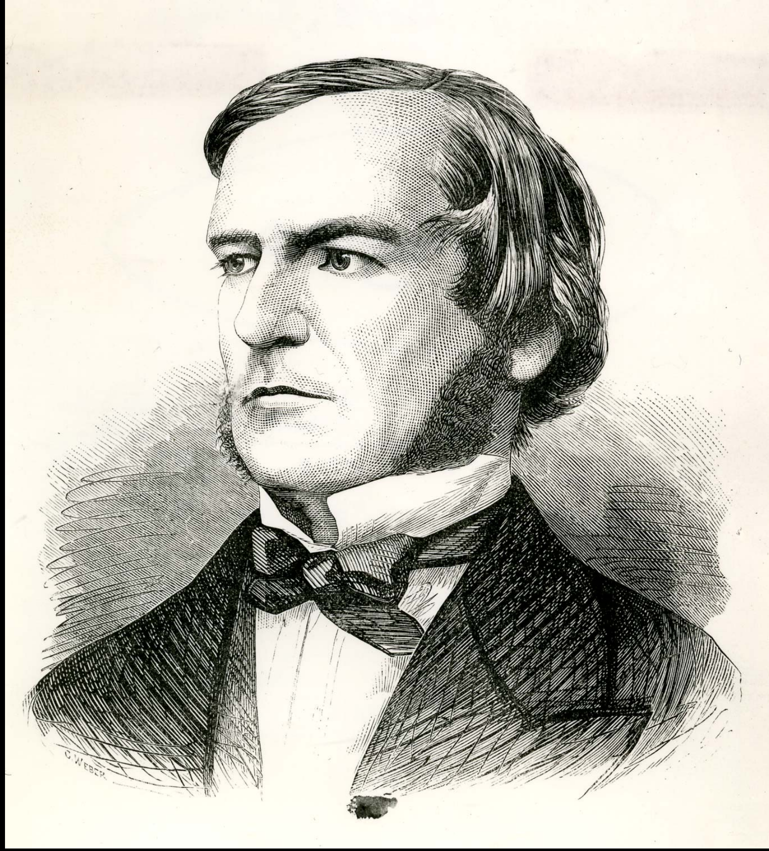
Boolean algebra

- 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.

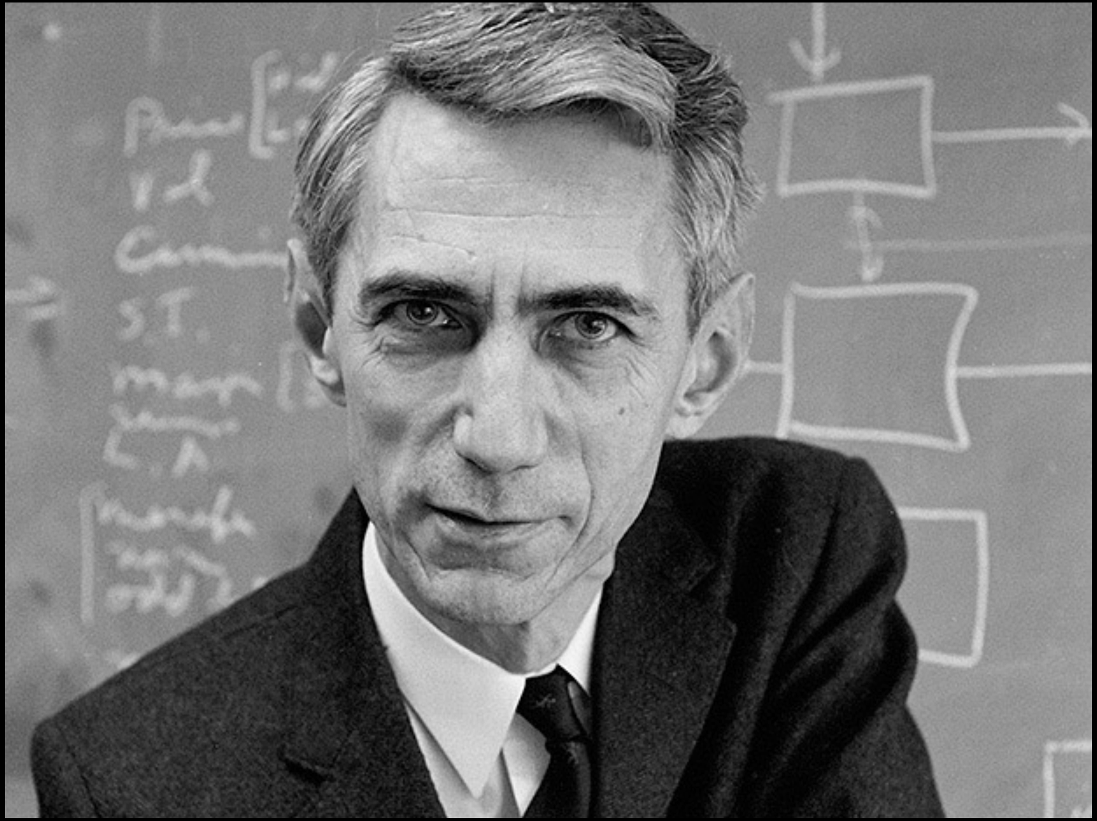
2. Instead of elementary algebra where the values of the variables are numbers, and the prime operations are addition and multiplication. The main operations of Boolean algebra are
   - The conjunction and denoted as ∧
   - The disjunction or denoted as ∨
   - The negation not denoted as ¬

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.

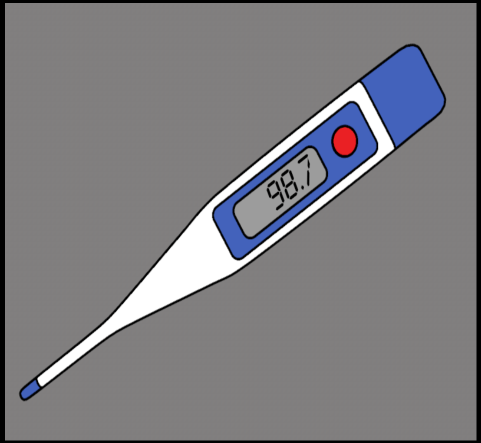
- Boolean constants are denoted by 0 or 1. Boolean variables are quantities that can take different values at different times. They may represent input, output or intermediate signals.

- 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.

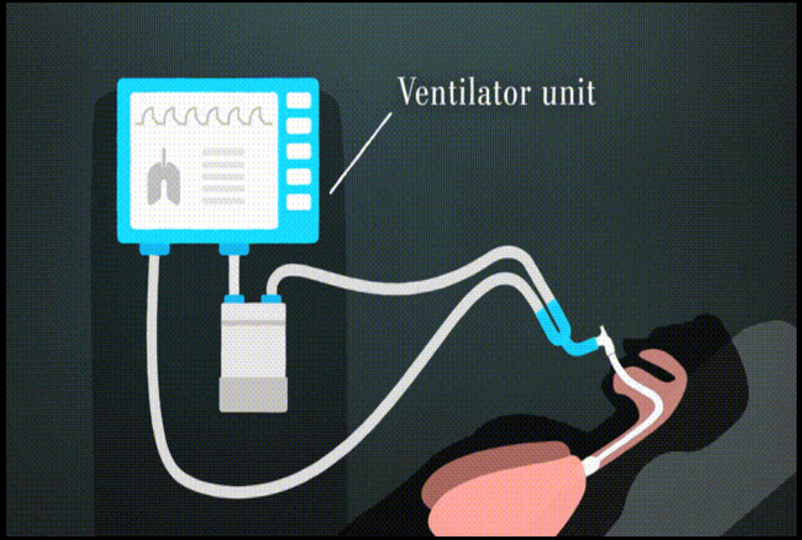
- Historically there have been 2 types of Computers:
    - **Fixed Program Computers / Dedicated device / Embedded system – **Their function is very specific and they couldn’t be reprogrammed, e.g. Calculators washing machine.
    - **Stored Program Computers / General purpose computer / von Neumann 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.

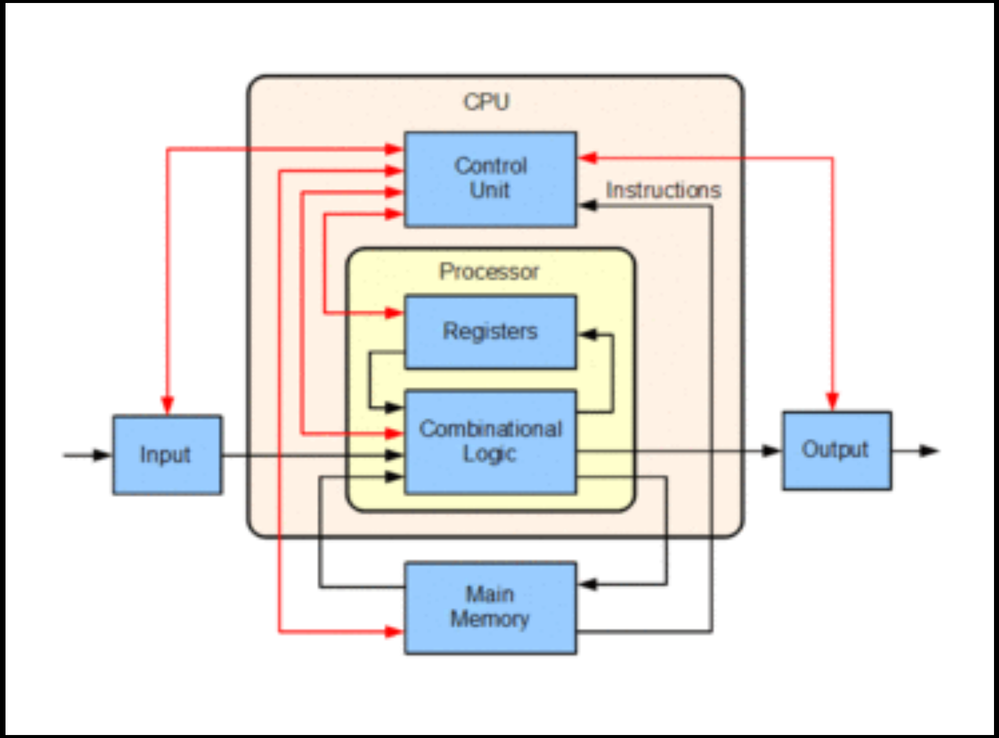


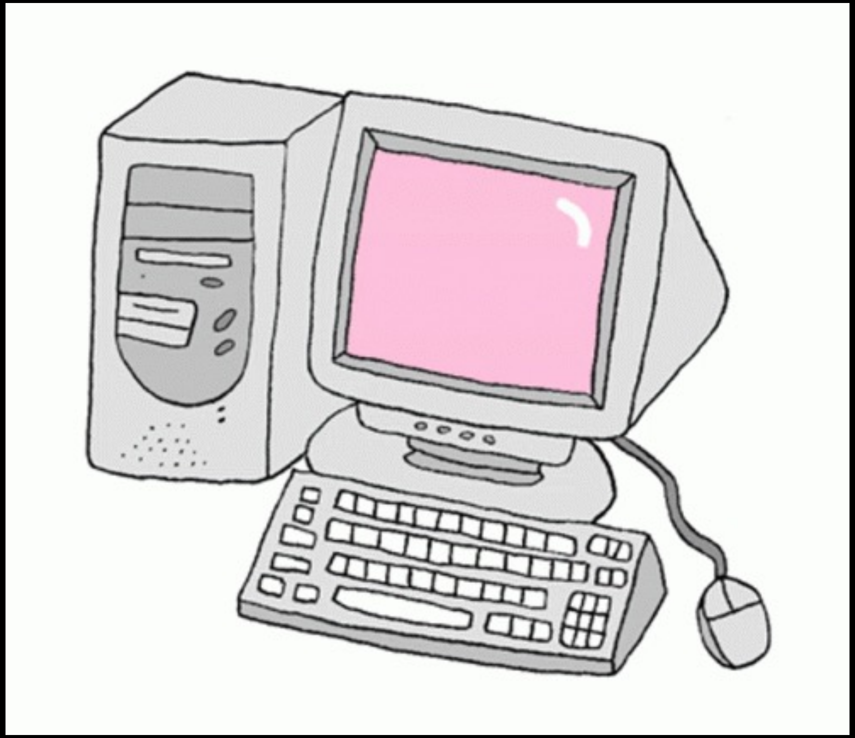
- Embedded system are kind of simple computers designed to do Specific task for e.g. microwave, washing machine etc., they have small microprocessors inside them. These microprocessors are hardwired and can not do everything, except specific task.



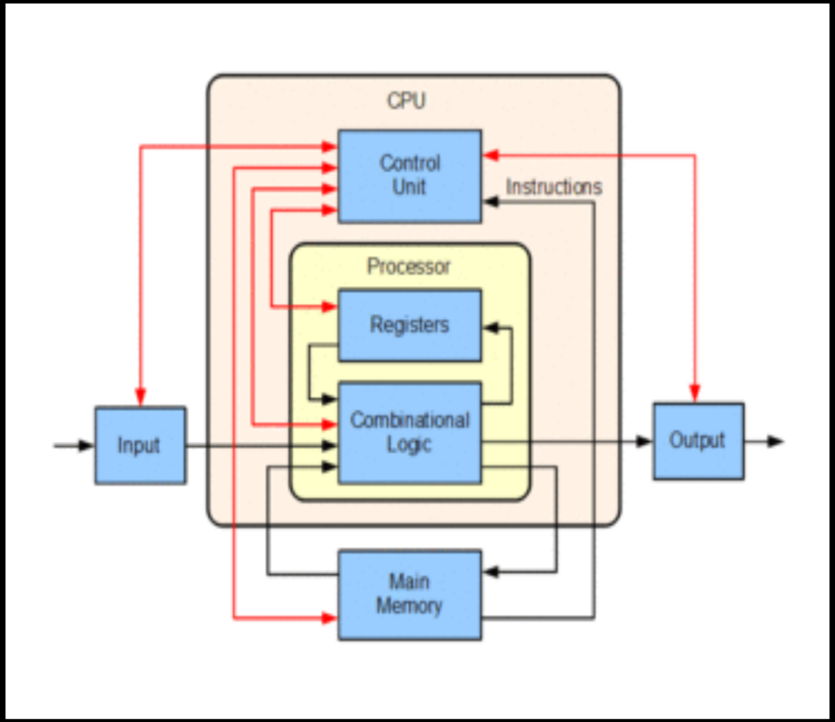
Stored Program Computers / General purpose computer / von Neumann architecture

- 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.

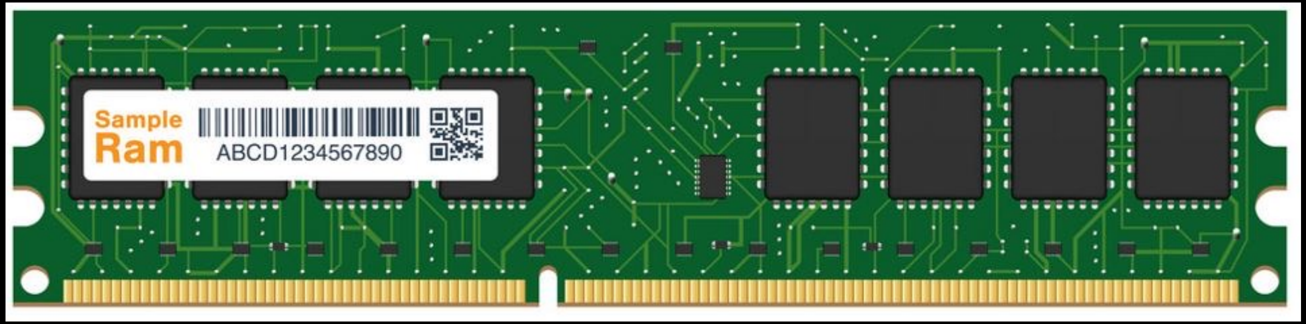




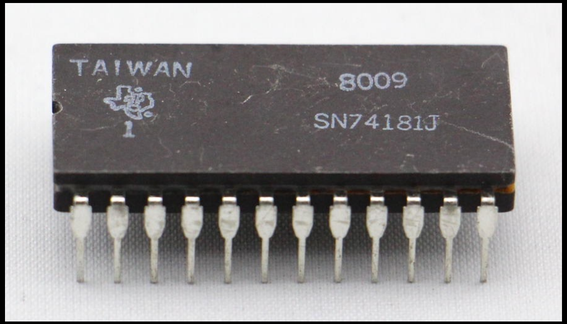
- **Machine / CPU architecture: -** In general a CPU contain 3 important components
  - **Control unit: -** Which generates control signal (master generator) to control every other part of the CPU. It directs all input and output flow, fetches code for instructions, and controls how data moves around 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)

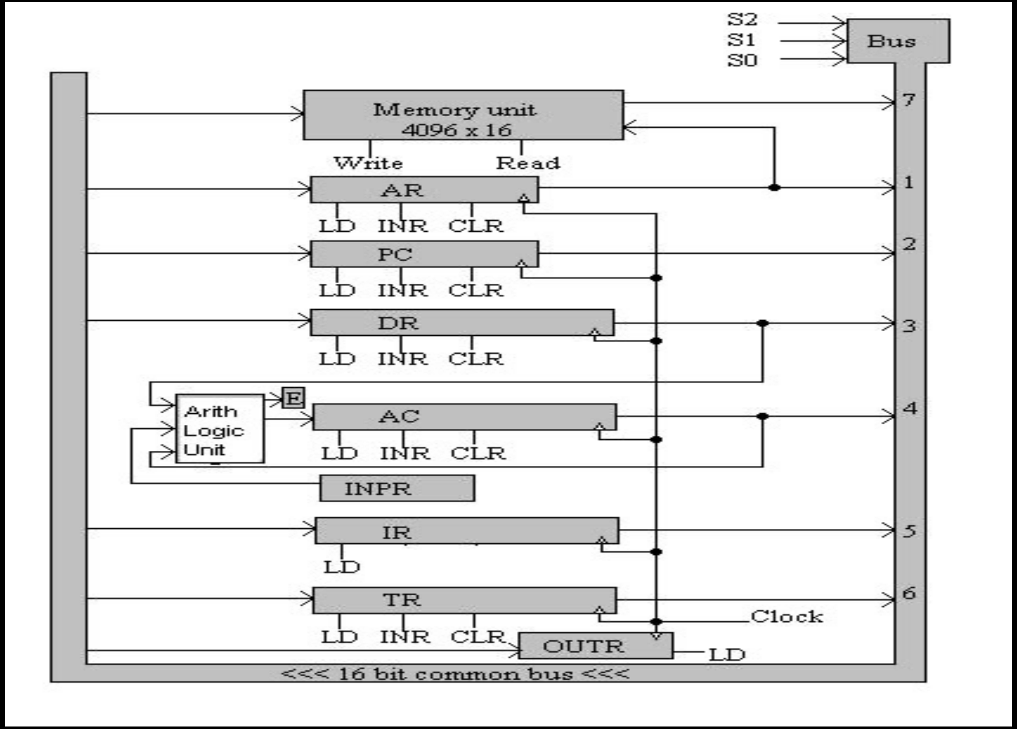


  - ALU (circuit which perform operations)



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

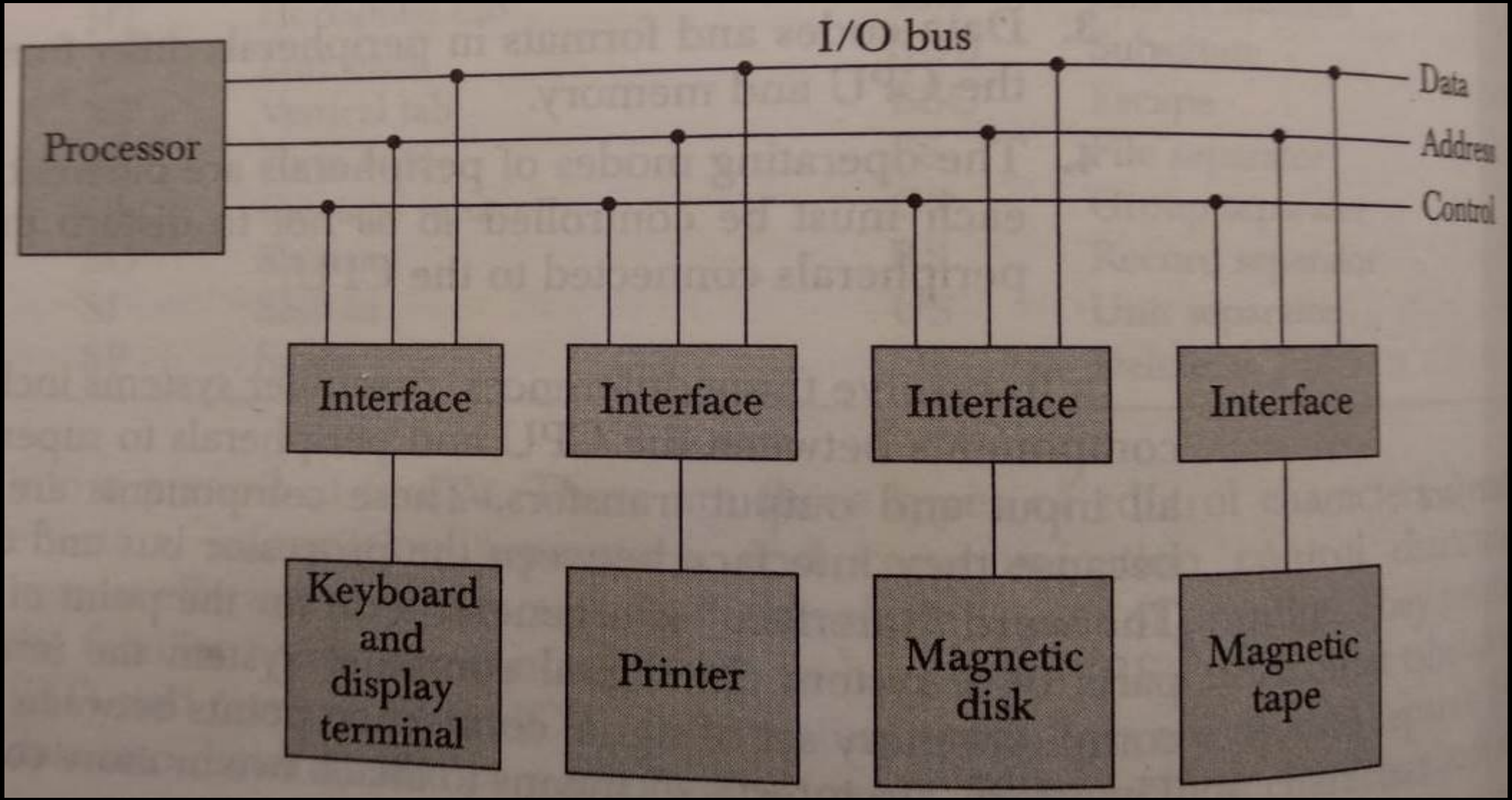
- Timing circuit (sequence counter) (to order certain operations, like fetch, decode, execute), will generate timing signals
- Control unit will generate control signals-to select registers, select other circuit, to give inputs to registers, So we need a special unit called control unit, which will give signals to all the components
- Flags – one-bit information
- Bus – using which we will connect different component together, and perform data transfer using multiplexer



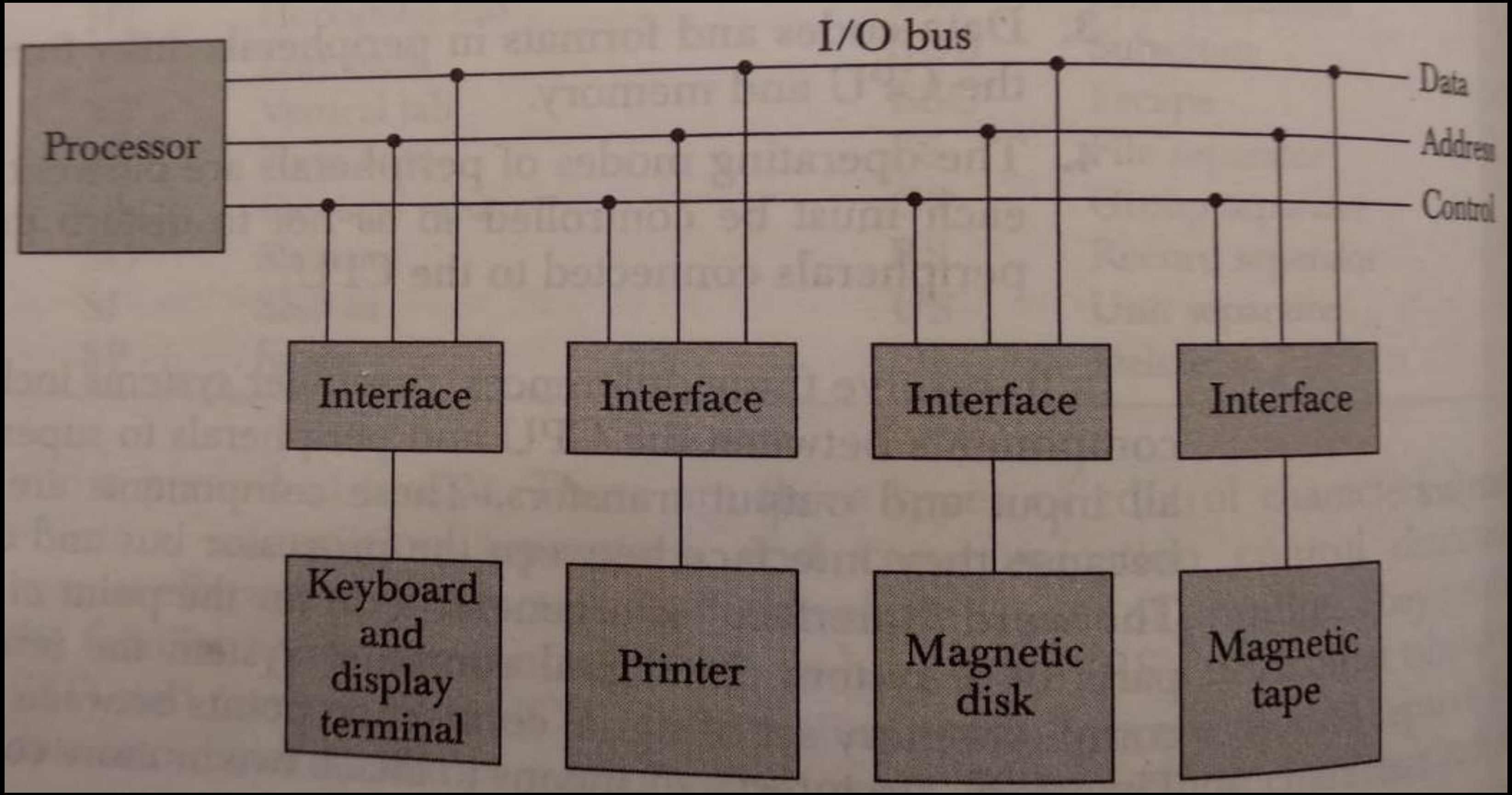
- how in general operation are performed?

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

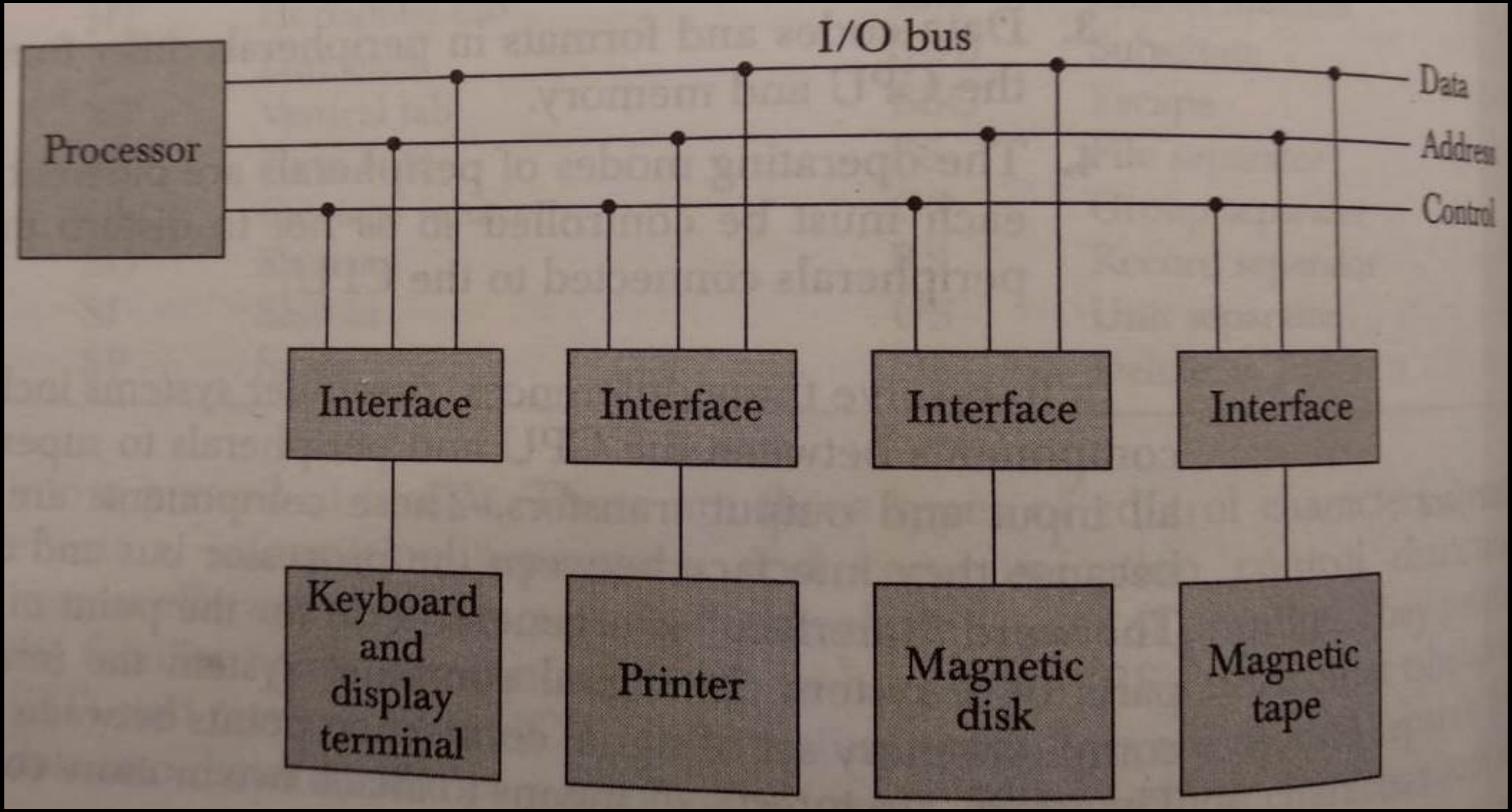
- **Address bus: -** Is used to identify the correct i/o devices among the number of i/o device , so CPU put an address of a specific i/o device on the address line, all devices keep monitoring this address bus and decode it, and if it is a match then it activates control and data lines.



- **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.



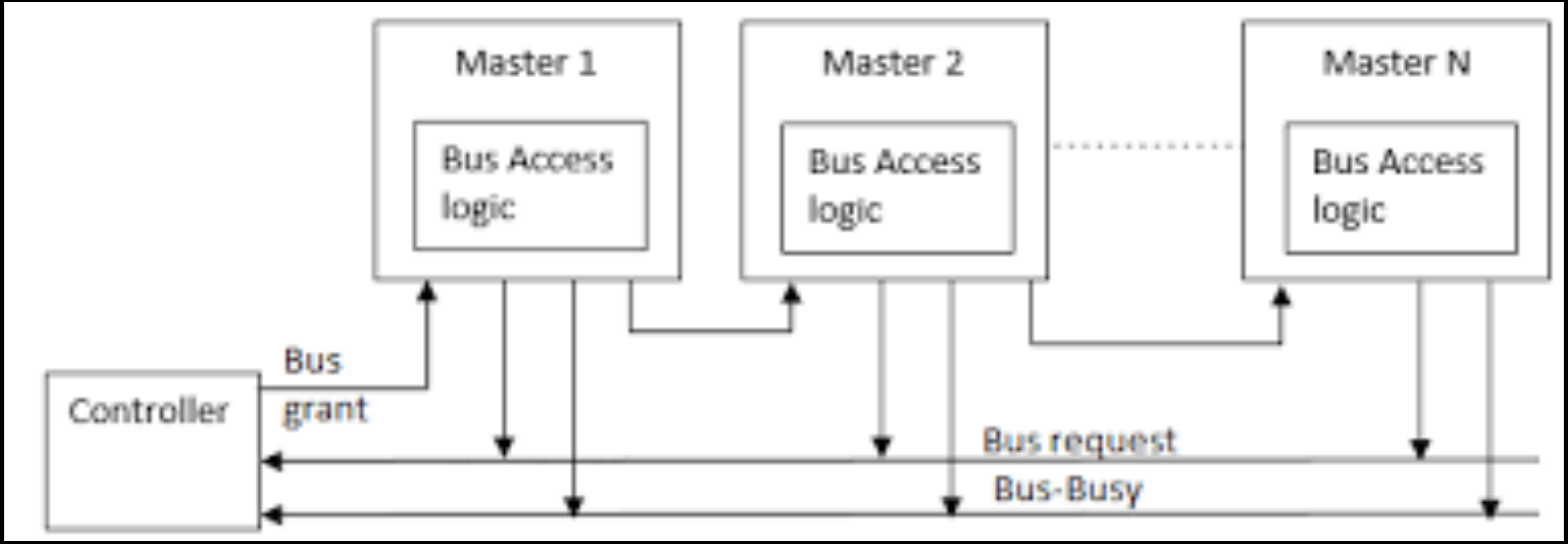
- **Data bus: -** In the final step depending on the operation either CPU will put data on the data line and device will store it or device will put data on the data line and CPU will store it.



Bus Arbitration

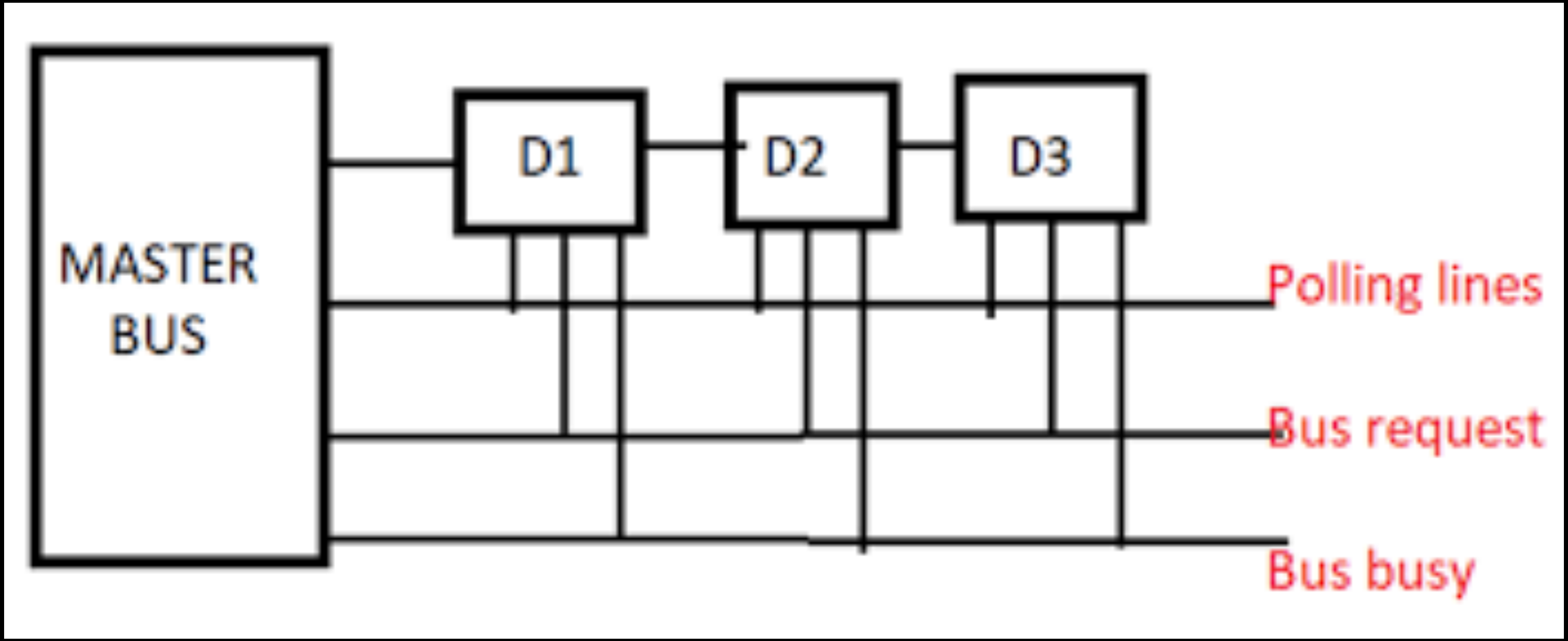
- **Bus Arbitration**: This is the method used to decide which device gets access to the common bus when multiple devices request it simultaneously, ensuring data integrity and system stability.
- **Conflict Resolution**: Without bus arbitration, simultaneous access could result in data corruption and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



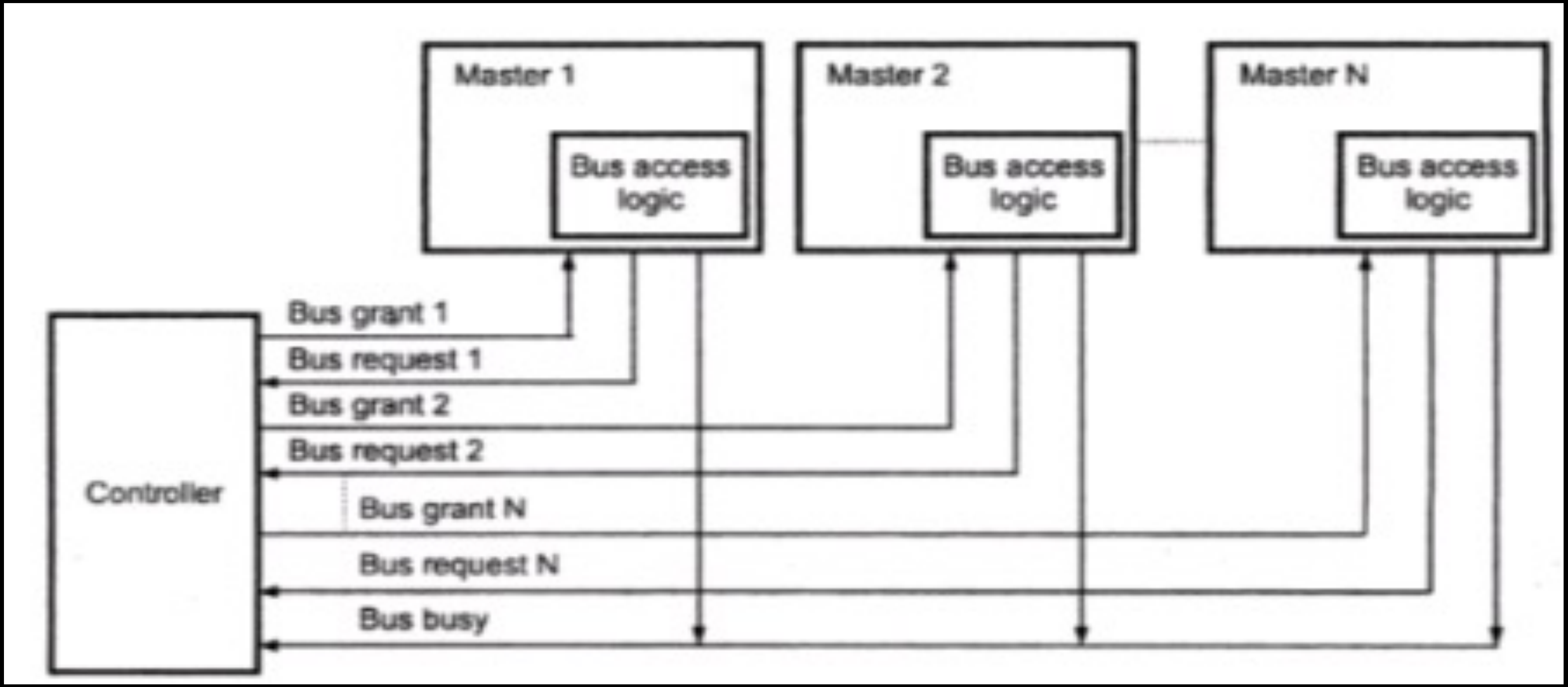
Polling

- **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.

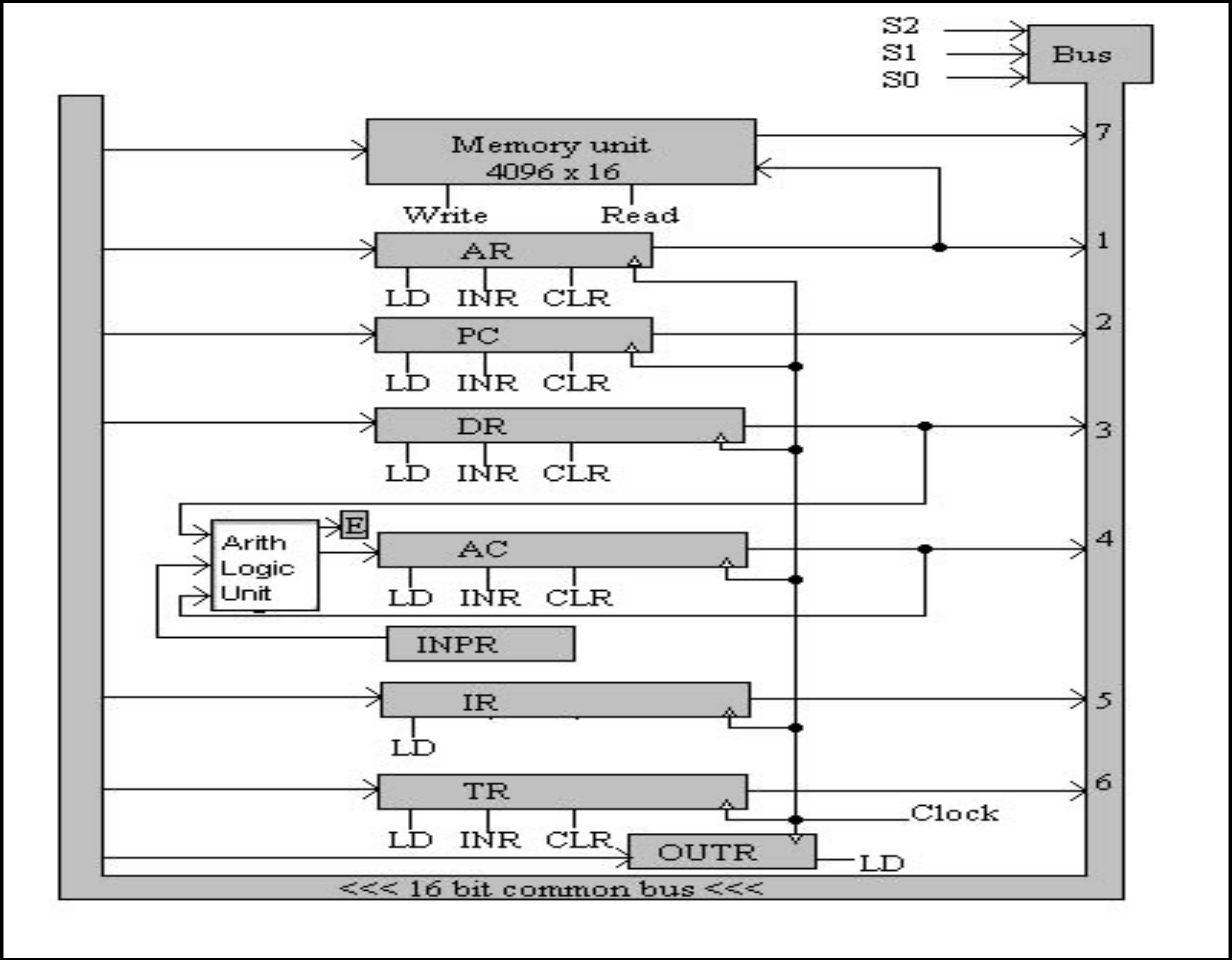


Fixed priority or Independent Request method

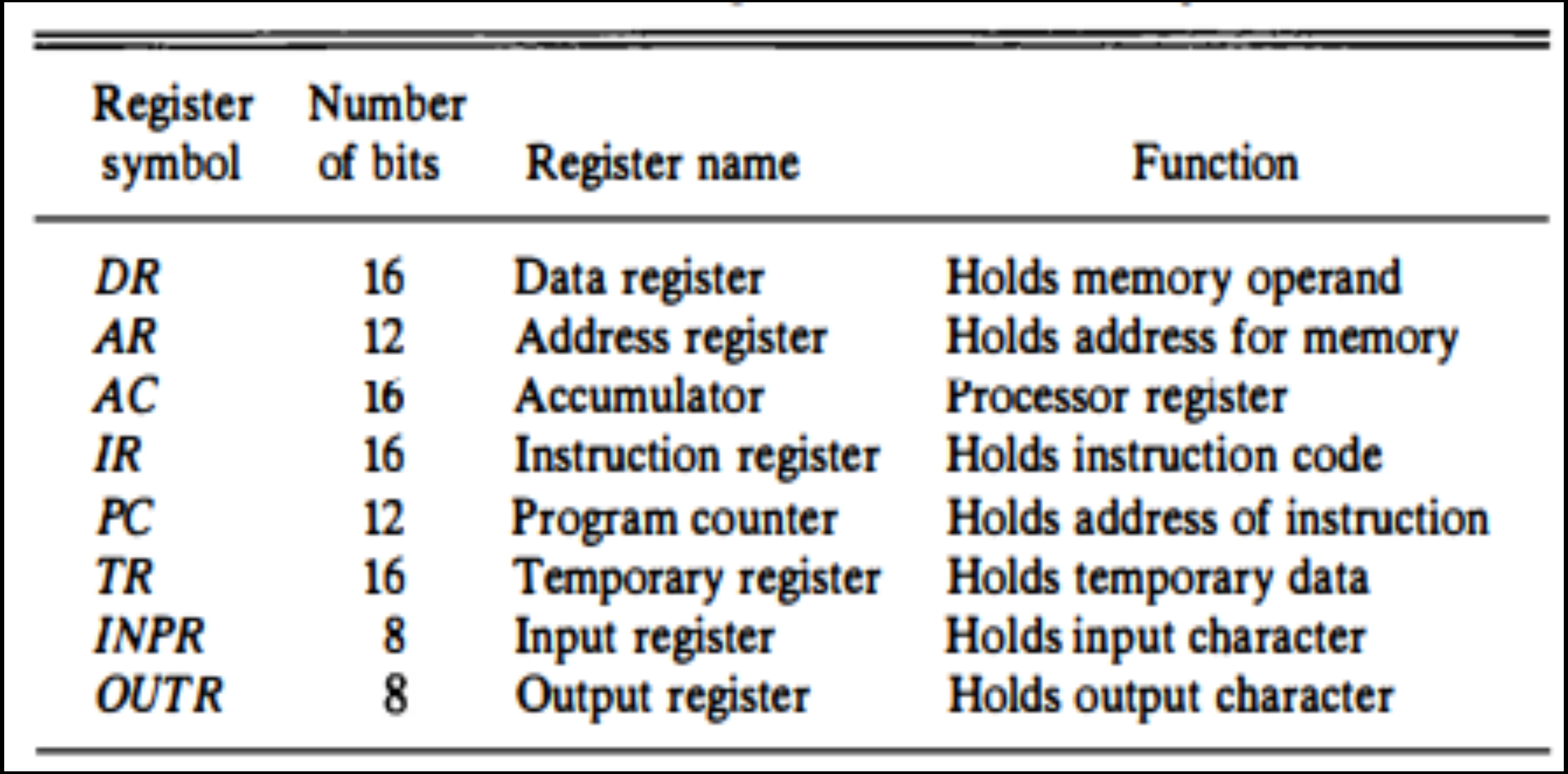
- In this, each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it.
- The built-in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.



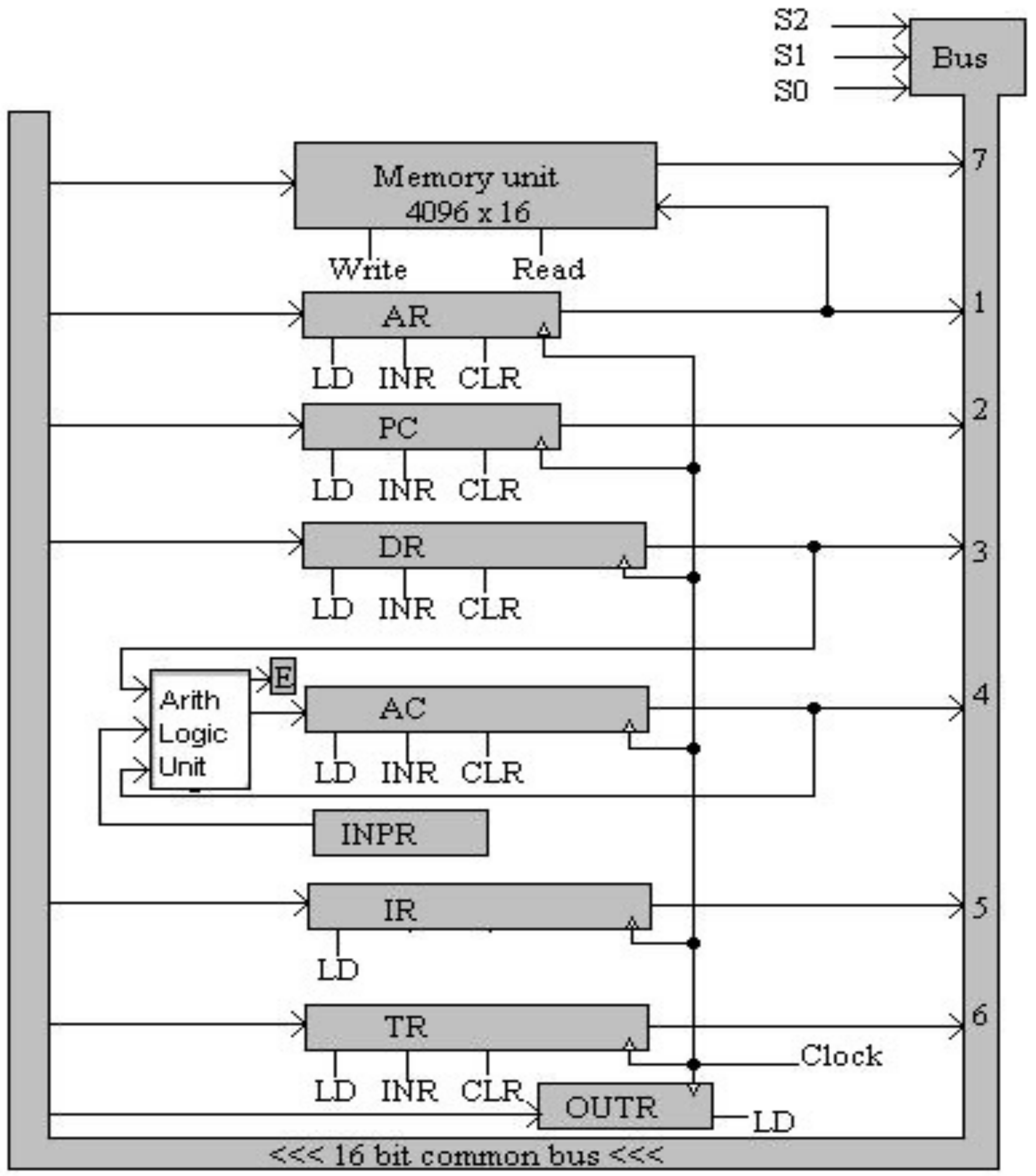
Processor Organization



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.

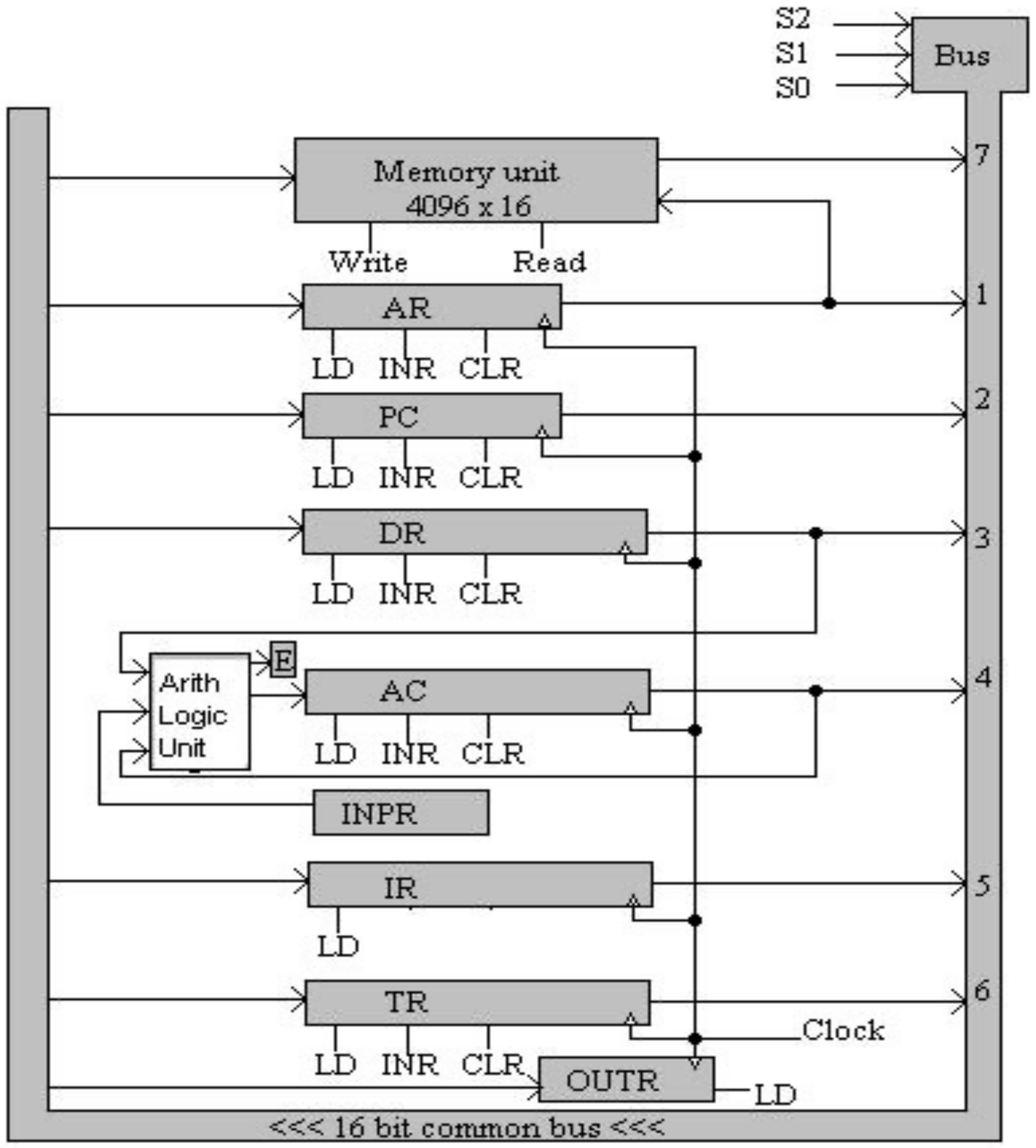


Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register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



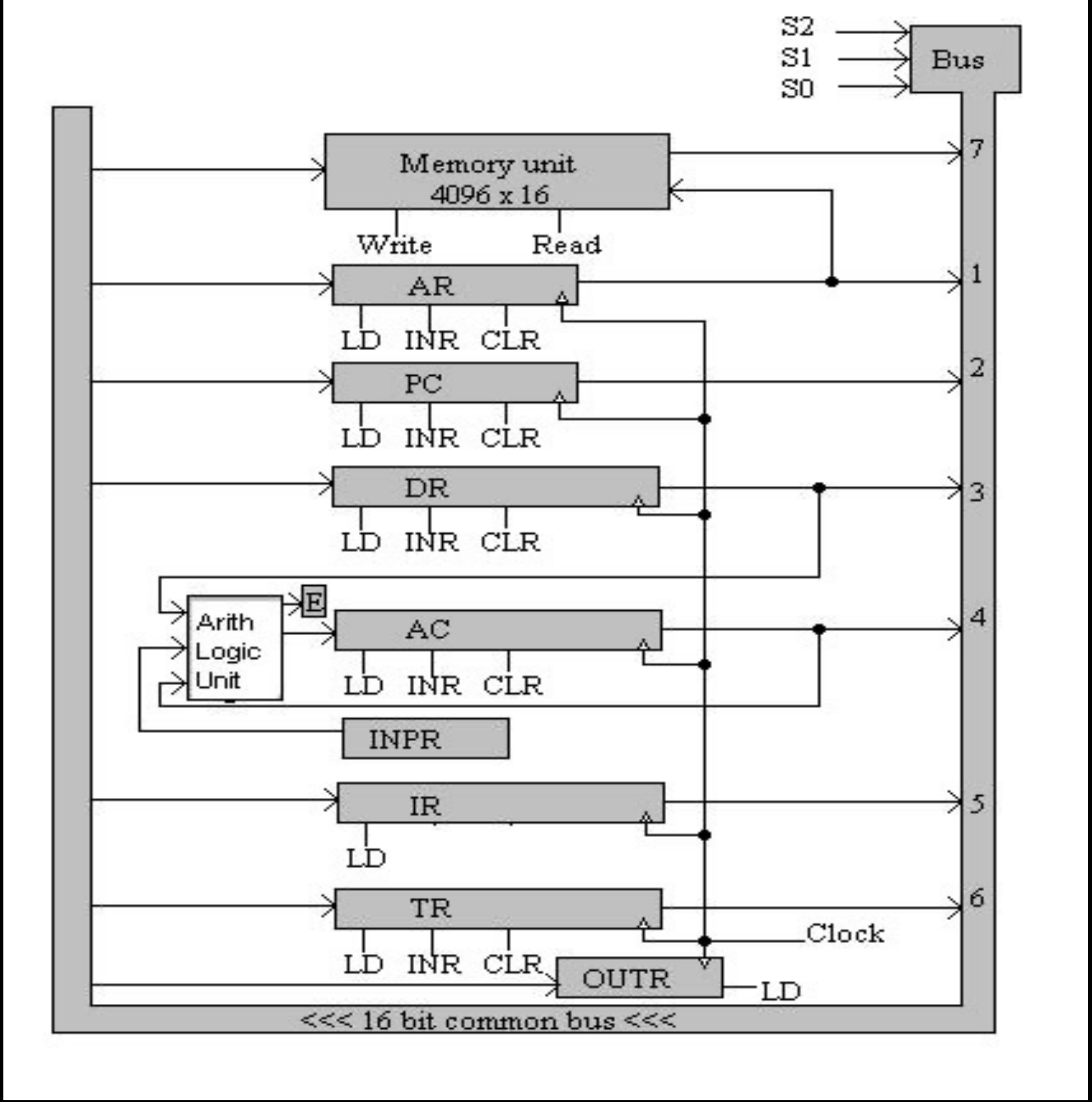
- The **data register (DR)** is a register which is used to store any data that is to be transferred to memory or which are fetched from memory.

- 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.



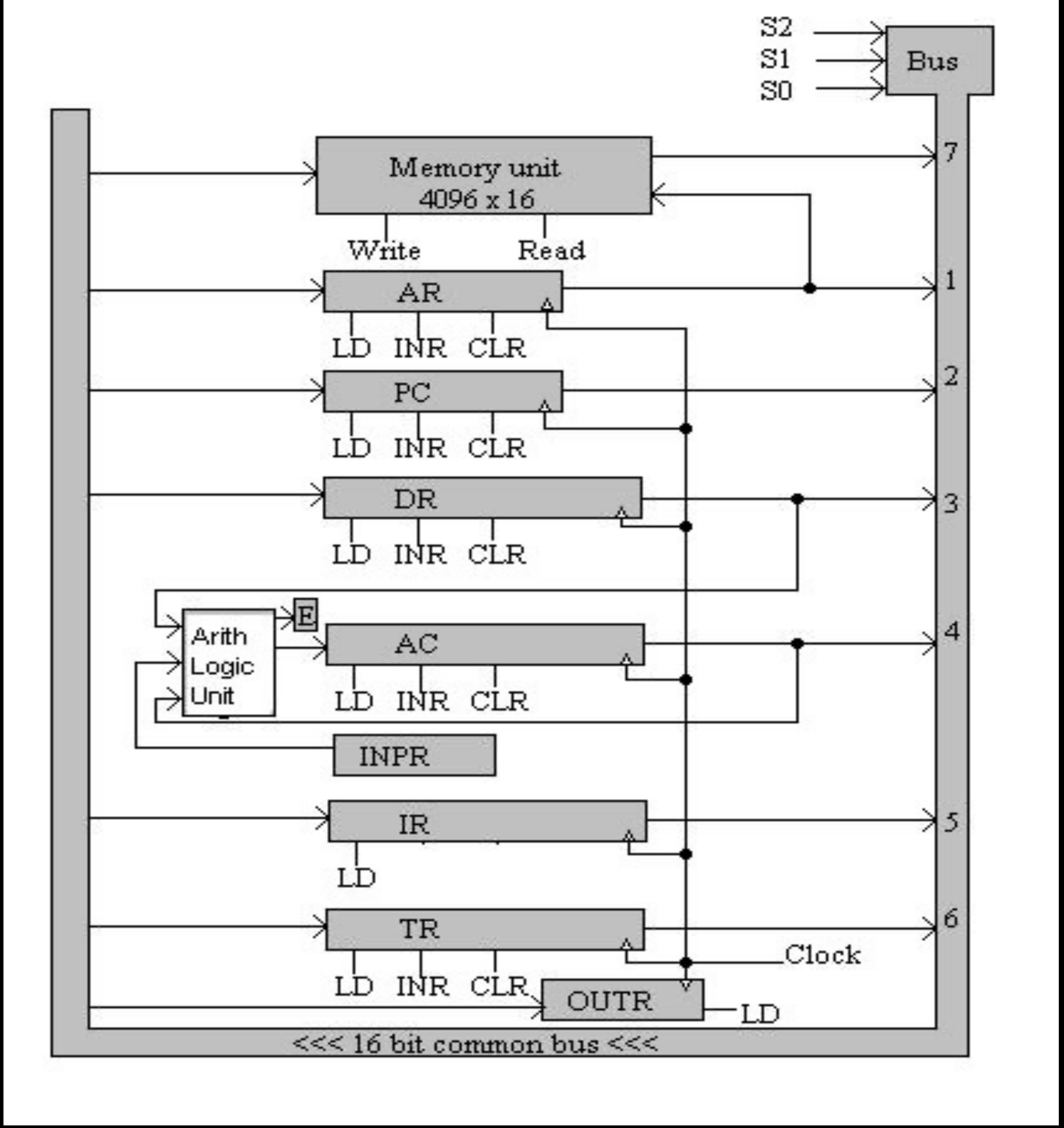
- **Instruction register**- is used to store instruction which we fetched from memory, so that we analyses the instruction using a decoder and can understand what instruction want to perform.

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



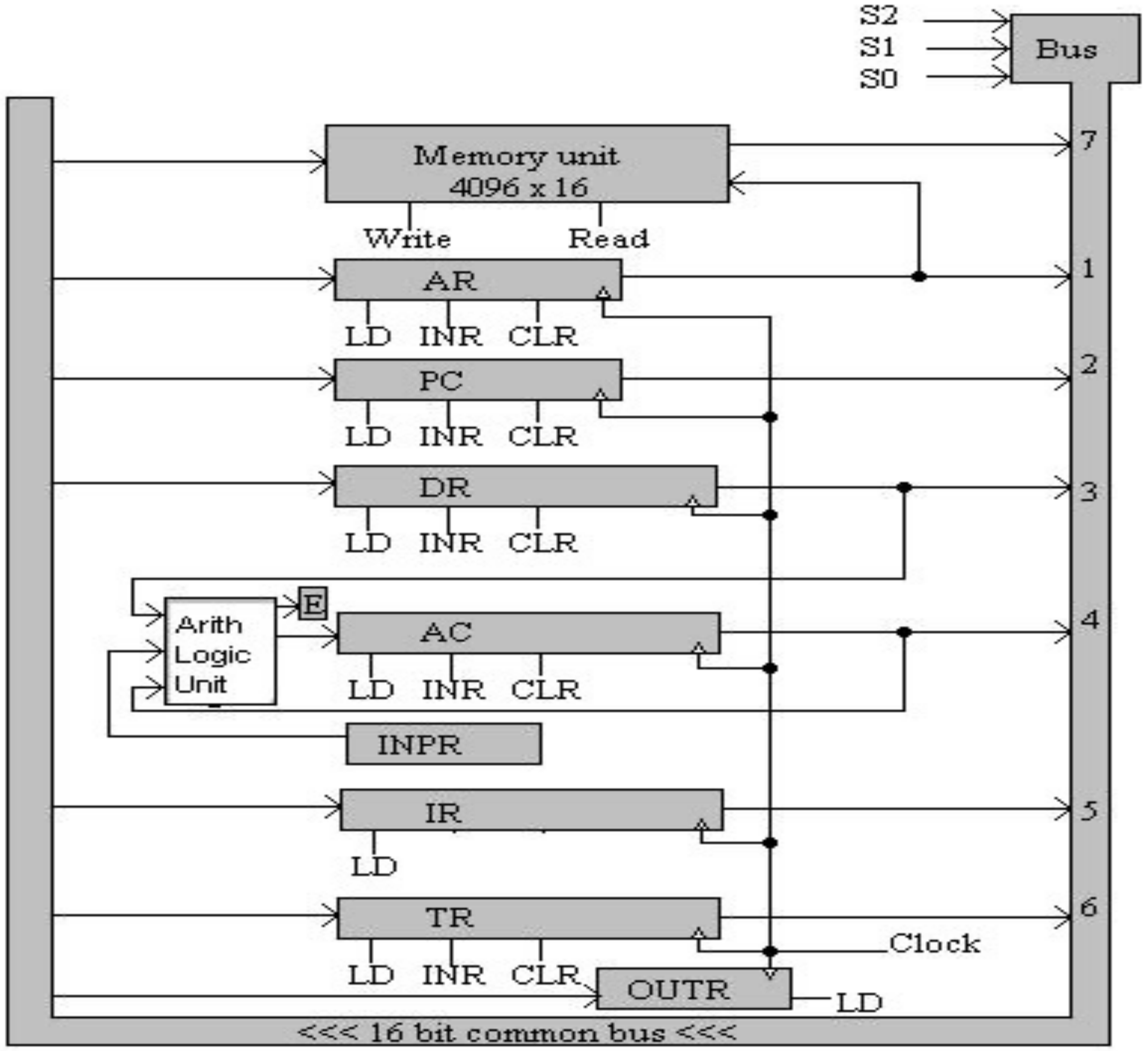
- The memory **address register (AR)** has 12 bits since this is the width of a memory address (always used least significant bits of bus). It stores the memory locations of instructions or data that need to be fetched from memory or stored in memory.

- 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.

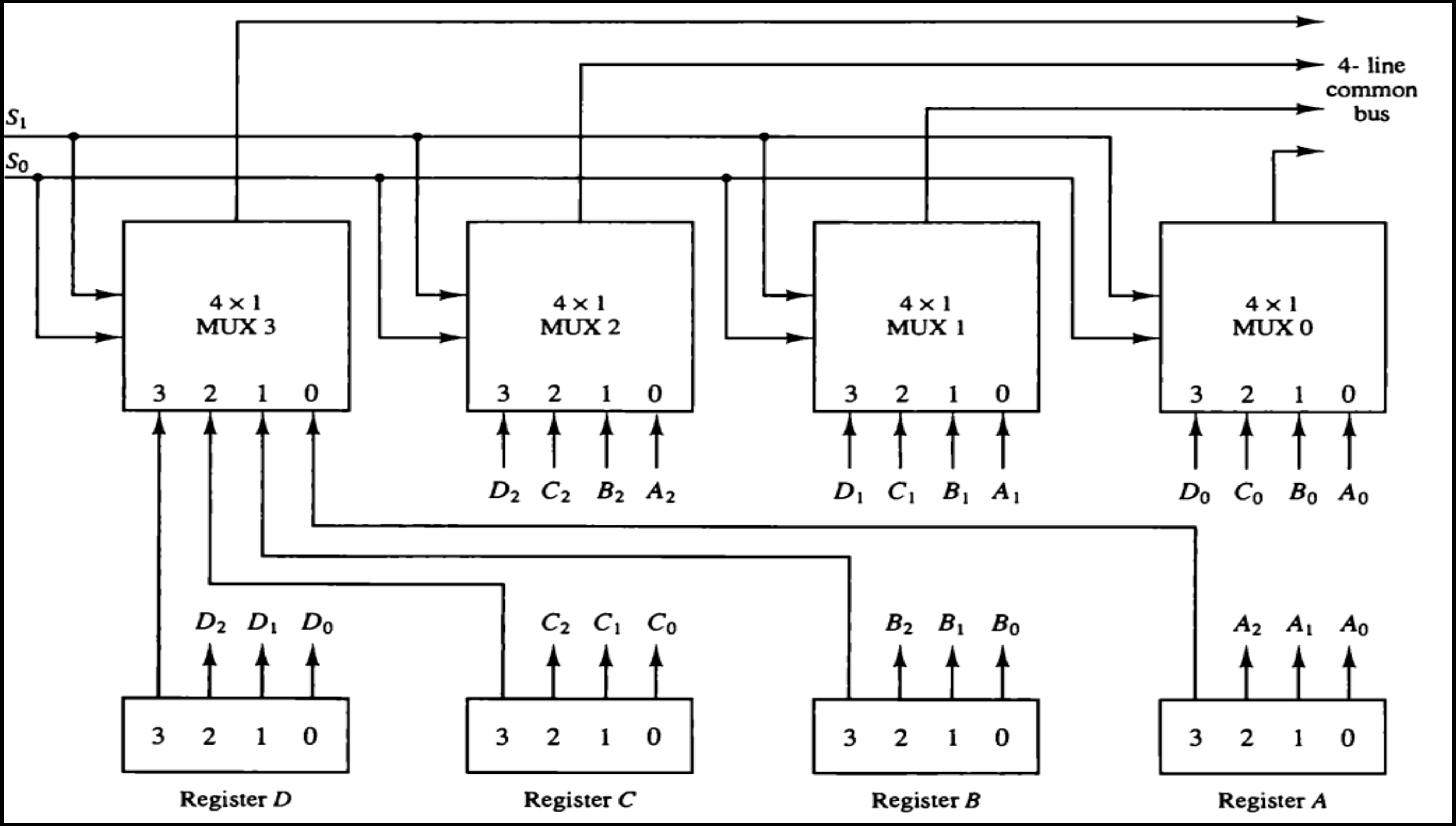


- Two registers are used for input and output. The **input register (INPR)** receives an 8-bit character from an input device, and pass it to ALU then accumulator and then to memory.

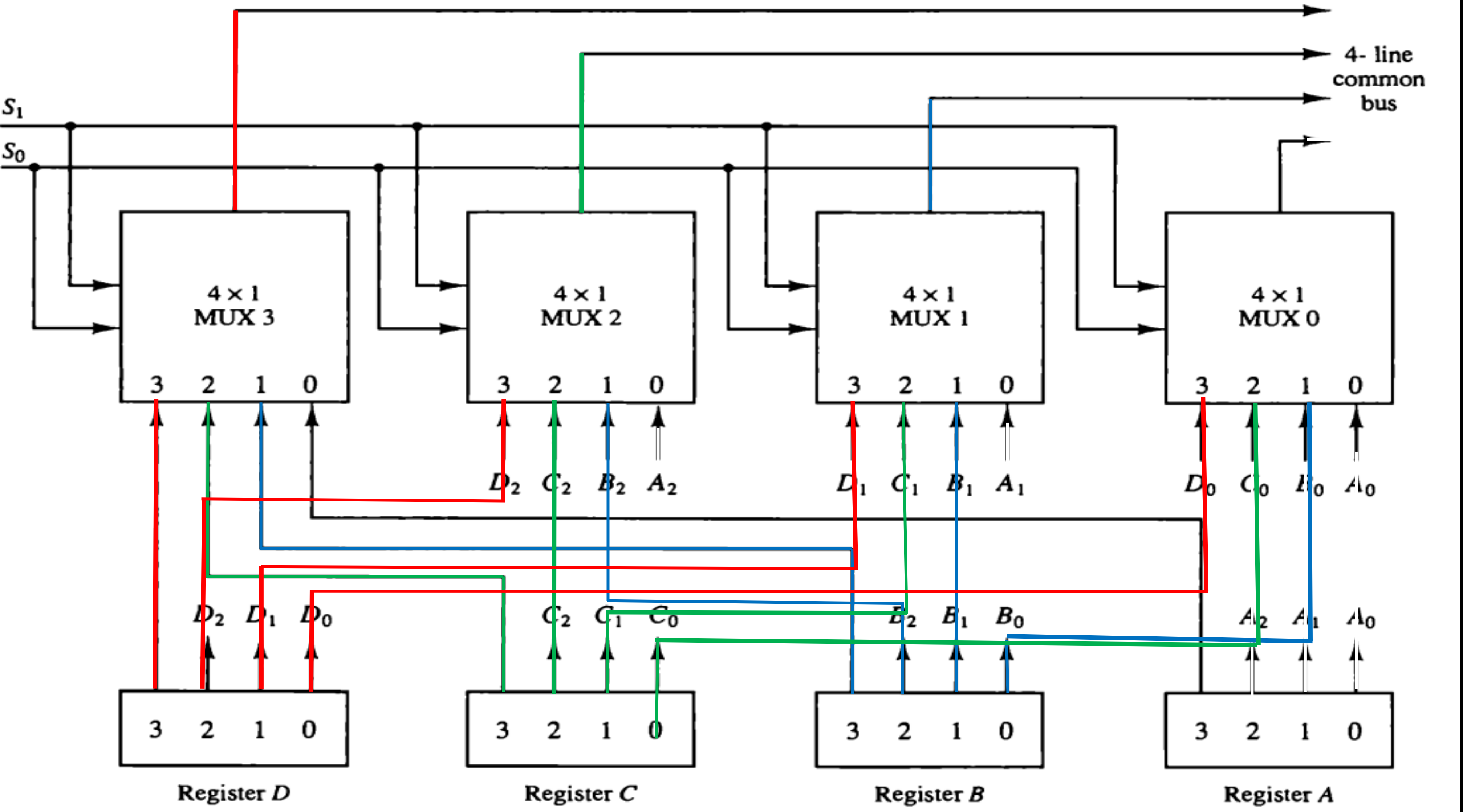
- The output **register (OUTR)** holds an 8-bit character for an output device, screen, printer etc.



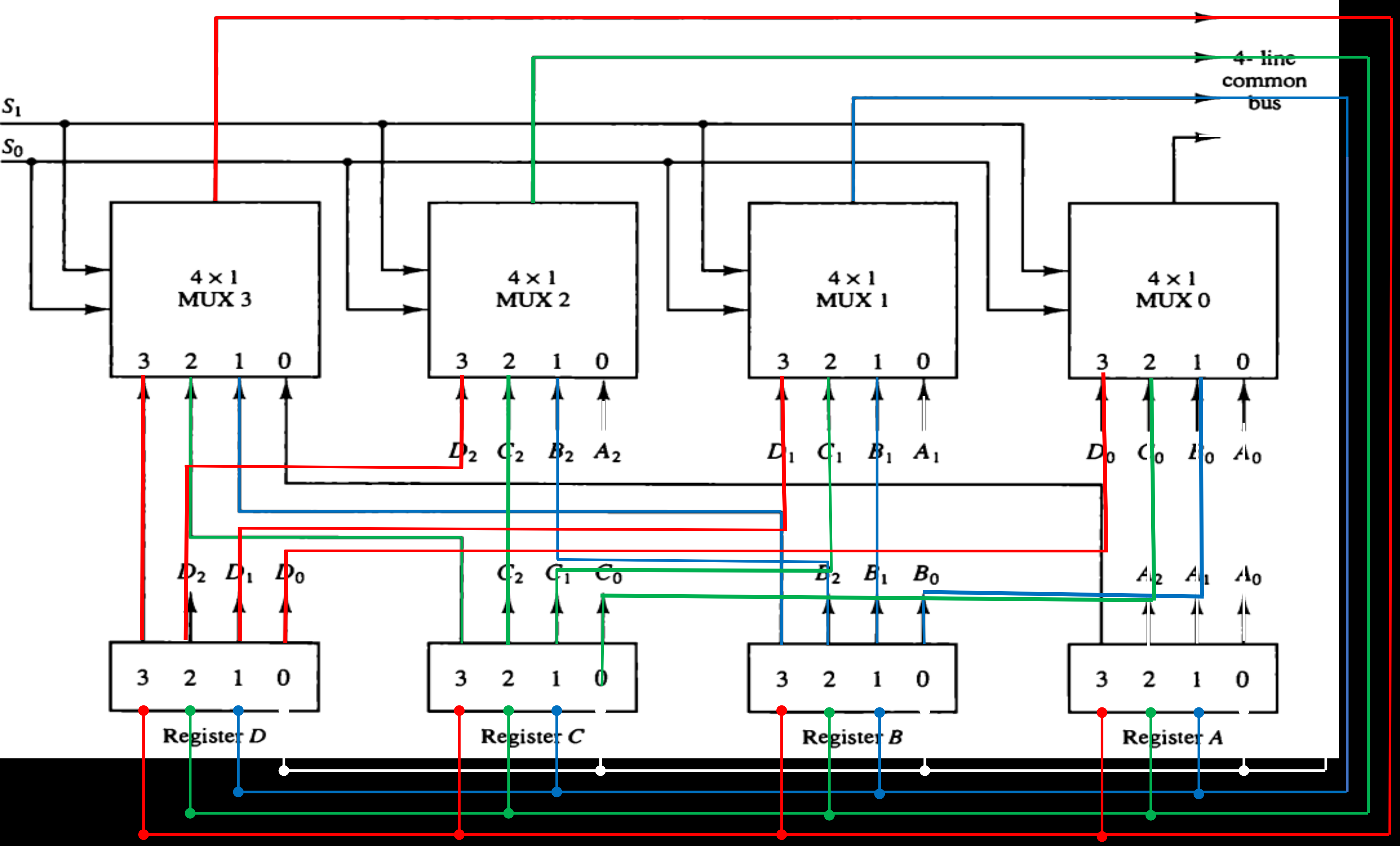
- The outputs of seven registers and memory are connected to the common bus.
- The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0.
- The number along each output shows the decimal equivalent of the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



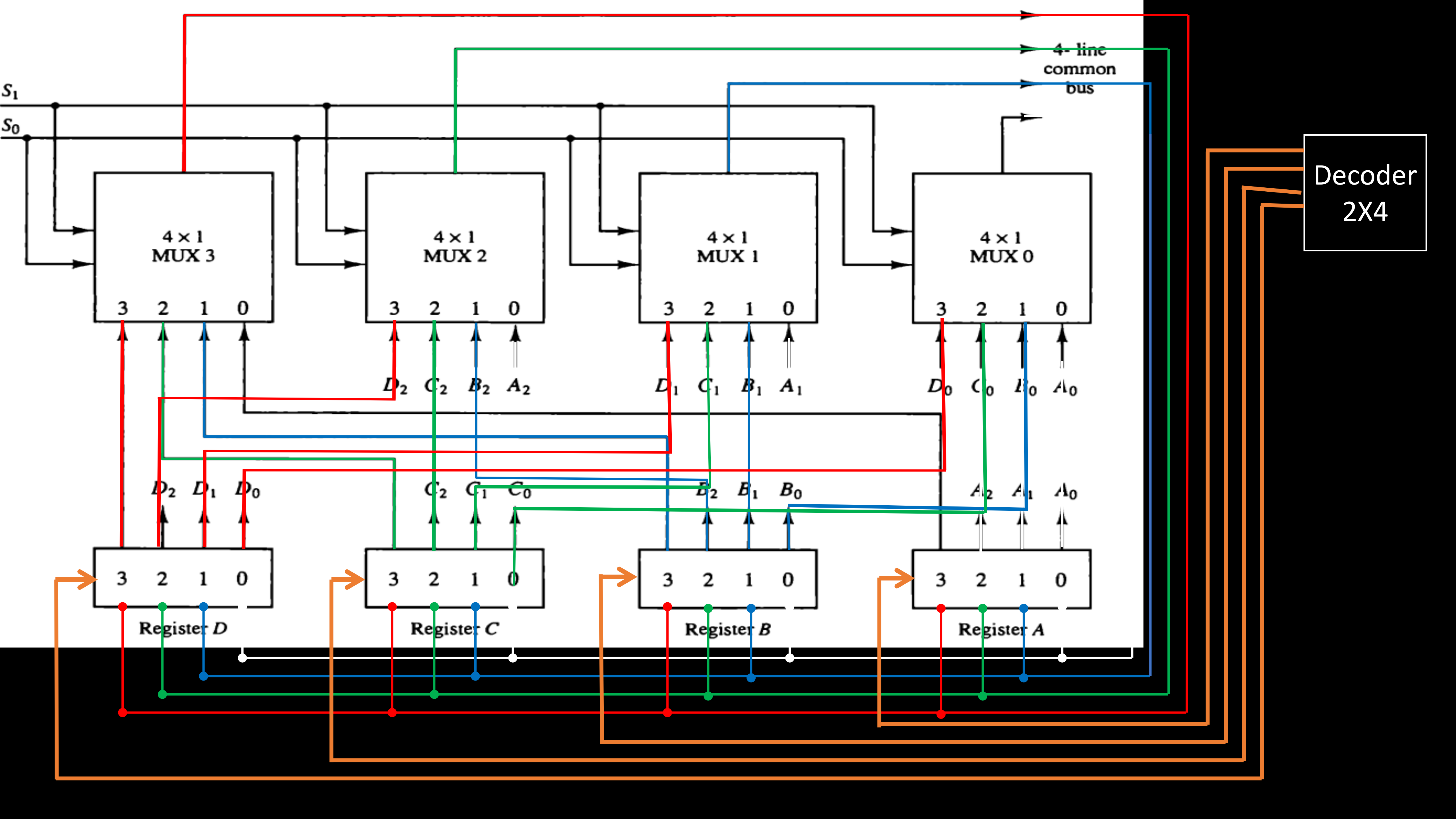
4- line common bus
S_1
S_0
4 × 1 MUX 3
4 × 1 MUX 2
4 × 1 MUX 1
4 × 1 MUX 0
3 2 1 0
D_2 C_2 B_2 A_2
D_1 C_1 B_1 A_1
D_0 C_0 B_0 A_0
D_2 D_1 D_0
C_2 C_1 C_0
B_2 B_1 B_0
A_2 A_1 A_0
Register D
Register C
Register B
Register A



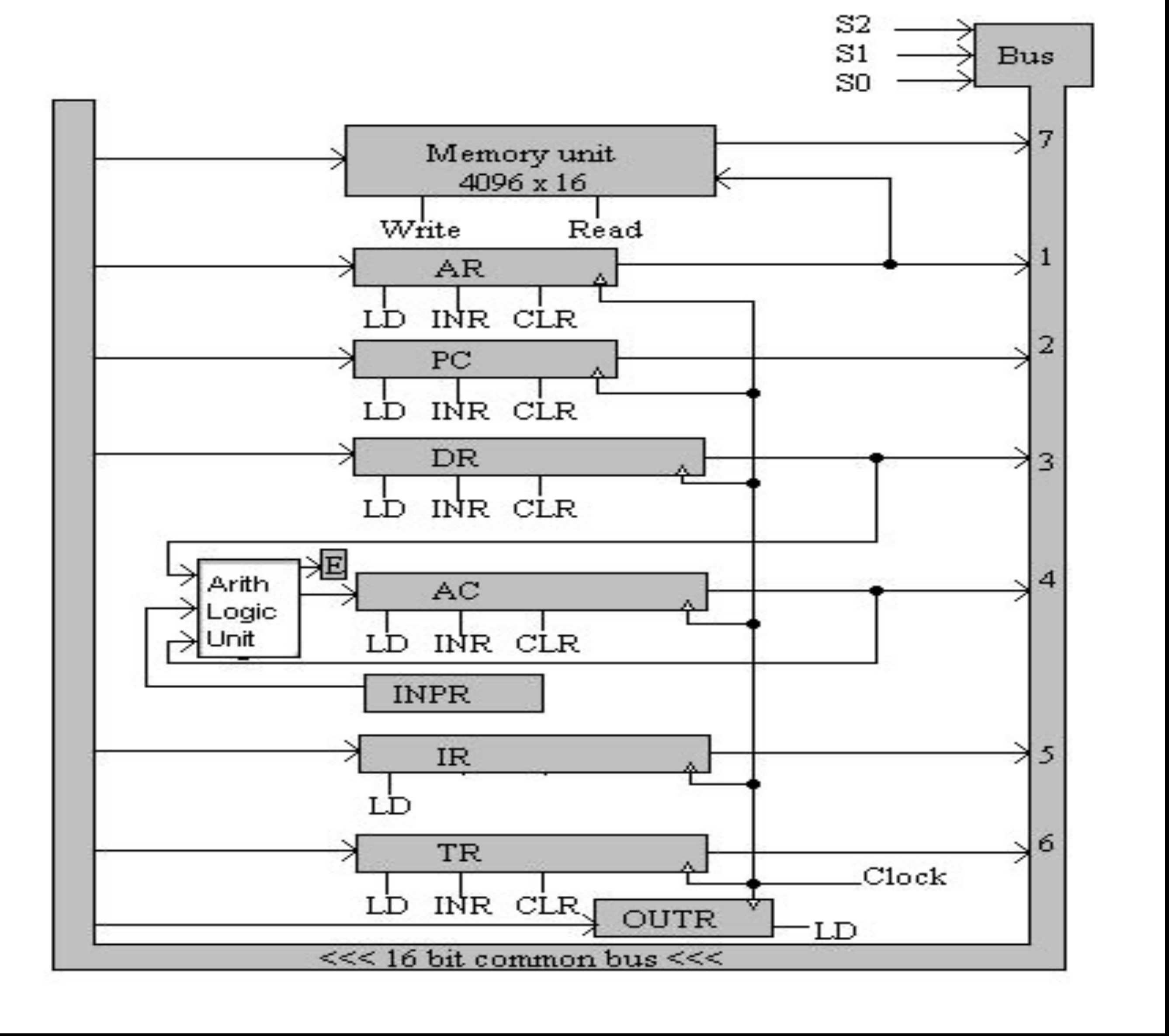
S_1
S_0
4- line common bus
4 × 1 MUX 3
4 × 1 MUX 2
4 × 1 MUX 1
4 × 1 MUX 0
3 2 1 0
D_2 C_2 B_2 A_2
D_1 C_1 B_1 A_1
D_0 C_0 B_0 A_0
D_2 D_1 D_0
C_2 C_1 C_0
B_2 B_1 B_0
A_2 A_1 A_0
Register D
Register C
Register B
Register A



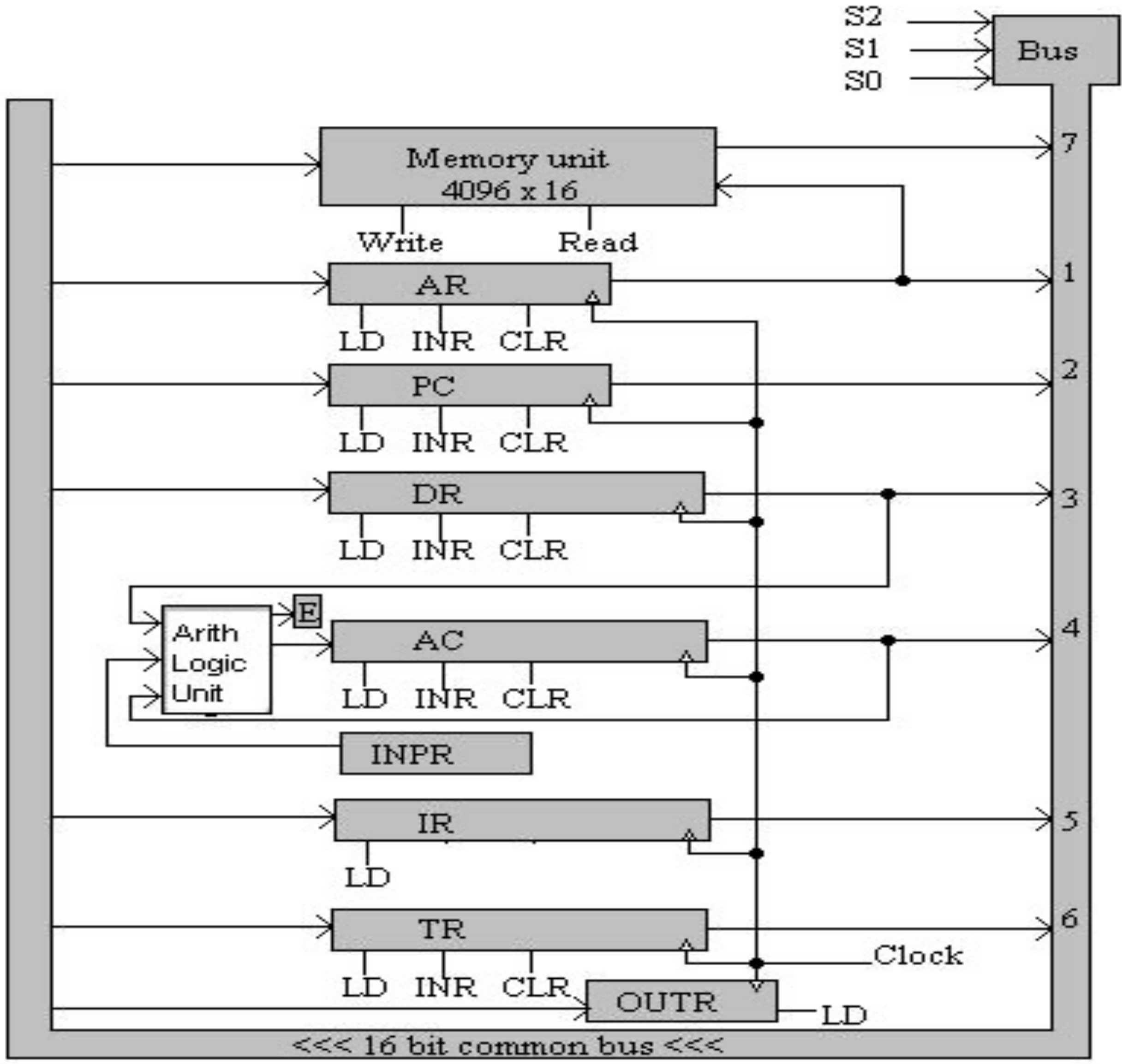
S_1
S_0
4 × 1 MUX 3
4 × 1 MUX 2
4 × 1 MUX 1
4 × 1 MUX 0
3 2 1 0
D_2 C_2 B_2 A_2
D_1 C_1 B_1 A_1
D_0 C_0 B_0 A_0
4- line common bus
D_2 D_1 D_0
C_2 C_1 C_0
B_2 B_1 B_0
A_2 A_1 A_0
Register D
Register C
Register B
Register A



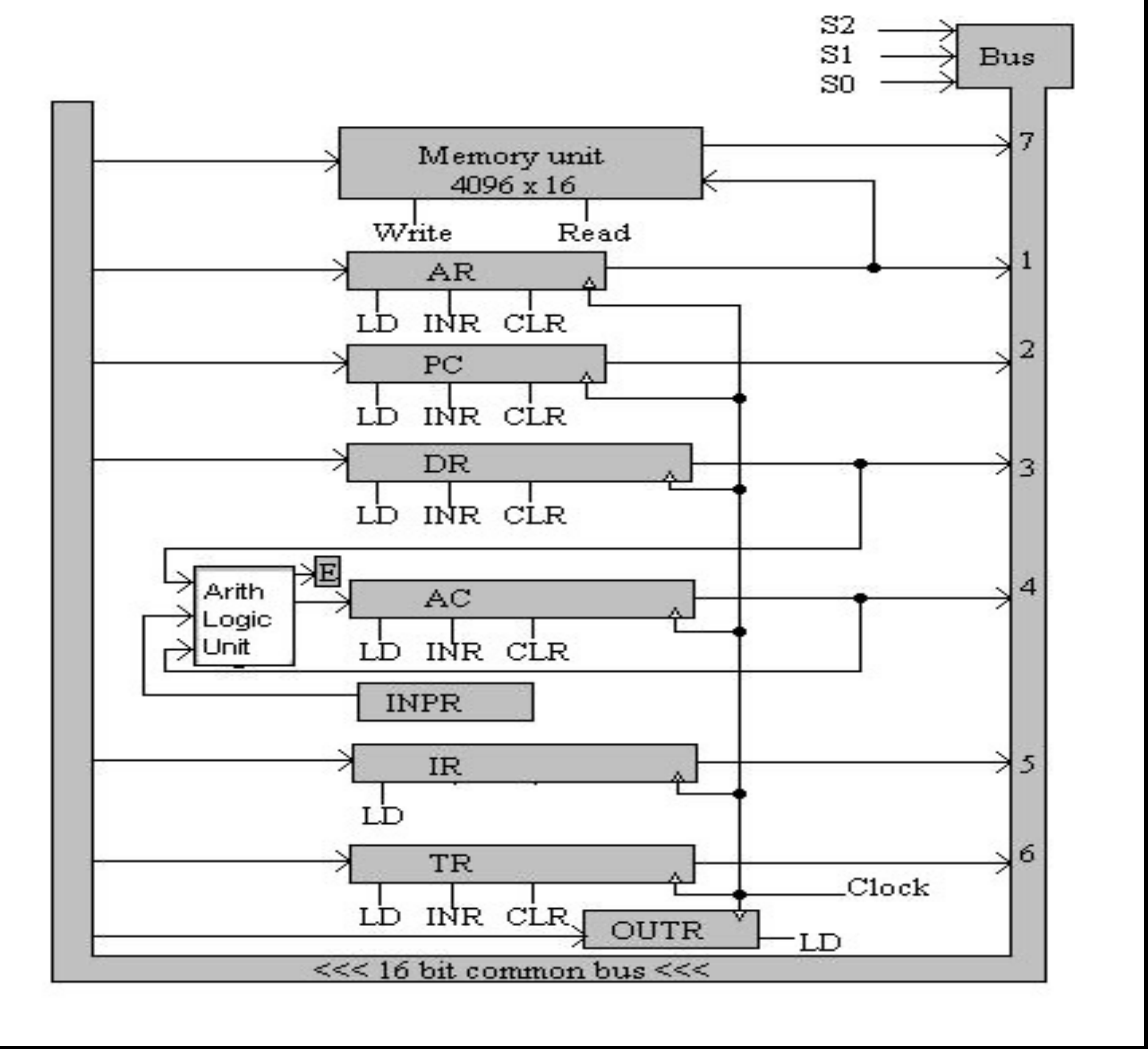
4- line common bus
S_1
S_0
4 × 1 MUX 3
4 × 1 MUX 2
4 × 1 MUX 1
4 × 1 MUX 0
3 2 1 0
D_2 C_2 B_2 A_2
D_1 C_1 B_1 A_1
D_0 C_0 B_0 A_0
D_2 D_1 D_0
C_2 C_1 C_0
B_2 B_1 B_0
A_2 A_1 A_0
Register D
Register C
Register B
Register A
Decoder 2X4



- 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.

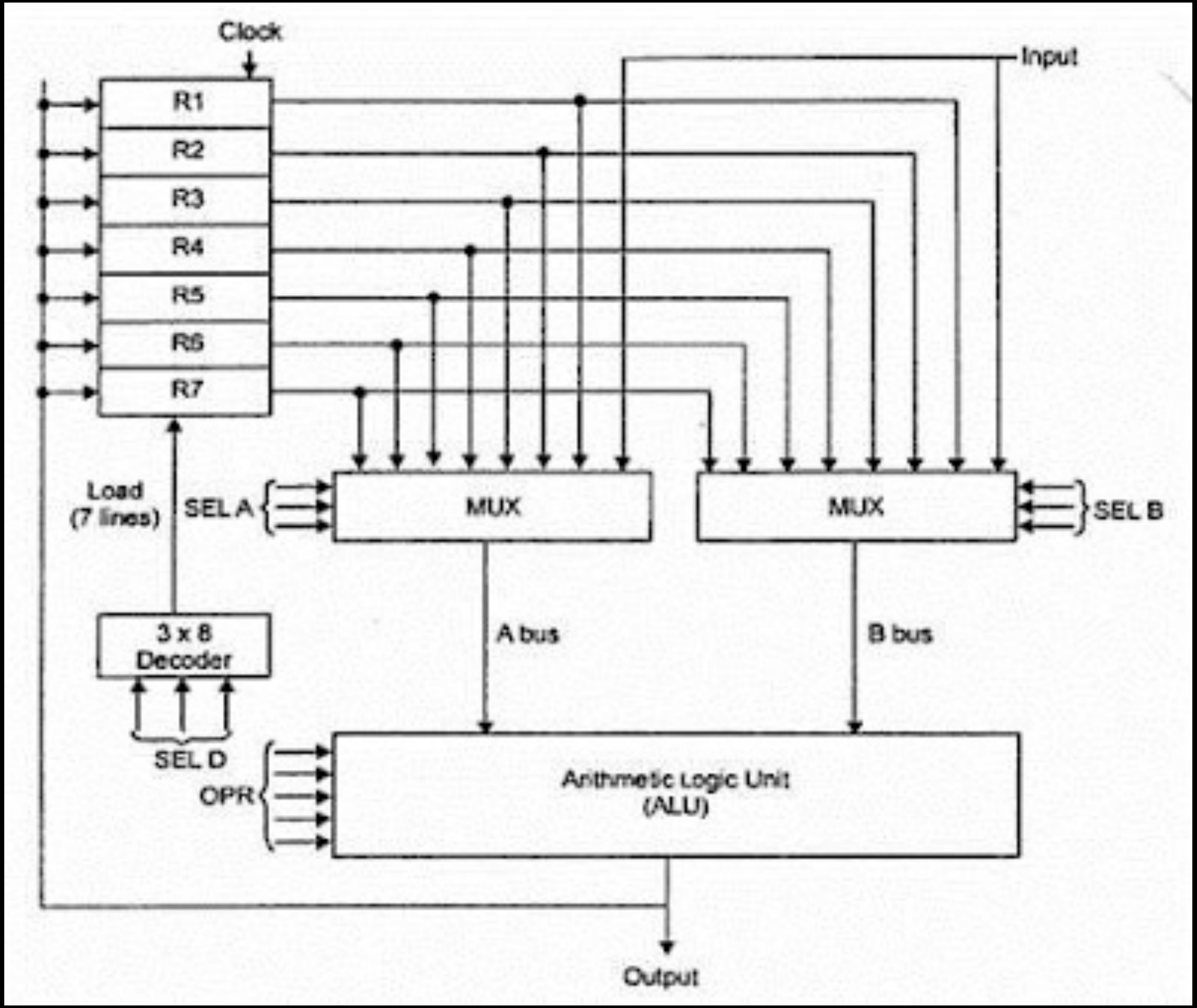


- The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus.
- INPR is connected to provide information to the bus but OUTR can only receive information from the bus. There is no transfer from OUTR to any of the other registers.



- 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



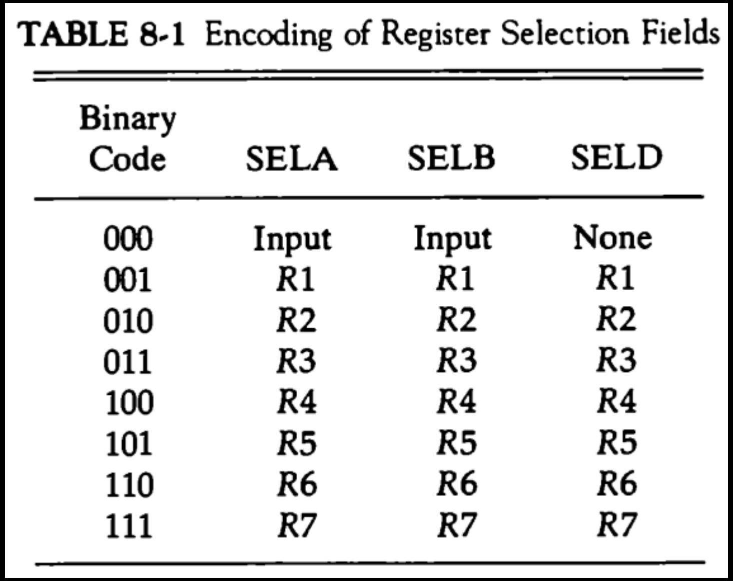


TABLE 8-1 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	*R*1	*R*1	*R*1
010	*R*2	*R*2	*R*2
011	*R*3	*R*3	*R*3
100	*R*4	*R*4	*R*4
101	*R*5	*R*5	*R*5
110	*R*6	*R*6	*R*6
111	*R*7	*R*7	*R*7

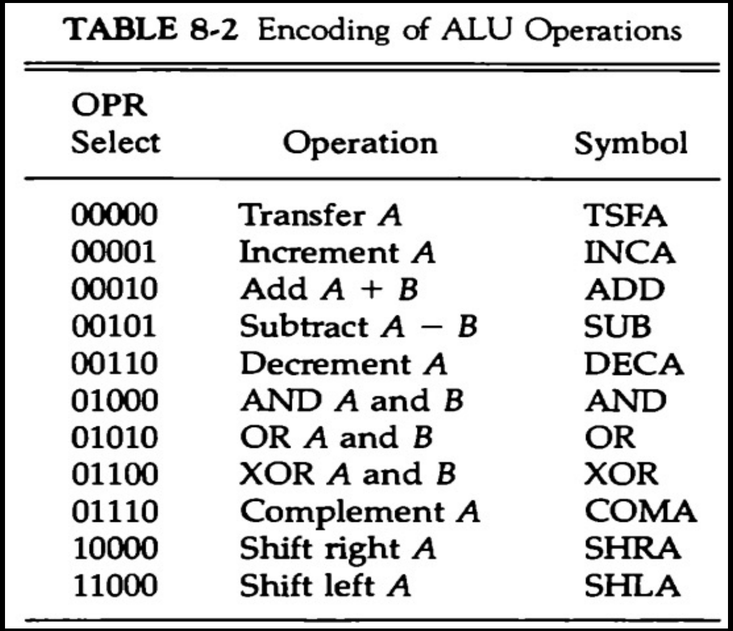
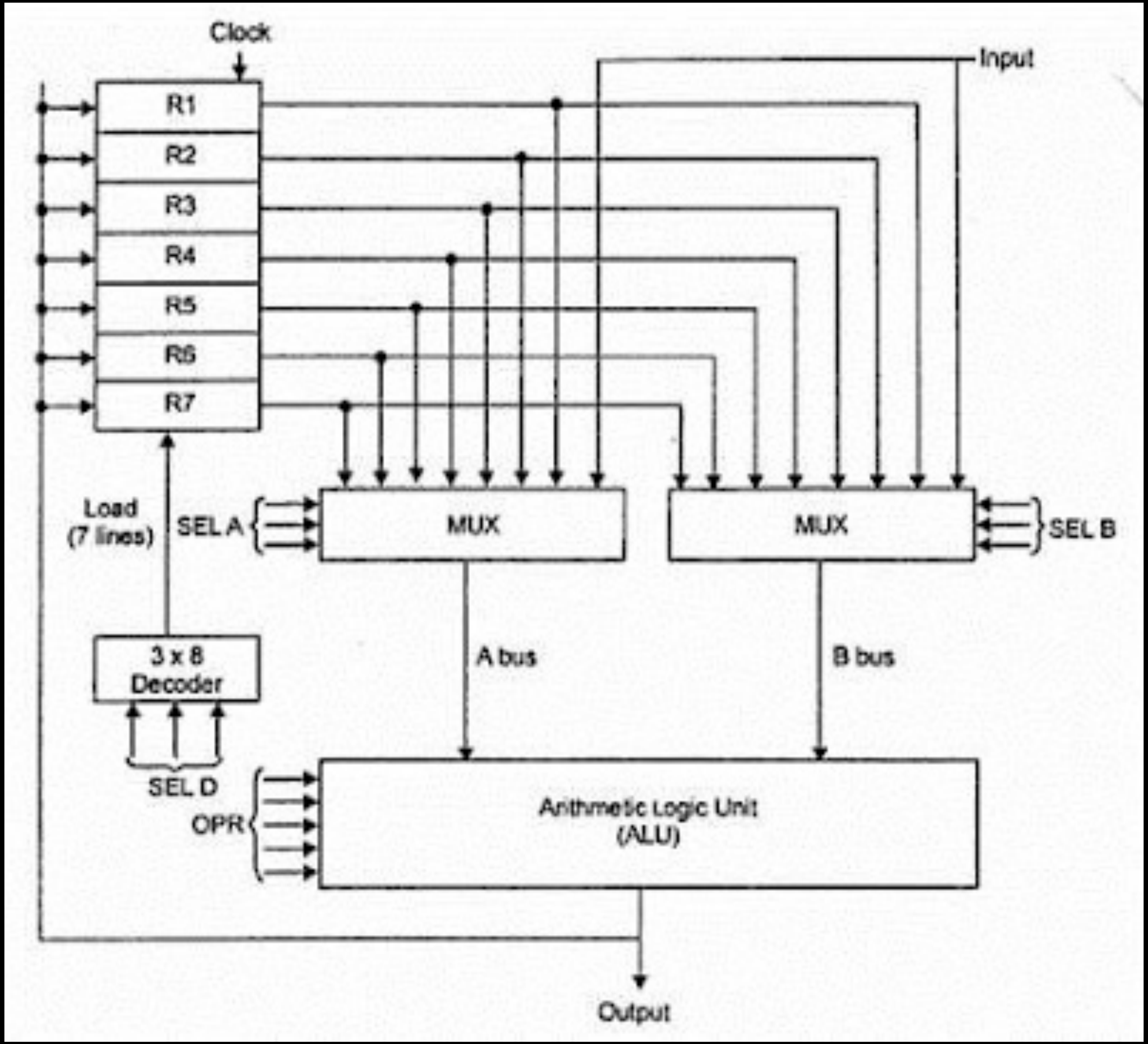


TABLE 8-2 Encoding of ALU Operations

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



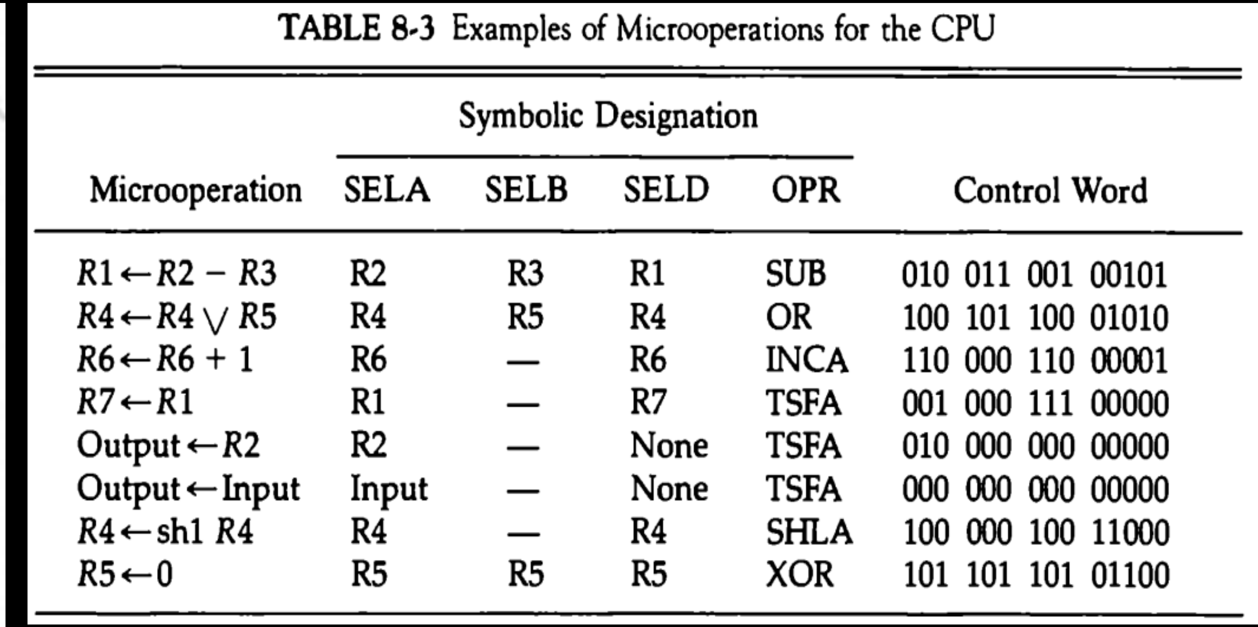
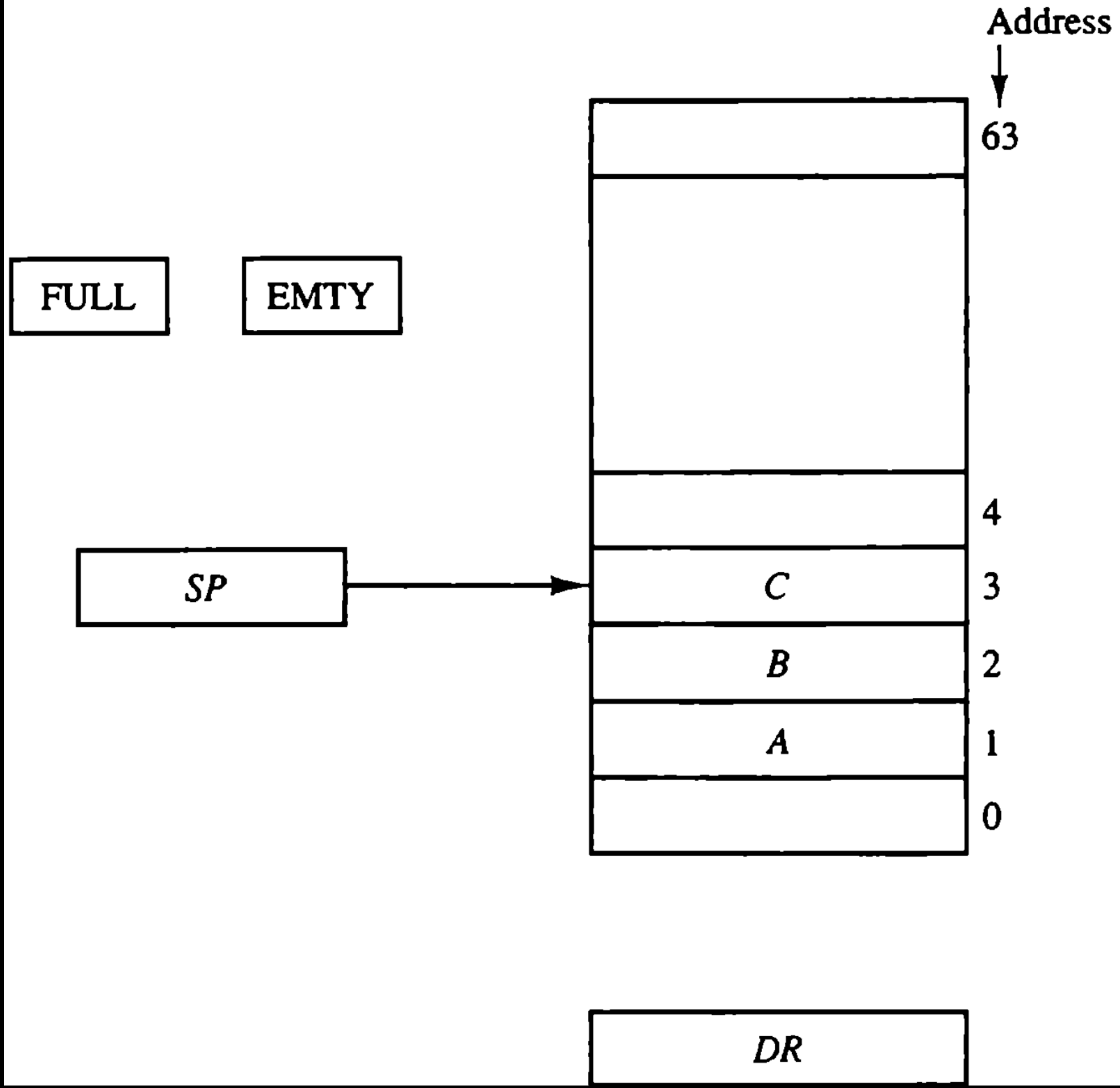


TABLE 8-3 Examples of Microoperations for the CPU

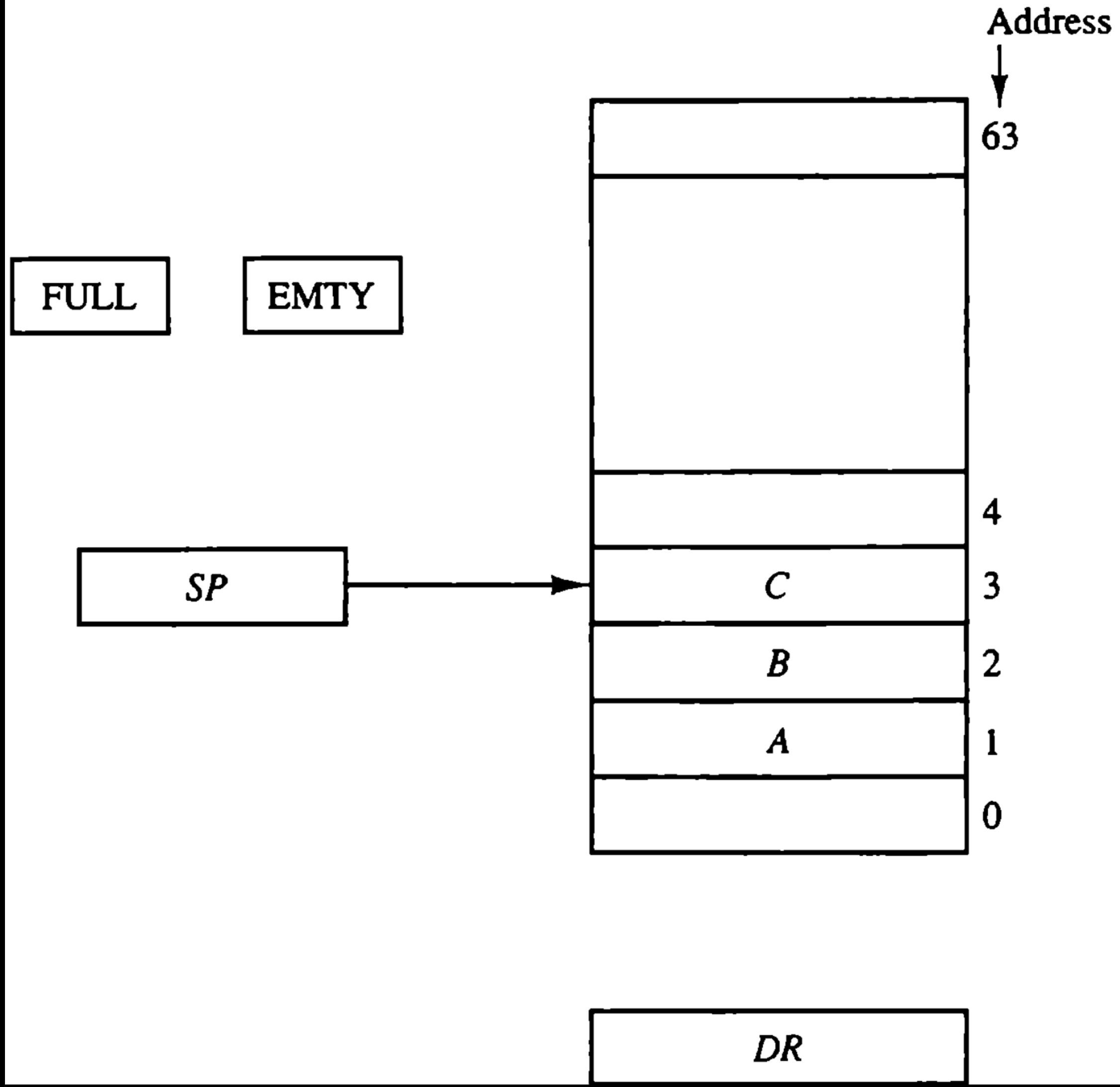
Microoperation	SELA	SELB	SELD	OPR	Control Word
	Symbolic Designation				
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ sh1 $R4$	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.

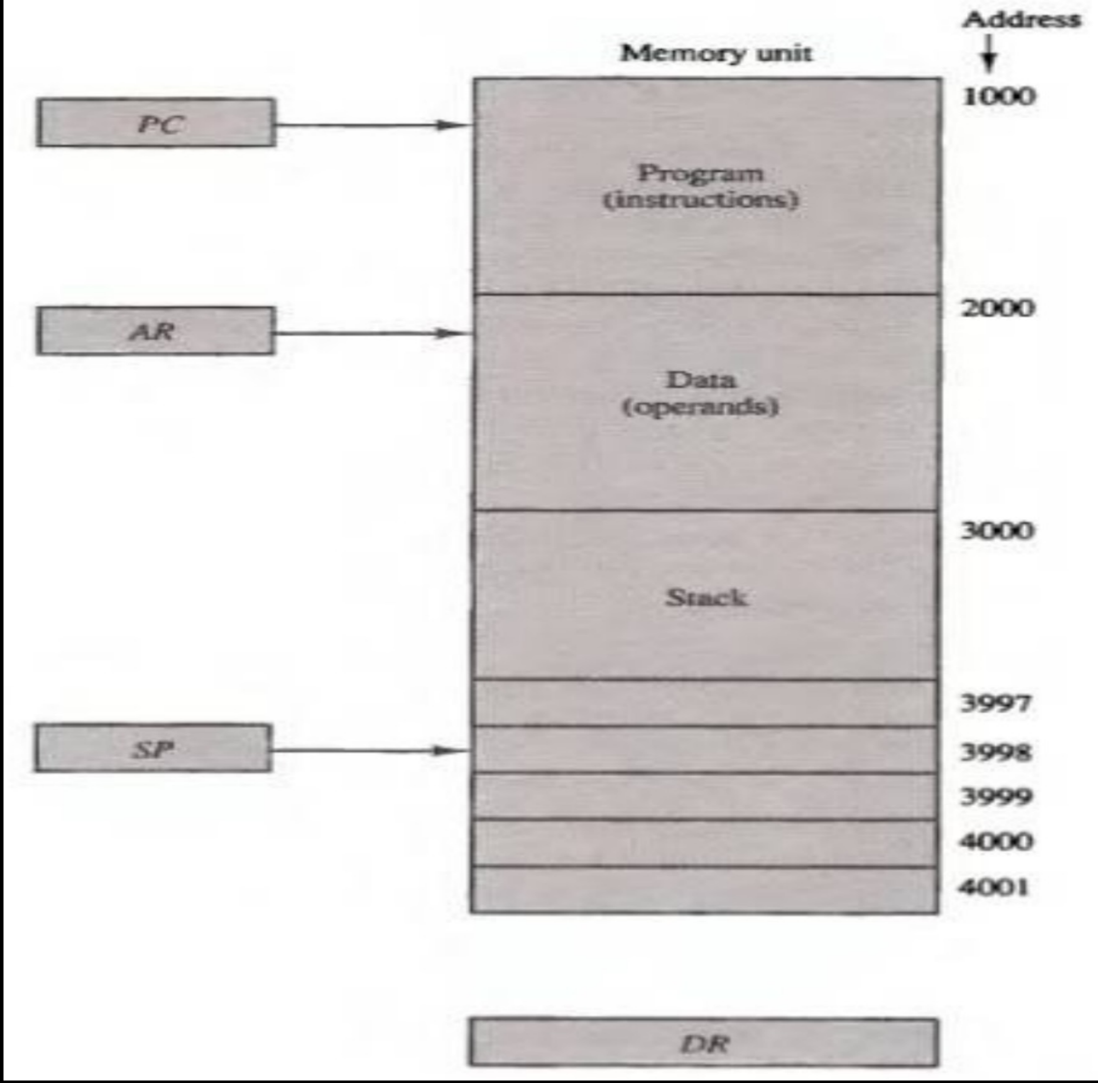


- Stack Pointer register (SP): It contains a value in binary each of 6 bits, which is the address of the top of the stack. Here, the stack pointer SP contains 6 bits and SP cannot contain a value greater than 111111 i.e. 63
- **Full Register**: it can store 1 bit of information, set to 1 when stack is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



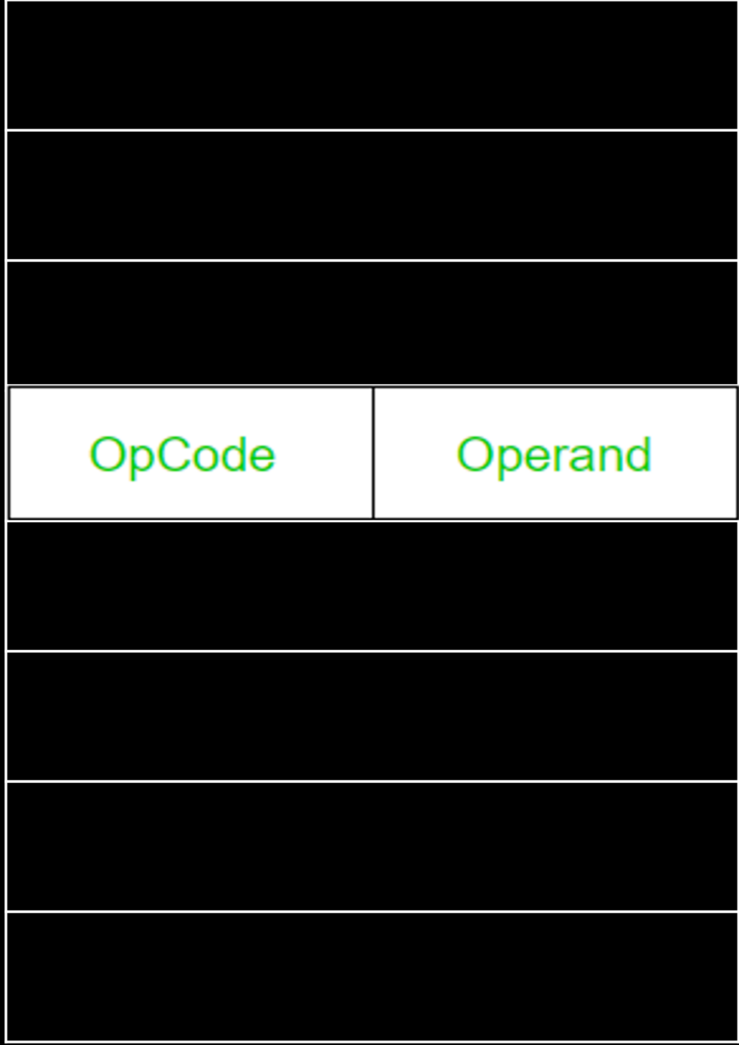
Addressing Mode

- 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



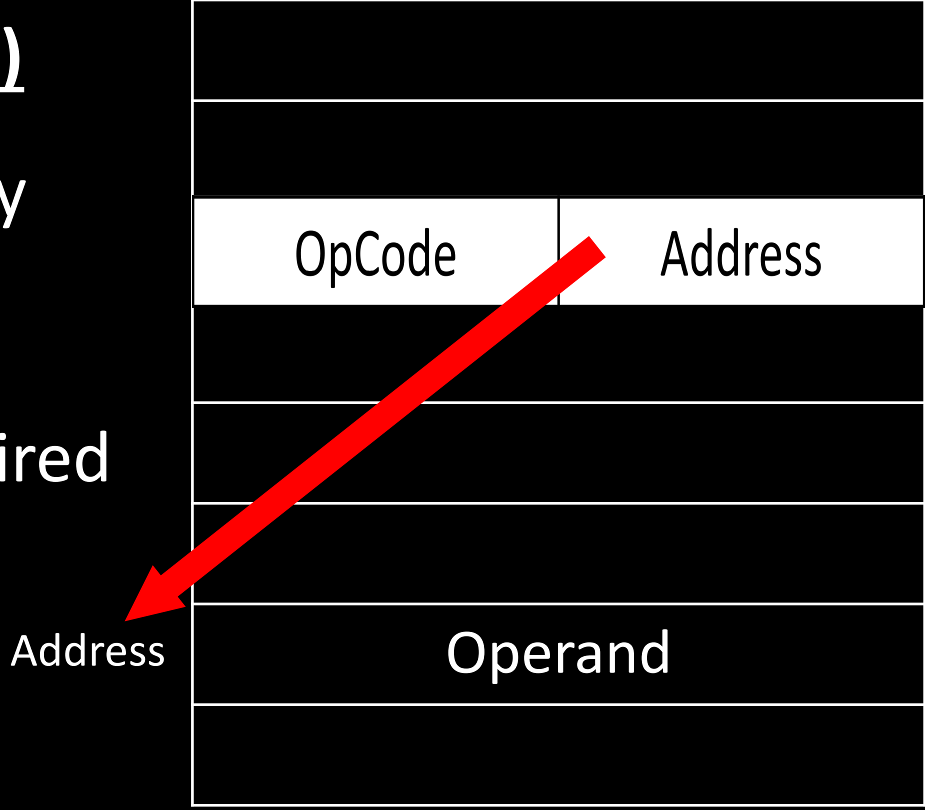
- Advantage
  - Can be used for constants, where values are already known.
  - Extremely fast, no memory reference is required.

- 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.

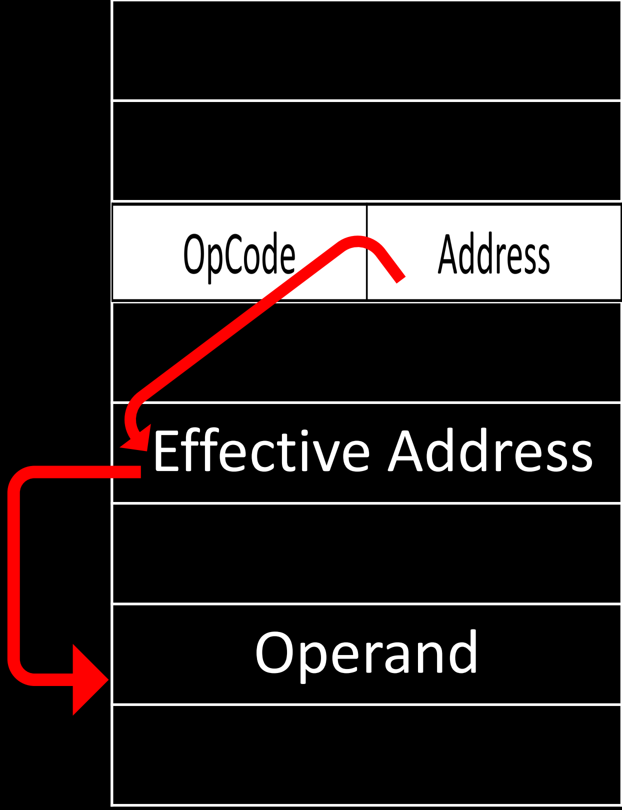
- Disadvantage
  - Relatively slow compare to immediate mode
  - No of variable used are limited
  - In large calculation it will fail

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.

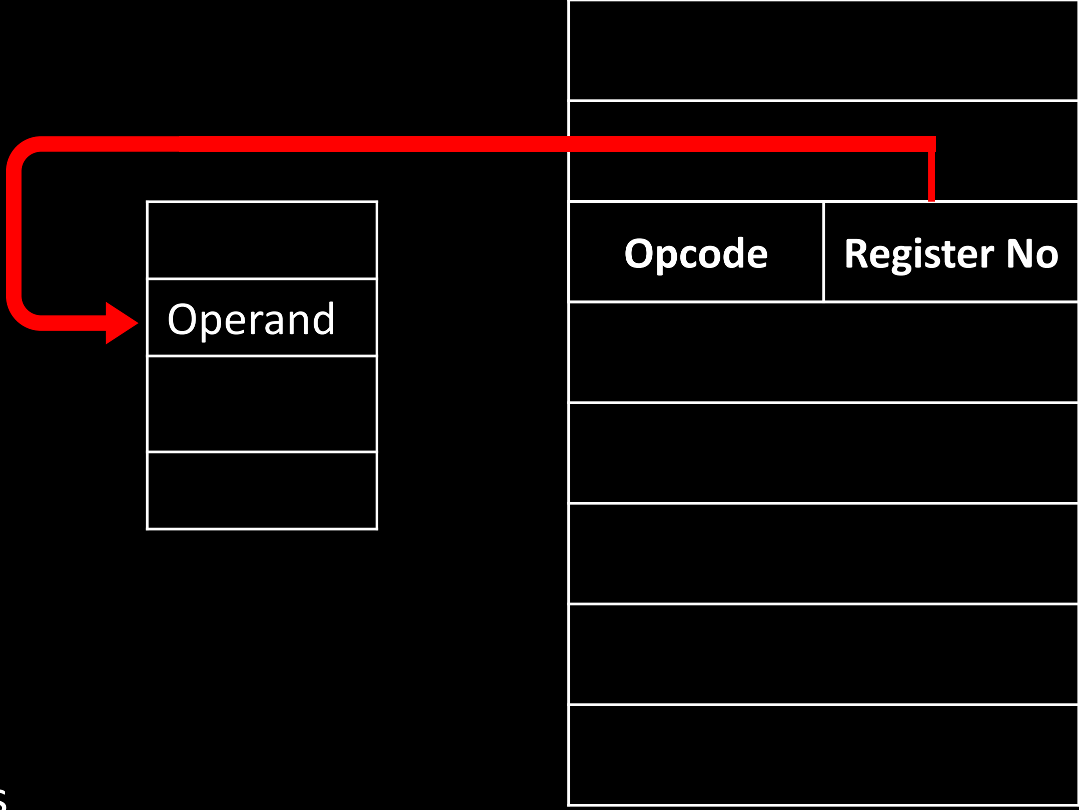
- **Disadvantage**
  - Relatively slow as memory must be referred more than one time.

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

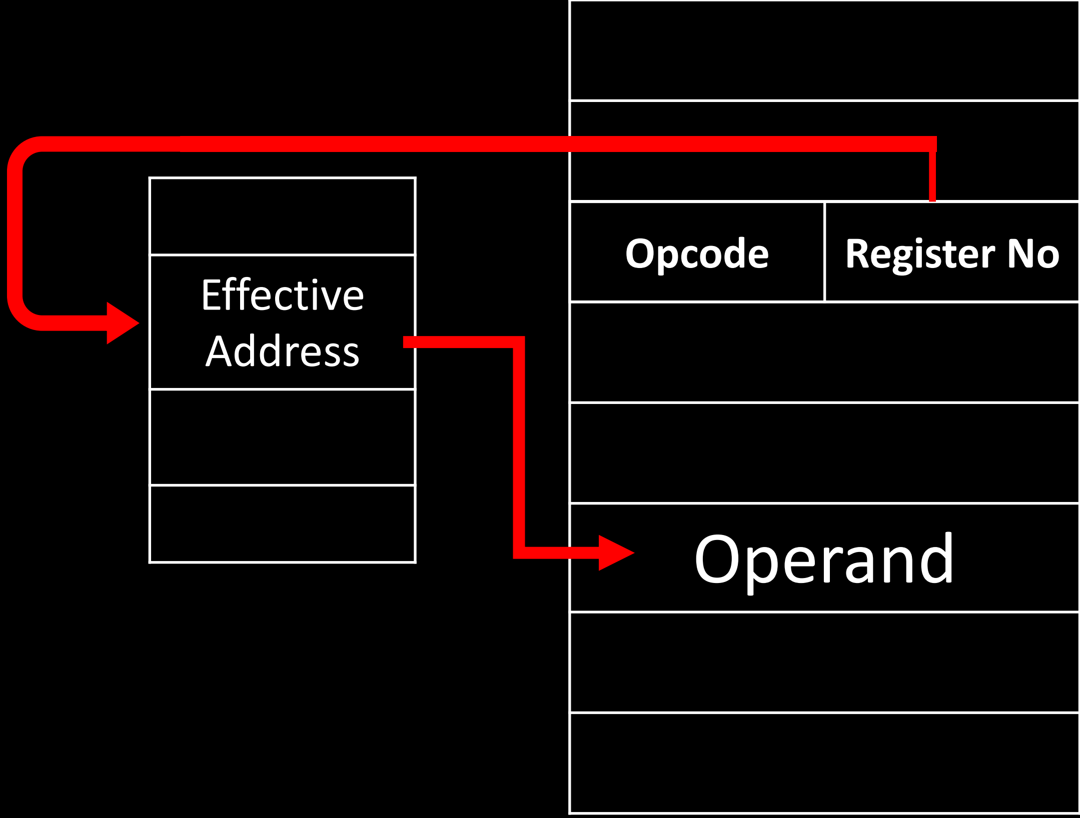
Register mode addressing

- 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

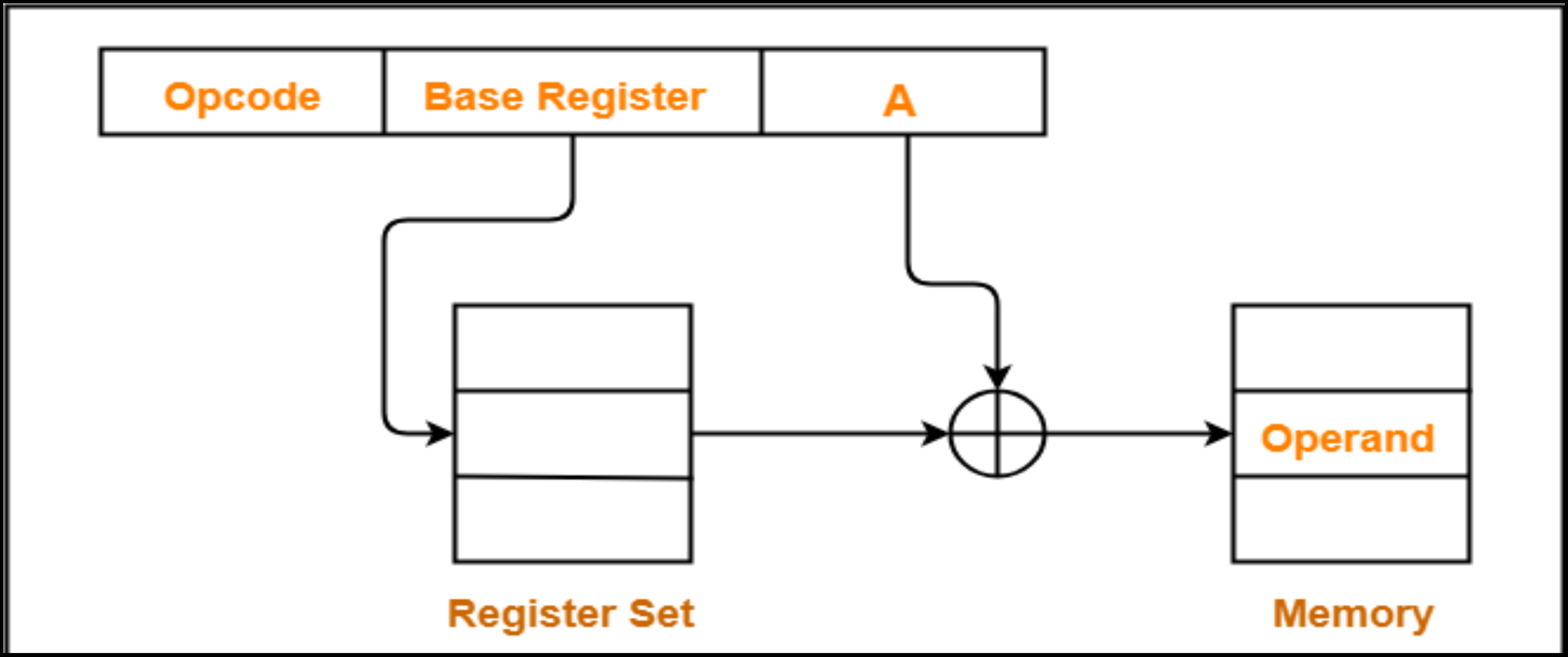
- In this mode in the instruction we will specify the address of the register where inside the register actual memory address of the variable is stored effective address.
- When same address is required again and again, this approach is very useful
- Here the same register can be used to provide different 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.

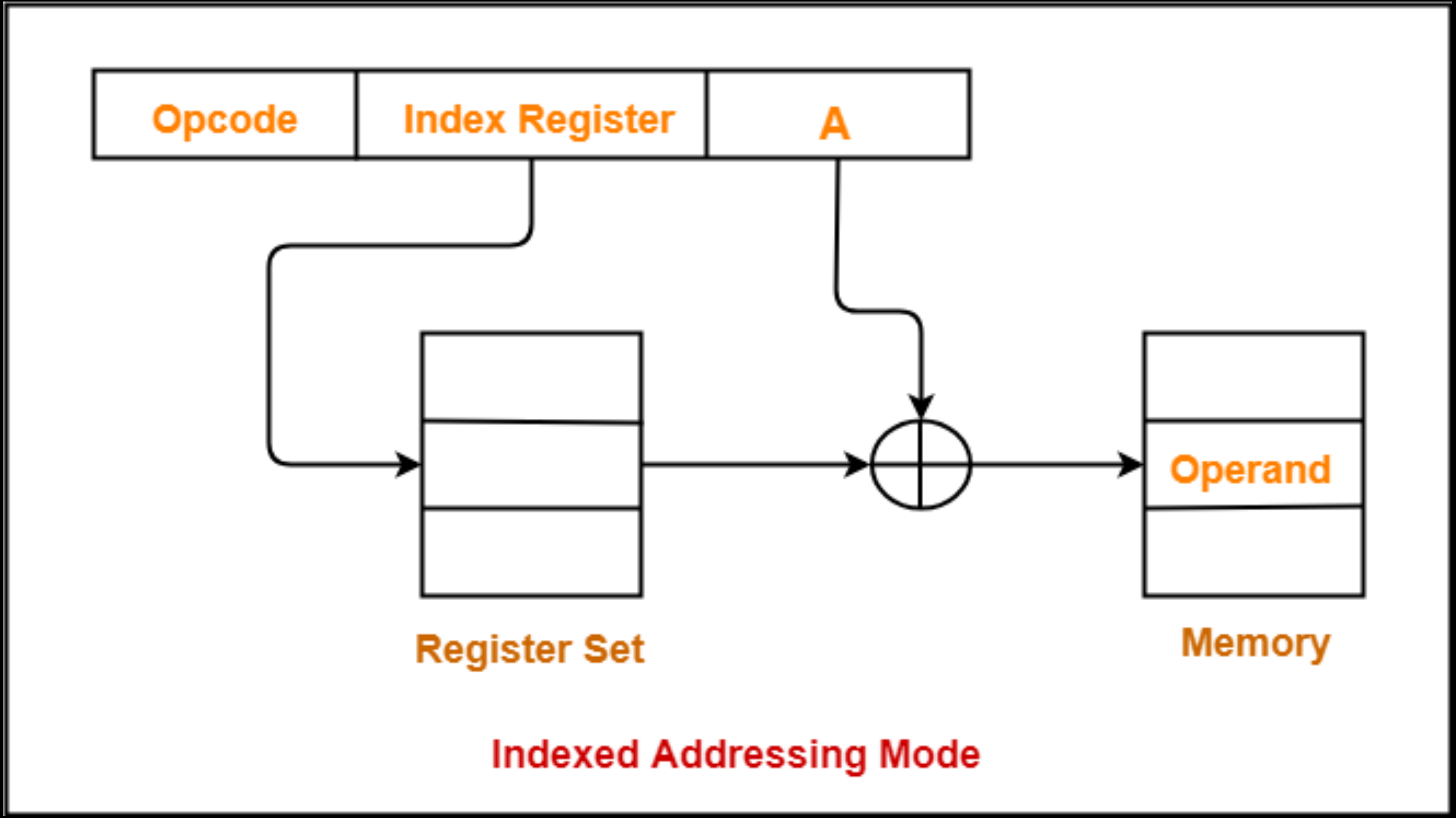
- Then instead to giving the direct branch address we give off set in the instruction
- Effective address = base address + off set (instruction)
- Now the advantage is even if we try to shift process in the memory, we only need to change the content of the base register.
- Final address will be the sum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



Index addressing mode

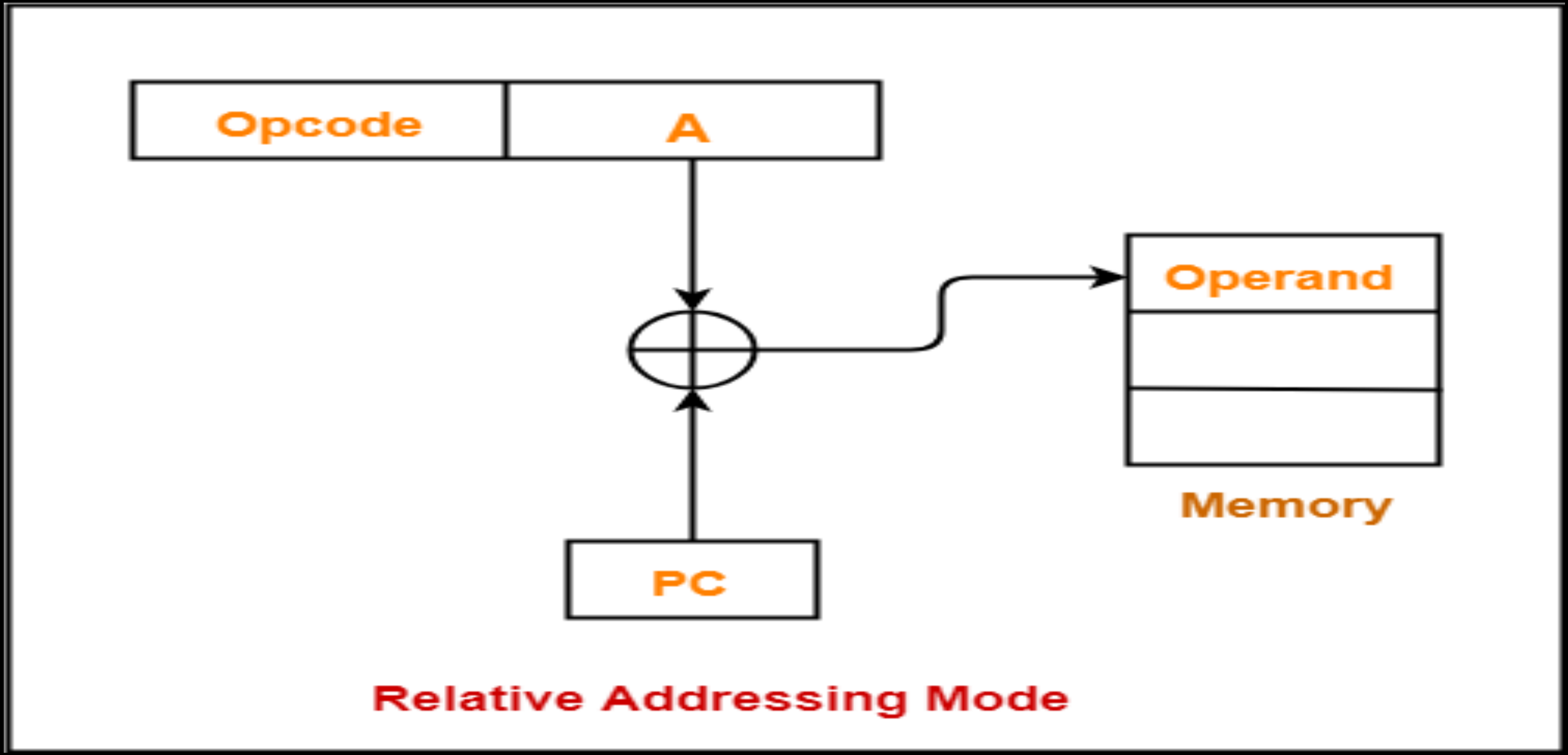
- This mode is generally used when CPU has a number of registers, and out of which one can be used an index register
- This mode is especially useful, when we try to access large sixed array elements.

- 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.


Opcode
A
Operand
Memory
PC
Relative Addressing Mode