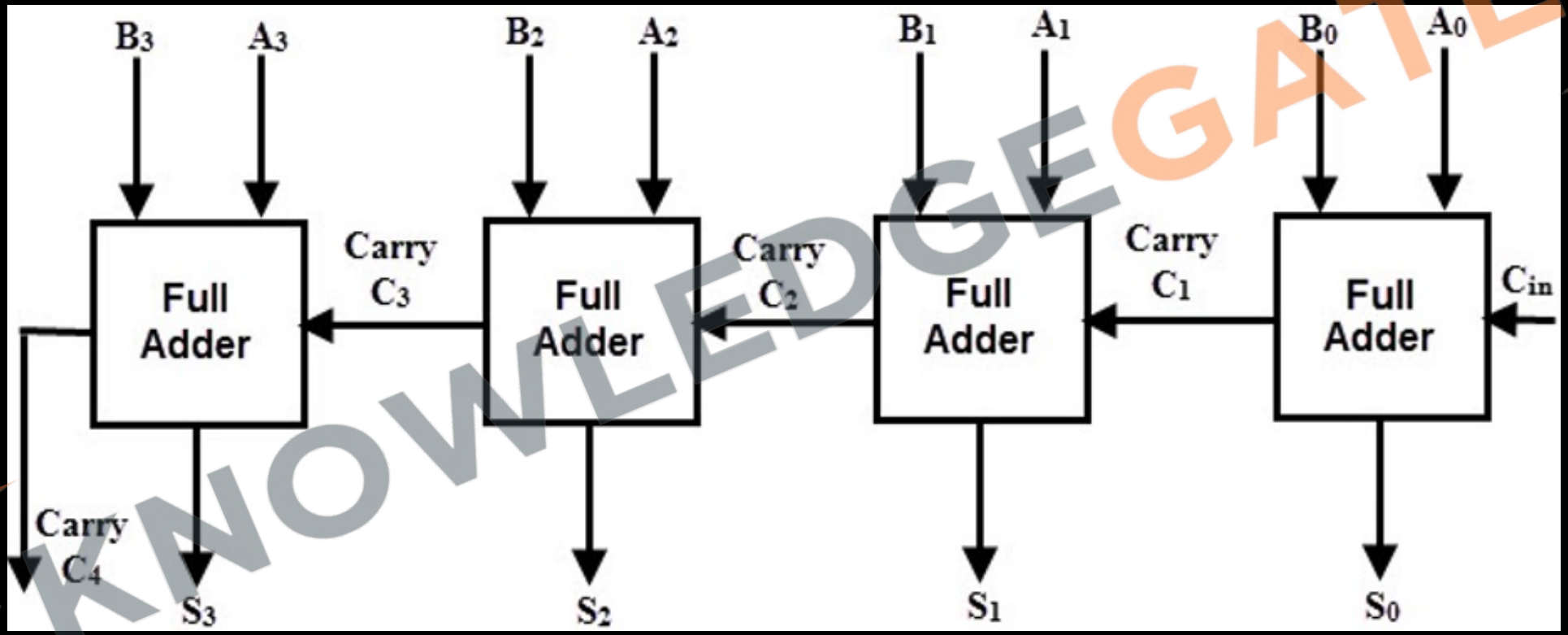


- All output carries are generated after a delay through two levels of gates. Thus, outputs S_1 through S_3 have equal propagation delay times.

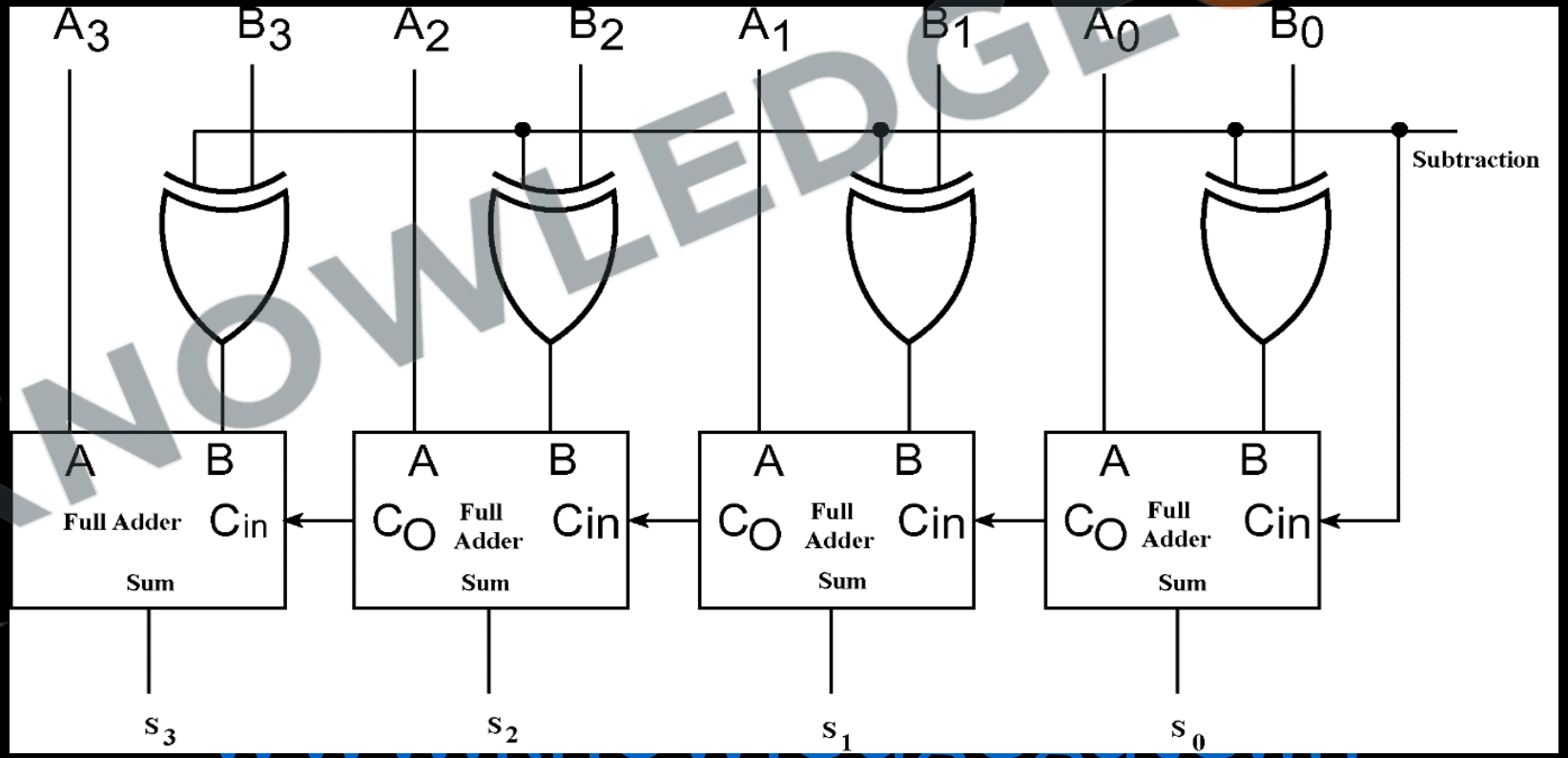


Four-bit ripple adder/subtractor

- The subtraction A - B can be done by taking the 2's complement of B and adding it to A.
- A + (-B)



- The circuit for subtracting A - B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The mode input M controls the operation. When M = 0, the circuit is an adder, and when M = 1, the circuit becomes a subtractor.
- When M = 0, we have B $\oplus$ 0 = B. The full adders receive the value of B, the input carry is 0, and the circuit performs A plus B.
- When M = 1, we have B $\oplus$ 1 = B' and C_0 = 1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B.



Booth’s algorithm

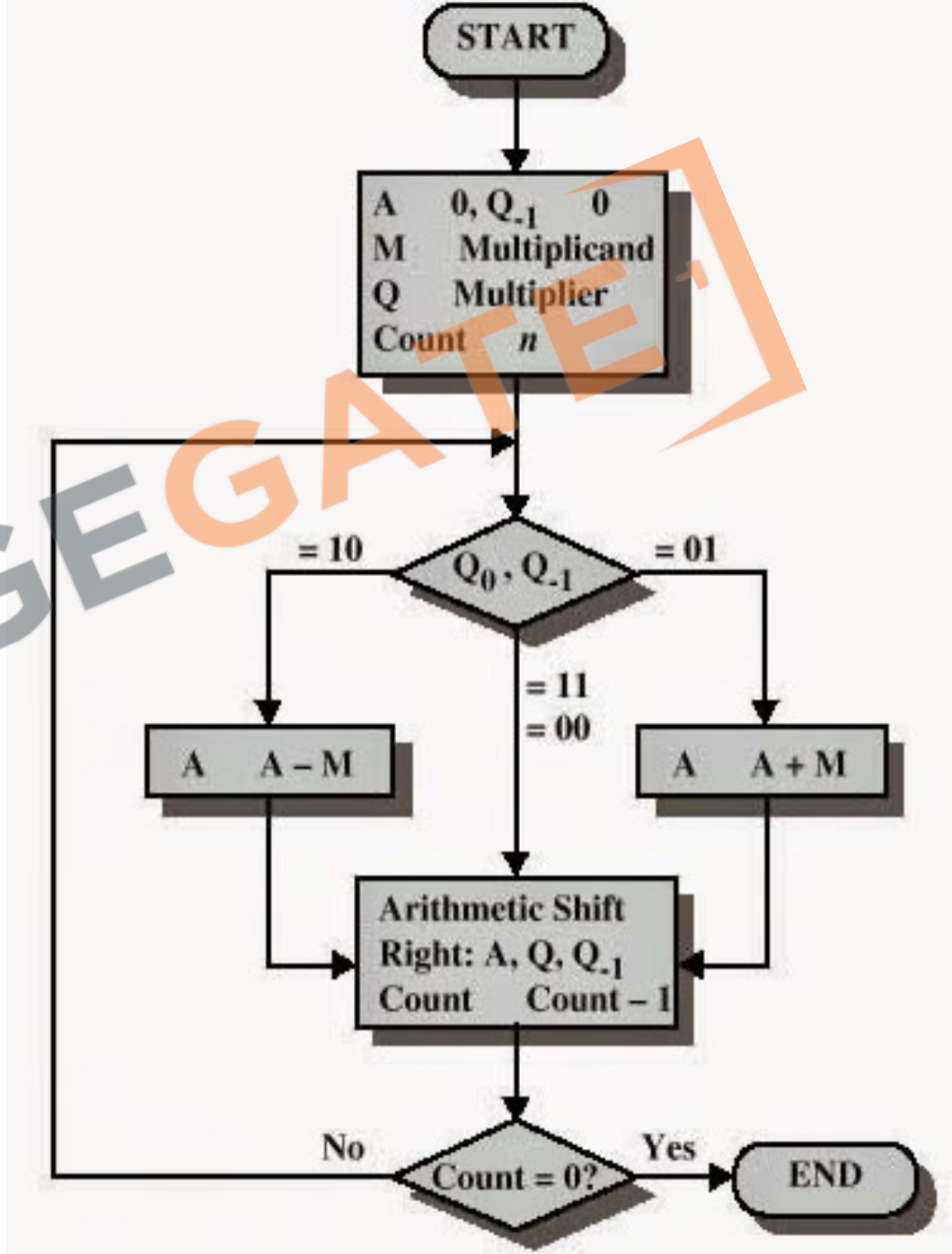
- **Andrew Donald Booth** (11 February 1918 – 29 November 2009) was a British electrical engineer, physicist and computer scientist, who was an early developer of the magnetic drum memory for computers. He is known for Booth's multiplication algorithm.

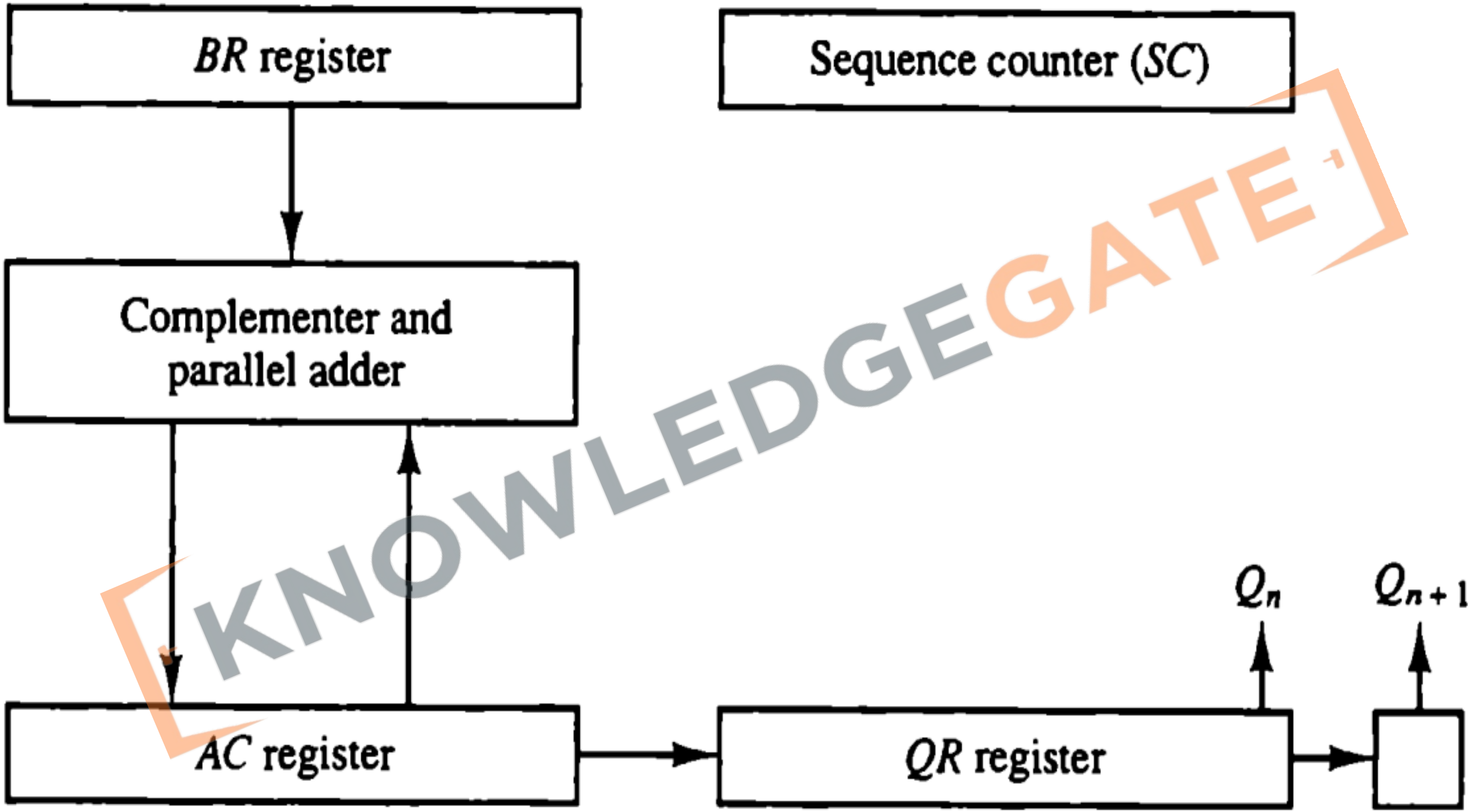
- Booth algorithm optimizes binary multiplication by reducing the number of additions and subtractions based on the multiplier bits.

- The process involves examining multiplier bits, then either adding, subtracting, or leaving the multiplicand unchanged before shifting the partial product.

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial values	
1001	0011	0	0111	A ← A − M	First cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A ← A + M	Third cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth cycle

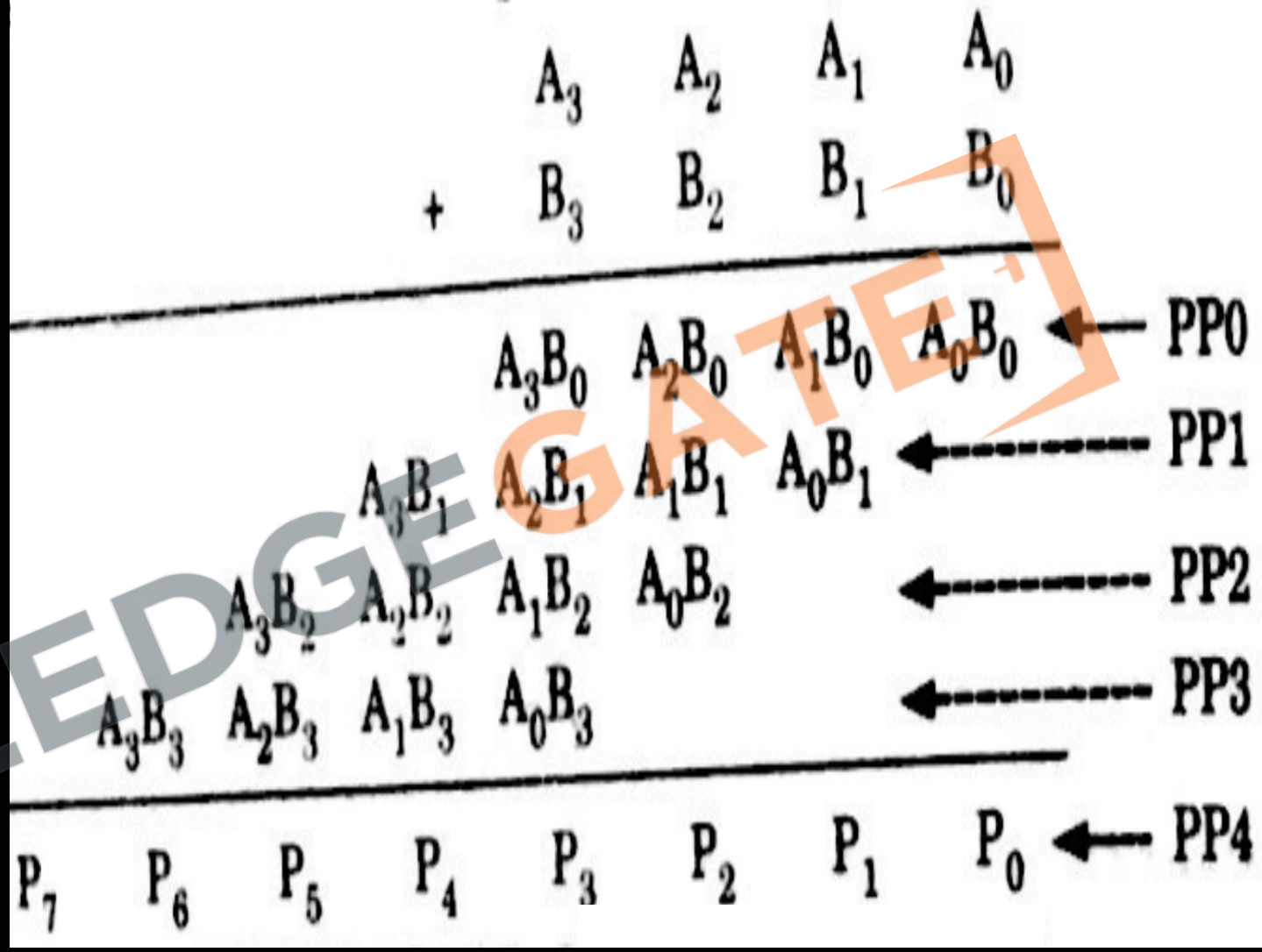
Example of Booth's Algorithm (7 × 3)

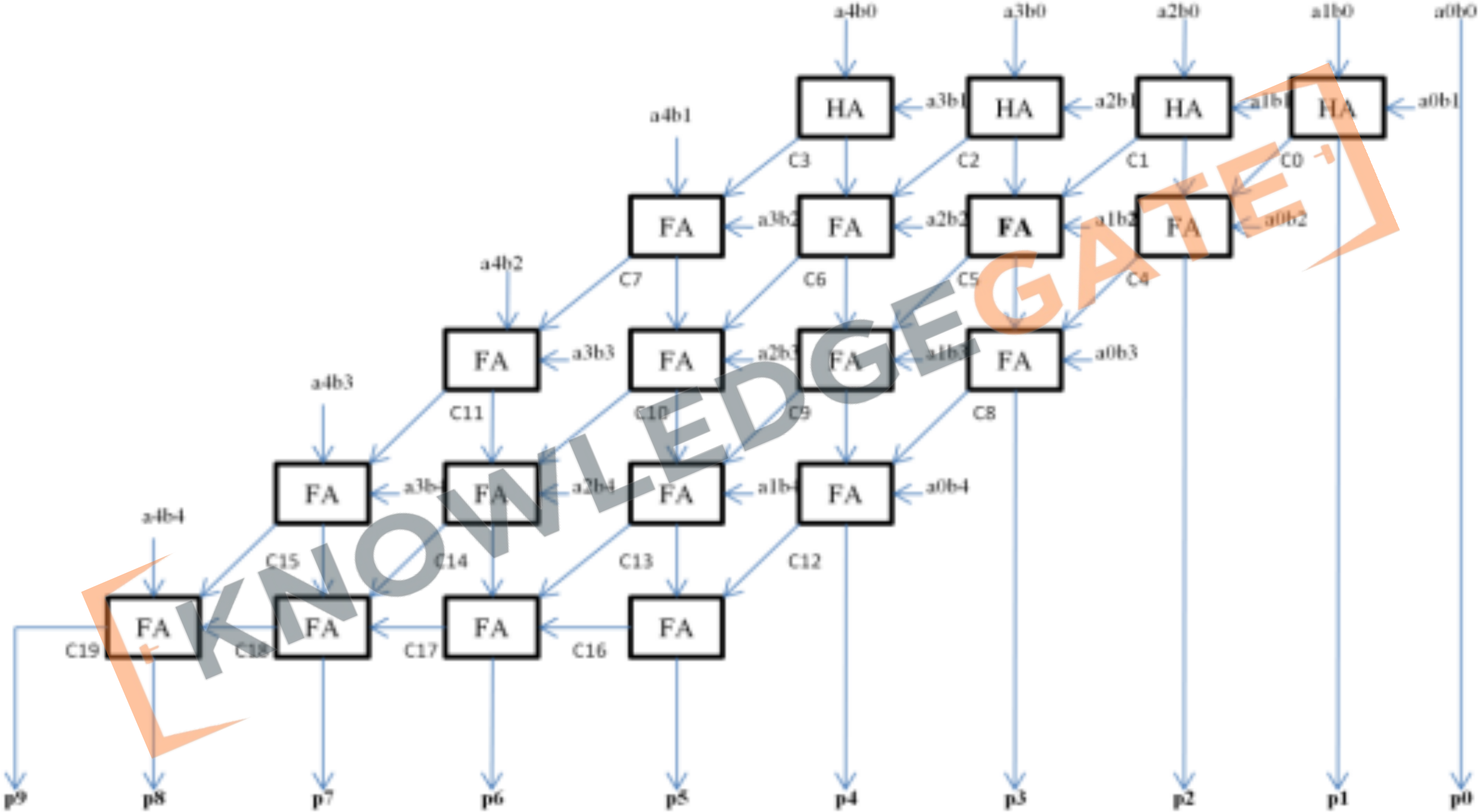




Array multiplier

- Array multiplier uses a grid of full and half adders to perform nearly simultaneous addition of product terms.
- AND gates are used to form these product terms before they are fed into the adder array.
- Unlike sequential multipliers that check bits one at a time, array multipliers form the product bits all at once, making the operation faster.
- The speed advantage comes at a cost of requiring a large number of gates, which became economical with the advent of integrated circuits.



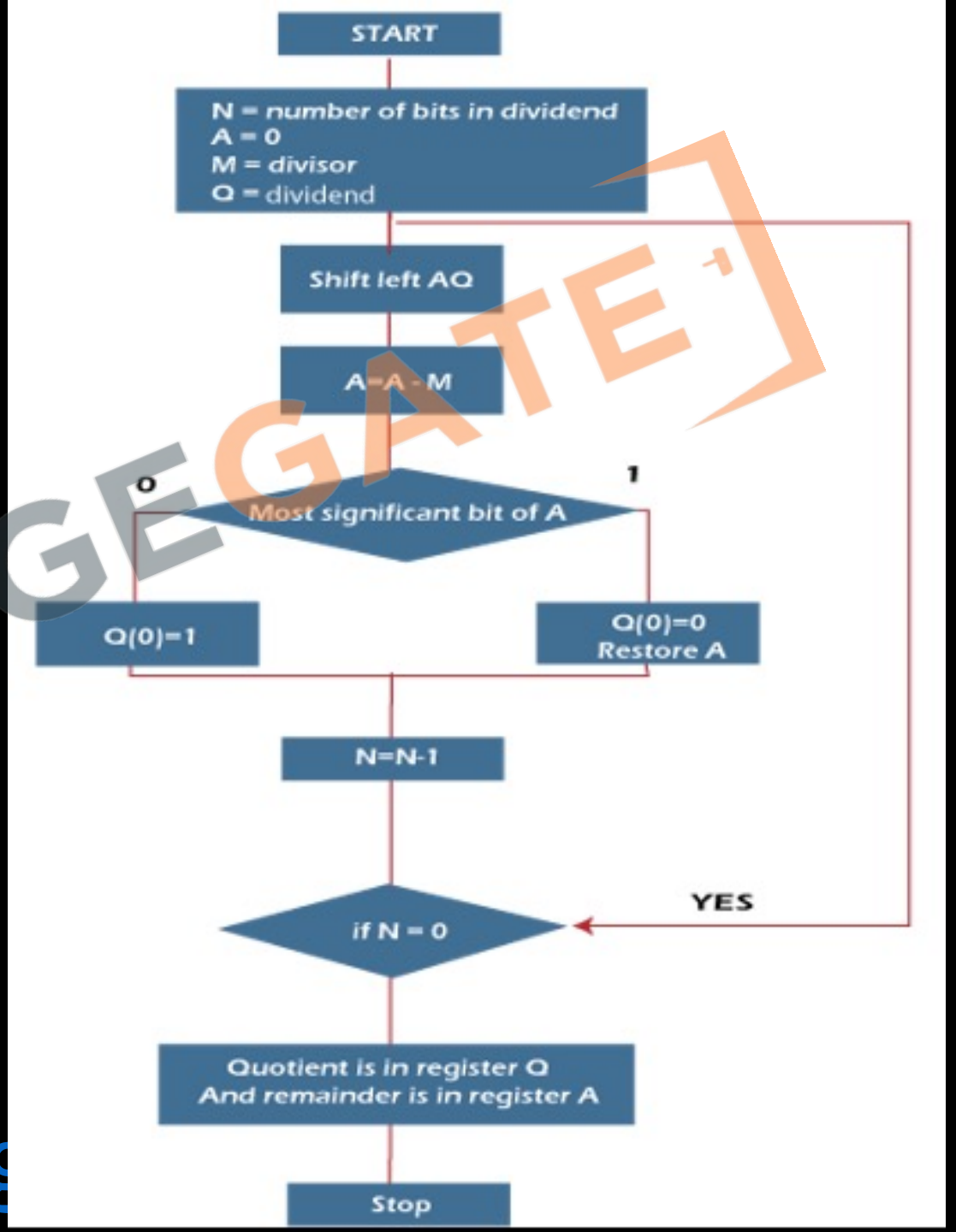


1 1 0 0

0 1 1 0

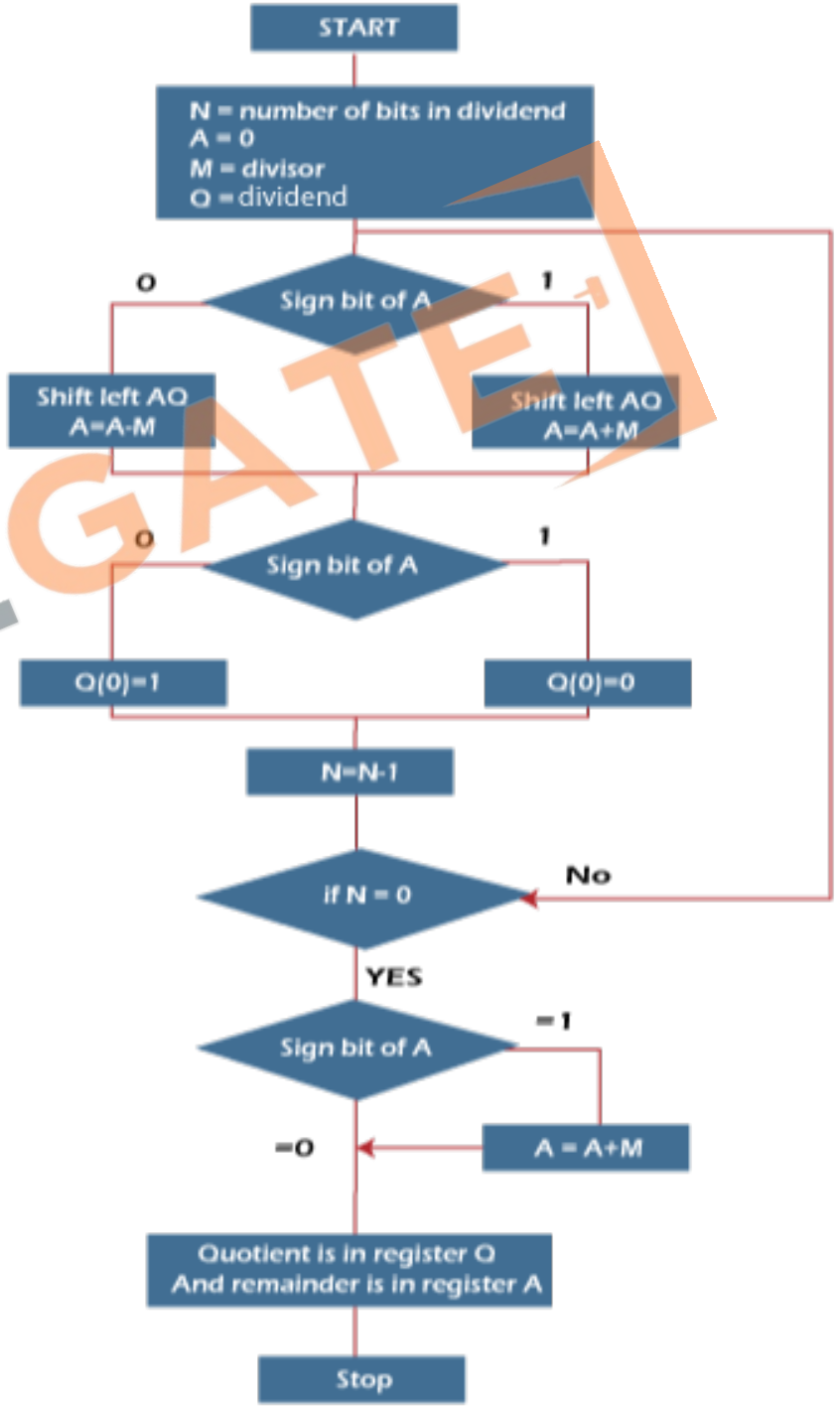
Restoring Division Algorithm

n	M	A	Q	Action/Operation
4	00011	00000	1011	Initialization
		00001	011?	SL AQ
		11110	011?	A = A-M
3	00011	00001	0110	QO ← 0
		00010	110?	SL AQ
		11111	110?	A = A-M
2	00011	00010	1100	QO ← 0
		00101	100?	SL AQ
		00010	100?	A = A-M
1	00011	00010	1001	QO ← 1
		00101	001?	SL AQ
		00010	001?	A = A-M
0	00011	00010	0011	QO ← 1



Non-Restoring Division Algorithm

n	M	A	Q	Action/Operation
4	00011	00000	1011	Initialization
		00001	011?	SL AQ
		11110	011?	A = A-M
3	00011	11110	0110	QO ← 0
		11100	110?	SL AQ
		11111	110?	A = A+M
2	00011	11111	1100	QO ← 0
		11111	100?	SL AQ
		00010	100?	A = A+M
1	00011	00010	1001	QO ← 1
		00101	001?	SL AQ
		00010	001?	A = A-M
0	00011	00010	0011	QO ← 1



Q Add -35 and -31 in binary using 8-bit registers, in signed 1's complement and signed 2's complement?

Floating point representation

- Problem with representation we have already studied is that it do not works well if the number to be stored is either too small or too large, is it take very large amount of memory.
- Imagine a number $6.023 * 10^{23}$ will require around 70 bits to be stored.
- So in scientific application or statics it is a problem to store very small or very large number.

- Floating point representation a special kind of sign magnitude representation.
- Floating point number is stored in mantissa/exponent form i.e. $m*r^e$
- Mantissa is a signed magnitude fraction for most of the cases.
- The exponent is stored in biased form.

- The biased exponent is an unsigned number representing signed true exponent.
- If the biased exponent field contains K bits, the biased = 2^{k-1}
- True value expression is $V = (-1)^{S}(.M)_2 * 2^{E-Bias}$, note it is explicit representation
- True value expression is $V = (-1)S(1.M)_2 * 2^{E-Bias}$, note it is implicit representation, it has more precision than explicit normalization.

- Biased exponent(E) = True exponent(e) + Bias
- Range of true exponent: from -2^{k-1} to $+2^{k-1}-1$, where k is number of bits assigned for exponent
- After adding bias 2^{k-1}, new range go from 0 to $2^{k}-1$

- How to convert a signed number into floating point representation
- The floating-point normalized number distribution is not uniform. Representable number are dense towards zero and spared towards max value
- This uneven distribution result negligible effect of rounding towards zero and dominant towards max value.

Q Represent -21.75 (s=1, k=7, m=8)

IEEE 754 floating point standard

- The **IEEE Standard for Floating-Point Arithmetic** (**IEEE 754**) is a technical standard for floating-point arithmetic established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE).

- The standard addressed many problems found in the diverse floating-point implementations that made them difficult to use reliably and portably. Many hardware floating-point units use the IEEE 754 standard.

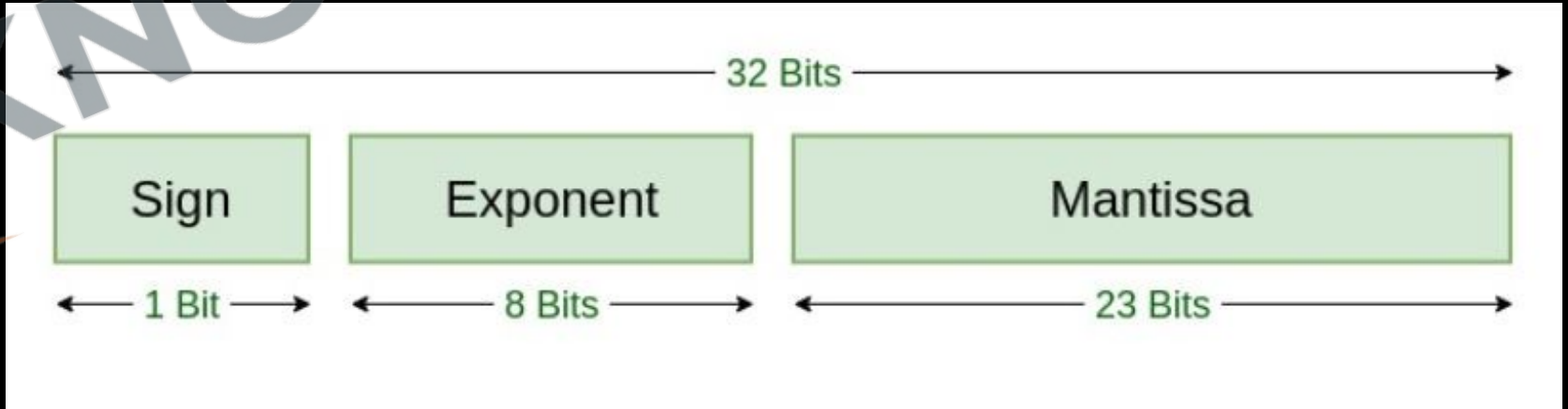
IEEE 754 floating point standard

- IEEE 754 standard represent floating point number standards
- It gives a provision for +-0 & +-∞ (by reserving certain pattern for Mantissa/Exponent pattern)
- there are number of modes for storage from half precision (16 bits) to very lengthy notations
- based of the system is 2
- the floating-point number can be stored either in implicit normalized form or in fractional form
- If the biased exponent field contains K bits, the biased = $2^{k-1} - 1$
- Certain Mantissa/Exponent pattern does not denote any number (NAN i.e. not a number)

Name	Common name	Mantissa Bits	Exponent bits	Exponent bias	E min	E max
binary16	Half precision	10	5	$2^{5-1}-1 = 15$	−14	+15
binary32	Single precision	23	8	$2^{8-1}-1 = 127$	−126	+127
binary64	Double precision	52	11	$2^{11-1}-1 = 1023$	−1022	+1023
binary128	Quadruple precision	112	15	$2^{15-1}-1 = 16383$	−16382	+16383
binary256	Octuple precision	236	19	$2^{19-1}-1 = 262143$	−262142	+262143

Single pression

Sign bit (1)	Exponent (8)	Mantissa (23)	Value
0	00…….0(E=0)	00…….0(M=0)	+0
1	00…….0(E=0)	00…….0(M=0)	-0
0	11……1(E=255)	00…….0(M=0)	+∞
1	11……1(E=255)	00…….0(M=0)	-∞
0/1	1<=E<=254	XX……X (M! =0)	Implicit normalised number
0/1	00…….0(E=0)	XX……X (M! =0)	Fraction
0/1	11……1(E=255)	XX……X (M! =0)	NAN (Not a Number)



Q Consider the following representation of a number in IEEE 754 single-precision floating point format with a bias of 127.

S : 1 E : 10000001 F : 11110000000000000000000

Here, S,E and F denote the sign, exponent, and fraction components of the floating point representation. The decimal value corresponding to the above representation (rounded to 2 decimal places) is _____.

Q Given the following binary number in 32-bit (single precision) IEEE-754 format:

00111110011011010000000000000000

The decimal value closest to this floating-point number is

Q The decimal value 0.5 in IEEE single precision floating point representation has

a) fraction bits of 000…000 and exponent value of 0

b) fraction bits of 000…000 and exponent value of −1

c) fraction bits of 100…000 and exponent value of 0

d) no exact representation

Double Precision

Instruction format

- In general, based on the number of operands or reference made in the instructions. Instructions can be classified into the following types: -
  - 3 Address Instruction
  - 2 Address Instruction
  - 1 Address Instruction
  - 0 Address Instruction

3 Address Instruction

Mode/Opcode	Destination	$Source_1$	$Source_2$

- Three-address instruction is a format of machine instruction. It has one opcode and three address fields. One address field is used for destination and two address fields for source.
- X = (A + B) x (C + D)

ADD R_1, A, B	R_1 <- M[A] + M[B]
ADD R_2, C, D	R_2 <- M[C] + M[D]
MUL X, R_1, R_2	M[X] <- R_1 x R_2

- It produces short program much easier to understand, but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

- Many bits are needed for instructions encoded in binary to define three addresses. More advanced processing and decoding circuits are needed.

2 Address Instruction

Mode/Opcode	Destination/Source$_1$	Source$_2$

- Here two addresses can be specified in the instruction. In effort to reduce the size of instruction here we remove the reference for result and Source$_1$ is used for storing the result.
- X = (A + B) x (C + D)

MOV	R_1, A	$R_1 \leftarrow M[A]$
ADD	R_1, B	$R_1 \leftarrow R_1 + M[B]$
MOV	R_2, C	$R_2 \leftarrow C$
ADD	R_2, D	$R_2 \leftarrow R_2 + D$
MUL	R_1, R_2	$R_1 \leftarrow R_1 * R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$

- Less length compares to 3 address, more efficient, no of register required are less.

- Here code optimization is relatively difficult.

1 Address Instruction

- One address instruction uses an implied accumulator (AC) register for all data manipulation.
- One operand is in the accumulator and the other is in the register or memory location. Implied means that the CPU already knows that one operand is in the accumulator so there is no need to specify it.
- X = (A + B) x (C + D)

LOAD	A	AC ← M[A]
ADD	B	AC ← AC + M[B]
STORE	T	M[T] ← AC
LOAD	C	AC ← M[C]
ADD	D	AC ← AC + M[D]
MUL	T	AC ← AC * M[T]
STORE	X	M[X] ← AC

0 Address Instruction

- Early computers which do not have complex circuit like ALU use few registers which are used as organised stack. While working on these stacks we do not require location of result as it is implied at the top of the stack. A stack organized computer does not use an address field for the instructions ADD and MUL. The push and pop instruction, however, need an address field to specify the operand that communicates with stack.
- X = (A + B) x (C + D)

PUSH	A	TOP ← A
PUSH	B	TOP ← B
ADD		TOP ← A+B
PUSH	C	TOP ← C
PUSH	D	TOP ← D
ADD		TOP ← C+D
MUL		TOP ← (C+D)*(A+B)
POP	X	M[X] ← TOP

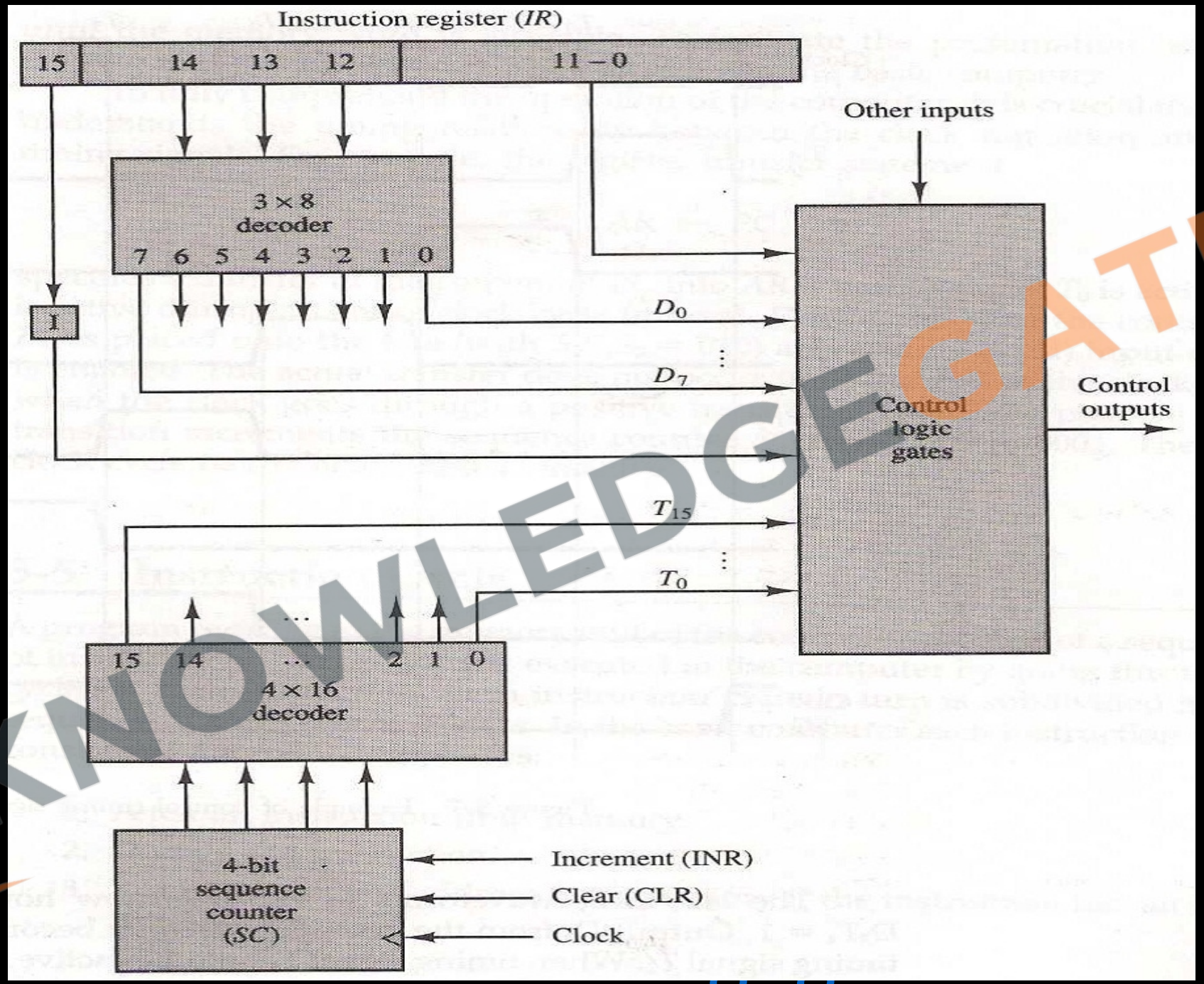
- To evaluate arithmetic expression in a stack computer, it is necessary to Covert the expression into reverse polish notation. The name zero-address is given to this type of computer because of the absences of an address field in the com.

- Here mode/opcode is sufficient to perform any operation.

- It is very simple and very short, but complex to design and speed is very poor.

Timing Circuit

- In order to perform a instruction we need to perform a number of micro-operators, and these micro-operations must be timed
  - AR ← PC
  - IR ← M[AR], PC ← PC + 1
- we should distinguish one clock pulse from another during the execution of instruction
- Sequence counter followed by decoder is used to generate time signals



Instruction Cycle - A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases. from fetching of instruction to the completion of execution of instruction whatever happens is called instruction cycle. The reason we called this cycle because it will happen for every instruction.

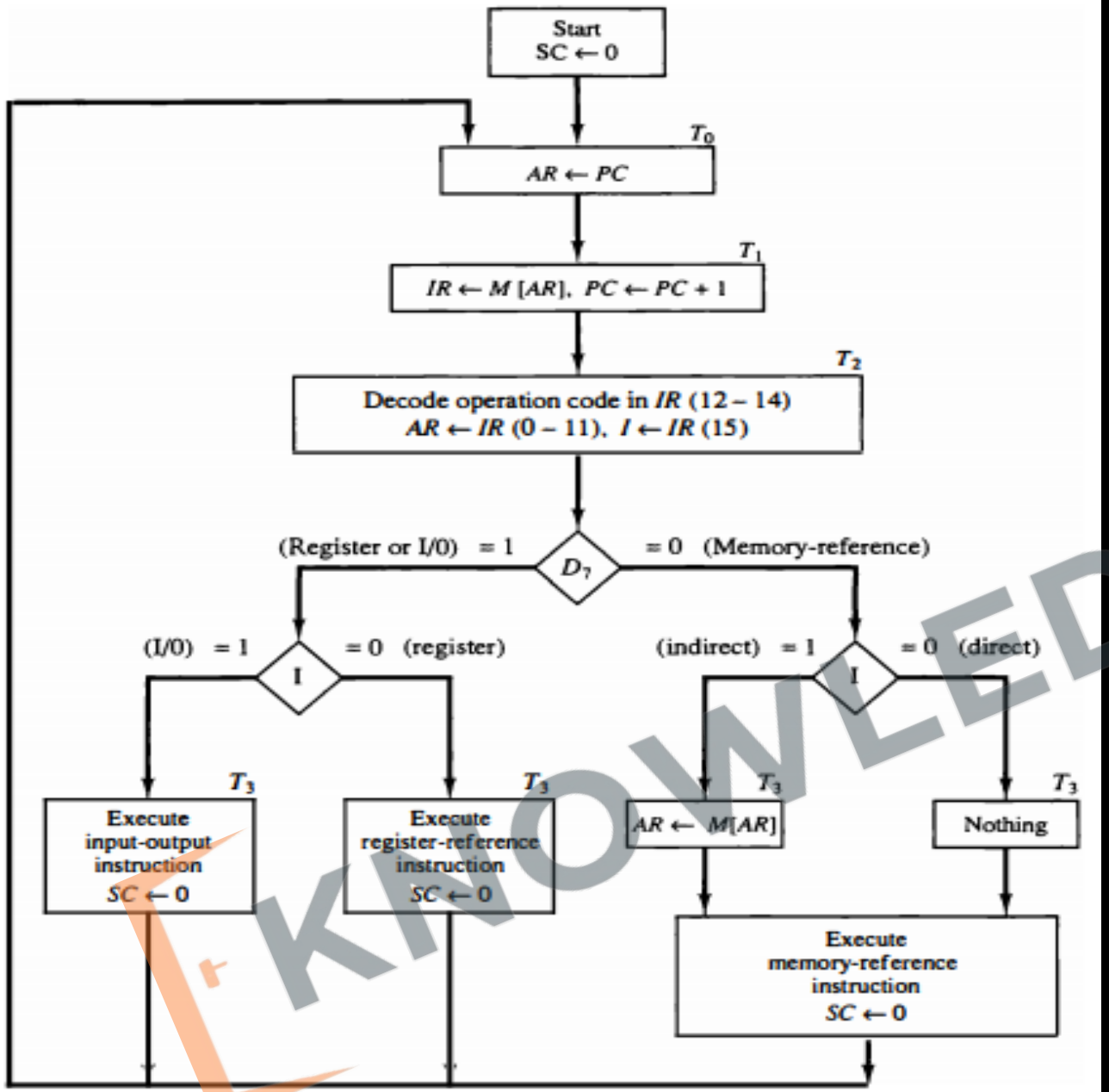
- Each instruction cycle consists of the following phases:
  - Fetch an instruction from memory.
  - Decode the instruction.
  - Read the effective address from memory if the instruction has an indirect address.
  - Execute the instruction.
  - Fetch and Decode

Micro-Operation

- A micro-operation is a simple operation that can be performed during one clock period.
- The result of this operation may replace the previous binary information of register or the result may be transferred to another register.
- Examples of micro-operations are shift, move, count, add and load etc.
- The micro-operations most often encountered in digital computers are classified into four categories:
  - **Register transfer micro-operations**: It transfer binary information from one register to another.
  - **Arithmetic micro-operations**: It perform arithmetic operation on numeric data stored in registers.
  - **Logic micro-operations**: It perform bit manipulation operations on non-numeric data stored in registers.
  - **Shift micro-operations**: It perform shift operations on data stored in registers.

- Initially, the program counter PC is loaded with the address of the first instruction in the program.
- The sequence counter SC is cleared to 0, providing a decoded timing signal T_0.
- After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2, and so on.
- The microoperations for the fetch and decode phases can be specified by the following register transfer statement

$$T_0: \quad AR \leftarrow PC$$
$$T_1: \quad IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$$
$$T_2: \quad D_0, \ldots, D_7 \leftarrow \text{Decode } IR(12\text{–}14), \quad AR \leftarrow IR(0\text{–}11), \quad I \leftarrow IR(15)$$

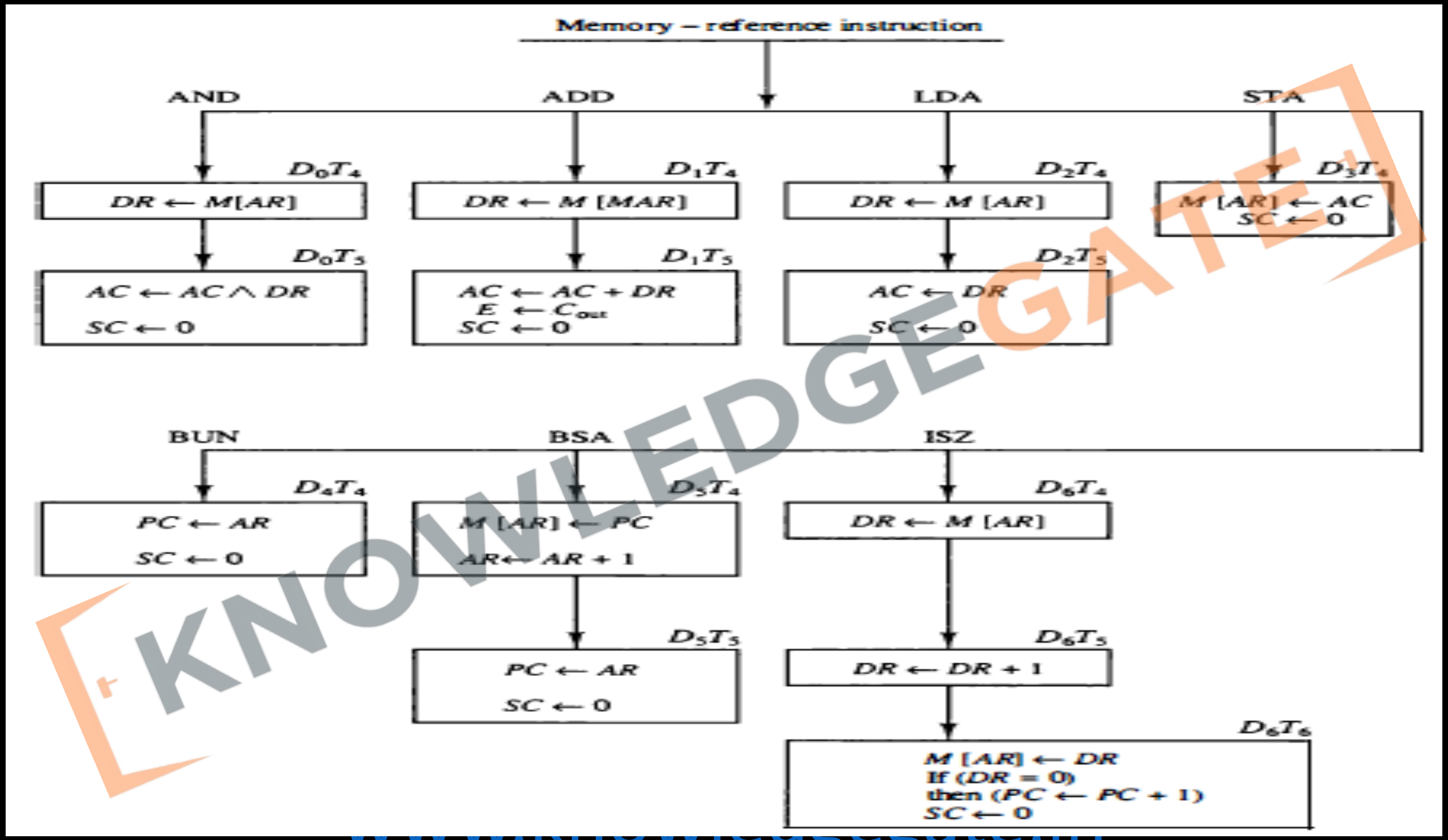


- Determine the Type of Instruction
- The timing signal that is active after the decoding is T_3.
- During time T_3, the control unit determines the type of instruction that was just read from memory.
- Decoder output D, is equal to 1 if the operation code is equal to binary 111.
- If $D_7 = 1$, the instruction must be a register-reference or input-output type.
- If $D_7 = 0$, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.
- Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I. If $D_7 = 0$ and I = 1, we have a memory reference instruction with an indirect address.
- In case of indirect address, it is necessary to read effective address from memory. The microoperation for the indirect address condition can be symbolized by the register transfer statement
- AR ← M [AR]
- When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR.
- The sequence counter SC is either incremented or cleared to 0 with every positive clock transition.

Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

- The effective address of the instruction is in the address register AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1. The execution of the memory-reference instructions starts with timing signal T4.
- Operation of each Instruction



- **AND to AC** - This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address.
- The result of the operation is transferred to AC. The microoperations that execute this instruction are:

$$D_0T_4: \quad DR \leftarrow M[AR]$$
$$D_0T_5: \quad AC \leftarrow AC \wedge DR, \quad SC \leftarrow 0$$

- **ADD to AC** - This instruction adds the content of the memory word specified by the effective address to the value of AC.

- The sum is transferred into AC and the output carry Cout is transferred to the E (extended accumulator) flip-flop. The microoperations needed to execute this instruction are:

$$D_1T_4: \quad DR \leftarrow M[AR]$$
$$D_1T_5: \quad AC \leftarrow AC + DR, \quad E \leftarrow C_{out}, \quad SC \leftarrow 0$$

LDA: Load to AC - This instruction transfers the memory word specified by the effective address to AC. The microoperations needed to execute this instruction are

$$D_2T_4: \quad DR \leftarrow M[AR]$$
$$D_2T_5: \quad AC \leftarrow DR, \; SC \leftarrow 0$$

STA: Store AC - This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation:

$$D_3T_4: \quad M[AR] \leftarrow AC, \quad SC \leftarrow 0$$

BUN: Branch Unconditionally - This instruction transfers the program to the instruction specified by the effective address. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation:

$$D_4T_4: \quad PC \leftarrow AR, \quad SC \leftarrow 0$$

BSA: Branch and Save Return Address - This instruction is useful for branching to a portion of the program called a subroutine or procedure

$$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$$

ISZ: Increment and Skip if Zero - This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.

D_6T_4: $DR \leftarrow M[AR]$

D_6T_5: $DR \leftarrow DR + 1$

D_6T_6: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

Register-Reference Instructions

- Register-reference instructions are recognized by the control when $D_7 = 1$ and $I = 0$.
- These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions. These 12 bits are available in IR(0-11).
- These instructions are executed with the clock transition associated with timing variable T_3.
- Each control function needs the Boolean relation $D_7I' T_3$.

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Input-Output Configuration

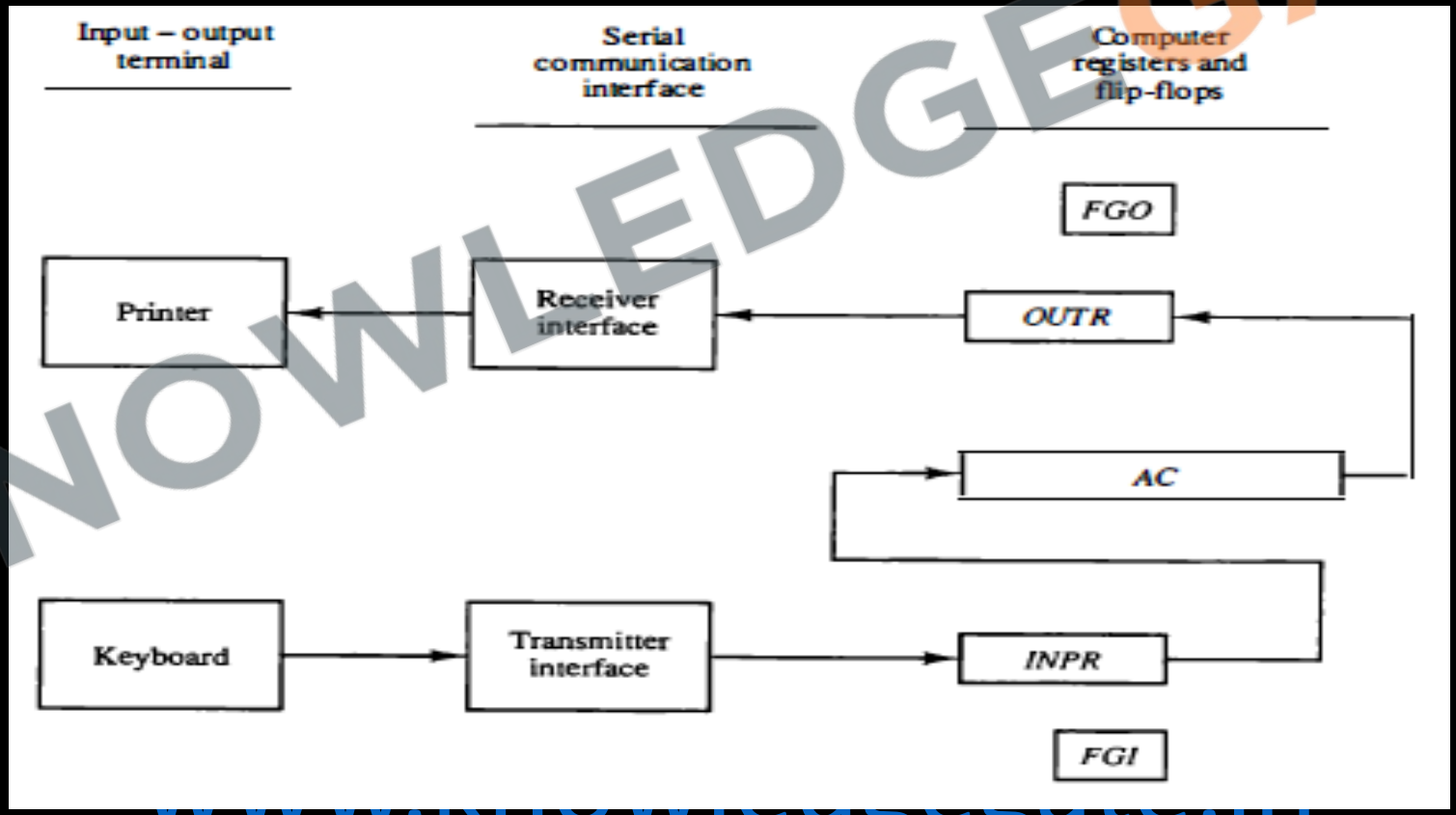
- The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code.
- The serial information from the keyboard is shifted into the input register INPR.
- The serial information for the printer is stored in the output register OUTR.
- These two registers communicate with a communication interface serially and with the AC in parallel.

$D_7IT_3 = p$
$IR(i) = B_i$, $i = 6, \ldots, 11$

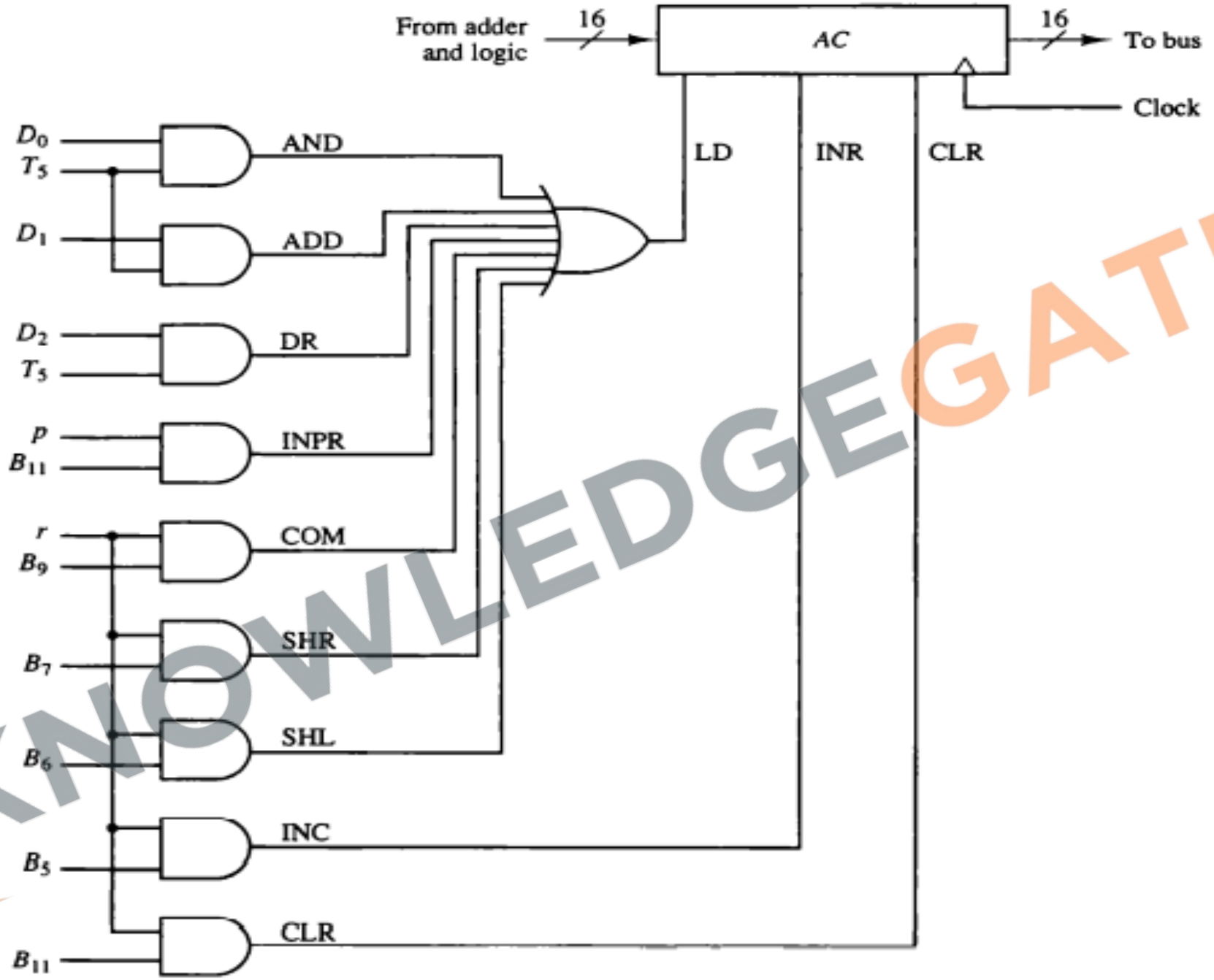
	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	Input char. to AC
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$	Output char. from AC
SKI	pB_9:	$if(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	$if(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

Input-Output Configuration

- The transmitter interface receives serial information from the keyboard and transmits it to INPR.
- The receiver interface receives information from OUTR and sends it to the printer serially.
- The 1-bit input flag FGI is a control flip-flop.
- The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- The flag is needed to synchronize the timing rate difference between the input device and the computer.



Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off



Reduced Instruction Set Computer (RISC)

- Computers that use fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often are classified as RISC.
- Relatively few instructions and few addressing modes
- Memory access limited to load and store instructions as all operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format, single-cycle instruction execution.
- Hardwired rather than microprogrammed control.
- The small set of instructions of a typical RISC processor consists mostly of register-to-register operations, Thus, each operand is brought into a processor register with a load instruction. All computations are done among the data stored in processor registers. Results are transferred to memory by means of store instructions
- A RISC computer has a small set of simple and general instructions, rather than a large set of complex and specialized ones.

- Another success from this era was IBM's effort that eventually led to the IBM POWER instruction set architecture, PowerPC, and Power ISA. As these projects matured, a variety of similar designs flourished in the late 1980s and especially the early 1990s, representing a major force in the Unix workstation market as well as for embedded processors in laser printers, routers and similar products.
- The many varieties of RISC designs include ARC, Alpha, Am29000, ARM, Atmel AVR, Blackfin, i860, i960, M88000, MIPS, PA-RISC, Power ISA (including PowerPC), RISC-V, SuperH, and SPARC.
- The use of ARM architecture processors in smartphones and tablet computers such as the iPad and Android devices provided a wide user base for RISC-based systems. RISC processors are also used in supercomputers such as Summit, which, as of January 2020, is the world's fastest supercomputer as ranked by the TOP500 project.

Capable of 200 petaFLOPS

Complex Instruction Set Computer (CISC)

- A computer with a large number of instructions is classified as a complex instruction set computer, abbreviated CISC.
- A large number of instructions-typically from 100 to 250 instructions
- Some instructions that perform specialized tasks and are used infrequently
- A large variety of addressing modes-typically from 5 to 20 different modes
- Variable-length instruction formats
- Instructions that manipulate operands in memory

- A **complex instruction set computer** (**CISC**) is a computer in which single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instructions.

- The term was retroactively coined in contrast to reduced instruction set computer (RISC) and has therefore become something of an umbrella term for everything that is not RISC, from large and complex mainframe computers to simplistic microcontrollers where memory load and store operations are not separated from arithmetic instructions.

- A modern RISC processor can therefore be much more complex than, say, a modern microcontroller using a CISC-labeled instruction set, especially in the complexity of its electronic circuits, but also in the number of instructions or the complexity of their encoding patterns.

- The only typical differentiating characteristic is that most RISC designs use uniform instruction length for almost all instructions, and employ strictly separate load/store-instructions.

- Examples of instruction set architectures that have been retroactively labeled CISC are System/360 through z/Architecture, the PDP-11 and VAX architectures, Data General Nova and many others.
- Well known microprocessors and microcontrollers that have also been labeled CISC in many academic publications include the Motorola 6800, 6809 and 68000-families; the Intel 8080, iAPX432 and x86-family; the Zilog Z80, Z8 and Z8000-families; the National Semiconductor 32016 and NS320xx-line; the MOS Technology 6502-family; the Intel 8051-family; and others.
- Some designs have been regarded as borderline cases by some writers. For instance, the Microchip Technology PIC has been labeled RISC in some circles and CISC in others. The 6502 and 6809 have both been described as "RISC-like", although they have complex addressing modes as well as arithmetic instructions that operate on memory, contrary to the RISC-principles.

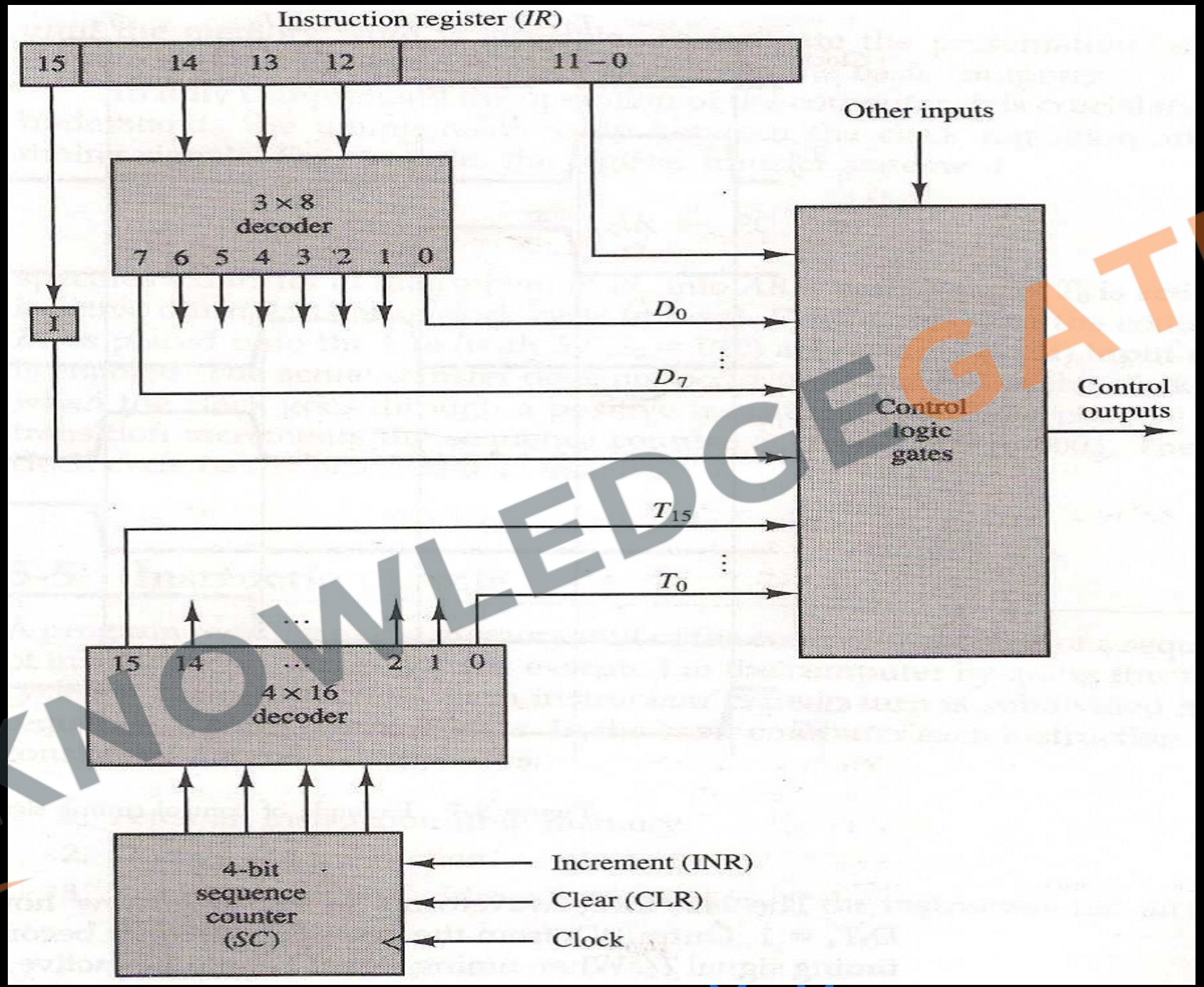
Complex instruction set format (CISC)	Reduced instruction set format (RISC)
Large number of instructions (around 1000) Application point of view they are useful but for system designer it is a headache	Relatively Few numbers of instruction, only those instructions which are strictly required, and in case of implementation of a complex function a set of simple instruction will perform together
Some instruction is there which perform specific task and are used infrequently for e.g. instructions for testing	Few instructions will be there which will be preforming more frequent work
Large number of addressing mode, leading to lengthen instruction, but compilation and translation will be faster	Number of instruction mode will be less, hardly 3 to 4
Variable length instruction format	Fixed length easy to decode instruction format
Instruction that manipulate operand in memory	Do not perform operation directly in memory, first load them in register and then perform, so all operations will be done with in the registers of CPU
Powerful but costly	Relatively less powerful but cheap
Microprogrammed control unit, relatively slow	Hardwired control unit, so fast

Designing of Control Unit

- The Control signal can be generated either by hardware (hardware Control unit design) or by (memory + h/w) (micro-program control unit design)
- In the hardware control unit design, each control signal is expressed as SOP expression and realized by digital hardware.
- The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as “hardwired”.
- Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.

Q Consider a hypothetical control unit implemented by hardware it uses three, 8-bits register A, B, C. It supports the two instruction I_1 and I_2, the following table gives control signals required for each micro-operation for both instructions?

Micro-operation	Control Signal	
	I_1	I_2
T_1	A_{in}, B_{out}	A_{in}, A_{out}
T_2	B_{in}, C_{out}	C_{in}, A_{out}
T_3	B_{in}, B_{out}	C_{in}, C_{out}
T_4	A_{in}, A_{out}	A_{in}, B_{out}



Conclusion

- It is the fastest control unit design and it is suitable for real time application. (The RISC machine employ hardware control unit), Hardwired control is faster than micro-programmed control.
- A controller that uses this approach can operate at high speed. RISC architecture is based on hardwired control unit.
- Limitation: any modification to the design require re-design & re-connection of H/W component. it is not flexible; hence hardware control unit is not suitable for design & de-bugging environment. This problem is resolved using Micro-programmed control unit design.

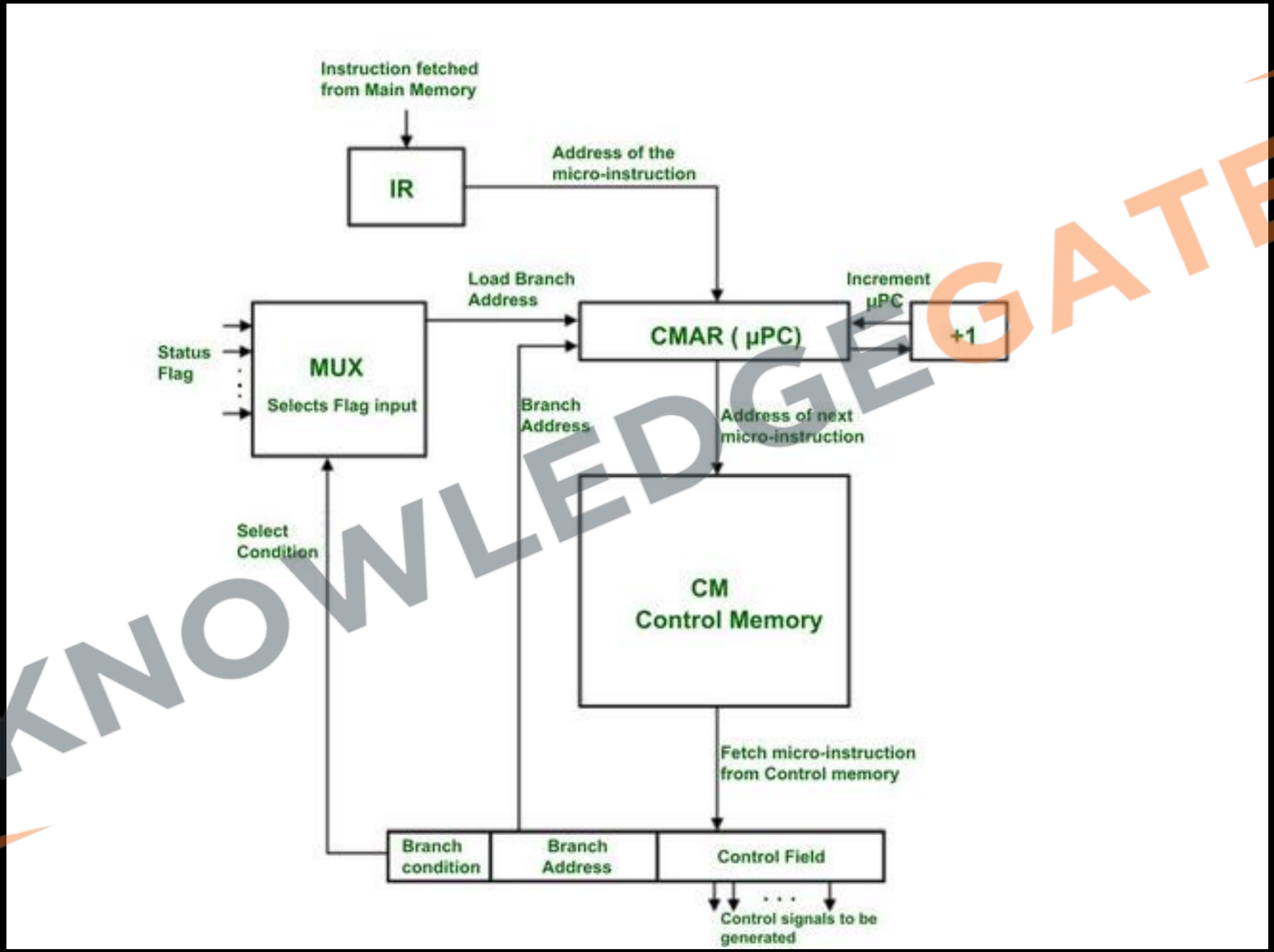
Micro-Programmed Control Unit

- The idea of microprogramming was introduced by Maurice Wilkes in 1951 as an intermediate level to execute computer program instructions.
- Microprograms were organized as a sequence of *microinstructions* and stored in special control memory.
- The main advantage of the microprogram control unit is the simplicity of its structure. Outputs of the controller are organized in microinstructions and they can be easily replaced.
- In this, the binary patterns of control signals are stored in control memory. After accessing word from the control memory, hardware is used to generate the control signals.

- Any changes can affects only the control word but not the hardware
- The design is flexible & it is used in CISC system.
- Based on the no of words in the control word, the micro-programming may be
  - Horizontal micro-Program
  - Vertical Micro-program

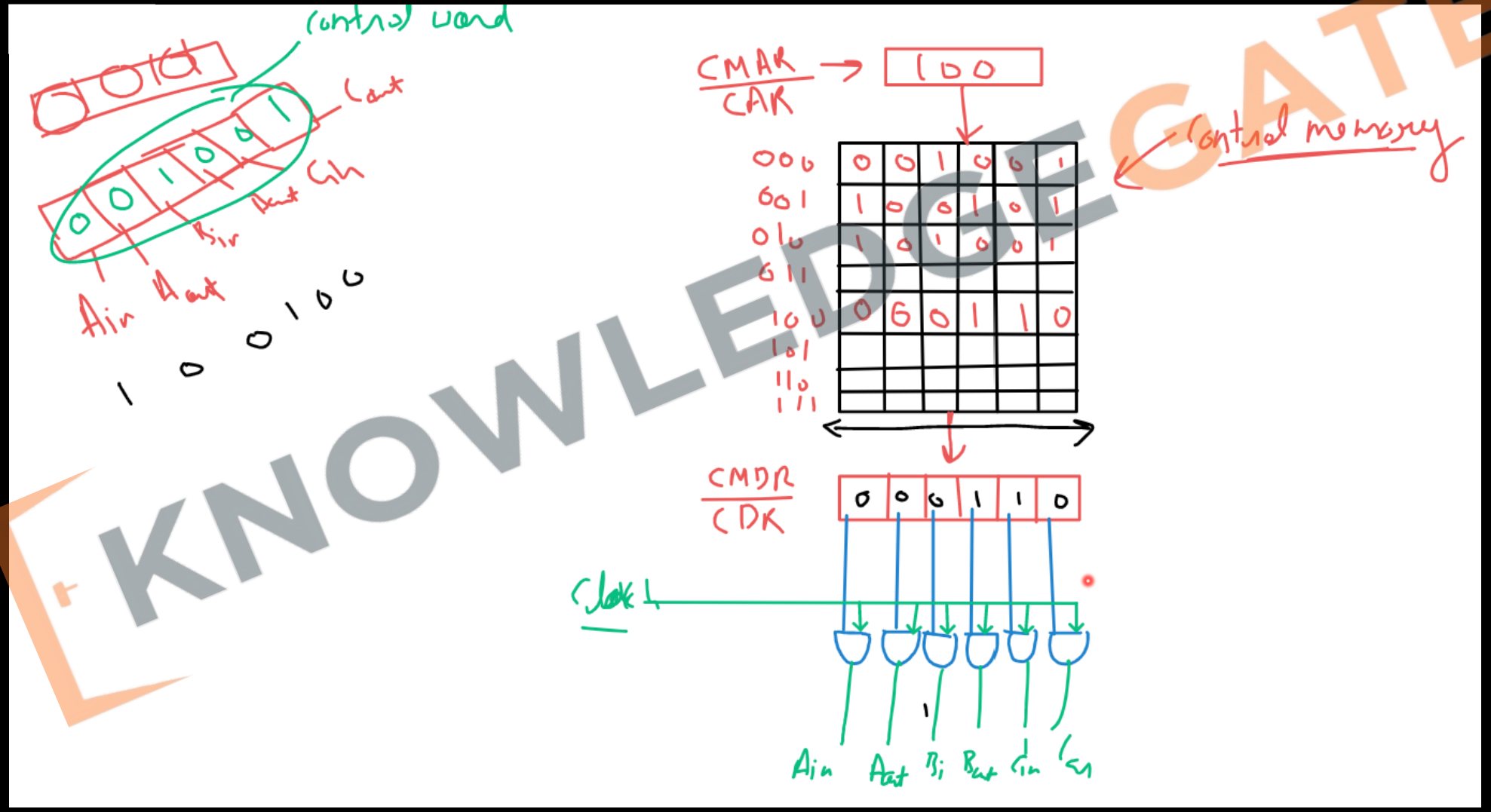
Characteristics	Hardwired	Micro-programmed Control
Speed	Fast	Slow
Implementation	Hardware	Software
Flexibility	Not Flexible	Flexible
Ability to handle complex instruction set	Diffcult	Easier
Design process	Diffcult for more opertion	Easy
Memory	Not Used	Control memory used
Chip are efficiency	Uses less area	Uses more area
Used in	RISC	CISC

Horizontal Micro-Programmed Control Unit



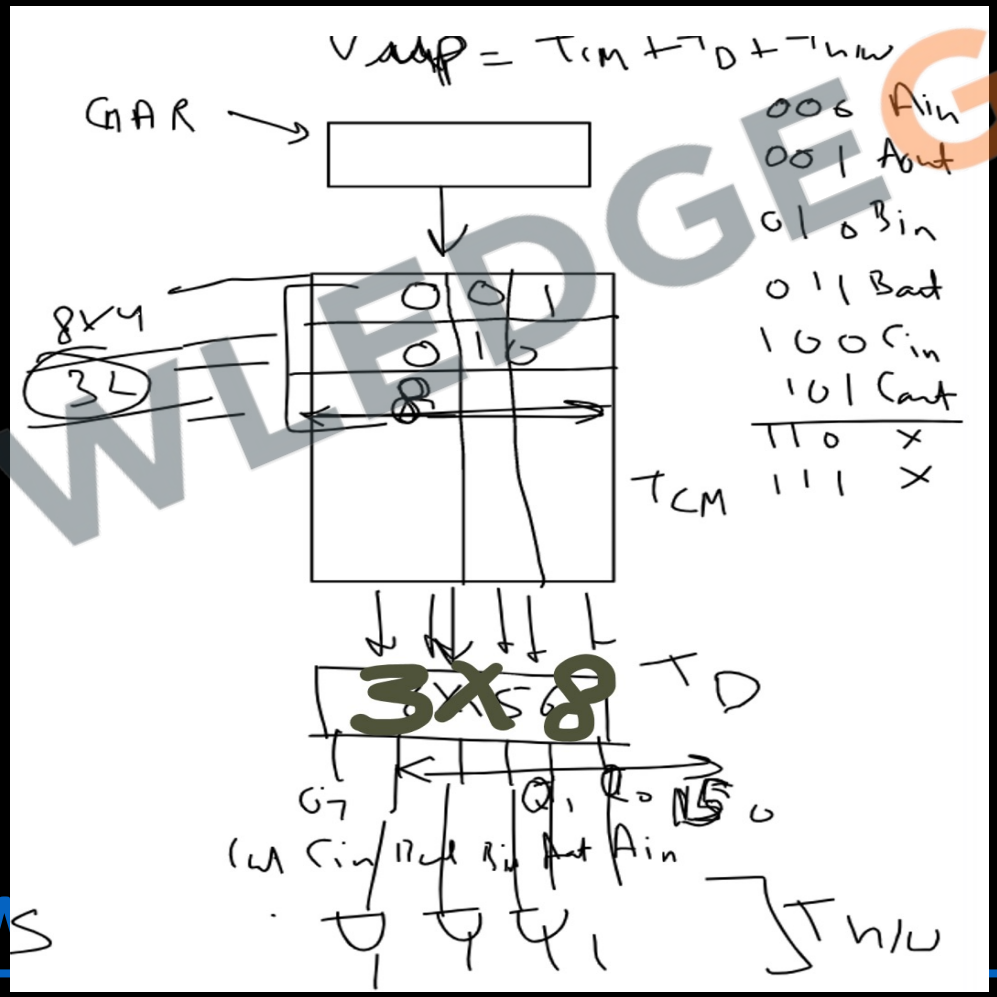
Horizontal micro-Program

- The horizontal micro-programmed provides higher degree of parallelism & it is suitable in multi-processor system. it requires more bits for control word (1 it for every control signal)



Vertical Micro-program

- The vertical micro-programming reduces the size of control words by encoding, control signal pattern before it is stored in control memory. it offers more flexibility than horizontal micro-programming.
- The pattern with vertical-programming is the maximum degree of parallelism is 1(due to the decoder).



Horizontal	Vertical
Long Format	Short Format
Ability to express a high degree of parallelism	Limited ability to express parallel micro-operations
Little encoding of control information	Considerable encoding of the control information
Usefull when higher operating speed is desired	Slower operating speed

Conclusion

- The combinational Horizontal micro-Program and Vertical Micro-program controlled unit is preferred in most application.

Control word sequencing

- In order to implement, the micro-program, control words are to be sequentially from control memory. One address instruction is used for performing the control word sequencing.

Flag	Control Signal	Address

Memory in Computer

- Memory is one of the most crucial components of a computer. Some reasons why memory plays a vital role in computer's overall performance :
  - **Storage of data**: Memory allows the processor to access the data quickly.
  - **Multitasking**: Memory enables multiple tasks simultaneously, as it allows the processor to switch between different programs and data efficiently.
  - **Running applications**: Applications and programs require a certain amount of memory to run.
  - **Operating system**: The operating system also requires memory to run smoothly.
- In summary, memory is critical for the efficient operation of a computer and plays a key role in determining its performance.

Importance of Memory

- The criteria for a ideal memory in a computer system depend on the specific requirements of the system, but there are several general factors that are important for any type of memory:
  - Speed:
  - Capacity:
  - Stability:
  - Cost:
  - Scalability

Cycle

Car

Airbus

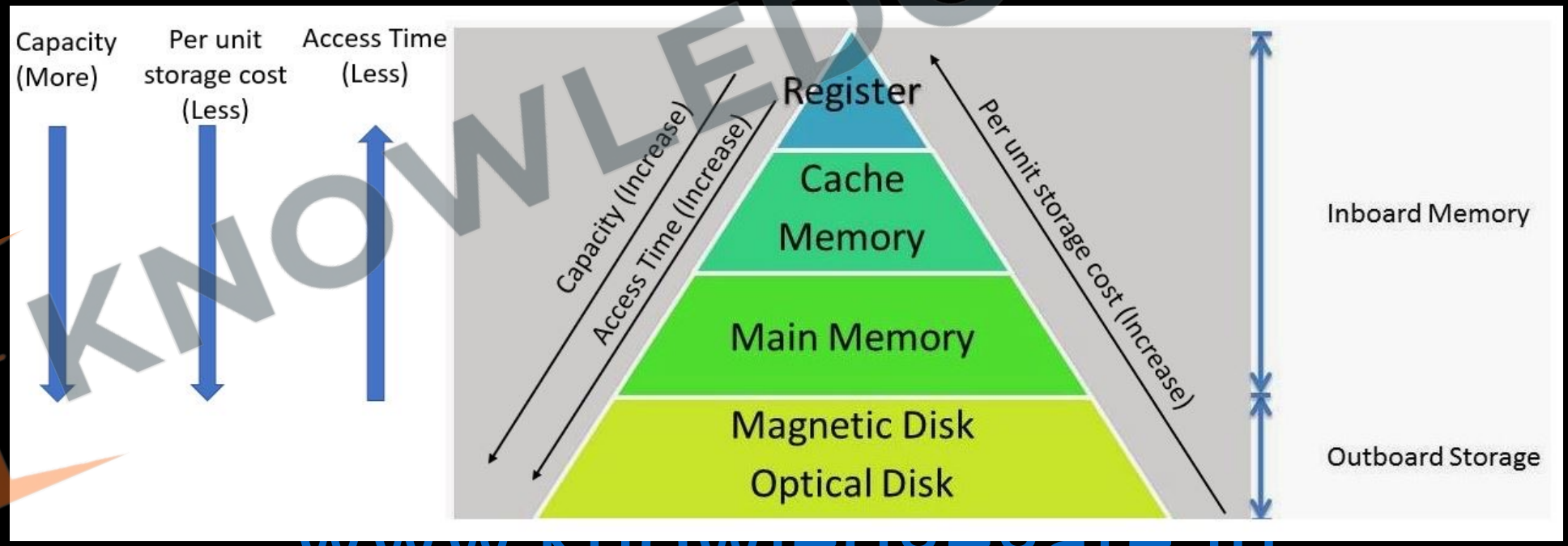
Lathi

303

AK-47

सत्यमेव जयते

- The memory hierarchy in a computer system refers to different levels of memory storage with varying capacity, speed, and cost. The memory hierarchy is designed to provide the CPU with fast access to data and a large storage area for longer-term data. The levels of the memory hierarchy, in order of speed, are:
  - **Registers**: Fastest and smallest memory built into the CPU.
  - **Cache**: Small, fast memory usually built into the CPU and used for frequently accessed data.
  - **Main Memory (RAM)**: Primary storage for data and instructions, faster than storage devices.
  - **Secondary Memory (Storage Devices):** Used for permanent storage, slower than main memory but with larger capacity.



How CPU Uses Memory Hierarchy

Cache

Showroom

Main Memory

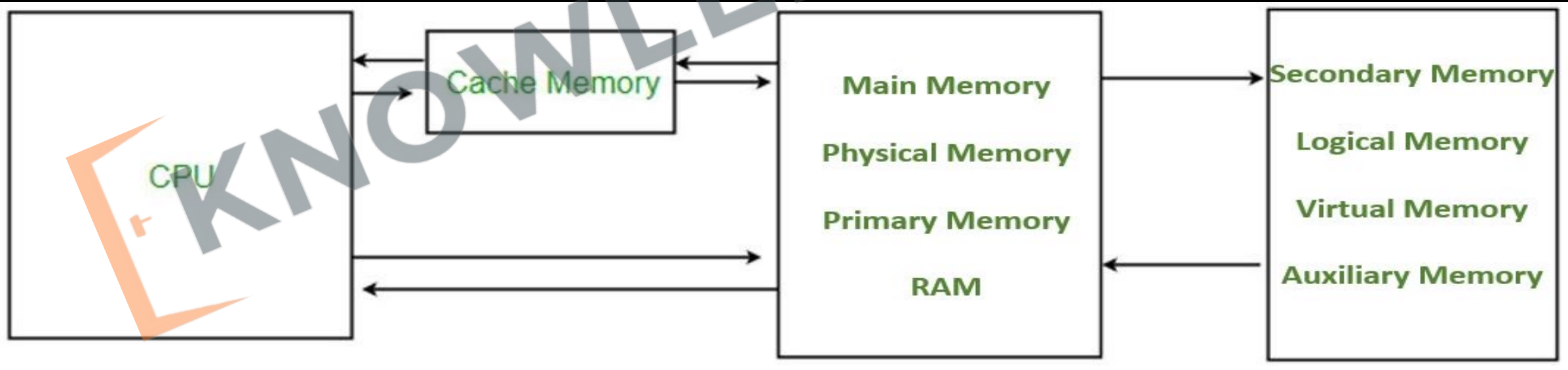
Go down

Secondary Memory

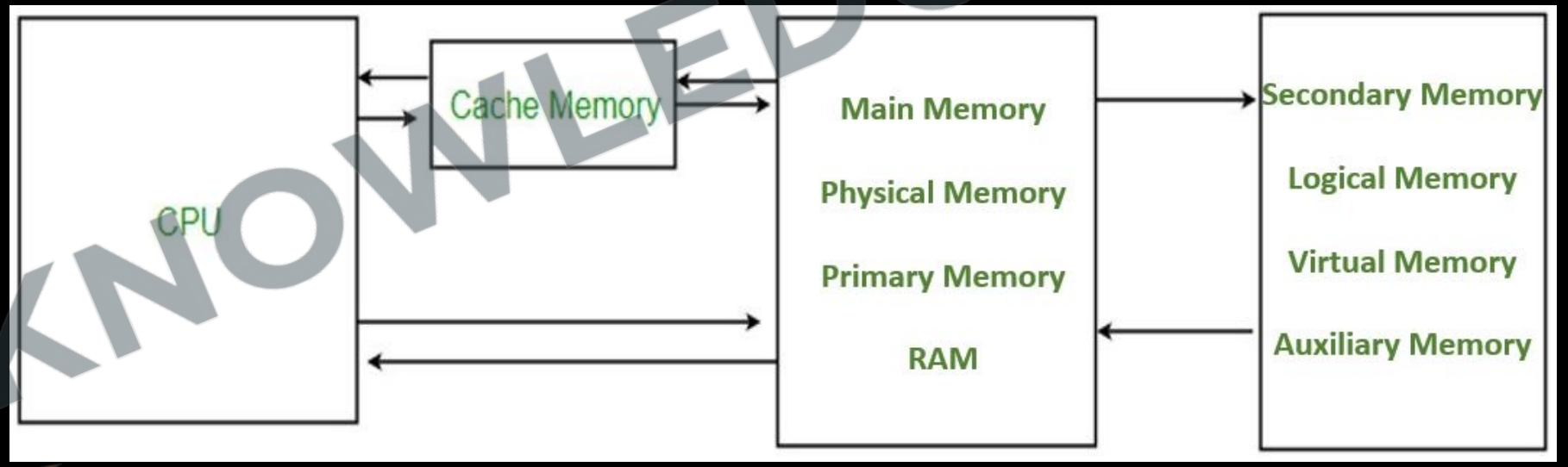
Factory

How CPU Uses Memory Hierarchy

- Cache memory boosts processing speed by swiftly providing data to the CPU. It bridges the speed gap between the processor and main memory.
- Main memory holds data for immediate access by the CPU. If data isn't in the main memory, it's fetched from the auxiliary or secondary memory.
- For efficient processing, data sought by the CPU should ideally be in Cache. If not, it's fetched from main memory or, as a last resort, secondary memory.

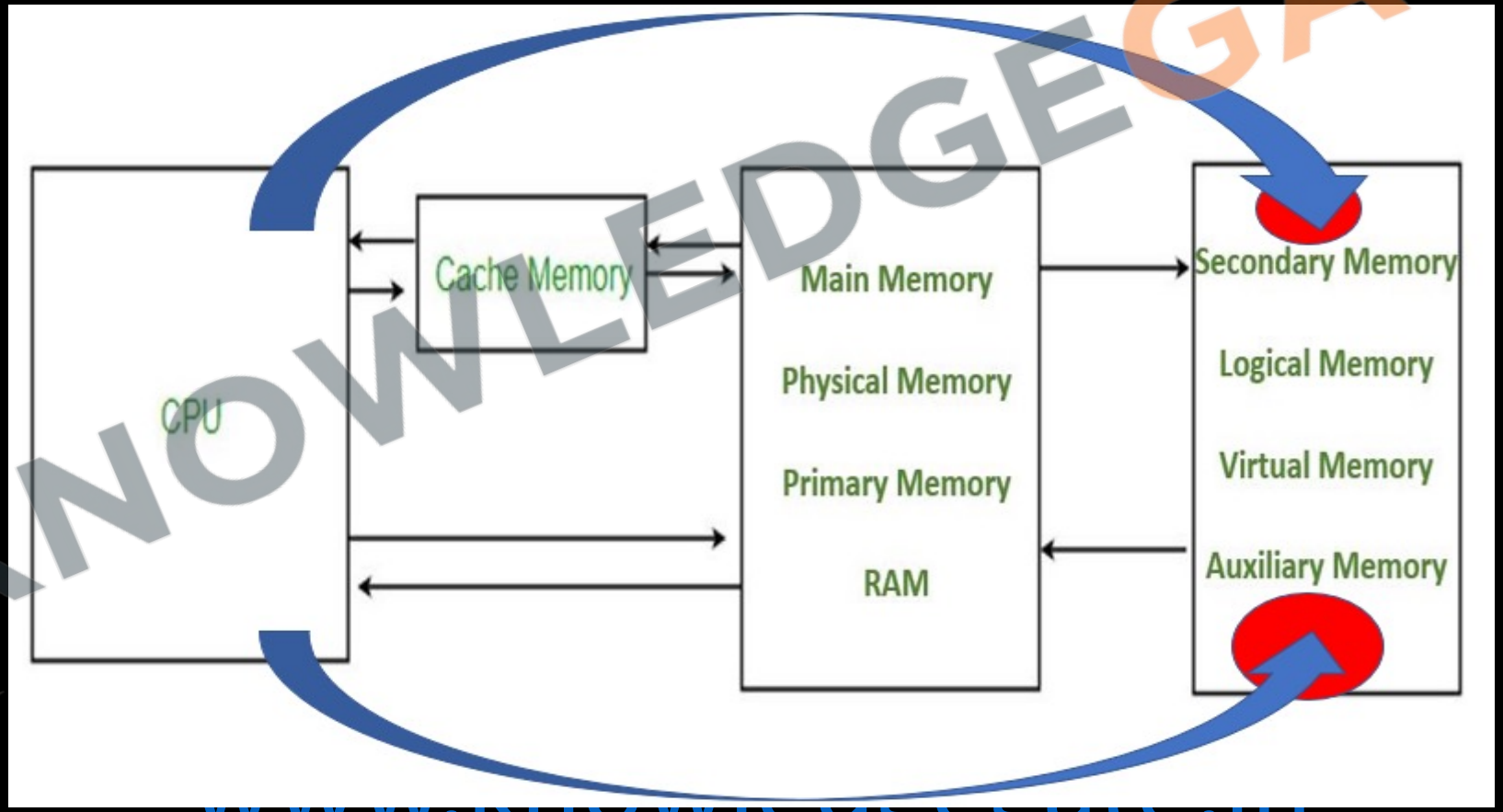


- But this is a difficult task to do on your computer as it has, 1 TB of Secondary Memory, 16 GB of Main Memory, but only 768KB, 4MB, 16 MB of L_1, L_2, L_3 cache respectively.
- If somehow we can estimate what data CPU will require if future we can prefetch in Cache and Main Memory. Locality of reference Helps we to perform this estimation.

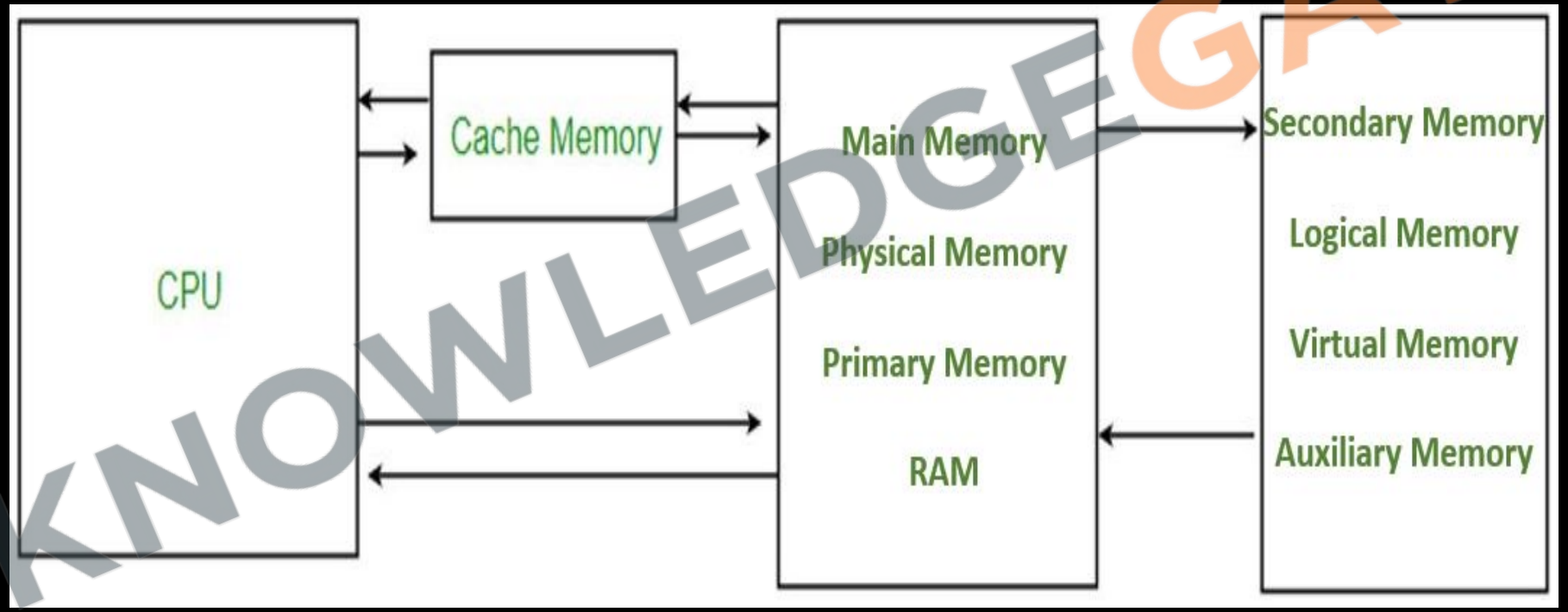


Locality of Reference

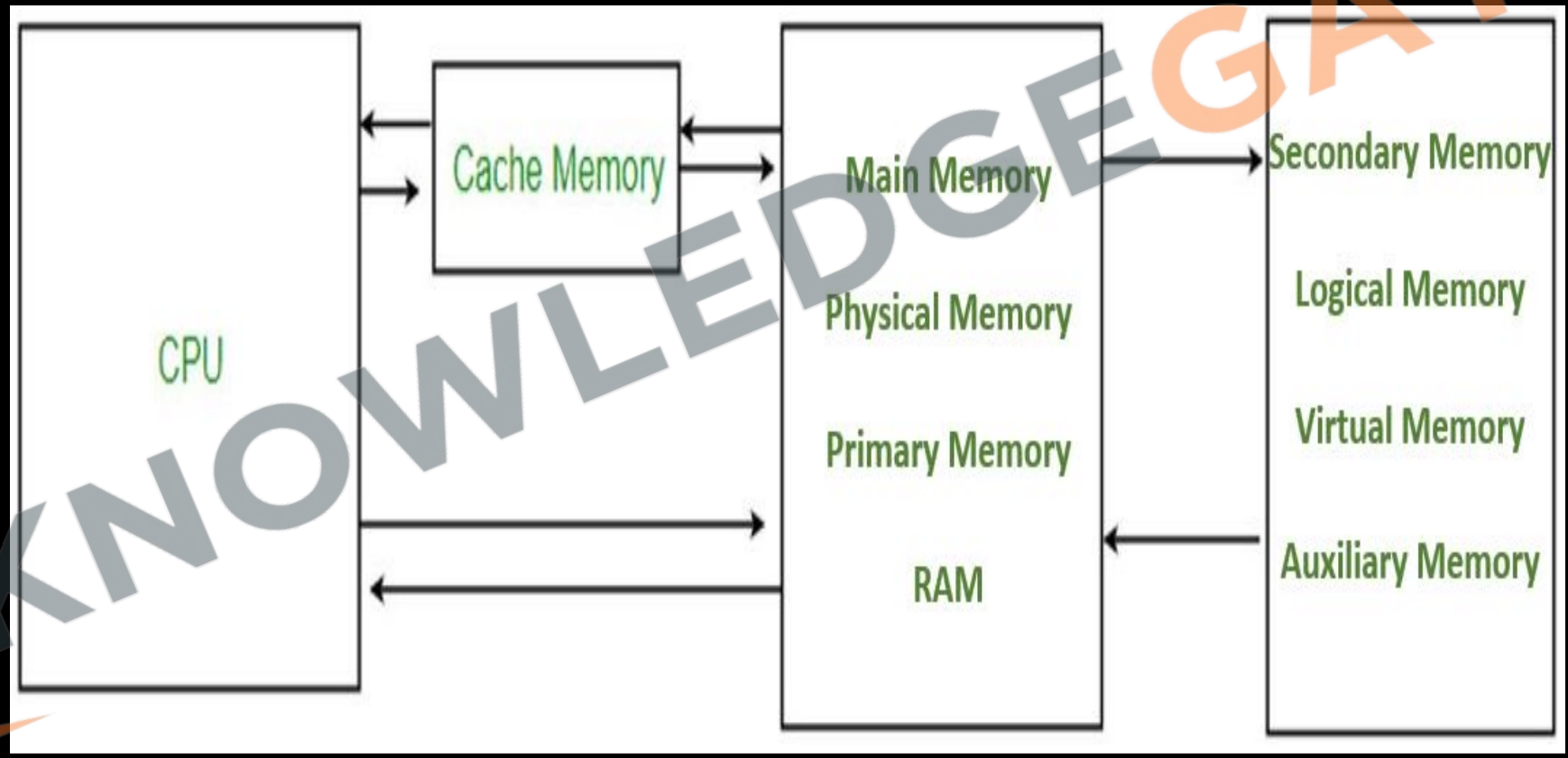
- Locality of reference refers to the tendency for a program to access a small portion of its memory at any given time, while the vast majority of its memory remains unused. There are two types of locality of reference: temporal locality and spatial locality.



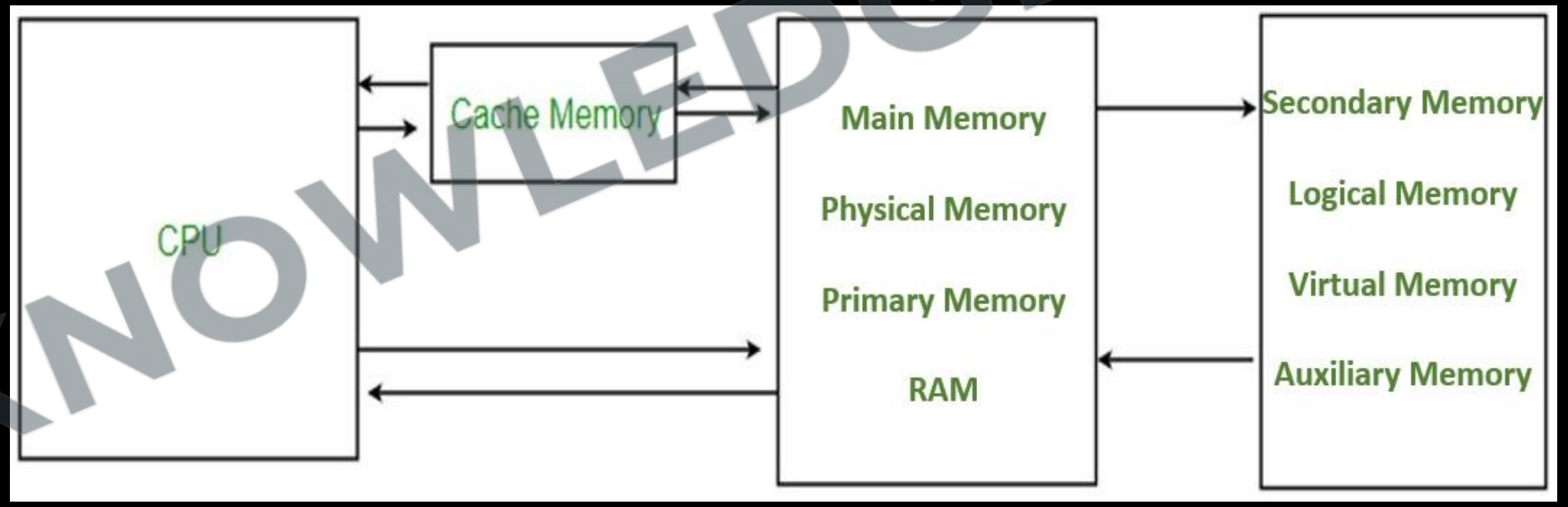
- **Spatial Locality:** Refers to the tendency for a program to access memory locations that are close to each other in memory. This means that if a program accesses a memory location, it is likely to access other nearby memory locations soon after.



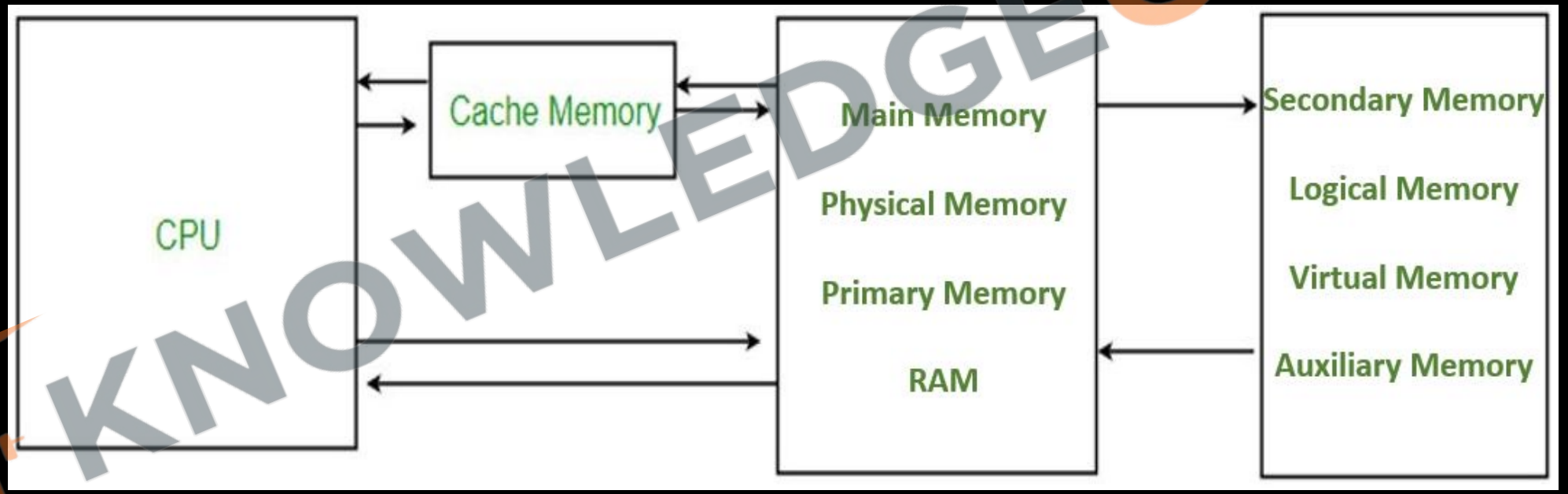
- **Temporal Locality:** Temporal locality refers to the tendency for a program to access recently used memory locations repeatedly. This means that if a program accesses a memory location, it is likely to access that same location again in the near future.

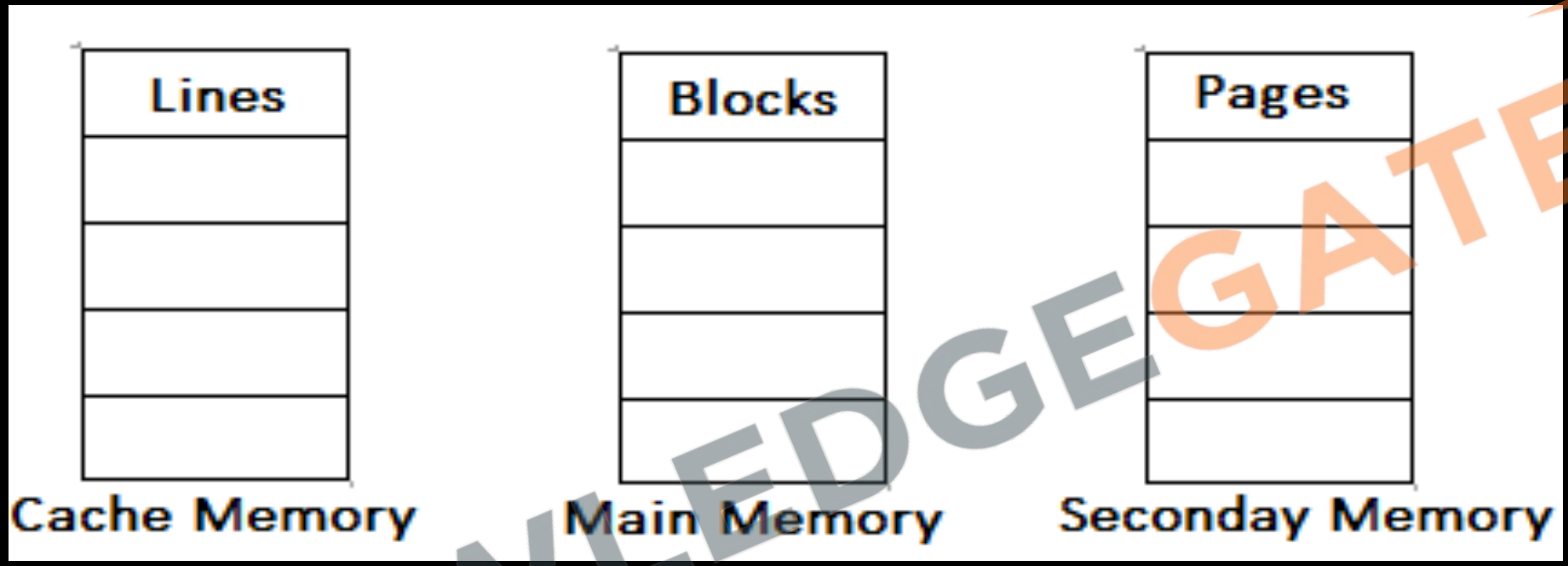


- **Cache Hit:** When a program requests data, the cache is searched first, and if the data is found in the cache, it is returned to the program. This is known as a cache hit.
- **Hit Ratio:** The cache hit rate is the ratio of data access requests that are being satisfied by the cache.
- **Hit Latency:** Hit latency refers to the time it takes for a computer system to retrieve data from the cache memory when a cache hit occurs.



- **Cache Miss:** A cache miss occurs when a computer system tries to retrieve data from the cache memory, but the data is not found in the cache.
- **Miss Latency:** Cache miss latency refers to the time it takes for a computer system to retrieve data from the main memory or disk storage when a cache miss occurs.



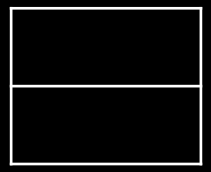


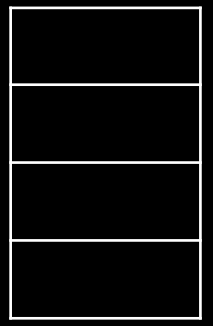
10^3	1 Thousand
10^6	1 Million
10^9	1 Billion
10^{12}	1 Trillion

10^3	1 kilo
10^6	1 Mega
10^9	1 Giga
10^{12}	1 Tera
10^{15}	1 Peta

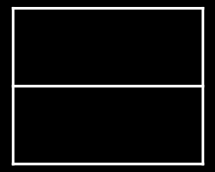
2^{10}	1 kilo
2^{20}	1 Mega
2^{30}	1 Giga
2^{40}	1 Tera
2^{50}	1 Peta

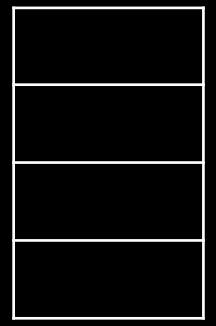
Address Length in bits	n
No of Locations	2^n





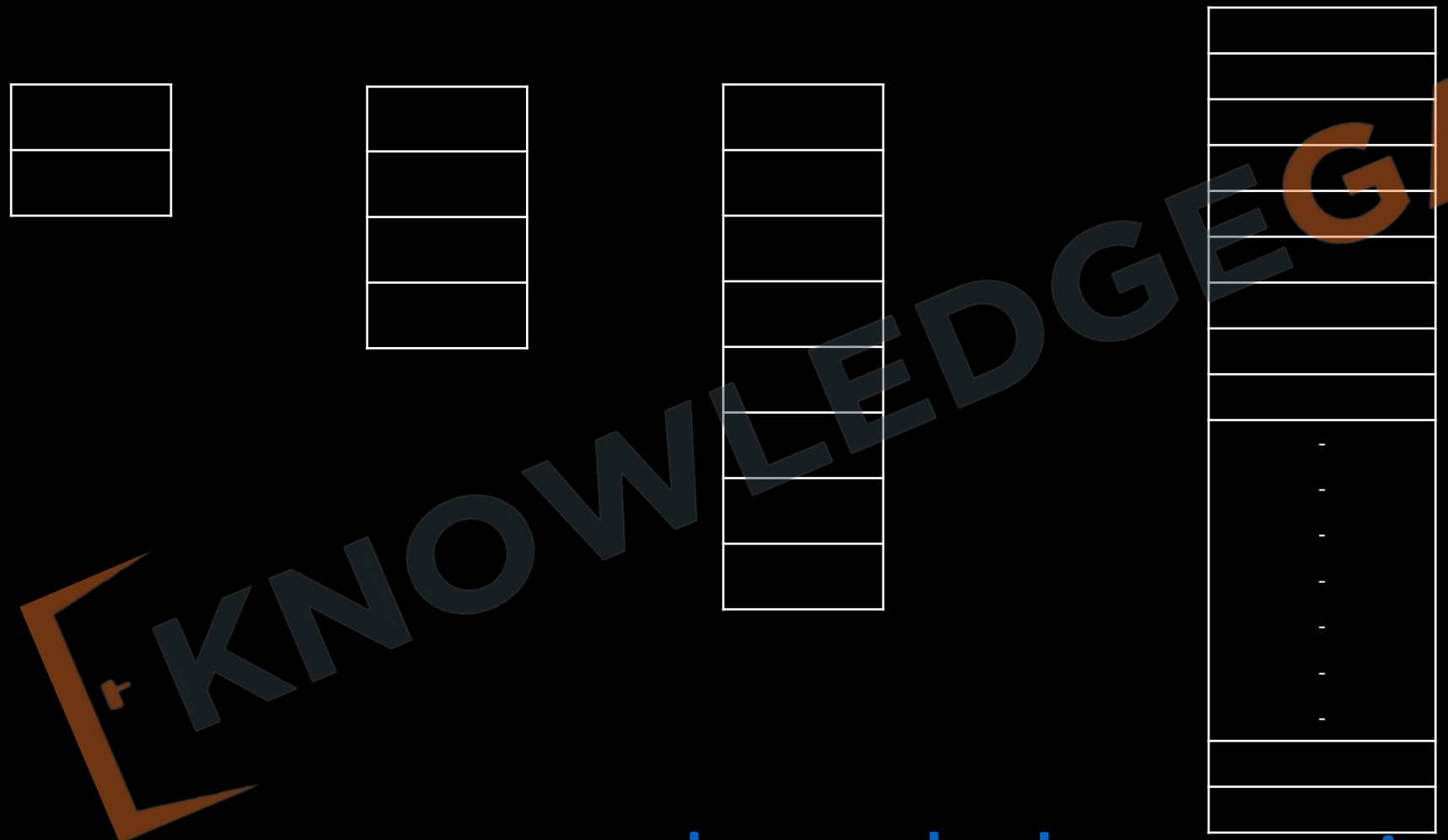
Address Length in bits	n
No of Locations	2^n





Memory Size = Number of Location * Size of each Location

Address Length in bits	Upper Bound($Log_2 n$)
No of Locations	n

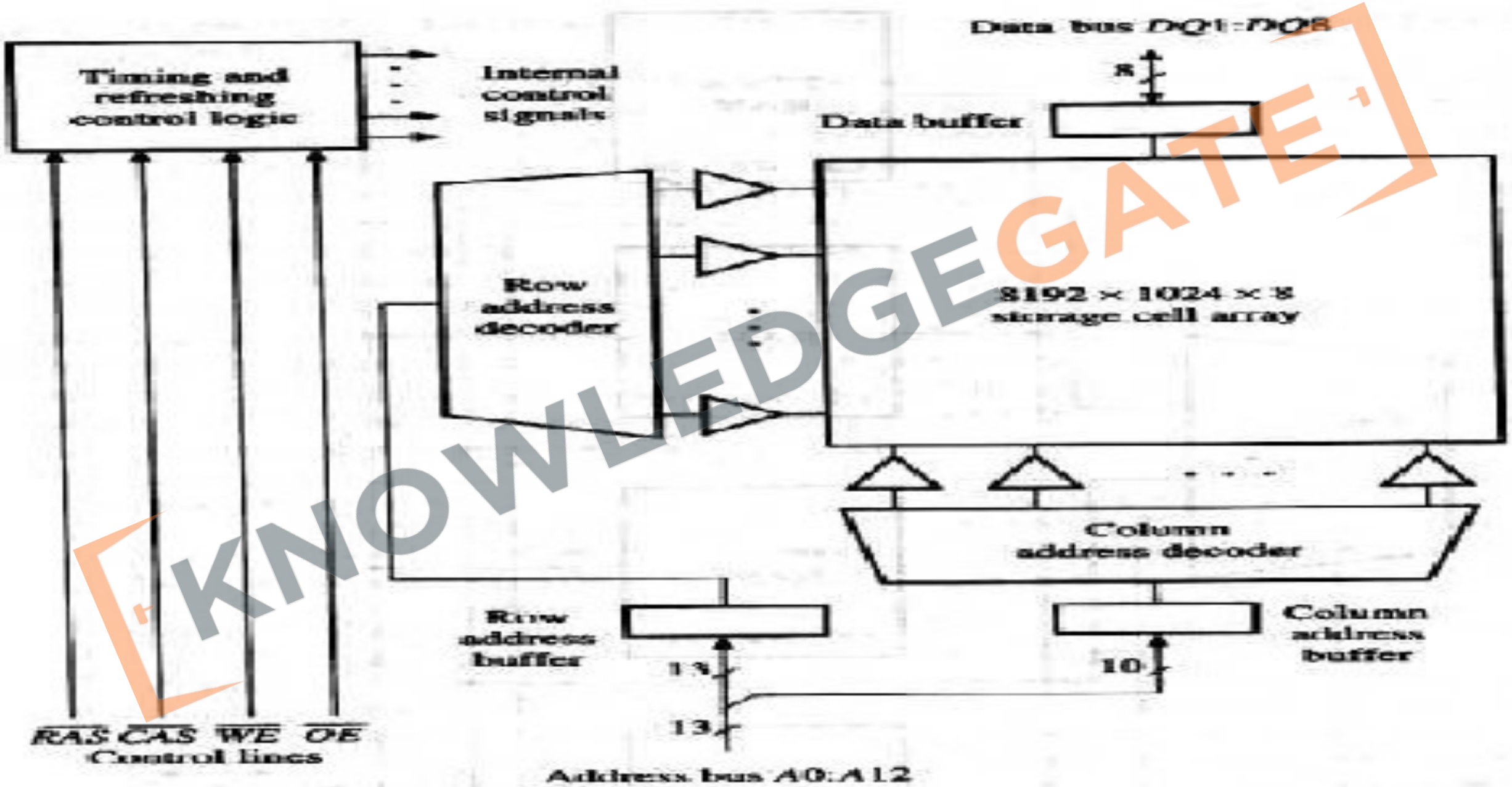


Semiconductor RAM

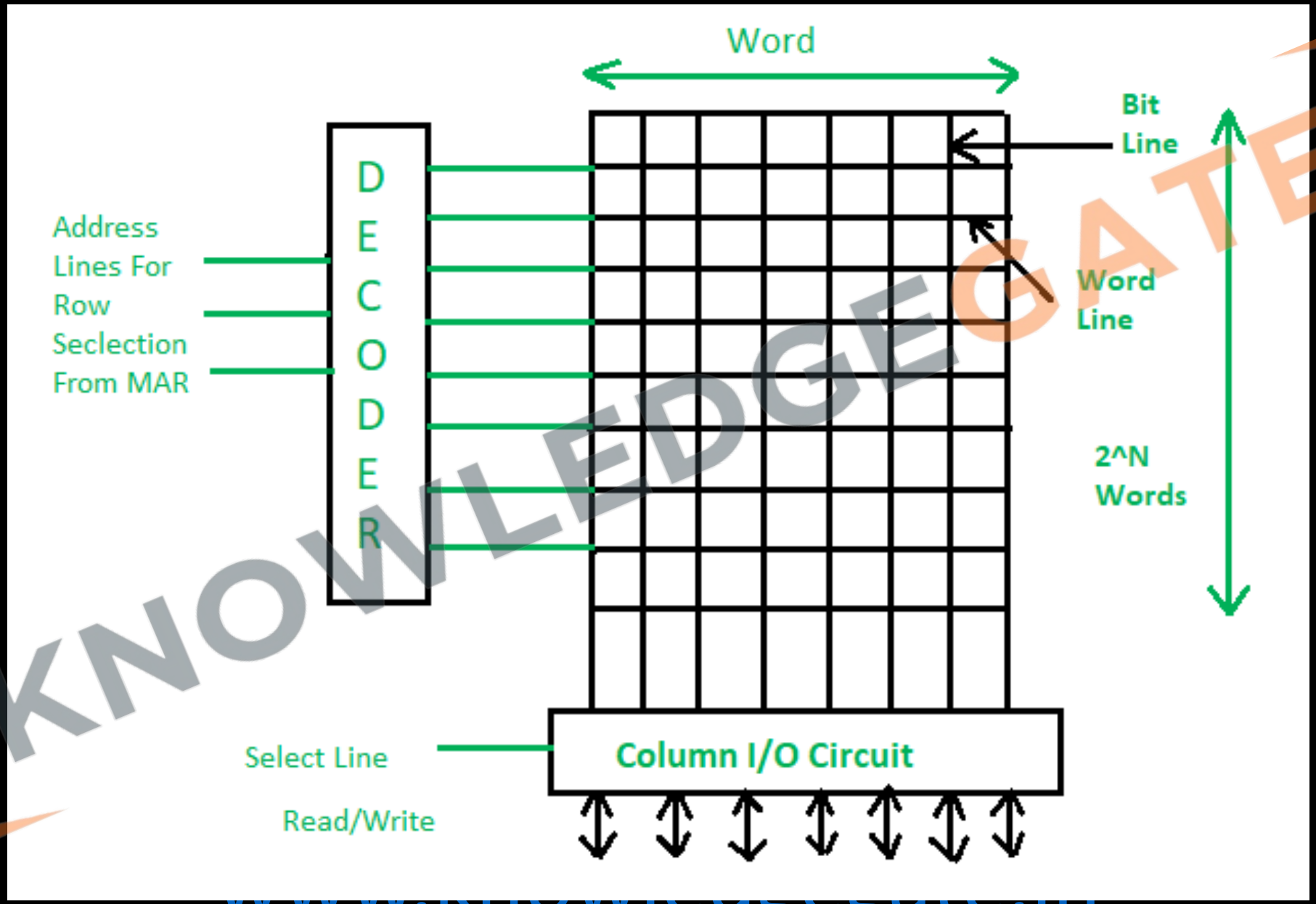
- Semiconductor RAM (Random Access Memory) is a type of computer memory that allows data to be read and written in almost the same amount of time irrespective of the physical location of data inside the memory. It uses semiconductor technology to store data.

Feature	Dynamic RAM (DRAM)	Static RAM (SRAM)
Storage	Uses capacitors	Uses bistable latching
Speed	Slower	Faster
Refresh	Requires frequent refresh	No refresh needed
Power	Lower power consumption	Higher power consumption
Cost	Cheaper	More expensive
Usage	Main system memory	Cache memory

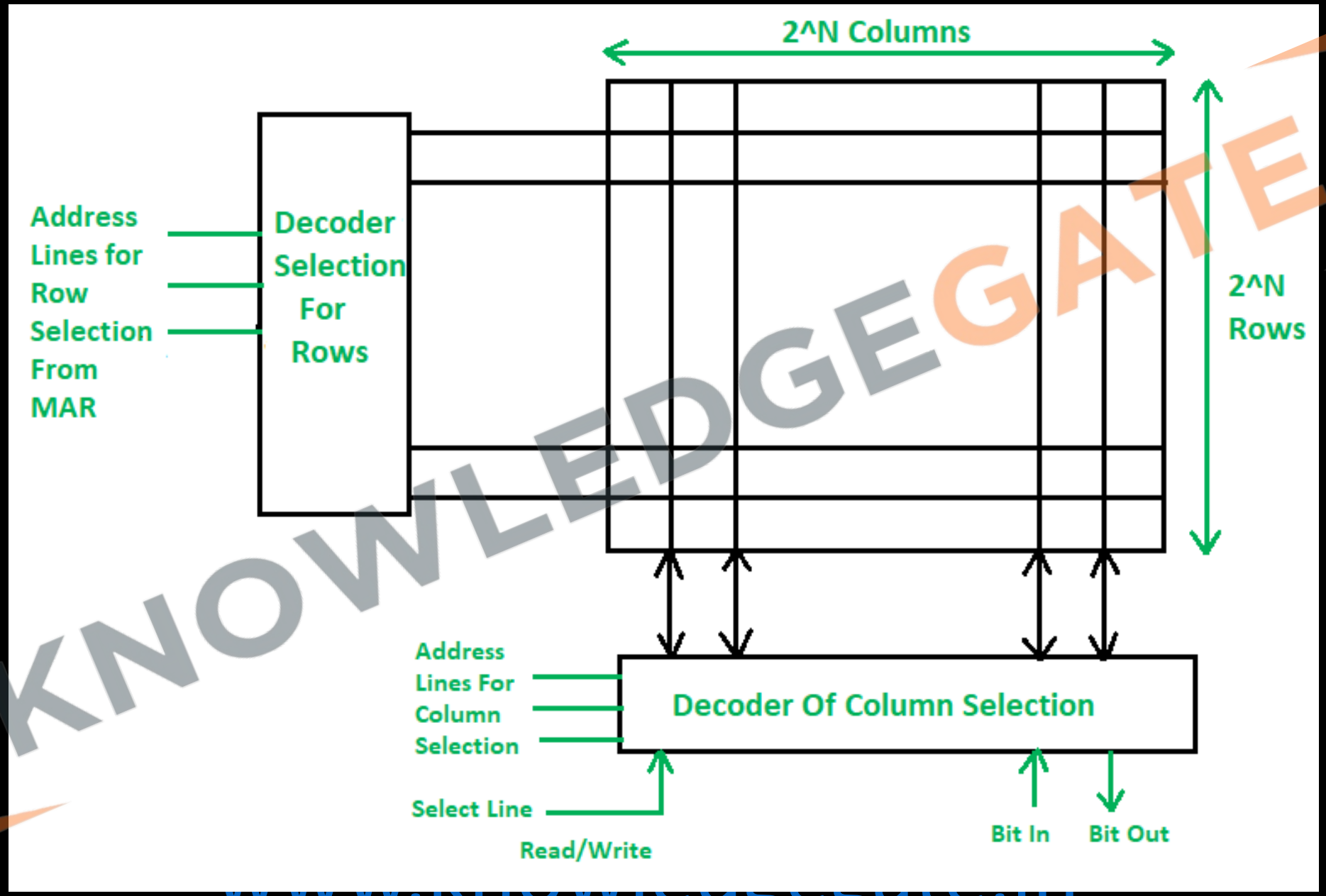
Commercial DRAM Chip



2D Organization



2.5 D Organization



ROM (Read-Only Memory)

- Non-volatile memory, retains data even when power is off.
- Used primarily for firmware, which contains the basic instructions for hardware.
- Data written during the manufacturing process; traditionally, it cannot be modified (hence "read-only").

Types of ROM:

1.MROM (Masked ROM):

1. Programmed during the manufacturing process.
2. Cannot be reprogrammed by the user.
3. Cheapest type of ROM.

2.PROM (Programmable Read-Only Memory):

1. Can be programmed by the user only once.
2. Uses a device called a PROM programmer or PROM burner.
3. Once programmed, data cannot be erased.

3.EPROM (Erasable Programmable Read-Only Memory):

1. Can be erased and reprogrammed multiple times.
2. Erased using UV light, hence often has a transparent window on the chip.
3. Takes more time to erase and reprogram compared to EEPROM.

4.EEPROM (Electrically Erasable Programmable Read-Only Memory):

1. Can be erased and reprogrammed using electrical signals.
2. Erasure can be done byte by byte, not all at once, allowing for selective editing.
3. Used in scenarios where data needs frequent updates, like in BIOS chips.

5.Flash Memory (a type of EEPROM):

1. Can erase and reprogram blocks of data, not just byte-by-byte.
2. Faster than regular EEPROM.
3. Commonly used for USB drives, memory cards, and Solid-State Drives (SSDs).

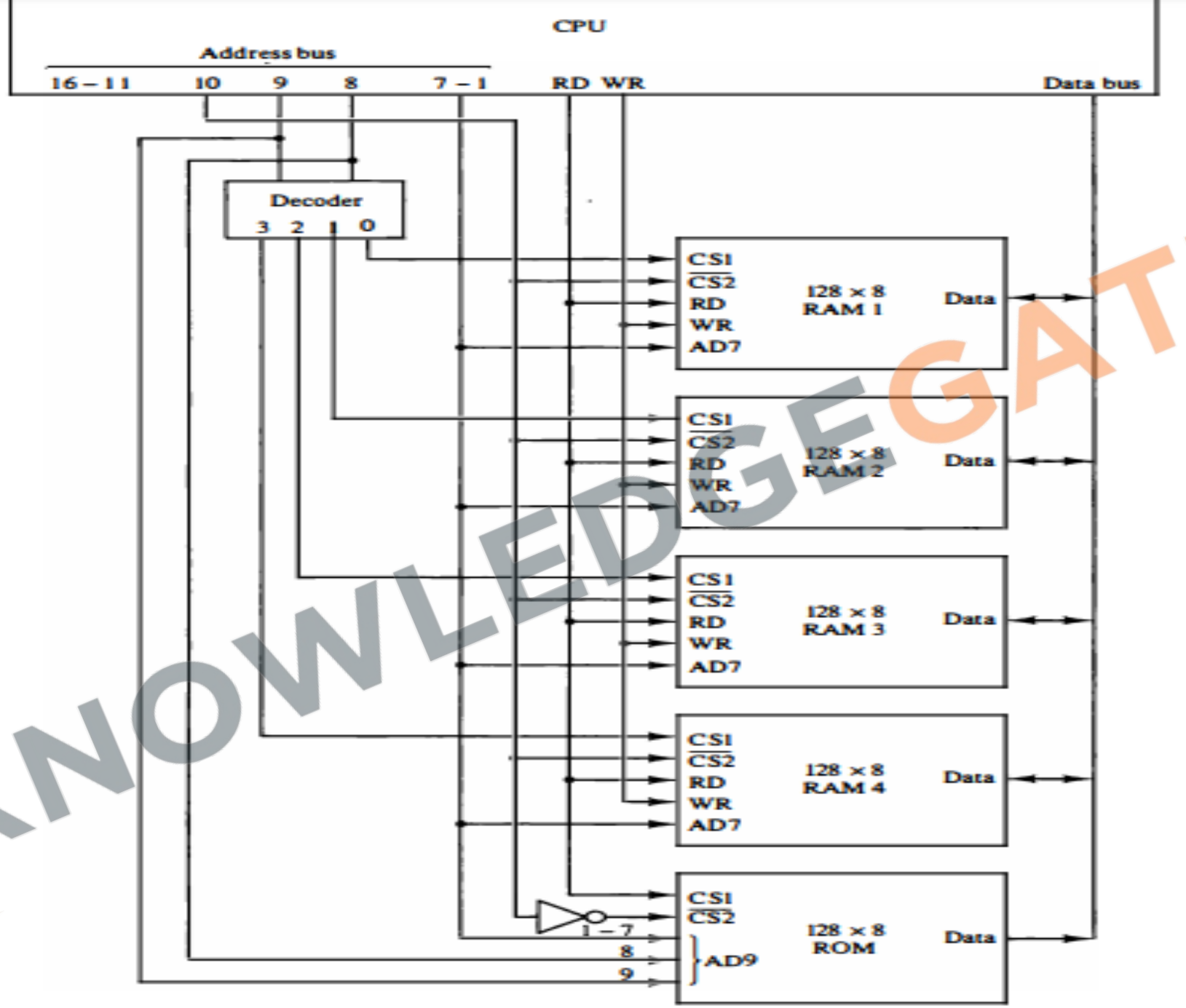
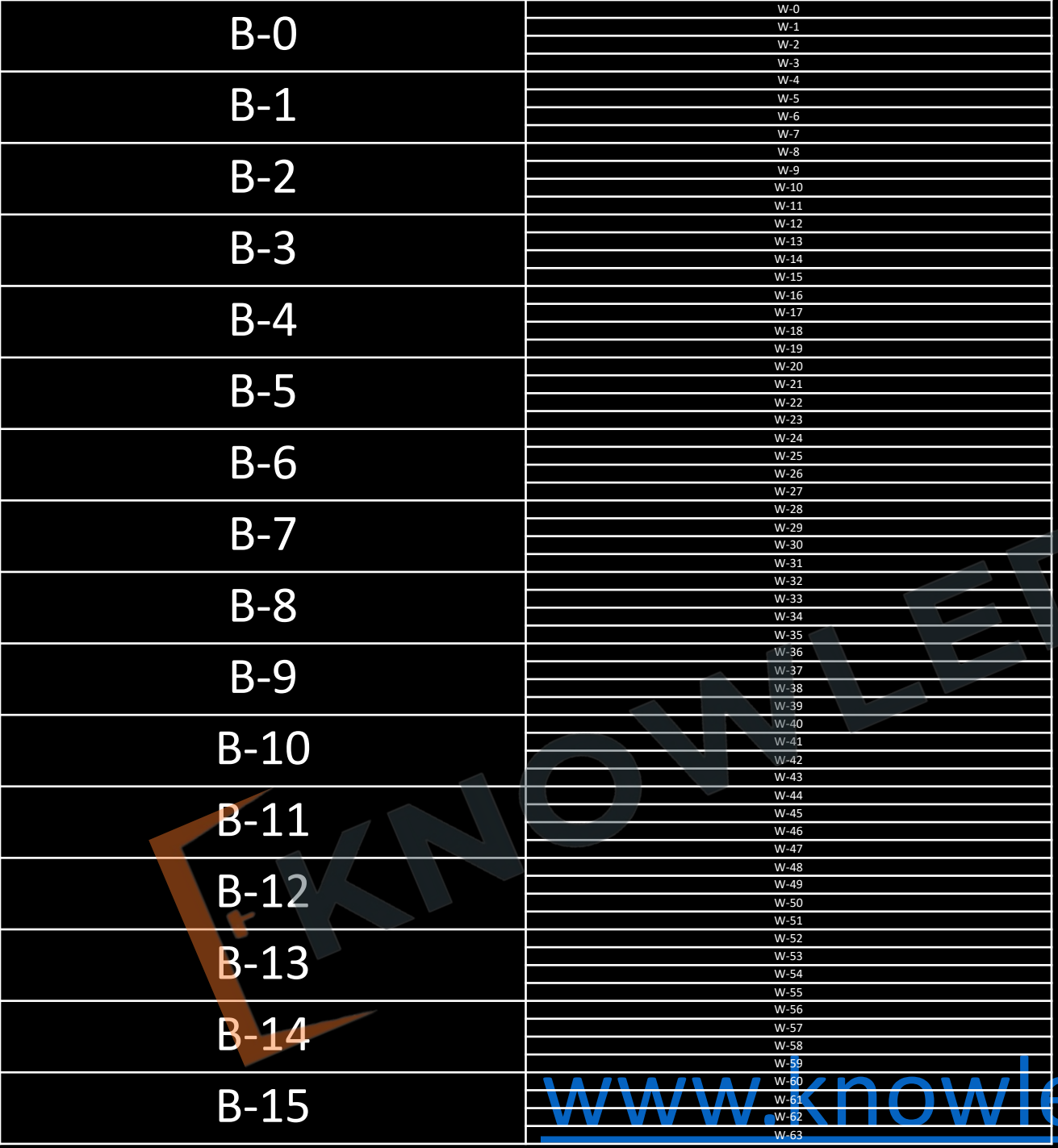


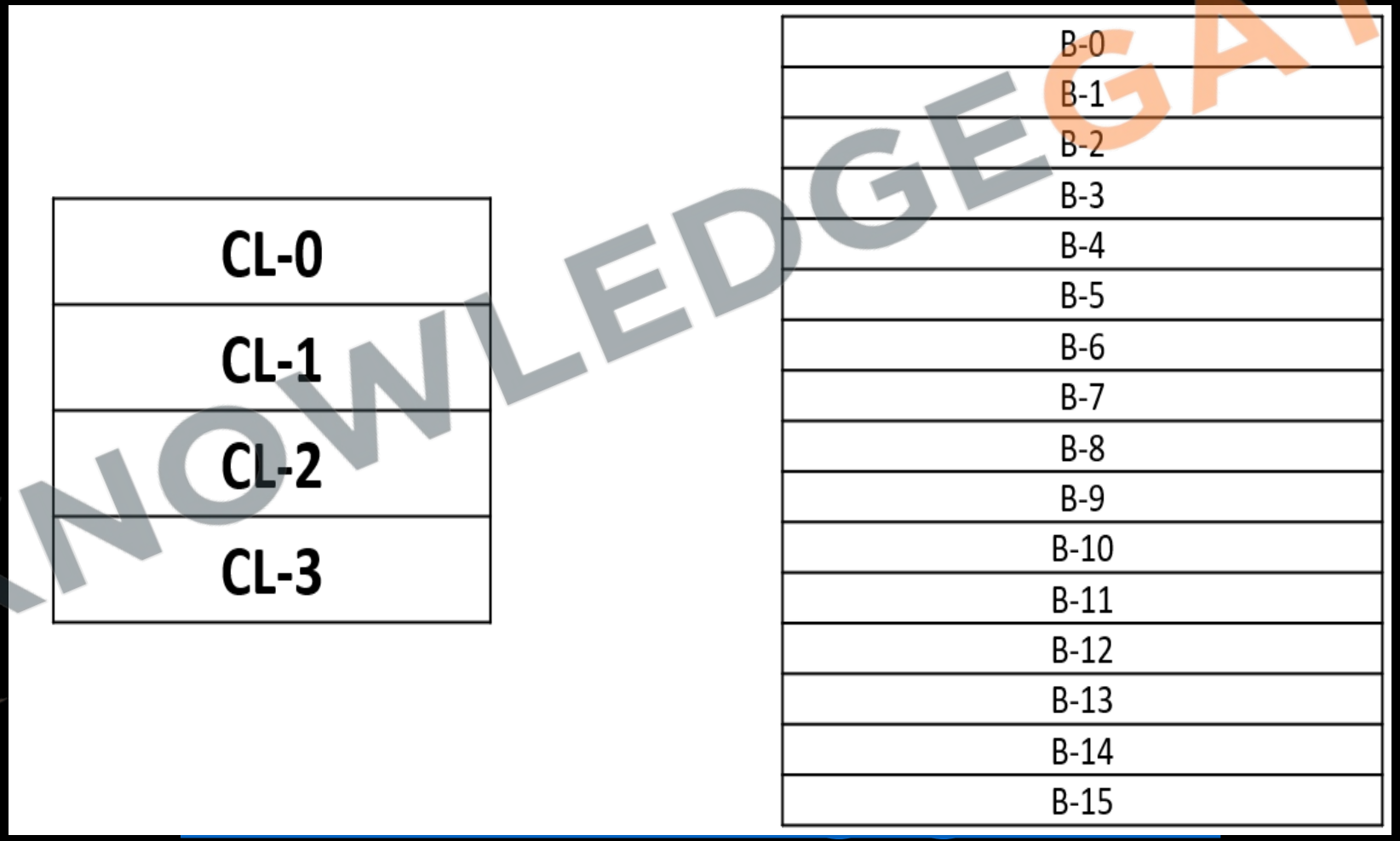
Figure 4 Memory connection to the CPU.



Main Memory				
B-0	W-0	W-1	W-2	W-3
B-1	W-4	W-5	W-6	W-7
B-2	W-8	W-9	W-10	W-11
B-3	W-12	W-13	W-14	W-15
B-4	W-16	W-17	W-18	W-19
B-5	W-20	W-21	W-22	W-23
B-6	W-24	W-25	W-26	W-27
B-7	W-28	W-29	W-30	W-31
B-8	W-32	W-33	W-34	W-35
B-9	W-36	W-37	W-38	W-39
B-10	W-40	W-41	W-42	W-43
B-11	W-44	W-45	W-46	W-47
B-12	W-48	W-49	W-50	W-51
B-13	W-52	W-53	W-54	W-55
B-14	W-56	W-57	W-58	W-59
B-15	W-60	W-61	W-62	W-63

- Cache mapping refers to the process of determining where data should be stored in the cache memory.

- The cache mapping algorithm determines which cache lines are assigned to which main memory blocks.



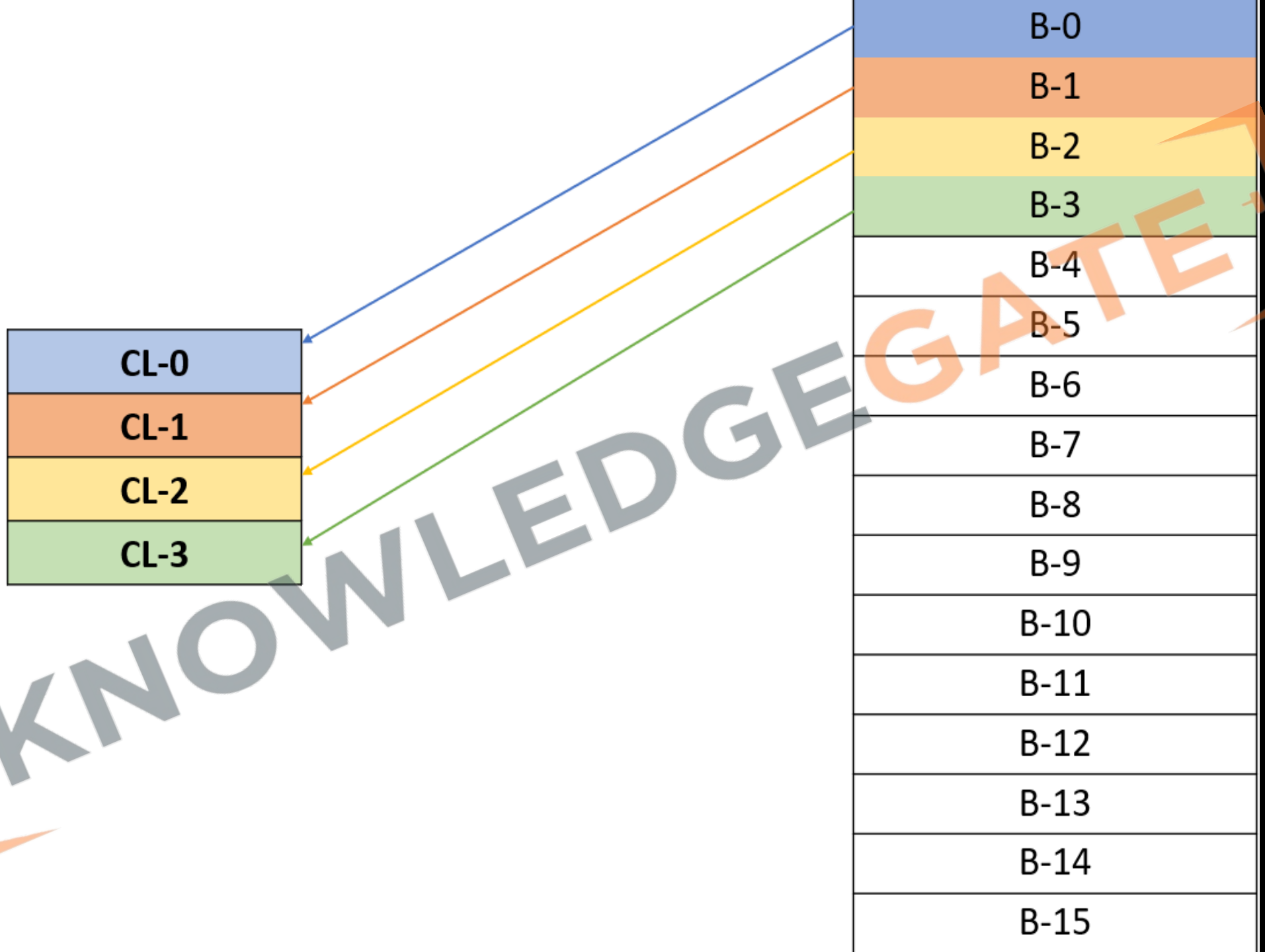
- There are several types of cache mapping algorithms, including:
  - **Direct mapping**: Each main memory block can be stored in only one specific cache line.
  - **Associative mapping**: Any main memory block can be stored in any cache line.
  - **Set-associative mapping**: The cache is divided into sets, each of which contains several cache lines. A main memory block can be stored in any cache line within a set.

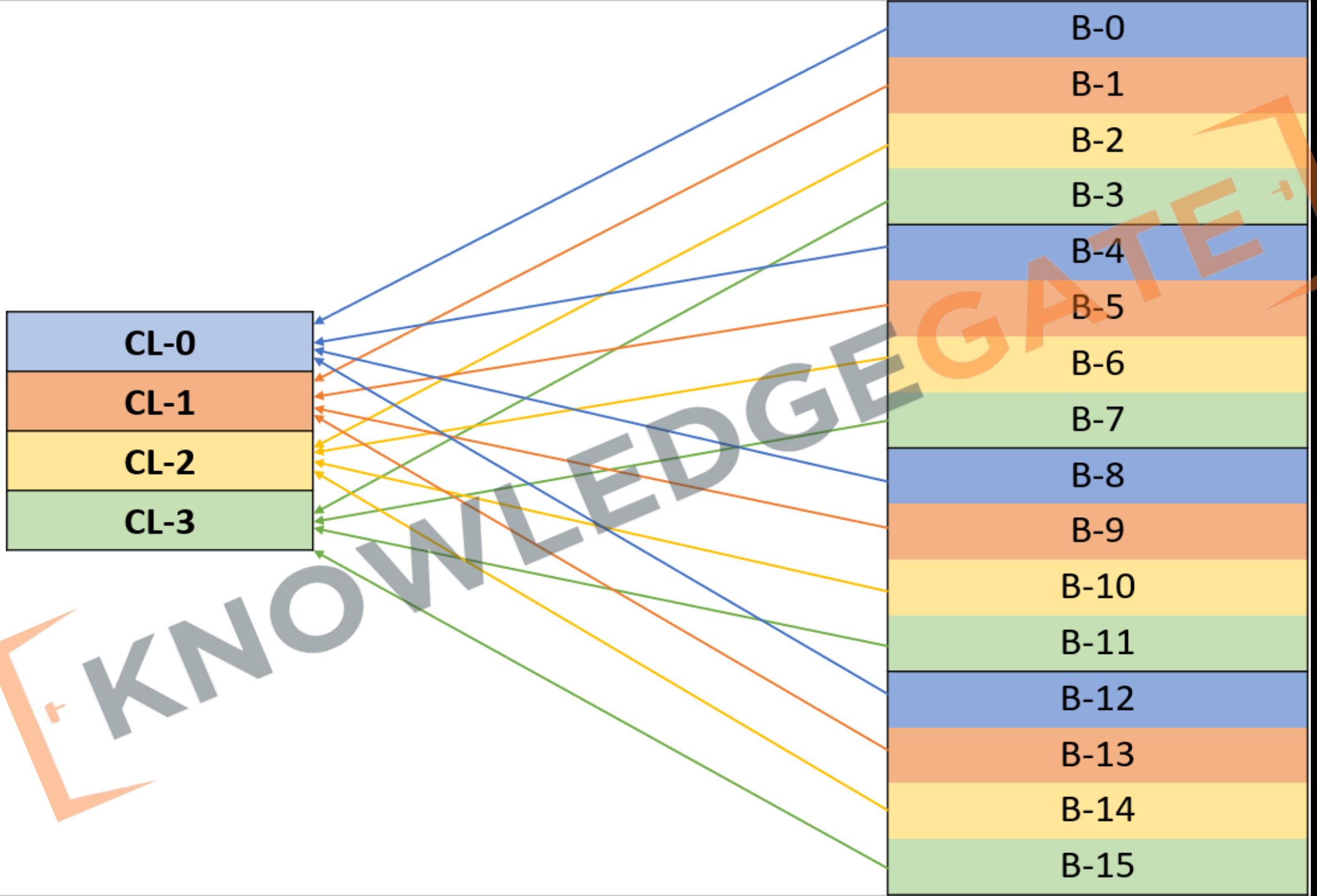
Direct Mapping

- Direct mapping is a type of cache mapping algorithm that maps each main memory block to a specific cache line.

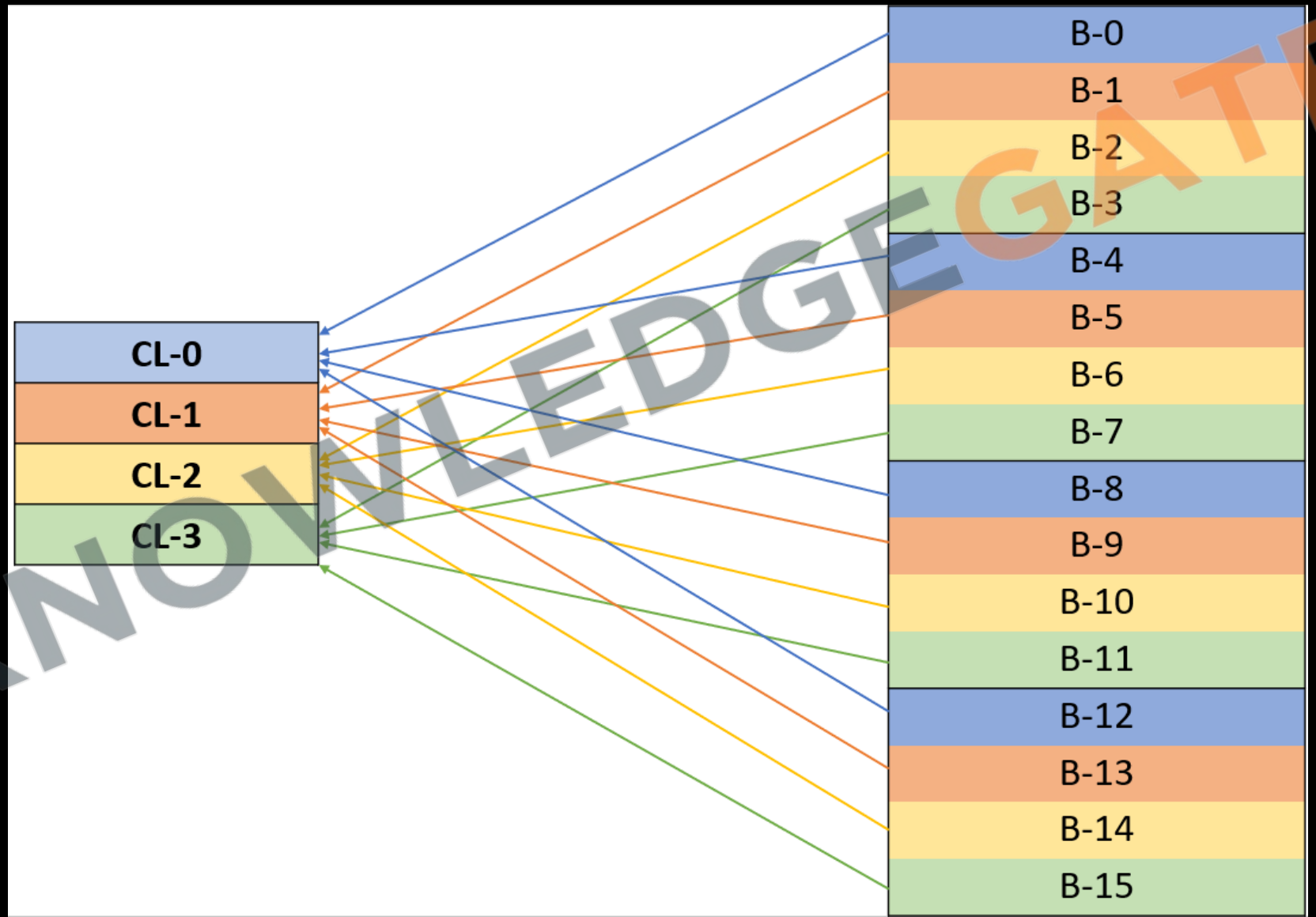
CL-0
CL-1
CL-2
CL-3

B-0
B-1
B-2
B-3
B-4
B-5
B-6
B-7
B-8
B-9
B-10
B-11
B-12
B-13
B-14
B-15





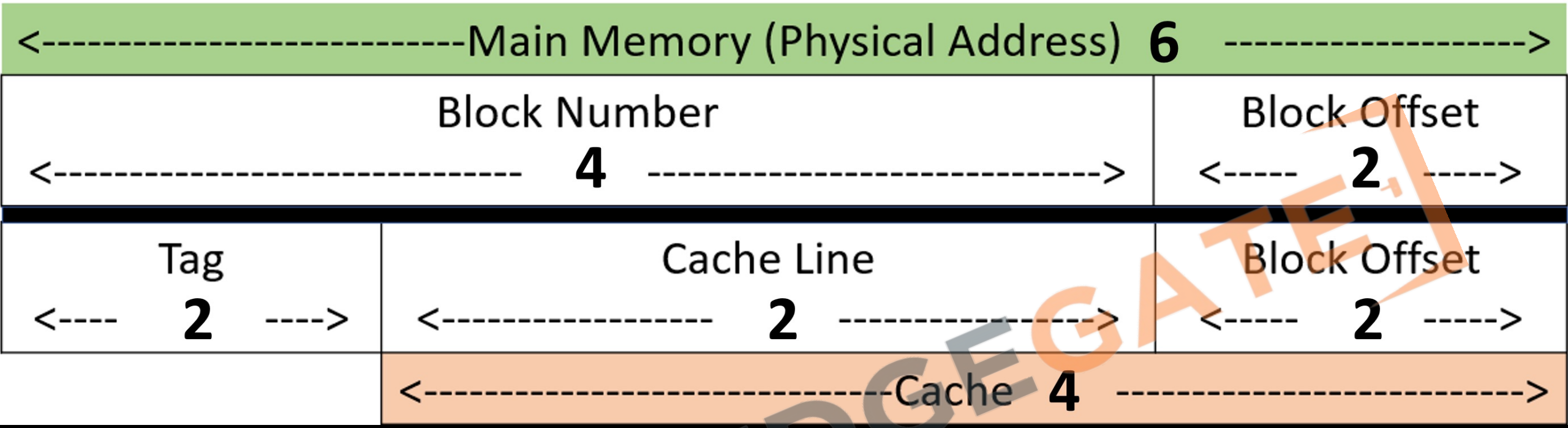
$$\frac{Block\ No}{No\ of\ Cache\ Line} = \text{Remainder is the Address of block in Cache}$$

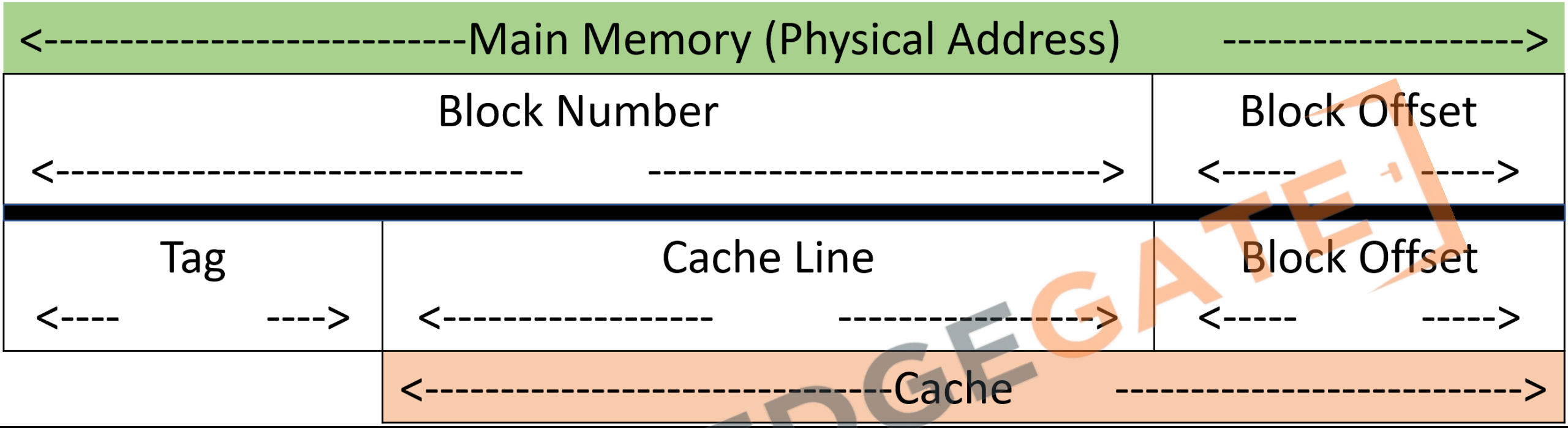


Cache Memory	
CL-0	B-0 / B-4 / B-8 / B-12
CL-1	B-1 / B-5 / B-9 / B-13
CL-2	B-2 / B-6 / B-10 / B-14
CL-3	B-3 / B-7 / B-11 / B-15

Cache Memory		
CL-0	TAG	W-
		W-
		W-
		W-
CL-1	TAG	W-
		W
		W-
		W-
CL-2	TAG	W-
		W-
		W-
		W-
CL-3	TAG	W-
		W-
		W-
		W-

Cache								
CL-0	B-0	W-0	B-4	W-16	B-8	W-32	B-12	W-48
		W-1		W-17		W-33		W-49
		W-2		W-18		W-34		W-50
		W-3		W-19		W-35		W-51
CL-1	B-1	W-4	B-5	W-20	B-9	W-36	B-13	W-52
		W-5		W-21		W-37		W-53
		W-6		W-22		W-38		W-54
		W-7		W-23		W-39		W-55
CL-2	B-2	W-8	B-6	W-24	B-10	W-40	B-14	W-56
		W-9		W-25		W-41		W-57
		W-10		W-26		W-42		W-58
		W-11		W-27		W-43		W-59
CL-3	B-3	W-12	B-7	W-28	B-11	W-44	B-15	W-60
		W-13		W-29		W-45		W-61
		W-14		W-30		W-46		W-62
		W-15		W-31		W-47		W-63





Cache Memory	
CL-0	B-0 / B-4 / B-8 / B-12
CL-1	B-1 / B-5 / B-9 / B-13
CL-2	B-2 / B-6 / B-10 / B-14
CL-3	B-3 / B-7 / B-11 / B-15

Cache Memory		
CL-0	TAG 11	W- W- W- 100111 W-
CL-1	TAG 01	W- W W- W-
CL-2	TAG 01	W- W- W- W-
CL-3	TAG 10	W- W- W- W-

Cache								
CL-0	B-0	W-0	B-4	W-16	B-8	W-32	B-12	W-48
		W-1		W-17		W-33		W-49
		W-2		W-18		W-34		W-50
		W-3		W-19		W-35		W-51
CL-1	B-1	W-4	B-5	W-20	B-9	W-36	B-13	W-52
		W-5		W-21		W-37		W-53
		W-6		W-22		W-38		W-54
		W-7		W-23		W-39		W-55
CL-2	B-2	W-8	B-6	W-24	B-10	W-40	B-14	W-56
		W-9		W-25		W-41		W-57
		W-10		W-26		W-42		W-58
		W-11		W-27		W-43		W-59
CL-3	B-3	W-12	B-7	W-28	B-11	W-44	B-15	W-60
		W-13		W-29		W-45		W-61
		W-14		W-30		W-46		W-62
		W-15		W-31		W-47		W-63

1) W-18
0 1 0 0 1 0

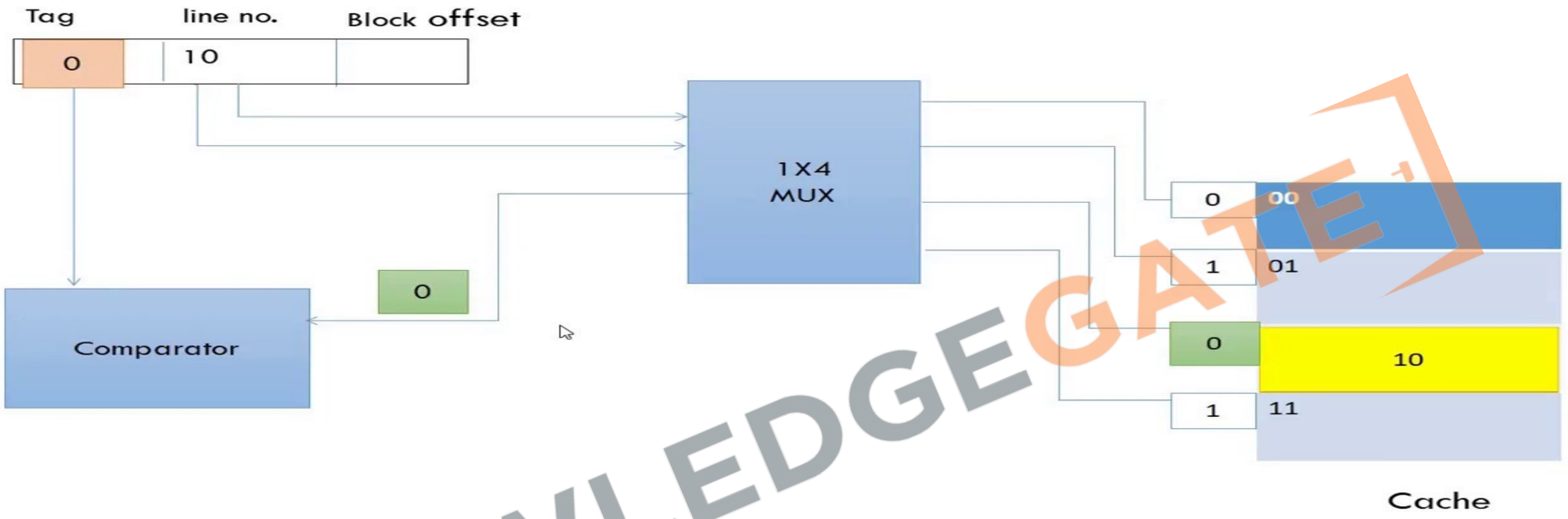
2) W-51
1 1 0 0 1 1

3) W-39
1 0 0 1 1 1

4) W-22
0 1 0 1 1 0

5) W-24
0 1 1 0 0 0

6) W-56
1 1 1 0 0 0

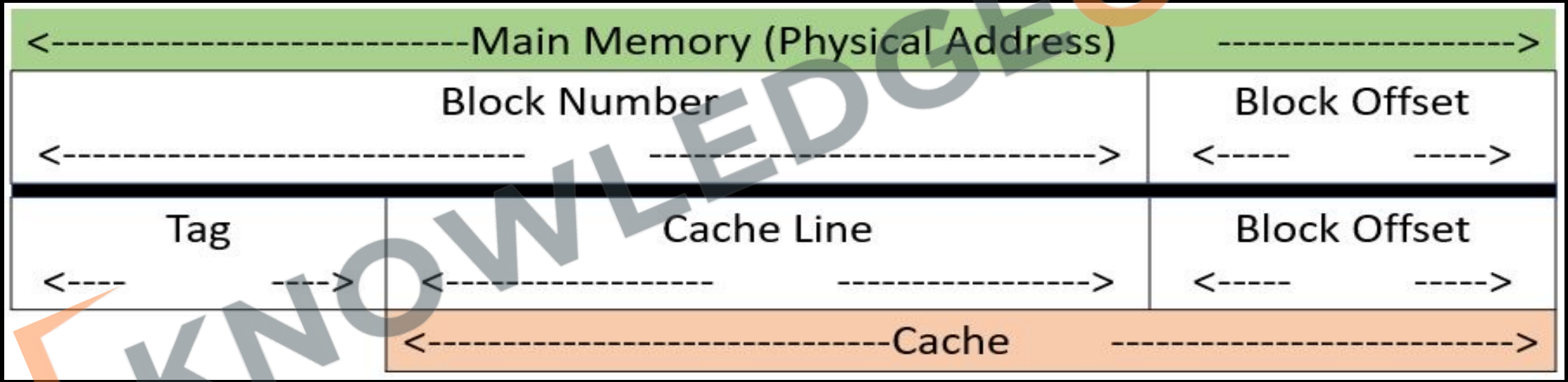


- **Tag directory size = Number of tags x Tag size = Number of lines in cache x Number of bits in tag**

- The main advantage of direct mapping is its simplicity. Since each main memory block maps to only one specific cache line, there is no need for complicated search algorithms to determine the location of a memory block in the cache. This makes direct mapping relatively fast and efficient.

- However, the simplicity of direct mapping also has its drawbacks. Since each cache line can store only one main memory block, it is possible for two memory blocks with different addresses to map to the same cache line. This is known as a cache conflict and can lead to cache thrashing, where the cache is constantly replacing one block with another, reducing the cache hit rate.
- Overall, direct mapping is suitable for small cache memories, where the likelihood of cache conflicts is low, and the benefits of simplicity outweigh the drawbacks.

MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size
16 GB	32 MB	4 KB		
128 MB	256 KB	512 B		
32 GB	128 MB	1 KB		
256 MB	16 KB	1 KB		
4 GB	8 MB	2 KB		
512 KB	2 KB	128 B		

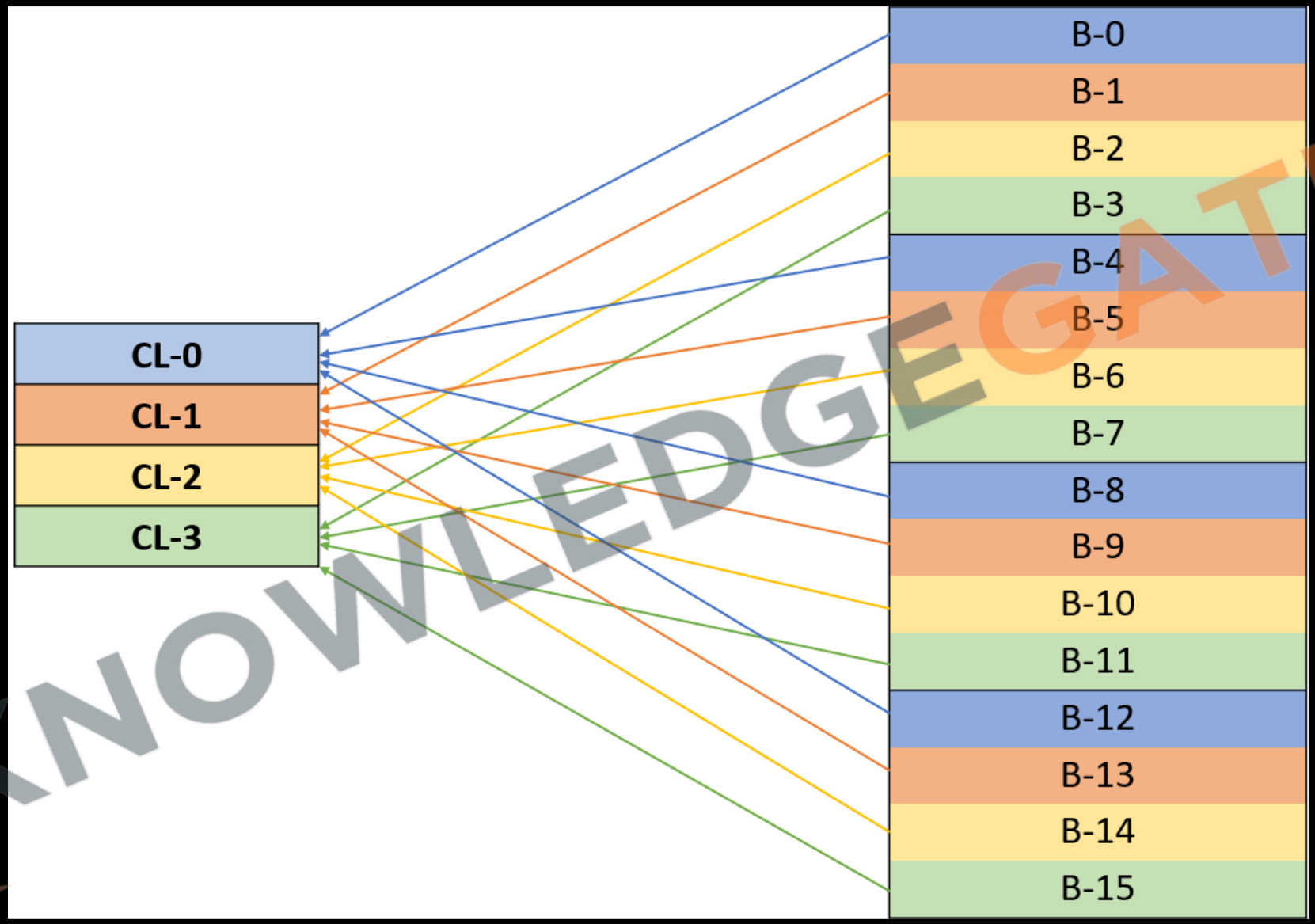


MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size
128 KB	16 KB	256 B	3	
32 GB	32 KB	1 KB	20	
2^{26}B	512 KB	1 KB	7	
16 GB	2^{24} B	4 KB	10	
64 MB	2^{16} B	?	10	
2^{26} B	512 KB	?	7	

MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size
128 KB	16 KB	256 B	3	$3 * 2^6$
32 GB	32 KB	1 KB	20	$20 * 2^5$
2^{26}B	512 KB	1 KB	7	$7*2^9$
16 GB	2^{24} B	4 KB	10	$10*2^{12}$
64 MB	2^{16} B	?	10	?
2^{26} B	512 KB	?	7	?

Associative Mapping

- To overcome the problem of conflict-miss in direct mapping we have Associative Mapping.
- A block of main memory can be mapped to any freely available cache line. This makes fully associative mapping more flexible than direct mapping.
- It is also known as many to many mappings.



Cache Memory		
CL-0	TAG = Block no	W-
		W-
		W-
		W-
CL-1	TAG = Block no	W-
		W
		W-
		W-
CL-2	TAG = Block no	W-
		W-
		W-
		W-
CL-3	TAG = Block no	W-
		W-
		W-
		W-

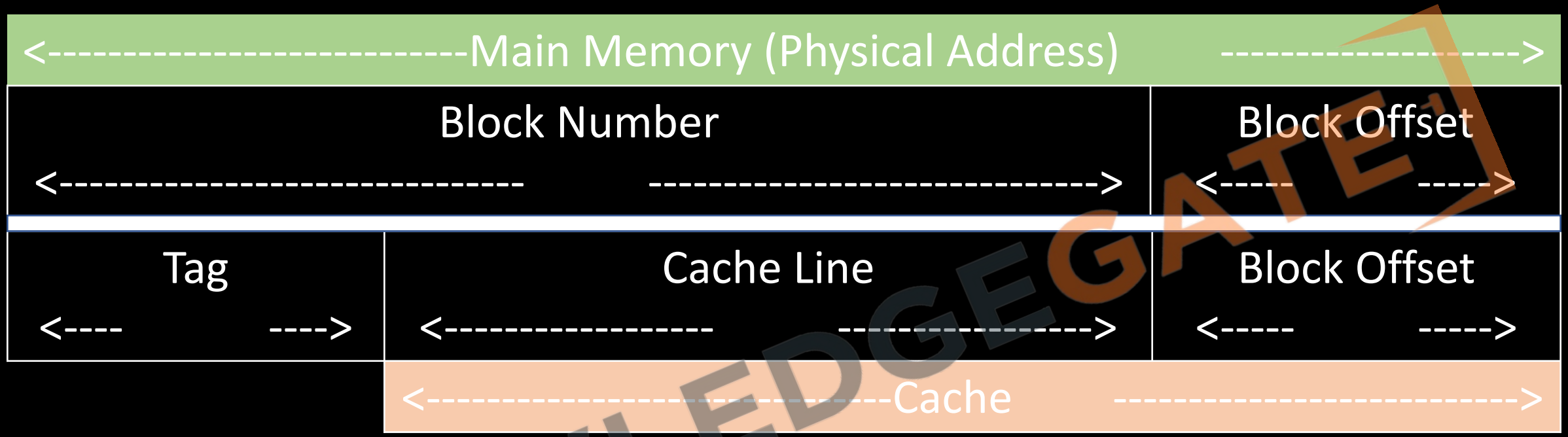
Main Memory				
B-0	W-0	W-1	W-2	W-3
B-1	W-4	W-5	W-6	W-7
B-2	W-8	W-9	W-10	W-11
B-3	W-12	W-13	W-14	W-15
B-4	W-16	W-17	W-18	W-19
B-5	W-20	W-21	W-22	W-23
B-6	W-24	W-25	W-26	W-27
B-7	W-28	W-29	W-30	W-31
B-8	W-32	W-33	W-34	W-35
B-9	W-36	W-37	W-38	W-39
B-10	W-40	W-41	W-42	W-43
B-11	W-44	W-45	W-46	W-47
B-12	W-48	W-49	W-50	W-51
B-13	W-52	W-53	W-54	W-55
B-14	W-56	W-57	W-58	W-59
B-15	W-60	W-61	W-62	W-63

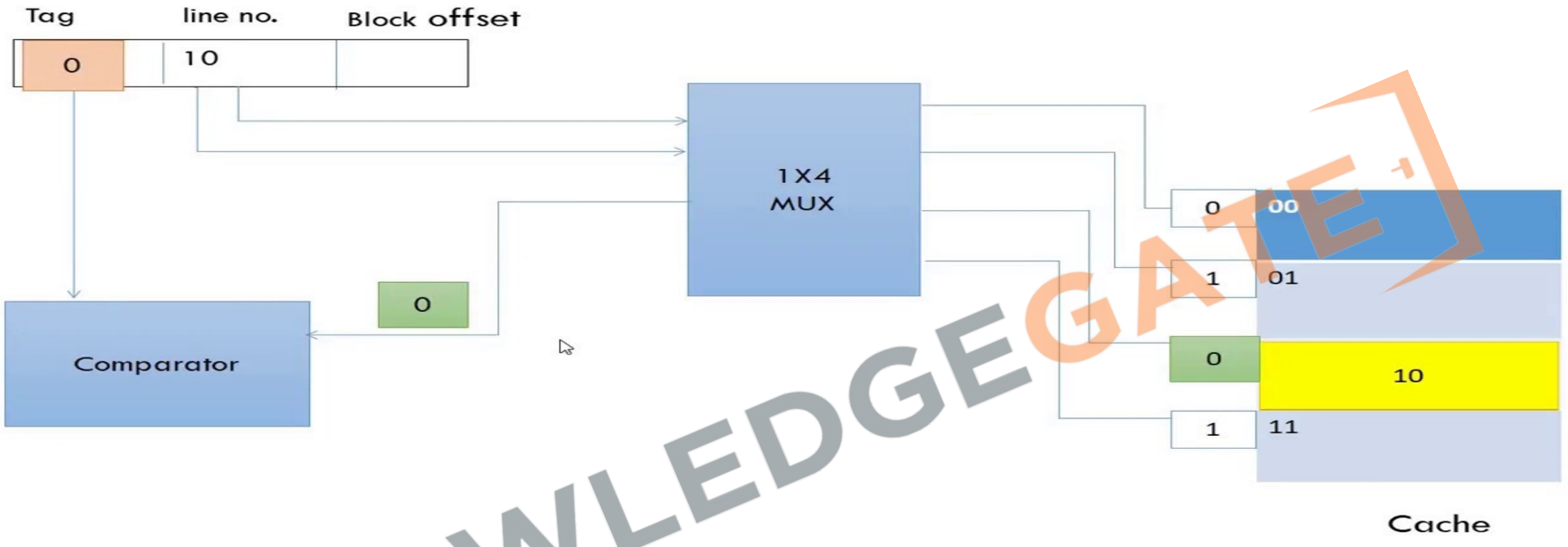
Cache Memory	
CL-0	B-0 / B-4 / B-8 / B-12
CL-1	B-1 / B-5 / B-9 / B-13
CL-2	B-2 / B-6 / B-10 / B-14
CL-3	B-3 / B-7 / B-11 / B-15

Cache Memory		
CL-0	TAG	W-
		W-
		W-
		W-
CL-1	TAG	W-
		W
		W-
		W-
CL-2	TAG	W-
		W-
		W-
		W-
CL-3	TAG	W-
		W-
		W-
		W-

Cache								
CL-0	B-0	W-0	B-4	W-16	B-8	W-32	B-12	W-48
		W-1		W-17		W-33		W-49
		W-2		W-18		W-34		W-50
		W-3		W-19		W-35		W-51
CL-1	B-1	W-4	B-5	W-20	B-9	W-36	B-13	W-52
		W-5		W-21		W-37		W-53
		W-6		W-22		W-38		W-54
		W-7		W-23		W-39		W-55
CL-2	B-2	W-8	B-6	W-24	B-10	W-40	B-14	W-56
		W-9		W-25		W-41		W-57
		W-10		W-26		W-42		W-58
		W-11		W-27		W-43		W-59
CL-3	B-3	W-12	B-7	W-28	B-11	W-44	B-15	W-60
		W-13		W-29		W-45		W-61
		W-14		W-30		W-46		W-62
		W-15		W-31		W-47		W-63

- A replacement algorithm is needed to replace a block if the cache is full.
- In fully associative mapping we only have two fields: Tag/Block Number field and a Block offset field.
- Here the number of bits in tag = number of bits to require to represent block number

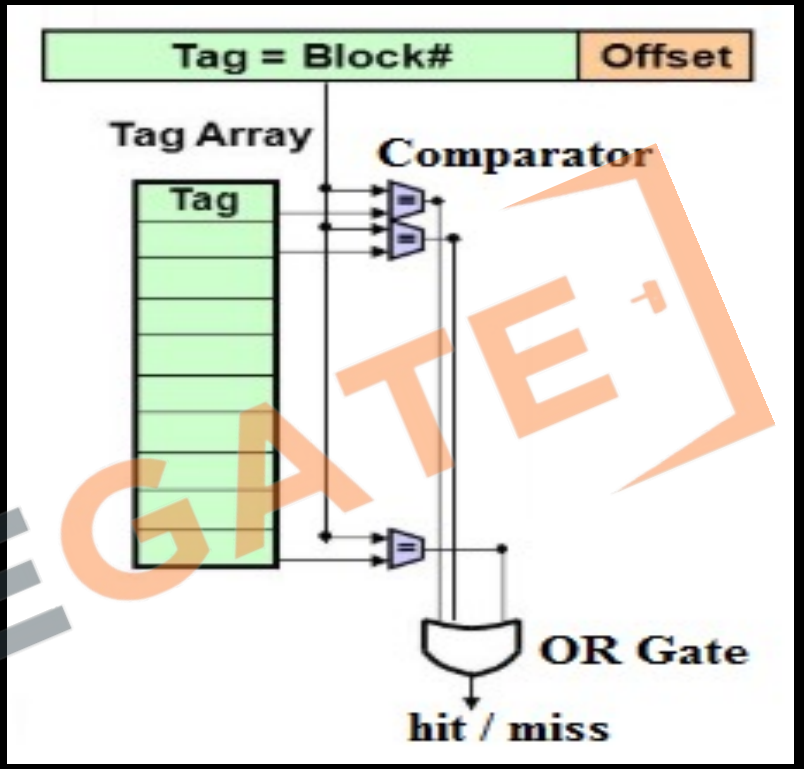


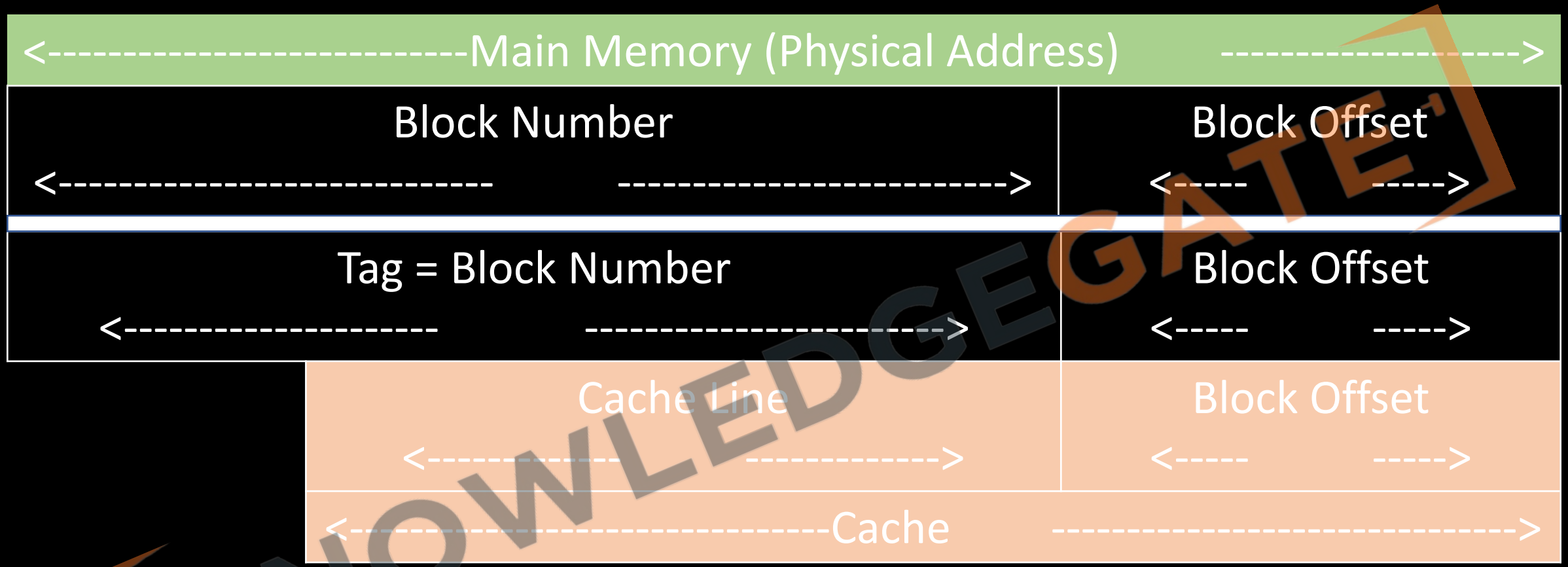


Tag directory size = Number of tags x Tag size = Number of lines in cache x Number of bits in tag

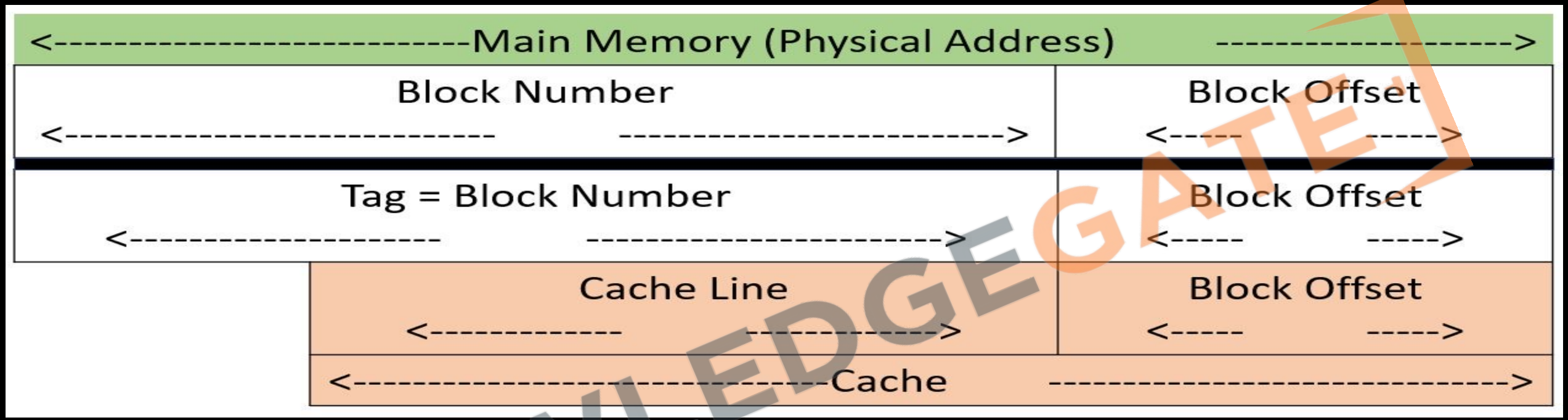
Hardware Architecture

- If we have ‘n’ lines in cache then ‘n’ number of comparators are required.
- Size of comparator = Size of Tag
- If we have ‘n’ bit tag then we require ‘n’ bit comparator.

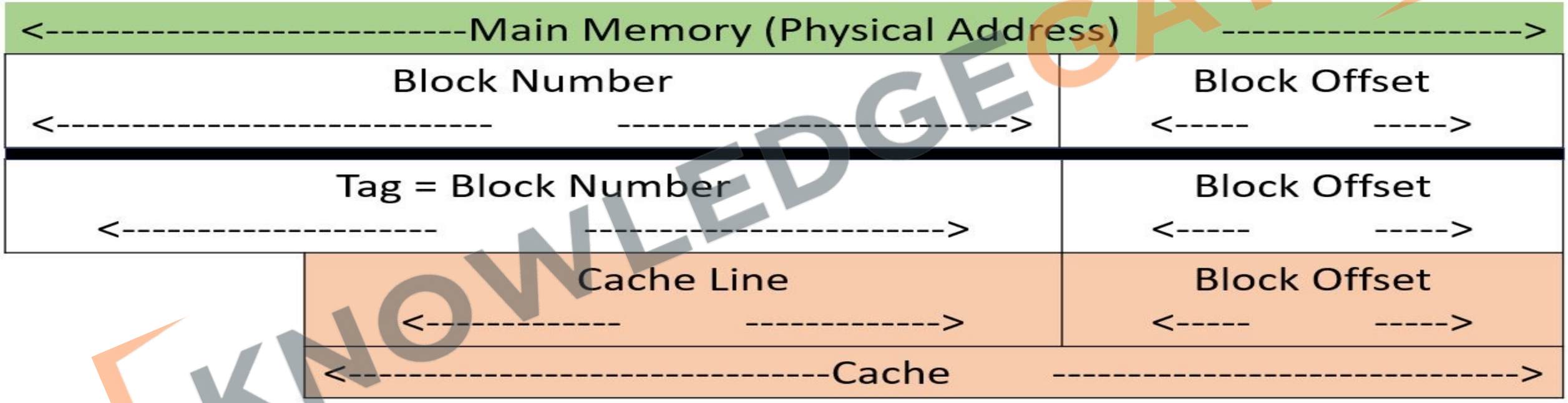




Q Consider a fully associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find out the: Number of bits in tag and Tag directory size?



MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size	Comp
128 KB	16 KB	256 B			
32 GB	32 KB	1 KB			
	512 KB	1 KB	17		
16 GB		4 KB	22		
64MB			10		
	512 KB		7		



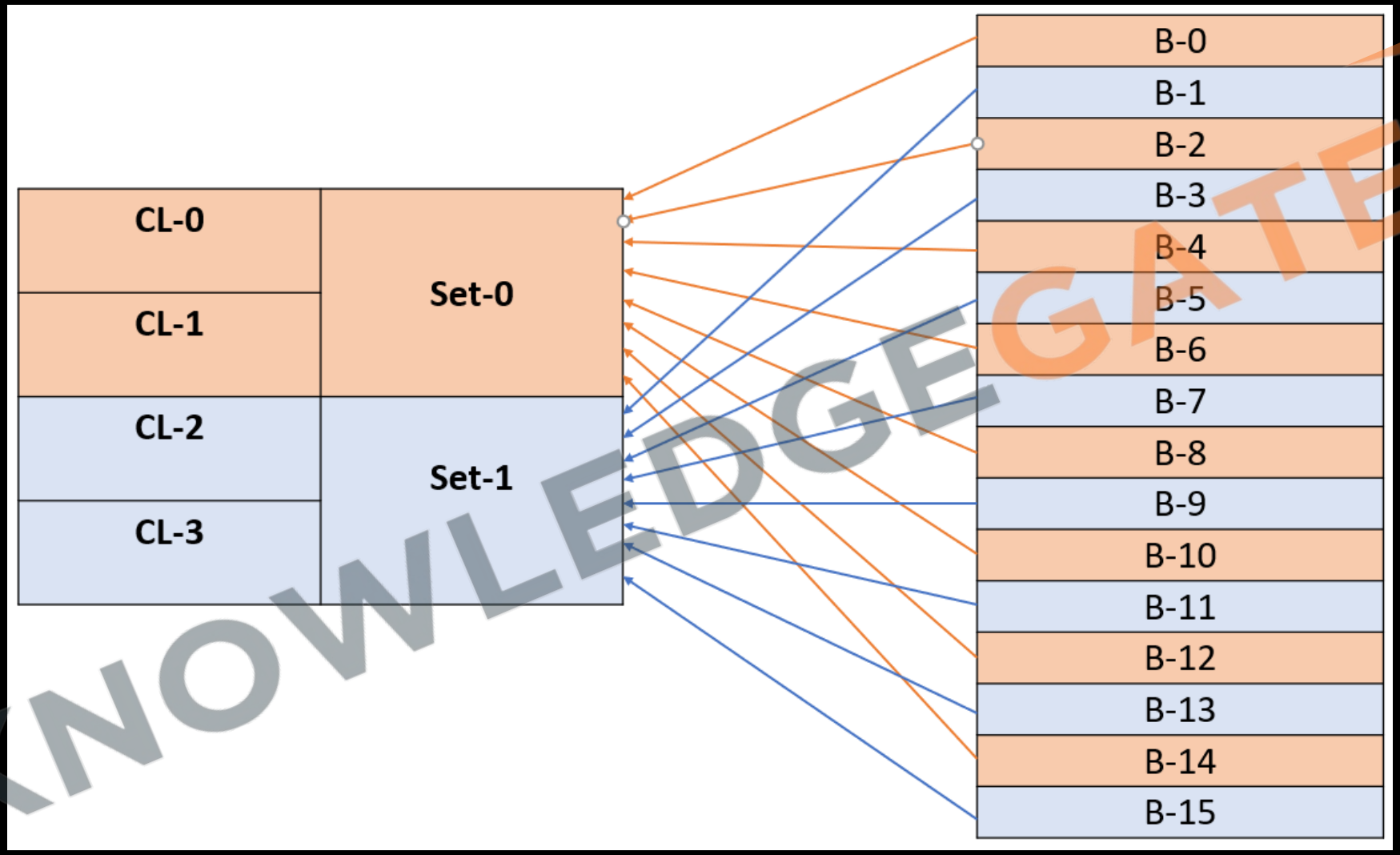
MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size	Comp
128 KB	16 KB	256 B	9	9 * 64	64
32 GB	32 KB	1 KB	25	25 * 32	32
128 MB	512 KB	1 KB	17	512 *17	512
16 GB	?	4 KB	22	?	?
64MB	?	64 KB	10	?	?
?	512 KB		7	?	?

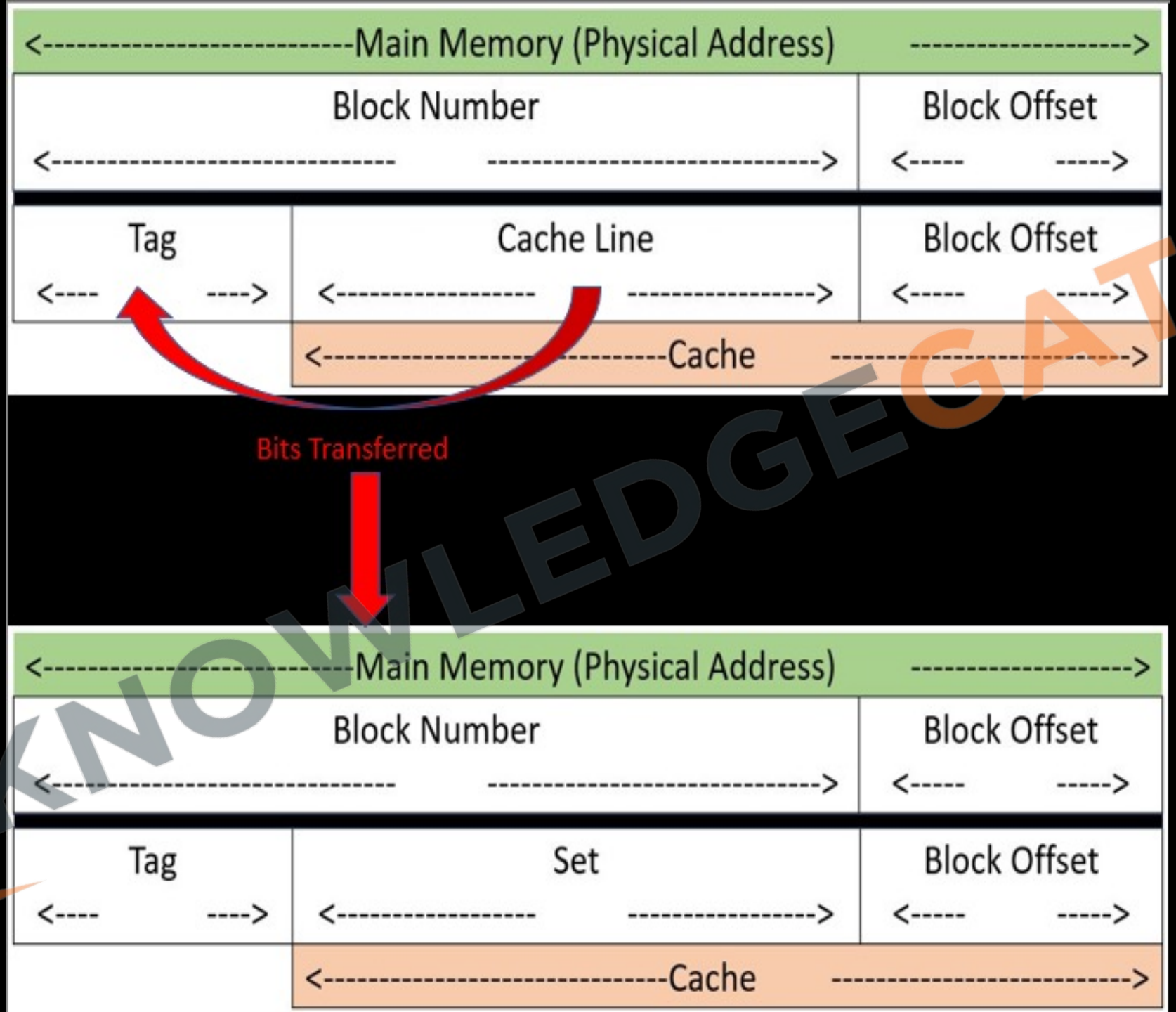
Disadvantage

- Hardware cost is high as compared to direct mapping.
- Tag directory size is more as compared to direct mapping.

Set Associative Mapping

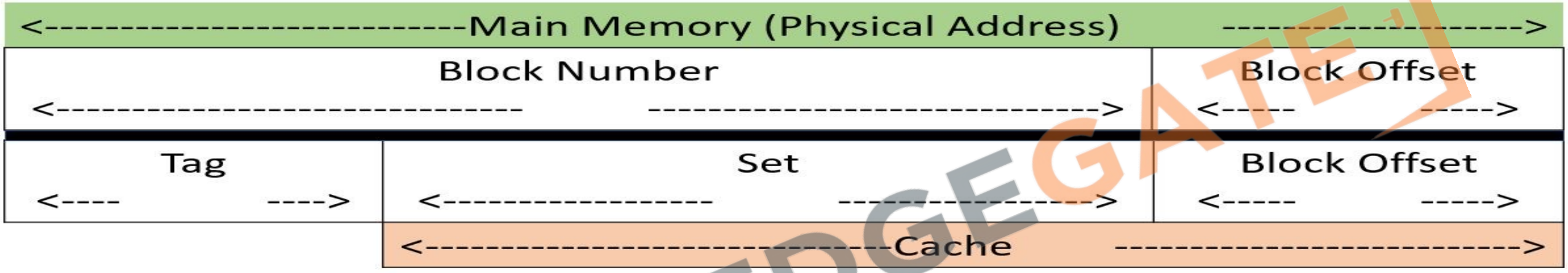
- In k-way set associative mapping, cache lines are grouped into sets where each set contains “k” number of lines.
- A particular block of main memory can map to only one particular set of the cache.
- However, within that set, the memory block can map to any freely available cache line.





Formulas

- Number of Sets = No of Lines in Cache / No of Cache line in a set(k)(k-way set associative)

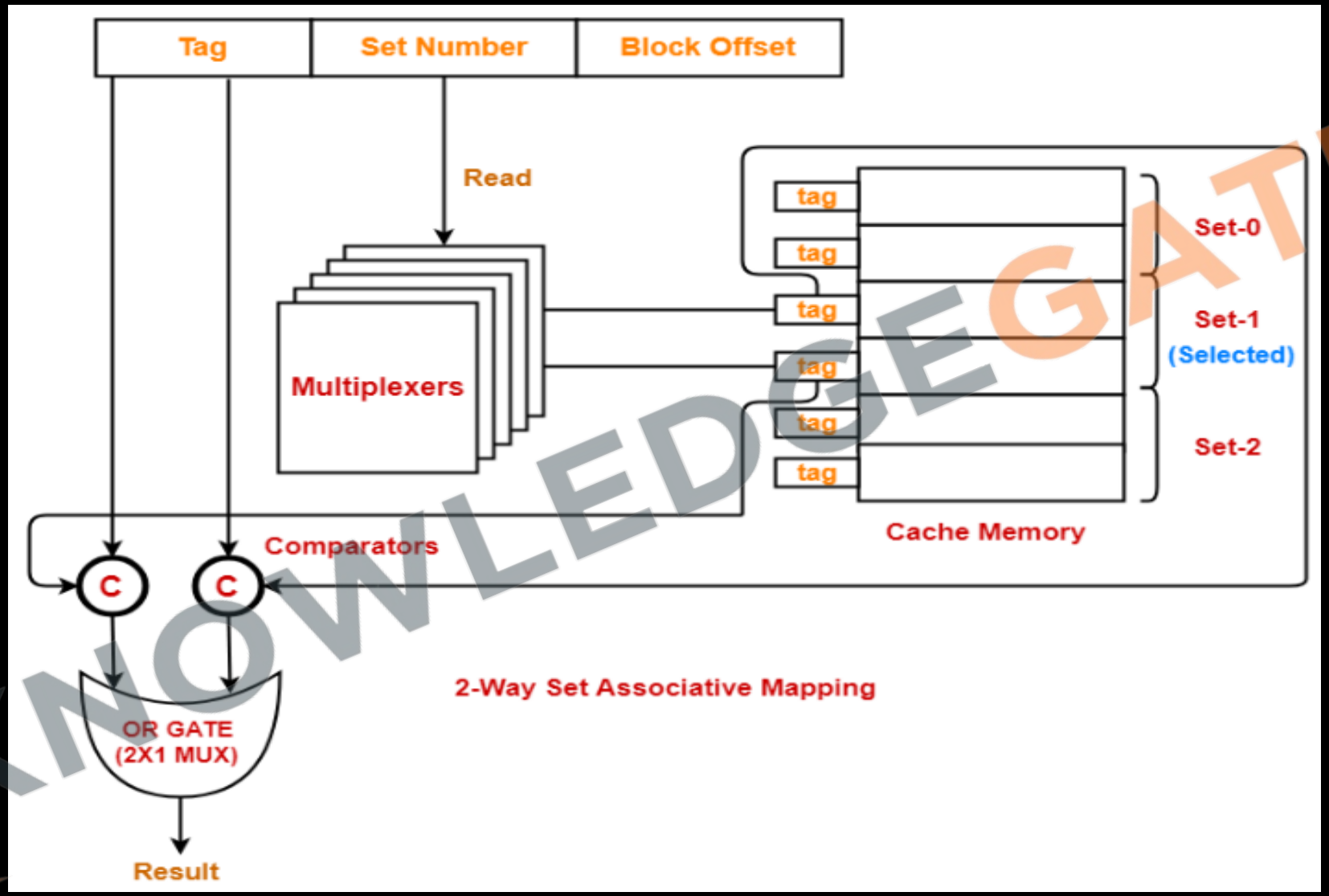


Main Memory				
B-0	W-0	W-1	W-2	W-3
B-1	W-4	W-5	W-6	W-7
B-2	W-8	W-9	W-10	W-11
B-3	W-12	W-13	W-14	W-15
B-4	W-16	W-17	W-18	W-19
B-5	W-20	W-21	W-22	W-23
B-6	W-24	W-25	W-26	W-27
B-7	W-28	W-29	W-30	W-31
B-8	W-32	W-33	W-34	W-35
B-9	W-36	W-37	W-38	W-39
B-10	W-40	W-41	W-42	W-43
B-11	W-44	W-45	W-46	W-47
B-12	W-48	W-49	W-50	W-51
B-13	W-52	W-53	W-54	W-55
B-14	W-56	W-57	W-58	W-59
B-15	W-60	W-61	W-62	W-63

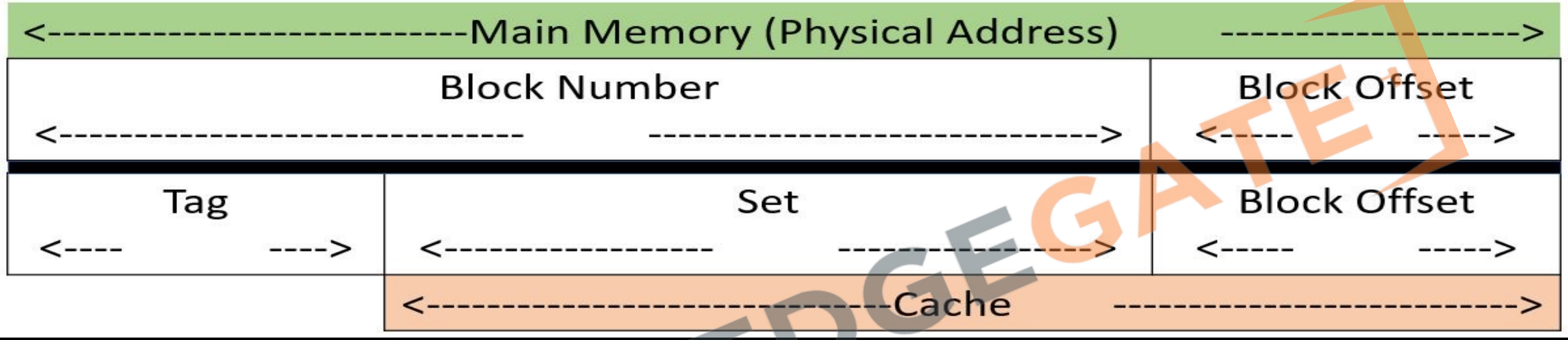
Cache			
Set Number-0	CL-0	TAG = Block no – CL	W-
			W-
			W-
			W-
	CL-1	TAG = Block no – CL	W-
			W
			W-
			W-
Set Number-1	CL-2	TAG = Block no – CL	W-
			W-
			W-
			W-
	CL-3	TAG = Block no - CL	W-
			W-
			W-
			W-

	Cache									
Set Number-0	CL-0	Tag	B-0	W-0	B-2	W-8	B-4	W-16	B-6	W-24
				W-1		W-9		W-17		W-25
				W-2		W-10		W-18		W-26
				W-3		W-11		W-19		W-27
	CL-1	Tag	B-8	W-32	B-10	W-40	B-12	W-48	B-14	W-56
				W-33		W-41		W-49		W-57
				W-34		W-42		W-50		W-58
				W-35		W-43		W-51		W-59
Set Number-1	CL-2	Tag	B-1	W-4	B-3	W-12	B-5	W-20	B-7	W-28
				W-5		W-13		W-21		W-29
				W-6		W-14		W-22		W-30
				W-7		W-15		W-23		W-31
	CL-3	Tag	B-9	W-36	B-11	W-44	B-13	W-52	B-15	W-60
				W-37		W-45		W-53		W-61
				W-38		W-46		W-54		W-62
				W-39		W-47		W-55		W-63

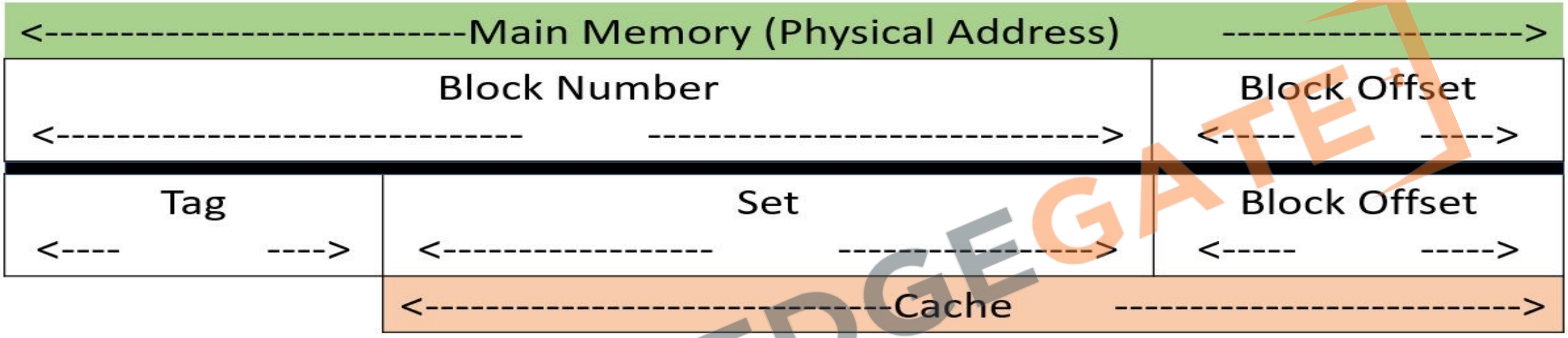
Hardware Architecture



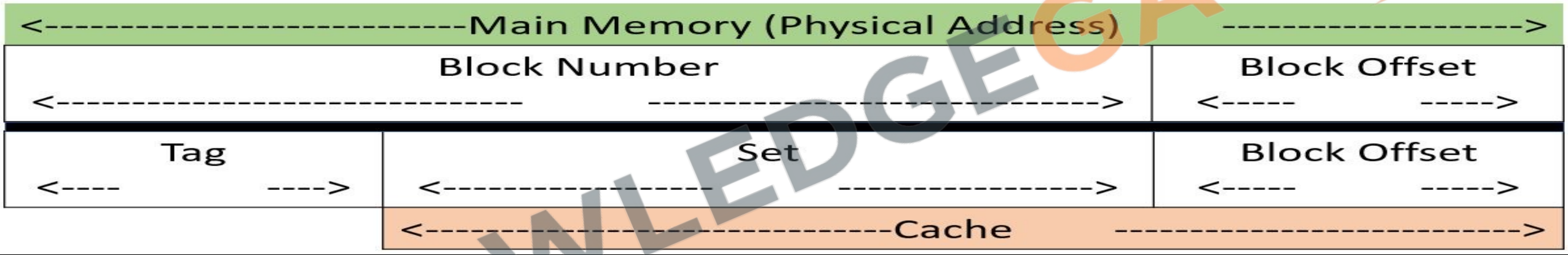
Q Consider the main memory size is of 128 KB, the cache size is of 16 KB, the block size is of 256 B, the set size is 2. Find Tag.



Q Main Memory = 32 GB, Cache Size = 32 KB, Block Size = 1 KB and it is a 4-way set associative cache?



MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size	Set Associative
128 KB	16 KB	256 B			2-way
32 GB	32 KB	1 KB			4-way
	512 KB	1 KB	7		8-way
16 GB		4 KB	10		4-way
64MB			10		4-way
	512 KB		7		8-way



MM Size	Cache Size	Block Size	No of bits in Tag	Tag Directory Size	Set Associative
128 KB	16 KB	256 B	4	$4 * 2^6$	2-way
32 GB	32 KB	1 KB	22	22 * 32	4-way
2^{23} B	512 KB	1 KB	7	$7 * 2^9$	8-way
16 GB	2^{26} B	4 KB	10	$10 * 2^{14}$	4-way
64MB	?	?	10	?	4-way
?	512 KB	?	7	?	8-way

- The choice of cache mapping algorithm depends on several factors, including the size of the cache, the size of the main memory, and the type of data being stored. The most common cache mapping algorithm used in practice is set-associative mapping, as it provides a good balance between the flexibility of associative mapping and the simplicity of direct mapping.
- In general, the cache mapping algorithm plays an important role in determining the performance of the cache memory, as it affects the number of cache hits and misses and the speed at which data can be retrieved from the cache.

Cache Replacement Policies

- In direct mapped cache, the position of each block is predetermined hence no replacement policy exists.
- In fully associative and set associative caches there exists policies.
- When a new block is brought into the cache and all the positions that it may occupy are full, then the controller needs to decide which of the old blocks it can overwrite.

FIFO Policy

- The block which have entered first in the memory will be replaced first.
- This can lead to a problem known as "**Belady's Anomaly**", it states that if we increase the number of lines in cache memory the cache miss will increase.

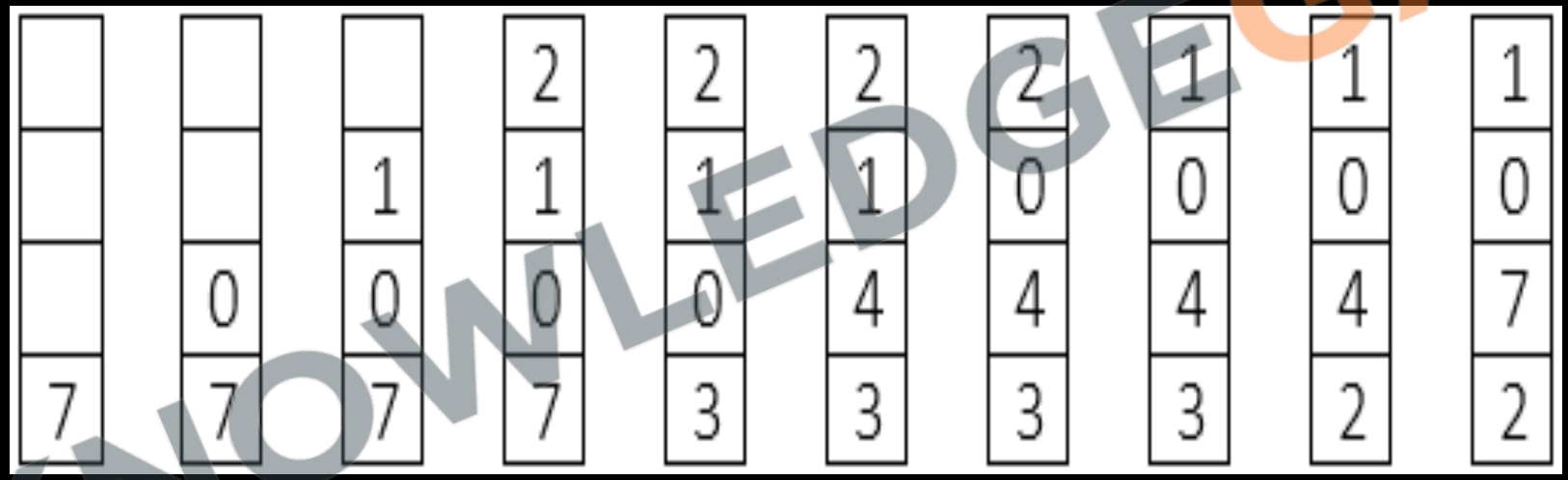
Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

FIFO Policy

- The block which have entered first in the memory will be replaced first.
- This can lead to a problem known as "**Belady's Anomaly**", it states that if we increase the number of lines in cache memory the cache miss will increase.

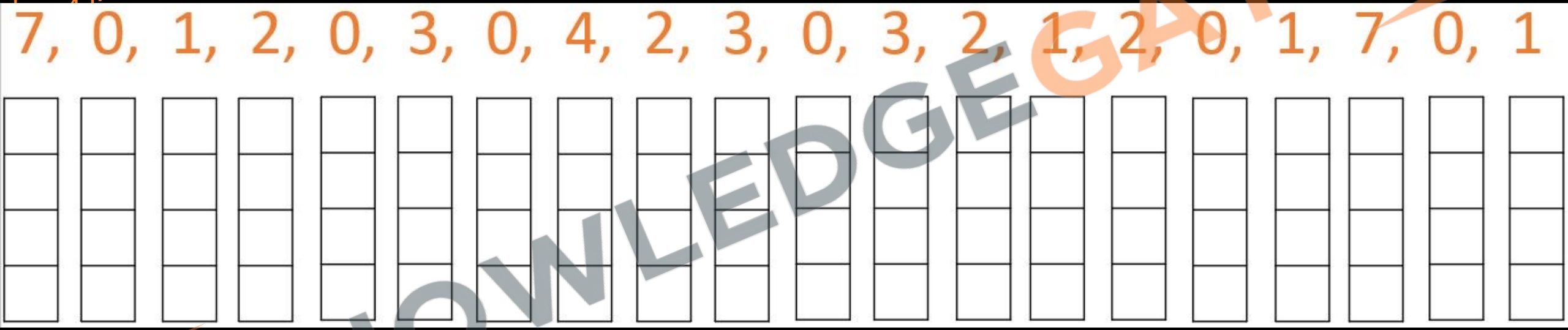
Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.



Optimal Algorithm

- The page which will not be used for the longest period of time in future references will be replaced first.
- The optimal algorithm will provide the best performance but it is difficult to implement as it requires the future knowledge of pages which is not possible.
- It is used as a benchmark for cache replacement algorithms.

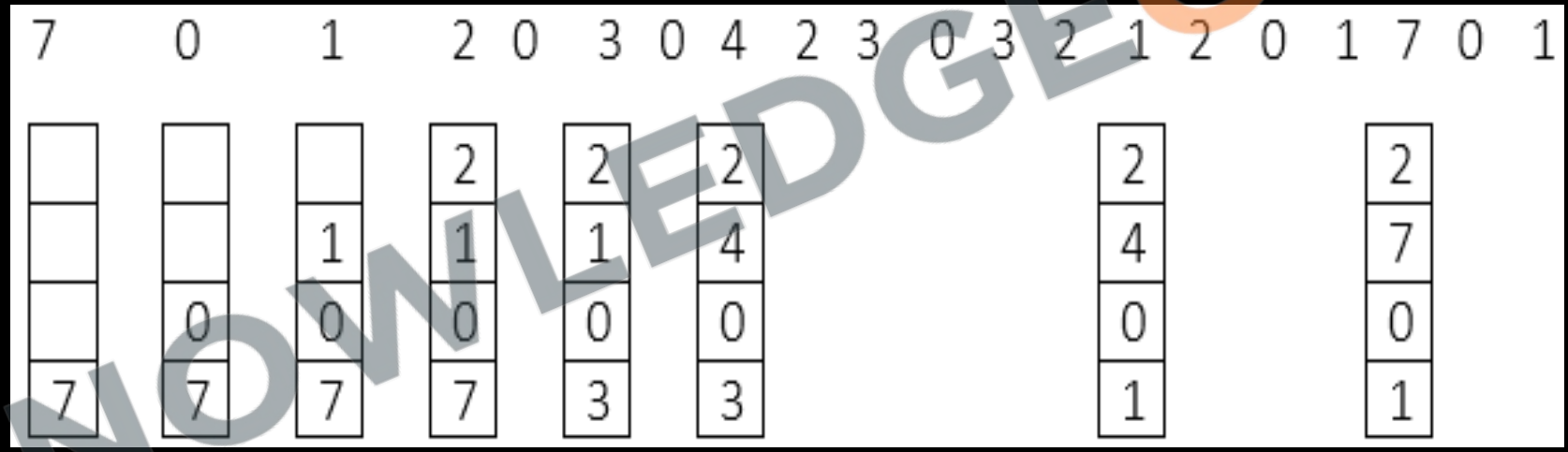
Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory



Optimal Algorithm

- The page which will not be used for the longest period of time in future references will be replaced first.
- The optimal algorithm will provide the best performance but it is difficult to implement as it requires the future knowledge of pages which is not possible.
- It is used as a benchmark for cache replacement algorithms.

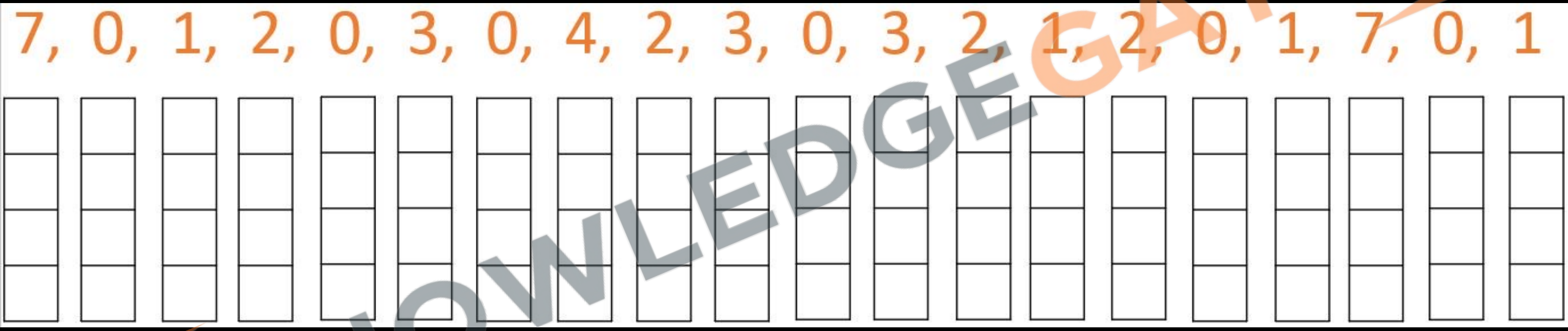
Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.



LRU (Least Recently Used)

- The page which was not used for the longest period of time in the past will get replaced first.

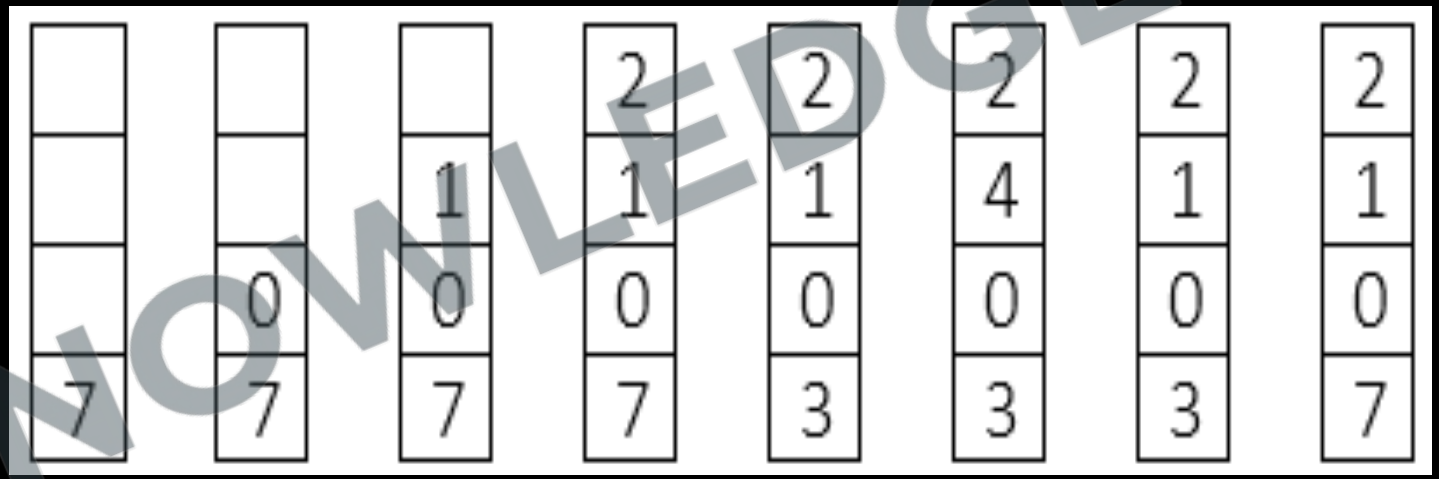
Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.



LRU (Least Recently Used)

- The page which was not used for the longest period of time in the past will get replaced first.

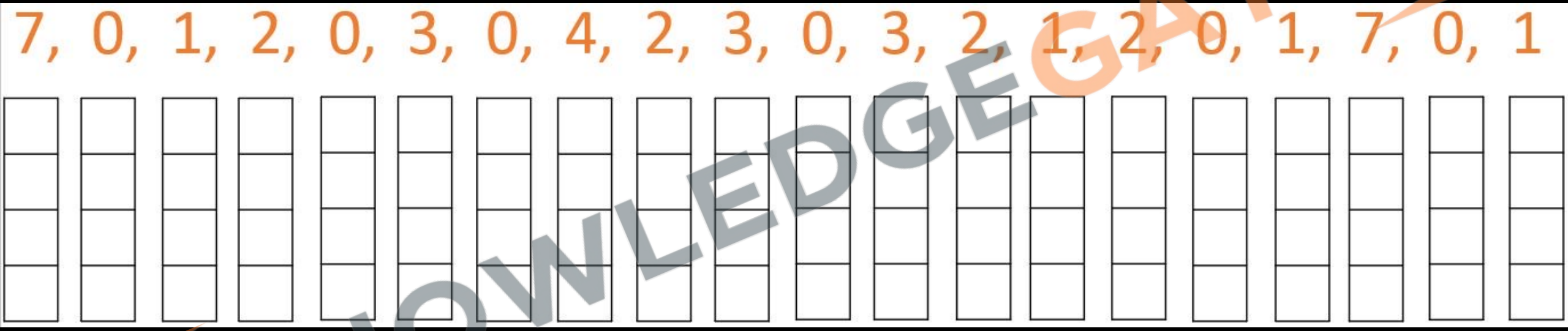
Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.



Most Recently Used (MRU)

- The page which was used recently will be replaced first.

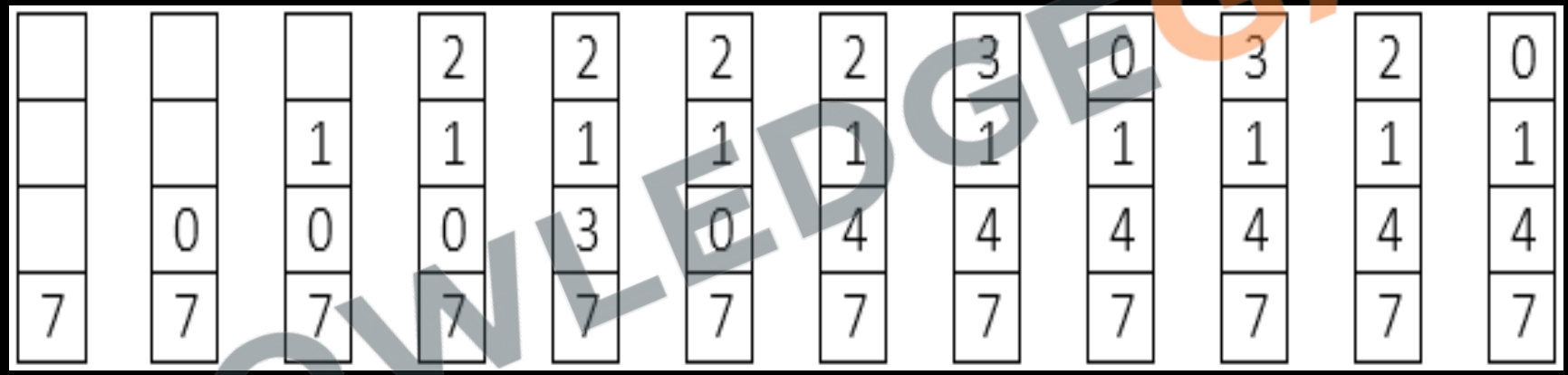
Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.



Most Recently Used (MRU)

- The page which was used recently will be replaced first.

Example: Let the blocks be in the sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 and the cache memory has 4 lines.



Types of Miss

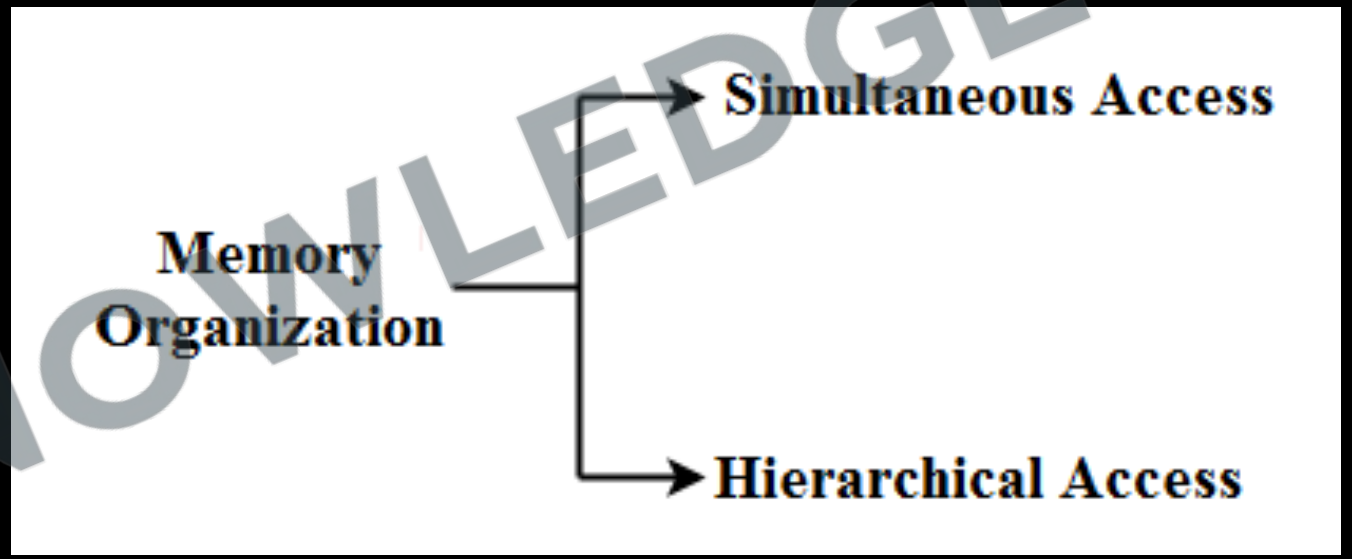
- **Compulsory Miss**
  - When CPU demands for any block for the first time then definitely a miss is going to occur as the block needs to be brought into the cache, it is known as Compulsory miss.

- **Capacity Miss**
  - Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution.

- **Conflict Miss**
  - In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.

Memory Organization

- Memory is organized at different levels.
- CPU may try to access different levels of memory in different ways.
- On this basis, the memory organization is broadly divided into two types



Hierarchical Access Memory Organization

- In this memory organization, memory levels are organized as
  - Level-1 is directly connected to the CPU.
  - Level-2 is directly connected to level-1.
  - Level-3 is directly connected to level-2 and so on

- Whenever CPU requires any word,
  - It first searches for the word in level-1.
  - If the required word is not found in level-1, it searches for the word in level-2.
  - If the required word is not found in level-2, it searches for the word in level-3 and so on.

T_1 = Access time of level L_1	T_2 = Access time of level L_2	T_3 = Access time of level L_3
S_1 = Size of level L_1	S_2 = Size of level L_2	S_3 = Size of level L_3
C_1 = Cost per byte of level L_1	C_2 = Cost per byte of level L_2	C_3 = Cost per byte of level L_3
H_1 = Hit rate of level L_1	H_2 = Hit rate of level L_2	H_3 = Hit rate of level L_3

Effective Memory Access Time

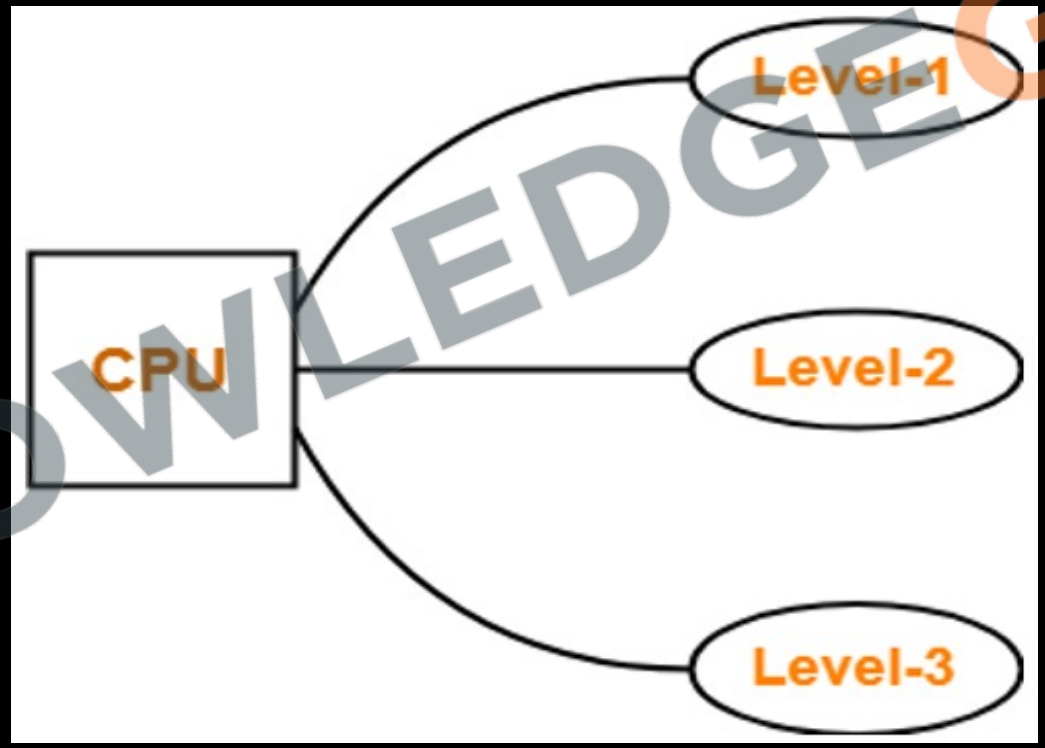
Average time required to access memory per operation =

$$H_1 * T_1 + (1 - H_1) * H_2 * (T_1 + T_2) + (1 - H_1)(1 - H_2) * H_3 * (T_1 + T_2 + T_3)$$

$$H_1 * T_1 + (1 - H_1) * H_2 * (T_1 + T_2) + (1 - H_1)(1 - H_2) * (T_1 + T_2 + T_3)$$

Simultaneous Access Memory Organization

- In simultaneous access all the levels of memory are directly connected to the CPU, whenever CPU requires any word, it starts searching for it in all the levels simultaneously.



- T_1 = Access time of level L_1
- S_1 = Size of level L_1
- C_1 = Cost per byte of level L_1
- H_1 = Hit rate of level L_1

- T_2 = Access time of level L_2
- S_2 = Size of level L_2
- C_2 = Cost per byte of level L_2
- H_2 = Hit rate of level L_2

- T_3 = Access time of level L_3
- S_3 = Size of level L_3
- C_3 = Cost per byte of level L_3
- H_3 = Hit rate of level L_3

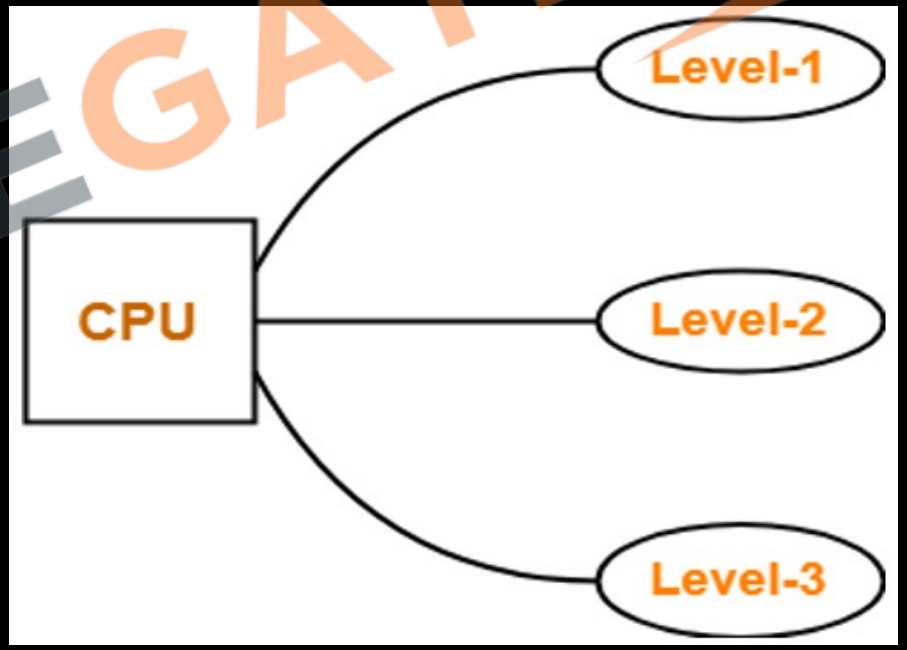
Effective Memory Access Time (EMAT) =

$$H_1 * T_1 + (1 - H_1) * H_2 * T_2 + (1 - H_1) (1 – H_2) * H_3 * T_3$$

In any memory organization, the data item being searched will definitely be present in the last level (or secondary memory).

Thus, hit rate for the last level is always 1. So,

$$H_1 * T_1 + (1 - H_1) * H_2 * T_2 + (1 - H_1) (1 – H_2) * T_3$$



Example: Calculate the EMAT for a machine with a cache hit rate of 80% where cache access time is 5ns and main memory access time is 100ns, both for simultaneous and hierarchical access.

Cache Coherence Problem

- If multiple copy of same data is maintained at different level of memories then inconsistency may occur, this problem is known as cache coherence problem.

- Cache coherence problem can be resolved using the following techniques:
  - Write Through
  - Write Back

Write Through

- Write through is used to maintain the consistency between the cache and main memory.
- According to it if the cache copy is updated, at the same time main memory is also updated.

- **Advantages**
  - It provides the highest level of consistency.

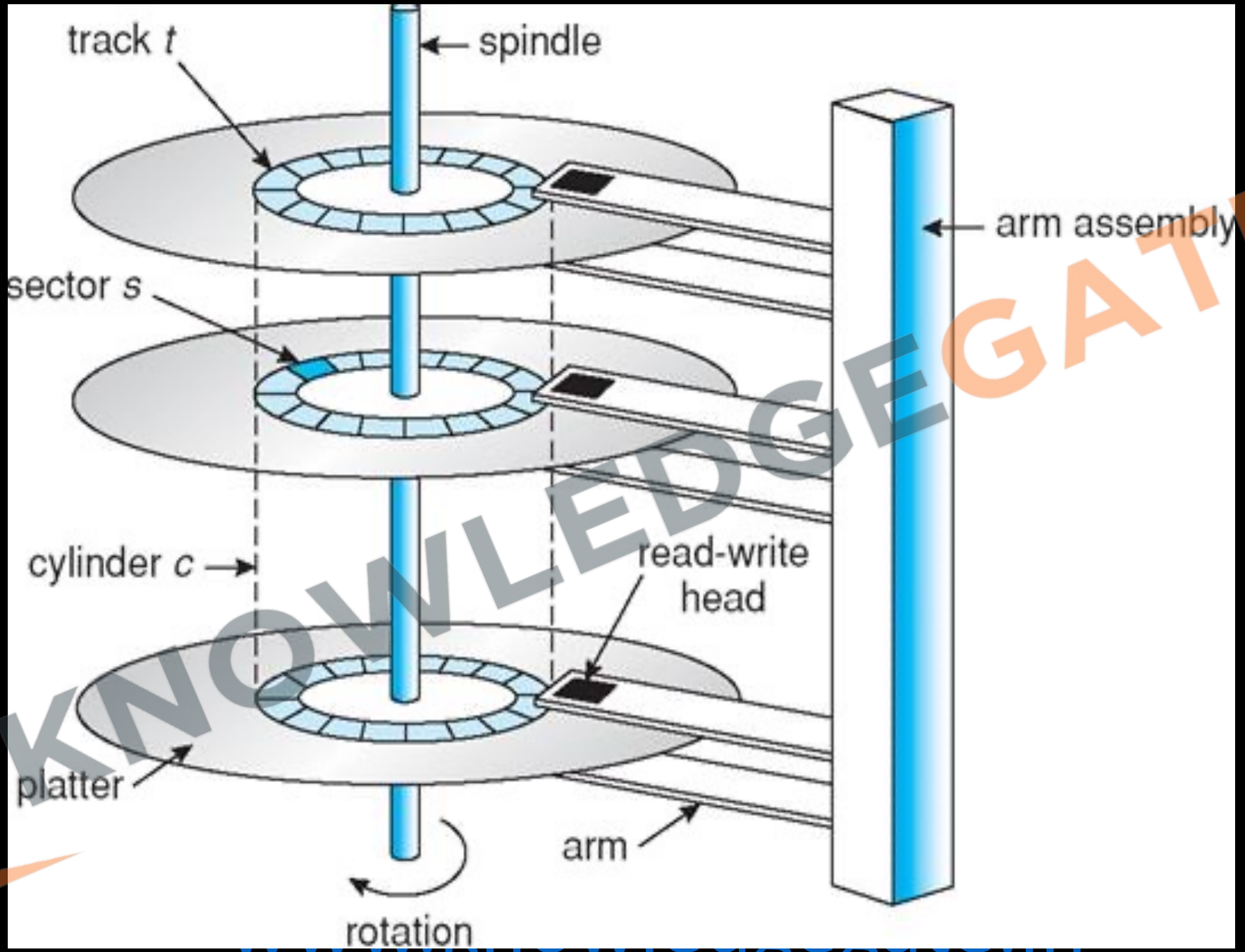
- **Disadvantages**
  - It requires more number of memory access.

Write Back

- Write back is also used to maintain the consistency between the cache and main memory.
- According to it all the changes performed on cache are reflected back to the main memory in the end.

- **Advantage**
  - Less number of memory accesses and less write operations.

- **Disadvantage**
  - Inconsistency may occur.



Total Transfer Time = Seek Time + Rotational Latency + Transfer Time

Seek Time: - It is a time taken by Read/Write header to reach the correct track. (Always given in question)

Rotational Latency: - It is the time taken by read/Write header during the wait for the correct sector. In general, it's a random value, so far average analysis, we consider the time taken by disk to complete half rotation.

Transfer Time: - it is the time taken by read/write header either to read or write on a disk. In general, we assume that in 1 complete rotation, header can read/write the either track, so
total time will be = (File Size/Track Size) *time taken to complete one revolution.

Q Consider a disk pack with 16 surfaces, 128 tracks per surface and 256 sectors per track. 512 bytes of data are stored in a bit serial manner in a sector. The capacity of the disk pack and the number of bits required to specify a particular sector in the disk are respectively: **(GATE-2007) (1 Marks)**

(A) 256 Mbyte, 19 bits

(B) 256 Mbyte, 28 bits

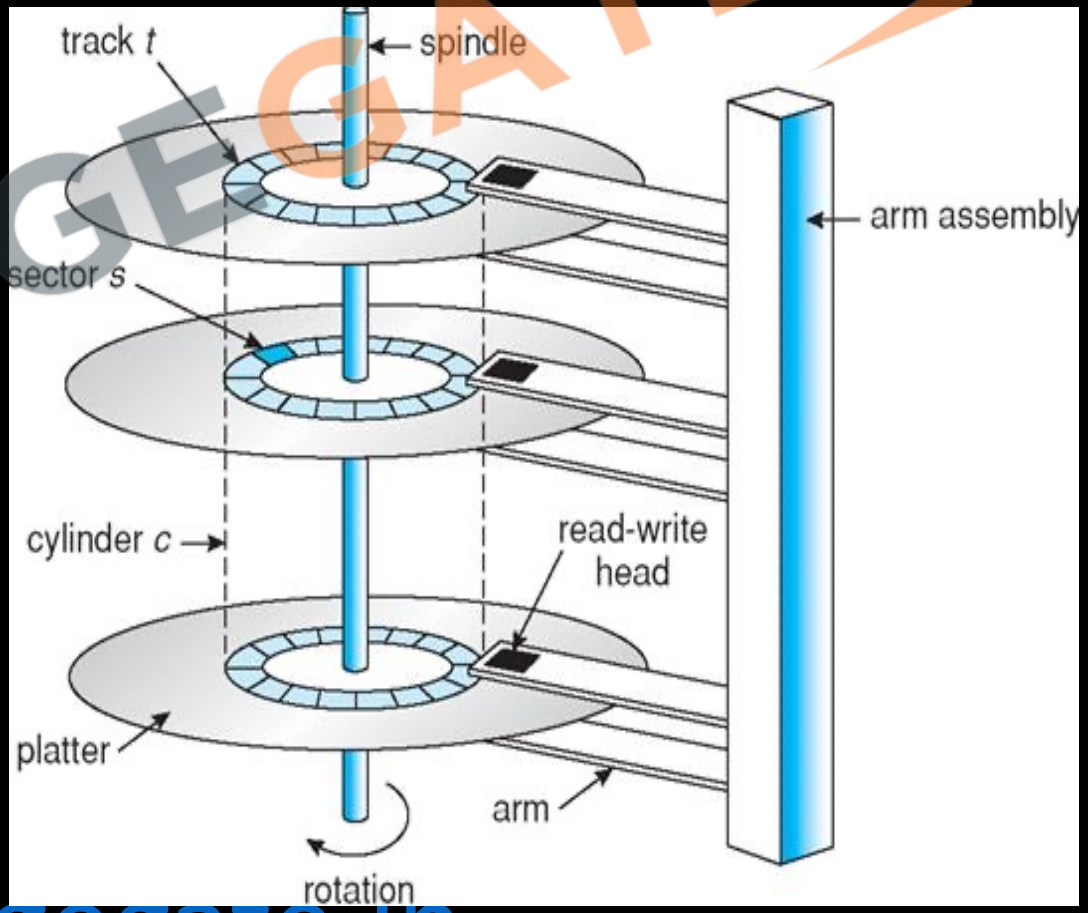
(C) 512 Mbyte, 20 bits

(D) 64 Gbyte, 28 bit

Q consider a disk where each sector contains 512 bytes and there are 400 sectors per track and 1000 tracks on the disk. If disk is rotating at speed of 1500 RPM, find the total time required to transfer file of size 1 MB. Suppose seek time is 4ms?

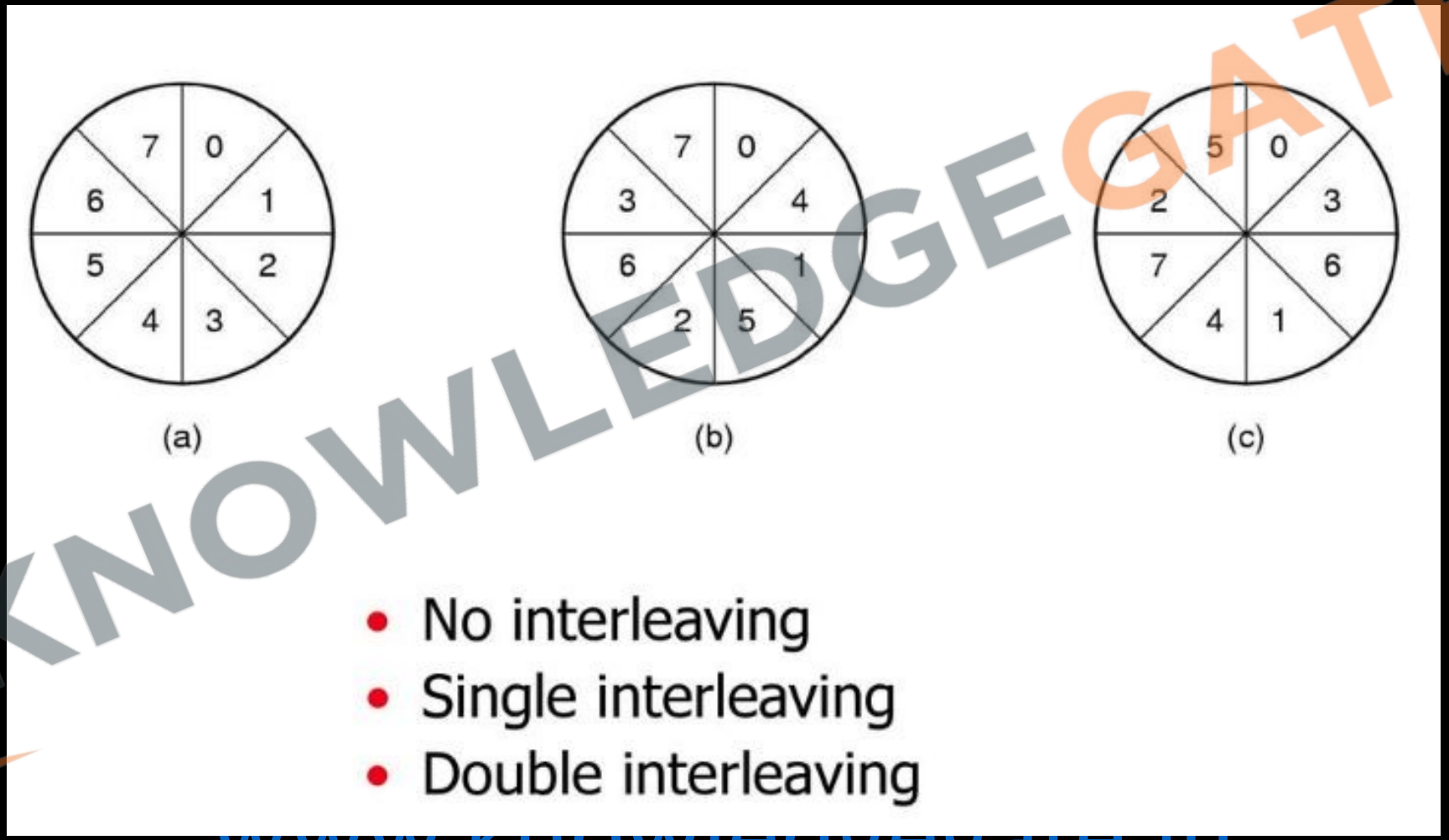
Q A hard disk has 63 sectors per track, 10 platters each with 2 recording surfaces and 1000 cylinders. The address of a sector is given as a triple (c, h, s), where c is the cylinder number, h is the surface number and s is the sector number. Thus, the 0th sector is addressed as (0, 0, 0), the 1st sector as (0, 0, 1), and so on The address <400,16,29> corresponds to sector number:

(A) 505035 **(B)** 505036 **(C)** 505037 **(D)** 505038



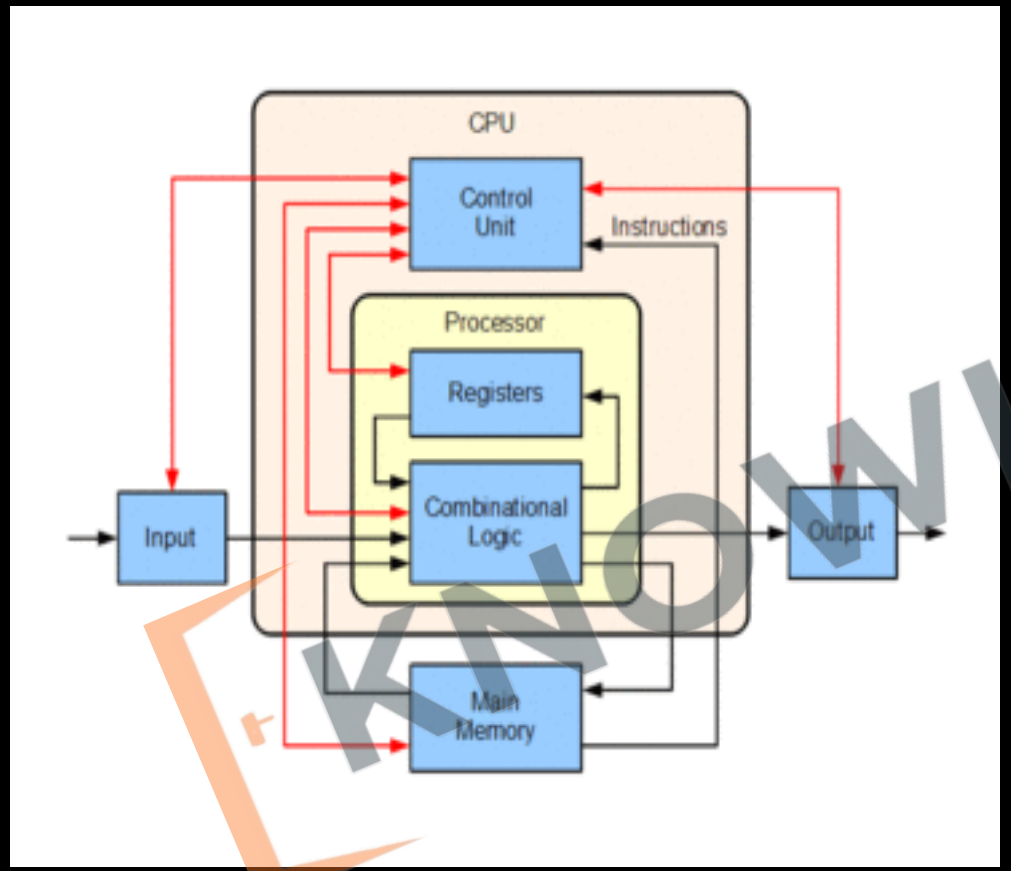
- **Single interleaving: -** in 2 rotation we read 1 track

- **Double interleaving: -** in 2.75 rotation we read 1 track



Input / Output management

- In general, when we say a computer, we understand a CPU + Memory (cache, main memory).
- But a computer does not serve any purpose if it cannot receive data from the outside world or cannot transmit the data to outside world.

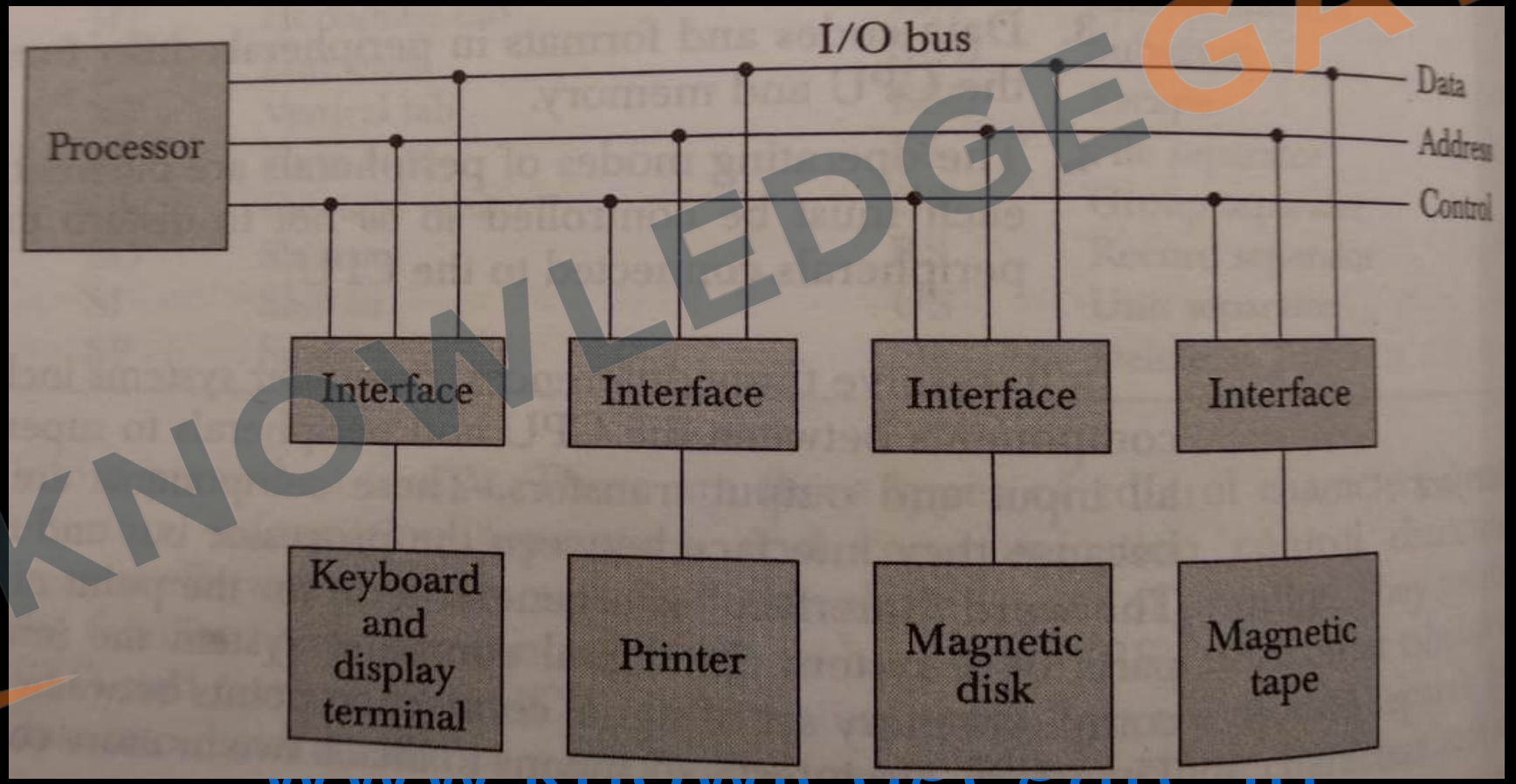


- I/o or peripheral devices are those independent devices which serve this purpose.
- So, while designing i/o for a computer we must know the number of i/o device and the capacity of each device.

Interface

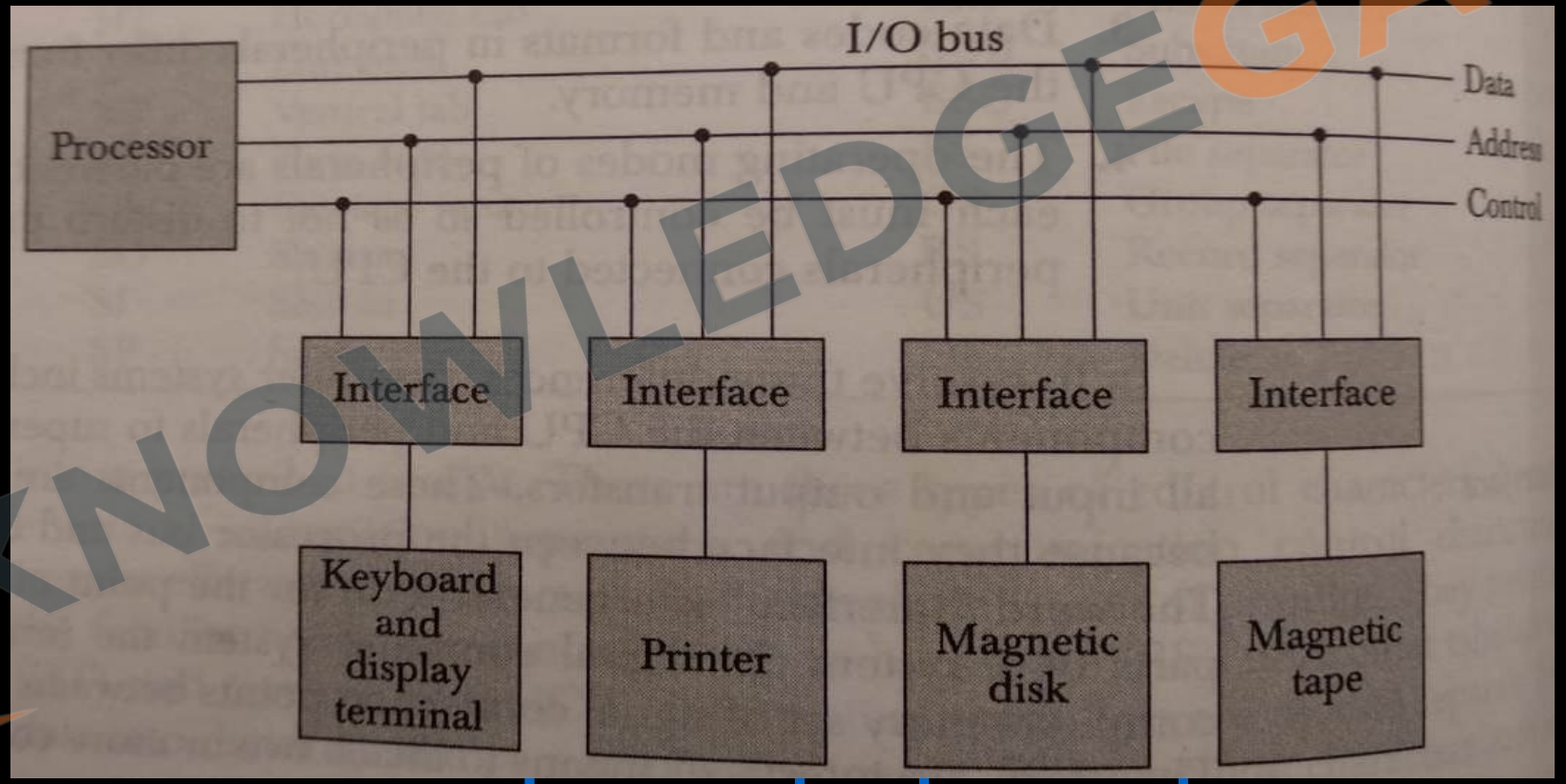
we cannot directly connect a i/o device to computer because of the following reasons

- **Speed: -** The speed of CPU and i/o device will usually different.
- **Format: -** The data code and format of CPU and peripherals may be different. E.g. ASCII, Unicode etc.



- **Physical orientation: -** Different device have organizations like optical, magnetic, electrochemical and different controlling functions.

- **Signal conversion: -** peripherals are electromagnetic and electrochemical device and their manner of operation is different from the operations of CPU and memory, which are electronic device, signal conversion is required.



- **I/O ports,** 1/0 ports, Interrupts: interrupt hardware, types of interrupts and exceptions

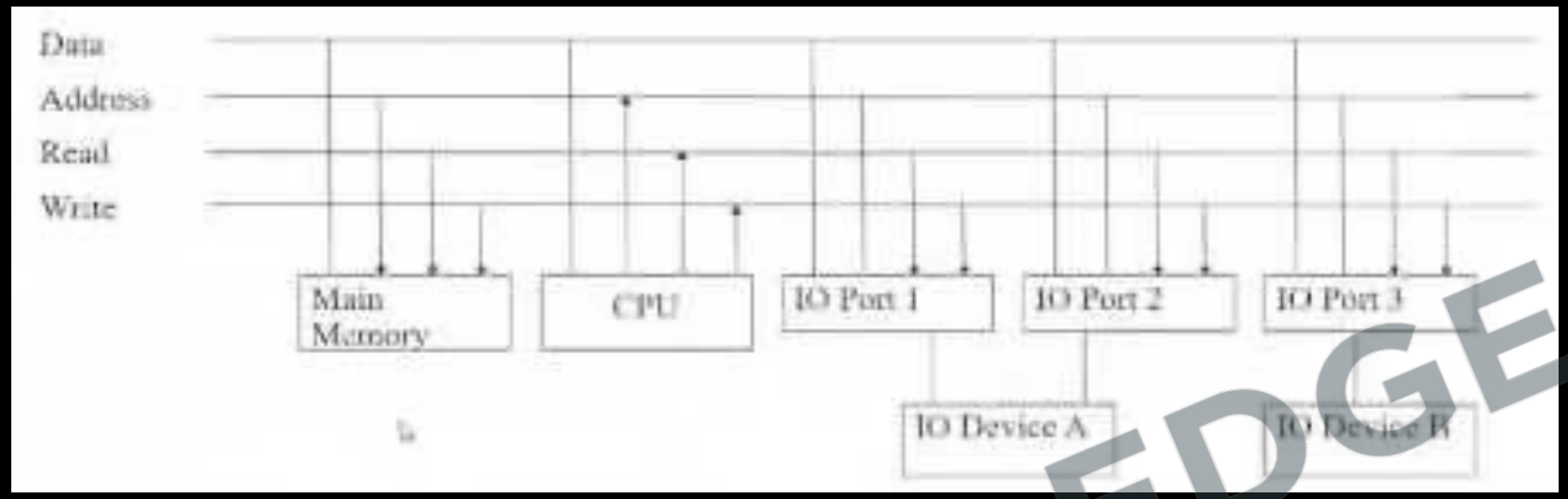
Interrupt

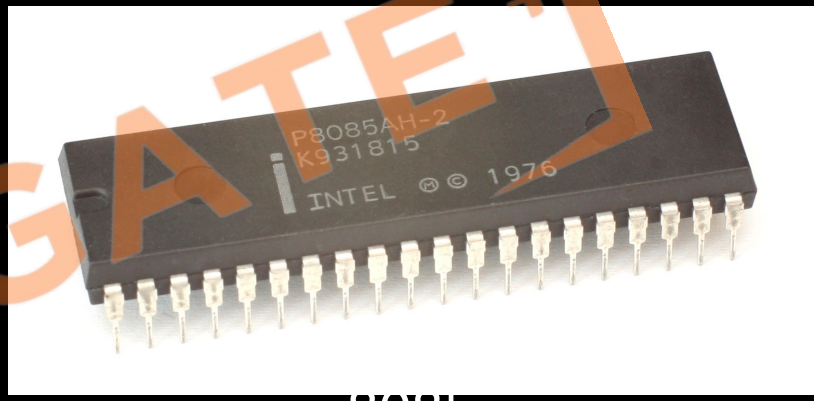
- It's a signal from a hardware device (like a keyboard or mouse) to get the processor's attention.
- **Polled Interrupt:**
  - Polled interrupt is like a "raising your hand" system in a classroom. In this method, the processor regularly checks each device one by one to see if it needs attention. It's like a teacher asking each student individually if they have a question. This can be slow because the processor has to keep asking every device, "Do you need anything?"
-
- **Vectored Interrupt:**
  - Vectored interrupt is more efficient, like a student coming directly to the teacher with a question. In this method, when a device sends an interrupt, it also sends a special code (vector) to the processor. The code tells the processor exactly what the device needs. So, the processor doesn't waste time checking all devices; it goes directly to the one that asked for help.

- each of the steps related to interrupt handling as single bullet points:
- **Interrupt Recognition**:
  - Processor identifies the interrupt, either from an external request or an internal mechanism.
  - Determines which device or CPU component issued the request.
- **Status Saving**:
  - Processor saves important information like flags and registers that might change during the interrupt.
  - Ensures the system can return to its previous state after handling the interrupt.
- **Interrupt Masking**:
  - Initially, all interrupts are blocked to prioritize the current interrupt.
  - Later, the processor allows higher-priority interrupts to be processed.
- **Interrupt Acknowledgment**:
  - Processor acknowledges the specific interrupt being serviced.
  - Allows the interrupting device to continue its task.
  - This acknowledgment can occur through an external signal line.
- **Interrupt Service Routine**:
  - Processor starts executing a specific routine designed to handle the interrupt.
  - The routine's address can be determined in various ways depending on the computer's architecture.
  - It's like following a set of instructions to deal with the interrupt.
- **Restoration and Return**:
  - After the interrupt service routine is completed, the processor restores all saved registers and flags to their initial state.
  - Ensures the system returns to its state before the interrupt occurred.

How a computer (CPU) deals with Memory and I/O devices

Memory Mapped i/o



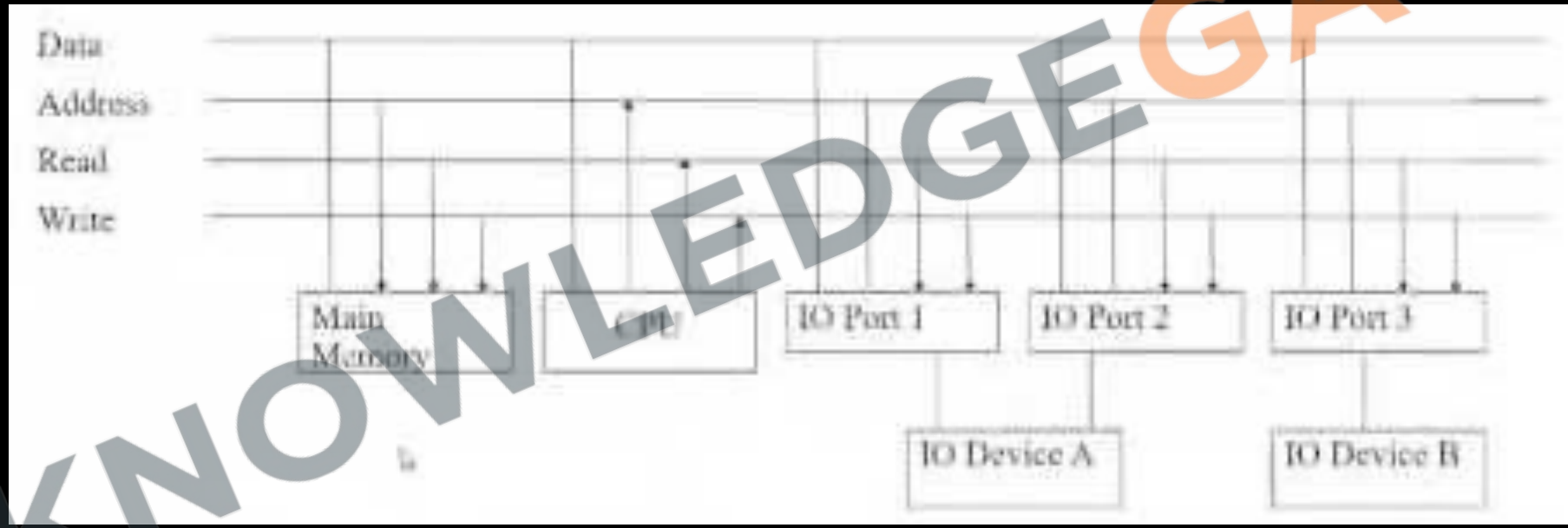


8085

- Here there is no separate i/o instructions. The CPU can manipulate i/o data residing in the interface registers with the same instructions that are used to manipulate memory words.
- Here computer can use memory type instructions for i/o data.
- E.g. 8085

- **Advantage: -** In typical computer there are more memory reference instruction than i/o instruction, but in memory mapped i/o all instructions that refer to memory are also available for i/o.

- **Disadvantage: -**Total address get divided, some range is occupied by i/o while some memory.



- The **Commodore 64**, also known as the **C64** or the **CBM 64**, is an 8-bit home computer introduced in January 1982 by Commodore International (first shown at the Consumer Electronics Show, in Las Vegas, January 7–10, 1982).

- It has been listed in the Guinness World Records as the highest-selling single computer model of all time, with independent estimates placing the number sold between 10 and 17 million units.

Isolated i/o

- Here common bus to transfer data between memory or i/o and CPU. The distinction between a memory and i/o transfer is made through separate read and write line.
- i/o read and i/o write control lines are enabled during an i/o transfer.
- Memory read and memory write control lines are enabled during a memory transfer. E.g 8086

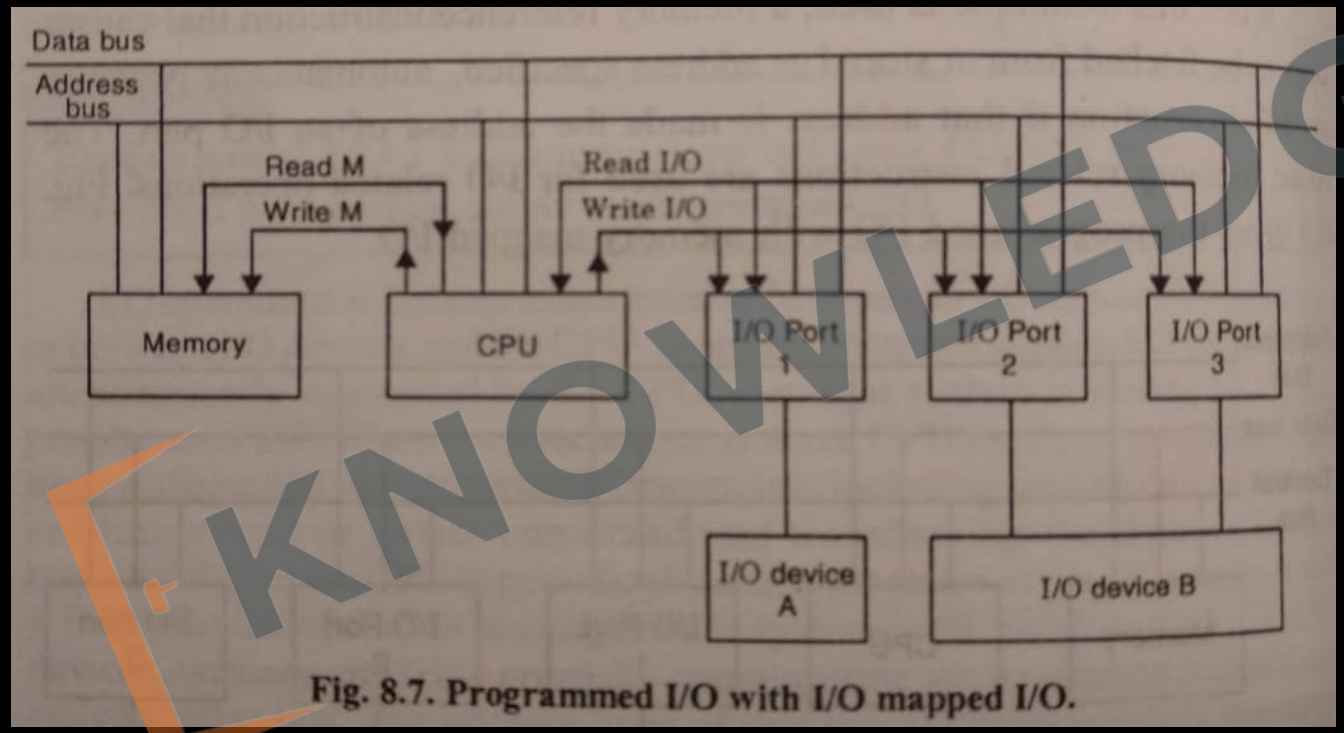
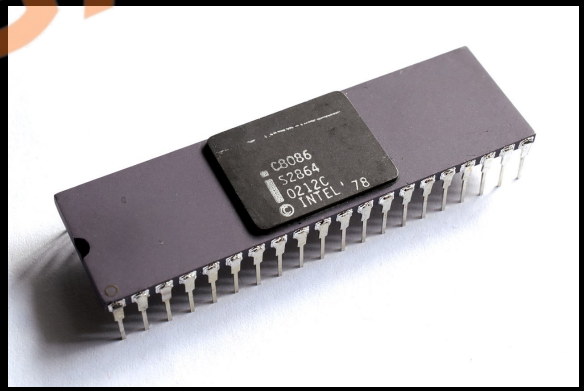


Fig. 8.7. Programmed I/O with I/O mapped I/O.



- **Advantage: -** Here memory is used efficiently as the same address can be used two times.

- **Disadvantage: -** Need different control lines one for memory and other for i/o devices.

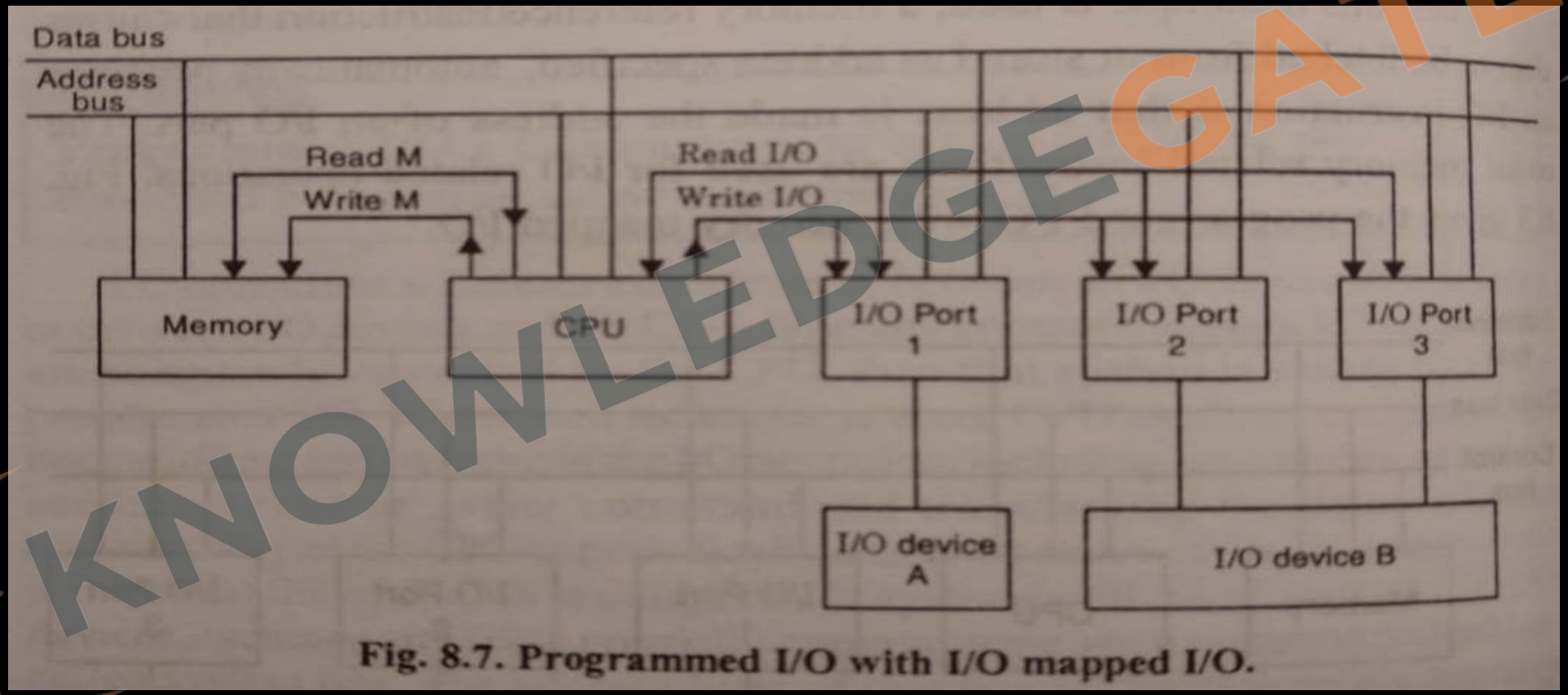
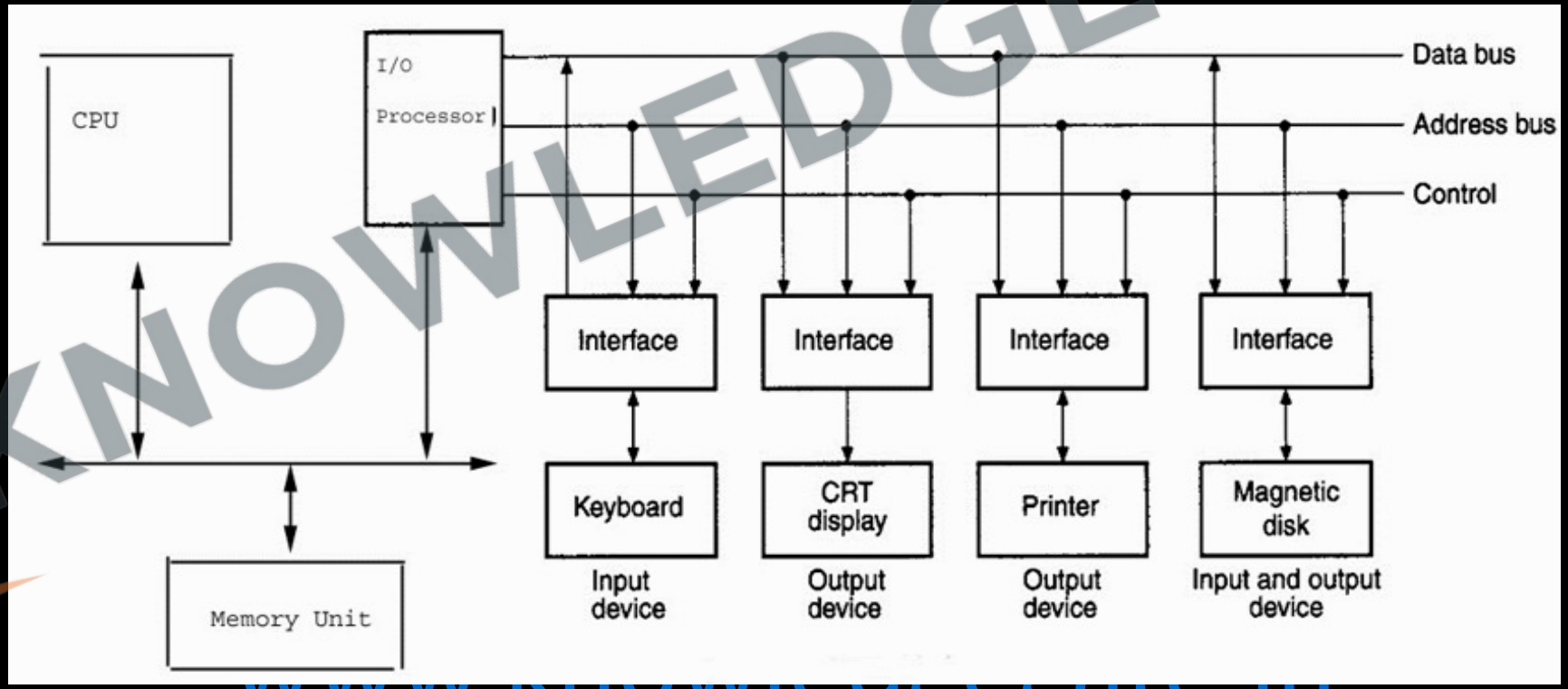


Fig. 8.7. Programmed I/O with I/O mapped I/O.

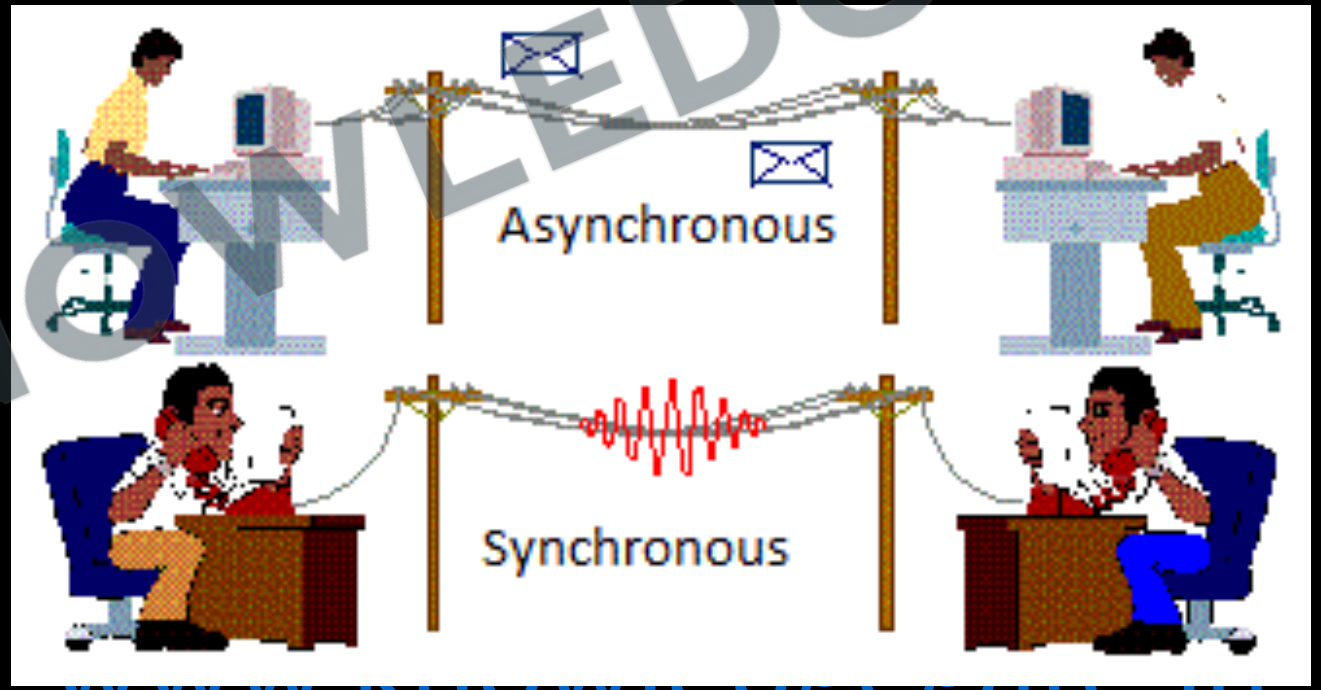
I/O Processor

- Computer has independent sets of data, address and control buses, one for accessing memory and the other for i/o. this is done in computers that provide a separate i/o processor other than CPU.
- Memory communicate with both the CPU and iop through a memory bus.
- Iop communicates also with the input and output devices through a separate i/o bus with its own address, data and control lines. The purpose of iop is to provide an independent pathway for the transfer of information between external devices and internal memory.



Synchronous Vs Asynchronous data transfer

- Synchronization is achieved by a device called master generator, which generate a periodic train of clock pulse.
- The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator.
- When communication happens between devices which are under a same control unit or same clock then it is called synchronous communication. e.g. communication between CPU and its registers.

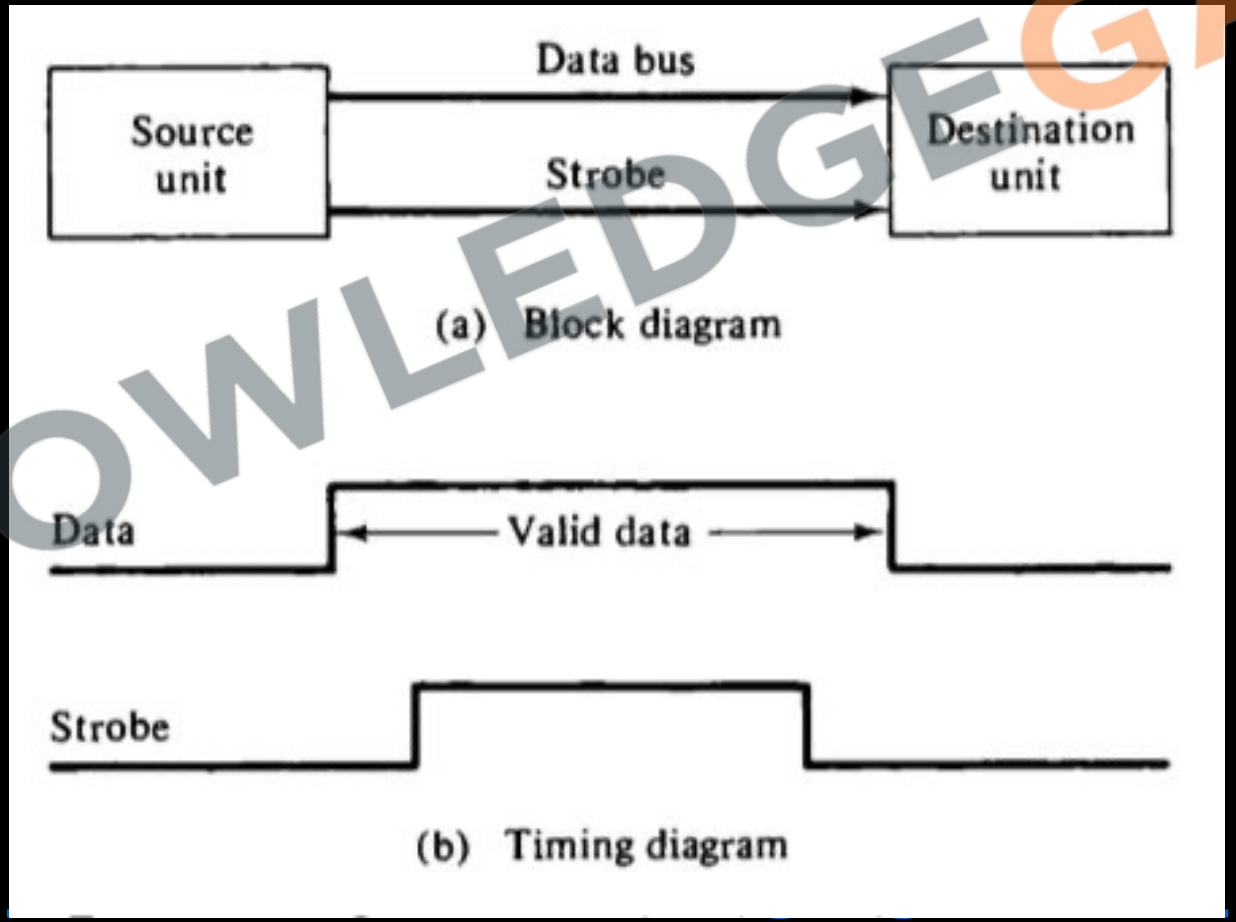


Asynchronous communication

- When the timing units of two devices are independent that is they are under different control then it is called asynchronous communication.
- Asynchronous data transfer between two independent units required that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted.
- One way to achieving this by means of strobe pulse supplied by one of the units to indicate to the other unit which when the transfer has to occur.

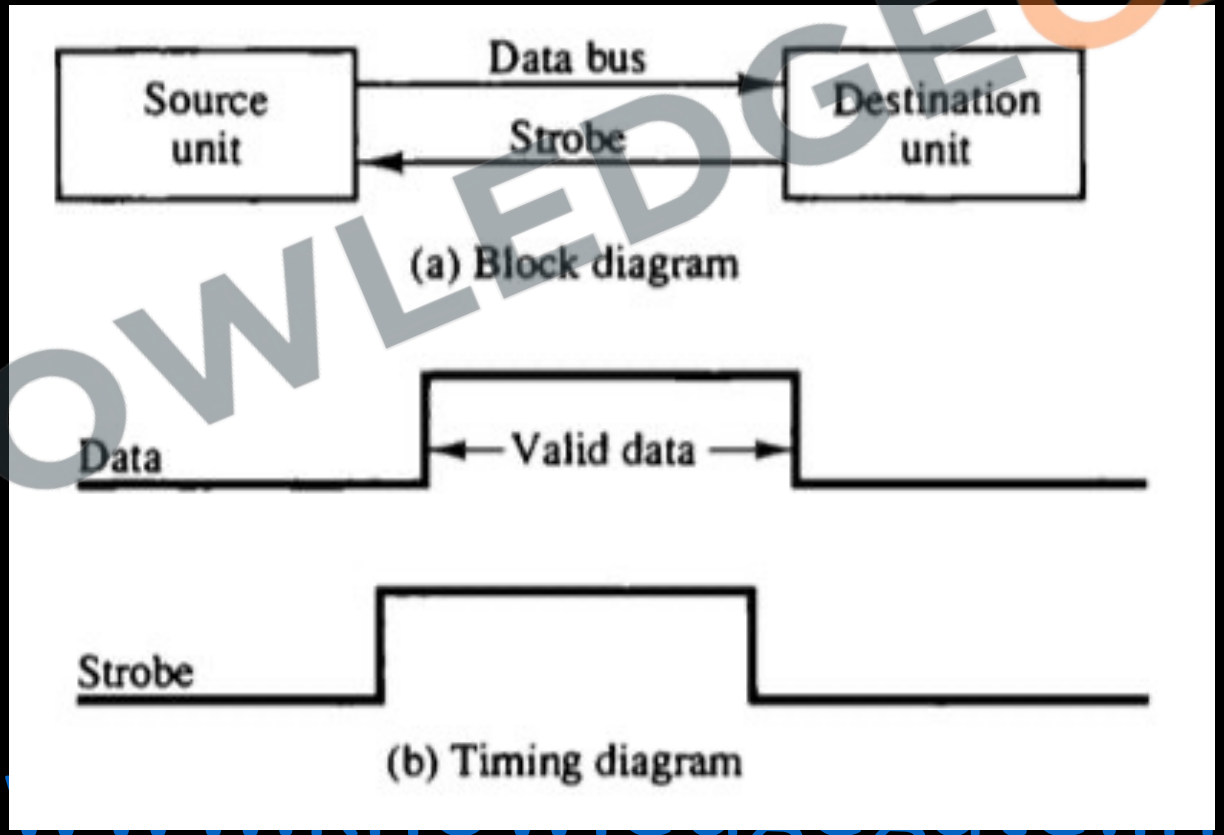
Source Initiated I/O

- the unit receiving the data item response with another control signal to acknowledge the receipt of the data. this type of agreement between two independent units is referred to as handshaking.
- The strobe pulse method and the handshaking method of asynchronous data transfer are not restricted to input output transfer. in fact they are used extensively on numerous occasions requiring the transfer of data between two independent units.



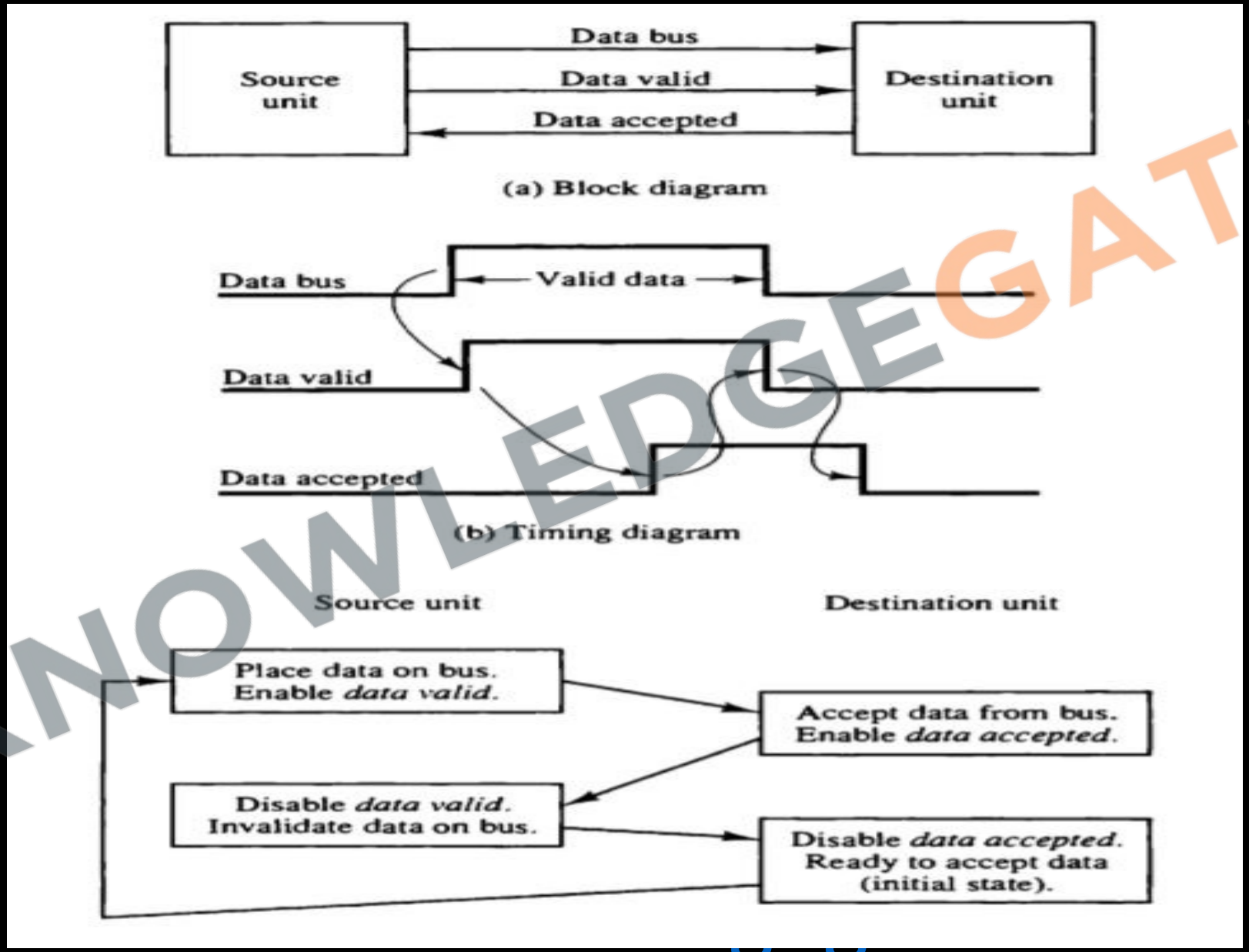
Destination Initiated I/O

- the unit receiving the data item response with another control signal to acknowledge the receipt of the data. this type of agreement between two independent units is referred to as handshaking.
- The strobe pulse method and the handshaking method of asynchronous data transfer are not restricted to input output transfer. in fact they are used extensively on numerous occasions requiring the transfer of data between two independent units.

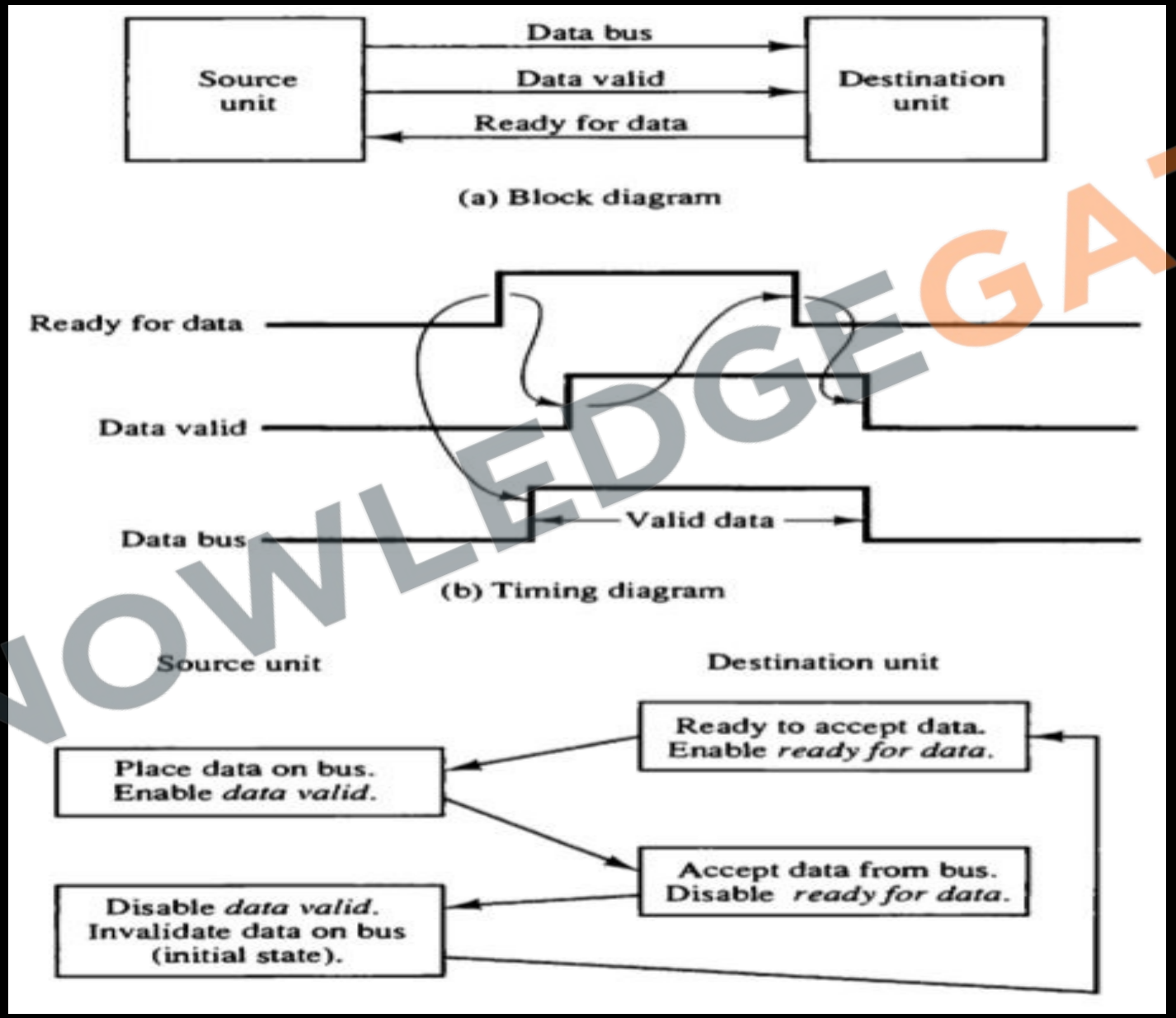


- the unit receiving the data item response with another control signal to acknowledge the receipt of the data. this type of agreement between two independent units is referred to as handshaking.
- The strobe pulse method and the handshaking method of asynchronous data transfer are not restricted to input output transfer. in fact they are used extensively on numerous occasions requiring the transfer of data between two independent units.

Source Initiated Transfer



Destination initiated transfer



Modes of Data Transfer

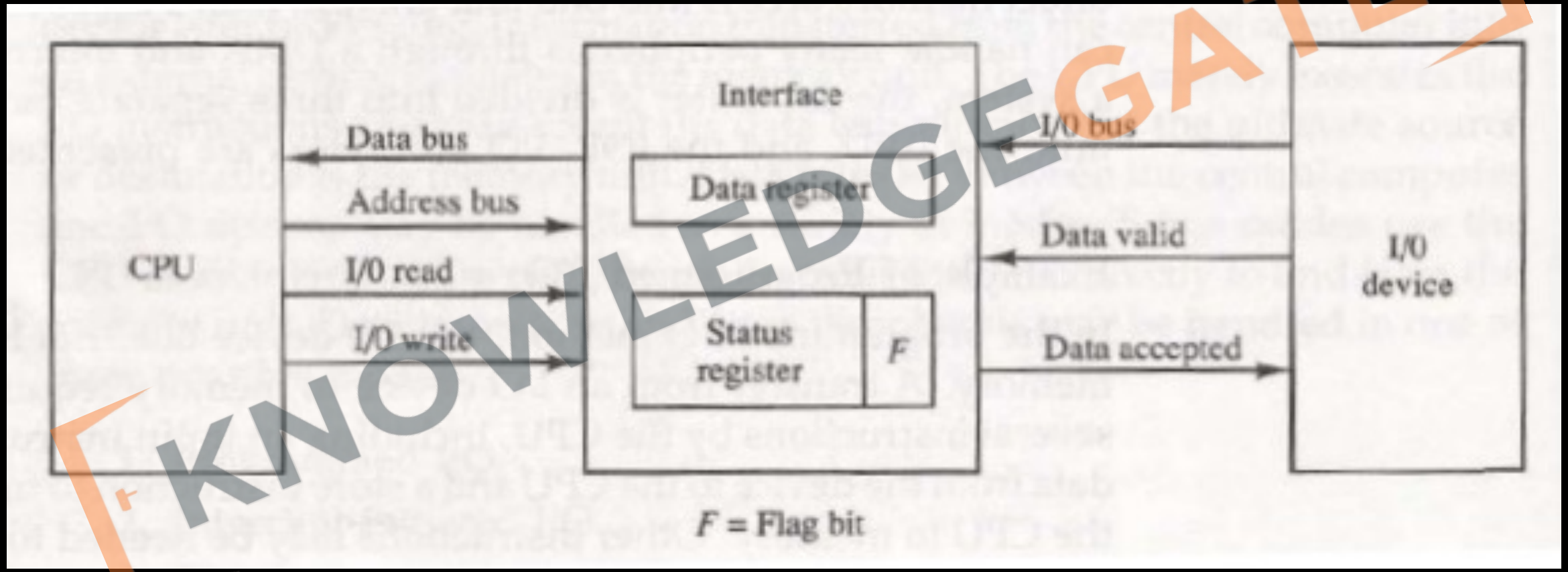
Here we deal with the problem that how data communication will take place CPU and i/o device.

There are popularly three methods of data transfer: -

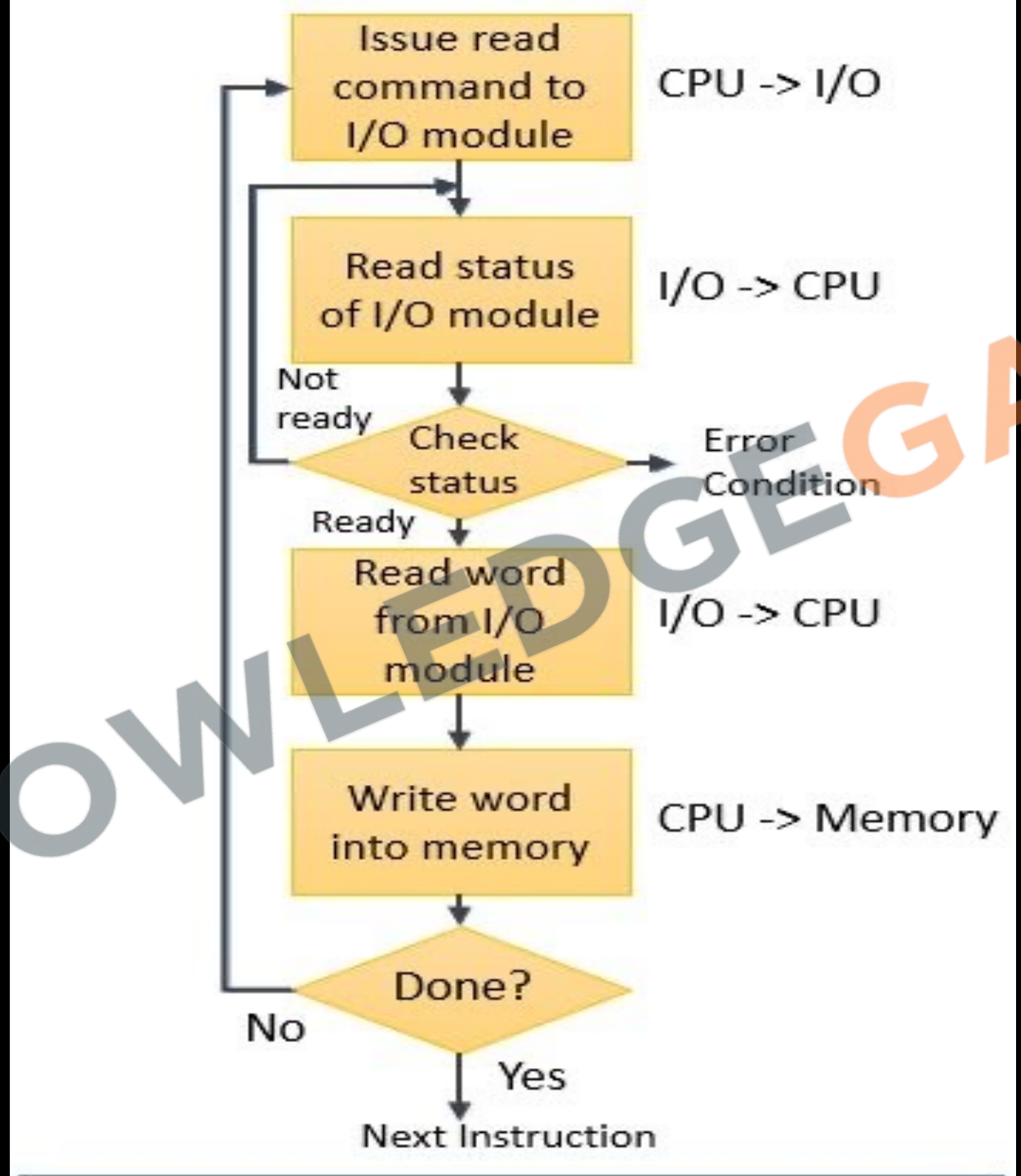
- **Programmed i/o**
- **Interrupt initiated i/o**
- **Direct Memory Transfer**

Programmed I/O

- In this i/o device cannot access the memory directly. To complete data transfer number of instructions will be executed out of which input instruction are those which transfer data from device to CPU and store instructions from CPU to memory.

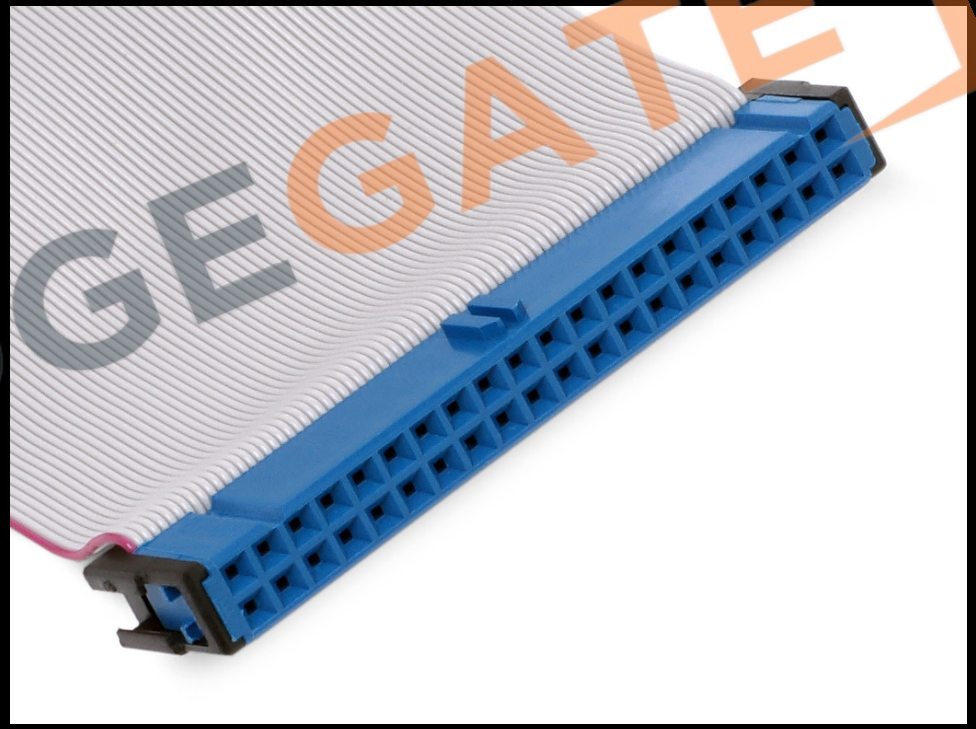


- **Step 1: -** i/o device will put data on the data bus and will enable valid data signal
- **Step 2: -** Interface is continuously sensing for data valid signal and when it receives the signal it will copy data from the data bus into it's data register and set its flag bit to 1 and enable data accepted line.
- **Step 3: -** CPU is continuously monitoring the status register in the programmed mode and as soon as it sees flag bit as 1, it immediately copies data from data register on the data bus and clear flag bit to zero.
- **Step 4: -** Now interface will disable data accepted line to tell i/o device, I am ready for new transitions.
- **Conclusion: -** CPU works in programmed mode or in busy wait mode so no of clock cycles are wasted. It will be difficult to handle multiple i/o device at the same time. It is not appropriate with the high-speed i/o devices.



- The best known example of a PC device that uses programmed I/O is the ATA interface

- **Parallel ATA** (**PATA**), originally **AT Attachment**, is an interface standard for the connection of storage devices such as hard disk drives, floppy disk drives, and optical disc drives in computers.



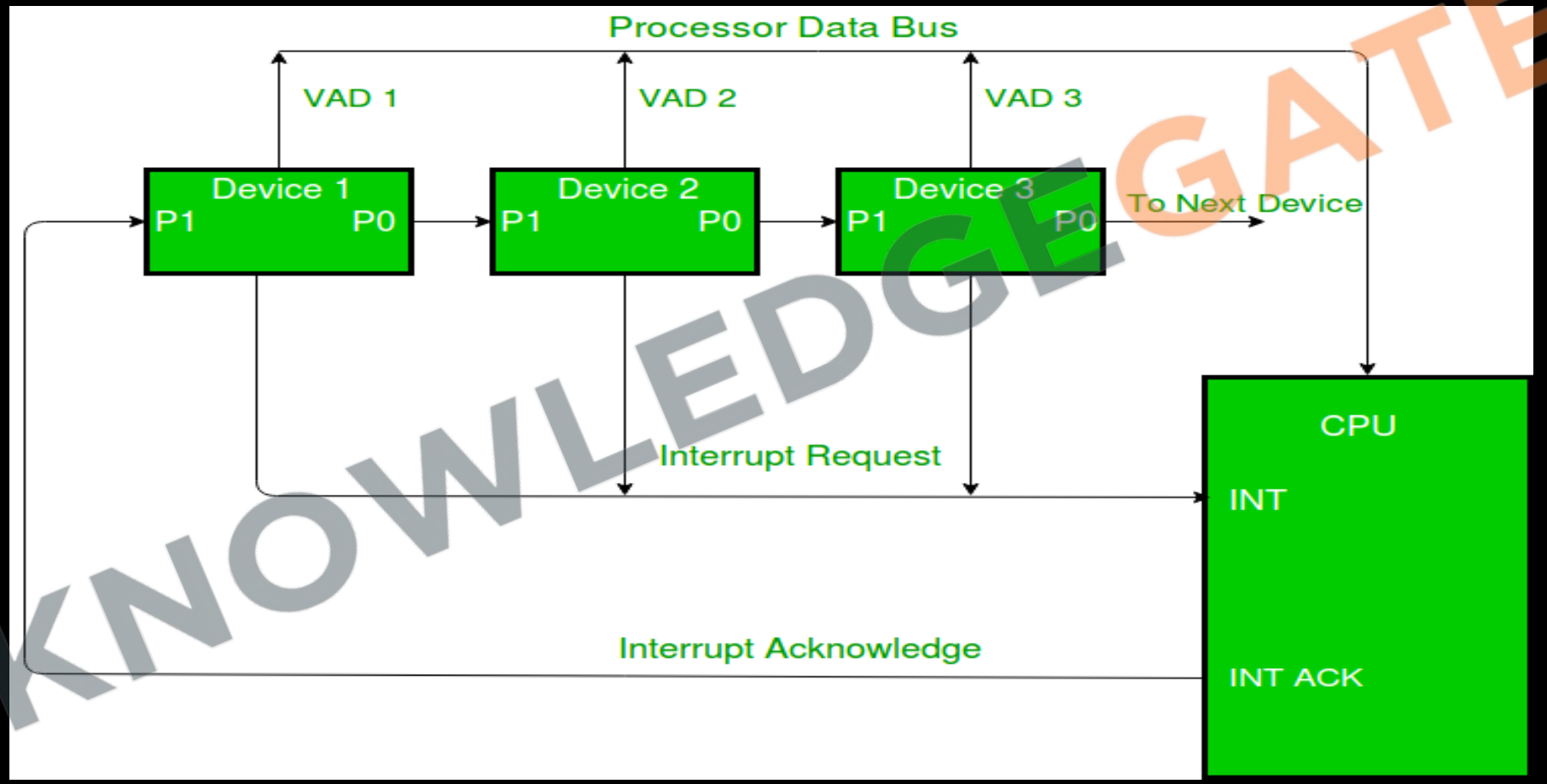
Interrupt initiated i/o

- **In** this method I/o device interrupt CPU when it is ready for data transfer.
- CPU keep executing instructions and after executing one instruction and before starting another instructions CPU wait and see if there is interrupt or not and if there is interrupt then takes a decision whether CPU should entertain this interrupt or continue with the execution.
- **Note: -** instruction is always absolute in nature and there is nothing like partial execution of instruction.
- The method of handling interrupts of different i/o devices are different, therefor every device has a program or routine called ISR (interrupt service routine) which tell CPU how the interrupt should be managed (and it saves CPU time).

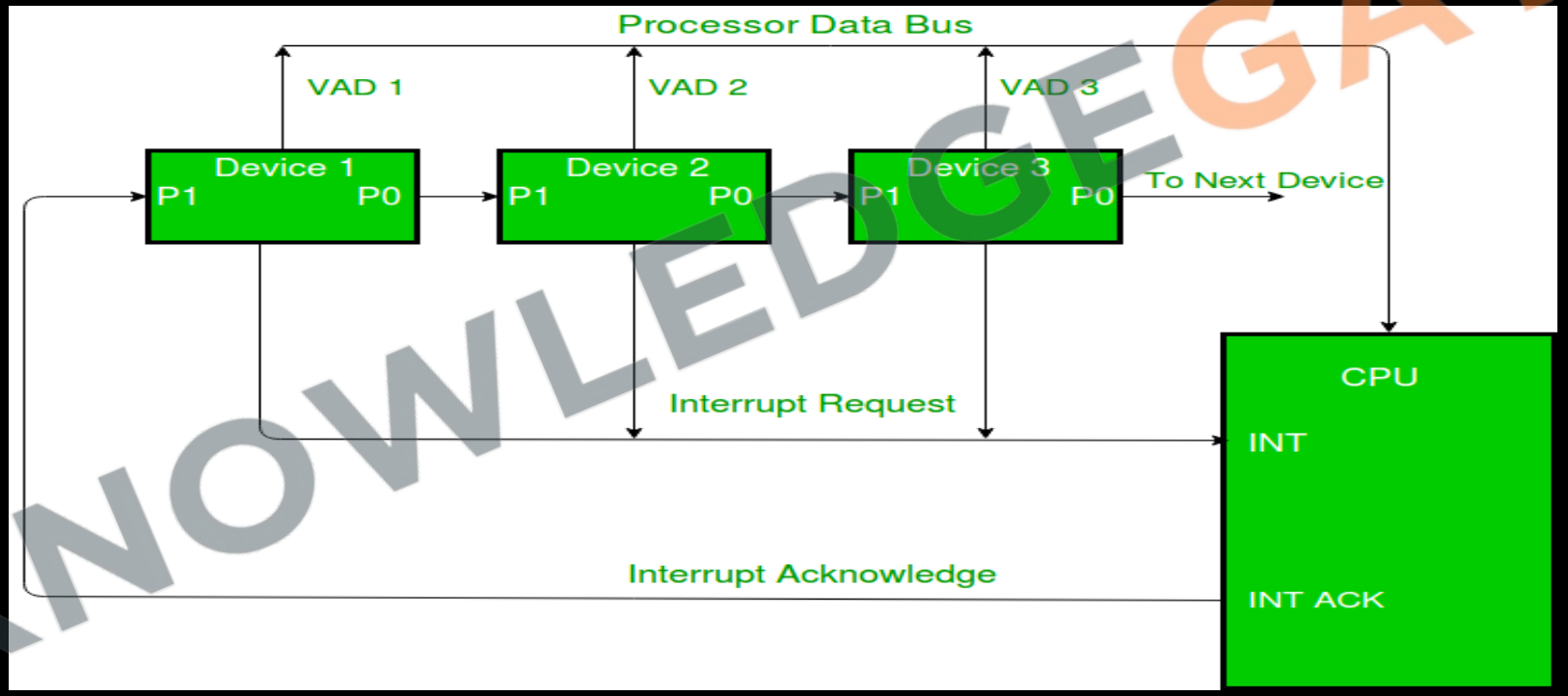
- Interrupt can be of two types: -
  - **Non-vectored interrupt: -** Here there is a mutual understanding between CPU and device that where this routine is stored in memory (high priority device)
  - **Vectored interrupt: -** Some interrupts may be vectored where the interrupting device will also tell the address that where this routine is stored in the memory.
- It is possible that different i/o devices interrupt at the same time. Now CPU must have a priority decision that which interrupt should be service first and which should be service later.

Feature	Vectored Interrupt	Non-Vectored Interrupt
Addressing	Directly jumps to specific ISR	Jumps to a fixed location
Efficiency	Faster, less overhead	Slower due to extra steps
Complexity	More complex to set up	Simpler to implement
Flexibility	Higher, allows multiple ISRs	Lower, usually one ISR
Use Case	Suitable for systems with multiple devices requiring different handling	More appropriate for simpler systems with fewer interrupt sources

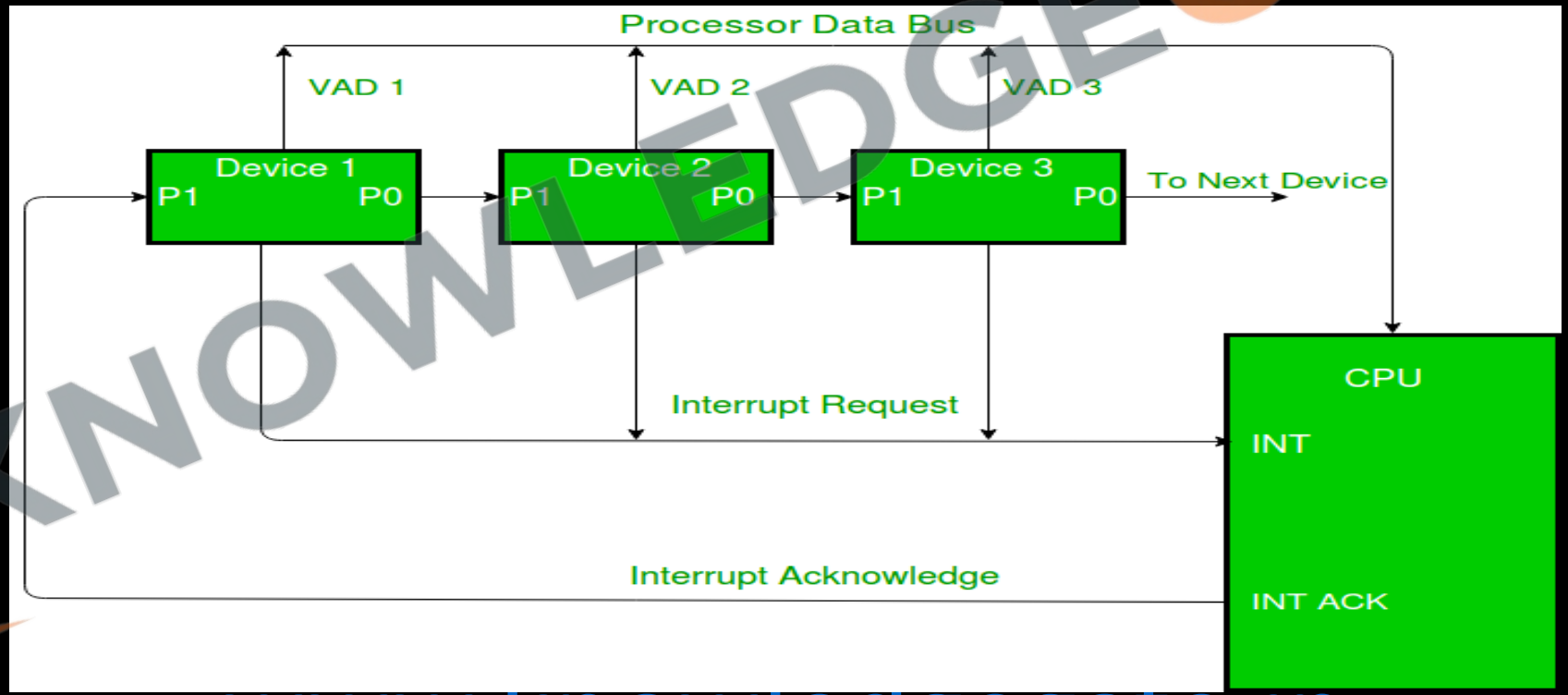
- **H/w solution: -** It can be serial or parallel, serial solution is known as Daisy Channing is a h/w solution which is used to decide priority among different i/o devices (VAD- vector address device)



- Out of all possible devices 1 or more device may send an interrupt with a common line.
- When CPU completes 1 instruction and check interrupt in line and found an interrupt, then CPU will enable interrupt acknowledgment line to 1.



- The i/o device placed first in the arrangement will always get acknowledgement first. If it wants to perform i/o then it will put 0 on priority outline and will put the address of its interrupt service routine (ISR) on the vector address line.
- If device do not want to perform i/o then will set priority output as 1 and will give chance to the second device, and the processor continue.
- **Advantage: -** very simple, easy to use, easy to understand, relatively fast.
- **Disadvantage: -** here priority fixed and even in case of requirement we cannot change it.



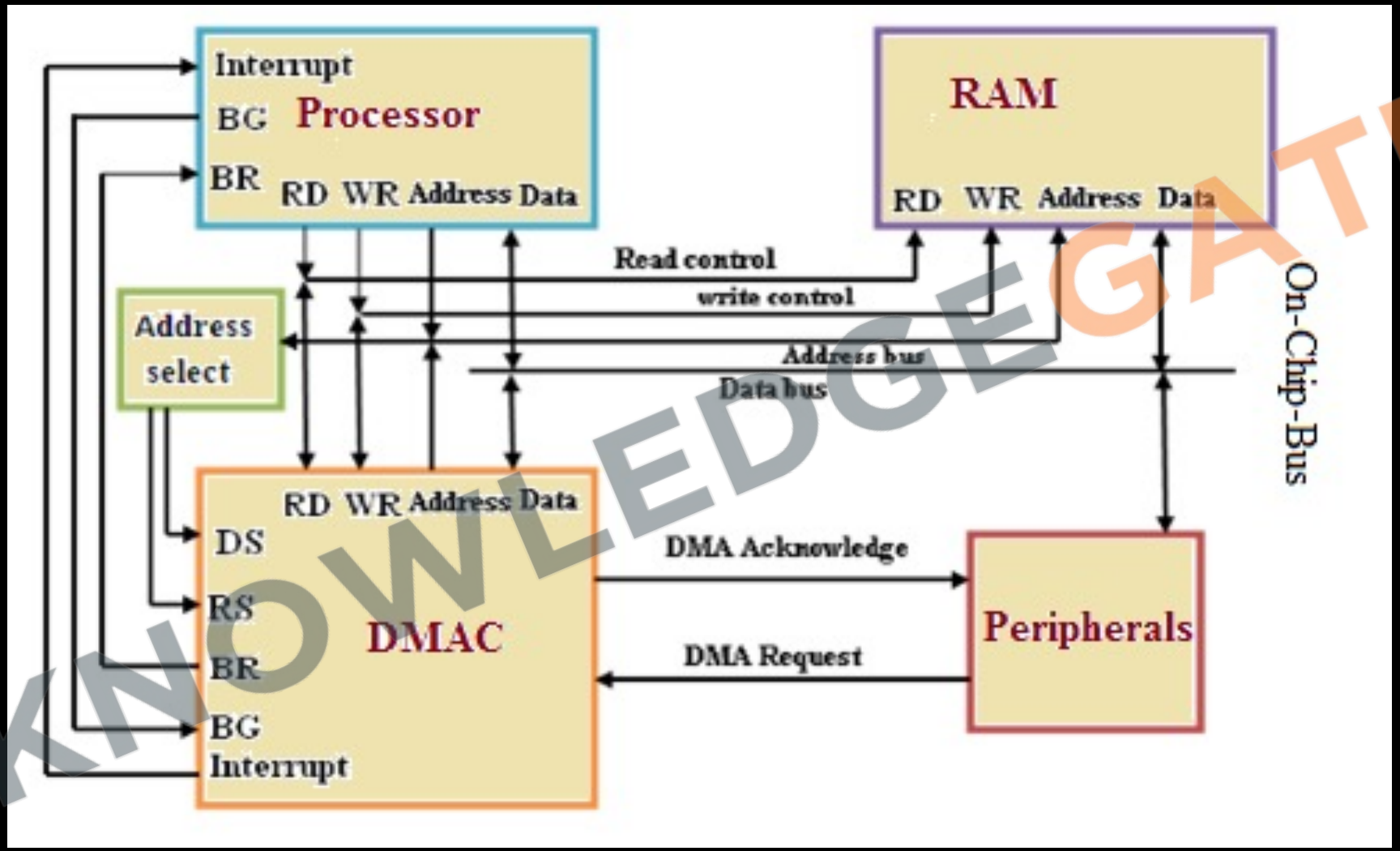
- **Polling for Identification**: When multiple devices send interrupts, the CPU polls them in a predefined sequence to identify which device sent the interrupt. The first device to confirm gets serviced first.
- **Static Priority via Polling**: The order in which devices are polled effectively sets their priority. Devices polled earlier are serviced first, making this a form of static priority assignment.
- **Simplicity & Limitations**: This approach is straightforward but rigid. It's suitable for systems with fixed priorities but may not be ideal for dynamic environments where priority needs can change rapidly.

- **Hardware and software interrupts** :
  - The interrupts initiated by an external hardware by sending an appropriate signal to the interrupt pin of the CPU is called hardware interrupt. The software interrupts are program instructions. These instructions are inserted at desired location in a program. While running a program, if software interrupt instruction is encountered the CPU initiates an interrupt

- **Maskable and non-maskable interrupts**.
  - The interrupts whose request can be either accepted or rejected by the CPU are called maskable interrupts.
  - The interrupts whose request has to be definitely accepted by the CPU are called non-maskable interrupts.

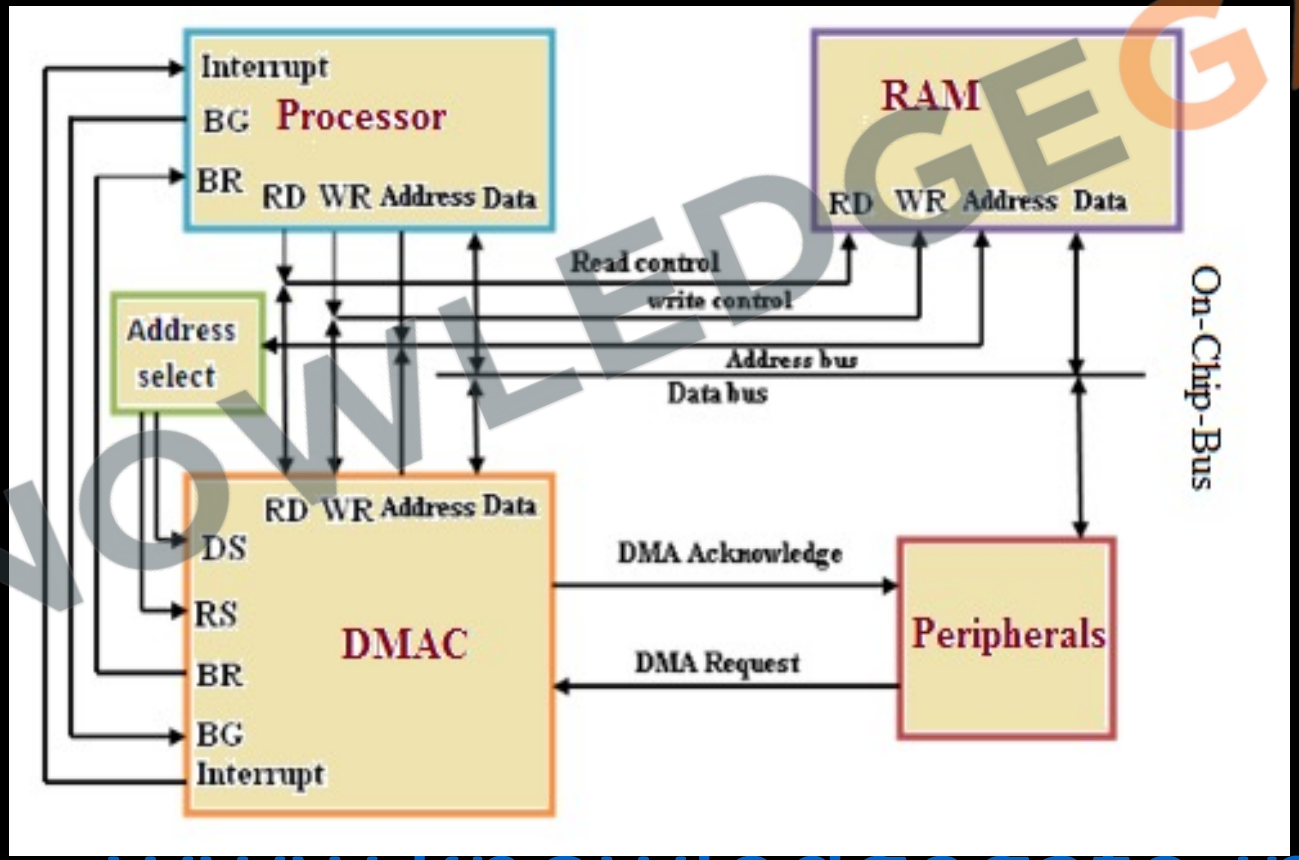
- Intel 82C59A

Direct Memory Access (DMA)



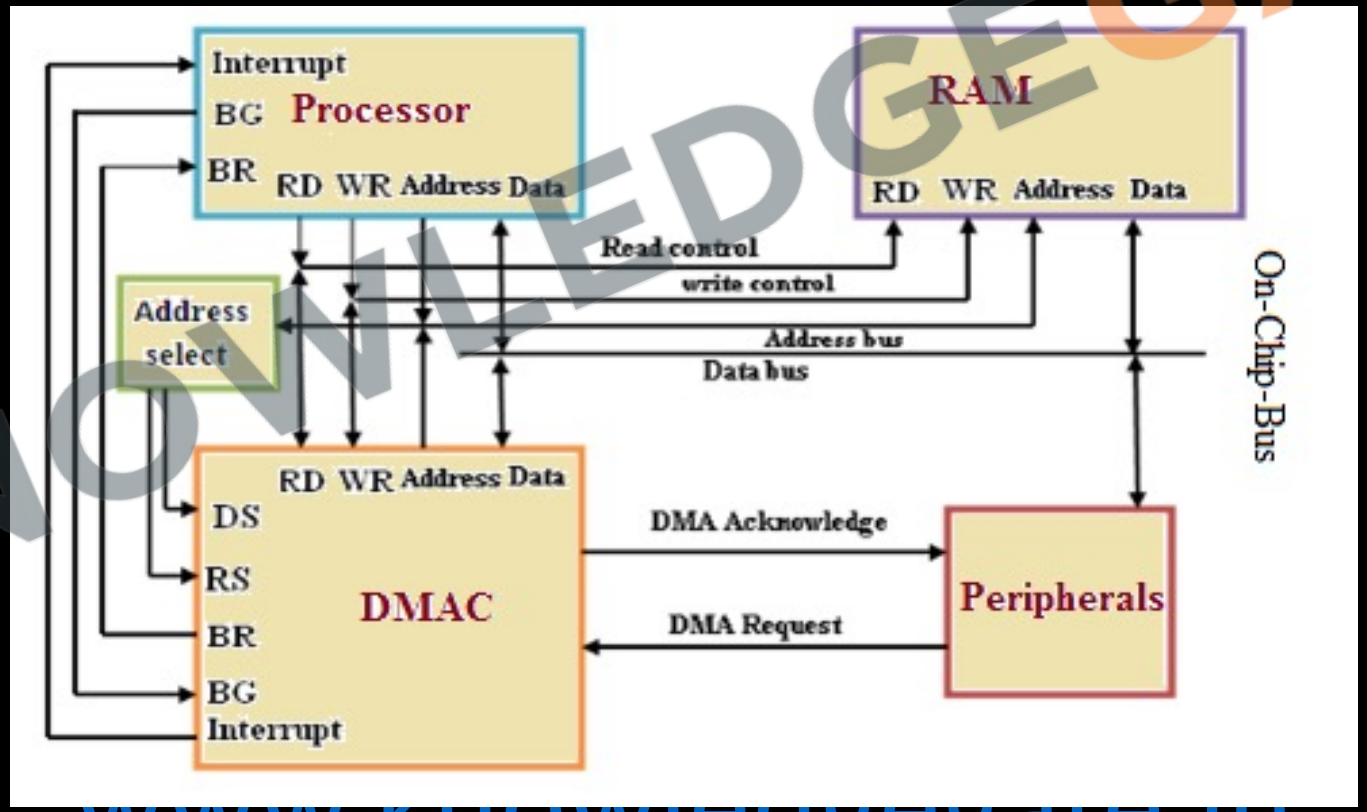
Direct Memory Access (DMA)

- When we want to perform i/o operations then the actual source or destination is either i/o device or memory, but CPU is placed in between just to manage and control the transfer.
- **DMA is** the idea where we use a new device is call DMA controller using which CPU allow DMA controller to take control of system buses and perform direct data transfer either from device to memory or from memory to device.



Sequence of DMA transfer: -

- **Step 1: -** I/O device will send a DMA request to DMA controller to perform an i/o operation.
- **Step 2: -** DMA controller will set interrupt and bus request line to 1.
- **Step 3: -** CPU using address bus will select device and registers and then will initiate i/o address register(location), counter (no of words)
- **Step 4: -** CPU will put on the bus grant line to tell DMA controller, now you are the master of system buses.
- **Step 5: -** now DMA controller will put 1 in DMA acknowledgement and using read/write control lines and address line will perform i/o directly between memory and device.

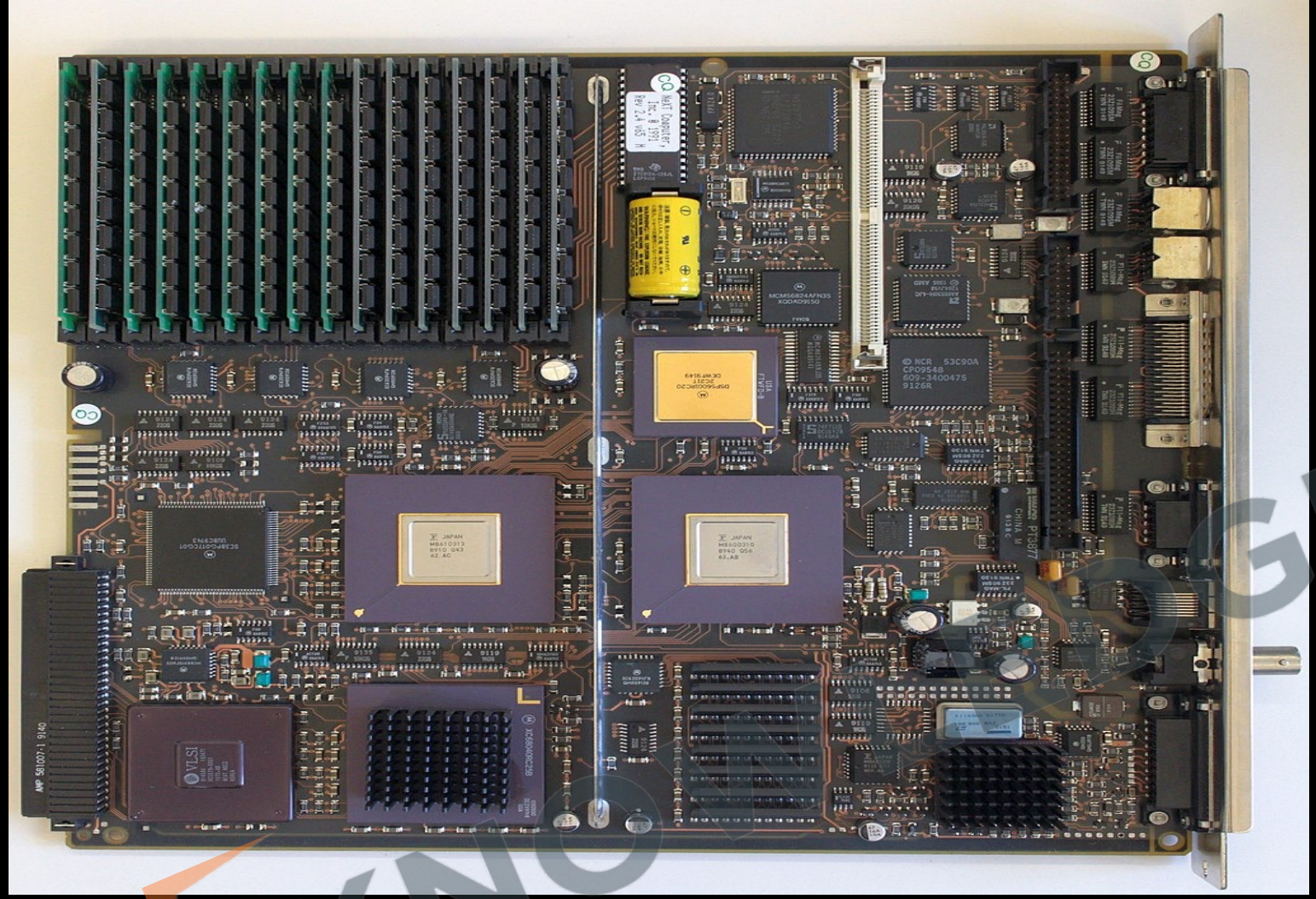


Mode of transfer: -

- **Burst mode: -** when entire i/o transfer is completed and then control comes back to CPU then it is called burst mode transfer. i.e. with high speed device like magnetic disk.
- **Cycle stealing mode: -** When CPU executes an instruction then normally their could following phases.
  - **IF –** Instruction Fetch
  - **ID –** Instruction Decode
  - **OF –** Operand fetch
  - **IX –** Instruction execute
  - **WB –** write back or store result
- Normally in ID and IE phases CPU don't require system buses and if only in that time control is giving to DMA controller then it is called cycle stealing.

- Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards.

- In the original IBM PC (and the follow-up PC/XT), there was only one Intel 8237 DMA controller capable of providing four DMA channels (numbered 0–3).

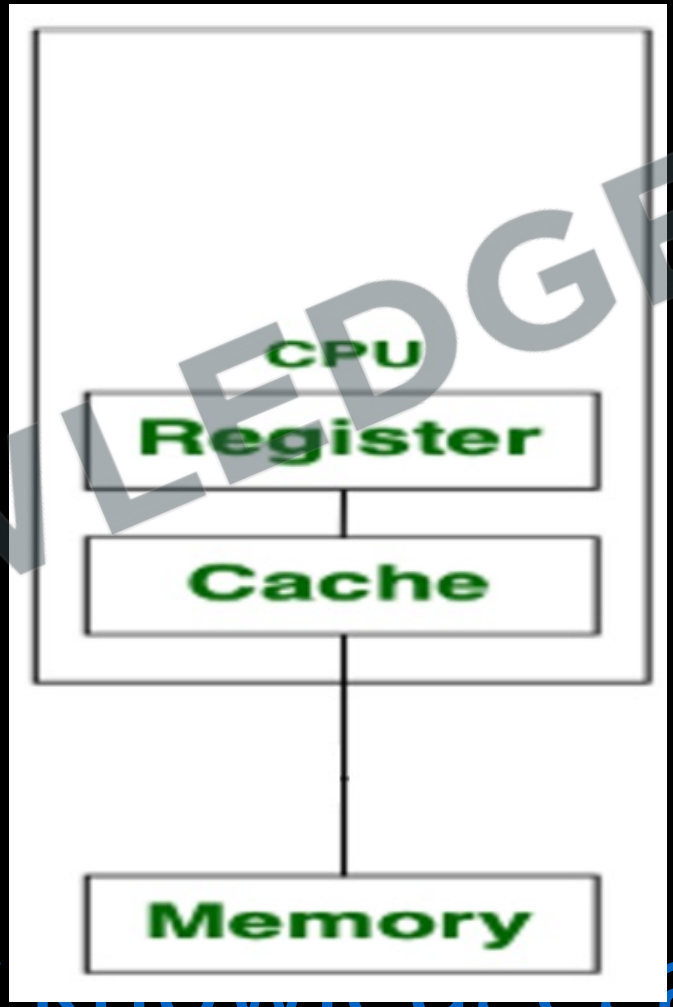


- Motherboard of a NeXTcube computer (1990).

- The two large integrated circuits below the middle of the image are the DMA controller (l.) and - unusual - an extra dedicated DMA controller (r.) for the magneto-optical disc used instead of a hard disk drive in the first series of this computer model.

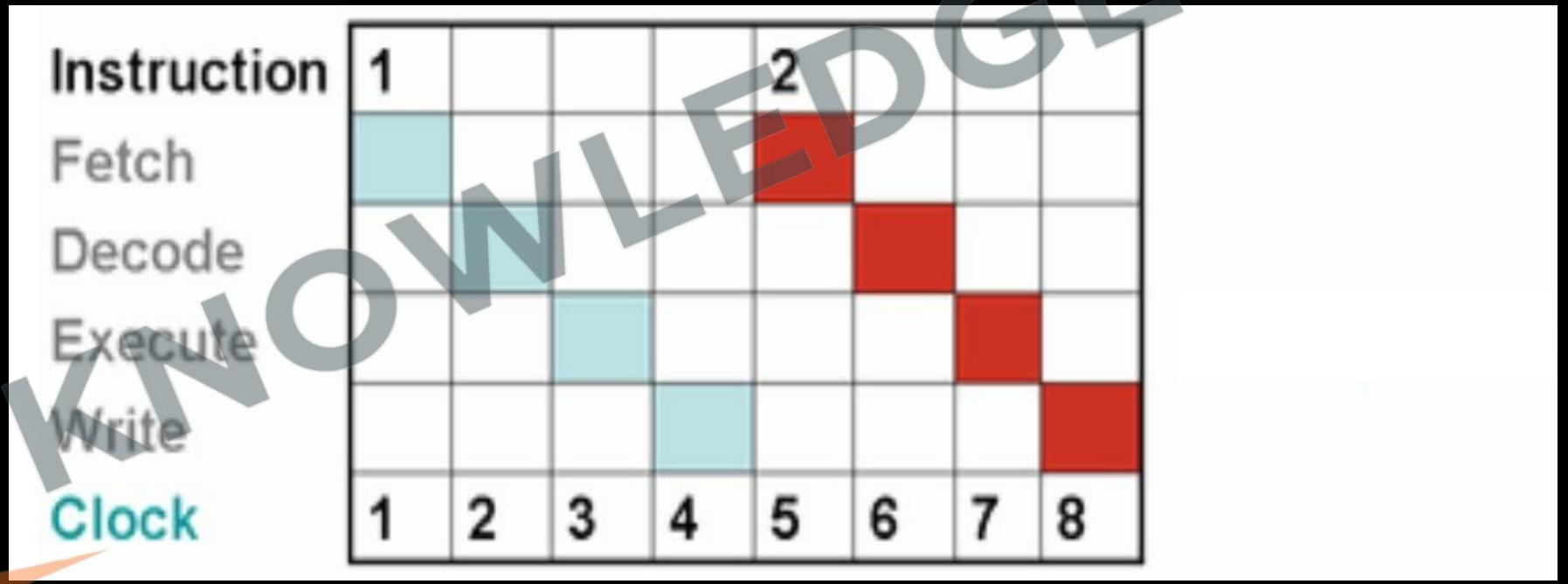
Uniprocessing

- If the system has only one processor then at most one instruction can be executed at a time.



- When we actually execute an instruction, it is executed into number of phases, like
  - Instruction Fetch
  - Instruction Decode
  - Instruction executes
  - Instruction Store

- Where when one phase is completed then only we start with next phase.



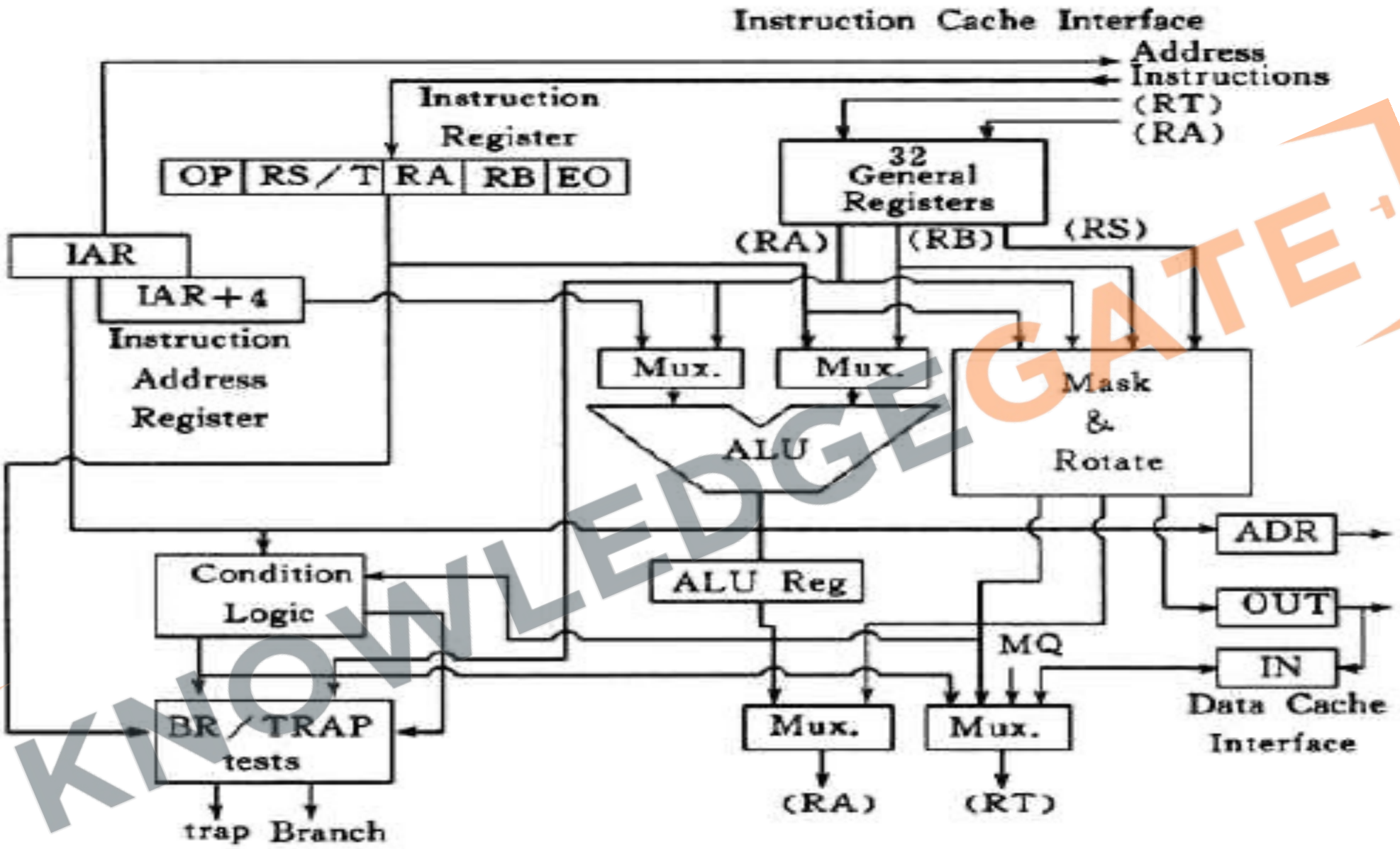
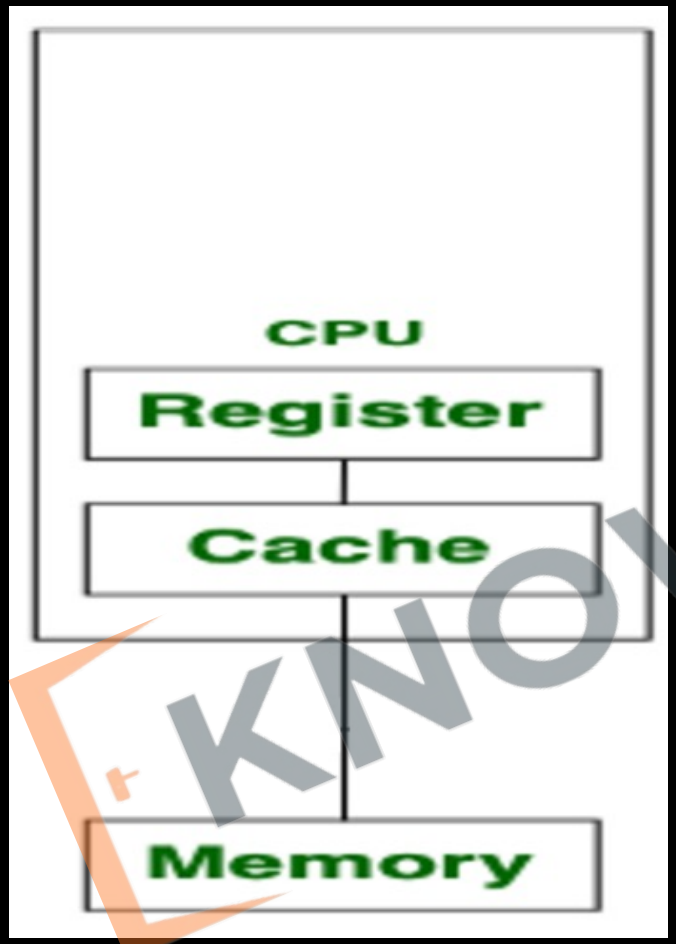
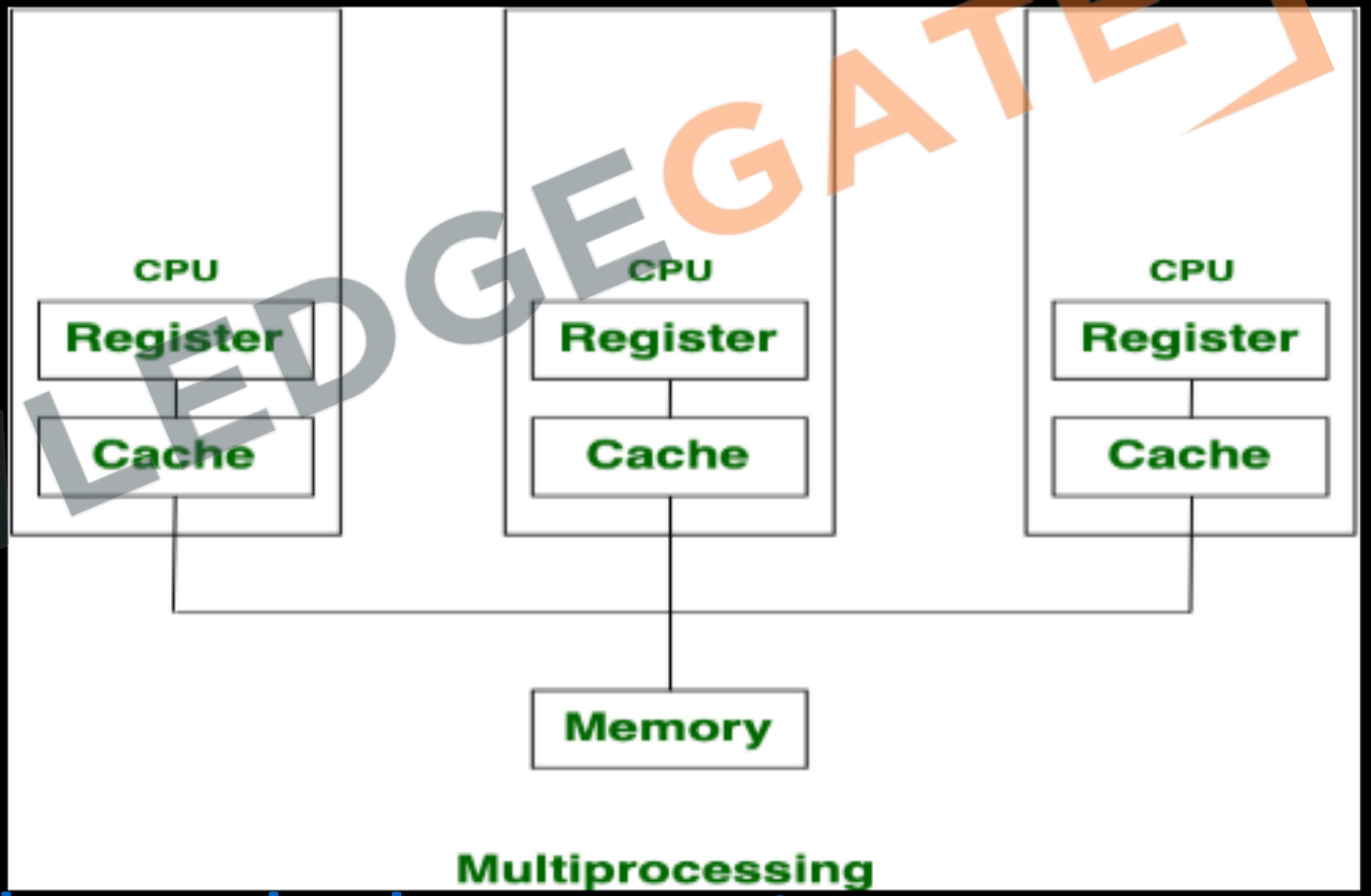


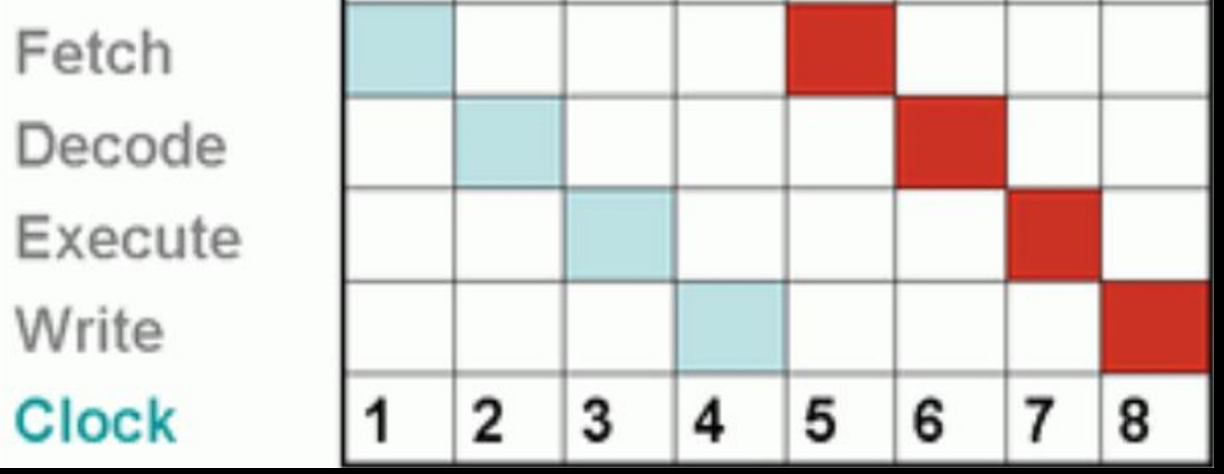
Fig. 3-2 IBM 801 Architecture

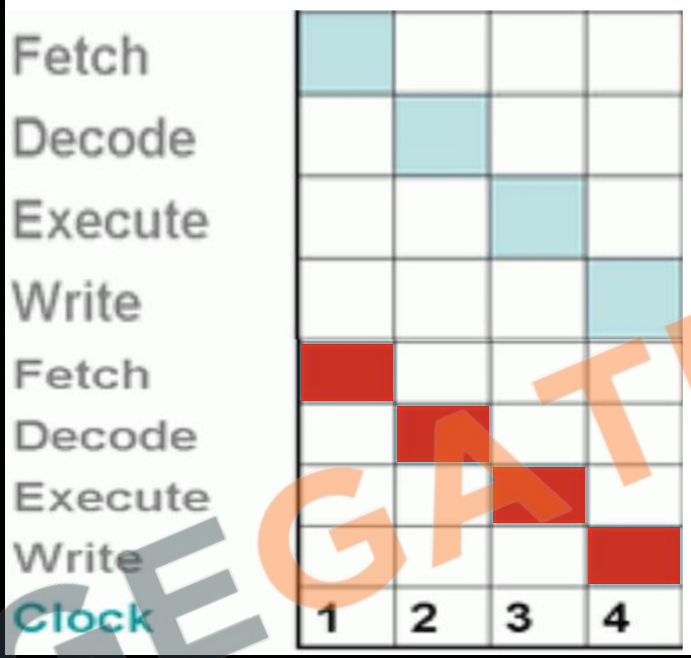
Uniprocessing vs Multiprocessing

- If we really want to execute multiple instruction together or concurrently then we must have multiple processors.









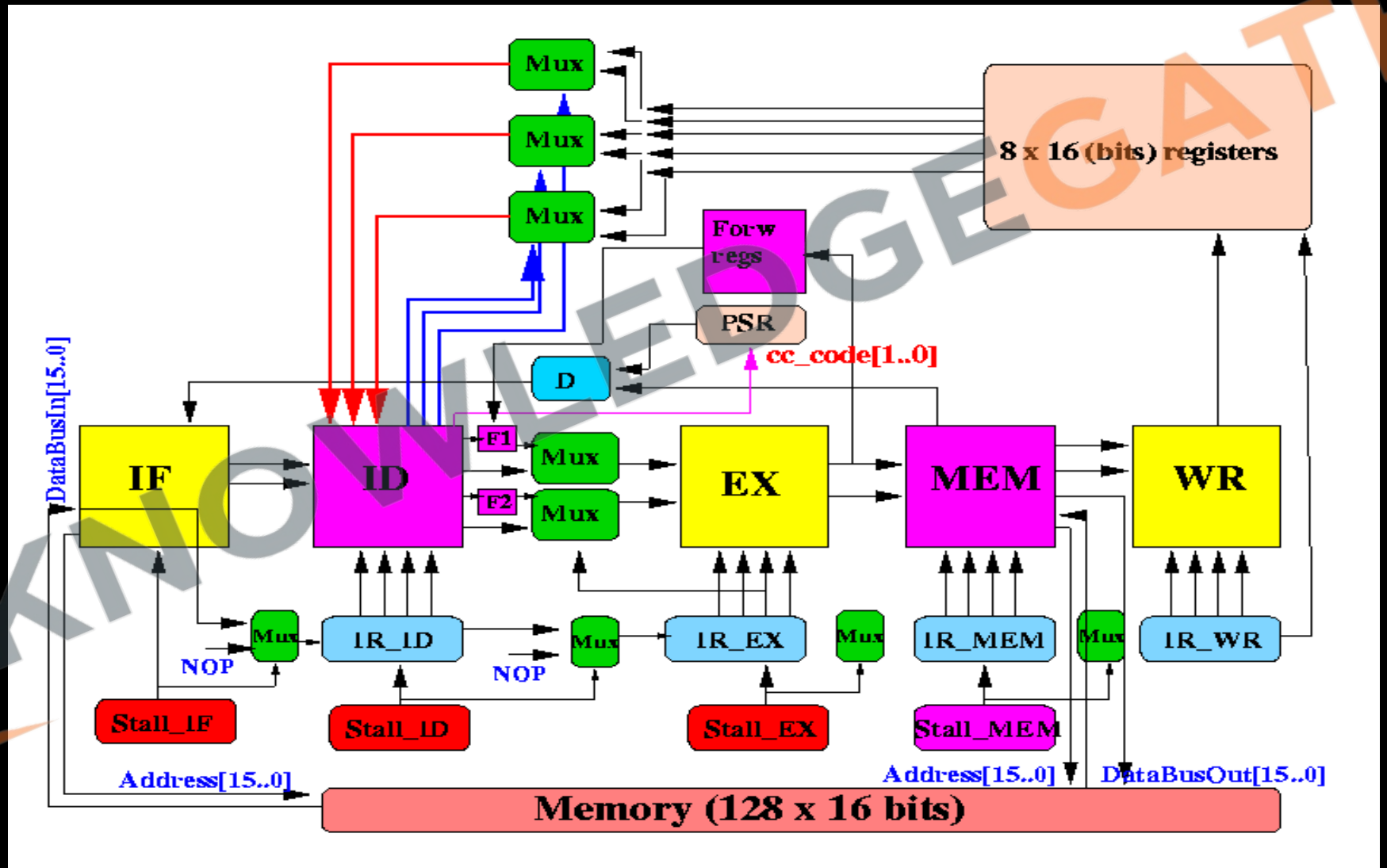
Processor$_1$

Processor$_2$

Pipelining

- Pipelining is a phenomena or method using which, we will able to run more than one instruction at the same time, on a single processor.

- The idea if make special processor (pipelined processor), where the circuit of every phase is different and buffers are placed between stages. Then we can start executing next instruction before completing all the phases of the current executing one. This idea is called pipelining.



- Note: Hardware architecture of non-pipelined and pipelined processor are different.

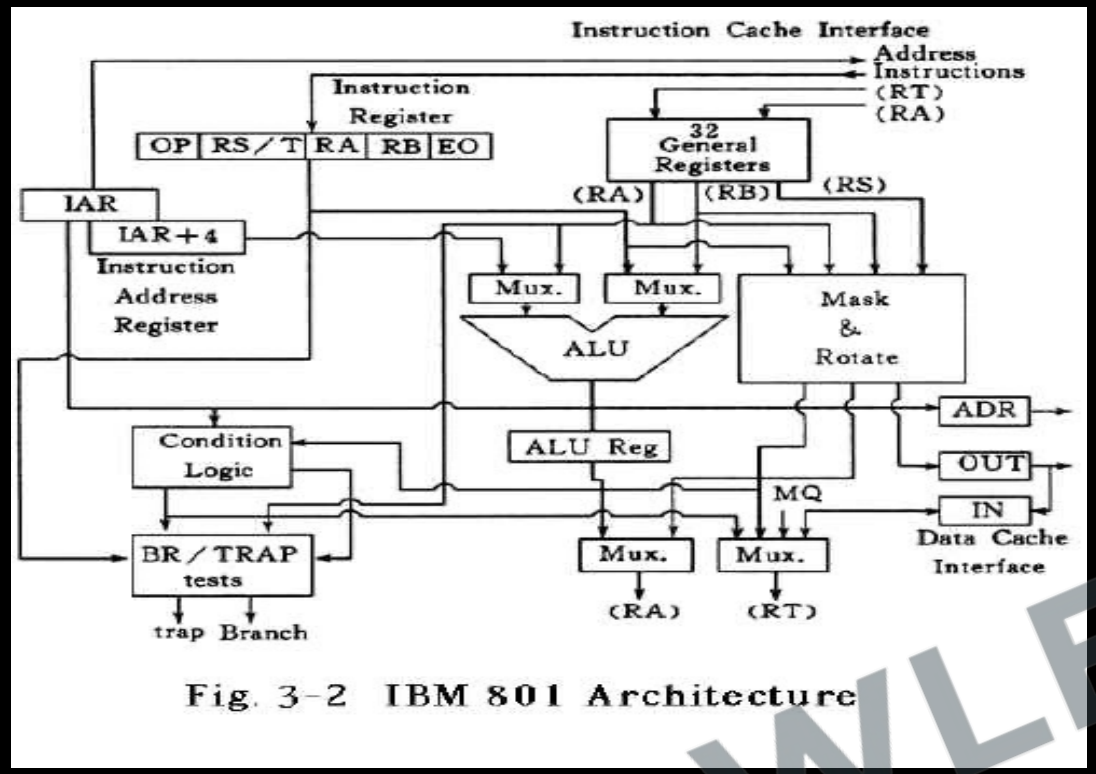
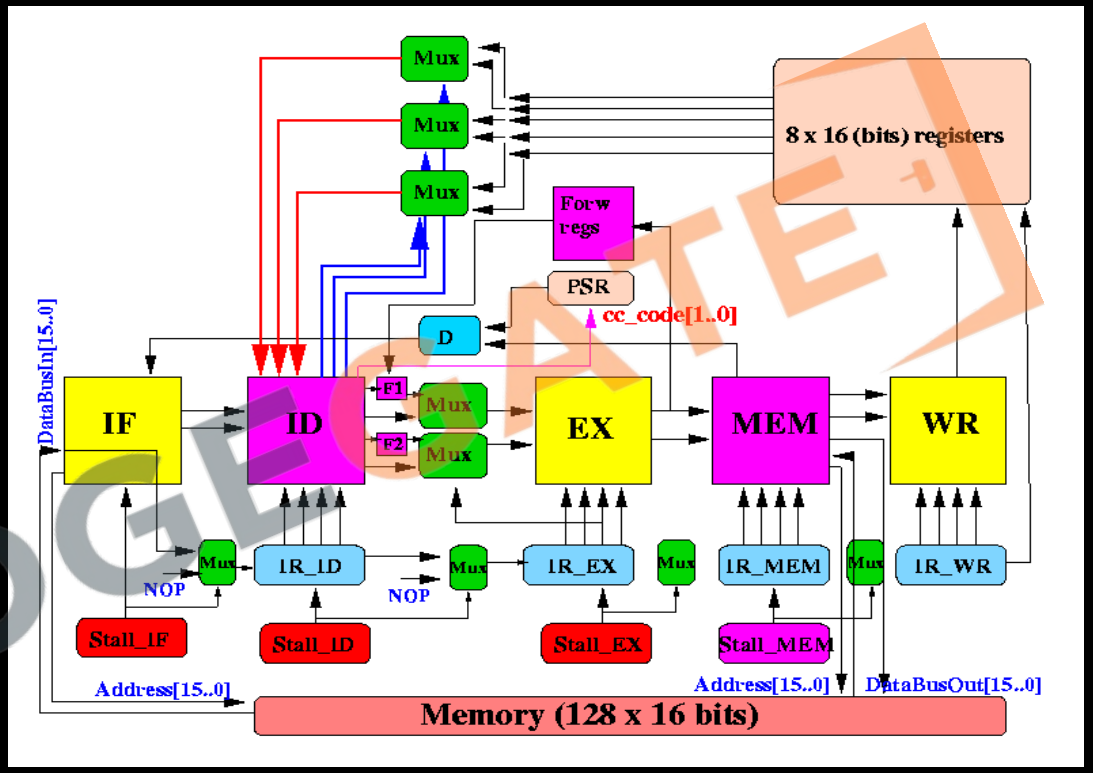
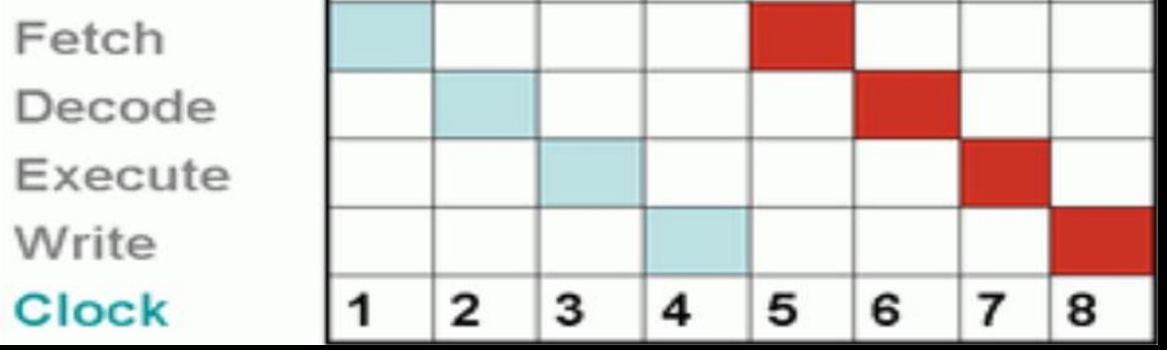


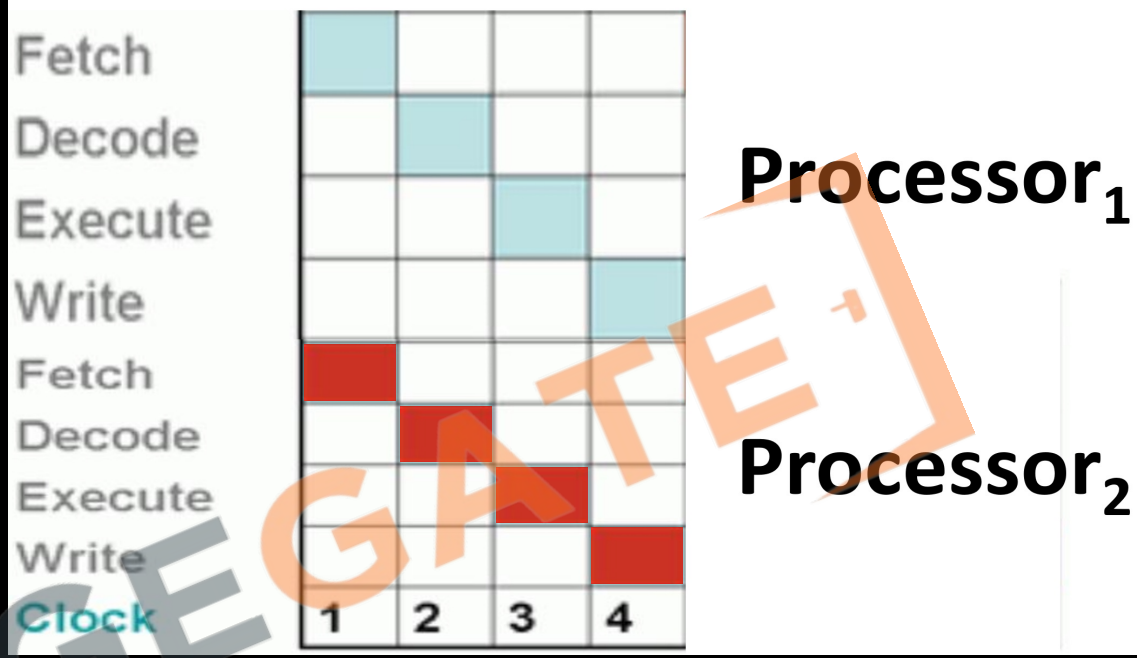
Fig. 3-2 IBM 801 Architecture



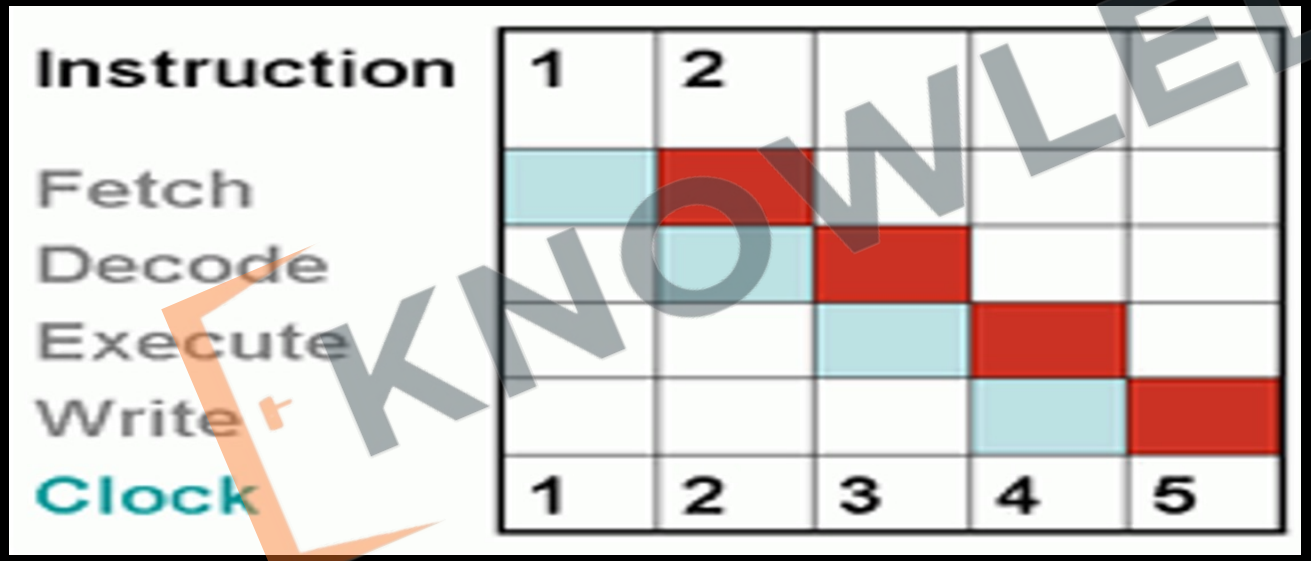
www.knowledgegate.in



Uniprocessing



Multiprocessing



Pipelining

Q Consider a system where clock is triggering at a speed of 1MHz (1 clock = 1 µs). In a pipelined processor there are 4 stages and each stage take only 1 clock, if a program has 10 instruction then it will take what time?

- On a non-pipelined processor

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
IF	I_1				I_2				I_3				I_4				I_5				I_6				I_7				I_8				I_9				I_{10}			
ID		I_1				I_2				I_3				I_4				I_5				I_6				I_7				I_8				I_9				I_{10}		
EX			I_1				I_2				I_3				I_4				I_5				I_6				I_7				I_8				I_9				I_{10}	
WB				I_1				I_2				I_3				I_4				I_5				I_6				I_7				I_8				I_9				I_{10}

- If all instructions are identical (time taken for specific phase is same for all instruction)

- Time without pipeline (T_{wp}) =
  - (sum of clocks for each phase of one instruction) *(no of instruction)*time of one clock

- If each phase requires same clock usually one (as we set the frequency in such a way)
  = (no of phases * no of instruction) * time of one clock

Q Consider a system where clock is triggering at a speed of 1MHz (1 clock = 1 µs). In a pipelined processor there are 4 stages and each stage take only 1 clock, if a program has 10 instruction then it will take what time?

- On a pipelined processor

	1	2	3	4	5	6	7	8	9	10	11	12	13
IF	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}			
ID		I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}		
EX			I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	
WB				I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}

- If all instructions are identical (time taken for specific phase is same for all instruction)

- If each phase requires same clock usually one (as we set the frequency in such a way)

- Time with pipeline (T_p) = ((no of phase) + (no of instruction - 1)) *time of one clock

- Speed up = (Time without pipeline (T_{wp}))/ (Time with pipeline (T_p)) = 40/13

- Max Speed up = no of stages = (In this case 4)

- Efficiency = (speed up/max speed up) * 100 = 76.9%

N = 10

- Speed up = (Time without pipeline (T_{wp}))/ (Time with pipeline (T_p)) =

- Max Speed up = no of stages = (In this case 4)

- Efficiency = (speed up/max speed up) * 100 = 97.08%

N = 100

- Speed up = (Time without pipeline (T_{wp}))/ (Time with pipeline (T_p)) = 40000/10003

- Max Speed up = no of stages = (In this case 4)

- Efficiency = (speed up/max speed up) * 100 = 99.97%

N = 10,000

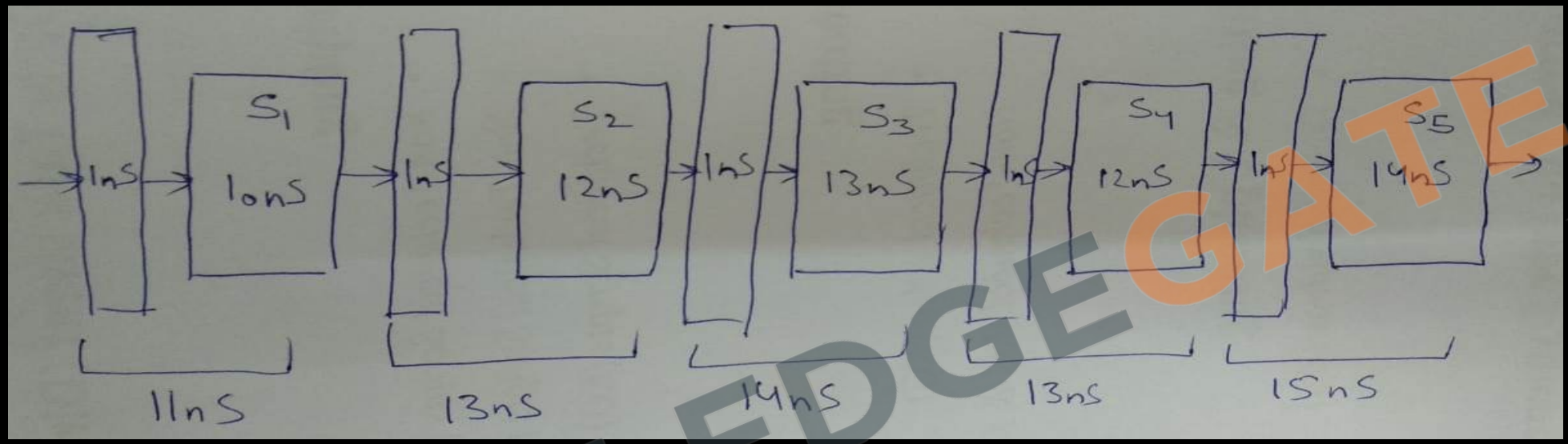
Q Consider a system where we have ‘m’ stages and program contains ‘n’ instruction such that m<<n, then find speed up?

Q Consider 5 instruction with following clock requirement?

	F	D	E	WB
I_1	1	2	1	1
I_2	1	2	2	1
I_3	2	1	3	2
I_4	1	3	2	1
I_5	1	2	1	2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I_1																		
I_2																		
I_3																		
I_4																		
I_5																		

Q After considering this diagram what must be the frequency of the processor to ensure that work of every stage will complete in 1 clock stage wise?

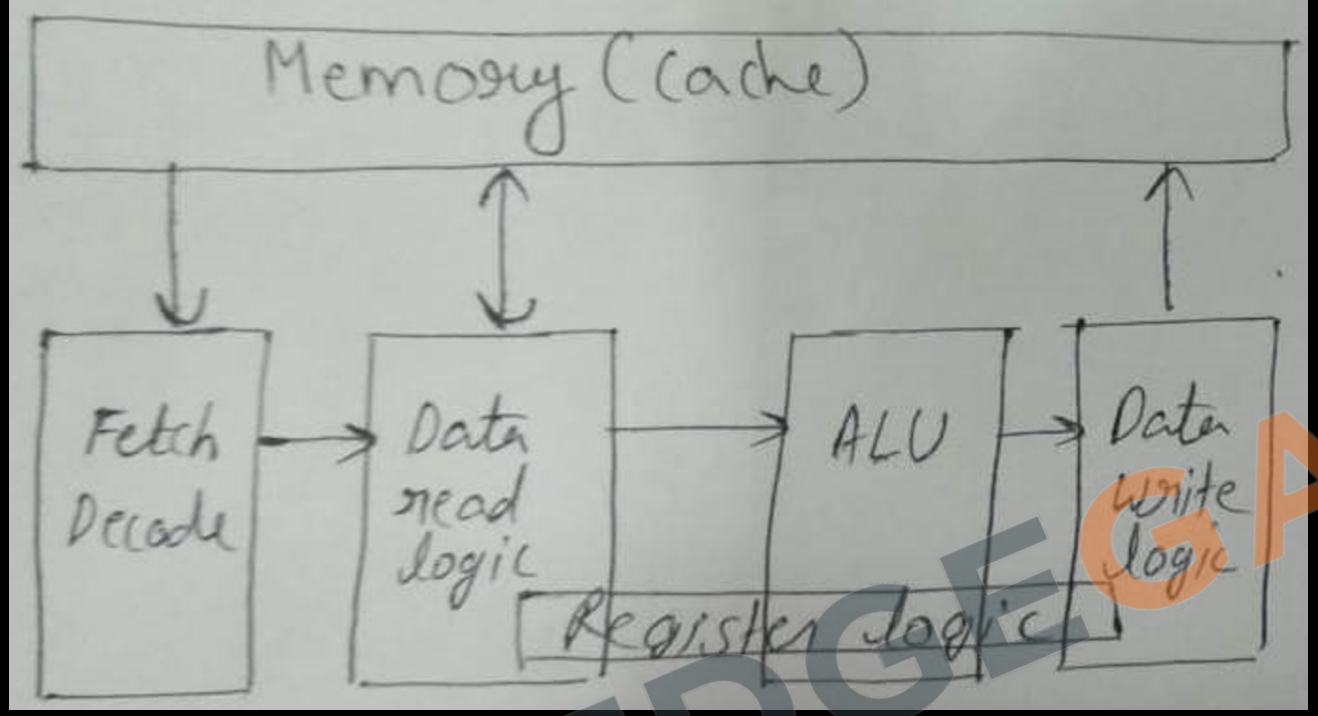


	S_1	S_2	S_3	S_4	S_5
1ns	11	13	14	13	15
2ns	6	7	7	7	8
3ns	4	5	5	5	5
5ns3	3	3	3	3	3
15ns	1	1	1	1	1

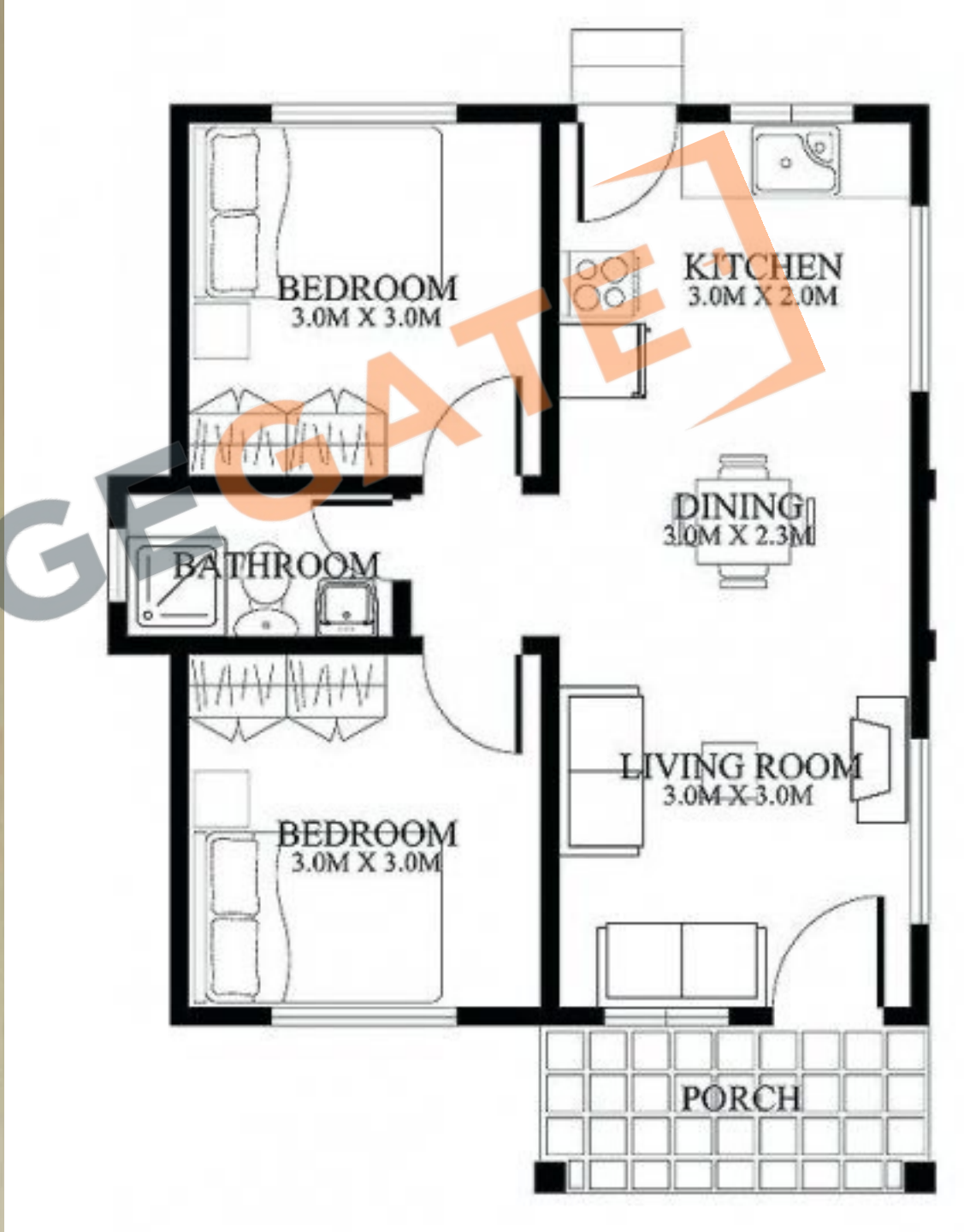
- We understand that different stages in a pipeline may have different delays, it also depends on the type of instruction that how much time a particular stage will take for a specific instruction.

- If we increase the time of a clock to the time taken by the slowest stage of the pipeline, then each instruction takes one clock then with pipe-line processor in long run, we achieve
  - CPI(Clock Per Instruction) = 1

- Sometimes we cannot execute instructions with full efficiency in a pipeline because of certain dependency or hazards.
  - **Structural hazards**
  - **Control hazards**
  - **Data hazards**

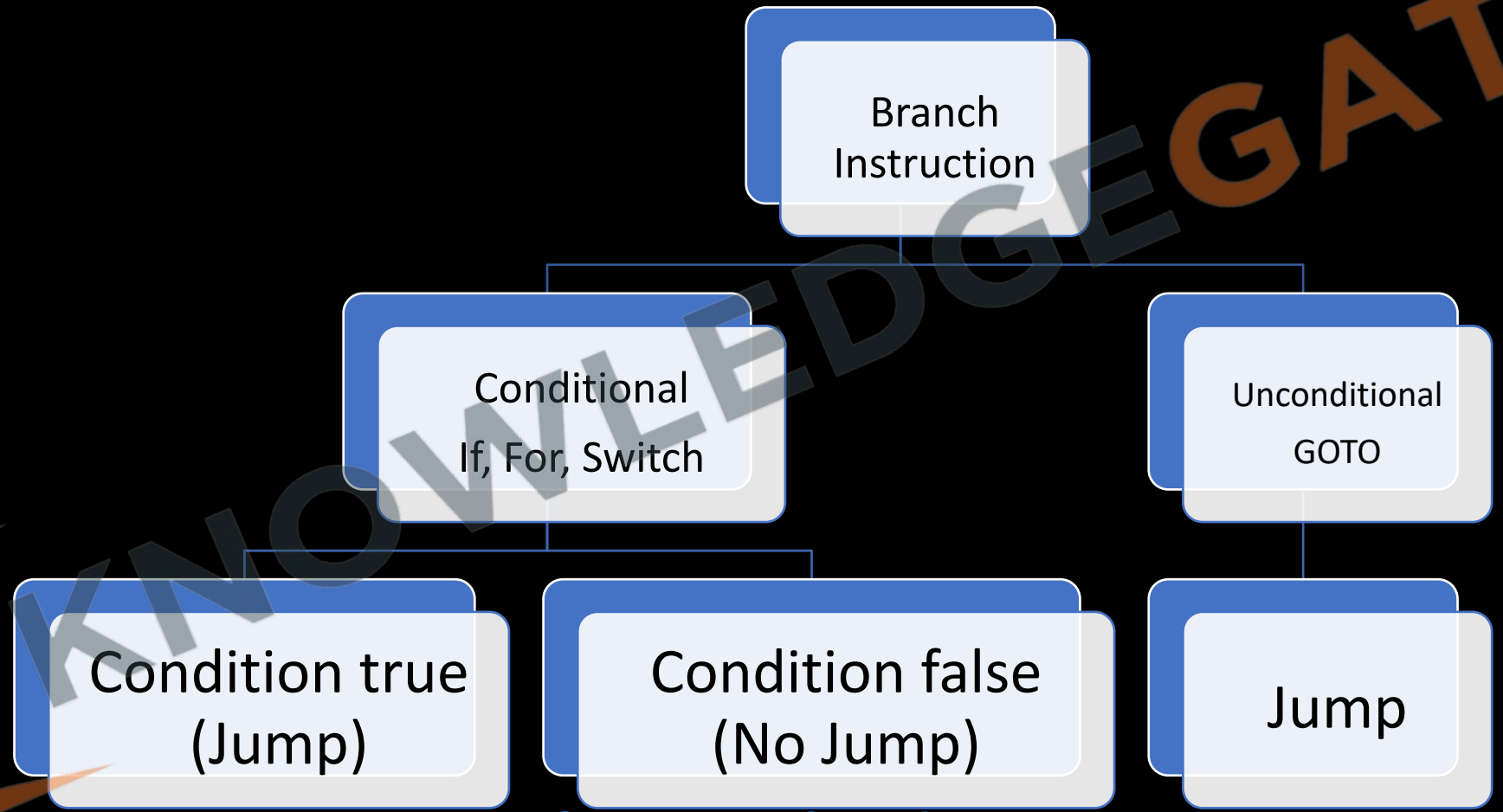


	1	2	3	4	5	6	7	8	9	10	11	12
I_1	FD	DR	ALU	DW								
I_2			FD		DR	ALU	DW					
I_3												
I_4												



- A structural hazard cannot be removed using efficient program. Therefore, one solution could be resource duplication but the cost of implementation will be very high.

- If some instruction I_1 is branch instruction and after executing the entire instruction we understand that there is jump and next instruction to be executed is I_{10}. This time we have already partially executed I_1, I_2, I_3 which is the problem.



	1	2	3	4	5	6	7	8	9	10	11	12
I_1	IF	ID	EX	WB								
I_2		IF	ID	EX	WB							
I_3			IF	ID	EX	WB						
I_4				IF	ID	EX						
I_5					IF	ID						
I_6												
I_7						IF	ID	EX	WB			
I_8							IF	ID	EX	WB		

Data Hazards

- Data hazards occur when instructions that exhibit data dependence, modify data in different stages of a pipeline. Hazard cause delays in the pipeline. E.g.

Instruction	Meaning of instruction
I_0: MUL R_2, R_0, R_1	$R_2 \neg R_0 * R_1$
I_1: DIV R_5, R_3, R_4	$R_5 \neg R_3 / R_4$
I_2: ADD R_2, R_5, R_2	$R_2 \neg R_5 + R_2$
I_3: SUB R_5, R_2, R_6	$R_5 \neg R_2 - R_6$

- There are mainly three types of data hazards: This condition is called **Bernstein condition.** RAW (Read after Write) [Flow/True data dependency].

- RAW hazard occurs when instruction J tries to read data before instruction I write it.

  I: $R_2 \leftarrow R_1 + R_3$

  J: $R_4 \leftarrow R_2 + R_3$

Here, J is trying to read R_2 before I have written it.

- WAR (Write after Read) [Anti-Data dependency]

- WAR hazard occurs when instruction J tries to write data before instruction, I read it.

  I: $R_2 <- R_1 + R_3$

  J: $R_3 <- R_4 + R_5$

Here, J is trying to write R_3 before I have read it.

- WAW (Write after Write) [Output data dependency]

- WAW hazard occurs when instruction J tries to write output before instruction, I write it.

  I: R_2 <- R_1 + R_3

  J: R_2 <- R_4 + R_5

Here J is trying to write R_2 before I.

WAR and WAW hazards occur during the out-of-order execution of the instructions.

We say that instruction S_2 depends in instruction S_1, when: This condition is called **Bernstein condition.**

Three cases exist:

Flow (data) dependence: $O(S_1) \cap I(S_2)$, $S_1 \rightarrow S_2$ and S_1 writes after something read by S_2

Anti-dependence: $I(S_1) \cap O(S_2)$, $S_1 \rightarrow S_2$ and S_1 reads something before S_2 overwrites it

Output dependence: $O(S_1) \cap O(S_2)$, $S_1 \rightarrow S_2$ and both write the same memory location.

$$[I(S1) \cap O(S2)] \cup [O(S1) \cap I(S2)] \cup [O(S1) \cap O(S2)] \neq \phi$$

Solution of Data dependency

- We can use code movement or code relocation and can execute the dependent instruction after some time.
- Here we can use operator forwarding using which we can directly access the result after execution instead of waiting that it gets store in memory.

Q A 5-stage pipelined processor has Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO)and Write Operand (WO)stages. The IF, ID, OF and WO stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD and SUB instructions,3 clock cycles for MUL instruction, and 6 clock cycles for DIV instruction respectively. Operand forwarding is used in the pipeline. What is the number of clock cycles needed to execute the following sequence of instructions?

Instruction	Meaning of instruction
I_0: MUL R_2, R_0, R_1	$R_2 \neg R_0 * R_1$
I_1: DIV R_5, R_3, R_4	$R_5 \neg R_3 / R_4$
I_2: ADD R_2, R_5, R_2	$R_2 \neg R_5 + R_2$
I_3: SUB R_5, R_2, R_6	$R_5 \neg R_2 - R_6$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IF	ID	OF	PO	PO	PO	WO													
	IF	ID	OF			PO	PO	PO	PO	PO	PO	WO							
		IF	ID										OF	PO	WO				
			IF	ID												OF	PO	WO	

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IF	ID	OF	PO	PO	PO	WO													
	IF	ID	OF			PO	PO	PO	PO	PO	PO	WO							
		IF	ID									PO	WO						
			IF	ID									PO	WO					

HOW IT WORKS

The plant

Tea leaves grow on bushes in vast crops. If left untended the tea plant could reach up to 20m (65ft) tall, but the bushes are usually pruned at the 'plucking table' around 1.2m (4ft) – this helps hand-picking and promotes bud growth.

Plucking

Leaves are never plucked from the plant individually; they are always removed as a group of one, two or three leaves along with the bud that forms at the end of the stem.

Withering

The freshly picked leaves are laid out in large troughs or shelves to wither for eight to 12 hours. Air is often passed through in order to help the removal of moisture, and after withering the leaves look wilted.

Rolling

The leaves are broken up and the enzymes are released in preparation for oxidation. There are two rolling methods: Orthodox, where rollers gently break leaves; and CTC – cut, tear, curl – where leaves are cut by a machine.

Packaging

After the tea is dried, it is sorted into grades depending on the dried leaf's size. Larger leaves are sold for loose-leaf tea and smaller ones are prepped for use in tea bags.

The cup

The tea is then ready for brewing. The dried tea leaves infuse hot water with the delicate taste that's governed by the growing conditions and careful preparation process.
Pop the kettle on!

Drying

Tea leaves are then dried in order to stop the oxidation process at precisely the right time to make sure the tea's flavour is just right. The oxidised leaves are gently heated to remove all excess moisture.

Oxidising

The withered and rolled tea leaves are laid out for a few hours to oxidise, which means they react with oxygen and begin to ferment. The leaves undergo chemical processes where they partially break down.