

Chap 4pg 194

Chap t 5 pg 260

MICROPROCESSOR AND COMPUTER ARCHITECTURE

CACS-155

UNIT 1: Fundamental of Microprocessor

- Introduction to Microprocessors.
- Microprocessor System with Bus organization
- Microprocessor architecture and operation
- 8085 microprocessor and its operation
- 8085 instruction cycle, machine cycle, T states, addressing modes in 8085
- Introduction to 8086

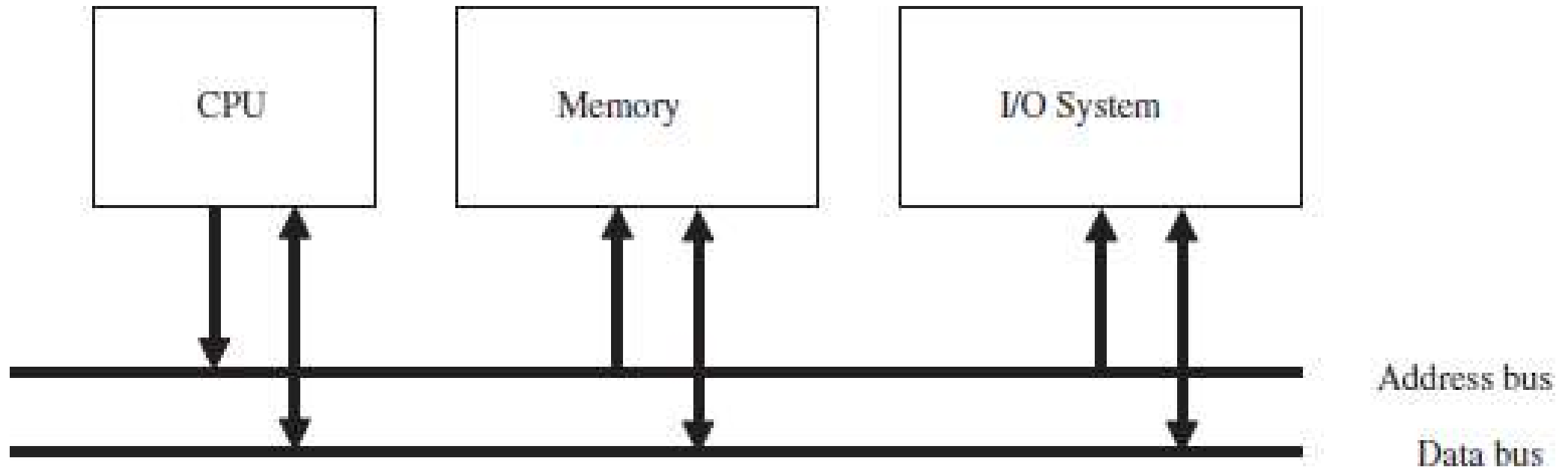
What is a microprocessor then?



What is a microprocessor then?

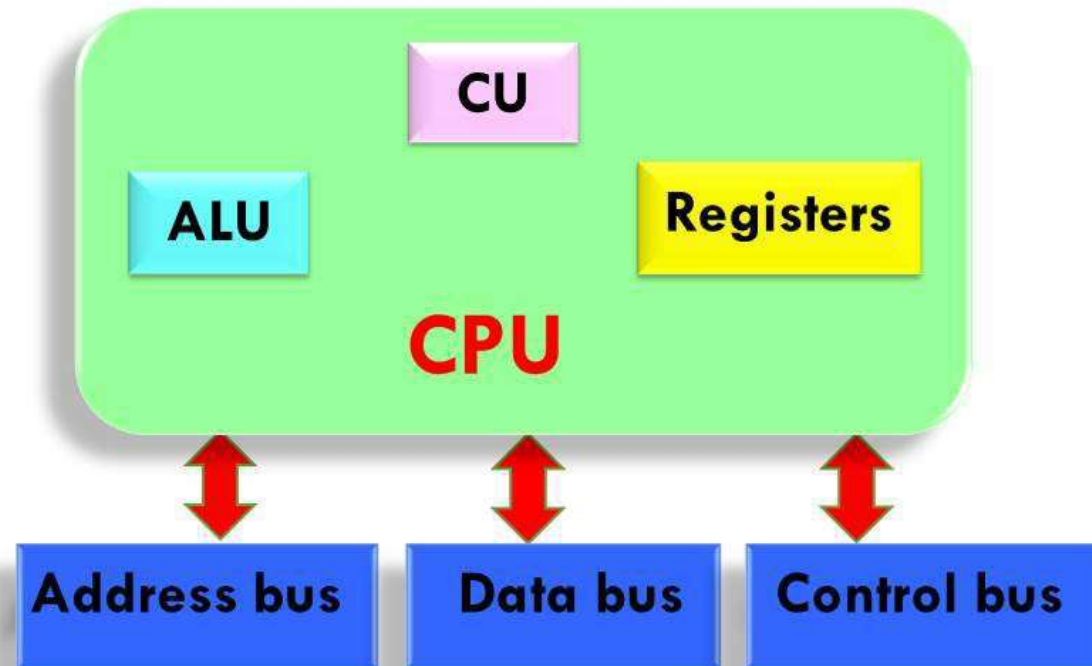
- Is a computer processor that incorporates the function of CPU in a single integrated circuit.
- Multipurpose, programmable, clock driven, register based electronic devices.
- Is a controlling unit for a computer.
- Can perform arithmetic and logical operations.
- Can communicate with other devices and i/o subsystems.

Lets look at microcomputer system



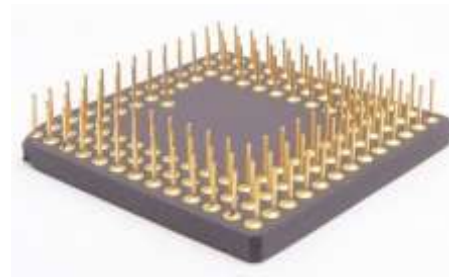
Microprocessor Consists of

MICROPROCESSOR – BLOCK DIAGRAM



Application Areas of a microprocessor

- First of all understand that the intel core i-7 kind of form factor are not the only microprocessors available. Some are like this.



1. Instrumentation

- Devices like frequency counters, function generators, spectrum analyzers, automated ventilators, life support systems, weather stations etc.
- Used for measurement of data both critical and non critical.

2. Control

- Microwave, washing machine, flight control system of a jumbo jet (explain MCAS of new max737)
- Controls different parameters based on instrumentation, like water supply in washing machine, microwave timing, flaps, slats and power of an airplane etc.

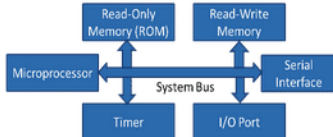
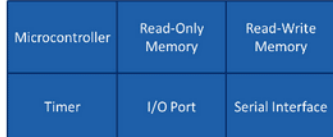
3. Communication

- Mobile phones, pagers, equipment on the telephone exchange, Internet routers (CISCO, JUNIPER).
- Used as controller as well as data communicator to relay large data from one side to other.

4. Consumer

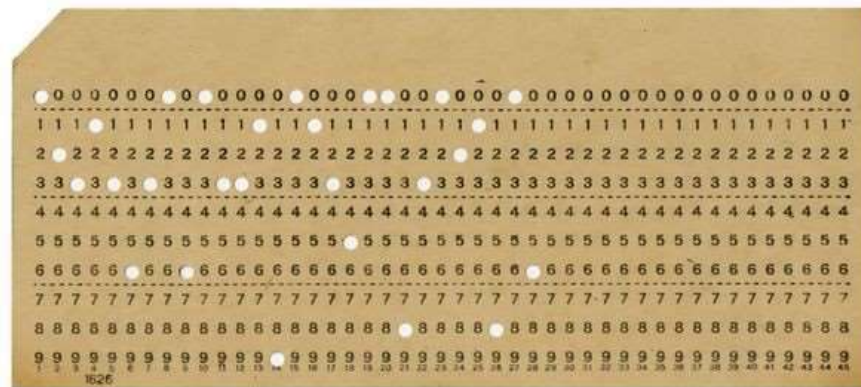
- At consumer end microprocessor is used for:
 - Calculations in calculator
 - Accounting system
 - Video Games
 - Smart watch
 - Traffic light control system
 - SCADA system
 - General microcomputer

Difference between microcontroller and microprocessor

Microprocessor	Micro Controller
	
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and I/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.
Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
Mainly used in personal computers	Used mainly in washing machine, MP3 players

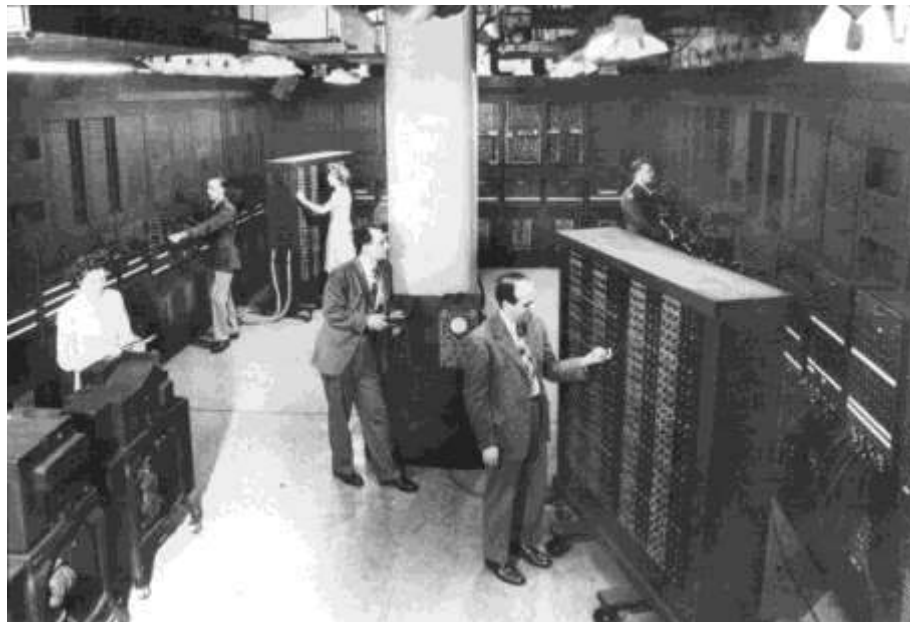
Historical Development of a computer

- Older computer were mechanical.
- Used to solve calculation problems.
- Charles Babbage made the first computers the difference engine and analytical engine to solve basic mathematical operation problems like addition and subtraction.



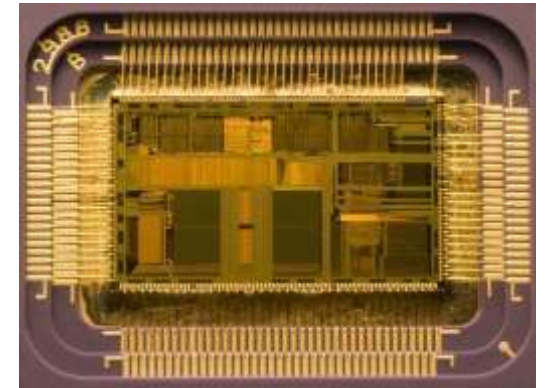
Historical Development of a computer

- Evolution of vacuum tubes made computers back then a electrical devices.
- ENIAC was the first programmable electronic computer.



Historical Development of a computer

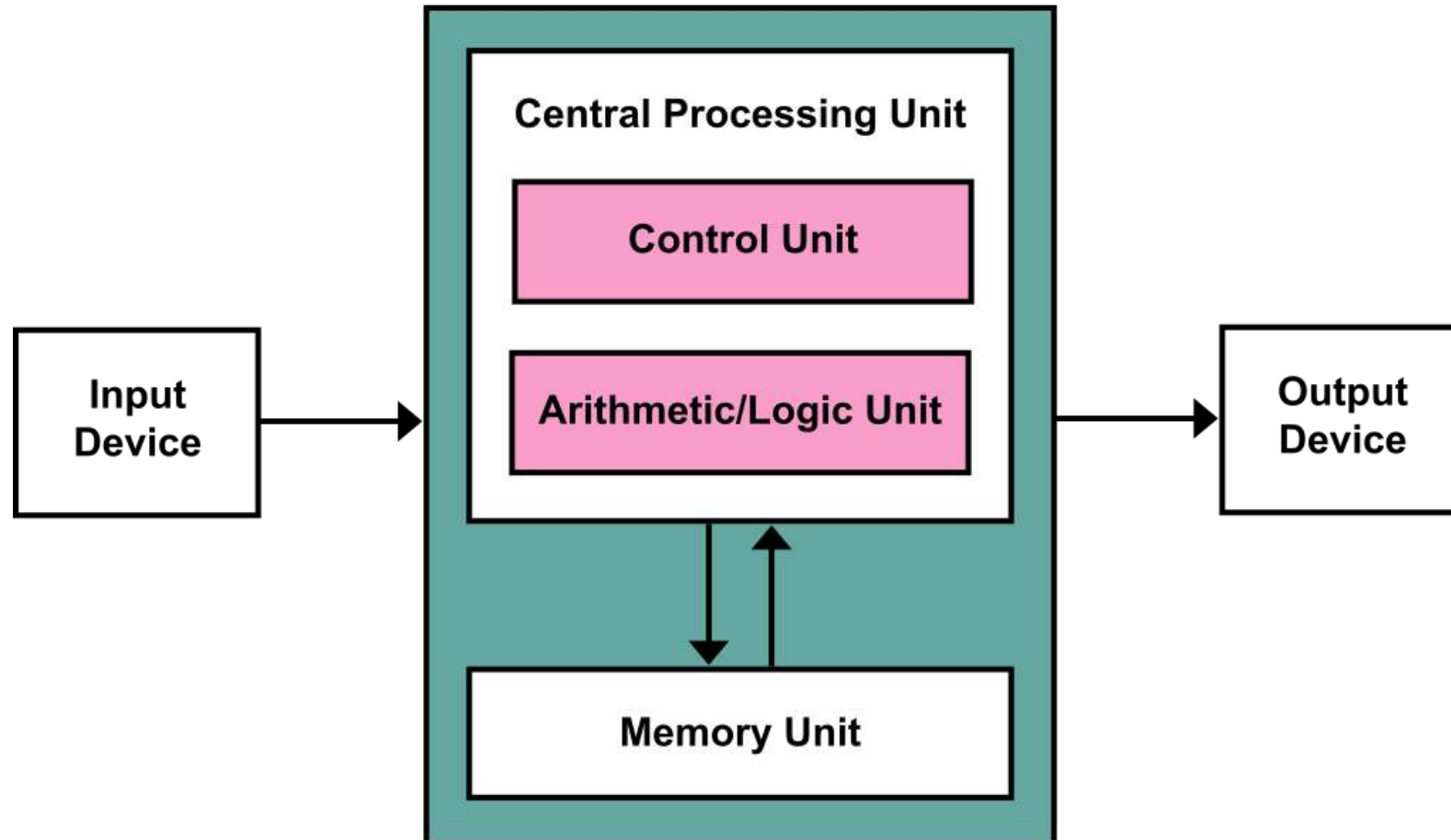
- Later on after the advent of transistors things changed and we have reached here where we have chargers more powerful than the computer that sent man to moon.



Stored Program concept

- A concept which emphasis on storing a computer instruction in electronically accessible medium.
- Stored program made it easy to change program in a computer system and make them do anything you like from calculations to editing a video.
- This concept made computers more general purpose.

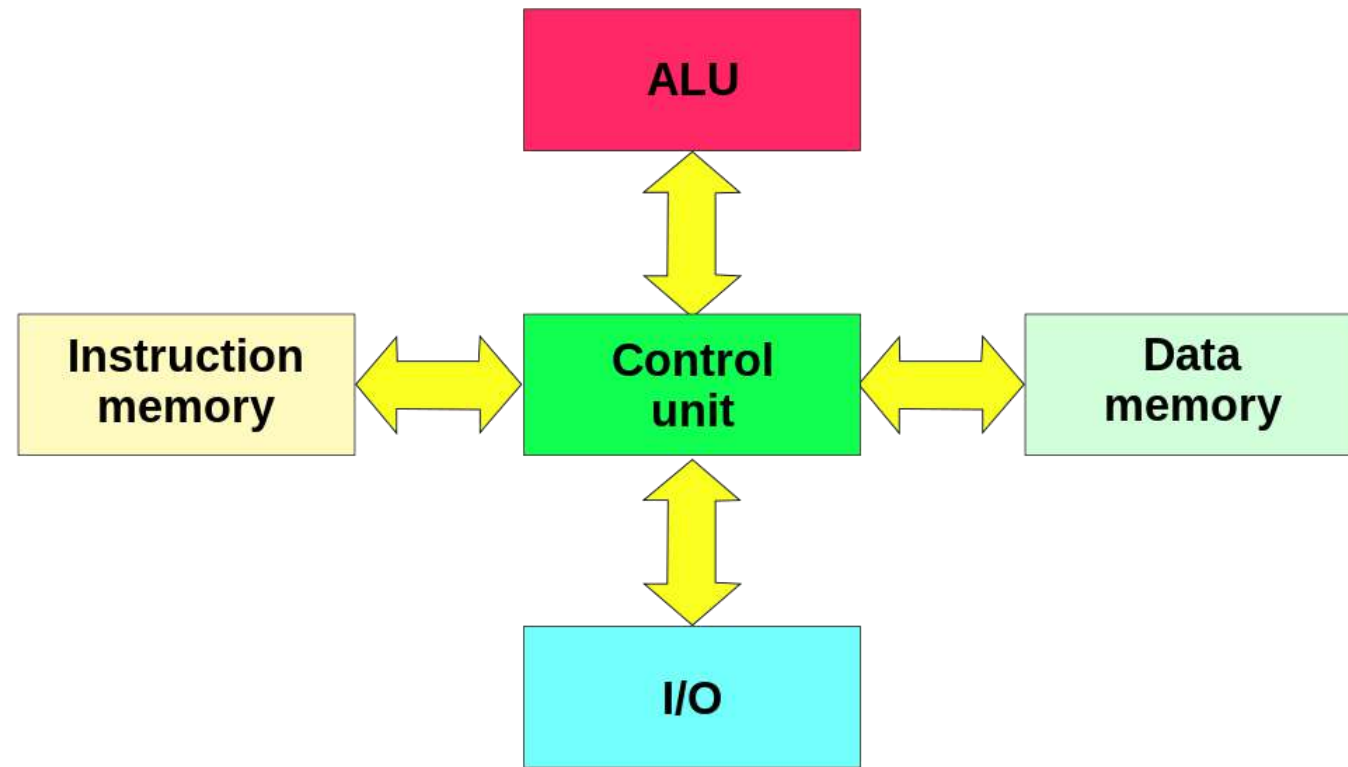
Von-Neumann Architecture



Von-Neumann Architecture

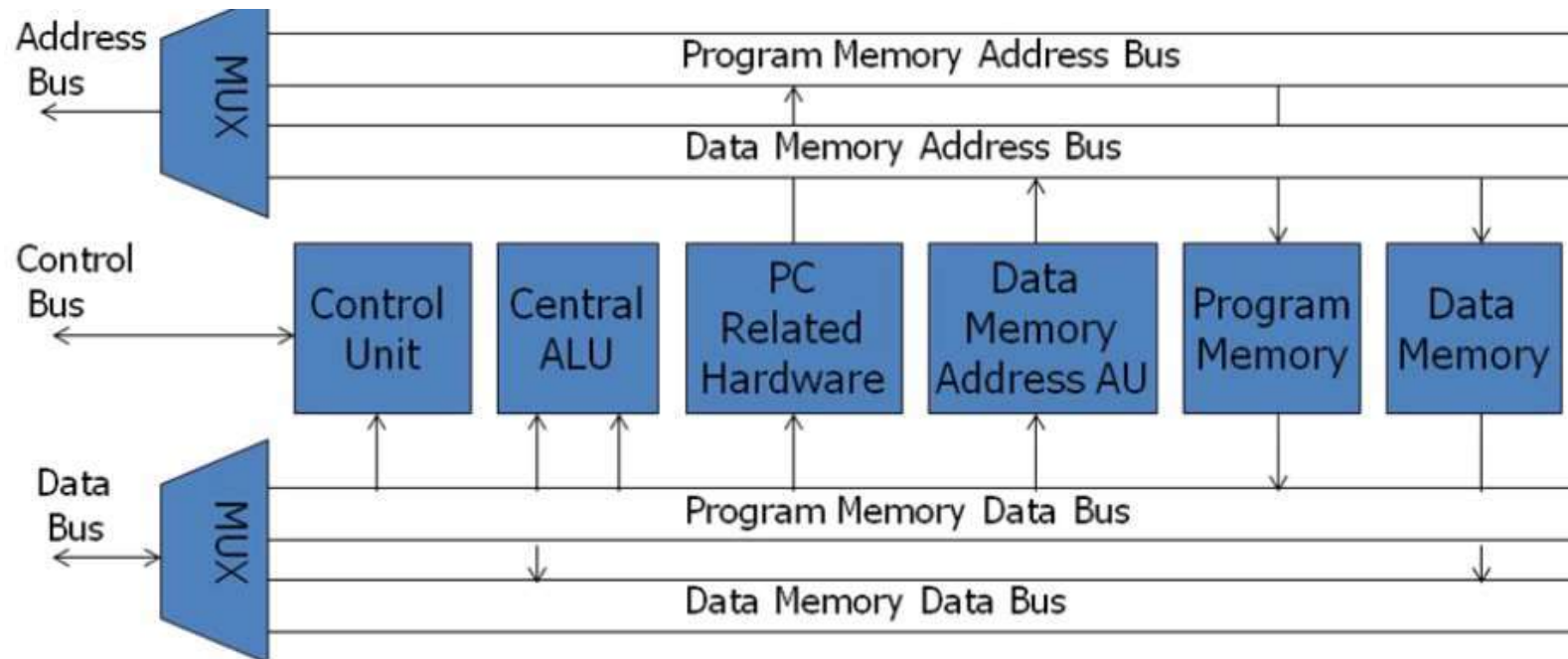
- A single memory is shared to both store data and the computer instruction.
- Fundamental basis for modern day computers.
- Various temporary scratchpad storage is allocated within the processor to store the fetched content from memory.
- MBR, MAR, IR, IBR, PC, ACC etc are the main registers used for the machine cycle operations.

Harvard Architecture

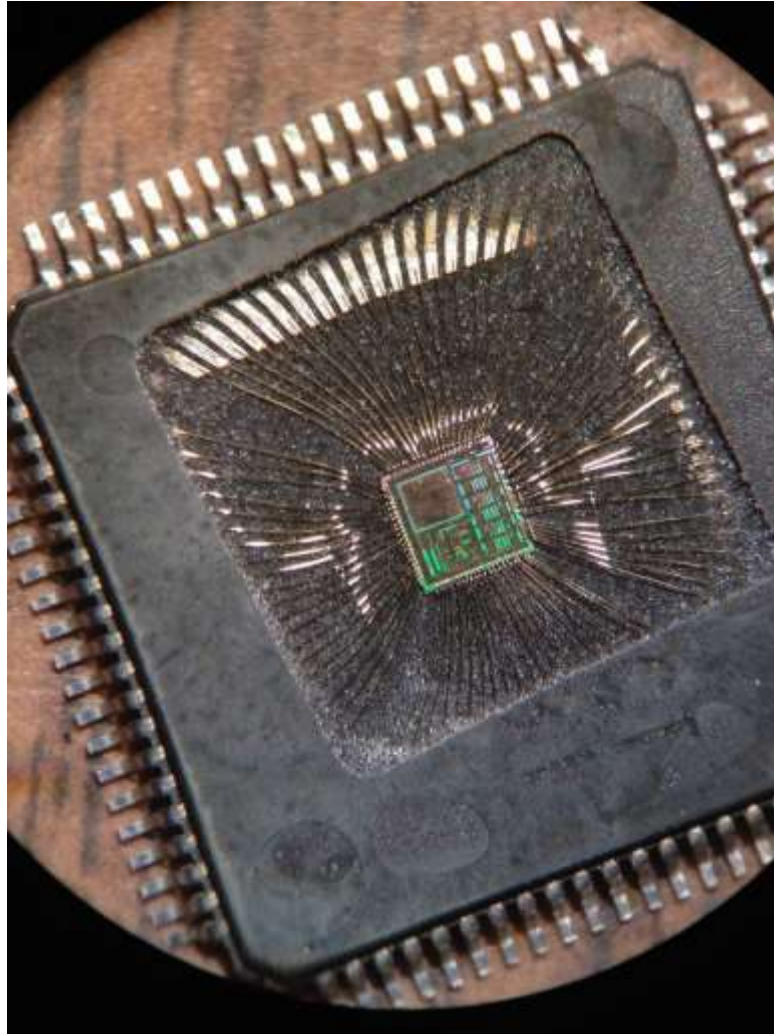


Harvard Architecture

- Separate memory for data and Instructions.
- Both the operand and operator can be fetched concurrently thus providing significant improvement in speed.

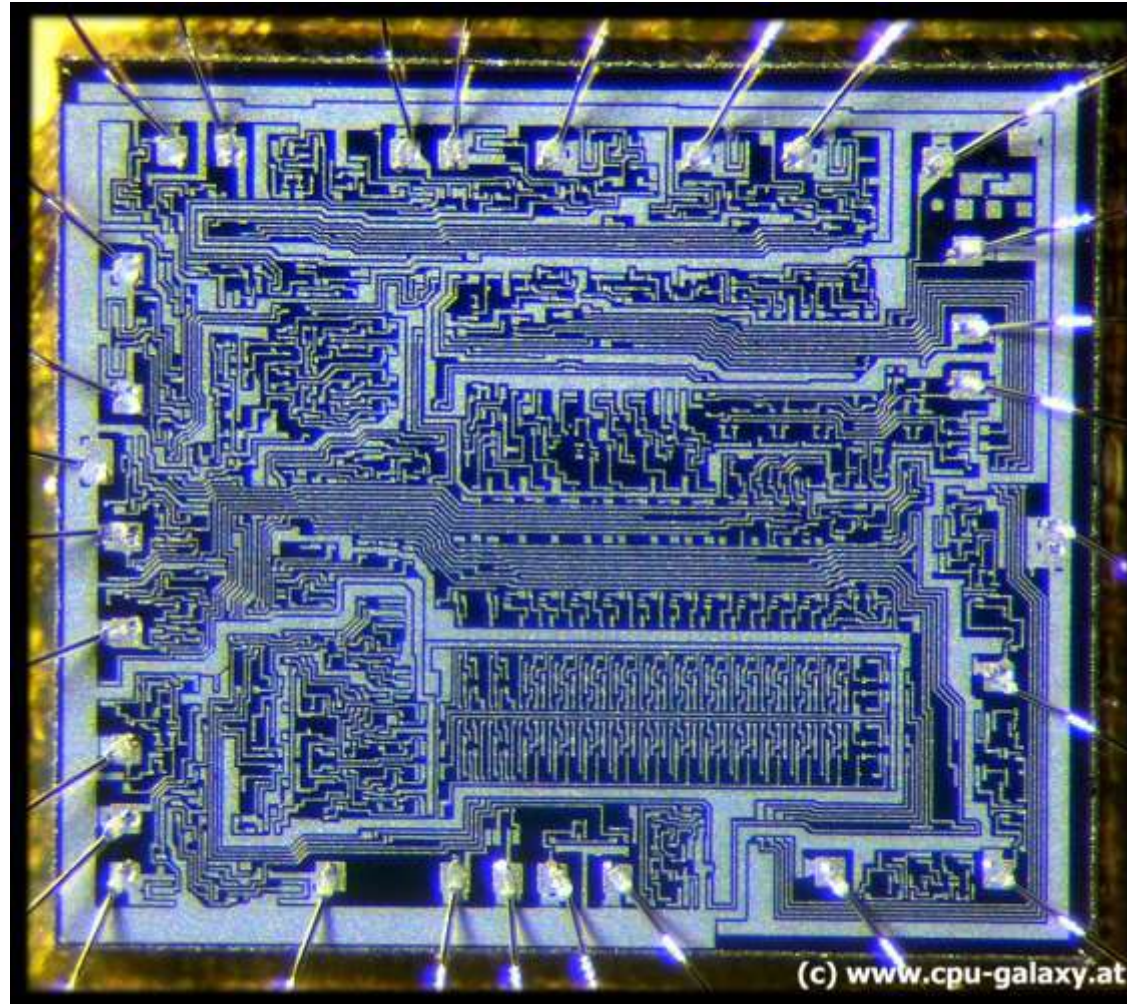


Details inside of a microprocessor



Niraj Khadka

Details inside of a microprocessor



Details inside of a microprocessor

- Control Unit
 - Associated with generating control signals for data flow.
- Arithmetic and Logical Unit
 - Performs arithmetic operations like add, multiplication, subtraction etc as well as perform logical operations like anding, oring, negating, bit shifting etc.
- Registers
 - Temporary storage or a scratchpad inside of a microprocessor to work. They consists of data or instructions and is operated. Basically consists of some numbers generally based on the processor technology.
 - For 8085: B,C,D,E,H,L,Accumulator etc.

Control Unit

- It gets instruction from memory.
- The control unit decides what the instructions mean and directs the necessary data to be moved from memory to ALU.
- It must communicate with both ALU and main memory.
- It coordinates all activities of processor unit, peripheral devices and storage devices.
- Two types:
 - Hardwired Control Unit
 - Micro-Programmed Control Unit

Hardwired Control Unit

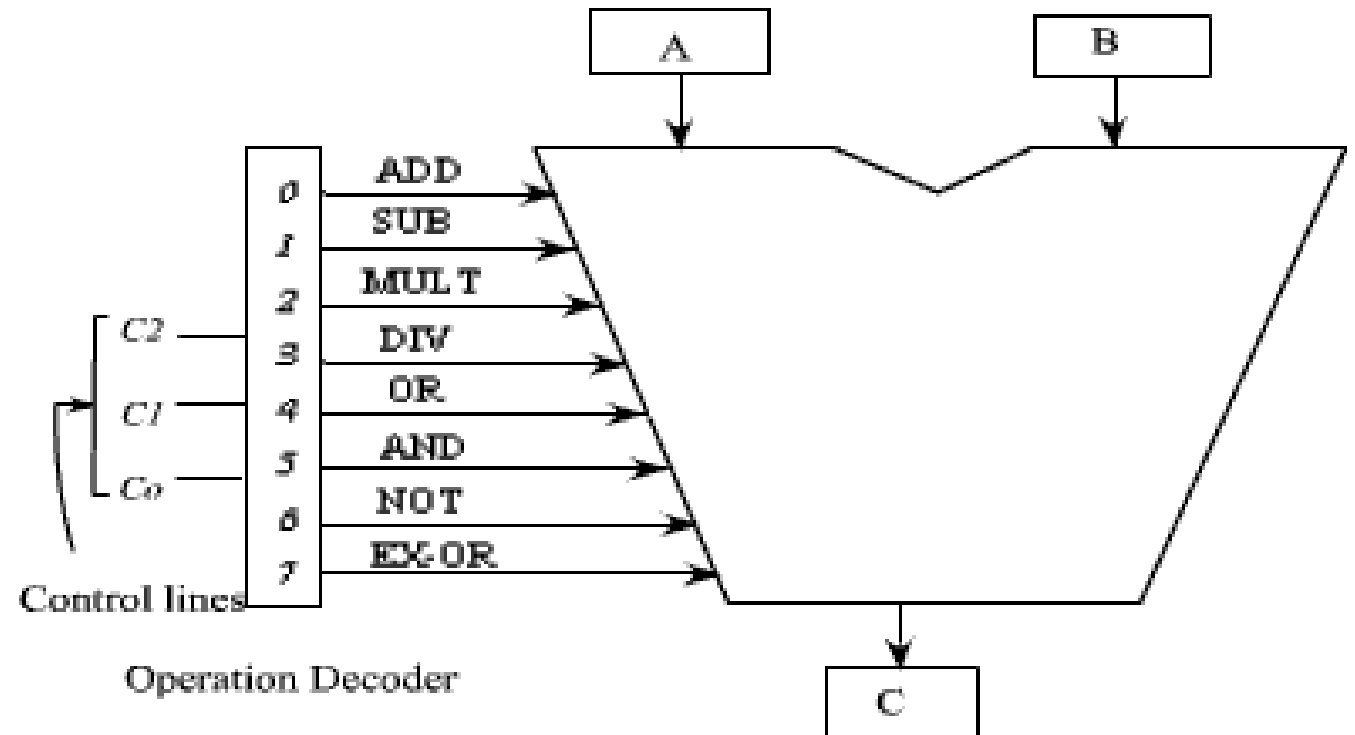
- is essentially a combinatorial circuit. It's i/p logic signals are transformed into set of o/p logic signals which are control signals.
- performs different operations in the basis of op-codes.
- have to derive the Boolean expression for each control signal as a function of input.
- Since modern processor needs a Boolean equation, it is very difficult to build a combinational circuit that satisfies all these operations.
- has faster mode of operation.
- needs rewiring if design has to be modified.

Micro-programmed Control Unit

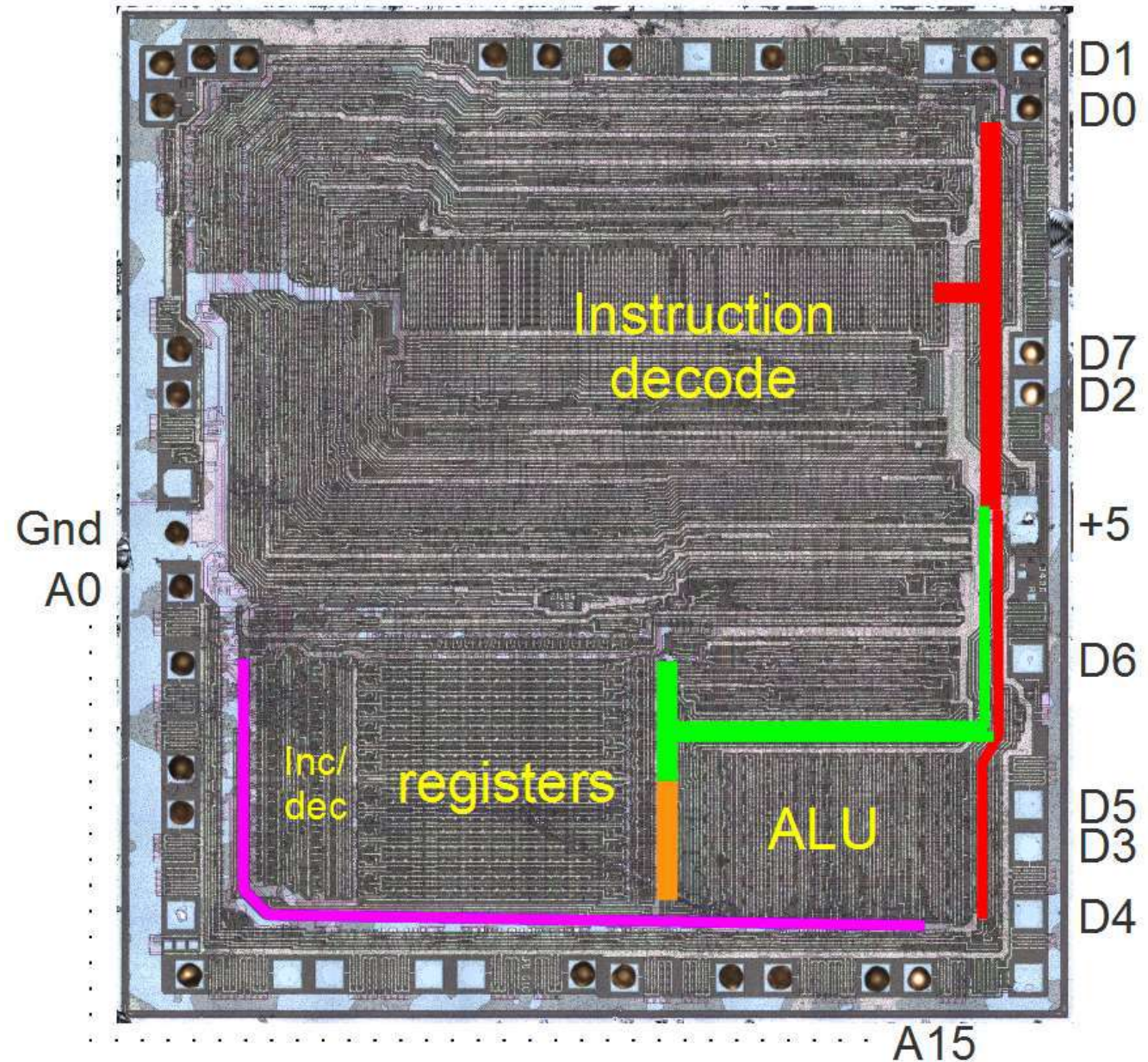
- An alternative to hardwired CU.
- the control information is stored in control memory.
- The control memory is programmed to initiate required sequence of operations.
- Use sequences of instructions to perform control operations performed by micro operations.
- Control address register contains the address of the next microinstruction to be read
- As it is read, it is transferred to control buffer register.
- Sequencing unit loads the control address register and issues a read command.
- is cheaper and simple than hardwired CU.
- is slower than hardwired CU.

ALU

- Combinational digital electronics circuit.
- Performs logical and arithmetic operations on integer binary numbers.
- Fundamental building block of computing circuits.



Registers



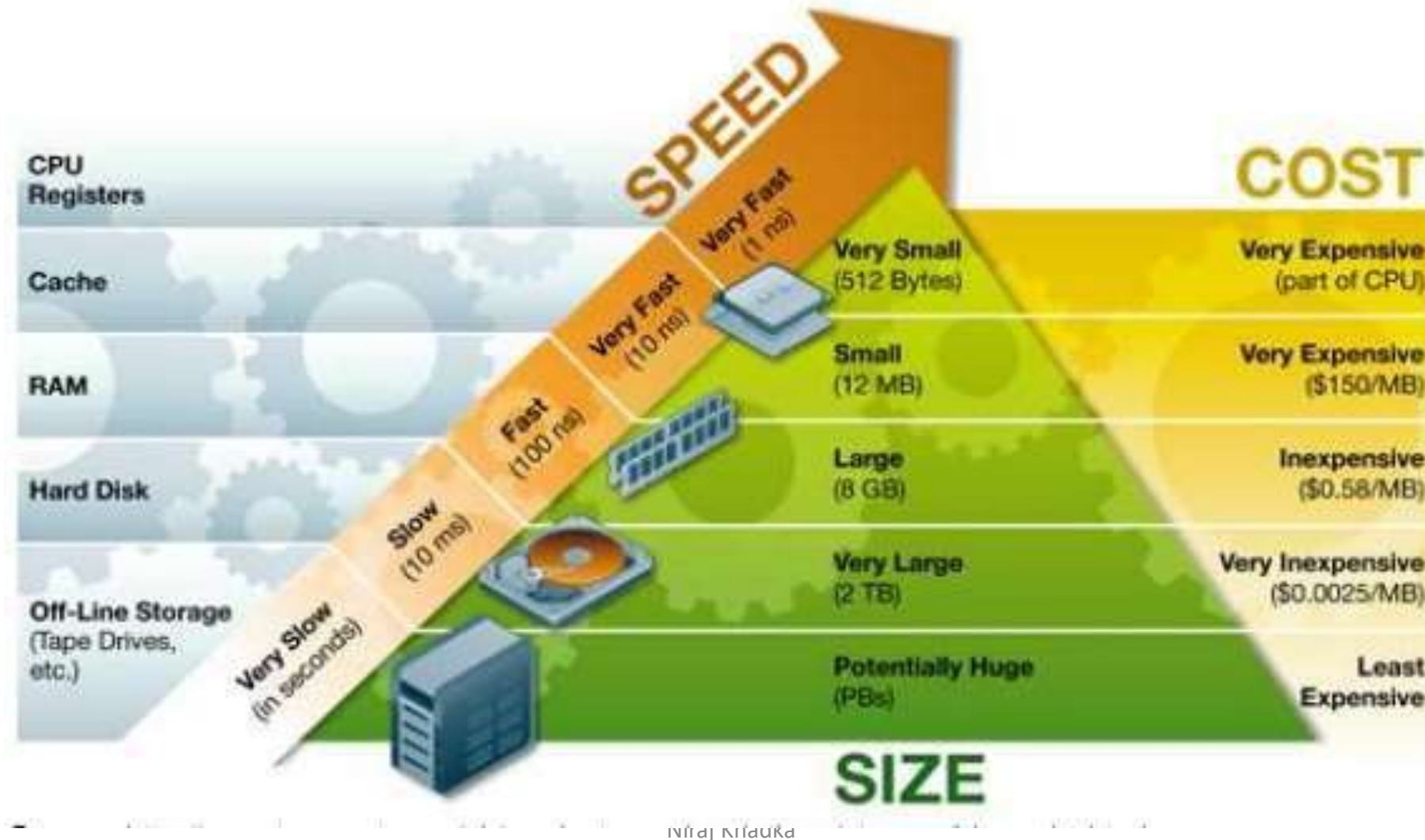
Registers

- Type of computer memory used for quickly accept, store, transfer data and instructions that are being actively used or required by the microprocessor.
- It is a scratchpad memory inside a microprocessor.
- A microprocessor needs the register bank or register array to hold information like current opcode, current data as operands, current flag values or status, current instruction pointer, current stack location pointer etc to complete the instruction cycle.

Memory

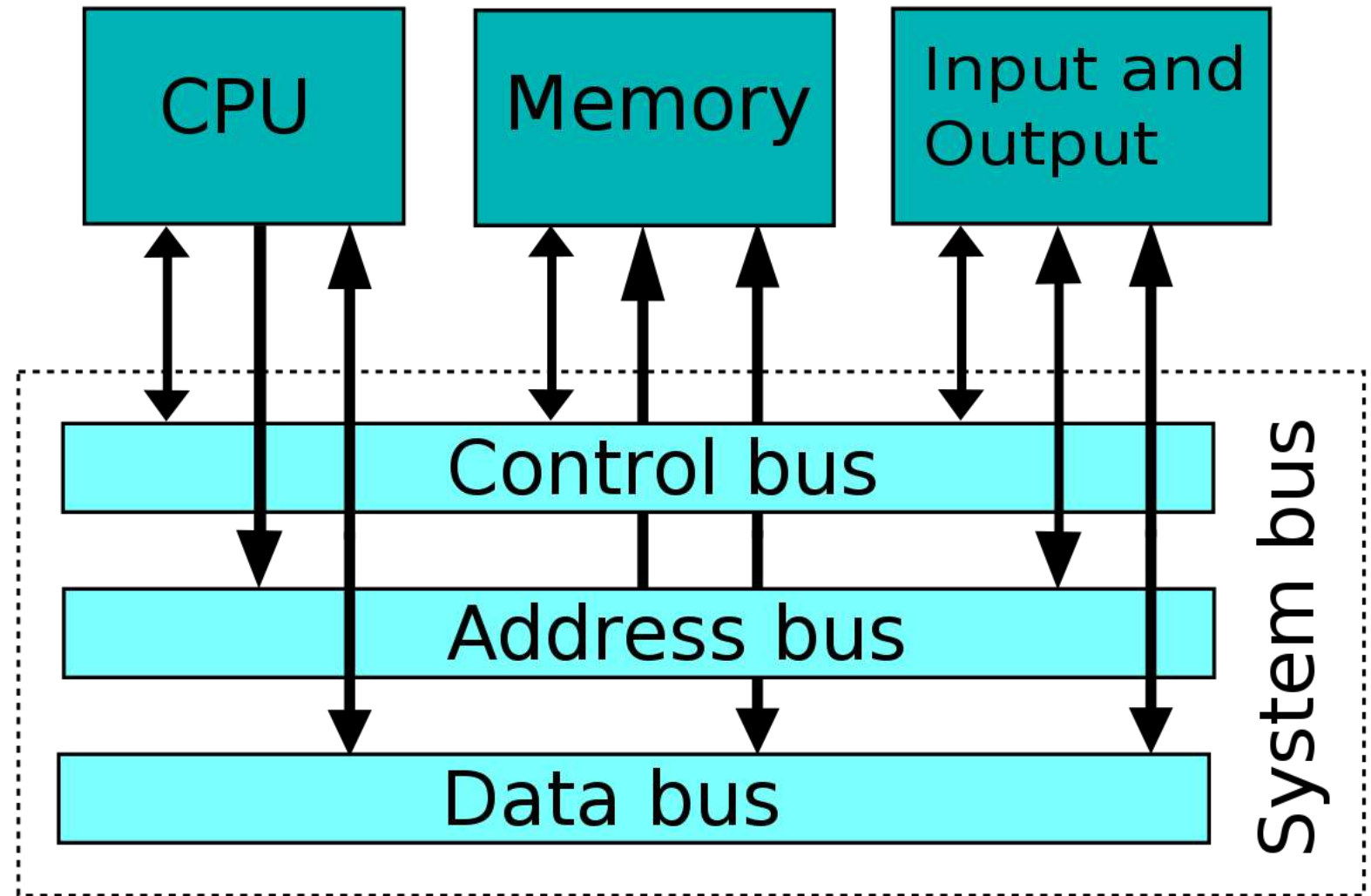
- Space to store data inside the computer.
- Are of many types, generally primary and secondary.
- Primary are RAM, cache and registers.
- Secondary are; SSD, HDD, optical storage.

Memory Hierarchy



System Bus

- 3 Bus
 - Address Bus
 - Data Bus
 - Control Bus



Address Bus

- Bus used to carry the address of memory location.
- Can be of any size ranging from 4 bits to 32 bit and more.
- The address bus width limits the size of addressable physical memory.

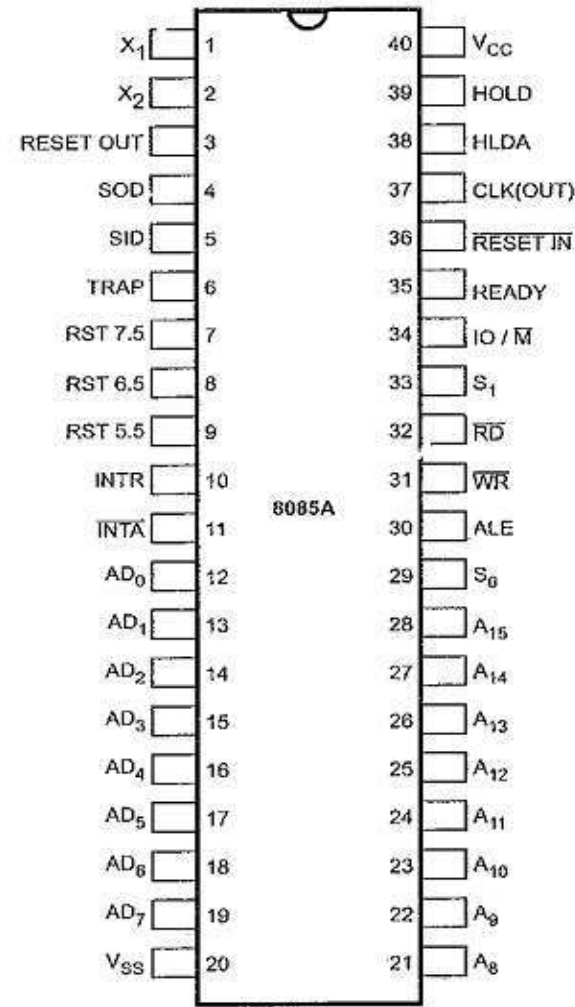


Fig. 1.3 (a) Pin configuration

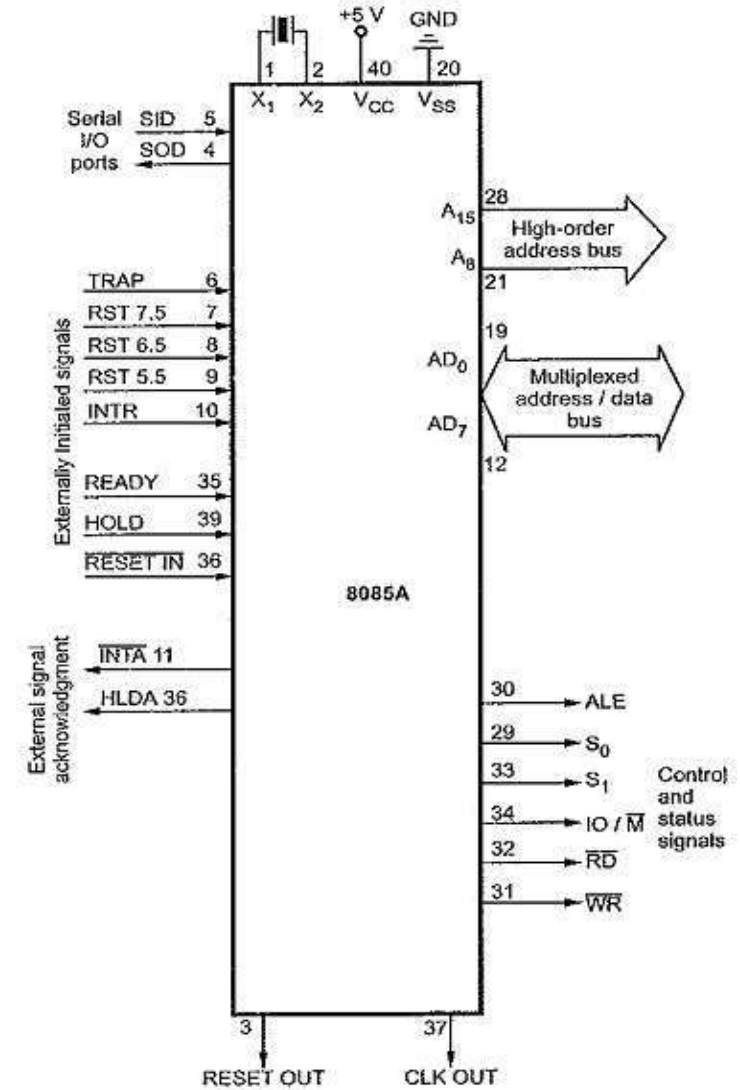


Fig. 1.3 (b) Functional pin diagram

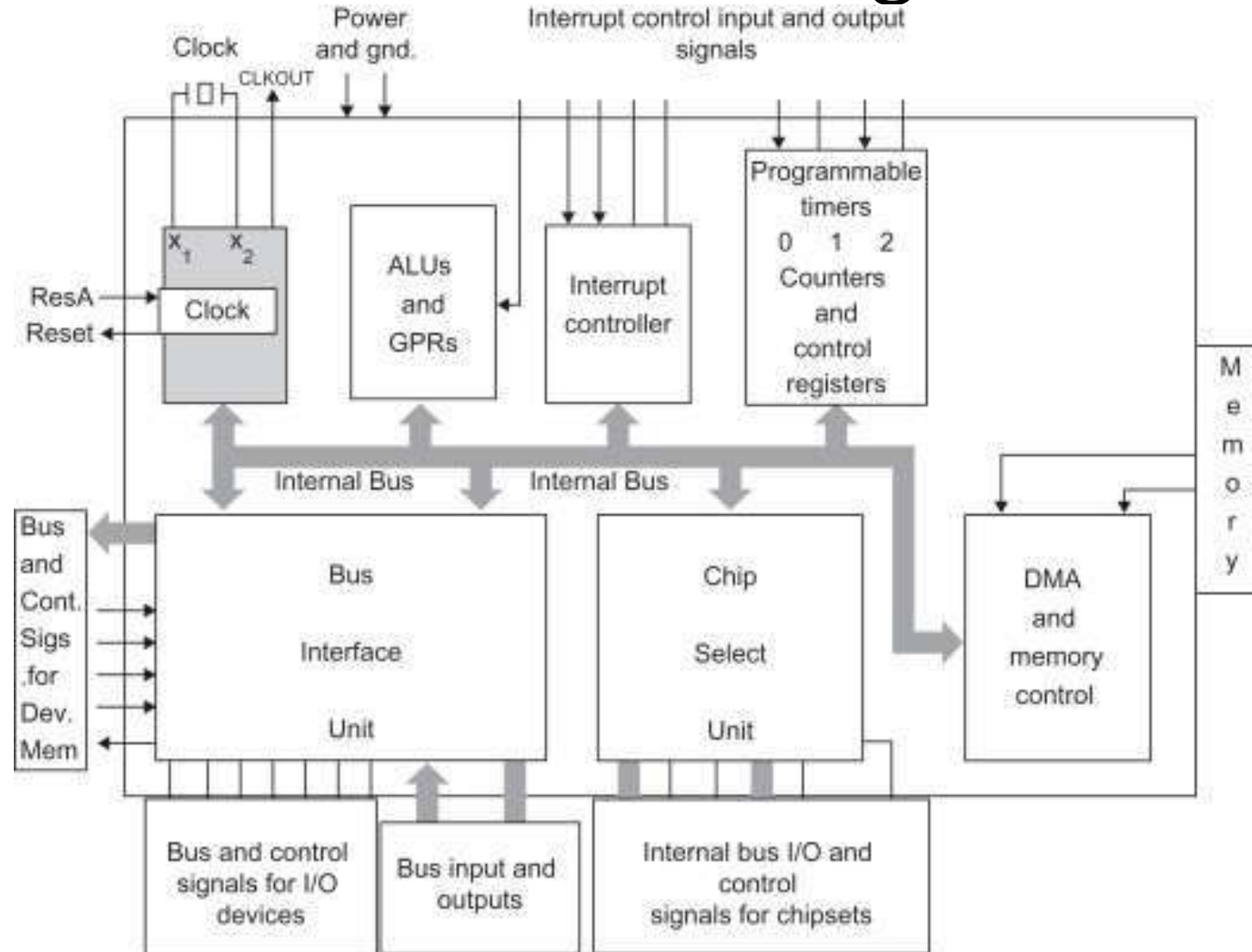
Data Bus

- Is of the bit depth of the microprocessor or main registers available.
- Used to carry the data from the processor to memory or vice versa.
Can also be used to supply data from the processor to the i/o devices.
- Can read or write the bits parallel based on the width of data bus.
- Generally ranges from 4 bit to 32 bits and more.

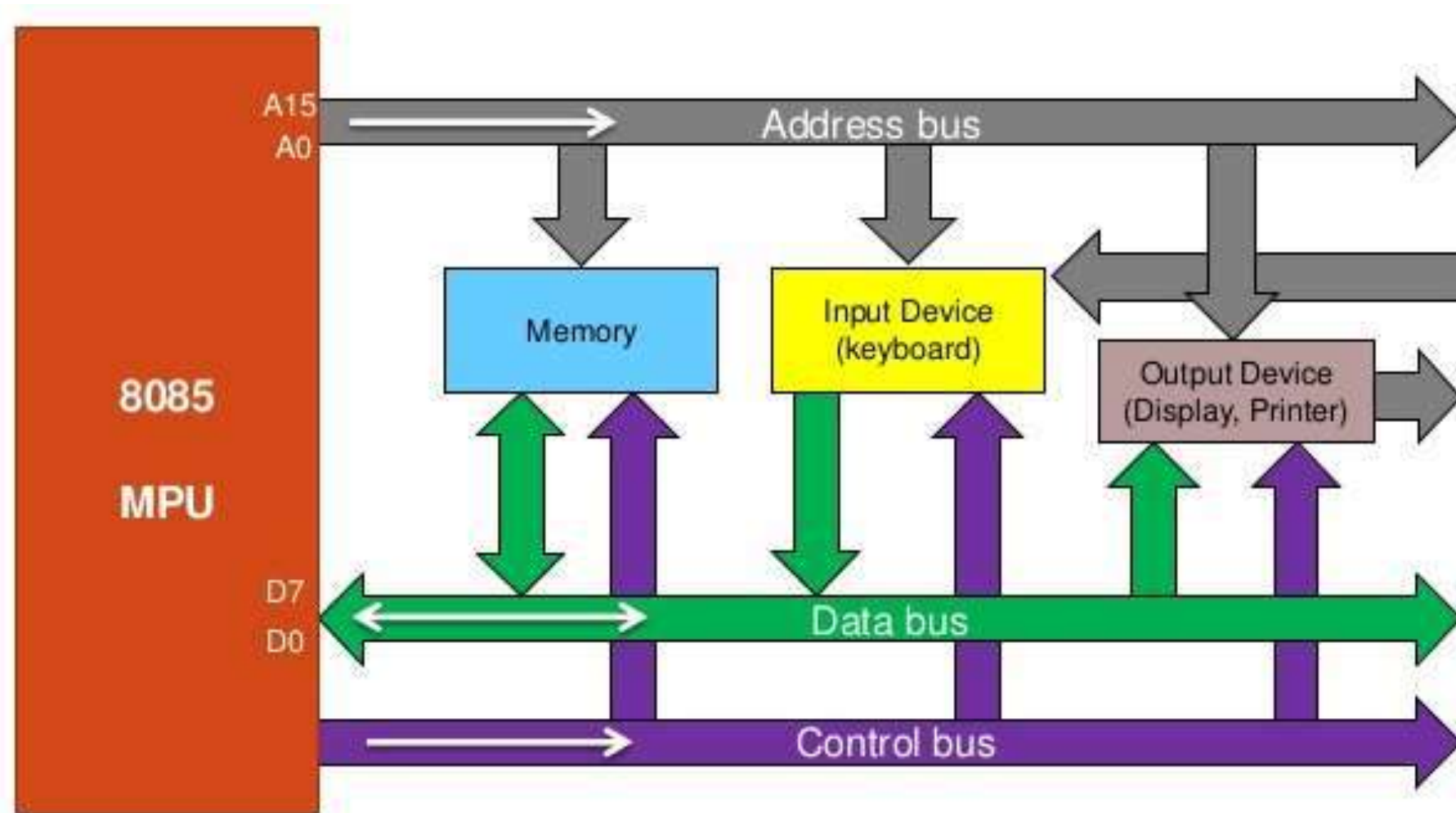
Control Bus

- Consists of 4-10 parallel signal lines.
- Sends control signal from these bus.
- Signals may include read operation, write operation, chip select operation, bus request/grant signals, clock signals, reset signals, interrupt request/ack signals.

Microprocessor with BUS organization



Intel 8085 bus organization



INTEL 8085 microprocessor

- 8-bit data bus
- 16-bit address bus, which can address up to 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock

8085 Functional Units

- Accumulator
 - 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.
- Arithmetic and Logical Unit
 - performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.
- General Purpose Registers
 - 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.
 - These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

8085 Functional Units

- Program Counter
 - 16-bit register used to store the memory address location of the next instruction to be executed.
 - Microprocessor increments the program counter whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.
- Stack Pointer
 - 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.
- Temporary Register
 - 8-bit register, which holds the temporary data of arithmetic and logical operations.

8085 Functional Units

- Flag Register
 - 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.
 - These are the set of 5 flip-flops –
 - Sign (S)
 - Zero (Z)
 - Auxiliary Carry (AC)
 - Parity (P)
 - Carry (C)

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

8085 Functional Units

- Instruction Register and Decoder
 - 8-bit register
 - When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.
- Timing and Control Unit
 - provides timing and control signal to the microprocessor to perform operations.
 - Control Signals: READY, RD', WR', ALE
 - Status Signals: S0, S1, IO/M'
 - DMA Signals: HOLD, HLDA
 - RESET Signals: RESET IN, RESET OUT

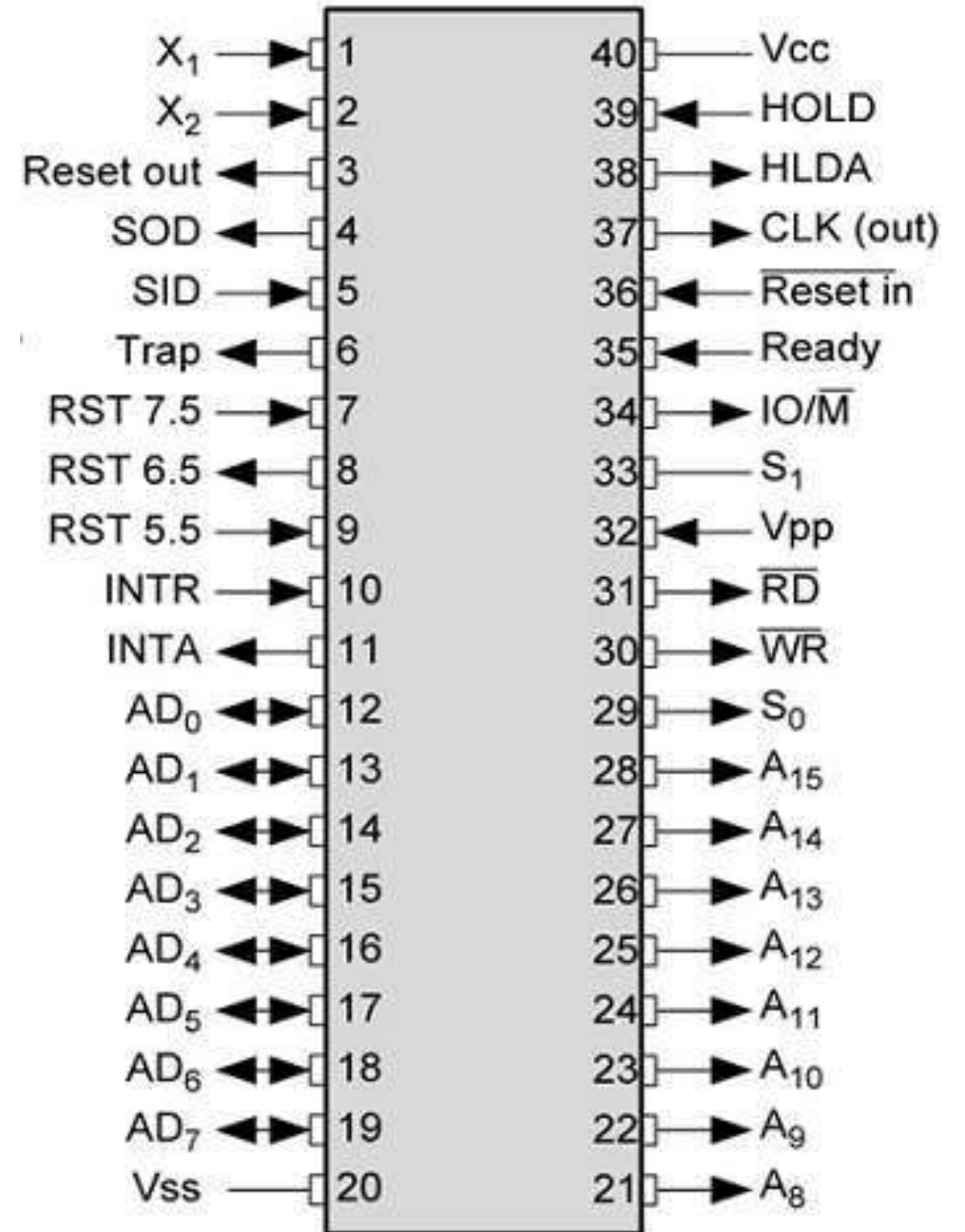
8085 Functional Units

- Interrupt Control
 - Control interrupt during a process.
 - 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.
- Serial I/O control
 - It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

8085 Functional Units

- Address buffer and Address data-buffer
 - content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU.
 - memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.
- Address Bus and Data Bus
 - Data bus carries the data to be stored. It is bidirectional.
 - Address bus carries the location to where it should be stored and it is unidirectional.

8085 Pin Diagram



Details of the 8085 Pins

- A8 – A15
 - it carries the most significant 8-bits of memory/IO address.
- AD0 – AD7
 - It is a multiplexed bus consisting of least significant 8-bit of memory I/O address as well as 8bit data bus.
- Control and Status Signals
 - Used to identify the nature of operation.
 - RD', WR', ALE control signal
 - IO/M', S0, and S1

Status Signals Meaning

MACHINE CYCLE STATUS:

IO/\overline{M}	S_1	S_0	Status
0	0	1	Memory write
0	1	0	Memory read
1	0	1	I/O write
1	1	0	I/O read
0	1	1	Opcode fetch
1	1	1	Interrupt Acknowledge
*	0	0	Halt
*	X	X	Hold
*	X	X	Reset

* = 3-state (high impedance)
X = unspecified

S_1 can be used as an advanced R/\overline{W} status. IO/\overline{M} , S_0 and S_1 become valid at the beginning of a machine cycle and remain stable throughout the cycle. The falling edge of ALE may be used to latch the state of these lines.

Details of the 8085 Pins

- Power Supply
 - VCC indicates +5v power supply and VSS indicates ground signal.
- Clock Signal
 - X1, X2 – A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
 - CLK OUT – This signal is used as the system clock for devices connected with the microprocessor.

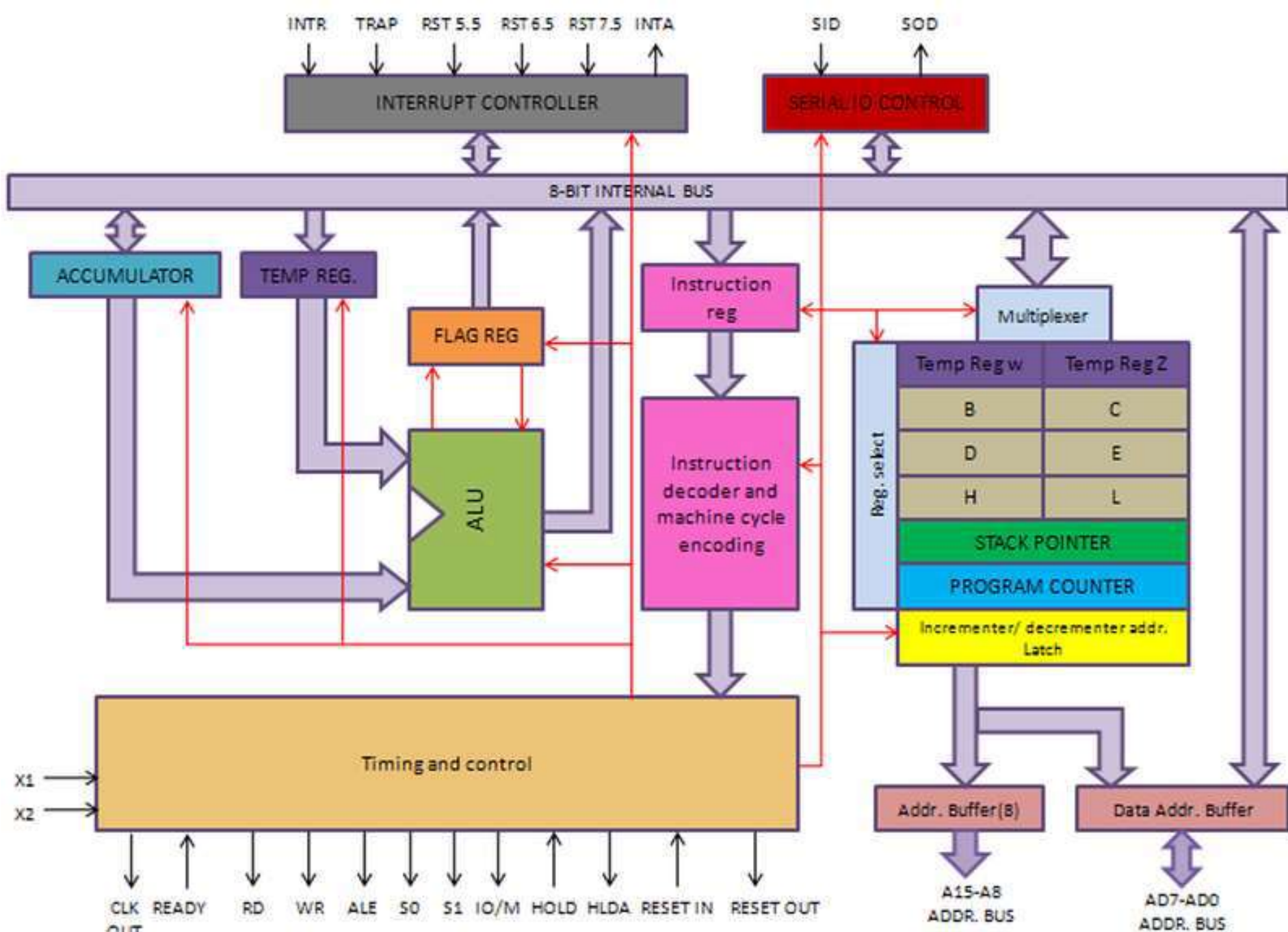
Externally initiated signal pins of 8085

- INTA – It is an interrupt acknowledgment signal.
- RESET IN – This signal is used to reset the microprocessor by setting the program counter to zero.
- RESET OUT – This signal is used to reset all the connected devices when the microprocessor is reset.
- READY – This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- HOLD – This signal indicates that another master is requesting the use of the address and data buses.
- HLDA (HOLD Acknowledge) – It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

Details of the 8085 Pins

- Interrupt pins
 - There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.
- 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.
 - SOD (Serial output data line) – The output SOD is set/reset as specified by the SIM instruction.
 - SID (Serial input data line) – The data on this line is loaded into accumulator whenever a RIM instruction is executed.

8085 ARCHITECTURE



Instruction Description and Format

- Instruction manipulates data.
- Instructions in sequence is called a program.
- Instruction has two parts; operation (op-code) and operands (data or address field).
- Data can be 8 bit, 16 bit , a register, a memory location.
- The opcode specifies how data is manipulated.

Instruction Description and Format

- For EG:

- MOVE X, Y

where; MOVE= opcode

X = source operand,

Y = destination operand ,

- Instruction format:

- 1 address format (1 byte) = here, 1 byte specifies the operation and the operands are default. Eg: ADD B, MOV B,D
 - 2 address format (2 byte) = First byte = opcode and one operand, 2nd byte = operand
Eg: IN 40H, MVI A, FFH
 - 3 address format (3 byte) = First byte = opcode, 2nd byte and 3rd byte = operand/data
 - 2nd byte = lower order data, 3rd byte = higher order data.

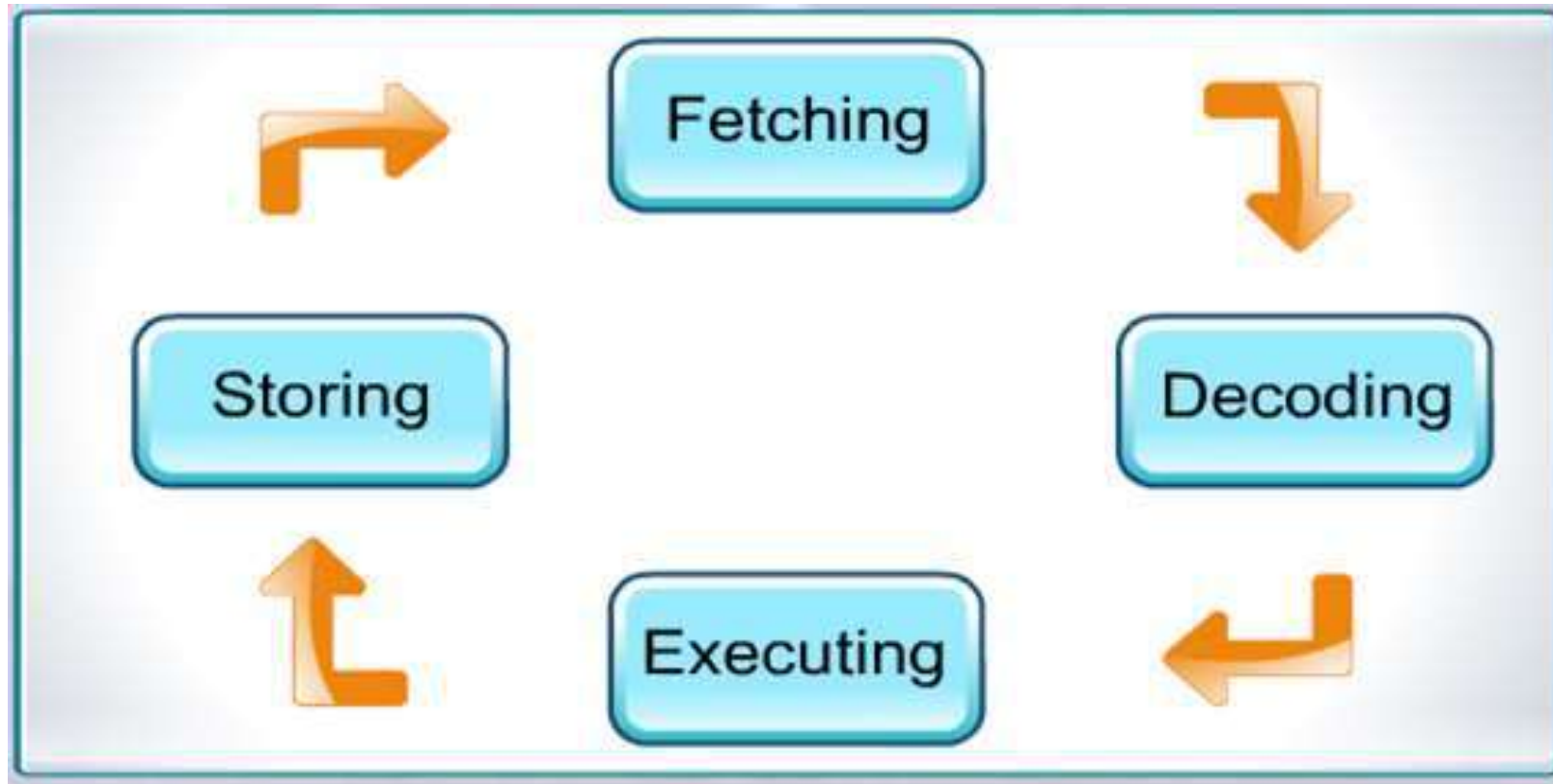
8085 Instruction Cycle

- Time required to execute and fetch an entire instruction is called *instruction cycle*.
- Consists of:
 - **Fetch cycle** – The next instruction is fetched by the address stored in program counter (PC) and then stored in the instruction register.
 - **Decode instruction** – Decoder interprets the encoded instruction from instruction register.
 - **Reading effective address** – The address given in instruction is read from main memory and required data is fetched. The effective address depends on direct addressing mode or indirect addressing mode.
 - **Execution cycle** – consists memory read (MR), memory write (MW), input output read (IOR) and input output write (IOW)

Machine Cycle

- For every instruction, a processor repeats a set of four basic operations, which comprise a machine cycle: (1) fetching, (2) decoding, (3) executing, and, if necessary, (4) storing.
- Fetching is the process of obtaining a program instruction or data item from memory.
- decoding refers to the process of translating the instruction into signals the computer can execute.
- Executing is the process of carrying out the commands.
- Storing, in this context, means writing the result to memory.

Machine Cycle



T- States

- The machine cycle and instruction cycle takes multiple clock periods.
- Single time period of frequency of microprocessor is called *t-state*.
- A portion of an operation carried out in one single clock period is called a t-state.
- A t-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.

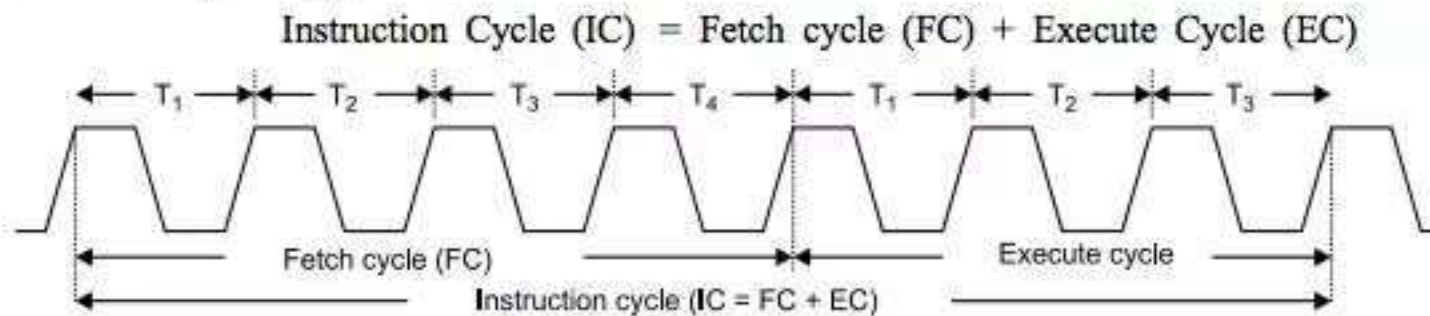
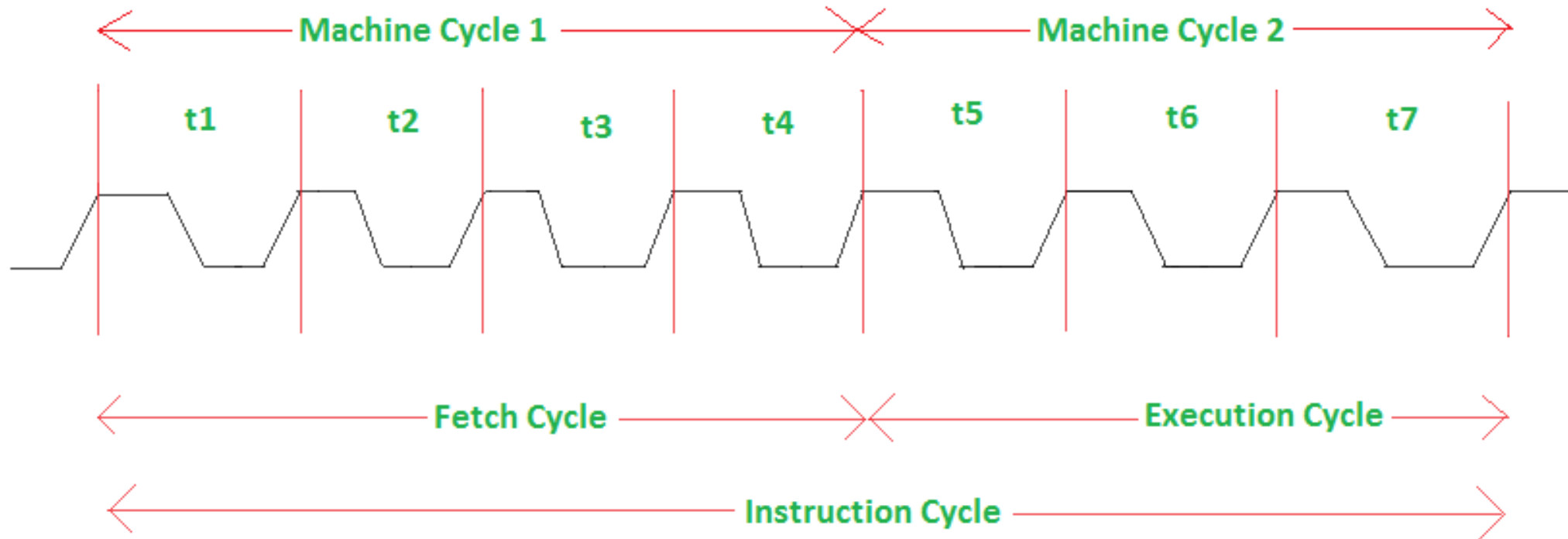


Fig. 5.2 (a) Processor cycle

8085 Instruction Cycle

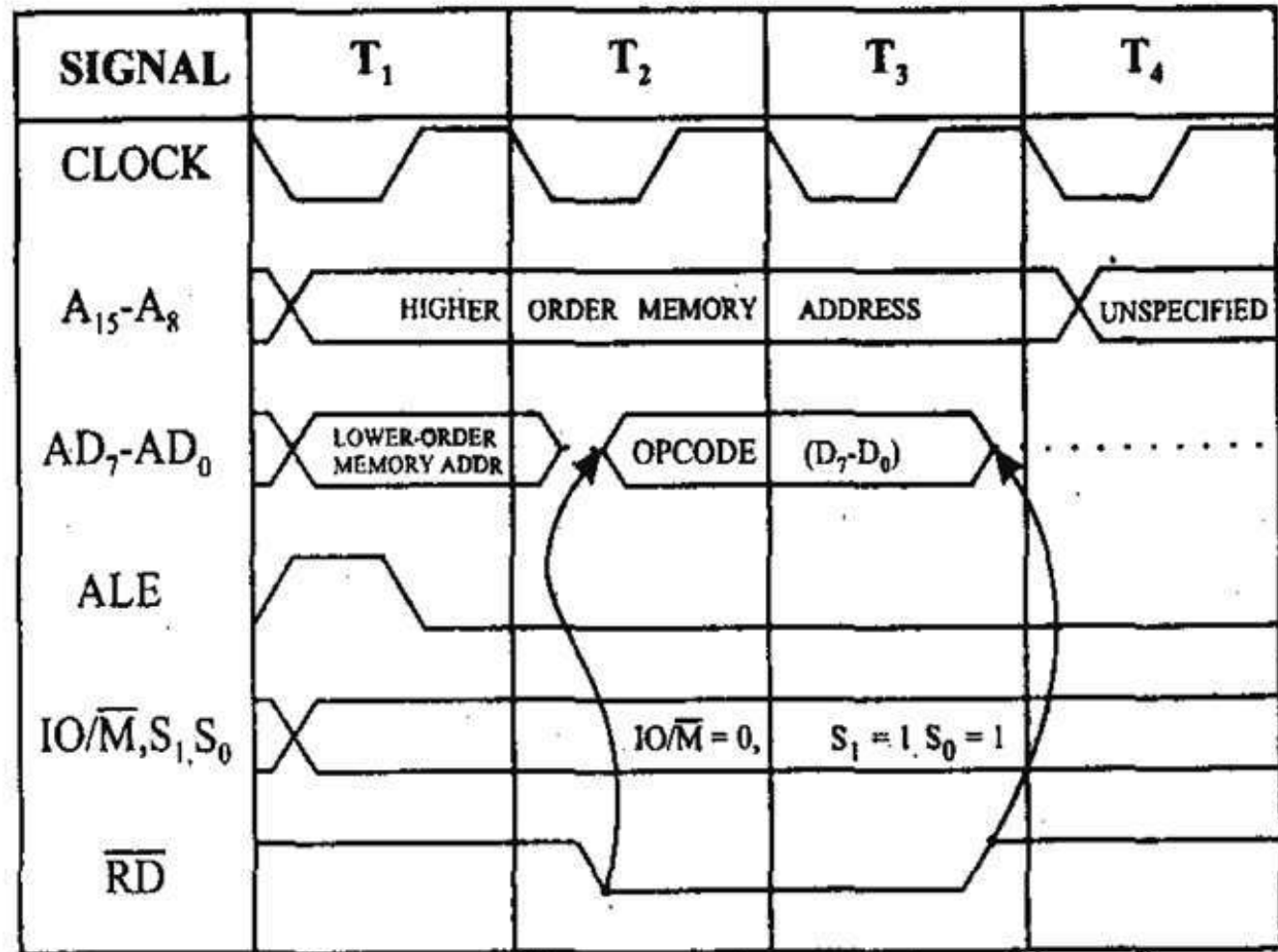


Fetch cycle takes four t-states and execution cycle takes three t-states.

Instruction cycle in 8085 microprocessor

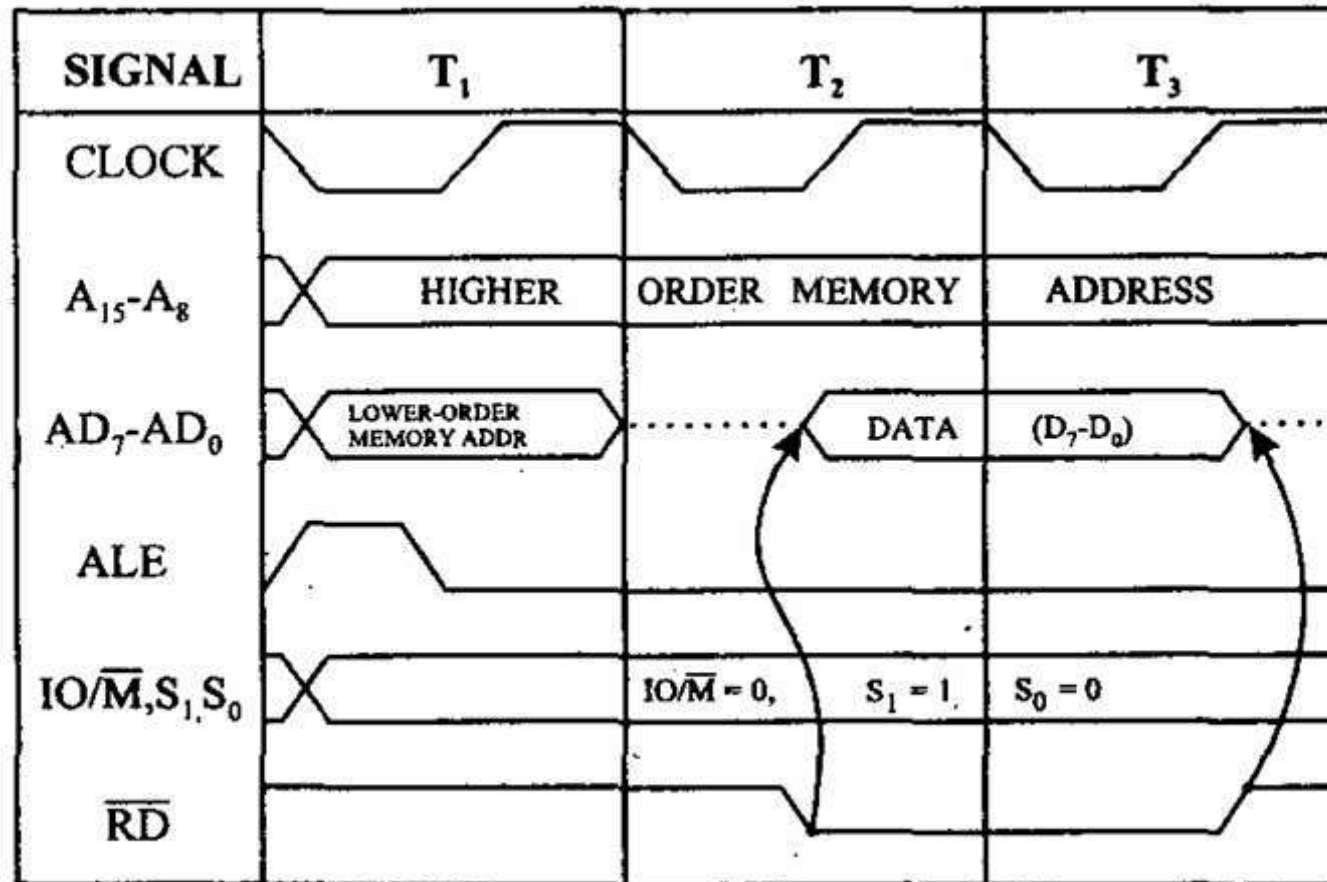
8085 Instruction Cycle

- Timing Diagram of opcode fetch as an example:



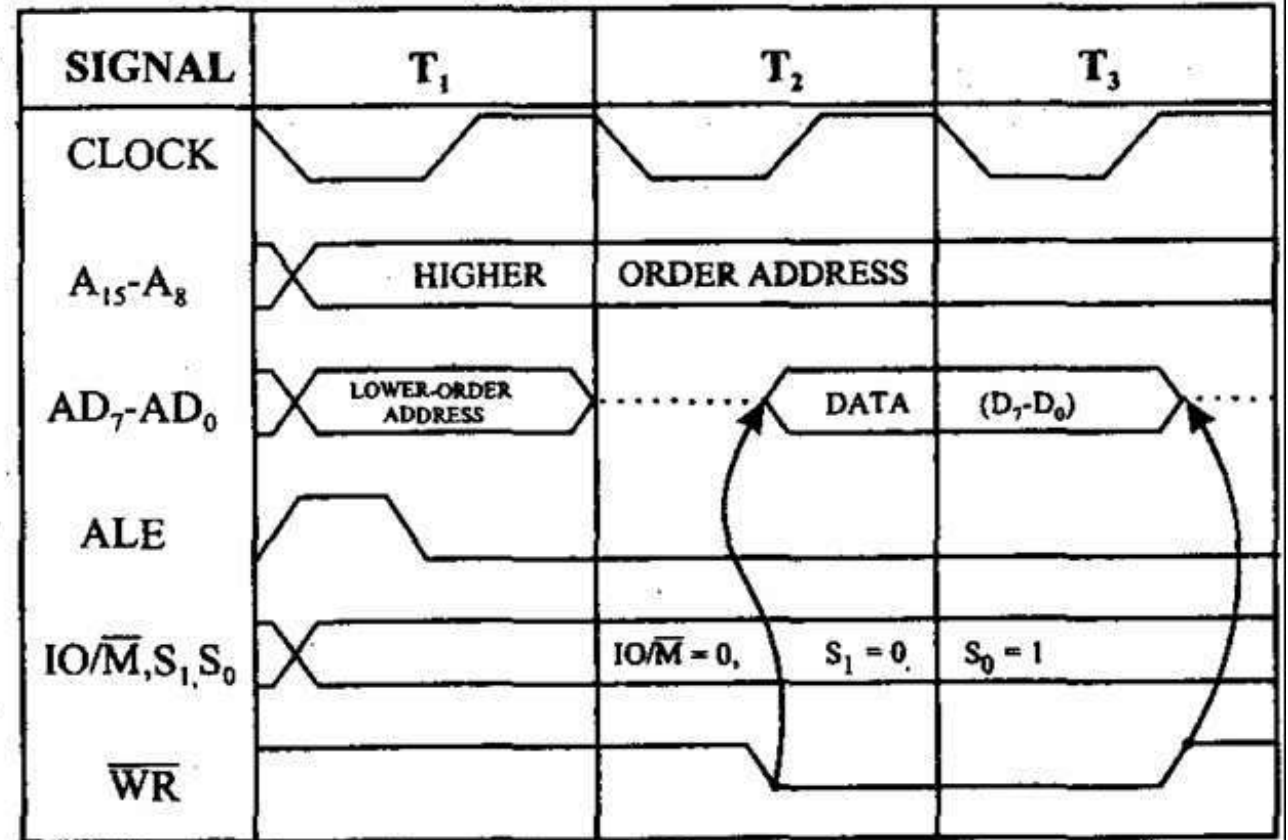
- Let us consider an instruction MOV A, B is in the memory location 0x5F7A. The opcode of MOV A, B is 0x78. This is a single byte instruction. Draw a timing diagram.

- Let's consider the instruction MVI A, 32 H stored at memory location 2000H.
- @ 2000 == MVI A opcode => 3EH
- @ 2001 == Data = > 32H
- 2 cycle: Opcode fetch cycle and memory read cycle.
- 8085 places the address 2001 on the address bus and increments PC to 2002H. ALE is asserted high IO/M =0, S1=1, S0=0 for memory read cycle. When RD =0, memory places the data byte 32H on the data bus.



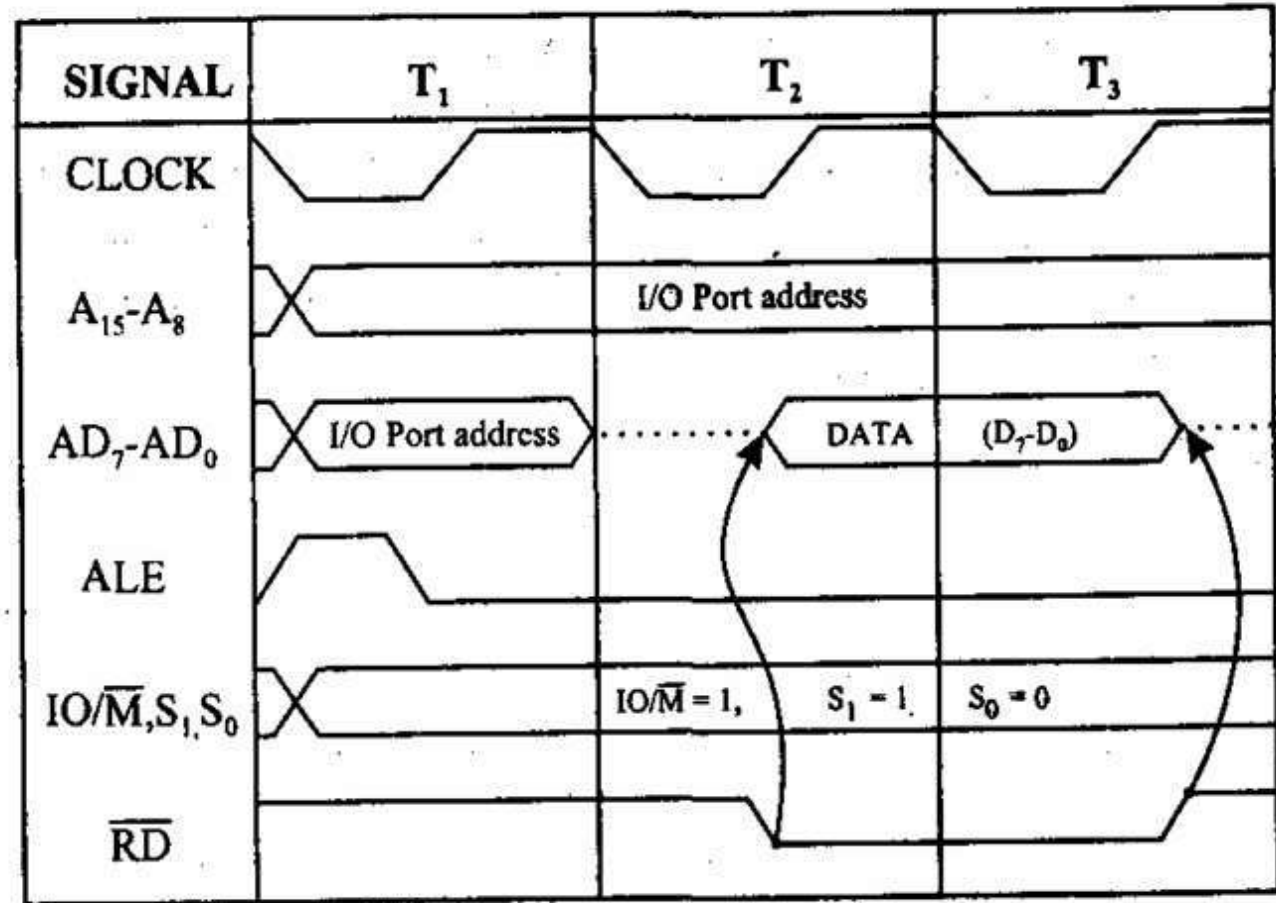
8085 Instruction Cycle

- Timing Diagram of memory write cycle



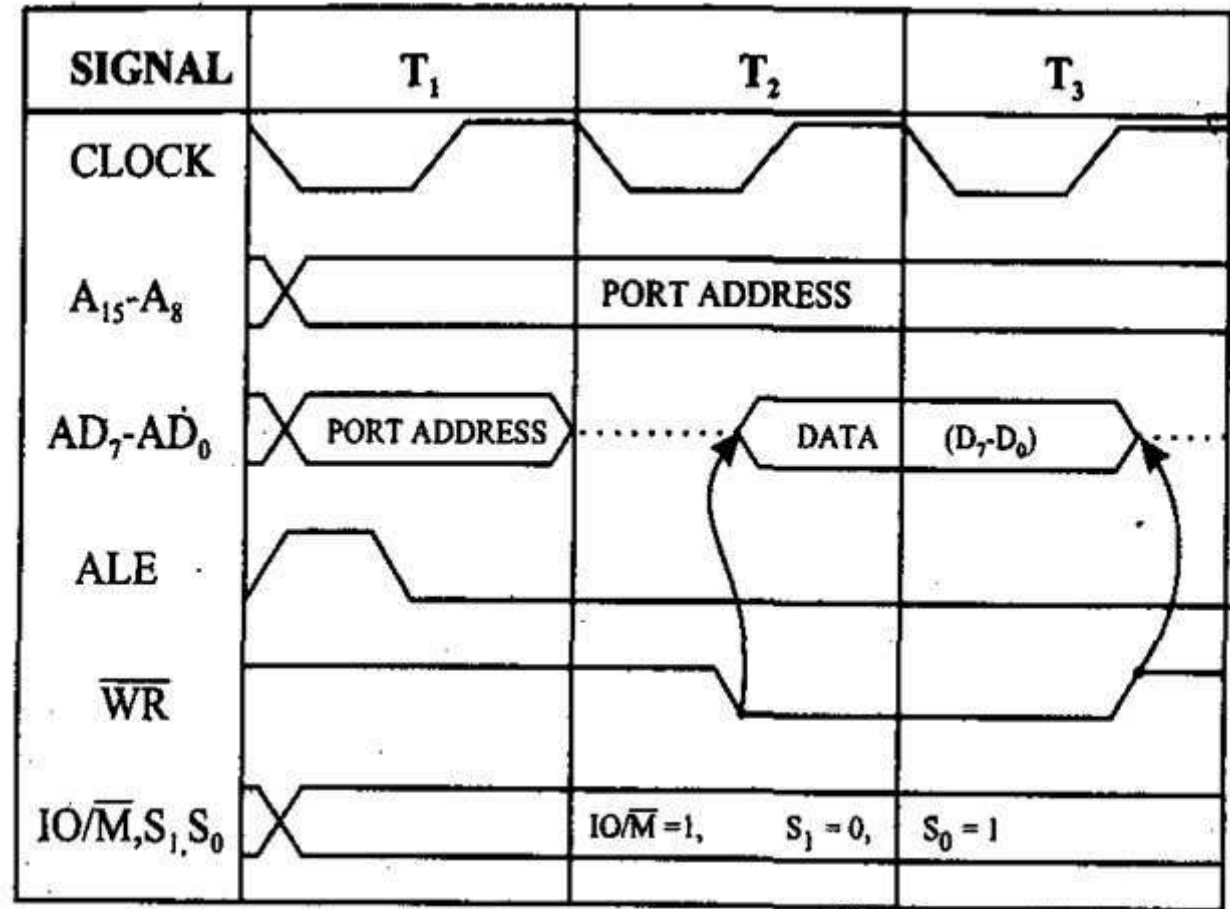
8085 Instruction Cycle

- Timing Diagram of IO read cycle
- IN instruction
- IN 40H



8085 Instruction Cycle

- Timing Diagram of IO write cycle
- OUT instruction
- OUT 40H

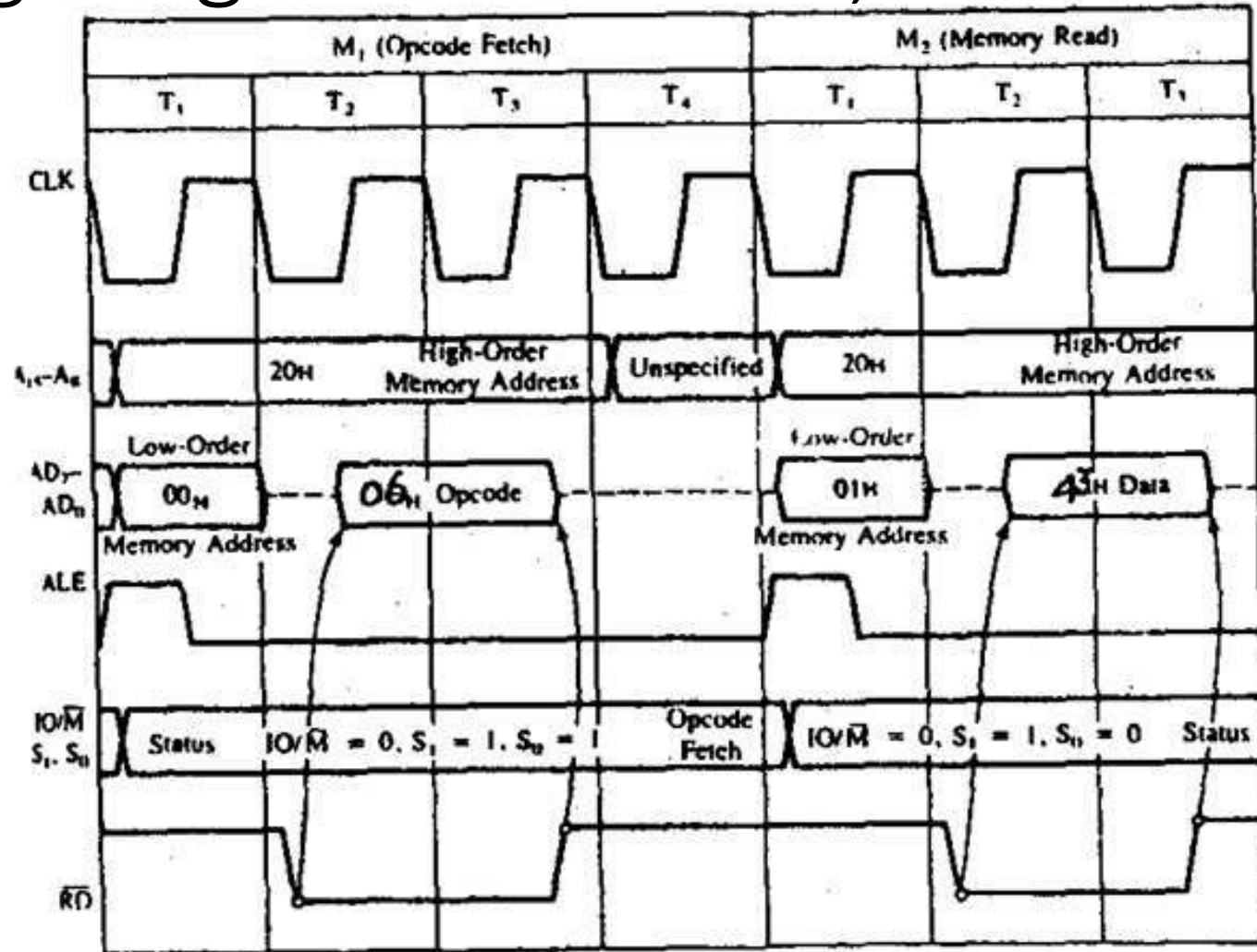


Timing Diagram for MVI B, 43H

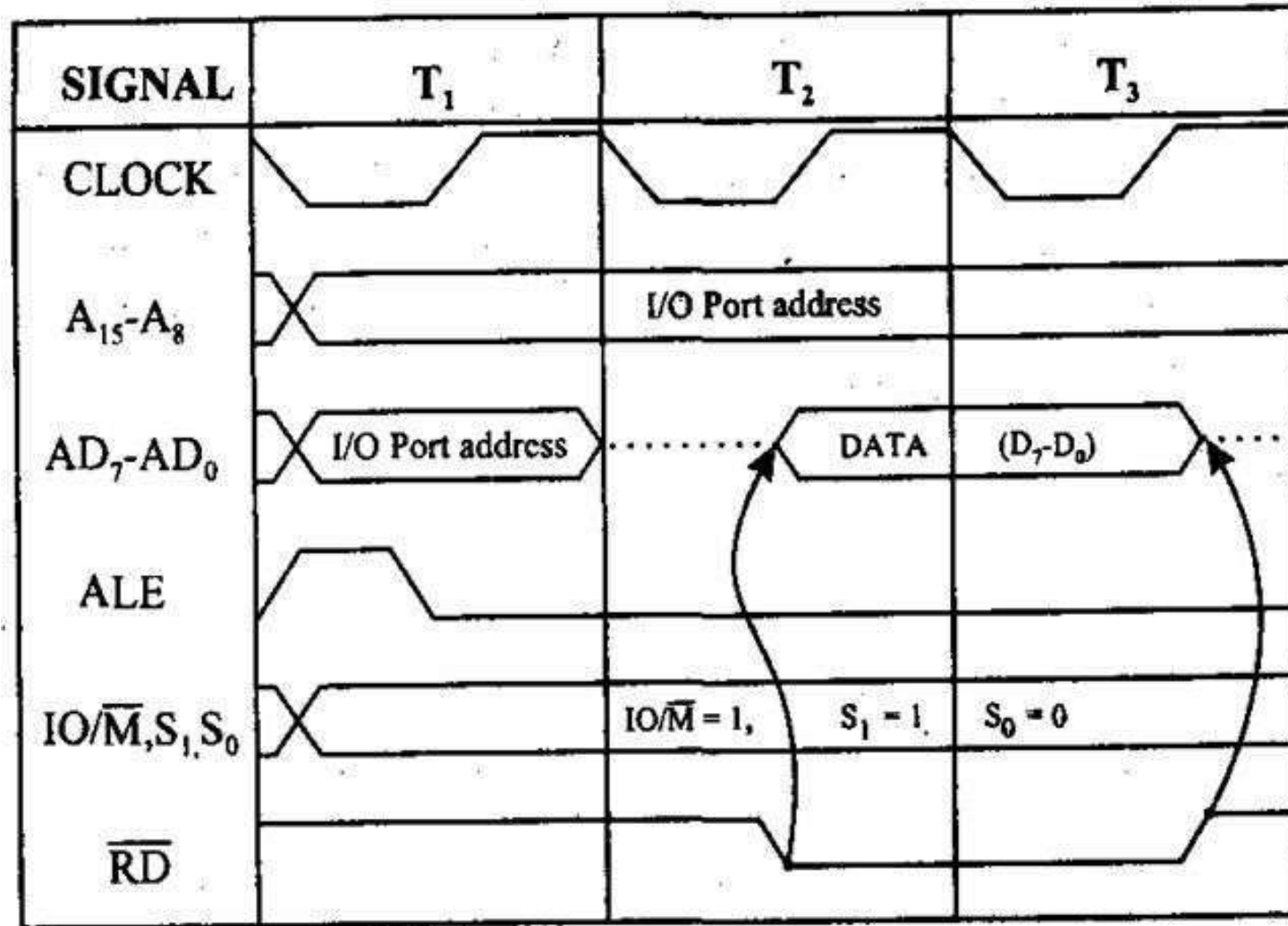
- Fetching the opcode 06H from the memory 2000H. (OF machine cycle)
- Read (move) the data) 43H from memory 2001H. (memory read)

Address	Mnemonics	Op code
2000	MVI B, 43H	06H
2001		43H

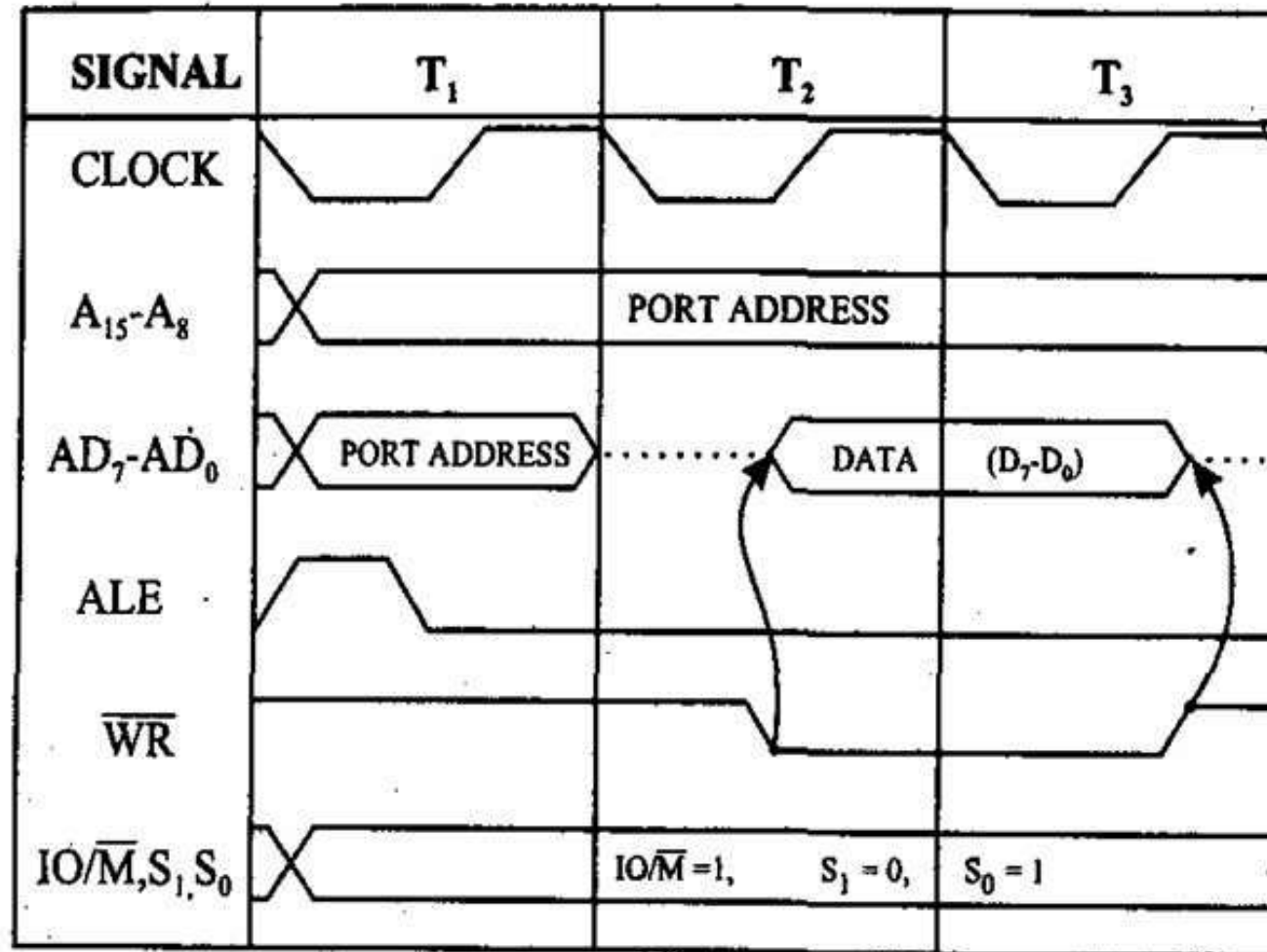
Timing Diagram for MVI B, 43H



IO Read Timing Diagram



IO Write Timing Diagram



Timing diagram for IN 40H

- Let us consider the instruction IN 40H is stored at memory location 0xCAFE. The port address 40H is stored sequentially in the memory location just after the instruction. The data to be read is “0x9F”. Draw the timing diagram.

Address	Op Codes	Mnemonic
0xCAFE	0xDB	IN 40H
0xCAFF	0x40	0x40 as port address

- Steps to perform
 - Opcode fetch cycle -> Memory read cycle -> port read cycle

Assignment

- Let us consider the instruction OUT CDH is stored at memory location 0xBABE. The port address CDH is stored sequentially in the memory location just after the instruction. The data to be written is “0xBE”. Draw the timing diagram.
- Due: September 28, 2020; 19:00 NST

Addressing modes in 8085

- Instructions used to transfer the data from one register to another register, from the memory to the register, and from the register to the memory without any alteration in the content.
- Addressing modes in 8085 is classified into 5 groups –
 - Immediate Addressing mode
 - Register Addressing mode
 - Direct Addressing mode
 - Register Indirect Addressing mode
 - Implied Addressing mode

Addressing modes in 8085

- Immediate addressing mode
 - In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand.
 - **For example:** MVI B, 20H: means 20H is copied into register B.
 - LXI H 3050 (load the H-L pair with the operand 3050H immediately)

Addressing modes in 8085

- Register addressing mode
 - In this mode, the data is copied from one register to another.
 - **For example:** MOV A, B: means data in register B is copied to register A.
 - ADD B (add contents of registers A and B and store the result in register A)
 - INR A (increment the contents of register A by one)

Addressing modes in 8085

- Direct addressing mode
 - In this mode, the data is directly copied from the given address to the register.
 - **For example:** LDB 5000: means the data at address 5000 is copied to register B.
 - LDA 2050 (load the contents of memory location into accumulator A)
 - LHLD address (load contents of 16-bit memory location into H-L register pair)
 - IN 35 (read the data from port whose address is 35)
 - OUT FB (write data from accumulator to the port address FB.)

Addressing modes in 8085

- Register Indirect addressing mode
 - In this mode, the data is transferred from one register to another by using the address pointed by the register.
 - **For example:** MOV A, M: means data is transferred from the memory address pointed by the register H-L pair to the register A.
 - LDAX B (move contents pointed by B-C register to the accumulator)
 - LXIH 9570 (load immediate the H-L pair with the address of the location 9570)

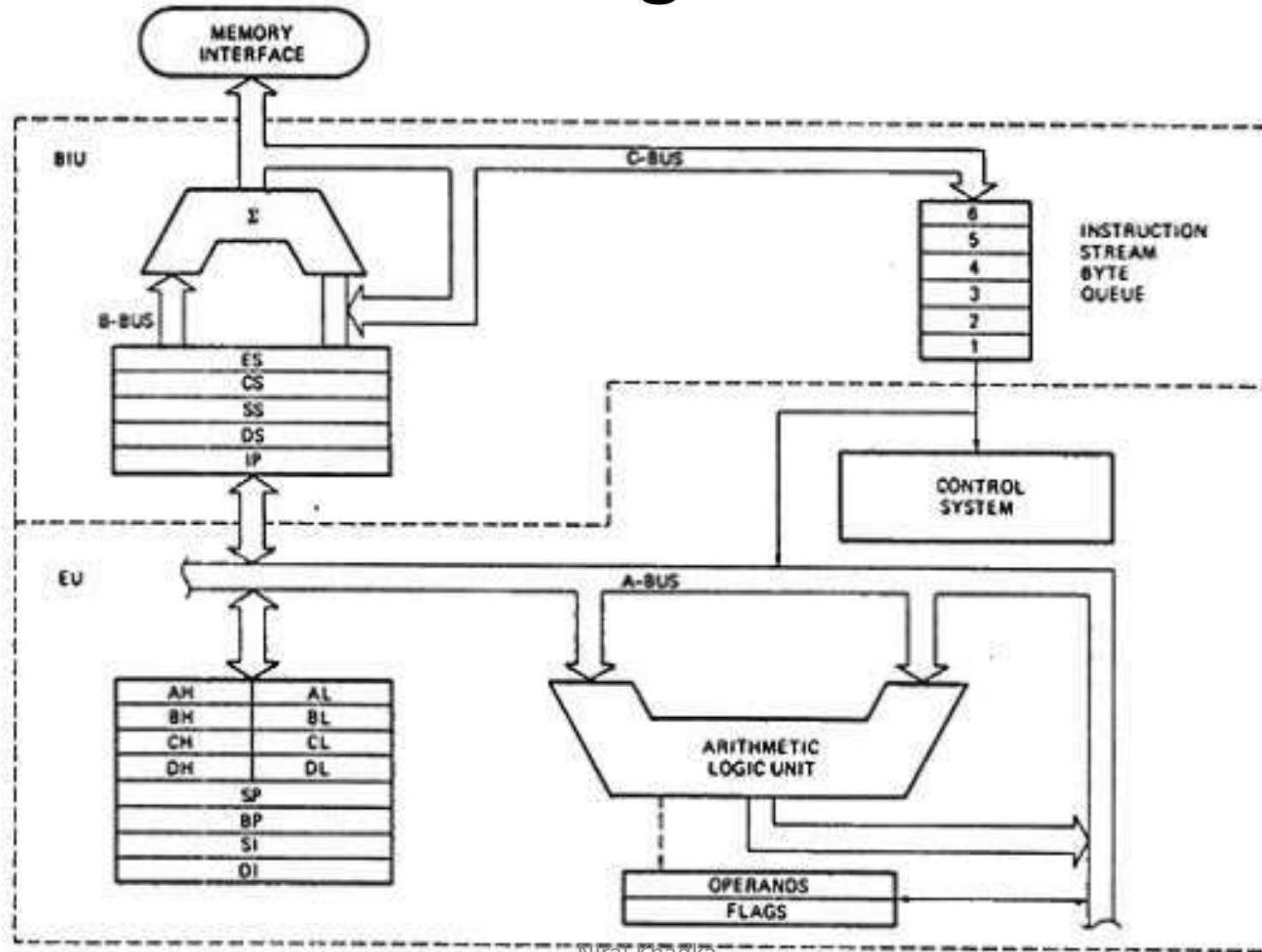
Addressing modes in 8085

- Implied addressing mode
 - This mode doesn't require any operand; the data is specified by the opcode itself.
 - **For example:** CMA.
 - CMA (finds and stores the 1's complement of the contents of accumulator A in A)
 - RRC (rotate accumulator A right by one bit)
 - RLC (rotate accumulator A left by one bit)

Introduction to 8086

- Features:
 - 16 bit microprocessor.
 - 16 bit data bus and a 20 bit address bus. Can address upto 1 MB of memory.
 - Can generate 16 bit I/O address which was only limited to 8 bit in 8085.
 - Has 14 registers each of 16 bit width.
 - Has multiplexed data and address bus to reduce pin count.
 - Is available with clock up to 10 MHz.
 - Has dedicated instructions for multiplication and division of integers.
 - Has a rich and powerful instruction set.

8086 internal block diagram



BIU (Bus Interface Unit)

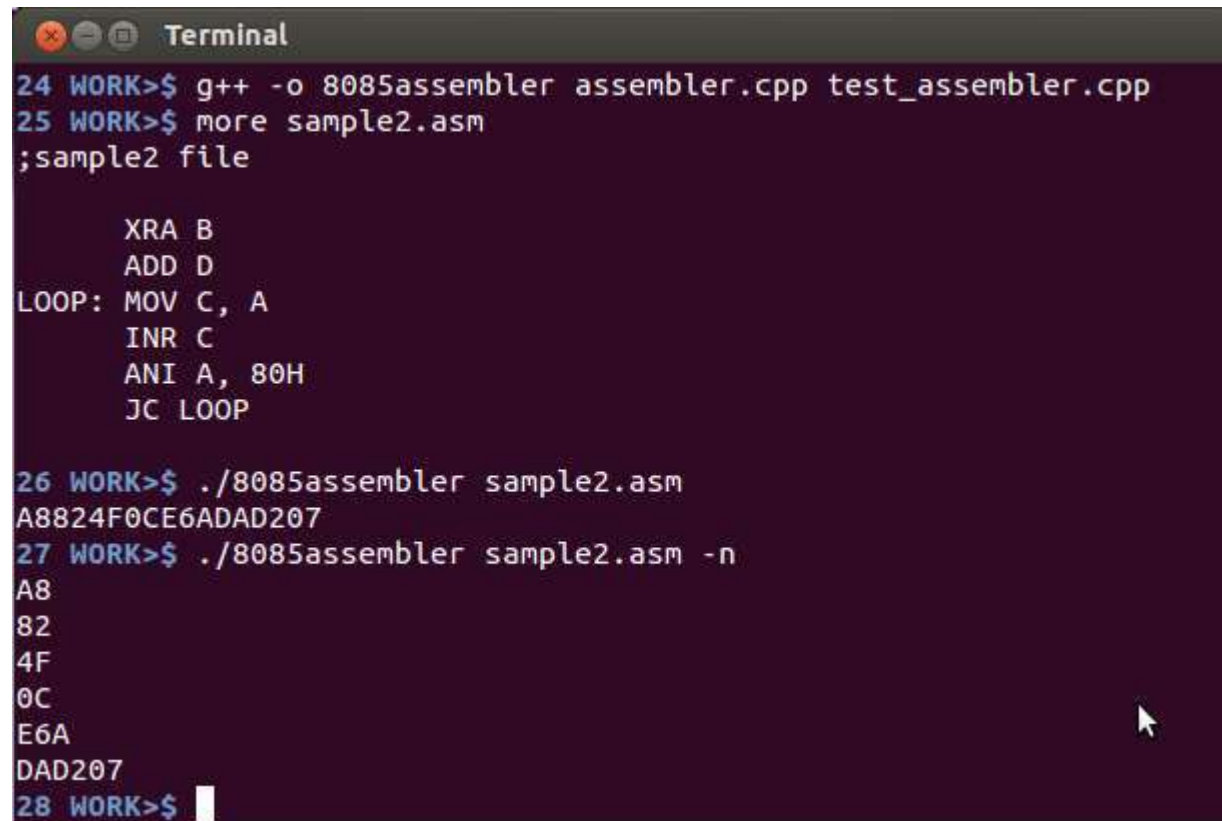
- The bus interface unit is the 8086 Internal Architecture to the outside world.
- It provides a full 16-bit bidirectional data bus and 20-bit address bus.
- The bus interface unit is responsible for performing all external bus operations, as listed below.
 - It sends address of the memory or I/O.
 - It fetches instruction from memory.
 - It reads data from port/memory.
 - It Writes data into port/memory.
 - It supports instruction queuing.
 - It provides the address relocation facility.

EU (Execution Unit)

- Execution unit gives instructions to BIU stating from where to fetch the data.
- EU decode and execute those instructions.
- Its function is to control operations on data using the instruction decoder & ALU.
- EU has no direct connection with system buses, it performs operations over data through BIU.

Unit 2

- INTRODUCTION TO ASSEMBLY PROGRAMMING LANGUAGE



```
Terminal
24 WORK>$ g++ -o 8085assembler assembler.cpp test_assembler.cpp
25 WORK>$ more sample2.asm
;sample2 file

        XRA B
        ADD D
LOOP:    MOV C, A
        INR C
        ANI A, 80H
        JC LOOP

26 WORK>$ ./8085assembler sample2.asm
A8824F0CE6ADAD207
27 WORK>$ ./8085assembler sample2.asm -n
A8
82
4F
0C
E6A
DAD207
28 WORK>$
```

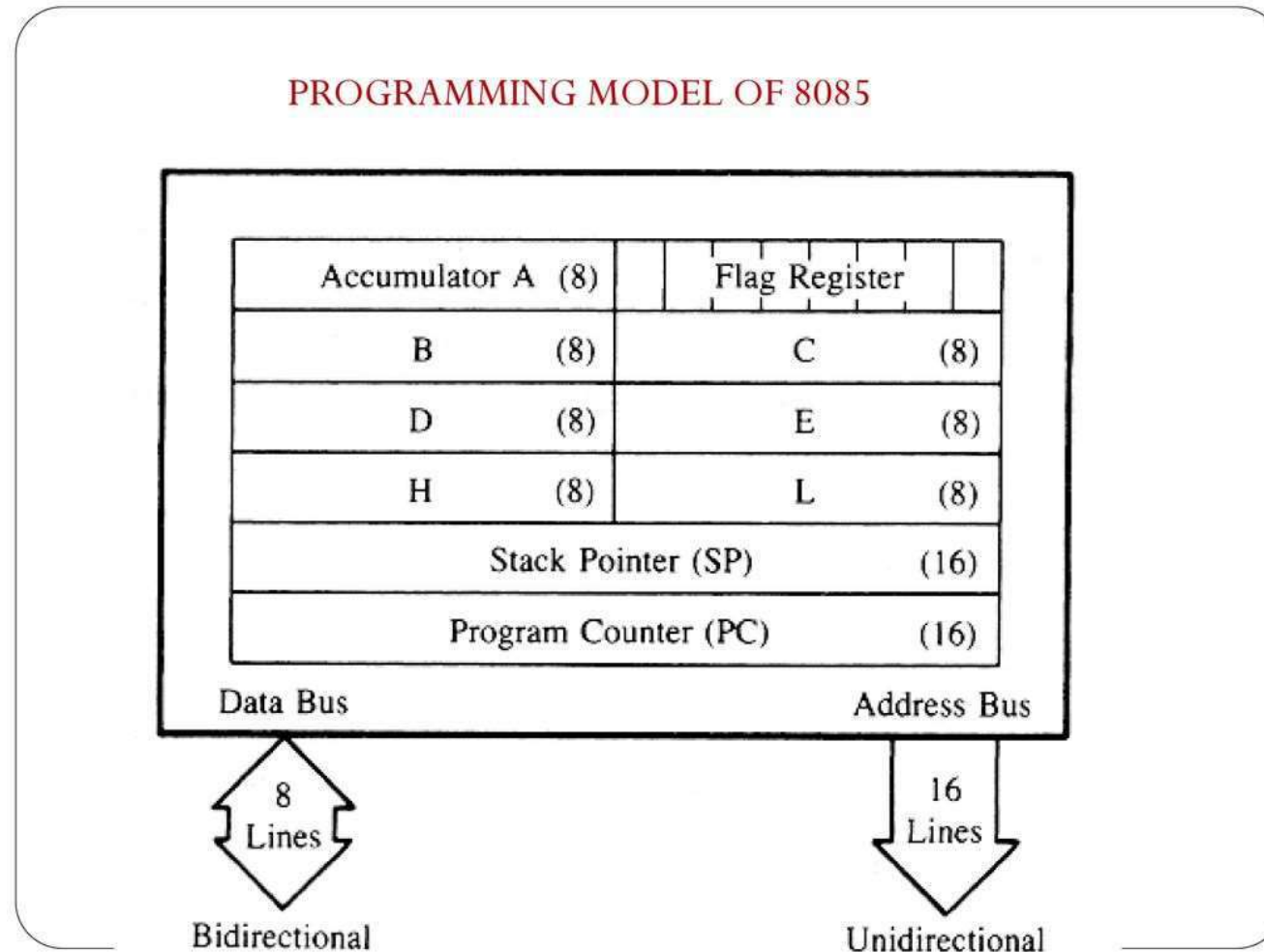
What is assembly programming language?

- Each microprocessor family has a set of instructions that handles various operations which are called machine language instruction which we often refer to as opcodes.
- A microprocessor only understands the machine language which are the chains of 1s and 0s and programming using them is too much problematic and complex.
- Assembly language is the low level programming language designed for specific processor that represents various machine instructions in symbolic code or more understandable form.

Advantages of Assembly language

- Requires less memory space and execution time.
- Allows direct control of hardware and resources.
- Suitable for time critical jobs.
- Suitable for writing ISR and memory resident programs.
- Understanding of how computer works.

8085 Programming Model



8085 Programming Model

- **Registers**

- The 8085 has six general purpose registers to store 8 bit data; these are identifies as B, C, D, E, H, L.
- They can be combined as register pairs - BC, DE and HL to perform some 16-bit operations.
- The programmer can use these registers to store or copy data into the registers by using data copy instructions.

8085 Programming Model

- **Accumulator**

- The accumulator is an 8-bit register that is a part of arithmetic/logic unit(ALU).
- This register is used to store 8-bit data and to perform arithmetic and logical operations.
- The result of an operation is stored in the accumulator.
- The accumulator is also identified as register A.

8085 Programming Model

- **Flags**

- Sign Flag (S): Sets or Resets based on the result stored in the accumulator. If the result stored is positive, the flag resets else if the result stored is negative the flag is set.
- Zero Flag (Z): Sets or Resets based on the result stored in the accumulator. If the result stored is zero the flag is set else it is reset.
- Auxiliary Carry Flag(AC) : This flag is set if there is a carry from low nibble(lowest 4 bits) to high nibble(upper 4 bits) or a borrow from high nibble to low nibble, in the low order 8-bit portion of an addition or subtraction operation.
- Parity Flag (P): This flag is set if there is even parity else it resets.
- Carry Flag (CY): This flag is set if there is a carry bit else it resets.

8085 Programming Model

- **Program Counter (PC)**

- This 16-bit register deals with sequencing the execution of instructions this register is a memory pointer.
- Memory locations have 16 bit addresses and that is why this is a 16 bit register.
- The function of the PC is to point to the memory address from which the next byte is to be fetched.
- When a byte(machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

8085 Programming Model

- **Stack Pointer (SP)**

- The stack pointer is also a 16-bit register used as a memory pointer.
- It points to a memory location in R/W memory called stack.
- The beginning of the stack is defined by loading 16-bit address in the stack pointer.

Instruction Word size

- Classified into 3 groups.
- Measured in bytes rather than word.
- Word means the width of data bus so the data bus width in 8085 is equal to a byte.
 - One Byte Instruction
 - Two Byte Instruction
 - Three Byte Instruction

Data Format in 8085

- Addresses
 - The address is a 16-bit unsigned integer ,number used to refer a memory location.
- Numbers and Logical Data
 - **Signed Integer** : A signed integer number is either a positive number or a negative number. In 8085, 8-bits are assigned for signed integer, in which most significant bit is used for sign and remaining seven bits are used for Sign bit 0 indicates positive number whereas sign bit 1 indicates negative number.
 - **Unsigned Integer** : The 8085 microprocessor supports 8-bit unsigned integer.
 - **BCD** : The term BCD number stands for binary coded decimal number. It uses ten digits from 0 through 9. The 8-bit register of 8085 can store two digit BCD
- Characters
 - uses ASCII code to represent characters. It is a 7-bit alphanumeric code that represents decimal numbers, English alphabets, and other special characters.

Classification of instructions in 8085

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. Stack Instructions
5. Branch Instructions
6. I/O Instructions
7. Interrupt Instructions.

Data Transfer Instructions

Opcode	Operand	Meaning	Explanation
MOV	Rd, Sc M, Sc Dt, M	Copy from the source (Sc) to the destination (Dt)	This instruction copies the contents of the source register into the destination register without any alteration. Example – MOV H, L
MVI	Rd, data M, data	Move immediate 8-bit	The 8-bit data is stored in the destination register or memory. Example – MVI B, 55H
LDA	16-bit address	Load the Accumulator	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the Accumulator. Example – LDA 2034H

Data Transfer Instructions

Opcode	Operand	Meaning	Explanation
LDAX	B/D Reg. pair	Load the Accumulator indirect	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the Accumulator. Example – LDAX B
LXI	Reg. pair, 16-bit data	Load the register pair immediate	The instruction loads 16-bit data in the register pair designated in the register or the memory. Example – LXI B, 3225H
LHLD	16-bit address	Load H and L registers direct	The instruction copies the contents of the memory location pointed out by the address into register L and copies the contents of the next memory location into register H. Example – LHLD 3225H

Data Transfer Instructions

Opcode	Operand	Meaning	Explanation
STA	16-bit address	16-bit address	<p>The contents of the Accumulator are copied into the memory location specified by the operand.</p> <p>This is a 3-Byte instruction, the second Byte specifies the low-order address and the third Byte specifies the high-order address.</p> <p>Example – STAAB00H</p>
STAX	B/D Reg. pair	Store the Accumulator indirect	<p>The contents of the Accumulator are copied into the memory location specified by the contents of the operand.</p> <p>Example – STAX B</p>
SHLD	16-bit address	Store H and L registers direct	<p>The contents of register L are stored in the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand.</p> <p>This is a 3-Byte instruction, the second Byte specifies the low-order address and the third Byte specifies the high-order address.</p> <p>Example – SHLD 3225H</p>

Data Transfer Instructions

Opcode	Operand	Meaning	Explanation
XCHG	None	Exchange H and L with D and E	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example – XCHG

Arithmetic Instructions

Opcode	Operand	Meaning	Explanation
ADD	R M	Add register or memory, to the Accumulator	The contents of the register or memory are added to the contents of the Accumulator and the result is stored in the Accumulator. Example – ADD B.
ADC	R M	Add register to the Accumulator with carry	The contents of the register or memory & M the Carry flag are added to the contents of the Accumulator and the result is stored in the Accumulator. Example – ADC B
ADI	8-bit data	Add the immediate to the Accumulator	The 8-bit data is added to the contents of the Accumulator and the result is stored in the Accumulator. Example – ADI 55H

Arithmetic Instructions

Opcode	Operand	Meaning	Explanation
ACI	8-bit data	Add the immediate to the Accumulator with carry	The 8-bit data and the Carry flag are added to the contents of the Accumulator and the result is stored in the Accumulator. Example –ACI 55H
DAD	Reg. pair	Add the register pair to H and L registers	The 16-bit data of the specified register pair are added to the contents of the HL register. Example – DAD B

Arithmetic Instructions

Opcode	Operand	Meaning	Explanation
SUB	R M	Subtract the register or the memory from the Accumulator	The contents of the register or the memory are subtracted from the contents of the Accumulator, and the result is stored in the Accumulator. Example – SUB B
SBB	R M	Subtract the source and borrow from the Accumulator	The contents of the register or the memory & M the Borrow flag are subtracted from the contents of the Accumulator and the result is placed in the Accumulator. Example – SBB B
SUI	8-bit data	Subtract the immediate from the Accumulator	The 8-bit data is subtracted from the contents of the Accumulator & the result is stored in the Accumulator. Example –SUI 55H

Arithmetic Instructions

Opcode	Operand	Meaning	Explanation
SBI	8-bit data	Subtract the immediate from the Accumulator with borrow	The 8-bit data is subtracted to the contents of the Accumulator and the result is stored in the Accumulator. Example –SBI 55H
INR	R M	Increment the register or the memory by 1	The contents of the designated register or the memory are incremented by 1 and their result is stored at the same place. Example – INR B
INX	R	Increment register pair by 1	The contents of the designated register pair are incremented by 1 and their result is stored at the same place. Example – INX B

Arithmetic Instructions

Opcode	Operand	Meaning	Explanation
DCR	R M	Decrement the register or the memory by 1	The contents of the designated register or memory are decremented by 1 and their result is stored at the same place. Example – DCR B
DCX	R	Decrement the register pair by 1	The contents of the designated register pair are decremented by 1 and their result is stored at the same place. Example – DCX B
DAA	None	Decimal adjust Accumulator	The contents of the Accumulator are changed from a binary value to two 4-bit BCD digits. If the value of the low-order 4-bits in the Accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the Accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. Example – DAA

Logical Instructions

Opcode	Operand	Meaning	Explanation
CMP	R M	Compare the register or memory with the Accumulator	The contents of the operand (register or memory) are M compared with the contents of the Accumulator. If A less than (R/M), the CY flag is set and Zero flag is reset. If A equals to (R/M), the Zero flag is set and CY flag is reset. If A greater than (R/M), the CY and Zero flag are reset.
CPI	8-bit data	Compare immediate with the Accumulator	The second Byte data is compared with the contents of the Accumulator.
ANA	R M	Logical AND register or memory with the Accumulator	The contents of the Accumulator are logically AND with M the contents of the register or memory, and the result is placed in the Accumulator.

Logical Instructions

Opcode	Operand	Meaning	Explanation
ANI	8-bit data	Logical AND immediate with the Accumulator	The contents of the Accumulator are logically AND with the 8-bit data and the result is placed in the Accumulator.
XRA	R M	Exclusive OR register or memory with the Accumulator	The contents of the Accumulator are Exclusive OR with M the contents of the register or memory, and the result is placed in the Accumulator.
XRI	8-bit data	Exclusive OR immediate with the Accumulator	The contents of the Accumulator are Exclusive OR with the 8-bit data and the result is placed in the Accumulator.

Logical Instructions

Opcode	Operand	Meaning	Explanation
ORA	R M	Logical OR register or memory with the Accumulator	The contents of the Accumulator are logically OR with M the contents of the register or memory, and result is placed in the Accumulator.
ORI	8-bit data	Logical OR immediate with the Accumulator	The contents of the Accumulator are logically OR with the 8-bit data and the result is placed in the Accumulator.
RLC	None	Rotate the Accumulator left	Each binary bit of the Accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7.

Logical Instructions

Opcode	Operand	Meaning	Explanation
RRC	None	Rotate the Accumulator right	Each binary bit of the Accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0.
RAL	None	Rotate the Accumulator left through carry	Each binary bit of the Accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7.
RAR	None	Rotate the Accumulator right through carry	Each binary bit of the Accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0.

Logical Instructions

Opcode	Operand	Meaning	Explanation
CMA	None	Complement Accumulator	The contents of the Accumulator are complemented. No flags are affected.
CMC	None	Complement carry	The Carry flag is complemented. No other flags are affected.
STC	None	Set Carry	Set Carry

Stack Instructions

Opcode	Operand	Meaning	Explanation
SPHL	None	Copy H and L registers to the stack pointer	<p>The instruction loads the contents of the H and L registers into the stack pointer register.</p> <p>The contents of the H register provide the high-order address and the contents of the L register provide the low-order address.</p> <p>Example – SPHL</p>
XTHL	None	Exchange H and L with top of stack	<p>The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register.</p> <p>The contents of the H register are exchanged with the next stack location (SP+1).</p> <p>Example – XTHL</p>

Stack Instructions

Opcode	Operand	Meaning	Explanation
PUSH	Reg. pair	Push the register pair onto the stack	<p>The contents of the register pair designated in the operand are copied onto the stack in the following sequence.</p> <p>The stack pointer register is decremented and the contents of the high order register (B, D, H, A) are copied into that location.</p> <p>The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location.</p> <p>Example – PUSH B</p>
POP	Reg. pair	Pop off stack to the register pair	<p>The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand.</p> <p>The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand.</p> <p>The stack pointer register is again incremented by 1.</p> <p>Example – POP B</p>

Branch Instructions

Opcode	Operand	Meaning	Explanation
JMP	16-bit address	Jump unconditionally	The program sequence is transferred to the memory address given in the operand.

Branch Instructions

Opcode			Operand	Meaning	Explanation
Opcode	Description	Flag Status	16-bit address	Jump conditionally	The program sequence is transferred to the memory address given in the operand based on the specified flag of the PSW.
JC	Jump on Carry	CY=1			
JNC	Jump on no Carry	CY=0			
JP	Jump on positive	S=0			
JM	Jump on minus	S=1			
JZ	Jump on zero	Z=1			
JNZ	Jump on no zero	Z=0			
JPE	Jump on parity even	P=1			
JPO	Jump on parity odd	P=0			

Branch Instructions

Opcode			Operand	Meaning	Explanation
Opcode	Description	Flag Status	16-bit address	Unconditional subroutine call	The program sequence is transferred to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack.
CC	Call on Carry	CY=1			
CNC	Call on no Carry	CY=0			
CP	Call on positive	S=0			
CM	Call on minus	S=1			
CZ	Call on zero	Z=1			
CNZ	Call on no zero	Z=0			
CPE	Call on parity even	P=1			
CPO	Call on parity odd	P=0			

Branch Instructions

Opcode	Operand	Meaning	Explanation
RET	None	Return from subroutine unconditionally	The program sequence is transferred from the subroutine to the calling program.

Branch Instructions

Opcode			Operand	Meaning	Explanation
Opcode	Description	Flag Status	None	Return from subroutine conditionally	The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW and the program execution begins at the new address.
RC	Return on Carry	CY=1			
RNC	Return on no Carry	CY=0			
RP	Return on positive	S=0			
RM	Return on minus	S=1			
RZ	Return on zero	Z=1			
RNZ	Return on no zero	Z=0			
RPE	Return on parity even	P=1			
RPO	Return on parity odd	P=0			
			Niraj Khadka		

I/O Instructions

Opcode	Operand	Meaning	Explanation
OUT	8-bit port address	Output the data from the Accumulator to a port with 8 bit address	The contents of the Accumulator are copied into the I/O port specified by the operand. Example – OUT 01H
IN	8-bit port address	Input data to Accumulator from a port with 8-bit address	The contents of the input port designated in the operand are read and loaded into the Accumulator. Example – IN 04H

Interrupt Instructions

Opcode	Operand	Meaning	Explanation	
RST	0-7	Restart	The RST instruction is used as software instructions in a program to transfer the program execution to one of the following eight locations.	
			Instruction	Restart Address
			RST 0	0000H
			RST 1	0008H
			RST 2	0010H
			RST 3	0018H
			RST 4	0020H
			RST 5	0028H
			RST 6	0030H
			RST 7	0038H

Interrupt Instructions

Opcode	Operand	Meaning	Explanation	
RST	0-7	Restart	The 8085 has additionally 4 interrupts, which can generate RST instructions internally and doesn't require any external hardware. Following are those instructions and their Restart addresses	
			Interrupt	Restart Address
			TRAP	0024H
			RST 5.5	002CH
			RST 6.5	0034H
			RST 7.5	003CH

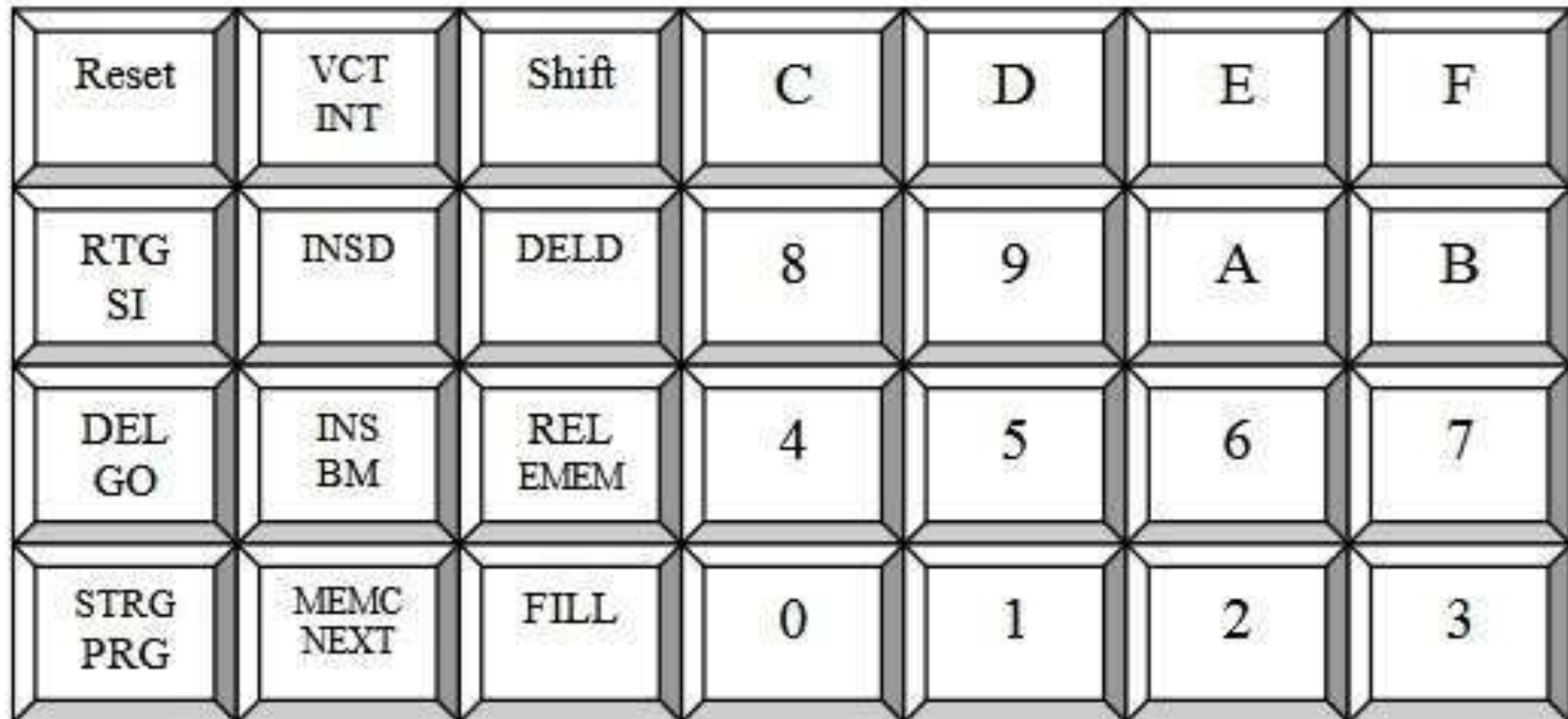
Assembling a program

- When we finish writing a assembly code program, the program is converted to binary machine code which is loaded into the ROM mapped at location 0x0000.
- On RESET, the processor starts executing the machine level code from the ROM, which is our program.
- Assembling refers to converting of the assembly mnemonics and operands to machine level codes.
- Generally when using a trainer board we do assembling by hand that is we convert each instruction manually to machine code referring to the ISA of 8085.

Assembling a program

Address	HEX Codes	Labels	Mnemonics	Comments
8000	21, 50, 80	START	LXI H, 8050H	Set up HL as a pointer for source memory
8003	11, 70, 80		LXI D, 8070H	Set DE for destination address
8006	06, 10		MVI B,10H	Set up B to count 16 bytes
8008	7E	NEXT	MOV A,M	Get data byte from source memory
8009	12		STAX D	Store data byte at destination
			Niraj Khadka	

HEX Keypad of 8085 trainer kit



The diagram shows a 4x7 grid of keys. Each key is a rectangle with a 3D effect, featuring a top face and two side faces. The keys are arranged in four rows and seven columns. The first three columns contain function keys, and the last four columns contain hexadecimal digits and letters. The keys are labeled as follows:

Reset	VCT INT	Shift	C	D	E	F
RTG SI	INSD	DELD	8	9	A	B
DEL GO	INS BM	REL EMEM	4	5	6	7
STRG PRG	MEMC NEXT	FILL	0	1	2	3

Executing a program

- The machine level code is programmed into a ROM. Since the ROM is usually mapped into 0x0000 location, the code is loaded from there to the IR and then execution starts.
- The execution process depends heavily on the microprocessor architecture. Microprocessor where segmentation is used has a different reset vector and starts execution with relocation option for programs.

Debugging assembly language programs

- Debugging is the process of identifying and removing bug from software or program.
- It refers to identification of errors in the program logic, machine codes, and execution.
- It gives step by step information about the execution of code to identify the fault in the program.

Debugging assembly language programs

- Debugging can help in determining:
 - Values of register.
 - Flow of program.
 - Entry and exit point of a function.
 - Entry into *if* or *else* statement.
 - Looping of code.
 - Calculation check.

Debugging assembly language programs

- **Common sources of error:**

- Selecting a wrong code
- Forgetting second or third byte of instruction
- Specifying wrong jump locations
- Not reversing the order of high and low bytes in a Jump instruction
- Writing memory addresses in decimal instead of hexadecimal
- Failure to clear accumulator when adding two numbers
- Failure to clear carry registers
- Failure to set flag before Jump instruction
- Specifying wrong memory address on Jump instruction
- Use of improper combination of rotate instructions

Debugging assembly language programs

- The debugging process is divided into two parts:
 - **Static Debugging:** It is similar to visual inspection of circuit board, it is done by a paper and pencil to check the flowchart and machine codes. It is used to the understanding of code logic and structure of program.
 - **Dynamic Debugging:** It involves observing the contents of register or output after execution of each instruction (in single step technique) or a group of instructions (in breakpoint technique).

Sample Programs

- 1. The memory location 2050H holds the data byte F7H. Write instructions to transfer the data byte to accumulator using different op-codes: MOV, LDAX and LDA.**

Sample Programs

1. The memory location 2050H holds the data byte F7H. Write instructions to transfer the data byte to accumulator using different op-codes: MOV, LDAX and LDA.

- Memory Location : 2050H = F7H
- Final condition A = F7H
- LDA = Load direct to accumulator.
- LDA syntax = LDA 16bit_memory_address

Sample Programs

- 1. The memory location 2050H holds the data byte F7H. Write instructions to transfer the data byte to accumulator using different op-codes: MOV, LDAX and LDA.**

LDA 2050H

Sample Programs

2. Register B contains 32H, Use MOV and STAX to copy the contents of register B in memory location 8000H.

- STAX = store to accumulator indirect.
- Syntax STAX B/D register pair
- First lets load D/E register pair with 8000H.
 - MVI E, 00H
 - MVI D, 80H
- Now lets copy Content of B to A.
 - MOV A, B

Sample Programs

2. Register B contains 32H, Use MOV and STAX to copy the contents of register B in memory location 8000H.

- Now lets store the content of A to the memory location pointed by D/E register pair.
- STAX D

Sample Programs

3. Write a program to load memory locations 7090 H and 7080 H with data 40H and 50H and then swap these data.

```
MVI A, 50H  
STA 7080H  
MVI A, 40H  
STA 7090H
```

```
MOV B, A  
LDA 7080H  
STA 7090H  
MOV A, B  
STA 7080H
```

Sample Programs

- **4. Pair B contains 1122H and pair D contains 3344H. WAP to exchange the contents of B and D pair using XCHG instruction.**

LXI B, 1122H

B=11, C=22

LXI D, 3344H

D=33, E=44

MOV H, B

MOV L, C

XCHG

(Exchange DE pair with HL pair)

MOV B, H

MOV C, L

HLT

Sample Programs

- **5. Register BC contain 2793H and register DE contain 3182H. Write instruction to add these two 16 bit numbers and place the sum in memory locations 2050H and 2051H.**

MOV A, C	93H:	1 0 0 1 0 0 1 1	
ADD E	<u>+82H:</u>	<u>1 0 0 0 0 0 1 0</u>	
MOV L, A	L=15	15H	(1) 0 0 1 0 1 0 1 15H
MOV A, B	27H:	0 0 1 0 0 1 1 1	
ADC D	<u>+31H:</u>	<u>0 0 1 1 0 0 0 1</u>	
MOV H, A	H=59H	0 1 0 1 1 0 0 1	59H
SHLD 2050H	; [2050]← 15H , [2051]← 59H		

Note: SHLD stores the contents of L in specified location and contents of H in next higher location.

Sample Programs

6. WAP to sort in ascending order for 10 bytes from 1120H.

```
START: LXI H, 1120H          L1:   DCR C
      MVI D, 00H             JNZ L2
      MVI C, 09H             MOV  A, D
L2:   MOV A, M               RRC
      INX H                  JC START
      CMP M                  HLT
      JC L1 ; if A<M
      MOV B, M
      MOV M, A
      DCX H
      MOV M, B
      INX H
      MVI D, 01H
```

Timer and Delay

- **Counter:**

- It is designed simply by loading an appropriate number into one of the registers and using the INR or DCR instructions.
- A loop is established to update a count, and each count is checked to determine whether it has reached the final number, if not the loop is repeated.

Timer and Delay

- **Time Delay:**

- When we use loop by counter, the loop causes the delay.
- Depending upon the clock period of the system, the time delay occurred during looping.
- The instructions within the loop use their own T-states so they need certain time to execute resulting delay.

Time Delay

- Suppose we have an 8085 micro processor with 2MHz clock frequency.
- Let us use the instruction MVI which takes 7 T-states.
- Clock frequency of system (f) = 2 mhz
- Clock period (T) = $1/f = \frac{1}{2} * 10^{-6} = 0.5 \text{Microsecond}$.
- Time to execute MVI = 7 T-states * 0.5 = 3.5Micro seconds

Time Delay

- Eg.

	MVI C, FFH	7
LOOP:	DCR C	4
	JNZ LOOP	10

- Here register C is loaded with count FFH (255) by using MVI which takes 7 T-states.
- Next 2 instructions DCR and JNZ form a loop with a total of 14 (4+10) T-states.
- The loop is repeated 255 times until C=0.

Time Delay

- Eg.

	MVI C, FFH	7
LOOP:	DCR C	4
	JNZ LOOP	10

- $Tl = (T * \text{loop T-states} * \text{count})$

Where, Tl = time delay in loop

T = system clock period

Count = decimal value for counter

$$Tl = 0.5 * 10^{-6} * 14 * 255 = 1785 \text{ ms}$$

Time Delay

- Eg.

	MVI C, FFH	7
LOOP:	DCR C	4
	JNZ LOOP	10

- But JNZ takes only 7 T-states when exited from loop i. e. last count = 0.50 adjusted loop delay
- $T_{la} = T_l - (3 \text{ T-states} - \text{clock period})$
= 1785 ms – 1.5 ms
= 1783.5 ms

Time Delay

- Eg.

	MVI C, FFH	7
LOOP:	DCR C	4
	JNZ LOOP	10

- Total delay loop of program TD= Time to execute outside code + TLA inside Loop

$$= 7 * 0.5\text{MS} + 1783.5 \text{ micros}$$

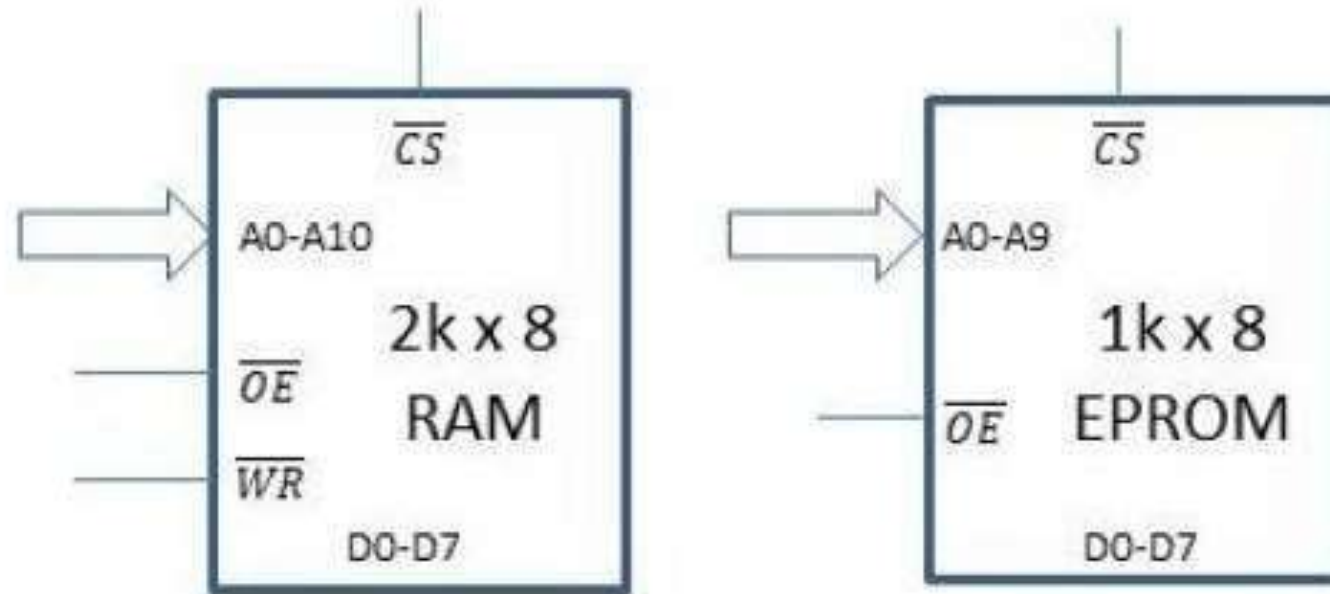
$$= 1787 \text{ micros}$$

$$\sim 1.8 \text{ ms}$$

Memory Interfacing Concepts

- Interface a 1kB EPROM and a 2 kB RAM with microprocessor 8085. The address allotted to 1 kB EPROM should be 2000H to 22FFH. You can assign the address range of your choice to the 2 kB RAM.

Pin Diagram of MEMORY CHIP



EEPROM CS' Generation

- Address Range given for EEPROM is from 2000H to 22FFH.
- In binary:

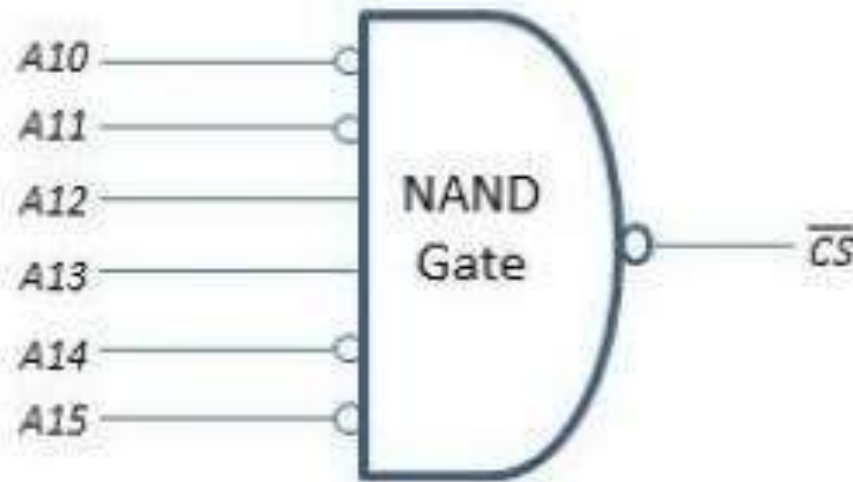
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	1	1	1	1	1	1	1	1

EEPROM CS' Generation

- So, for the Chip to be activated we find the unique situation is:
 - $A_{15} = 0$
 - $A_{14} = 0$
 - $A_{13} = 1$
 - $A_{12} = 0$
 - $A_{11} = 0$
 - $A_{10} = 0$
- $CS' = (A_{15}' * A_{14}' * A_{13} * A_{12}' * A_{11}' * A_{10}')'$

EEPROM CS' Generation

- $CS' = (A15' * A14' * A13 * A12' * A11' * A10')'$



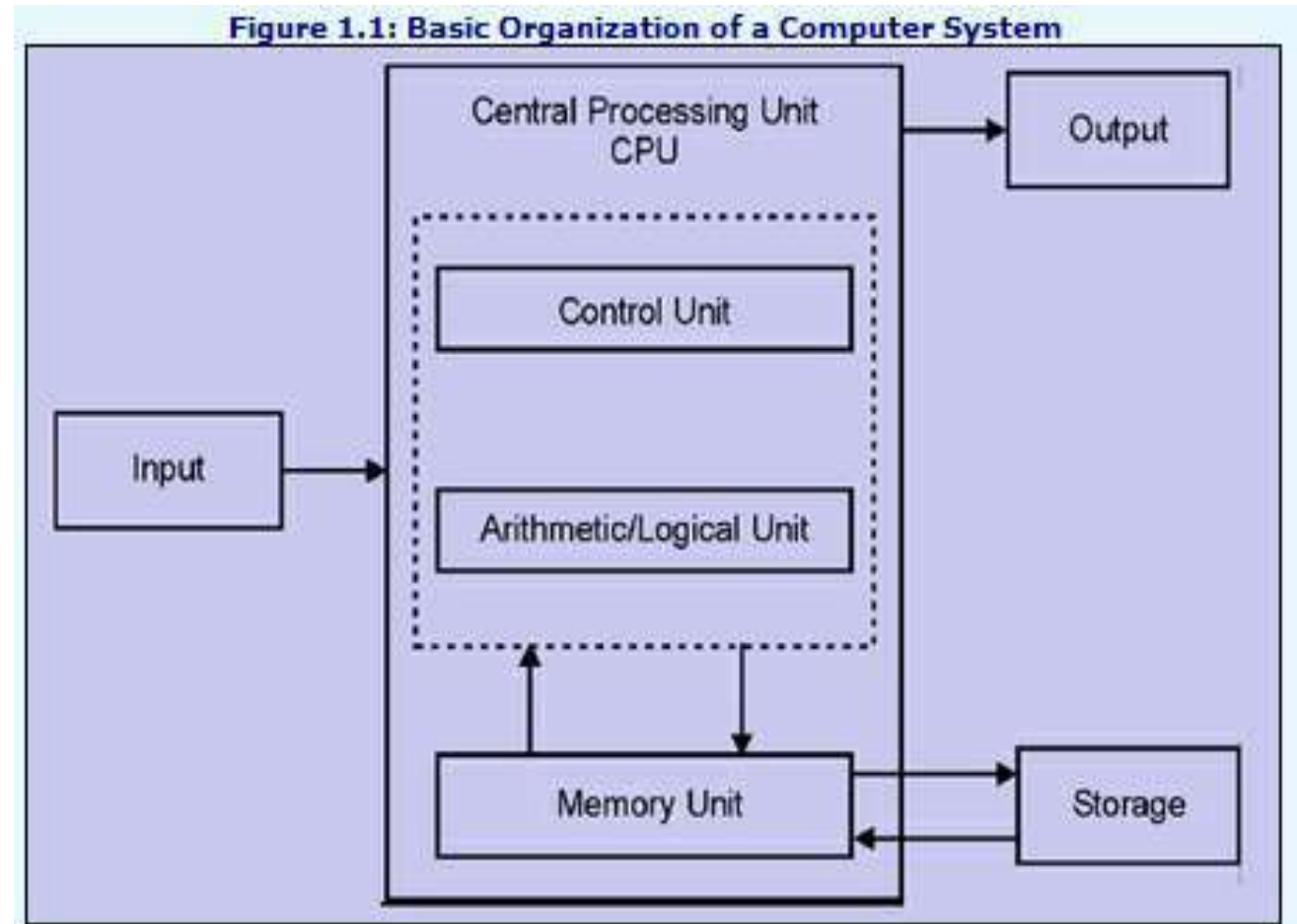
Chip select logic for
1k x 8 EPROM

RAM CS' Generation

- Now similarly find the CS generation ckt for RAM. 2k RAM.

UNIT 3 : BASIC COMPUTER ARCHITECTURE

- a specification detailing how a set of software and hardware technology standards interact to form a computer system or platform is called computer architecture.
- refers to how a computer system is designed and what technologies it is compatible with



- **Input Unit**
 - Accepts data and instructions given by user and converts to machine readable code. Common i/p device are mouse, keyboard, scanner, punched cards etc.
- **Processor / CPU**

Executes instruction of the computer program to perform specific task.

 - **Register**
 - Primary memory, used to store data during the time of processing inside ALU. Special purpose temporary storage location.
 - **Control Unit**
 - Controls overall operation of the CPU.
 - **ALU**
 - Carries out arithmetic and logical operation on data provided.
- **Main Memory**
 - Volatile memory, RAM, used to store instruction and data.
- **Output Unit**
 - Produces information or result after the computer finishes processing in user readable or understandable form.

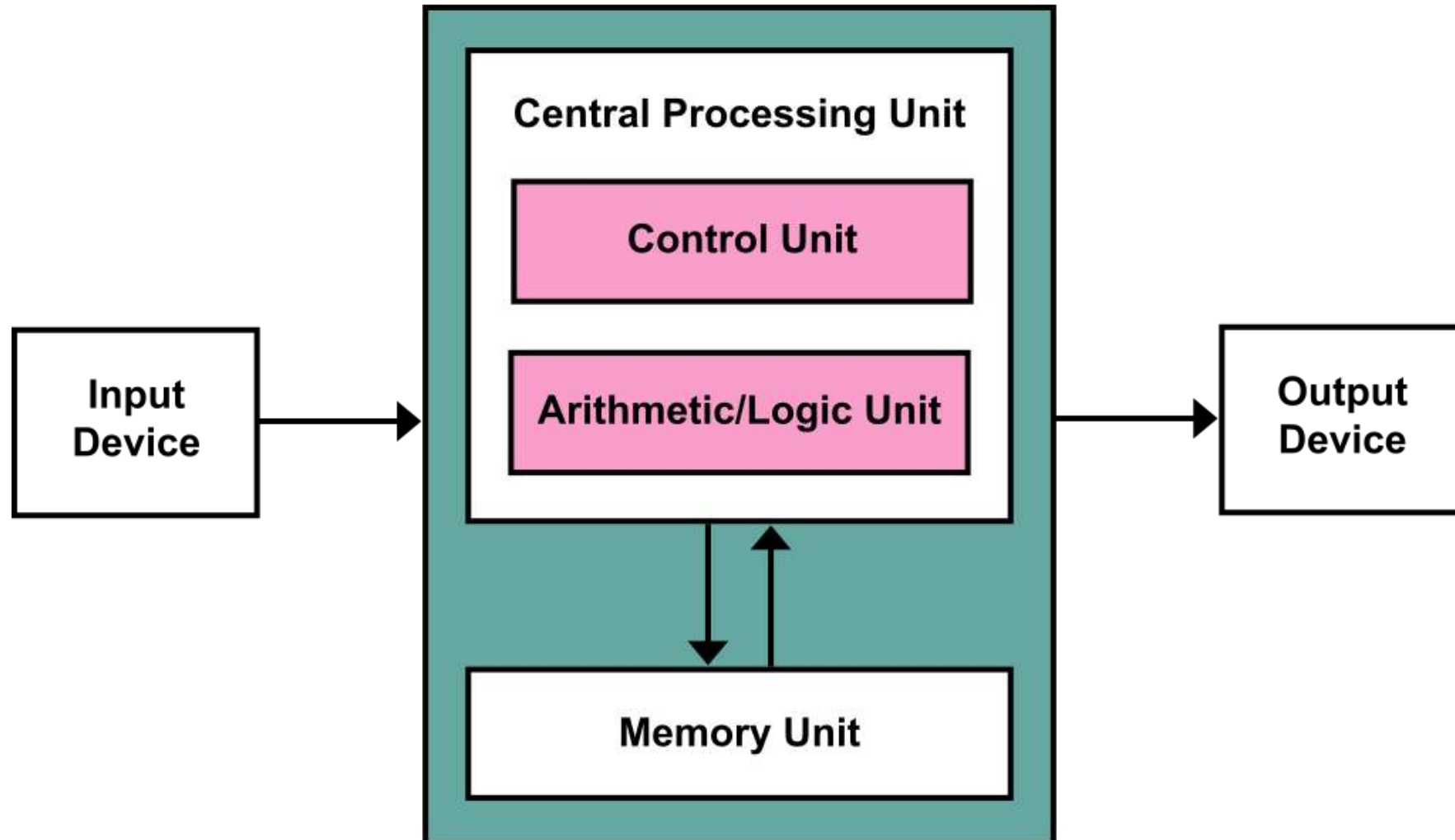
History of Computer Architecture

- First documented computer is known as Analytical engine.
- Designed and developed by Charles Babbage.
- Charles Babbage known as father of computer.
- Alan Turing is the person who designed the automatic computing engine.

John Von Neumann

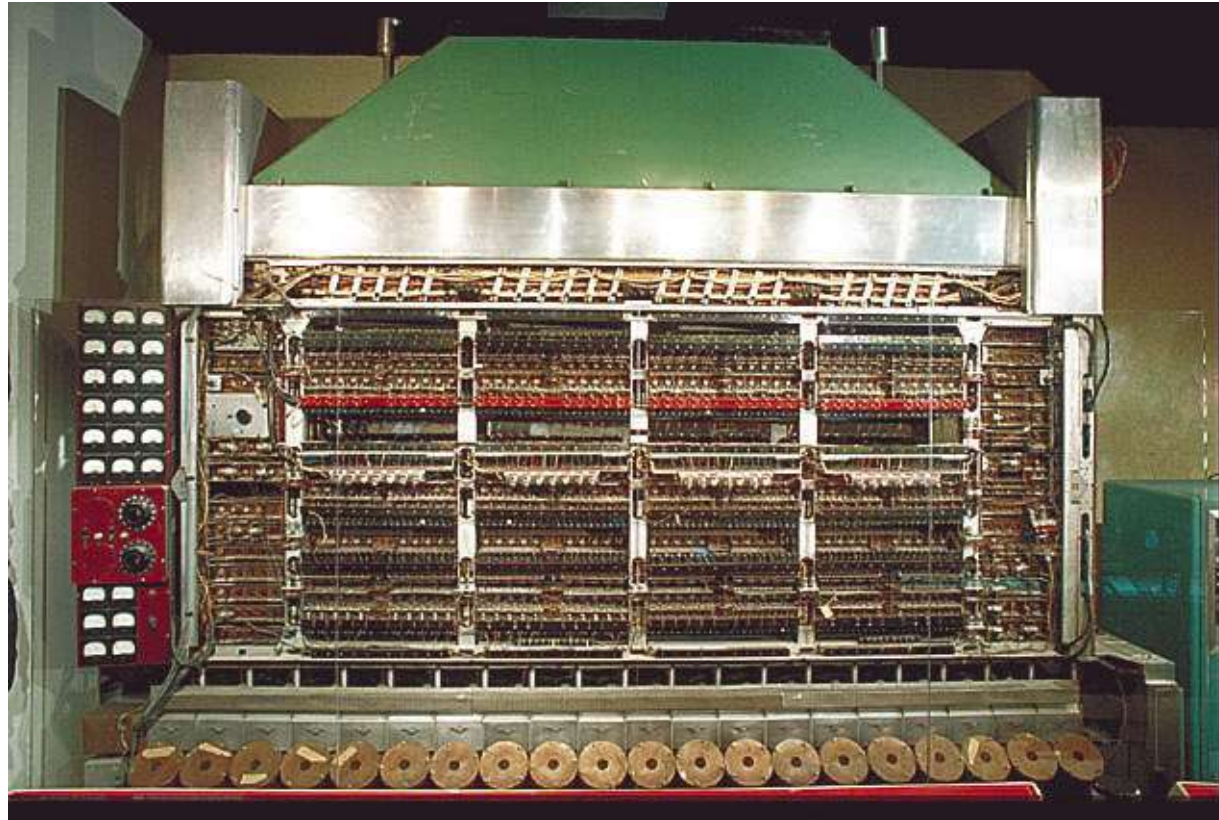
- Introduced stored program concept.
- Use of memory to store both the program and data was laid out by von Neumann.
- Processing unit consist of ALU and registers.
- Singe bus for memory access.
- Control unit consists of IR and PC.
- I/O mechanisms
- “Compare above features with 8085”

John Von Neumann



IAS Machine

- Princeton Institute for Advanced Studies (IAS) Computer



Features of IAS Machine

- Binary Computer
- 40 bit word size
- Memory comprising of 1024 words
- Total memory = $1024 * 40$ bits = 5kilobytes
- Negative number represented in two's Complement form.
- Two general purpose register available, Accumulator(AC) and Multiplier/Quotient(MQ)
- Asynchronous machine.
- Each word contains two 20-bit instructions comprising of 8 bit opcode and 12 bit address.

Features of IAS Machine

- Set of registers in CPU
 - MBR (Memory Buffer Register)
 - MAR (Memory Address Register)
 - IR (Instruction Register)
 - IBR (Instruction Buffer Register)
 - PC (Program Counter)
 - Accumulator (AC)
 - Multiplier Quotient (MQ)

Generations of Computers

- 1st Generation → Vacuum Tubes → 1946 – 1957
- 2nd Generation → Transistors → 1958 – 1964
- 3rd Generation
 - → Small Scale Integration → 1965 on → Up to 100 devices on a single chip.
 - → Medium Scale Integration → to 1971 → 100 to 3000 devices on a single chip
 - → Large Scale Integration → 1971 - 1977 → 3000 to 100000 devices on a single chip.
- 4th Generation → Very Large Scale Integration → 1978 – 1991 → 100000 to 1000000000 devices on a single chip
- 5th Generation → Ultra Large Scale Integration → 1991 - → Over 1000000000 device on a single chip.

Difference Between Computer Organization and Computer Architecture

Computer Organization	Computer Architecture
It is concerned with the structure and behavior of computer system as seen by user.	It is concerned with the way the hardware component interacts with the software.
Deals with components of connection in a system.	Acts as an interface between hardware and software.
It tells how exactly all the system are arranged and interconnected.	It helps us to understand the functionalities of the system.
Organization is done on the basis of architecture.	Architecture is done while designing the system in the first place.

Difference Between Von Neumann Architecture and Harvard Architecture

Von Neumann Architecture	Harvard Architecture
It is the theoretical design based on the stored program concept.	Modern design based on Harvard mark relay-based computer model
It uses same physical address bus for both instruction and data.	It uses separate memory address bus for instruction and data.
Process need two cycle to execute and instruction. EG: Instruction fetch, memory read.	Process need a single cycle as both instructions fetch and memory read can go concurrently.
Simple control unit design, hence development is cheaper and faster.	Control unit needed to be designed for two buses and is complicated.

Memory Hierarchy and Cache

- A modern memory subsystem combines
 - Fast small memories.
 - Slower large memories.
- Programmers wish for the memory to be
 - Large
 - Fast
 - Randomly accessible
- These wishes are not achievable by single kind of memory.
 - Cost factor
 - Fast technologies slow down when scaled to larger capacities.

Memory Examples

Technology	Typical access time	\$ per GB in 2008
SRAM	0.5 - 2.5 ns	\$2000 - \$5000
DRAM	50 - 70 ns	\$20 - \$75
Magnetic disk	5 - 20 ms	\$0.2 - \$2

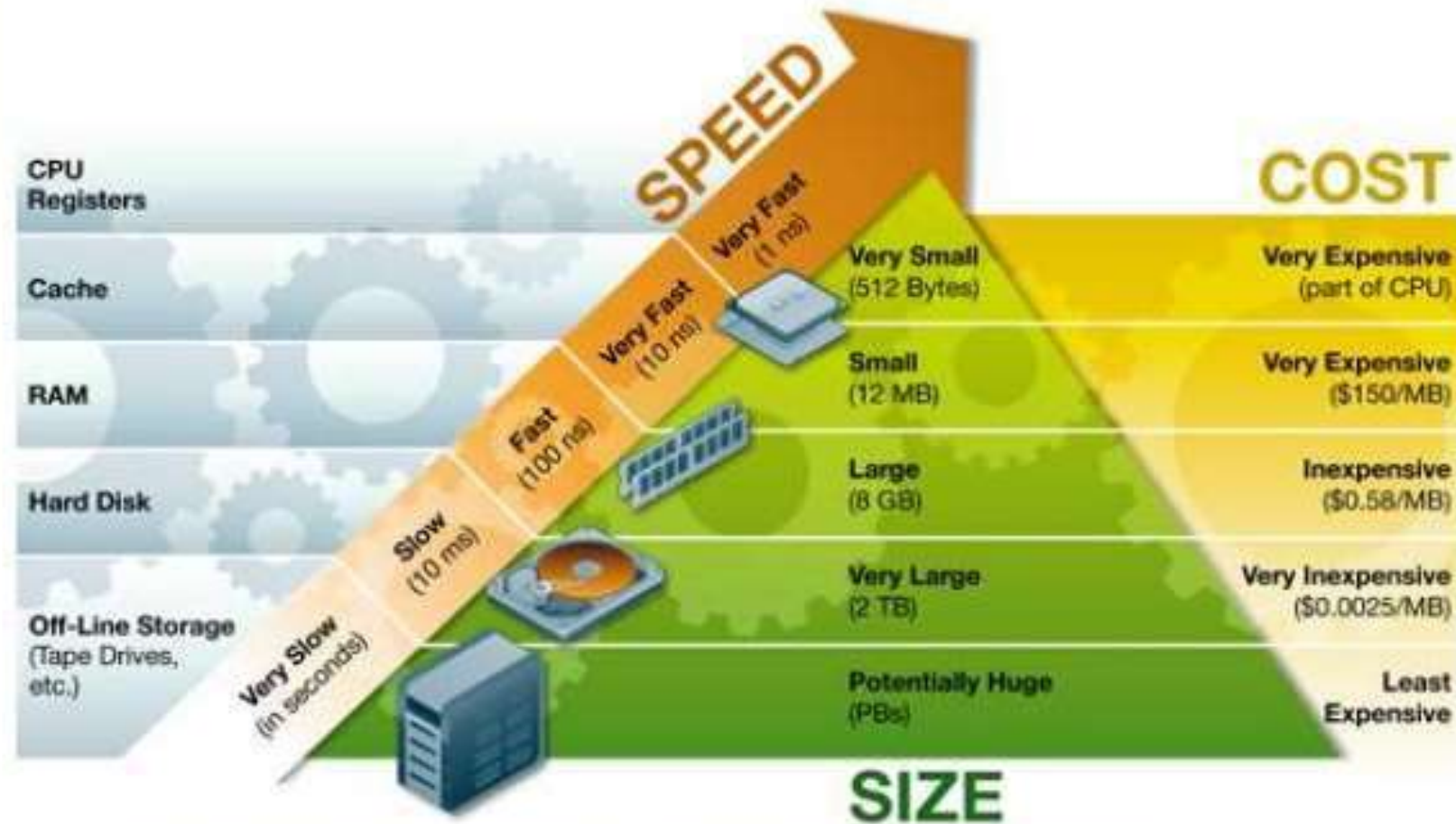
Locality of memory references

- Useful properties of memory references
 - Temporal Locality: a recently accessed memory location is likely to be accessed again.
 - Spatial Locality: memory location close to recently accessed locations are likely to be accessed in near future.
- These properties are exploited by memory hierarchy.

Ideas of memory hierarchy

- Use combinations of memory kinds
 - Larger amounts of cheaper slower memories.
 - Smaller amount of expensive faster memories.
- Take advantage of temporal locality
 - If access data from slower memory, move it to faster memory.
 - If data in faster memory unused recently, move it to slower memory.
- Take advantage of spatial locality
 - If need to move a word from slower to faster memory, move adjacent words at same time.

Extended Memory Hierarchy



Source: http://www.ts.avnet.com/uk/products_and_solutions/storage/hierarchy.html

What is a Cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spatial and temporal locality
- In computer architecture, almost everything is a cache!
 - Registers “a cache” on variables – software managed
 - First-level cache a cache on second-level cache
 - Second-level cache a cache on memory
 - Memory a cache on disk (virtual memory)
 - TLB a cache on page table
 - Branch-prediction a cache on prediction information?

Some Terminologies

- **Block (or line):** the minimum amount of data transferred between 2 adjacent memory levels
 - E.g. in range 16-256 bytes
- **Hit:** data is found at higher level – the ideal case
 - Operation performed quickly
- **Miss:** data not found
 - Must continue the search at the next level down
 - After data is eventually located, it is copied at the memory level where the miss happened
- **Hit/Miss ratio:** proportion of accesses that hit/miss, respectively

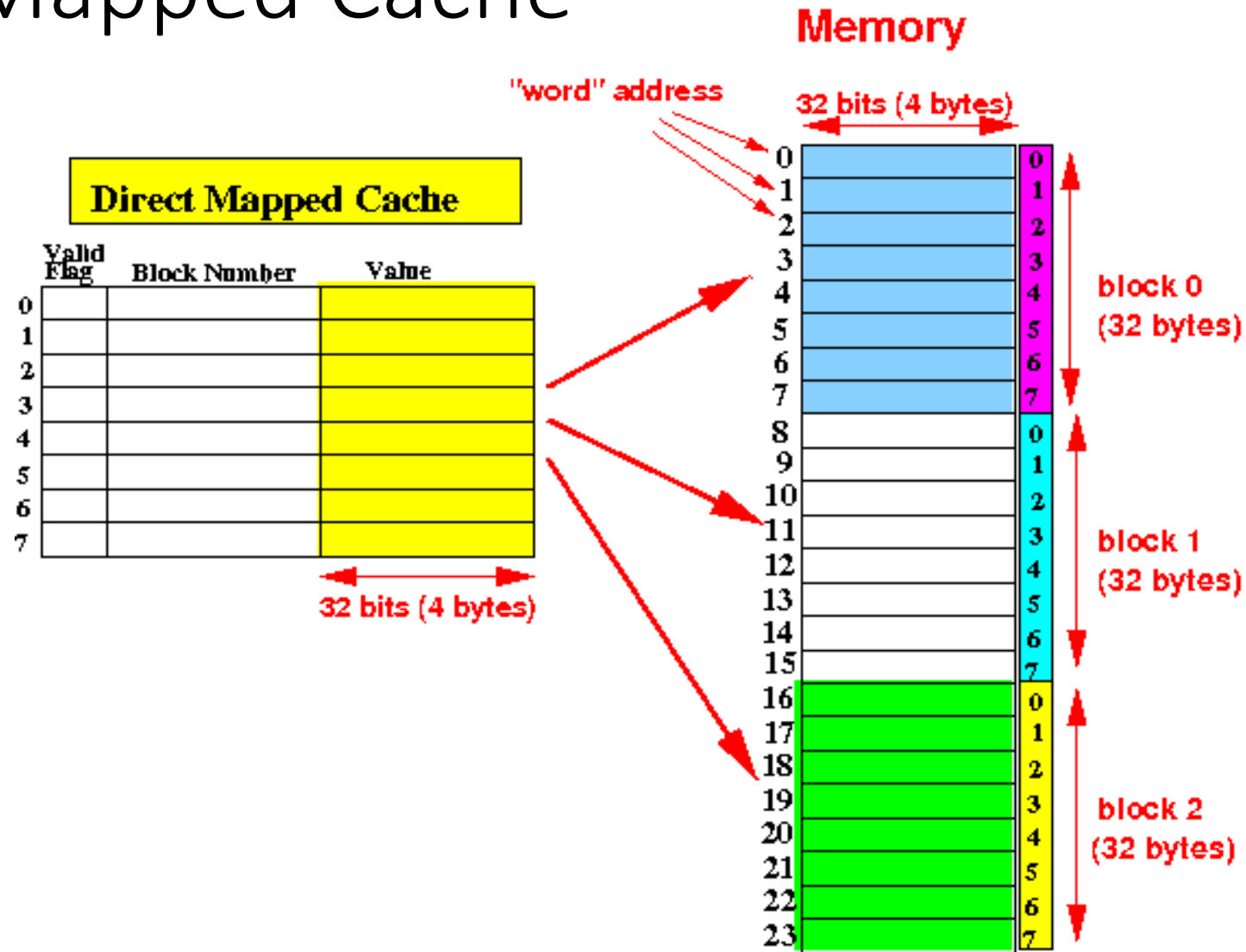
Some More Terminologies

- **Hit time:** Time required to access a level of hierarchy
- **Miss penalty:** Time required to fetch a block into a level of the hierarchy from the next level down

Cache Mapping Techniques

- Direct Mapped Cache
- Fully Associative Mapping Cache
- Set Associative Mapping Cache

Direct Mapped Cache



Fully Associative Cache

- ◆ **An address is partitioned to**

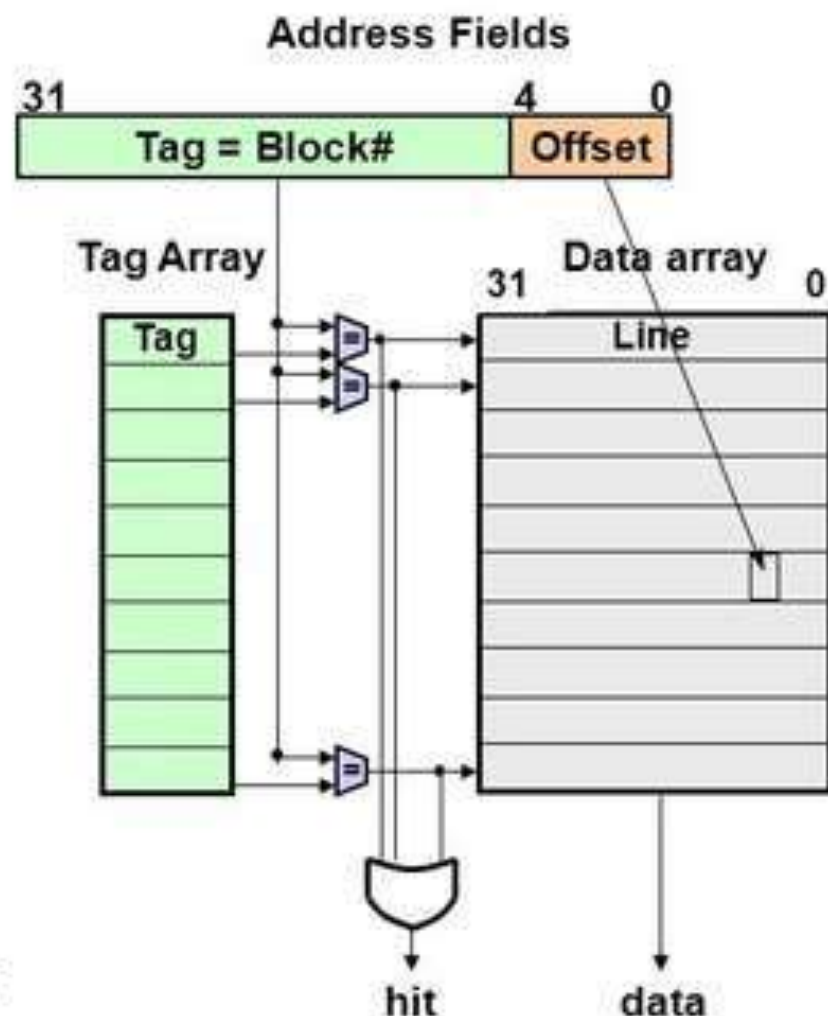
- ❖ offset within block
- ❖ block number

- ◆ **Each block may be mapped to each of the cache lines**

- ❖ Lookup block in all lines

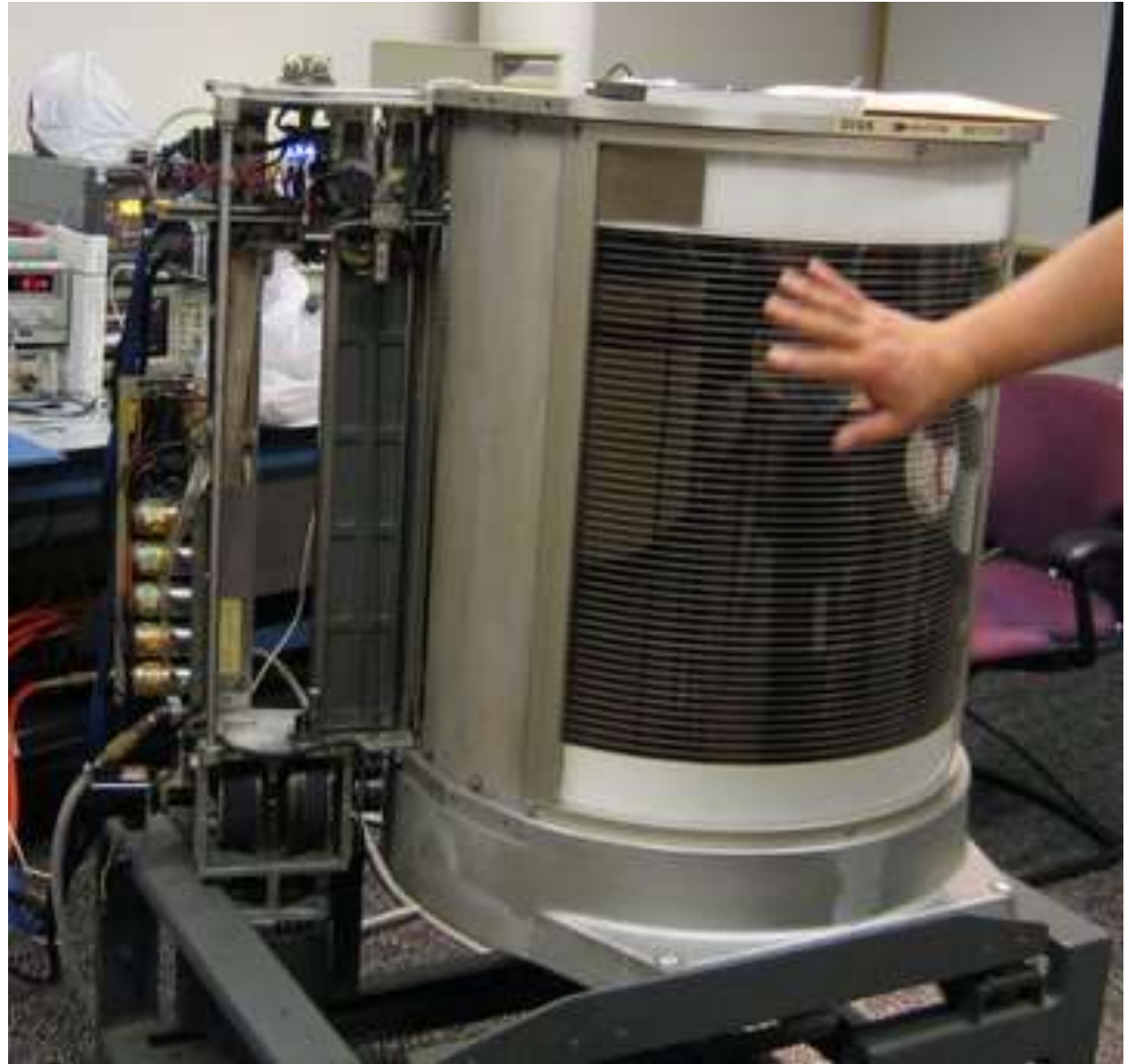
- ◆ **Each cache line has a tag**

- ❖ All tags are compared to the block# in parallel
- ❖ Need a comparator per line
- ❖ If one of the tags matches the block#, we have a hit
 - Supply data according to offset



Hard Disk

- 65 years old hard disk.



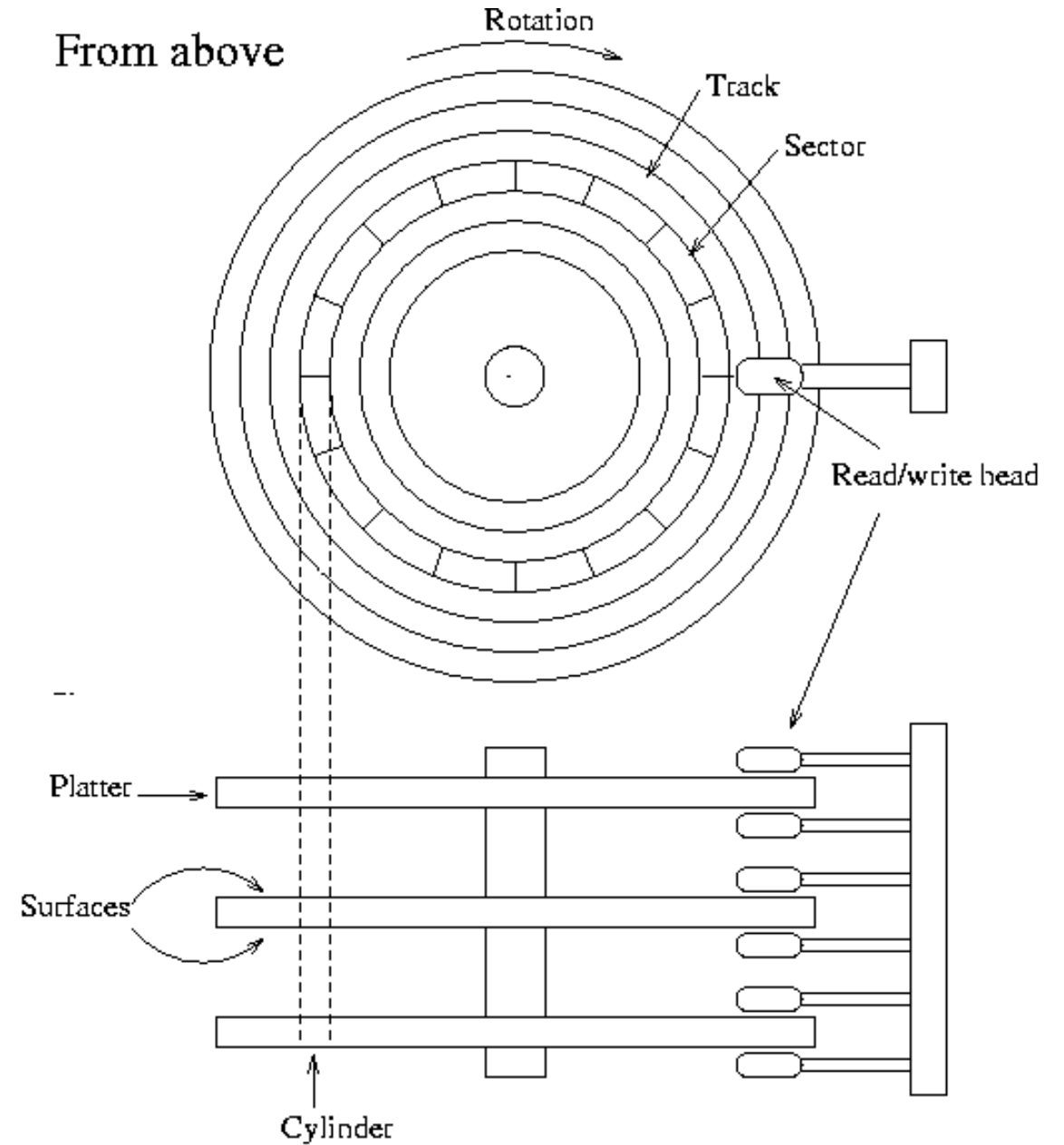
Hard Disk

- The hard disk is the primary storage unit of the computer.
- A hard disk consists of a stack of disk platters that are made up of aluminum alloy or glass coated with a magnetic material.
- The surface of a disk is divided into imaginary tracks and sectors.
- Tracks are concentric circles where the data is stored.
- These tracks are numbered from the outermost ring to the innermost ring, starting from zero.
- Disk sectors refer to the number of fixed size areas that can be accessed by one of the disk drive's read/write heads, in one rotation of the disk, without the head having to change its position.

Hard Disk

- An intersection of a track and a disk sector is known as track sector. Each sector is uniquely assigned a disk address before a disk drive can access a piece of data.
- In order to make the disk usable, first it must be formatted to create tracks and sectors.
- The track sectors are grouped into a collection known as cluster. It refers to the basic allocation unit for storage on a disk

Schematic picture of hard disk



Computer Architecture

- **computer architecture** is a set of rules and methods that describe the functionality, organization, and implementation of computer systems.
- is a specification detailing how a set of software and hardware technology standards interact to form a computer system or platform.
- the art of determining the needs of the user/system/technology, and creating a logical design and standards based on those requirements.

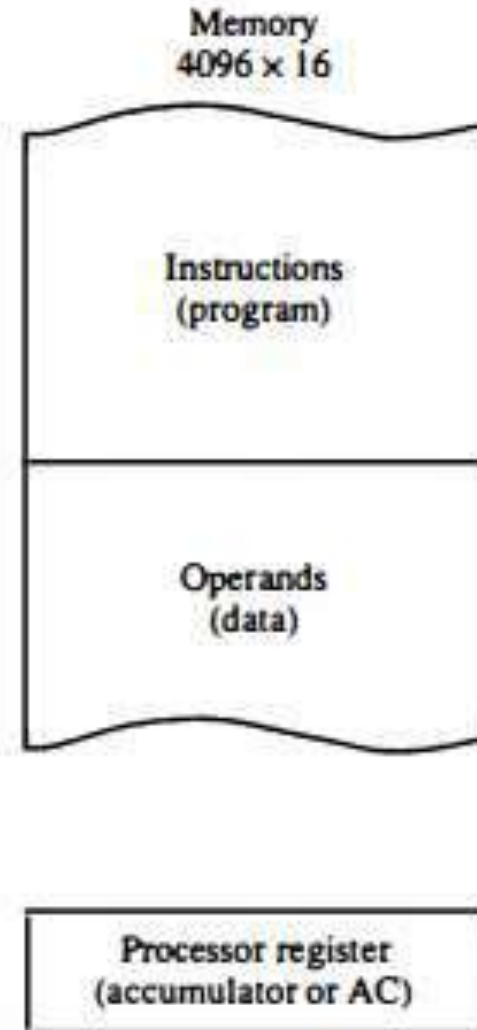
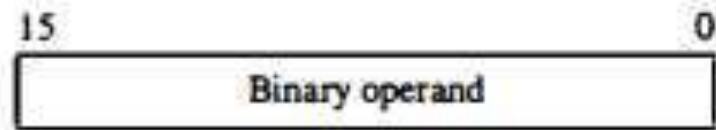
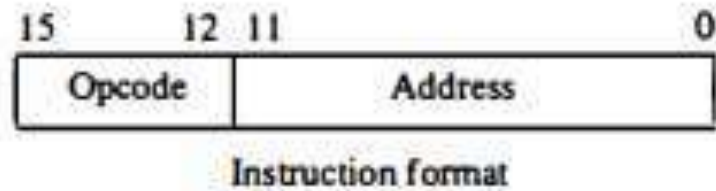
Sub Categories of Computer Architecture

- **Instruction set architecture (ISA):** defines the machine code that a processor reads and acts upon as well as the word size, memory address modes, processor registers, and data type.
- **Microarchitecture:** also known as "computer organization", this describes how a particular processor will implement the ISA. The size of a computer's CPU cache for instance, is an issue that generally has nothing to do with the ISA.
- **Systems design:** includes all of the other hardware components within a computing system, such as data processing other than the CPU (e.g., direct memory access), virtualization, and multiprocessing

Stored Program Organization

- The simplest way to organize a computer is to have one processor register and an instruction code format with two parts.
- The first part specifies the operation to be performed and the second specifies an address.
- The memory address tells the control where to find an operand in memory.
- This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.

Stored Program Organization

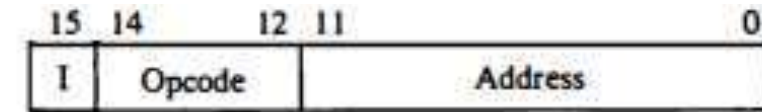


Direct Address

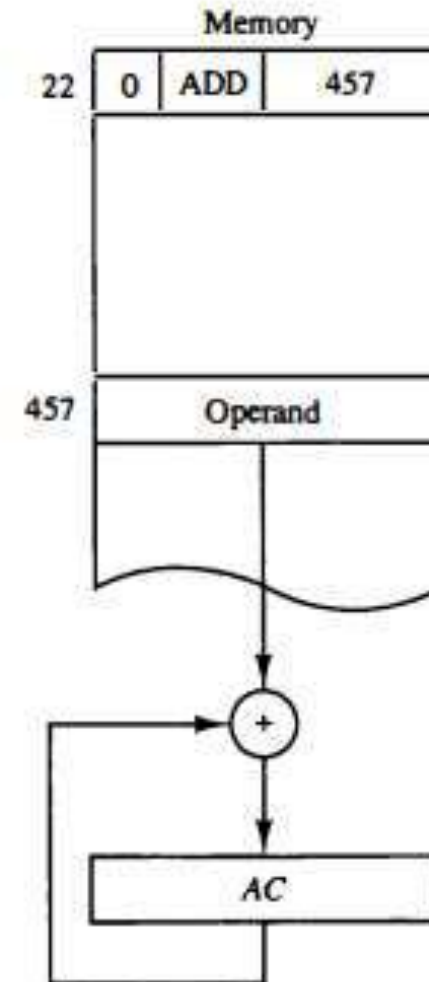
- It is sometimes convenient to use the address bits of an instruction code not as an address but as the actual operand.
- When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand.
- When the second part specifies the address of an operand, the instruction is said to have a direct address. This is in contrast to a third possibility called indirect address, where the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found.
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.

Indirect Address

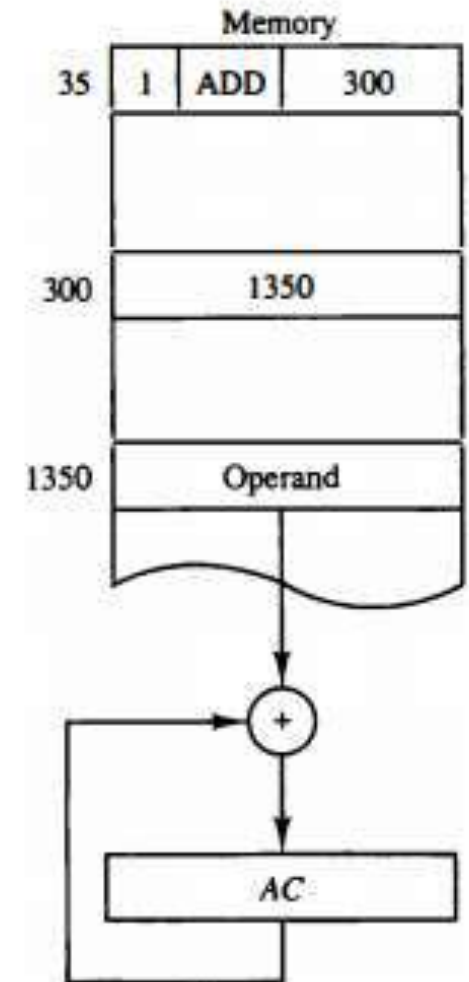
- It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I.
- The direct addressing is shown in (b) where the 457 as the address of 12 bit is the address and hence the operand is fetched from there.
- The indirect addressing is shown in (c) where the 300 stored in address bit of the instruction is the address where the address location from 300 is to be read and then fetched from memory again. So the memory fetch operation is done twice. Once to read the 300 location and the next to read the operand location of 1350.



(a) Instruction format



(b) Direct address



(c) Indirect address

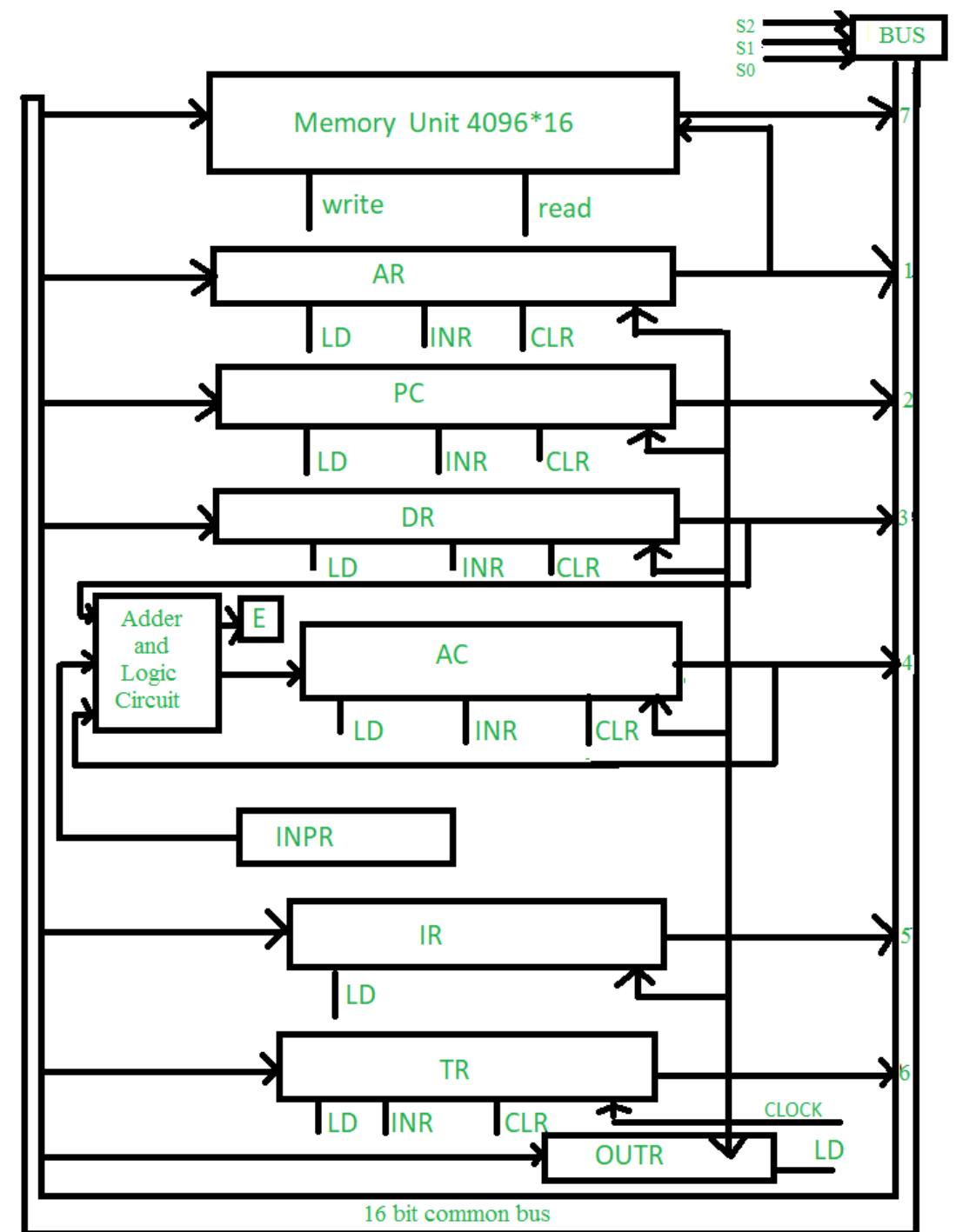
Computer Registers

Register Symbol	Number of Bits	Register Name	Function
DR	16	Data Register	Holds memory operands
AR	12	Address Register	Holds Address for memory
AC	16	Accumulator	Processor Register connected to ALU
IR	16	Instruction Register	Holds Instruction code
PC	12	Program Counter	Holds address of next Instruction
TR	16	Temporary Register	Holds Temporary data, usually connected with ALU
INPR	8	Input Register	Holds Input byte from port
OUTR	8	Output Register	Holds Output byte to port

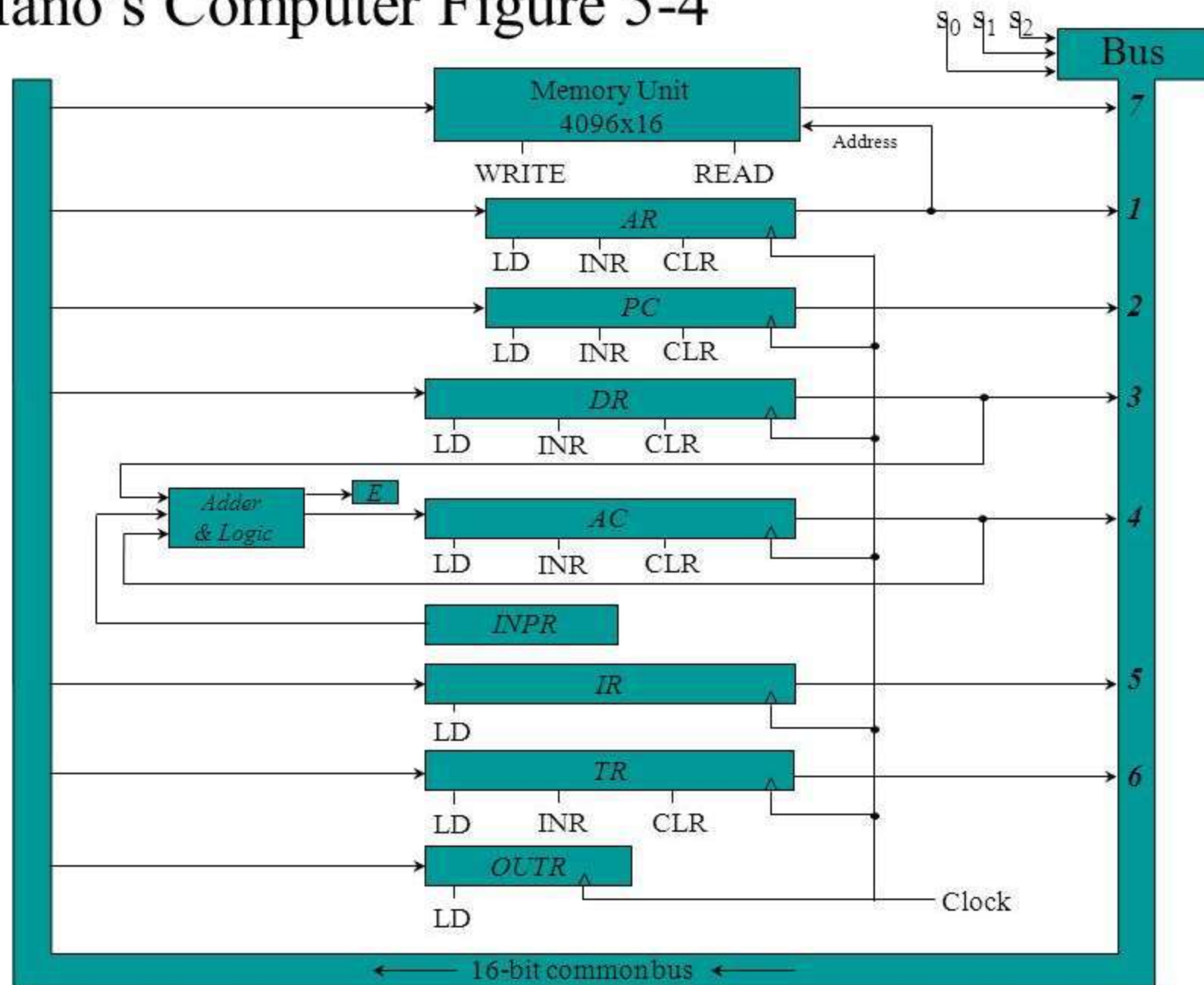
Common Bus System

- The basic computer has eight registers, a memory unit, and a control unit.
- Paths must be provided to **transfer information from one register to another and between memory and registers**.
- The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers.
- A more efficient scheme for transferring information in a system with many registers is to use a common bus.
- We can construct a bus system **using multiplexers or three-state buffer gates**. We just have to connect the **registers and memory** of the basic computer to a common bus system.

Niraj Khadka



Mano's Computer Figure 5-4



Common Bus System

- 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.
- The **lines from the common bus are connected to the inputs of each register and the data inputs of the memory.**
- 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 $S_2S_1S_0 = 111$.

Common Bus System

- Four registers, **DR, AC, IR, and TR**, have 16 bits each. Two registers, AR and PC, have 12 bits each since they hold a memory address.
- When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to 0's.
- When AR or PC receive information from the bus, only the 12 least significant bits are transferred into the register.
- 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. This is because **INPR receives a character from an input device which is then transferred to AC .**

Common Bus System

- OUTFR receives a character from AC and delivers it to an output device. There is no transfer from OUTFR to any of the other registers.
- 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). This type of register is equivalent to a binary counter with parallel load and synchronous.
- The increment operation is achieved by enabling the count input of the counter.

Common Bus System

- The input data and output data of the memory are connected to the common bus, but the **memory address is connected to AR** .
- Therefore, AR must always be used to specify a memory address. By using a single register for the address, we eliminate the need for an address bus that would have been needed otherwise.
- The content of any register can be specified for the memory data input during a write operation. Similarly, any register can receive the data from memory after a read operation except AC .

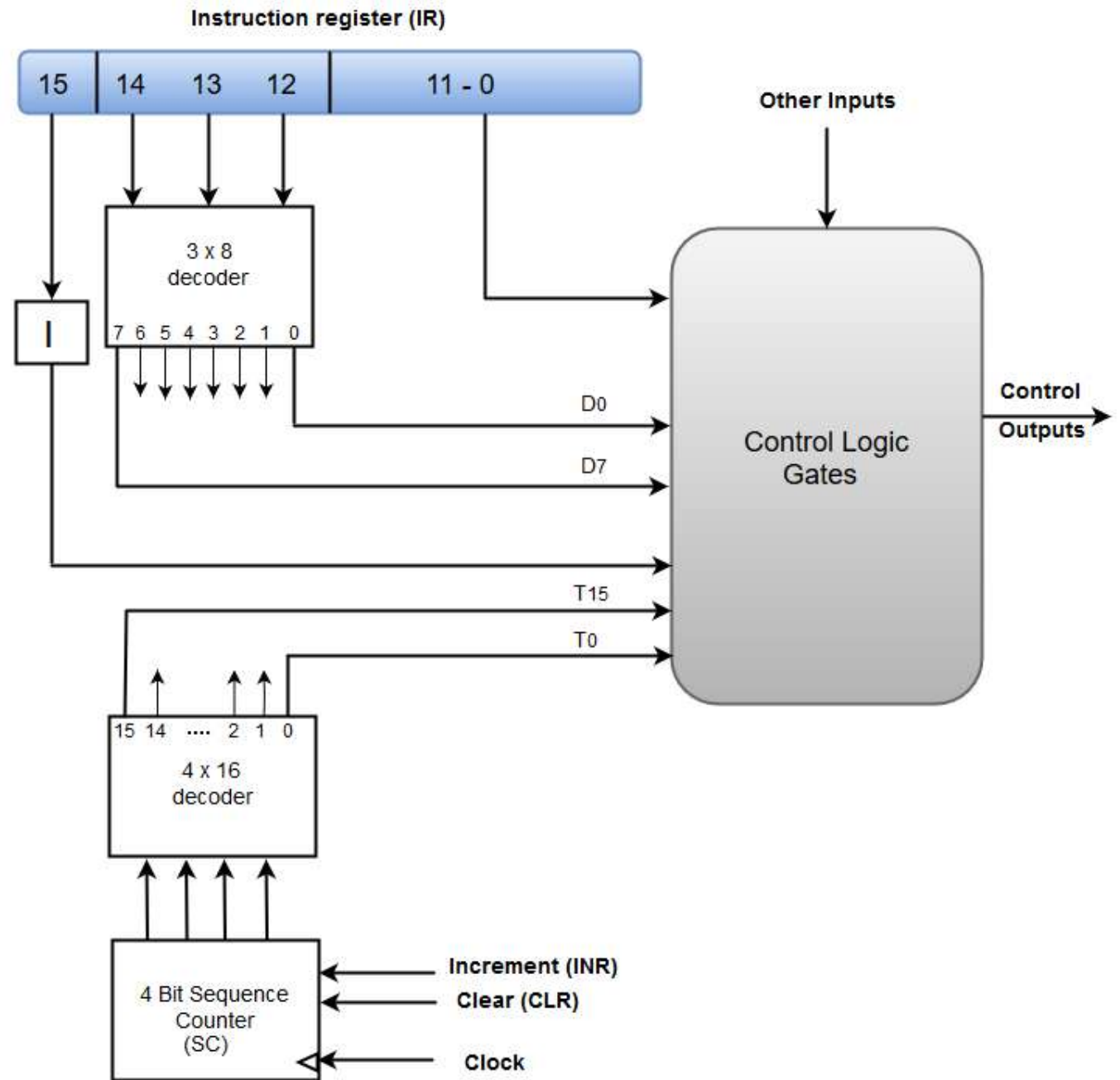
Common Bus System

- The 16 inputs of AC come from an adder and logic circuit. **This circuit has three sets of inputs.** One set of 16-bit inputs come from the outputs of AC .
- They are used to implement register microoperations such as complement AC and shift AC .
- Another set of 16-bit inputs come from the data register DR .
- **The inputs from DR and AC are used for arithmetic and logic microoperations,** such as add DR to AC or AND DR to AC .
- The result of an addition is transferred to AC and the end carry-out of the addition is transferred to flip-flop **E**

Common Bus System

- Note that the content of any register can be applied onto the bus and an **operation can be performed in the adder and logic circuit during the same clock cycle.**
- The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC .
- For example, the two microoperation. **$DR \leftarrow AC$** and **$AC \leftarrow DR$** can be executed at the same time.
- This can be done by placing the content of AC on the bus (with $S_2S_1S_0 = 100$), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC, and enabling the LD (load) input of AC, all during the same clock cycle. The two transfers occur upon the arrival of the clock pulse transition at the **end of the clock cycle.**

Timing and Control Instruction Cycle



Timing and Control Instruction Cycle

- All sequential circuits in basic computer are driven by a master clock with a exception of INPR.
- The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit.
- The clock pulses do not change the state of a register unless the register is enabled by a control signal.
- The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.

Timing and Control Instruction Cycle

- There are two major types of control organization
 - Hardwired Control
 - Microprogrammed Control
- **In the hardwired organization**, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
- It has the advantage that it can be optimized to produce a fast mode of operation.
- A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed.

Timing and Control Instruction Cycle

- In the microprogrammed organization, the control information is stored in a control memory.
- The control memory is programmed to initiate the required sequence of microoperations.
- **In the microprogrammed control**, any required changes or modifications can be done by updating the microprogram in control memory.

Timing and Control Instruction Cycle

- In the microprogrammed organization, the control information is stored in a control memory.
- The control memory is programmed to initiate the required sequence of microoperations.
- **In the microprogrammed control**, any required changes or modifications can be done by updating the microprogram in control memory.

UNIT 4: MICROPROGRAMMED CONTROL

- Terminology

- Hardwired Control Unit

- When control signals are generated by hardware using logic gates and conventional logic design techniques , the control unit is said to be Hardwired Control unit.

- Microprogrammed Control unit

- When control signals are generated by micro instructions stored in a sperate memory, it is called microprogrammed control unit.

- Control Memory

- Storage in a microprogrammed control unit to store the microprogram.

- Control Word

- Control variables at any time can be represented by a control word string of 1's and 0's called control word.

- Terminology

- Microoperations

- Perform basic operations on data stored in one or more registers, including transferring data between registers or between registers and external buses of the CPU, and performing arithmetic or logical operations on registers.

- Microcode

- Very low level instruction set which is stored permanently in a computer or peripheral controller and controls the operation of the device.

- Microinstructions

- A single instruction in microcode. The most elementary instruction in a computer, such as moving the content of a register to ALU.

- Microprogram

- A set or sequence of microinstructions.

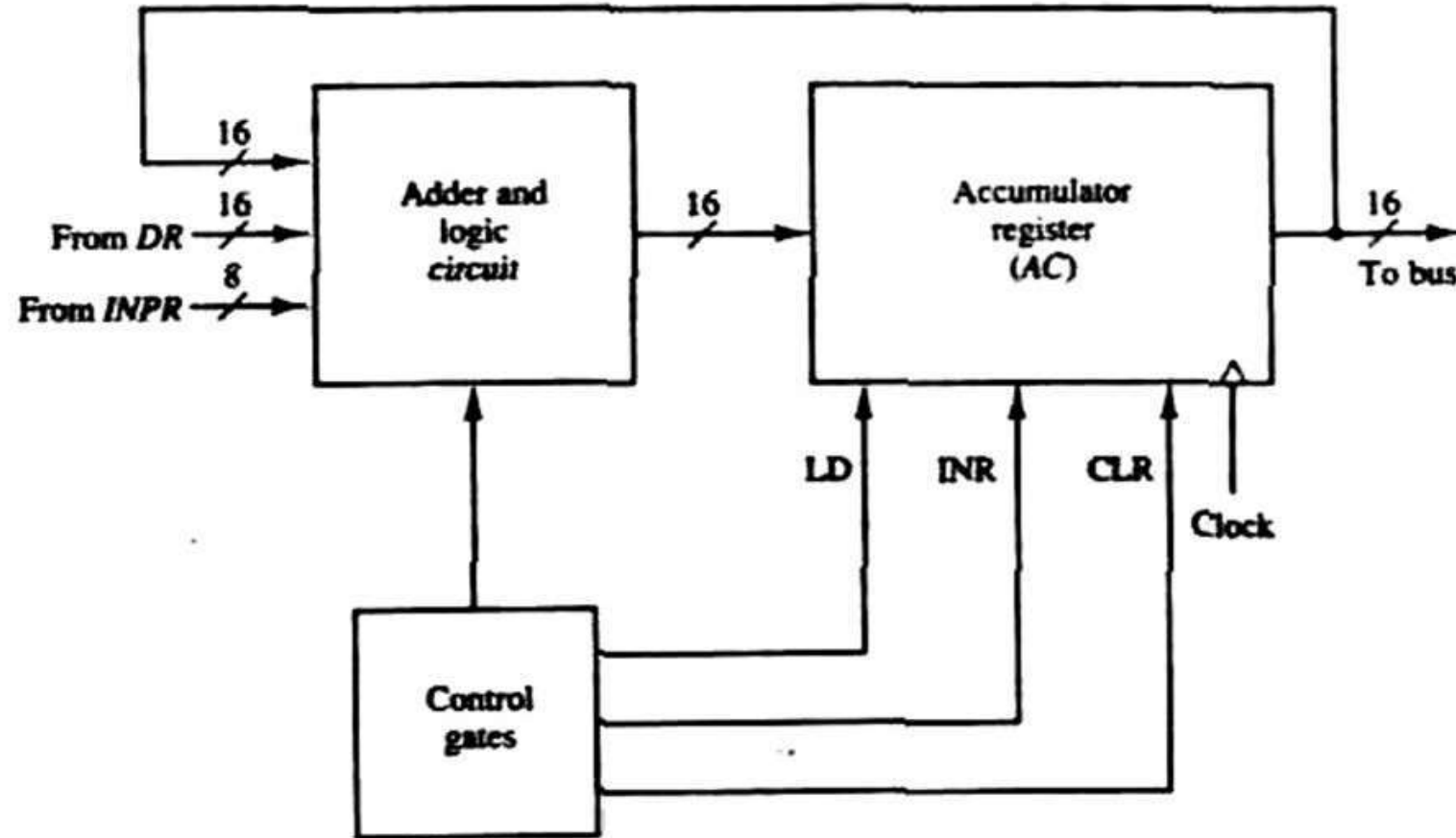
Design of Basic Computer

- The basic computer consists of the following hardware components.
 - A memory unit with 4096 words of 16 bit each.
 - Nine registers: AR (Address Register), PC (Program Counter), DR (Data Register), AC (Accumulator), IR (Instruction Register), TR (Temporary Register), OUTR (Output Register), INPR (Input Register) and SC (Sequence Counter).
 - Flip-Flops: IEN (Interrupt Enable), FGI (Input Flag), FGO (Output Flag).
 - Two Decoders: a 3x8 operation decoder and a 4x16 timing decoder.
 - A 16 bit common bus.
 - Control logic gates.
 - Adder and Logic Circuit connected to the input of AC.

Design of Accumulator Logic

- The circuits associated with the AC register are shown in next slides.
- The adder and logic circuits has three set of inputs.
- One set of 16 bit input comes from the output of AC.
- Another set of 16 bit comes from the DR.
- The third set of 8 bit comes from the INPR.
- The output of the adder and logic circuit provide the data input for the register.
- In addition, it is necessary to include logic gates for controlling the LD, INR, and CLR in the register and for controlling the operation of the adder and logic circuit.

Design of Accumulator Logic



Circuits associated with AC

Control of AC Register – Gate Structure

- The gate structure that controls the LD, INR and CLR inputs of AC is shown in next slide.
- The output of the AND gate that generates this control function is connected to the CLR input of the register.
- Similarly, the output of the gate that implements the increment micro operation is connected to the INR input of the register.
- The other seven micro operations are generated in the adder and logic circuit and are loaded into AC at the proper time.
- The outputs of the gate for each control function is marked with symbolic name. These outputs are used in the design of the adder and logic circuit.

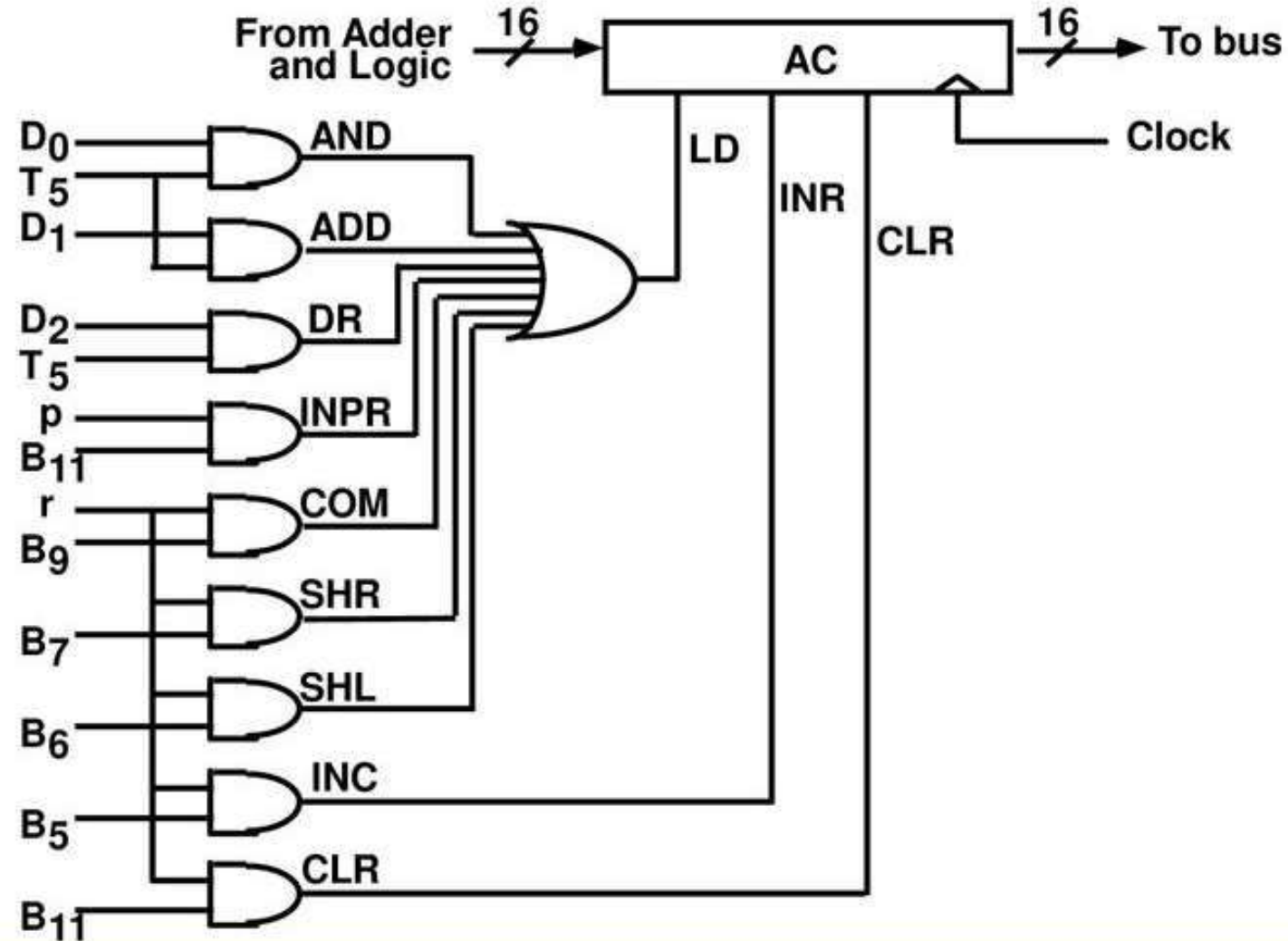
Operations that change Accumulator Data

- $D_0T_5 \Rightarrow$ and with DR $\Rightarrow AC \leftarrow AC \wedge DR$
- $D_1T_5 \Rightarrow$ add with DR $\Rightarrow AC \leftarrow AC + DR$
- $D_2T_5 \Rightarrow$ transfer from DR $\Rightarrow AC \leftarrow DR$
- $PB_{11} \Rightarrow$ transfer from INPR $\Rightarrow AC(0-7) \leftarrow INPR$
- $rB_9 \Rightarrow$ complement $\Rightarrow AC \leftarrow AC'$
- $rB_7 \Rightarrow$ shift right $\Rightarrow AC \leftarrow AC \gg 1, AC(15) \leftarrow E$
- $rB_6 \Rightarrow$ shift left $\Rightarrow AC \leftarrow AC \ll 1, AC(0) \leftarrow E$
- $rB_{11} \Rightarrow$ clear $\Rightarrow AC \leftarrow 0$
- $rB_5 \Rightarrow$ increment $\Rightarrow AC \leftarrow AC + 1$

Operations that change Accumulator Data

- $LD = D_0T_5 + D_1T_5 + D_2T_5 + PB_{11} + rB_9 + rB_7 + rB_6$
- $CLR = rB_{11}$
- $INC = rB_5$
- Now draw the figure according to the Boolean expression above.

Control of AC Register – Gate Structure



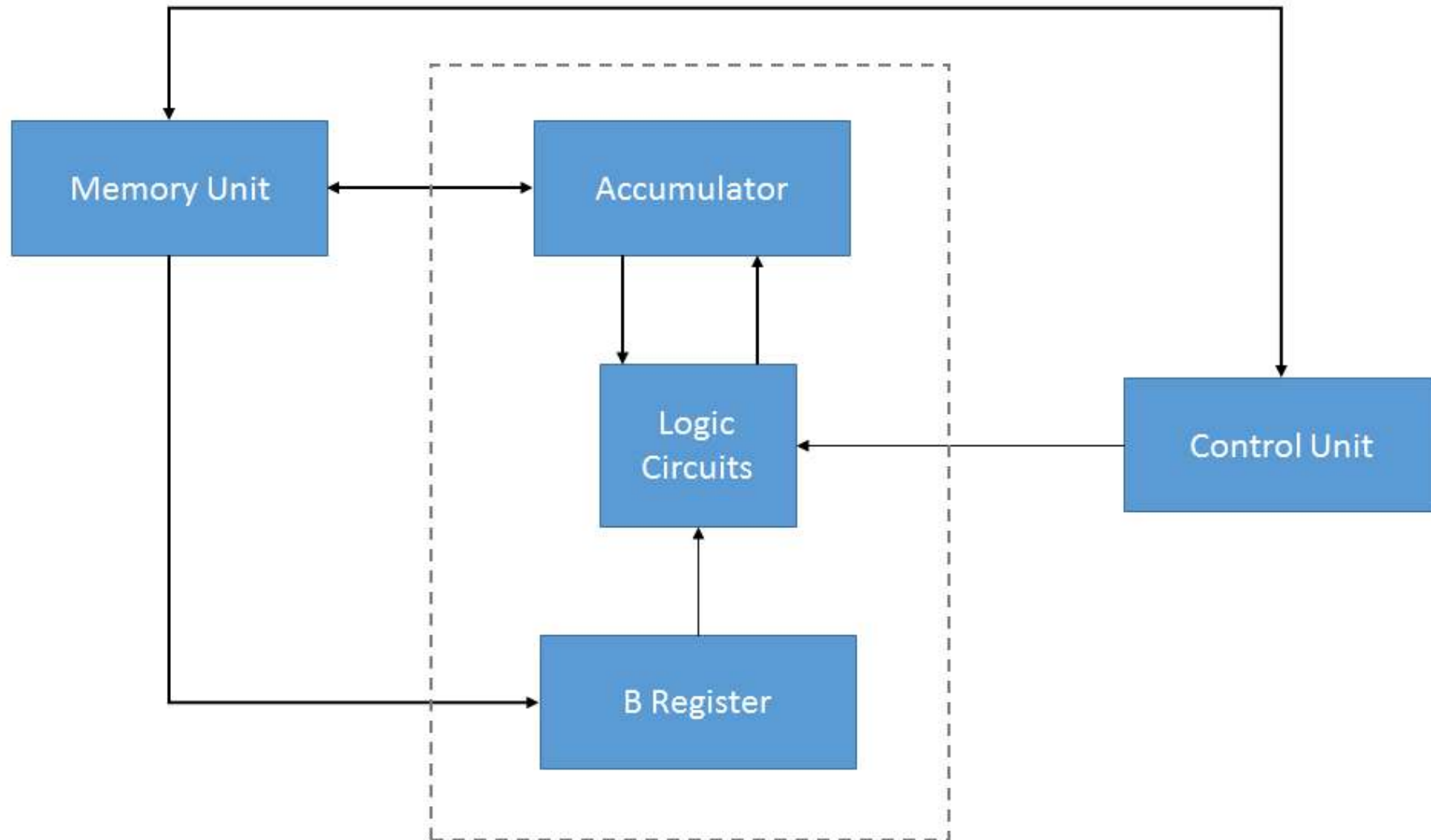
Design control of AR register – Gate Structure

- The statements that change the content of the AR register are given below:
 - R'T0 : AR <- PC
 - R'T2 : AR <- IR(0-11)
 - D7'IT3 : AR <- M[AR]
 - RT0 : AR <- 0
 - D5T4 : AR <- AR+1

ALU ORGANIZATION

- Various circuits are required to process data or perform arithmetical operations which are connected to microprocessor's ALU.
- Accumulator and data buffer stores data temporarily. These data are processed as per control instructions to solve problems. Such problems are addition, multiplication etc.

ALU ORGANIZATION



Functions of ALU

- Can be categorized in 3 types.
- Arithmetic Operations
 - Addition, multiplication etc.
 - Finding greater, smaller or equal to by using subtraction.
- Logical Operations
 - AND, OR, NOR, NOT etc using logical circuitry.
- Data Manipulations
 - Clearing a register, shifting contents of register to right and left.

CONTROL MEMORY

- The function of control unit in a digital computer is to initialize the sequence of micro-operations.
- The principle of microprogramming is an elegant and systematic method for controlling the microoperation sequences in a digital computer.
- A control unit whose binary control variables are stored in memory is called microprogrammed control unit.
- The memory within a control unit that stores the binary control variables to generate different control signals is called control memory.

CONTROL MEMORY

- Each word in a control memory contains a microinstruction within it.
- A sequence of microinstructions constitute a microprogram.
- The memory can either be read only or writable (dynamic microprogramming) too.
- A computer that employs a microprogram control unit will have two separate memories, a main memory and a control memory.

MicroProgram

- Program residing in a control memory constituting many microinstructions to generate all the required control signals to execute the instruction set correctly is called a microprogram.

MicroInstruction

- Contains a control word and a sequencing word.
 - Control Word - All the control information required for one clock cycle
 - Sequencing Word - Information needed to decide the next microinstruction address
- Vocabulary to write a microprogram.

Address Sequencer

- The next address generator is sometimes called a microprogram sequencer as it determines the address sequence that is read from the control memory.
- Function of microprogrammed sequence are
 - Incrementing of the Control Address Register.
 - Branching based on status bits.
 - Mapping process from bits of instruction to an address for control memory.
 - Facility for subroutine call and return.

ADDRESS SEQUENCER

- Incrementing CAR
 - $CAR = CAR + 1$
 - CAR is incremented by 1 so that the next microinstruction can be fetched to produce desired control signal.

ADDRESS SEQUENCER

- Branching based on Status bit
 - Can be conditional and unconditional branching.
 - Branching refers to jumps.
 - Some are done based on the status bits of the microprocessor.
 - Used when the control has to return to the fetch routine.

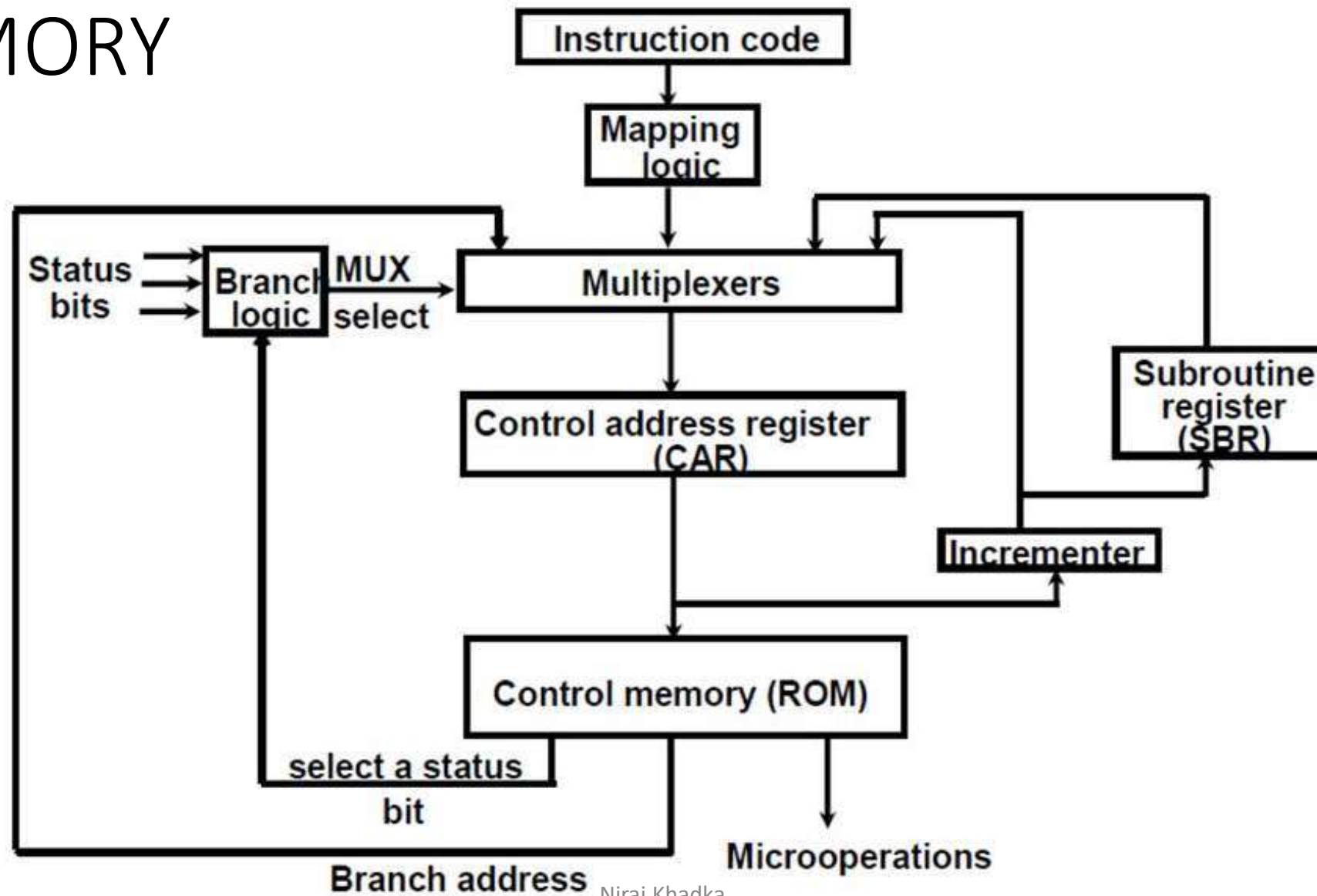
ADDRESS SEQUENCER

- Mapping Process
 - Mapping procedure is a rule that transforms the instruction code into a control memory address.
 - The opcode part from the instructions can be mapped to output different address for CAR which in turns can sequence the loading of the microinstructions.

ADDRESS SEQUENCER

- Subroutine call and return
 - Microprograms that employ subroutines will require an external register for storing the return address.
 - When the execution of the instruction is completed, control must return to the fetch routine.
 - Accomplished by unconditional branch microinstruction to the first address of fetch routine.

SELECTION OF ADDRESS FROM CONTROL MEMORY

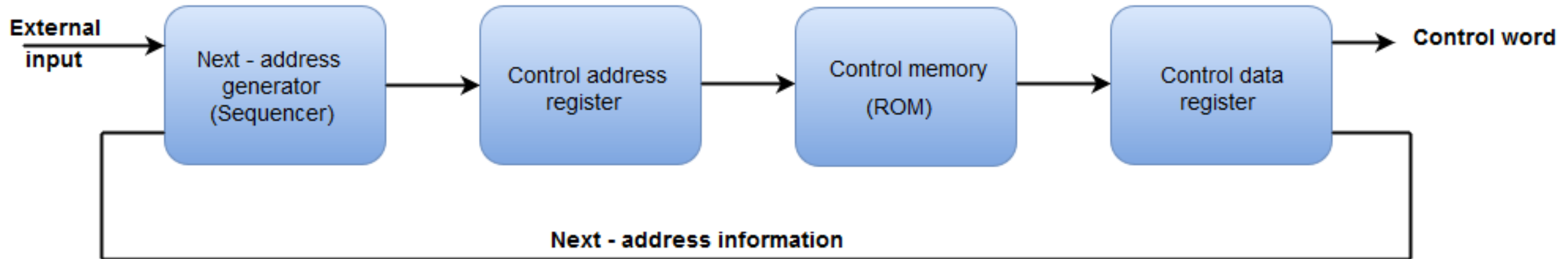


SELECTION OF ADDRESS FROM CONTROL MEMORY

- The microinstructions in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.

Microprogrammed Control Organization

Microprogrammed Control Unit of a Basic Computer:



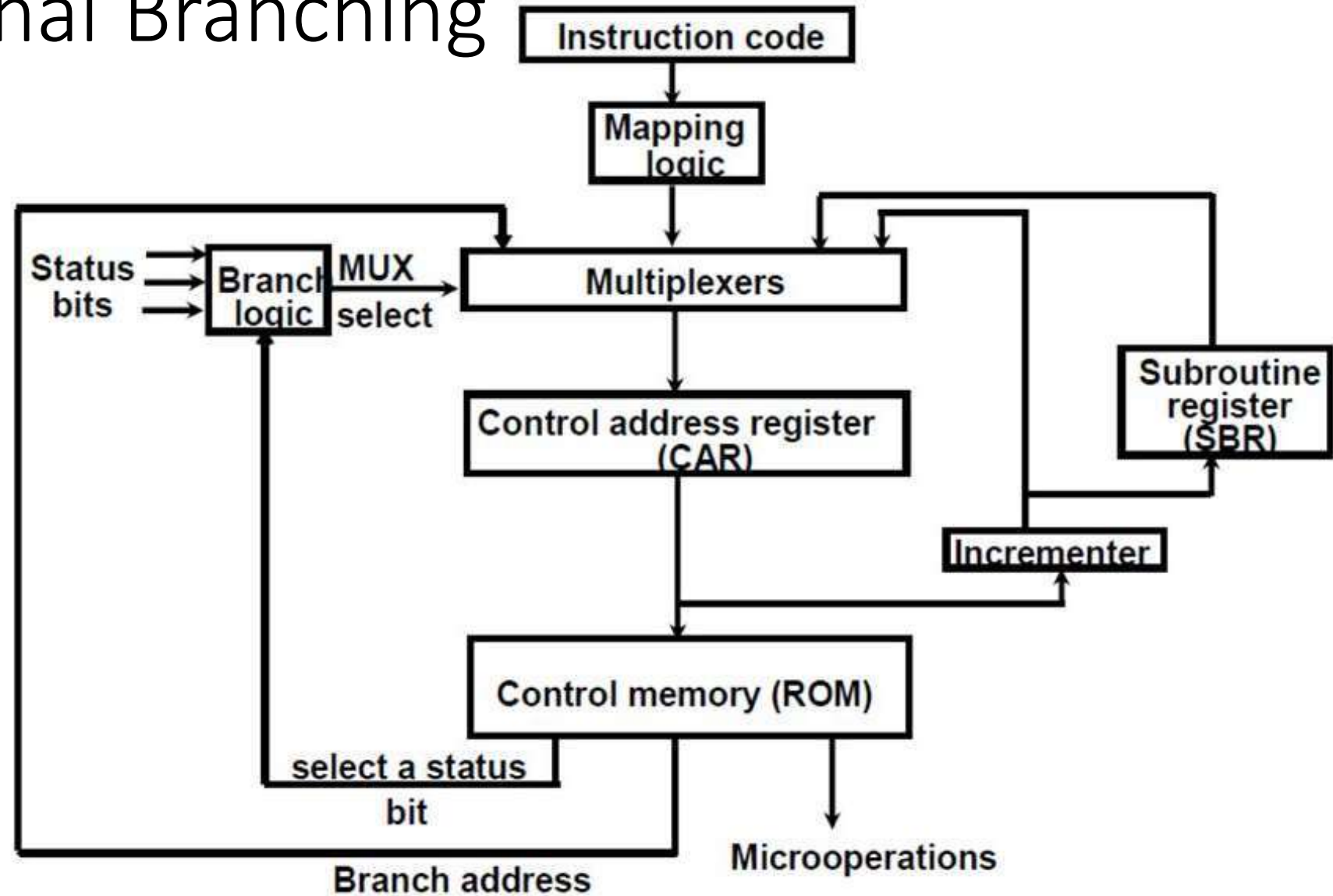
Conditional Branching

- Branch logic provides decision making capabilities in the CU.
- Status conditions are special bits in the system that provides parameter info such as the carry, sign, mode bit and i/o status.
- Information on the status condition can be tested and action can be initiated as per their values.
- The status bits together with the field in microinstruction that specify the branch address control the conditional branch decisions generated in branch logic.

Conditional Branching

- Simplest implementations a mux.
- If suppose there are 8 status bits then we can assign 3 bits to select the 8 status bits and feed it into the branch logic. The 1 in the status register outputs 1 from the branch logic which generates control signal to transfer the branch address from microinstructions into the CAR. A 0 in status register can get the CAR to increment.
- Unconditional can be just implemented by loading the branch address from the control memory to the CAR.

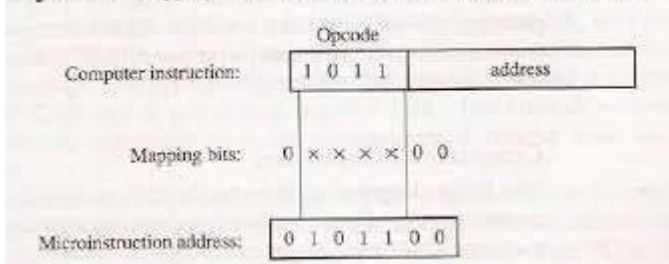
Conditional Branching



Mapping of Instruction

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine is located.
- The status bits for this type of branch are the bits in the opcode.
- Assume an opcode of four bits and a control memory of 128 locations. The mapping process converts the 4-bit opcode to a 7-bit address for control memory shown in below figure.

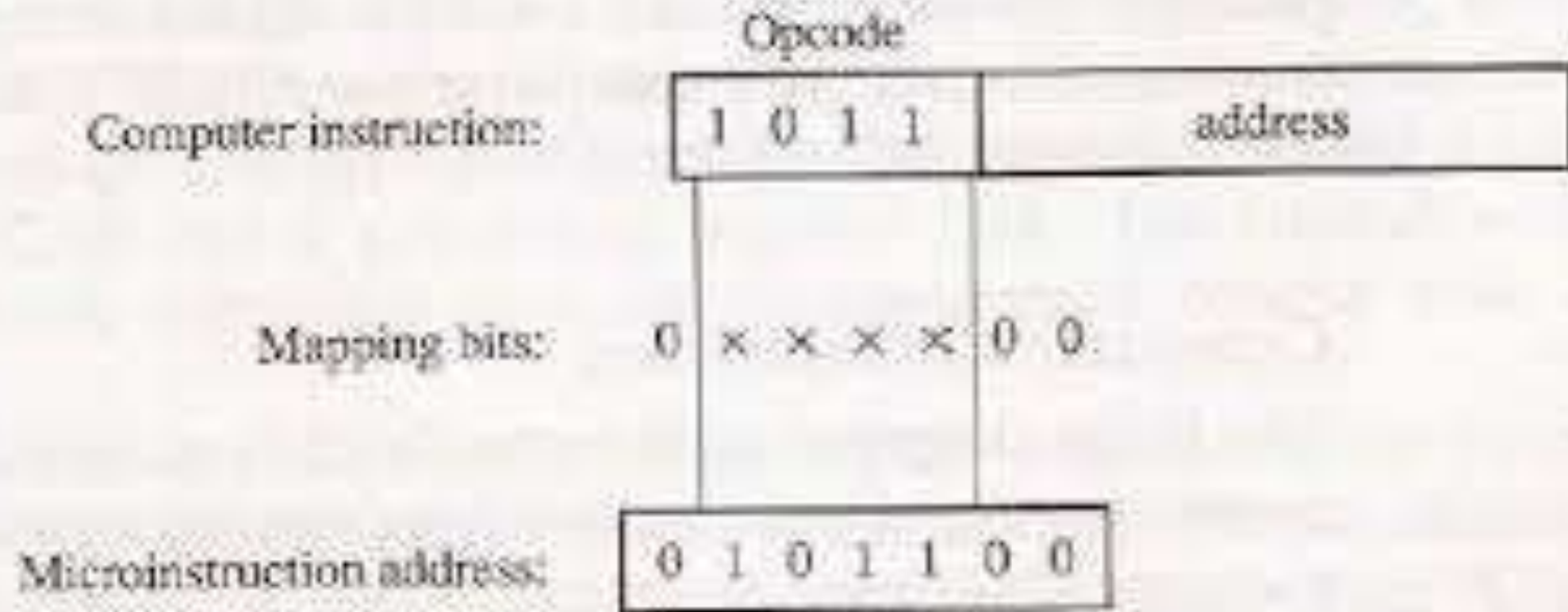
Figure 7-3 Mapping from instruction code to microinstruction address.



Put 0 in MSB and 00 in LSB

Mapping of Instruction

Figure 7-3 Mapping from instruction code to microinstruction address.



Mapping of Instruction

- Mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions

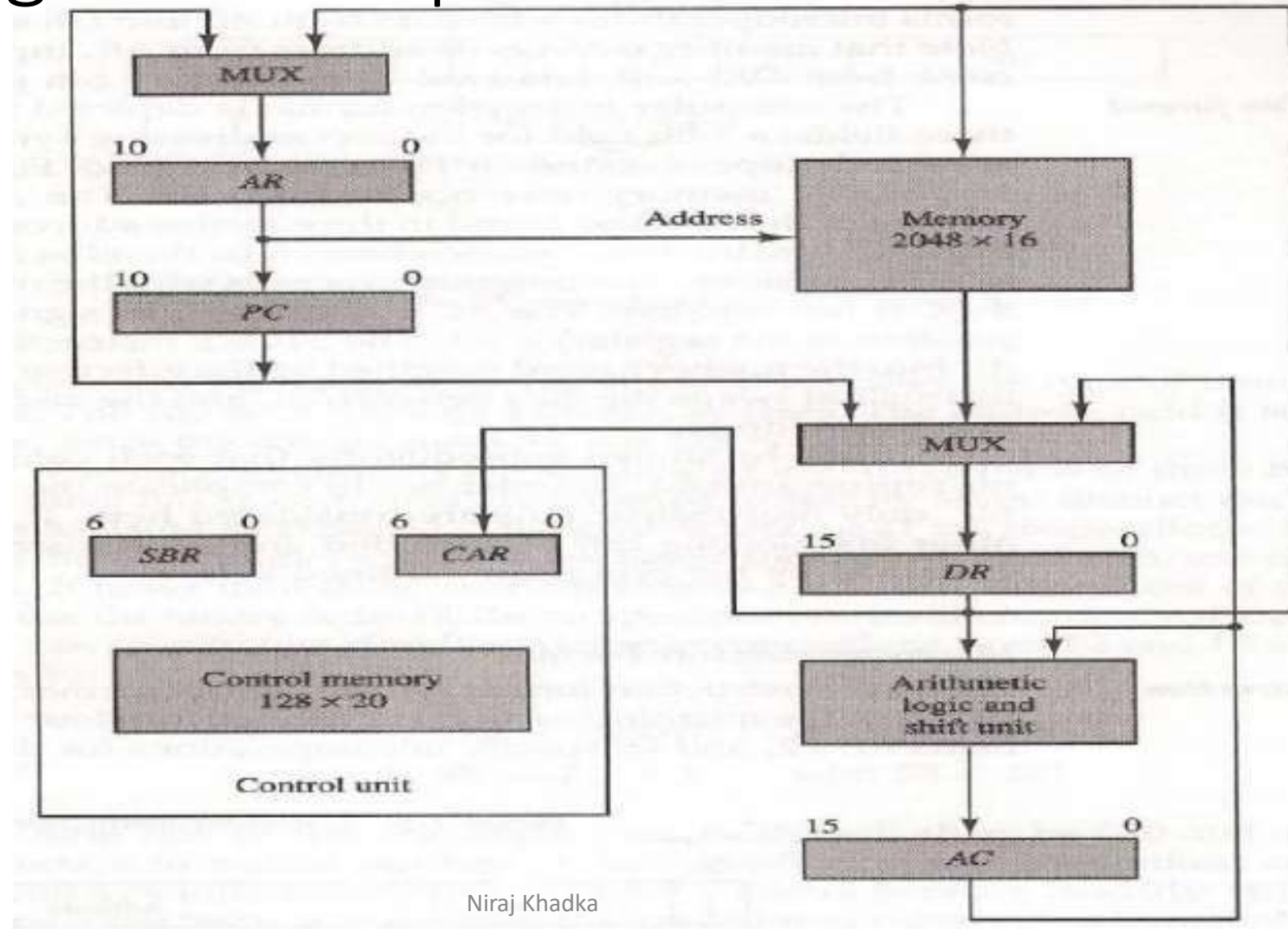
Subroutines

- Subroutines are programs that are used by other routines to accomplish a particular task and can be called from any point within the main body of the microprogram.
- Frequently many microprograms contain identical section of code.
- Microinstructions can be saved by employing subroutines that use common sections of microcode.
- Microprograms that use subroutines must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return.
- A subroutine register is used as the source and destination for the addresses

Microprogram Example

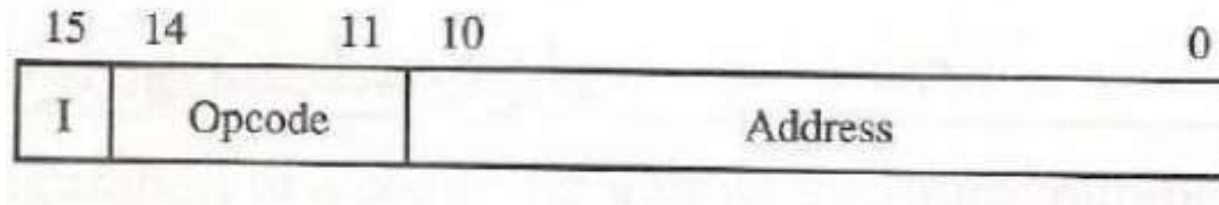
- The process of code generation for the control memory is called microprogramming.
- The block diagram of the computer configuration is shown in below figure.
- Two memory units:
 - Main memory – stores instructions and data
 - Control memory – stores microprogram
- Four processor registers
 - Program counter – PC
 - Address register – AR
 - Data register – DR
 - Accumulator register - AC
- Two control unit registers
 - Control address register – CAR
 - Subroutine register – SBR
- Transfer of information among registers in the processor is through MUXs rather than a bus.

Microprogram Example



Microprogram Example

- The computer instruction format is shown in below figure.



- Three fields for an instruction:
 - 1-bit field for indirect addressing
 - 4-bit opcode
 - 11-bit address field

Microprogram Example

- The example will only consider the following 4 of the possible 16 memory instructions.

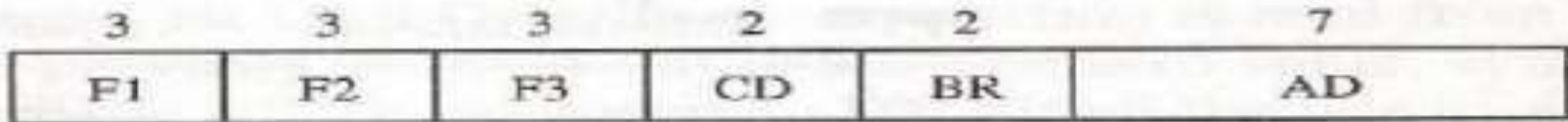
Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address

(b) Four computer instructions

Microprogram Example

- The microinstruction format for the control memory is shown in below figure.



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

Microprogram Example

- The microinstruction format is composed of 20 bits with four parts to it .
- Three fields F1, F2, and F3 specify microoperations for the computer [3 bits each]
 - The CD field selects status bit conditions [2 bits]
 - The BR field specifies the type of branch to be used [2 bits]
 - The AD field contains a branch address [7 bits]
- Each of the three microoperation fields can specify one of seven possibilities.
- No more than three microoperations can be chosen for a microinstruction.
- If fewer than three are needed, the code 000 = NOP.

Microprogram Example

- The three bits in each field are encoded to specify seven distinct microoperations listed in below table.

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Microprogram Example

- Five letters to specify a transfer-type microoperation
 - First two designate the source register
 - Third is a 'T'
 - Last two designate the destination register
 - $AC \leftarrow DR\ F1 = 100 = DRTAC$
- The condition field (CD) is two bits to specify four status bit conditions shown below

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

Microprogram Example

- The branch field (BR) consists of two bits and is used with the address field to choose the address of the next microinstruction.

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD, SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2-5) \leftarrow DR(11-14), CAR(0,1,6) \leftarrow 0$

Microprogram Example

- Each line of an assembly language microprogram defines a symbolic microinstruction and is divided into five parts
 - 1. The label field may be empty or it may specify a symbolic address. Terminate with a colon (:).
 - 2. The microoperations field consists of 1-3 symbols, separated by commas. Only one symbol from each field. If NOP, then translated to 9 zeros
 - 3. The condition field specifies one of the four conditions
 - 4. The branch field has one of the four branch symbols
 - 5. The address field has three formats
 - a. A symbolic address – must also be a label
 - b. The symbol NEXT to designate the next address in sequence
 - c. Empty if the branch field is RET or MAP and is converted to 7 zeros
- The symbol ORG defines the first address of a microprogram routine.
- ORG 64 – places first microinstruction at control memory 1000000.

Fetch Routine

- Fetch routine
 - Read instruction from memory
 - Decode instruction and update PC

Microinstructions for fetch routine:

$AR \leftarrow PC$
 $DR \leftarrow M[AR], PC \leftarrow PC + 1$
 $AR \leftarrow DR(0-10), CAR(2-5) \leftarrow DR(11-14), CAR(0,1,6) \leftarrow 0$

Symbolic microprogram for fetch routine:

FETCH: **ORG 64**
 PCTAR **U JMP NEXT**
 READ, INCPC **U JMP NEXT**
 DRTAR **U MAP**

Binary microporgram for fetch routine:

Binary address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

Symbolic Microprogram

- The symbolic microprogram is a convenient way of writing microprogram.
- It uses assembly like mnemonics which are easy to read and understand for people.
- It is not the way how a microprogram is stored in the control memory.
- The symbolic microprogram must be translated into binary form either by the assembler or by the user.

Binary Microprogram

- The representation of symbolic microprogram using string of 0's and 1's that are exactly according to the architecture of control unit design.
- The microprogram stored in the control memory as the group of microinstructions are in binary form which is called binary microprogram.

Design of Control Unit

- Why is control unit there in first place?
 - Control unit generates the timing and control signal for the operations of the computer.
 - Control unit communicates with the ALU and main memory.
 - Control unit also controls the data transmission between memory, processor and various other peripherals.
 - It also instructs ALU on what operation to perform.

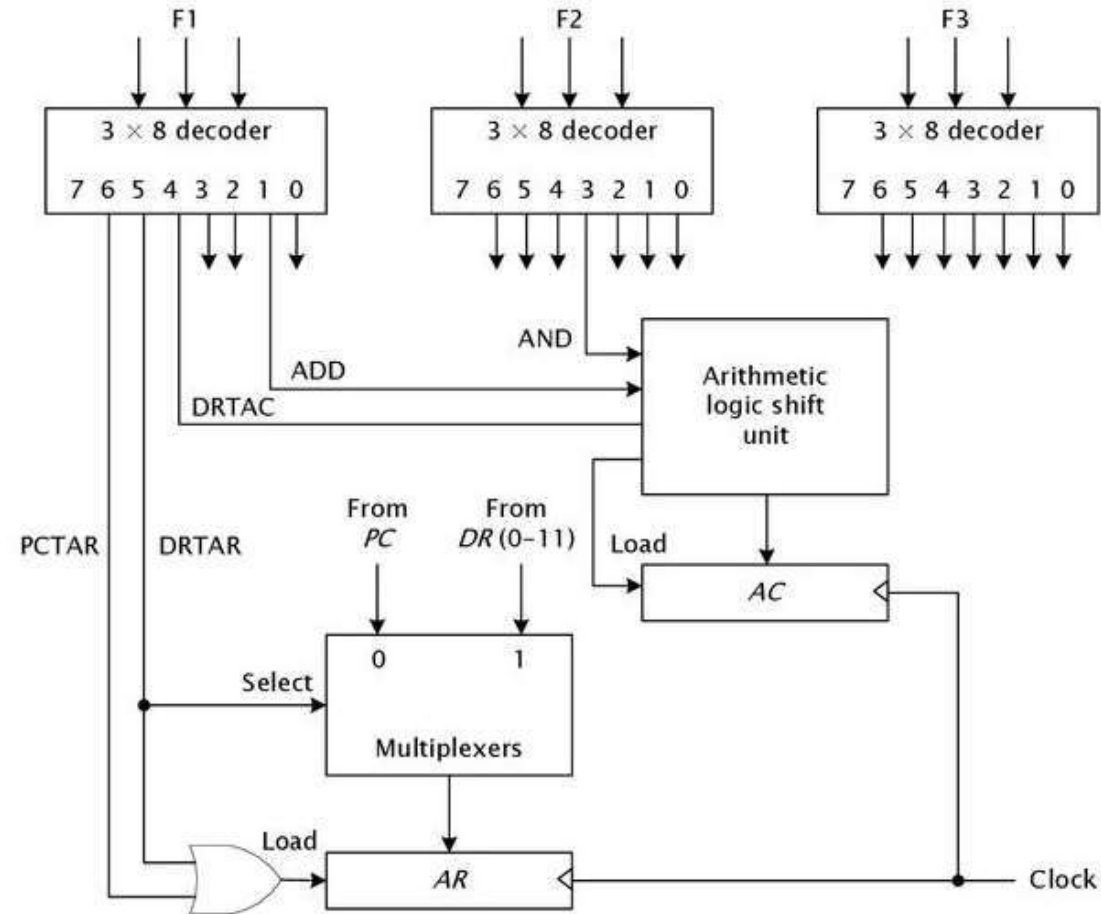
Design of Control Unit

- As from the previous example of a microprogram we found the microinstructions are divided in several fields.
- Each field has a separate and distinct function defined.
- For example to implement and design the three microoperations we can choose a decoder to decode the binary bits from F1, F2 and F3 to perform the related operations.
- The figure in next slide shows and explains how the CU is designed.

Design of Control Unit

Decoding of F fields

Fig. 7-7 Decoding of microoperation fields



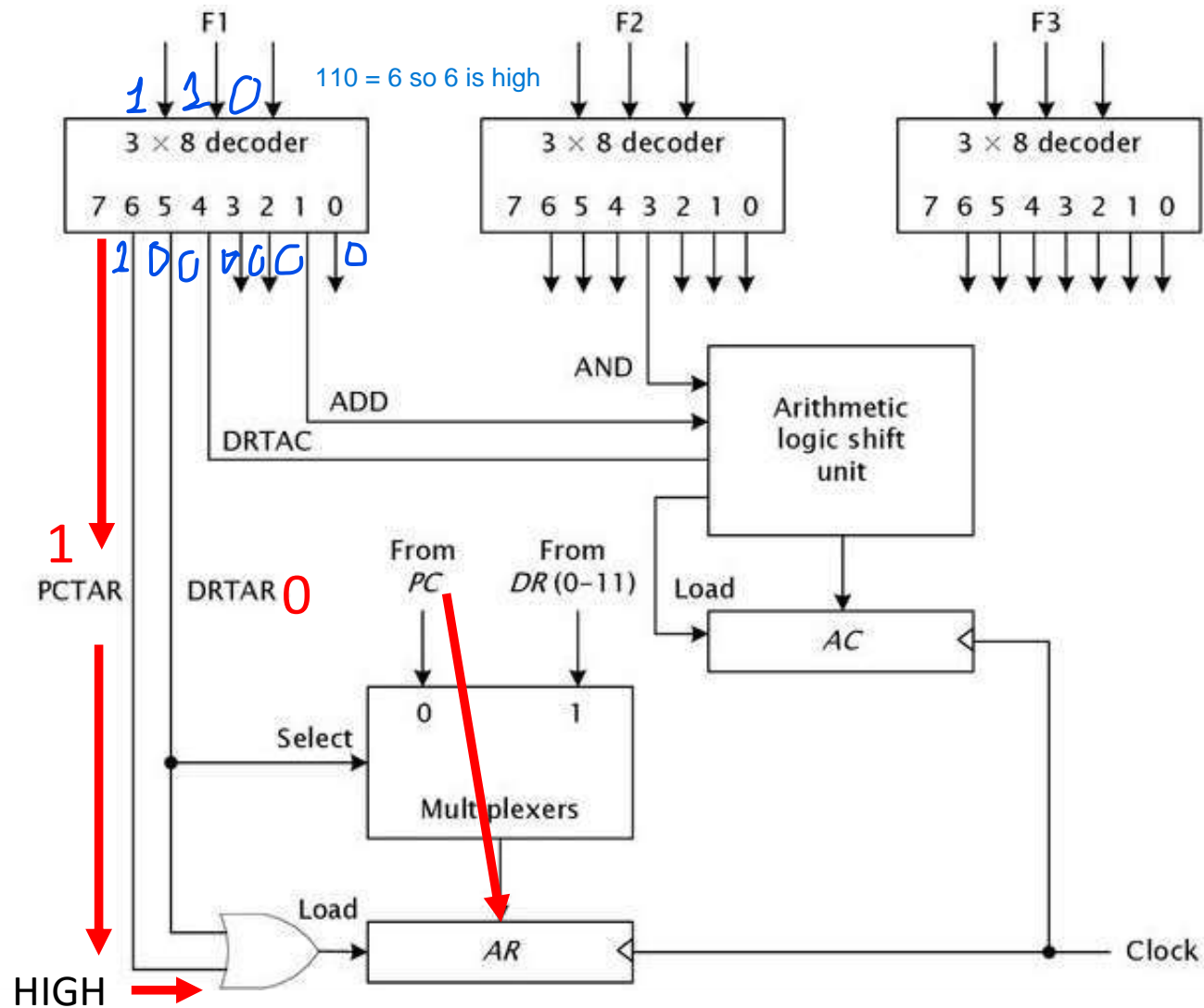
Design of Control Unit

- For reducing the complexity let us only look into the following microoperations only.
 - PCTAR
 - DRTAR
 - DRTAC
 - ADD
 - AND

Design of Control Unit

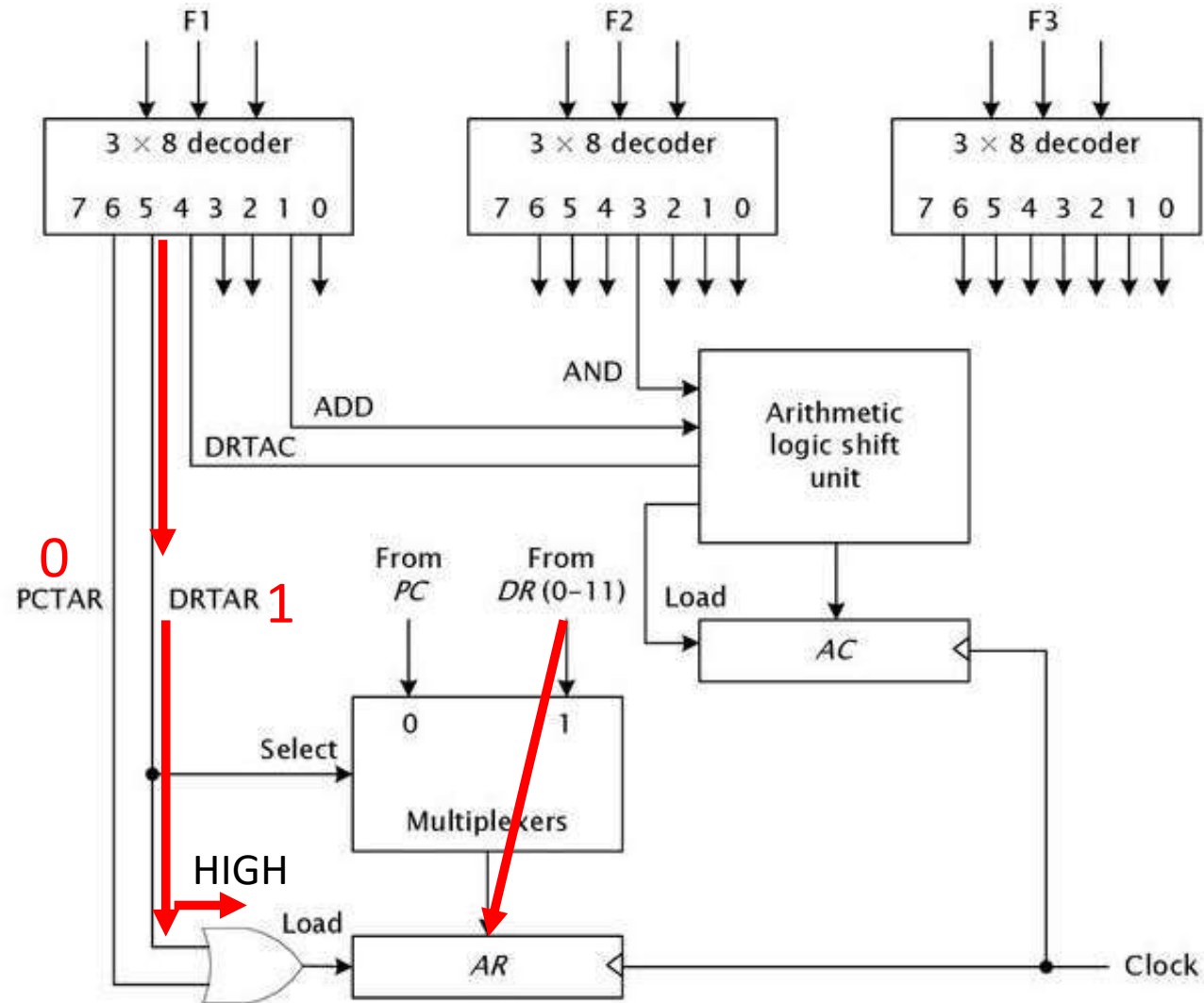
- PCTAR
 - Load the content of PC to AR.
 - The load line of the AR register is put high which is then loaded from the PC.
 - But for this the DRTAR line must be low which means the multiplexer that is connected to the AR is selected to be connected to PC.
- DRTAR
 - Load the content of DR(0-10) to AR.
 - The load line of the AR register is put high and the SELECT signal connected to DRTAR is high which results in loading the AR with selecting the DR(0-10) lines.

Design of Control Unit - PCTAR



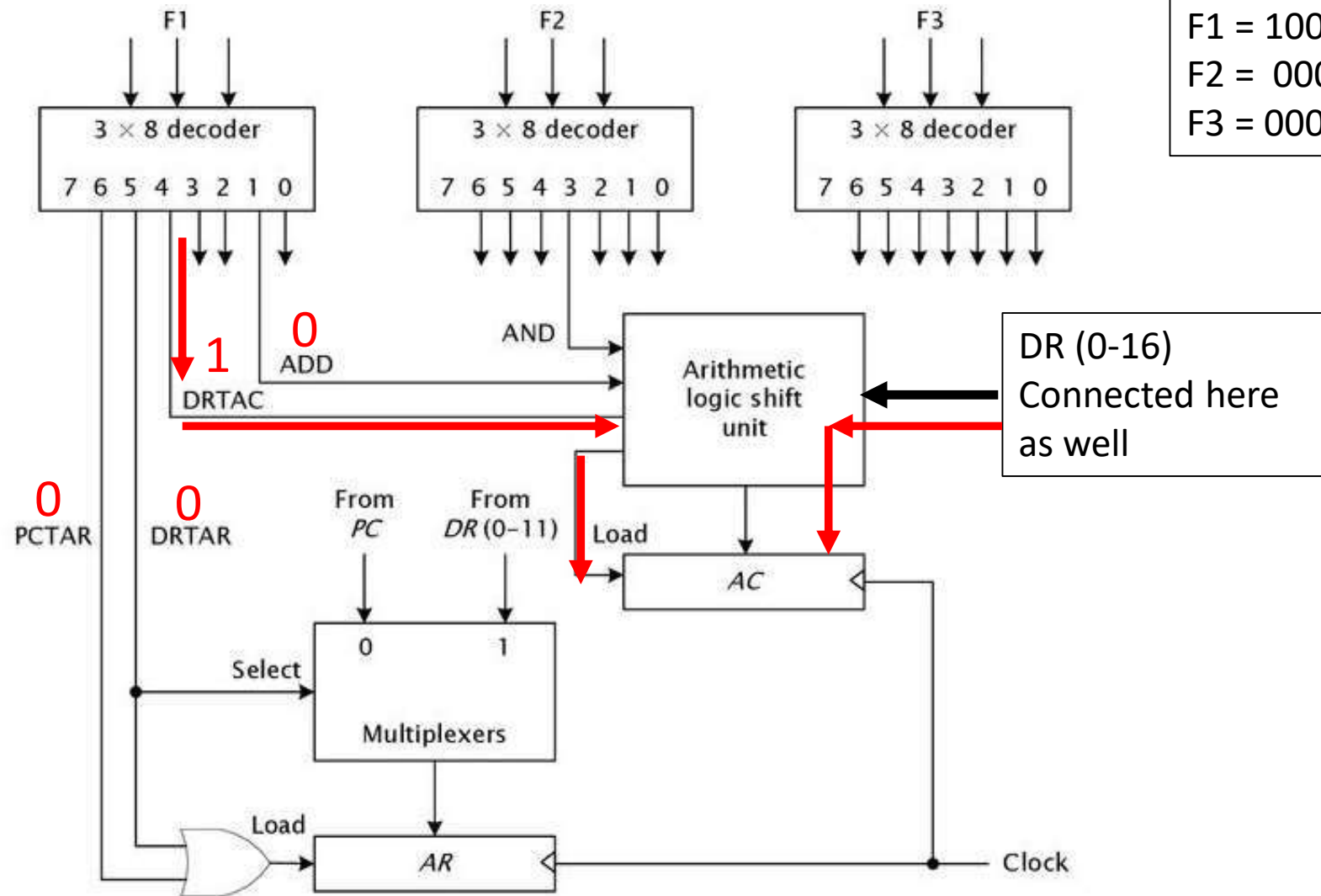
F1 = 110
F2 = 000
F3 = 000

Design of Control Unit - DRTAR

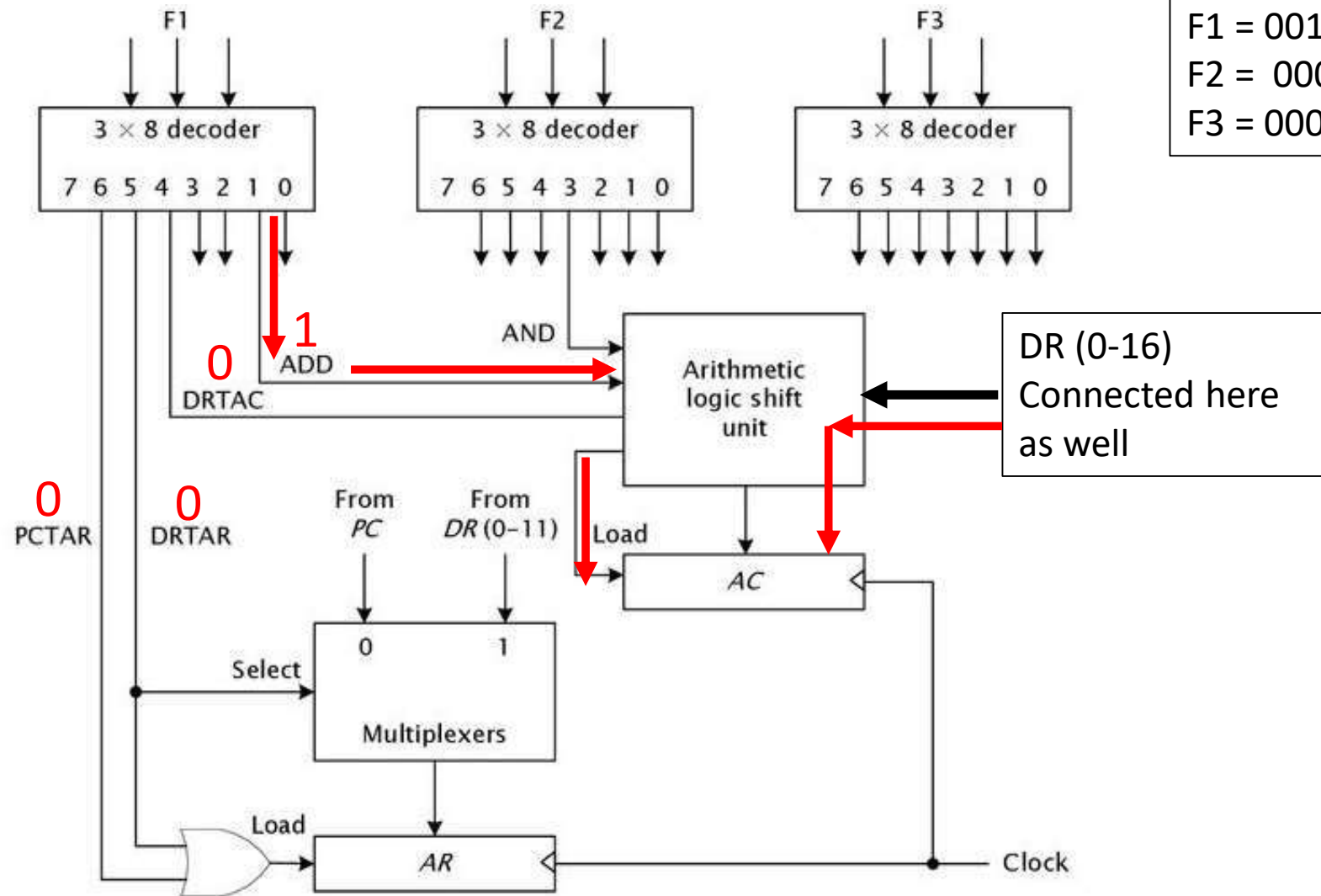


F1 = 101
F2 = 000
F3 = 000

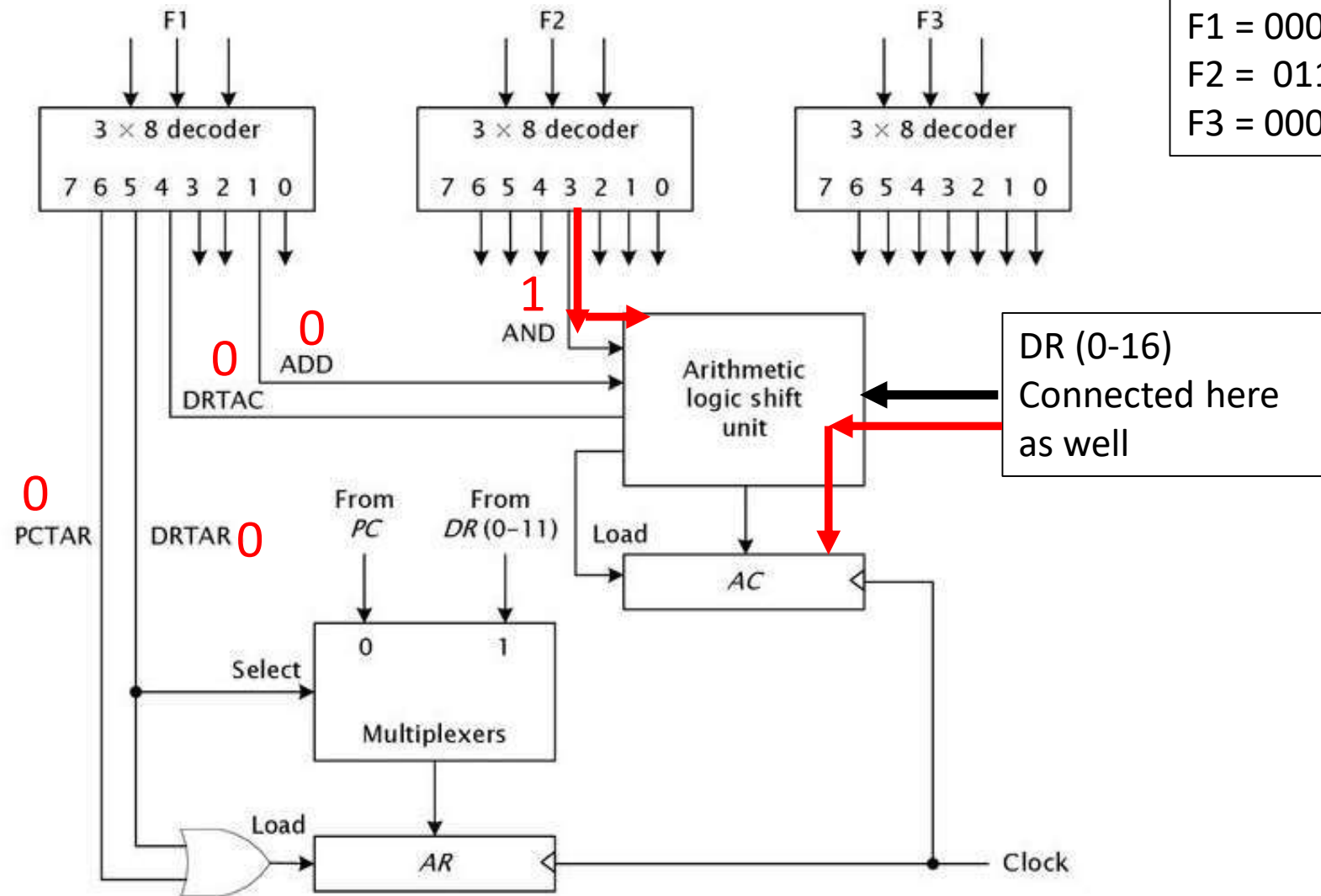
Design of Control Unit - DRTAC



Design of Control Unit - ADD



Design of Control Unit - AND



Basic Requirement of Control Unit

- Functional requirement of control unit are:
 - Define the basic elements of processor.
 - Describe the microoperations that the processor performs.
 - Determine the functions that the control unit must perform to cause the microoperations to be carried out.

Basic Elements of Processor

- ALU
- Registers

Types of microoperations

- Transfer data between registers.
- Transfer data from register to ~~external~~^{memory}.
- Transfer data from ~~external~~^{memory} to register.
- Perform Arithmetic and Logical operations.

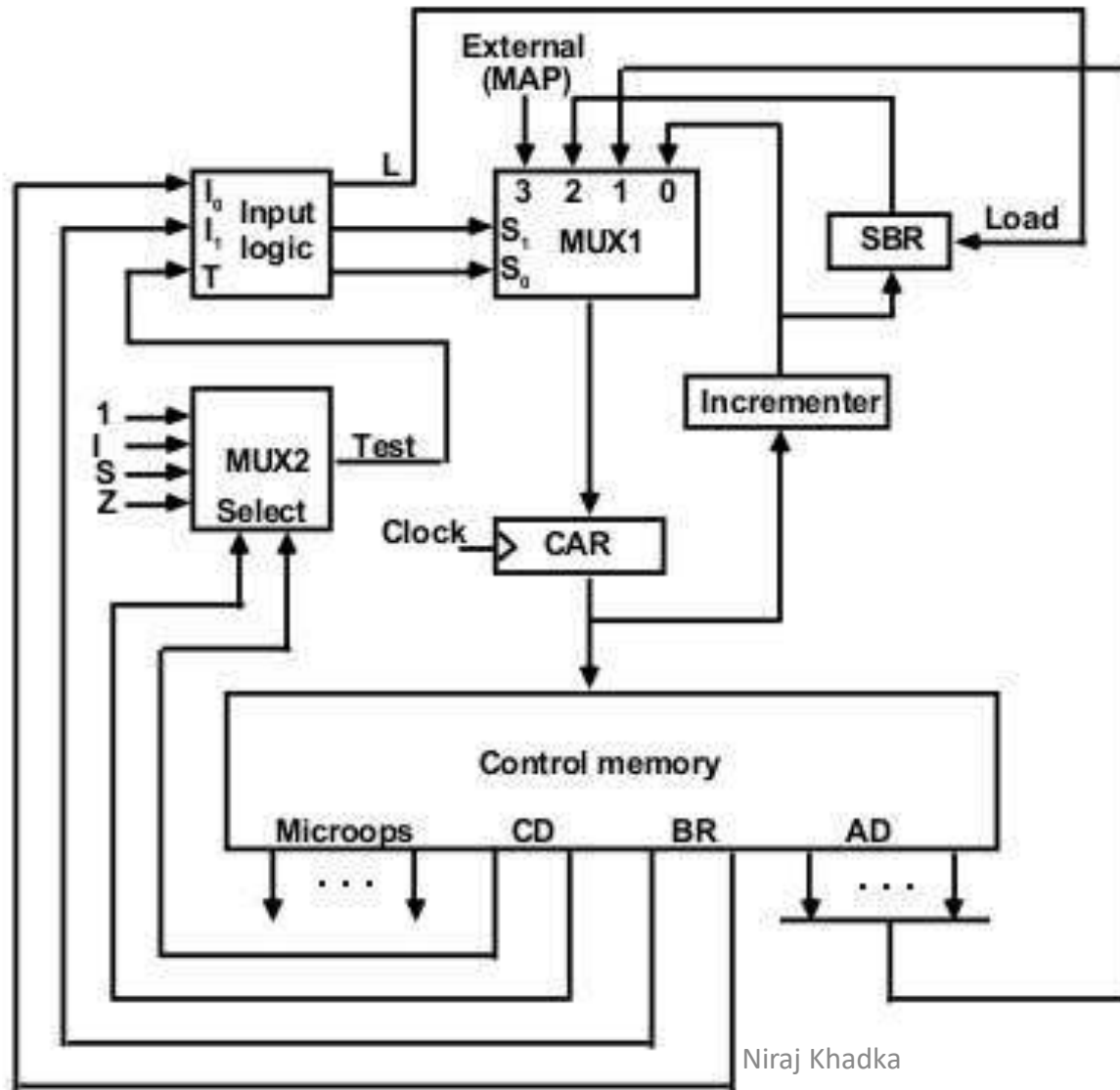
Functions of Control Unit

- Sequencing
 - The CU causes the CPU to step through a series of microoperations in proper order based on the program being executed.
- Execution
 - The CU causes each microoperations to be performed.

Microprogram Sequencer

- Basic component of microprogrammed control unit are
 - Control memory
 - The circuits that select the next address.
- The address selection part is called the microprogram sequencer.
- Purpose of the microprogram sequencer is to present an address to the control memory so the microinstructions may be read and executed.
- Let us look into a typical block diagram of a microprogram sequencer.

Microprogram Sequencer for Control Memory



Microprogram Sequencer

- Two multiplexers
 - MUX 1: selects an address from one of the four sources and route it into CAR.
 - MUX 2: tests the value of selected status bit and the result of test is applied to input logic circuit.
- Output from CAR provides the address for the control memory.
- Content of CAR is also incremented by 1 and applied to SBR as well as one of the inputs to MUX 1.
- The other three inputs in MUX 1 are:
 - From address field of current microinstruction.
 - External(MAP) source that maps the opcode.
 - From SBR register

Microprogram Sequencer

- Input Logic Circuit
 - Input logic in particular sequencer will determine the type of operations that are available.
 - Typical sequencer operations are:
 - Increment
 - Branch or jump
 - Call and Return from subroutine
 - Load an external address
 - Push or Pop on the stack.
 - With 3 inputs the ILC can provide up to eight address sequencing operation.
 - I_0 , I_1 , T are the 3 inputs to the Input Logic Circuit.

Microprogram Sequencer

- Input Logic Circuit

- S_0, S_1, L are the output from the Input Logic Circuit.
- S_0 and S_1 is to select the source for CAR from the available four.
- From figure we can get the following Boolean Table.

- $S_1 = I_1$
- $S_0 = I_1 I_0 + I_1' T$
- $L = I_1' I_0 T$

BR Field	Input			MUX 1		Load SBR
	I_1	I_0	T	S_1	S_0	L
00	0	0	0	0	0	0
00	0	0	1	0	1	0
01	0	1	0	0	0	0
01	0	1	1	0	1	1
10	1	0	x	1	0	0
11	1	1	x	1	1	0

Microprogram Sequencer

- The circuit from the above truth table can be constructed with three AND gates an OR gate and an Inverter.
- The incrementor circuit is also not a flip flop based but a combinational one made by using series of half adders.

Microprogrammed Control Unit

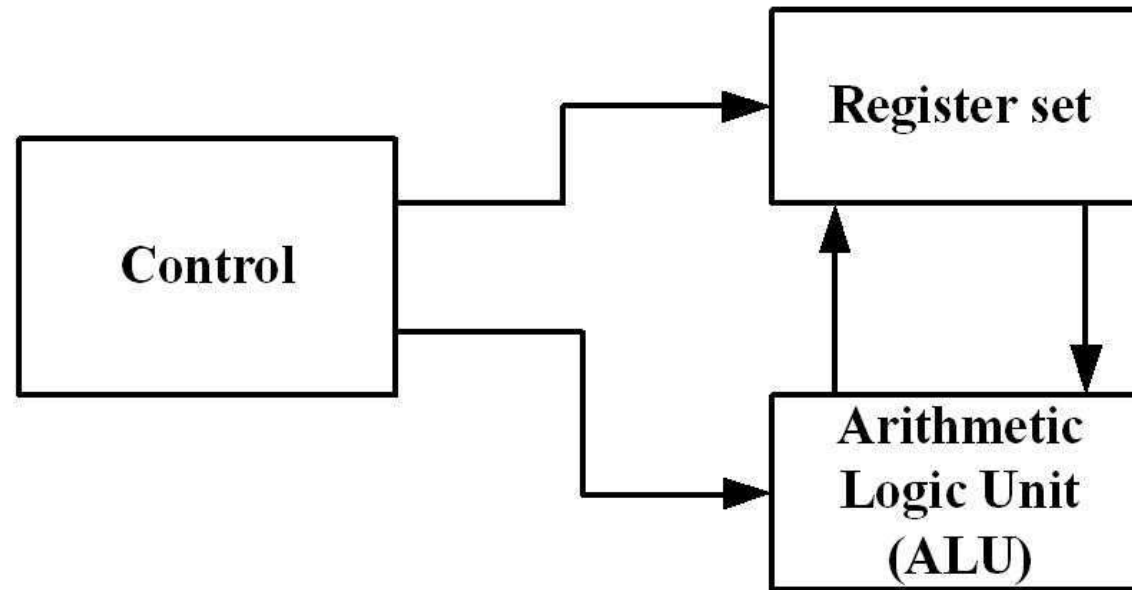
- Thank You.

Unit V : Central Processing Unit

- Part of computer that performs the bulk of data processing .
- Made up of 3 major parts:
 - Register Set
 - Stores intermediate data used during the execution of instructions.
 - ALU
 - Performs required microoperations for executing the instructions.
 - Control Unit
 - Supervises the transfer of information among the registers and instructs ALU on which operation to perform.

Major CPU Component

Central Processing Unit



Major components of CPU

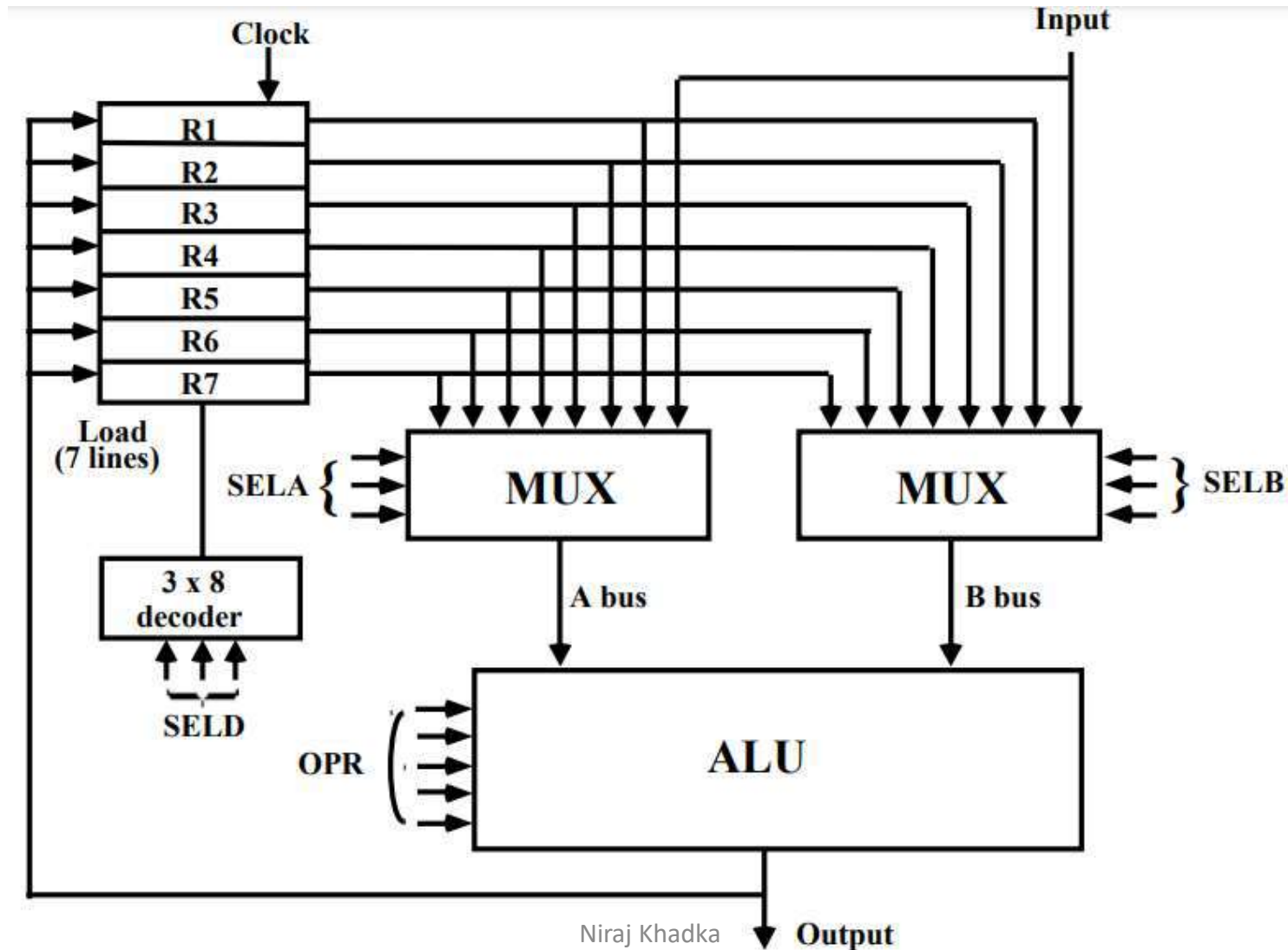
General Register Organization

- Whenever a large number of registers are included in CPU, it is efficient to connect them with common bus system.
- The registers communicate with each other not only for direct data transfer but also while performing various micro operations.
- Register Organization shows how registers are selected and how data flows between register and ALU.

General Register Organization

- A decoder is used to select a particular register. The output of each register is connected to two multiplexers to form the two buses A and B. The selection lines in each multiplexer select the input data for the particular bus.
- The A and B bus forms the two inputs for the ALU. The operation select lines decide the micro operation to be performed by ALU.
- The result of the micro operation is available at the output bus. The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

Register Set with Common ALU

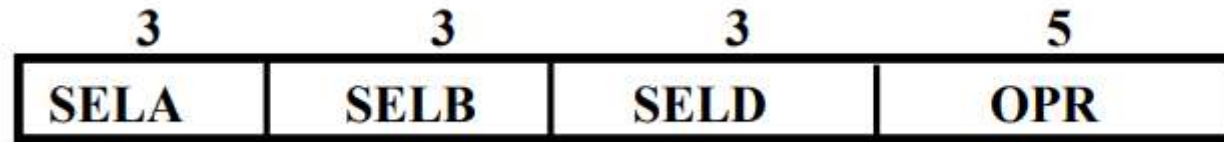


General Register Organization

- The control unit that operates the CPU bus system directs the information flow through the register and ALU by selecting the various components in the system. For example to perform the operation $R1 \leftarrow R2 + R3$
- The control unit must provide binary selection variables to the following selector inputs.
 1. MUX A Selector (SELA): to place the content of R2 into bus A.
 2. MUX B Selector (SELB): to place the content of R3 into bus B.
 3. ALU Operation Selector (OPR): to provide addition operation.
 4. Decoder Destination Selector (SELD): to transfer content of output bus into R1.

Control Word

- From the figure we know that there are 14 bit binary selection inputs, and their combined value specifies a control word.



- 3 bit of SELA selects the source register for A bus to ALU.
- 3 bit of SELB selects the source register for B bus to ALU.
- 3 bits of SELD selects the destination register from output of ALU.
- 4 bits of OPR selects the operation to be performed by ALU.

Encoding of Register Selection Field

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

Encoding of ALU OPR Field

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

Sample Micro Operations

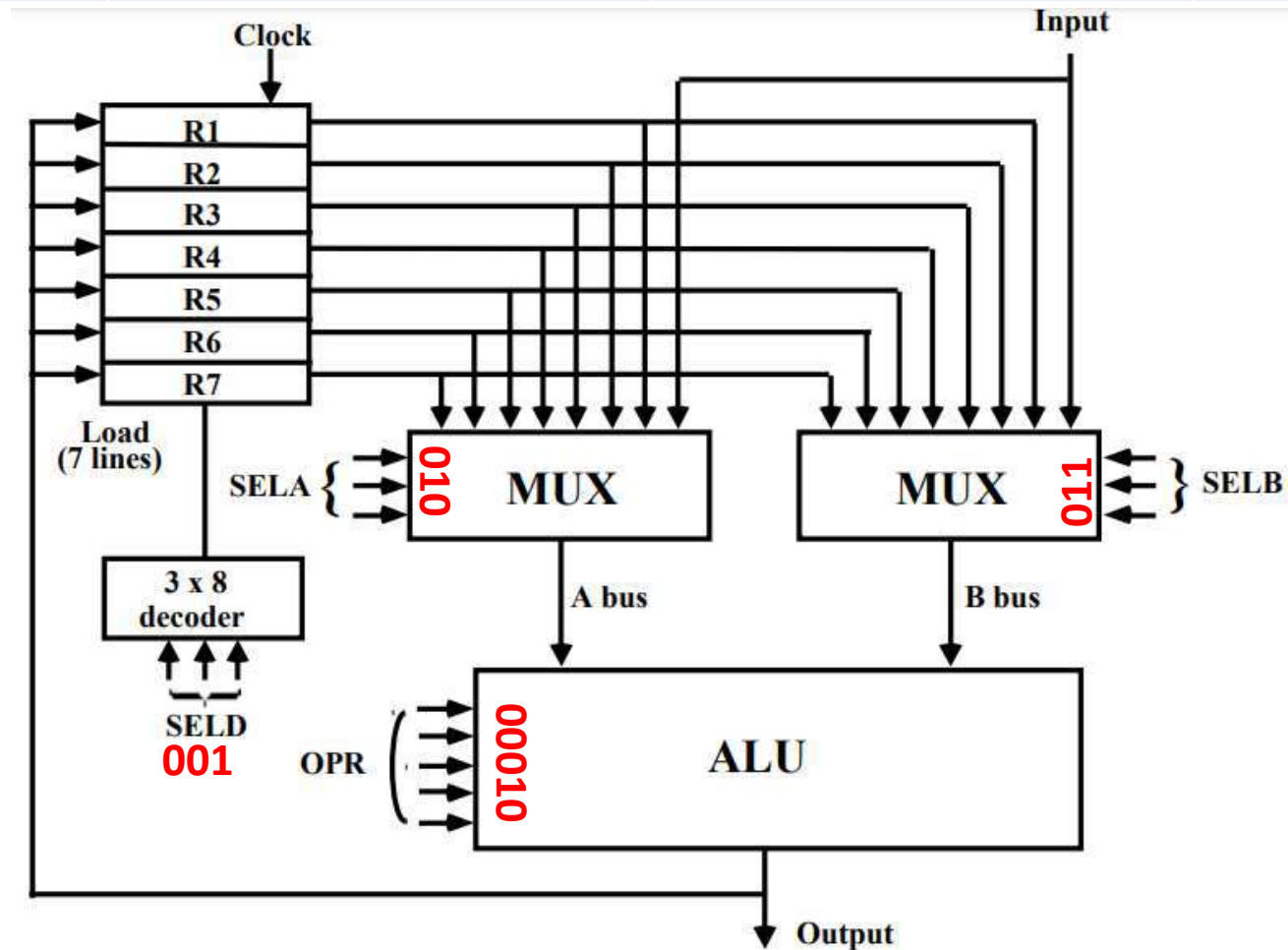
Microoperation	Symbolic Designation				Control Word
	SELA	SELB	SELD	OPR	
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \oplus R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	-	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	-	R7	TSFA	001 000 111 00000
Output $\leftarrow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

Ex: $R1 \leftarrow R2 + R3$

SELA (3)	SELB (3)	SELD (3)	OPR (3)
R2 = 010	R3 = 011	R1 = 001	ADD = 000100

Ex: $R1 \leftarrow R2 + R3$

SELA (3)	SELB (3)	SELD (3)	OPR (5)
R2 = 010	R3 = 011	R1 = 001	ADD = 00010

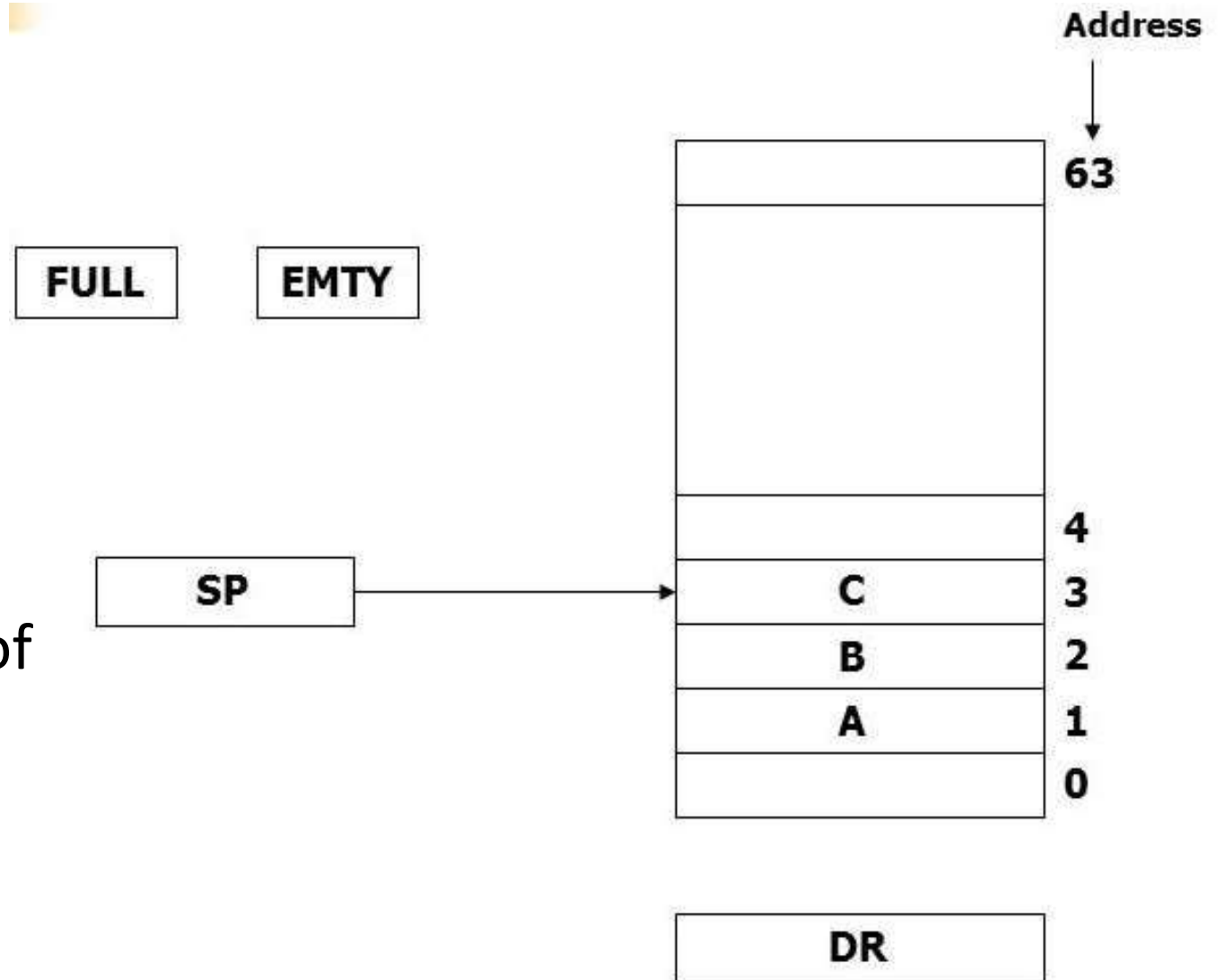


Stack Organization

- Very useful feature for nested subroutines, nested loops control.
- Also efficient for arithmetic expression evaluation.
- Storage which can be accessed in LIFO.
- Pointer: SP
- Only PUSH and POP operations are applicable.

Register Stack

- A stack can be placed in a portion of large memory or it can be organized as a collection of finite number of memory words or registers.
- SP contains the binary value of the address of word that is currently on top of the stack.



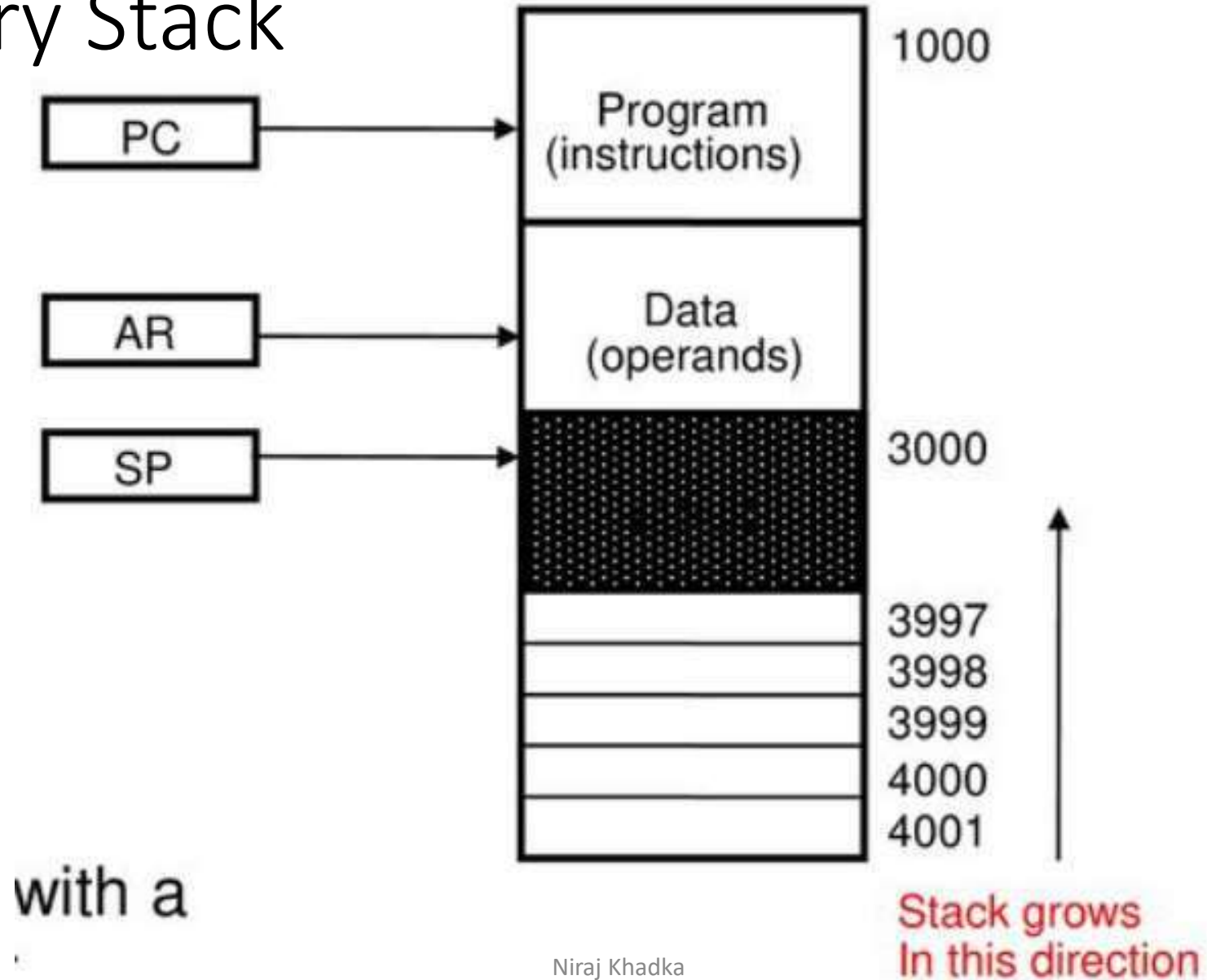
Register Stack

- Initially SP is cleared to 0, EMPT is set to 1, and FULL is cleared to 0, so SP points to the word at memory location 0.
- When a PUSH operation is done, it is implemented as:
 - $SP \leftarrow SP + 1$ Increment the Stack Pointer.
 - $M[SP] \leftarrow DR$ Write item on top of the stack from DATA Register
- When a POP operation is done, it is implemented as:
 - $DR \leftarrow M[SP]$
 - $SP \leftarrow SP - 1$

Memory Stack

- A stack kind of data structure can be implemented in a random access memory connected to CPU.
- Implementation of stack is done by assigning a portion of memory to the stack operation and then using processor register as a stack pointer.
- Computer memory is partitioned into three segments:
 - Program
 - Data
 - Stack
- The PC points at the address of next instruction in the program.
- The AR points to an array of data in memory.
- SP points to the top of the stack.

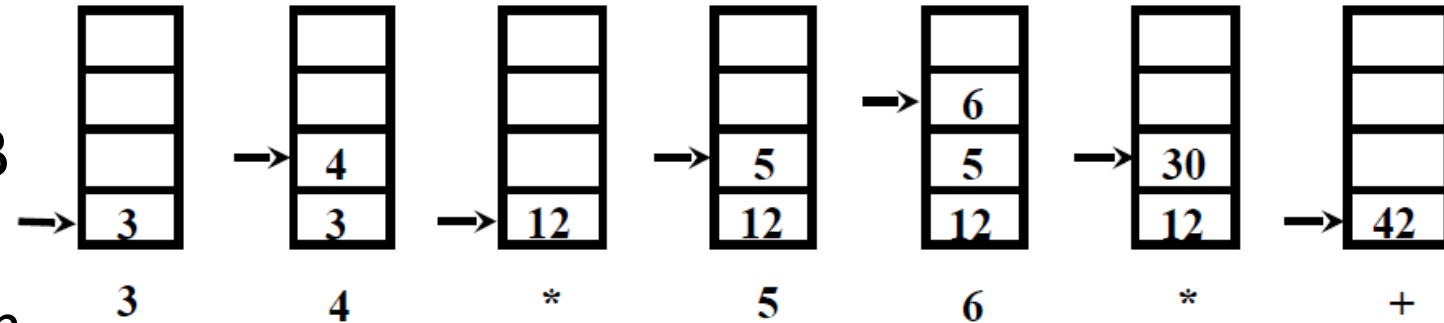
Memory Stack



Reverse Polish Notation

- Arithmetic Expression: $A + B$

- $A + B$ Infix notation
- $+ A B$ Prefix or Polish notation
- $A B +$ Postfix or reverse Polish notation
- - The reverse Polish notation is very suitable for stack manipulation



- Evaluation of Arithmetic Expressions

- Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation
- $(3 * 4) + (5 * 6) \Rightarrow 3 \ 4 \ * \ 5 \ 6 \ * \ +$

Program Interrupt

- Program interrupt refers to transfer of control from the currently running program to another service program because of internal or external generated request.
- Control returns to the original program after the service program is executed.
- Interrupt procedure is similar to procedure call except for the below:
 - Interrupt is usually initiated by an internal or external signal rather than from the execution of instruction (except for software interrupt).
 - Address of the interrupt service program is determined by the hardware rather than from the address field of an instruction.
 - Interrupt procedure usually stores all the information necessary to define the state of CPU rather than storing only the program counter.

Types of Interrupts

- External Interrupt
- Internal Interrupt
- Software Interrupt

External Interrupts

- Comes from I/O devices or timing devices or circuit monitoring and power supply or from any other external sources.
- For eg: if a device on a system connected to a port wants to initialize a data transfer it must let the CPU know that it wants to initiate the transfer and then get the permission as well as resource from the CPU. For that the external device must generate an interrupt signal, which is generally connected to the interrupt pin of the CPU.
- Timeout interrupt is generated from hanging inside of an infinite loop while unable to process anything. These kind of interrupt can then help CPU to recover from infinite deadlock situations.

Internal Interrupts

- Arises from illegal or erroneous use of instruction or data.
- Often called TRAPS
- EG: Register Overflow, divide by zero attempt, invalid opcode, stack overflow, general protection violation etc.
- These error condition usually occurs as a result of premature termination of the instruction execution.
- The service program that handles the internal interrupt determines the corrective measures and takes it.

Software Interrupts

- Software interrupt is generated by the program instruction explicitly.
- It is a special call instruction that behaves like the interrupt rather than a subroutine call.
- Software interrupts are used to access certain features of the device connected, change the modes of CPU to access supervisory mode from user mode and vice versa.
- A program written by user must be only run in user mode. So if the program requires special service and features from underlying software or hardware the user program can call the software interrupt to access those services in supervisory mode.

H/W

- Write down the difference between External interrupts and Internal interrupts.
- Define stack. Explain the stack organization.
- Explain the general register organization.

Instruction Format

- Four types of instruction format for a simple CPU.
 - Three Address Instructions
 - Two Address Instructions
 - One Address Instructions
 - Zero Address Instructions
- For the ease of understanding let us take an arithmetic expression $X = (A+B)*(C+D)$
 - We will use ADD, SUB, MUL and DIV for arithmetic operations.
 - MOV for transfer type instructions.
 - LOAD and STORE for transfer from and to memory and AC register.
 - We assume operands are in memory address A, B, C, D, and result is stored in address X.

Three Address Instructions

Opcode	Destination	Source 1	Source 2
--------	-------------	----------	----------

3-address Instruction Format

ADD	R1, A, B	$R1 \leftarrow M[A] + M[B]$
ADD	R2, C, D	$R2 \leftarrow M[C] + M[D]$
MUL	X, R1, R2	$M[X] \leftarrow R1 * R2$

- Assumed the CPU has two process register R1 and R2.
- Benefit of 3 address instructions is that it takes few number of instructions to carry out the task.
- Disadvantage is that, it takes too many bits for the binary coded instructions.

Two Address Instructions

opcode	Destination address	Source address
--------	---------------------	----------------

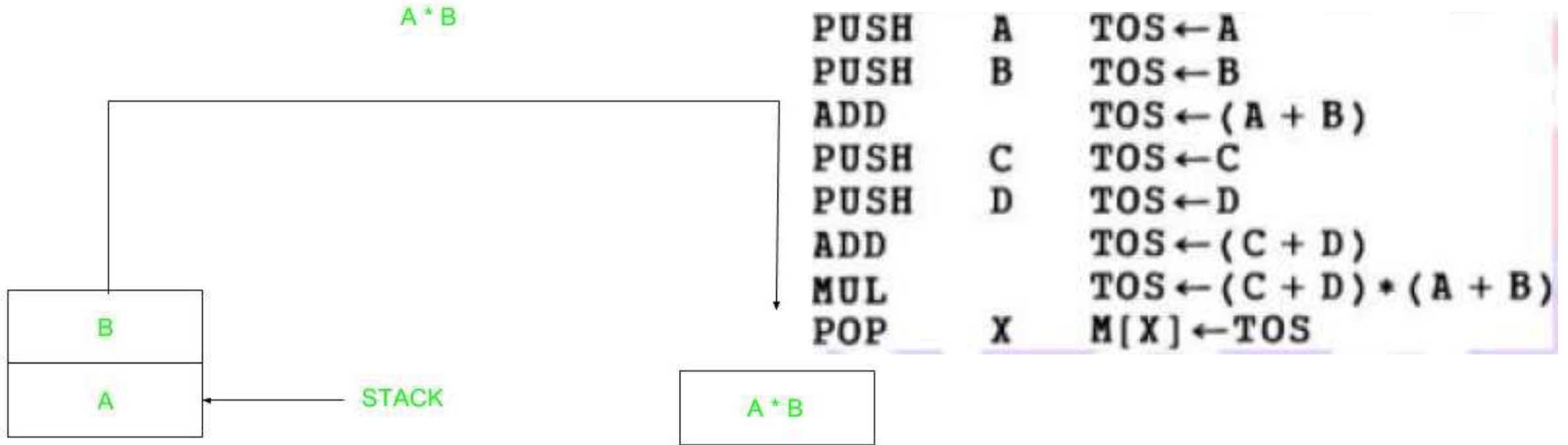
MOV	R1, A	$R1 \leftarrow M[A]$
ADD	R1, B	$R1 \leftarrow R1 + M[B]$
MOV	R2, C	$R2 \leftarrow M[C]$
ADD	R2, D	$R2 \leftarrow R2 + M[D]$
MUL	R1, R2	$R1 \leftarrow R1 * R2$
MOV	X, R1	$M[X] \leftarrow R1$

One Address Instructions

opcode	operand/address of operand
--------	----------------------------

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

Zero Address Instructions



PUSH A
PUSH B

Instructions in RISC Machines

- The ISA is restricted to the use of LOAD and STORE instruction when communicating with memory.
- All other instructions are executed within the registers without referring to memory.

LOAD	R1, A	$R1 \leftarrow M[A]$
LOAD	R2, B	$R2 \leftarrow M[B]$
LOAD	R3, C	$R3 \leftarrow M[C]$
LOAD	R4, D	$R4 \leftarrow M[D]$
ADD	R1, R1, R2	$R1 \leftarrow R1 + R2$
ADD	R3, R3, R2	$R3 \leftarrow R3 + R2$
MUL	R1, R1, R3	$R1 \leftarrow R1 * R3$
STORE	X, R1	$M[X] \leftarrow R1$

Addressing Modes

- The instruction format of any microprocessor consists of opcode, mode field and some address field.



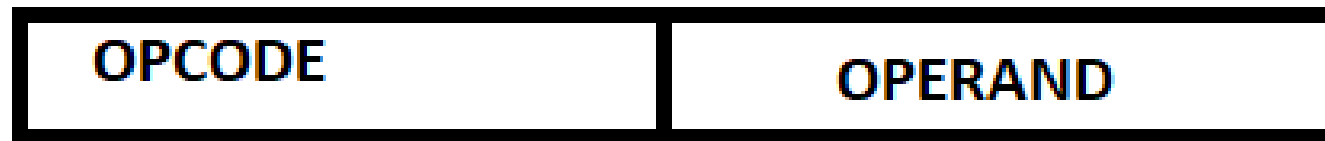
Implied Addressing Mode



- No operand(register or memory location or data) is specified in the instruction.
- Operand is specified implicitly in the definition of the instruction itself.
- EG: CMA, RLC, RRC
- All register reference instructions that use an accumulator only are implied mode instructions.
- Zero address instructions in the stack organized computer are implied mode instructions since the operand are implied to be on the top of the stack.

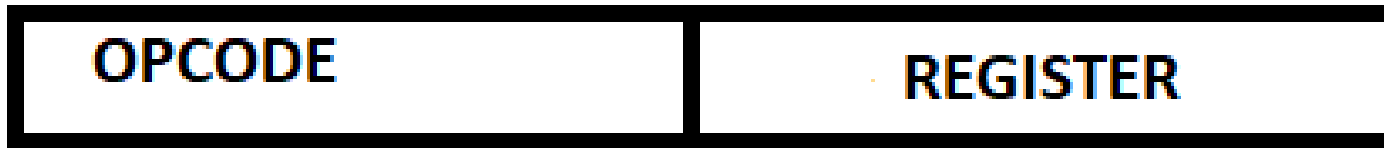
Immediate Addressing Mode

- The operand is specified in the instruction as well.
- Has an operand field rather than the address field.
- Operand field contains actual operand.
- Used to initialize the registers with actual data values.



Register Addressing Mode

- This mode is similar to the immediate addressing mode but instead of immediate data there is a register involved.
- The particular register is selected from the register field within an instruction.



Register Indirect Mode

- In this mode the instruction specifies the register in the CPU whose contents give the address of the operand in memory.
- That is the selected register contains the address of operand rather than the operand itself.
- EG: MOV A, M
 - Here M specifies the memory address by HL pair.
 - The real operand is in the address pointed by the contents of HL pair.

Auto Update Mode

- Similar to Register indirect mode except that the register is incremented or decremented after or before its value is used to access memory.
- After accessing the operand the contents of the register are automatically incremented to the next value.
- Before accessing the operand, the contents of the register are automatically decremented to the next value.

Direct Address Mode

- Operand resides in the memory.
- Address is given directly by the address field.
- If instruction is of branch type, the address field specifies the actual branch address.

Indirect Address Mode

- The address field of the instruction gives the address of the memory where the effective address is stored.
- Control fetches the content from memory which is a address, and then again fetches the content from that address.

Relative Address Mode

- The content of PC is added to the address part of instruction in order to obtain the effective address.
- $EA = PC + \text{offset}$
- Address part of instruction is usually a signed number (in 2's complement form).

Indexed Addressing Mode

- Content of index register is added to address part of instruction to obtain the effective address.
- Index register is a special CPU register that contains the index value.
- The address field defines the starting address of a data array in memory.
- Each operand in array is stored in memory relative to the beginning address.
- The distance between the beginning address and the address field of the operand is the index value stored in the index register.

Base Register Addressing Mode

- The content of base register is added to the address part of the instruction to obtain the Effective address.
- Similar to indexed addressing mode except the register is a base register instead of index register.
- Difference between the indexed mode and base register mode is in the use rather than the calculation of EA.
- Index register holds index number while the base register holds the base address.

Data Transfer and Manipulation

- Most computer instructions can be classified into three categories.
- 1. Data Transfer Instructions
- 2. Data Manipulation Instructions
- 3. Program Control Instructions

Data Transfer Instructions

- Moves data from one place to another be it, memory to memory, memory to register, register to memory or register to register.

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD \$ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$
Autodecrement	LD -(R1)	$R1 \leftarrow R1 - 1, AC \leftarrow M[R1]$

Data Manipulation Instructions

- Performs operation on data and provide computational capabilities for computer.
- Are basically divided into three types.
- i) Arithmetic Instructions
- ii) Logical and Bit Manipulation Instructions
- iii) Shift Instructions

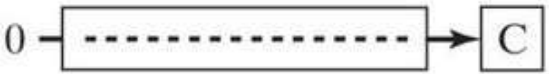
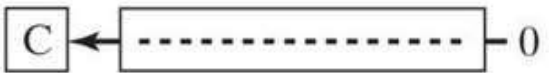
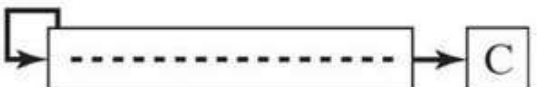

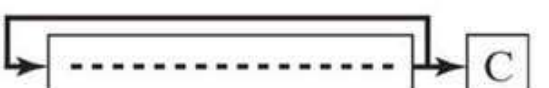
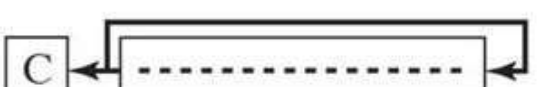
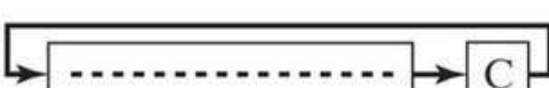
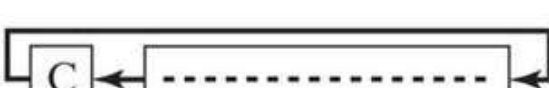
Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Subtract reverse	SUBR
Negate	NEG

Logical and Bit Manipulation Instructions

Name	Mnemonic
CLEAR	CLR
COMPLEMENT	COM
AND	AND
OR	OR
EXCLUSIVE OR	XOR
CLEAR CARRY	CLRC
SET CARRY	SETC
COMPLEMENT CARRY	COMC
ENABLE INTERRUPT	EI
DISABLE INTERRUPT	DI

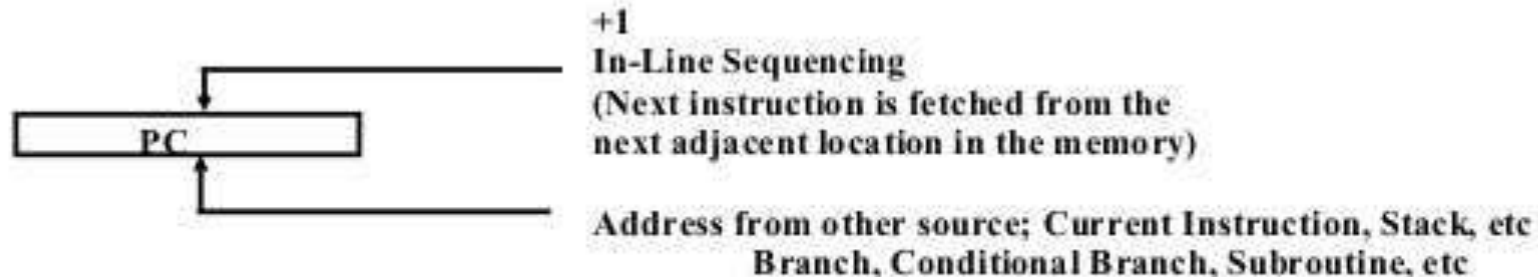
Shift Instructions

Name	Mnemonic	Diagram
Logical shift right	SHR	
Logical shift left	SHL	
Arithmetic shift right	SHRA	
Arithmetic shift left	SHLA	
Rotate right	ROR	
Rotate left	ROL	
Rotate right with carry	RORC	
Rotate left with carry	ROLC	

Program Control Instructions

- Specifies condition for altering the content of PC.
- The change in the value of PC causes break in the sequential execution of instructions and causes the program to jump to some other location and execute from there.
- This is an important feature in digital computer, as it provides control over the flow of program execution and a capability of branching to different program segment.

PROGRAM CONTROL INSTRUCTIONS

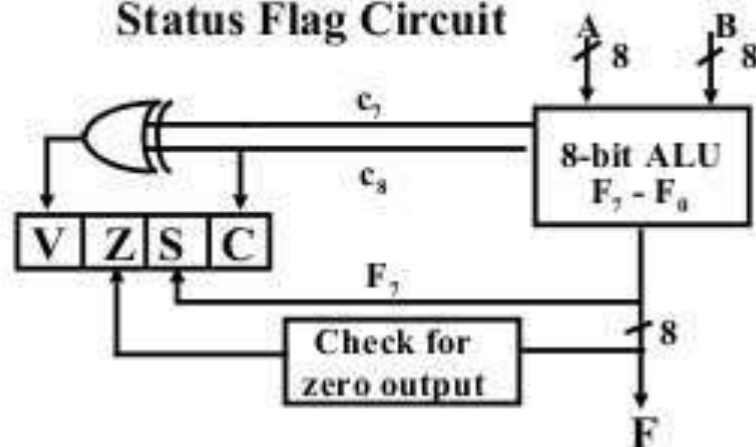


Program Control Instructions

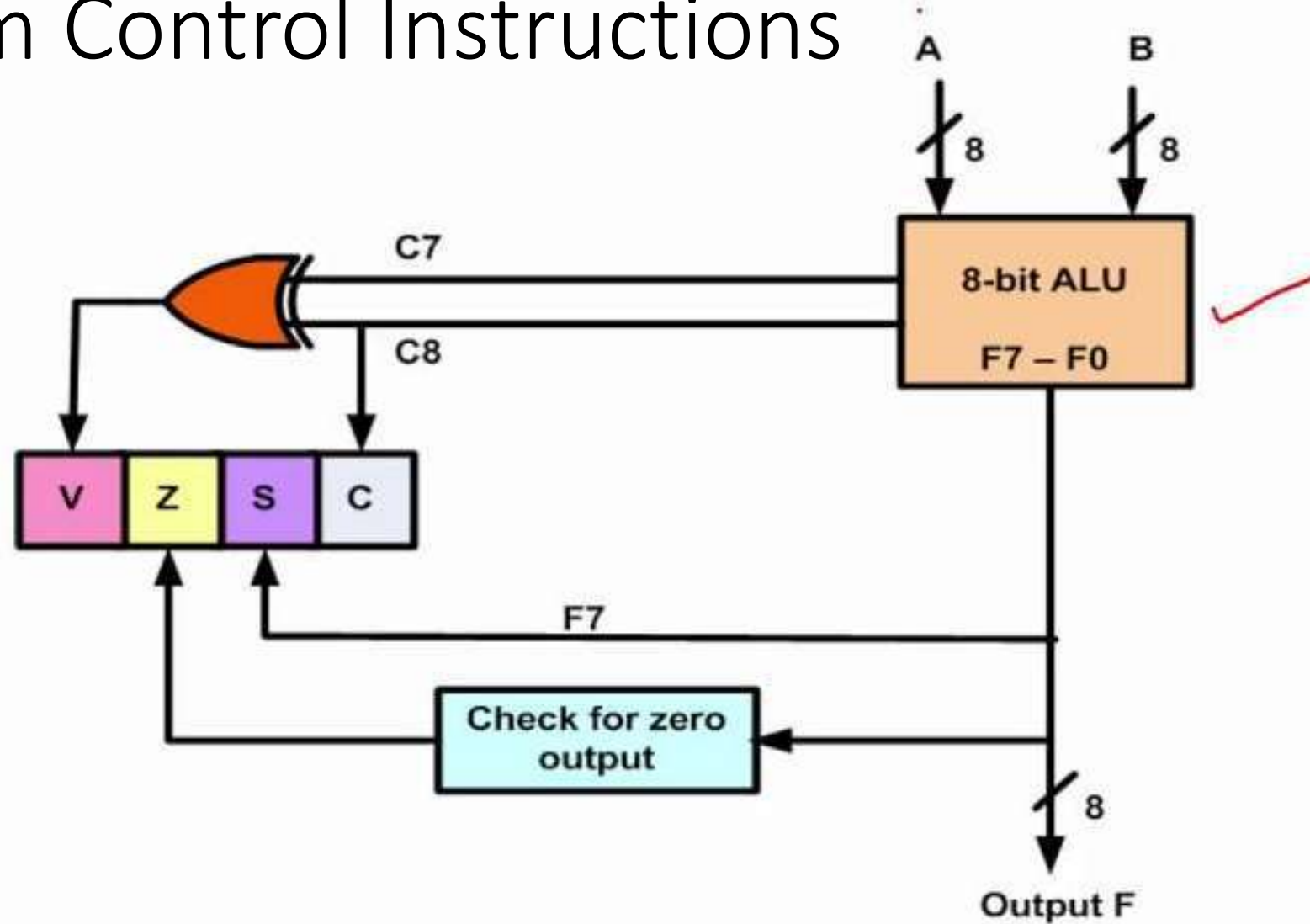
Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by -)	CMP
Test (by AND)	TST

* CMP and TST instructions do not retain their results of operations(- and AND, respectively). They only set or clear certain Flags.

Status Flag Circuit



Program Control Instructions



Program Control Instructions

- V (overflow) indicates overflow in 2's complement.
- C (carry) indicates unsigned overflow or shift out.
- S (sign) indicates a negative result. (Also called N)
- Z (zero) indicates a result of 0. (all bits are 0)

Conditional Branch Instructions

Mnemonic	Branch condition	Tested condition
<u>BZ</u>	Branch if zero	<u>Z = 1</u>
<u>BNZ</u>	Branch if not zero	<u>Z = 0</u>
<u>BC</u>	Branch if carry	<u>C = 1</u>
<u>BNC</u>	Branch if no carry	<u>C = 0</u>
<u>BM</u>	Branch if minus	<u>S = 1</u>
<u>BP</u>	Branch if plus	<u>S = 0</u>
<u>BV</u>	Branch if overflow	<u>V = 1</u>
<u>BNV</u>	Branch if no overflow	<u>V = 0</u>

A, B
A - B

← ALU ←

← \neq

$A + \bar{B} + 1$

} C₈

} (F₂)

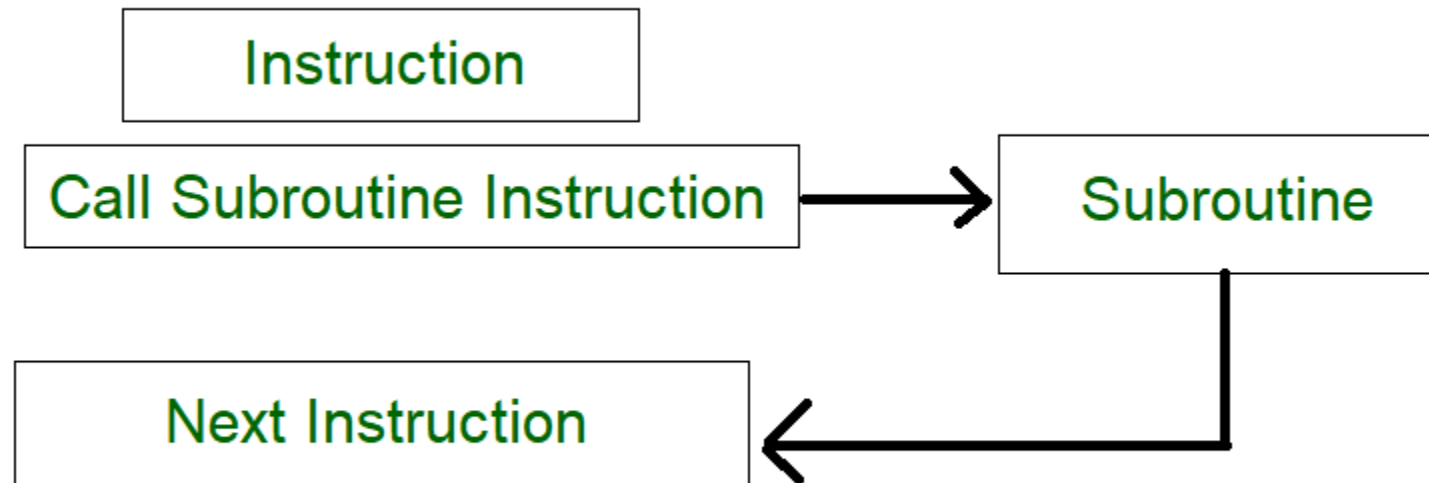
} C₂
C₈
XDR

Subroutine Call and Return

- Subroutine is a self contained sequence of instructions that perform a given computational task.
- During execution of a program a subroutine may be called to perform its functions many times at various points in the main program.
- Each time a subroutine is called, a branch is executed to the beginning of subroutine to start executing its set of instructions.
- After the subroutine is executed, a branch is made back to the main program.
- EG: CALL and RETURN instructions.

Subroutine Call and Return

- The content of the PC must be Saved by the call Subroutine Instruction to make a correct return to the calling program.



RISC and CISC

- CISC characteristics
 - 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.
 - Variable length instruction format.
 - Relatively small number of registers in the CPU.
- EG: Intel x86 architecture, 8085 etc.

RISC and CISC

- RISC characteristics
 - Relatively few instructions in the ISA.
 - Memory access limited to LOAD and STORE instructions.
 - Relatively few addressing modes.
 - Fixed length, easily decoded instruction format.
 - Relatively large number of registers in the CPU, thus all operations are done within the registers of the CPU.
- EG: POWER PC, MIPS

Difference between RISC and CISC

RISC	CISC
1. RISC stands for Reduced Instruction Set Computer.	1. CISC stands for Complex Instruction Set Computer.
2. RISC processors have simple instructions taking about one clock cycle. The average clock cycle per instruction (CPI) is 1.5	2. CSIC processor has complex instructions that take up multiple clocks for execution. The average clock cycle per instruction (CPI) is in the range of 2 and 15.
3. Performance is optimized with more focus on software	3. Performance is optimized with more focus on hardware.
4. It has no memory unit and uses separate hardware to implement instructions..	4. It has a memory unit to implement complex instructions.
5. It has a hard-wired unit of programming.	5. It has a microprogramming unit.
6. The instruction set is reduced i.e. it has only a few instructions in the instruction set. Many of these instructions are very primitive.	6. The instruction set has a variety of different instructions that can be used for complex operations.

Difference between RISC and CISC

7. The instruction set has a variety of different instructions that can be used for complex operations.	7. CISC has many different addressing modes and can thus be used to represent higher-level programming language statements more efficiently.
8. Complex addressing modes are synthesized using the software.	8. CISC already supports complex addressing modes
9. Multiple register sets are present	9. Only has a single register set
10. RISC processors are highly pipelined	10. They are normally not pipelined or less pipelined
11. The complexity of RISC lies with the compiler that executes the program	11. The complexity lies in the microprogram
12. Execution time is very less	12. Execution time is very high
13. Code expansion can be a problem	13. Code expansion is not a problem
14. The decoding of instructions is simple.	14. Decoding of instructions is complex
15. It does not require external memory for calculations	15. It requires external memory for calculations
16. The most common RISC microprocessors are Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, Power Architecture, and SPARC.	16. Examples of CISC processors are the System/360, VAX, PDP-11, Motorola 68000 family, AMD, and Intel x86 CPUs.
17. RISC architecture is used in high-end applications such as video processing, telecommunications, and image processing.	17. CISC architecture is used in low-end applications such as security systems, home automation, etc.

END of Chapter.

- Any Questions??????

PIPELINE, VECTOR PROCESSING AND MULTIPROCESSING

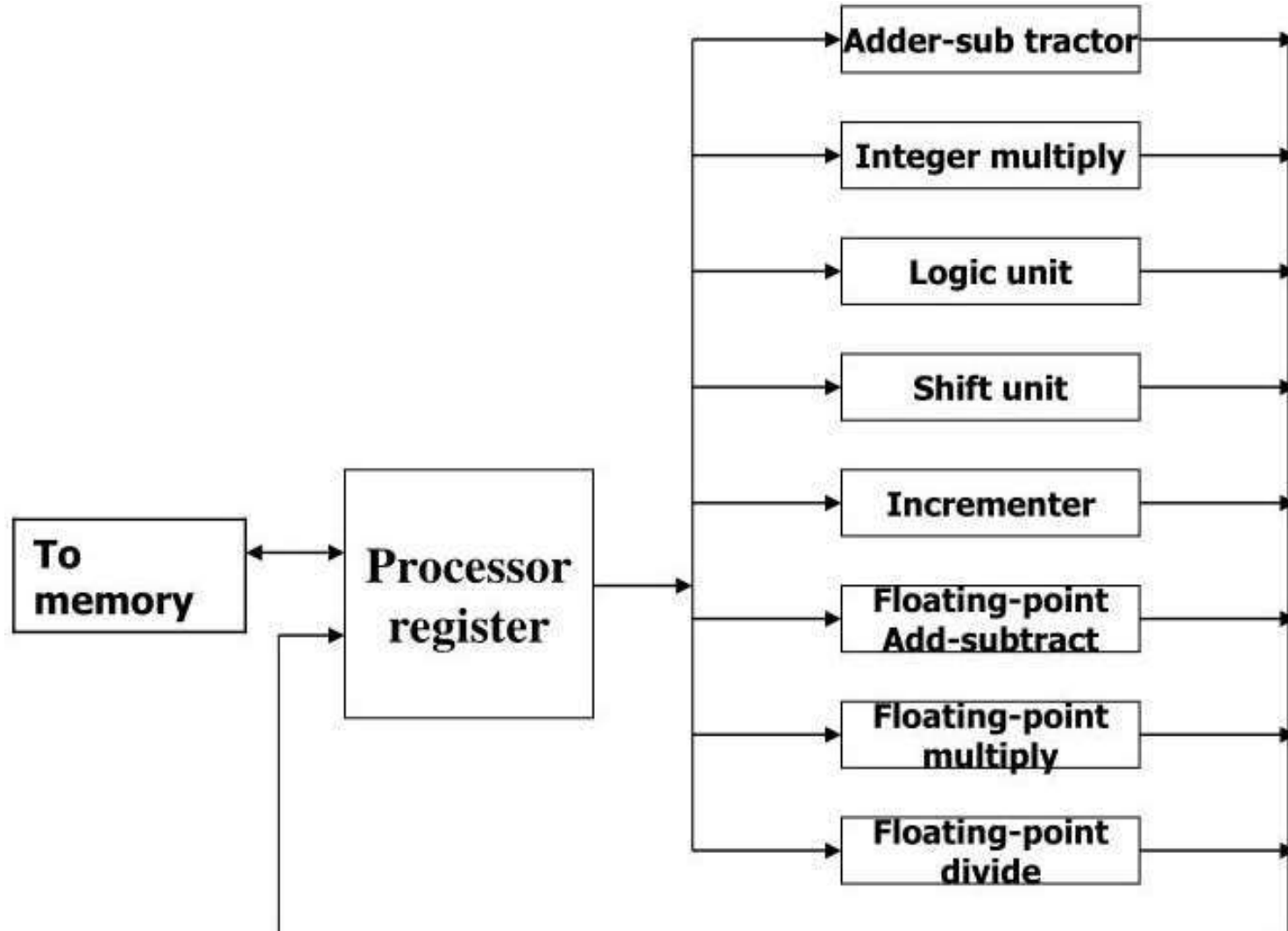
PARALLEL PROCESSING

- Class of technique that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed.
- Computer is able to perform concurrent data processing to achieve faster execution time.
- EG:
 - When an instruction is being executed by the ALU, the next instruction can be read from the memory.
 - The system can be more than one ALU to perform multiple operations at same time.
 - Sometime the System may have multiple processor operating concurrently.

PARALLEL PROCESSING

- The idea is to increase the computing processing capability and increase its throughput.
- Throughput is the amount of processing that can be accomplished during a given interval of time.
- Processor can have many functional units.
- Parallel processing is established by distributing the data among the multiple functional units.
- Eg: arithmetic, logic and shift operations can be separated into three units and the operand diverted into each unit.

PARALLEL PROCESSING



FLYNN'S CLASSIFICATION

- MJ Flynn has classified the parallel processing by the type of instruction stream and data stream.
- The four classification are:
 - SISD (Single Instruction Single Data)
 - SIMD (Single Instruction Multiple Data)
 - MISD (Multiple Instruction Single Data)
 - MIMD (Multiple Instruction Multiple Data)

SISD

- Represents a single computer containing CU, a processor and a memory unit.
- Instructions are executed sequentially and the system may or may not have internal parallel processing capability.
- Parallel processing in this case can be achieved by means of multiple functional units or by pipeline processing.

SIMD

- Represents organization that includes many processing units under the supervision of a common control unit.
- All processor receive the same instruction from the control unit but operate on different items of data.
- The shared memory unit contains multiple modules so that it can communicate with all the processors simultaneously.

MISD

- It is theoretical and has no practical realization or use.

MIMD

- Refers to computer system capable of processing several programs at the same time.
- Most multiprocessor and multicomputer system can be classified in this category.

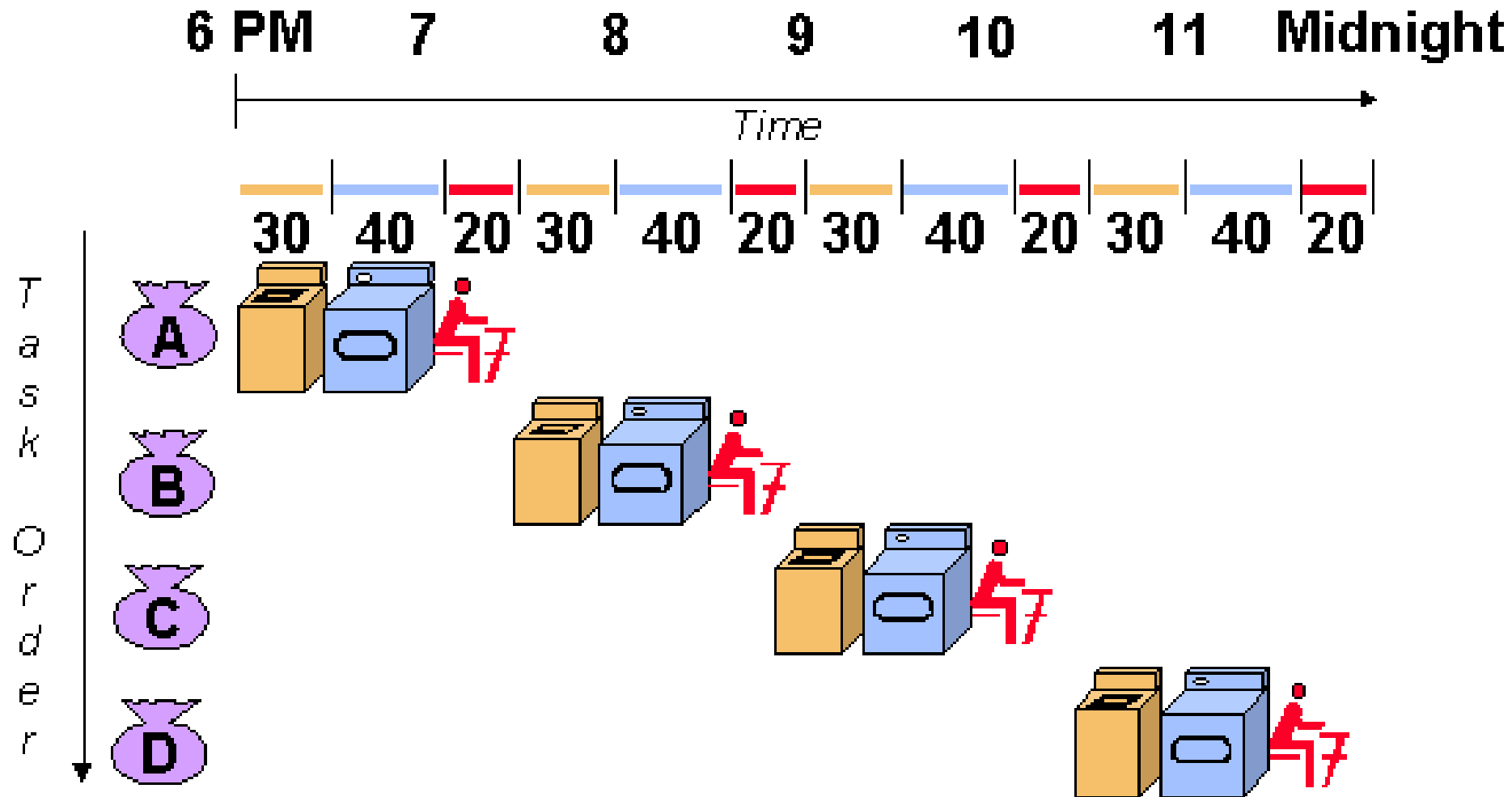
PIPELINING

- A technique of decomposing a sequential process into sub operations, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments.
- Can be visualized as a collection of processing segment through which binary information flows.
- Each segment performs partial processing dictated by the way the task is partitioned. The result obtained from the computation in each segment is transferred to the next segment in the pipeline.

PIPELINING ANALOGY

- A useful method of demonstrating this is the laundry analogy.
- Let's say that there are four loads of dirty laundry that need to be washed, dried, and folded.
- We could put the first load in the washer for 30 minutes, dry it for 40 minutes, and then take 20 minutes to fold the clothes.
- Then pick up the second load and wash, dry, and fold, and repeat for the third and fourth loads.
- Supposing we started at 6 PM and worked as efficiently as possible, we would still be doing laundry until midnight.

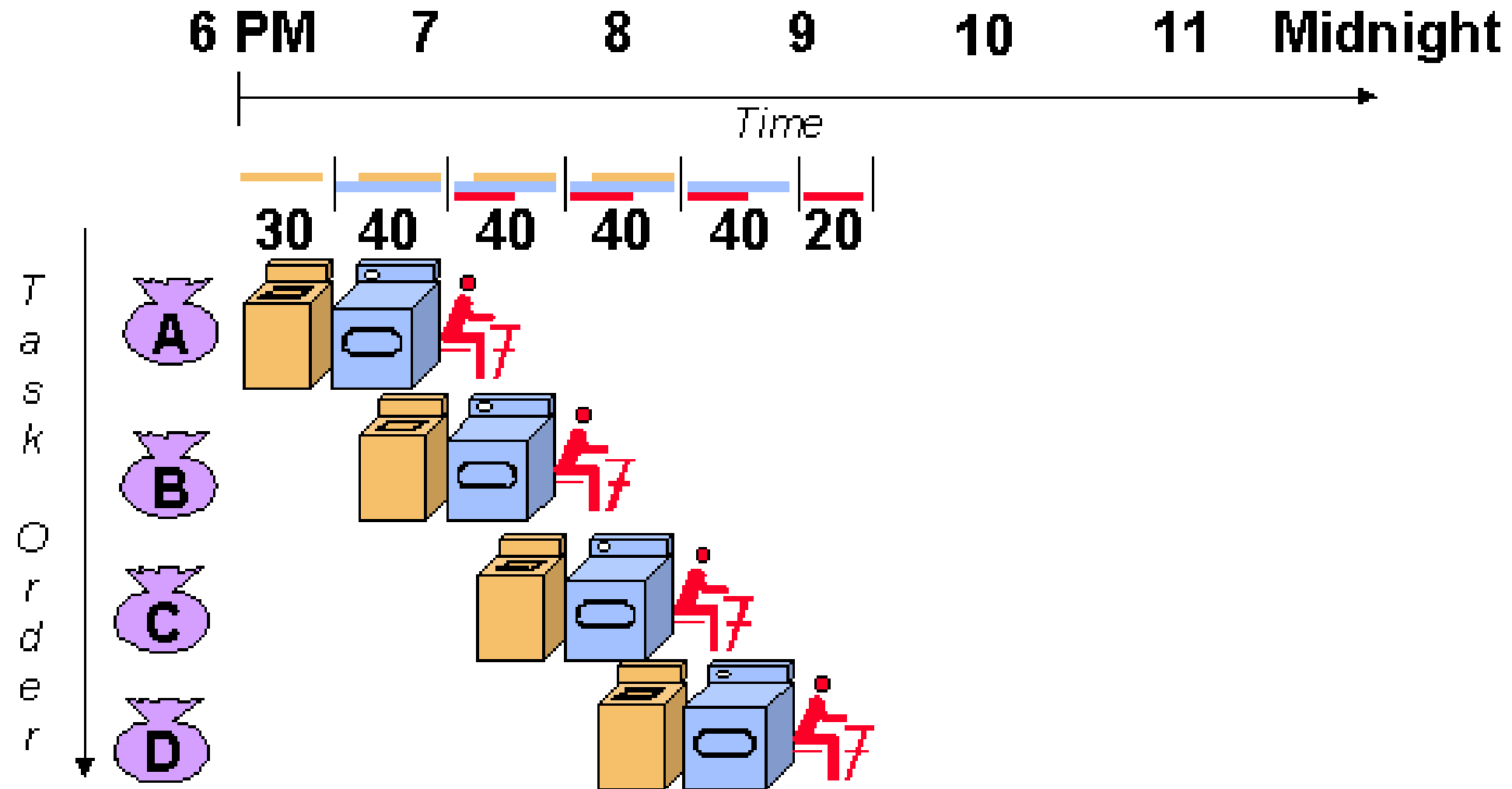
PIPELINING ANALOGY



PIPELINING ANALOGY

- However, a smarter approach to the problem would be to put the second load of dirty laundry into the washer after the first was already clean and whirling happily in the dryer.
- Then, while the first load was being folded, the second load would dry, and a third load could be added to the pipeline of laundry.
- Using this method, the laundry would be finished by 9:30.

PIPELINING ANALOGY

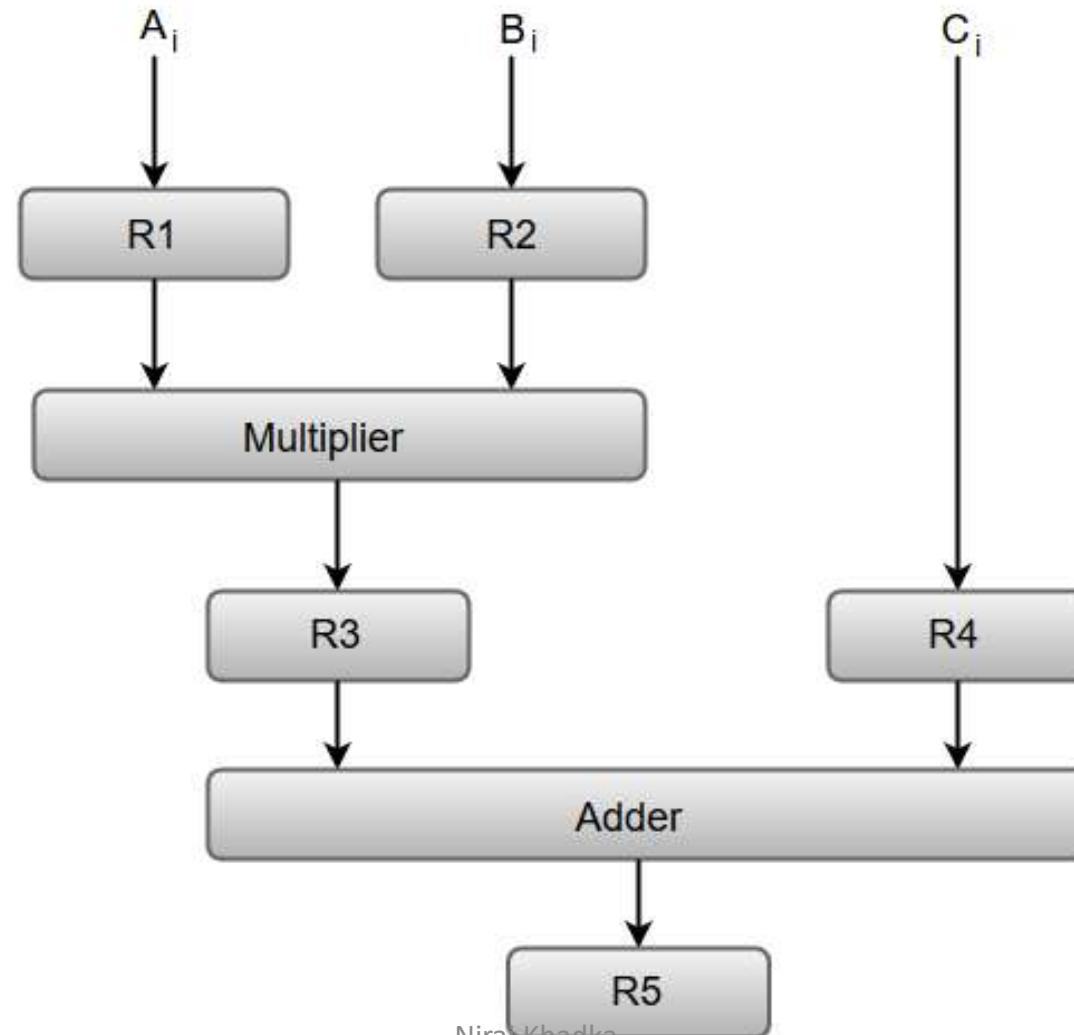


Example of Pipeline Processing

- $A_i * B_i + C_i$ for $i = 1, 2, 3, \dots, 7$
 - $R1 \leftarrow A_i$; $R2 \leftarrow B_i$
 - $R3 \leftarrow R1 * R2$; $R4 \leftarrow C_i$
 - $R5 \leftarrow R3 + R4$

Example of Pipeline Processing

Pipeline Processing:



Example of Pipeline Processing

Clock Pulse number	Segment1		Segment2		Segment3
	R1	R2	R3	R4	R5
1	A1	B1	----	----	----
2	A2	B2	A1*B1	C1	----
3	A3	B3	A2*B2	C2	A1*B1+C1
4	A4	B4	A3*B3	C3	A2*B2+C2
5	A5	B5	A4*B4	C4	A3*B3+C3
6	A6	B6	A5*B5	C5	A4*B4+C4
7	A7	B7	A6*B6	C6	A5*B5+C5
8	----	----	A7*B7	C7	A6*B6+C6
9	----	----	----	----	A7*B7+C7

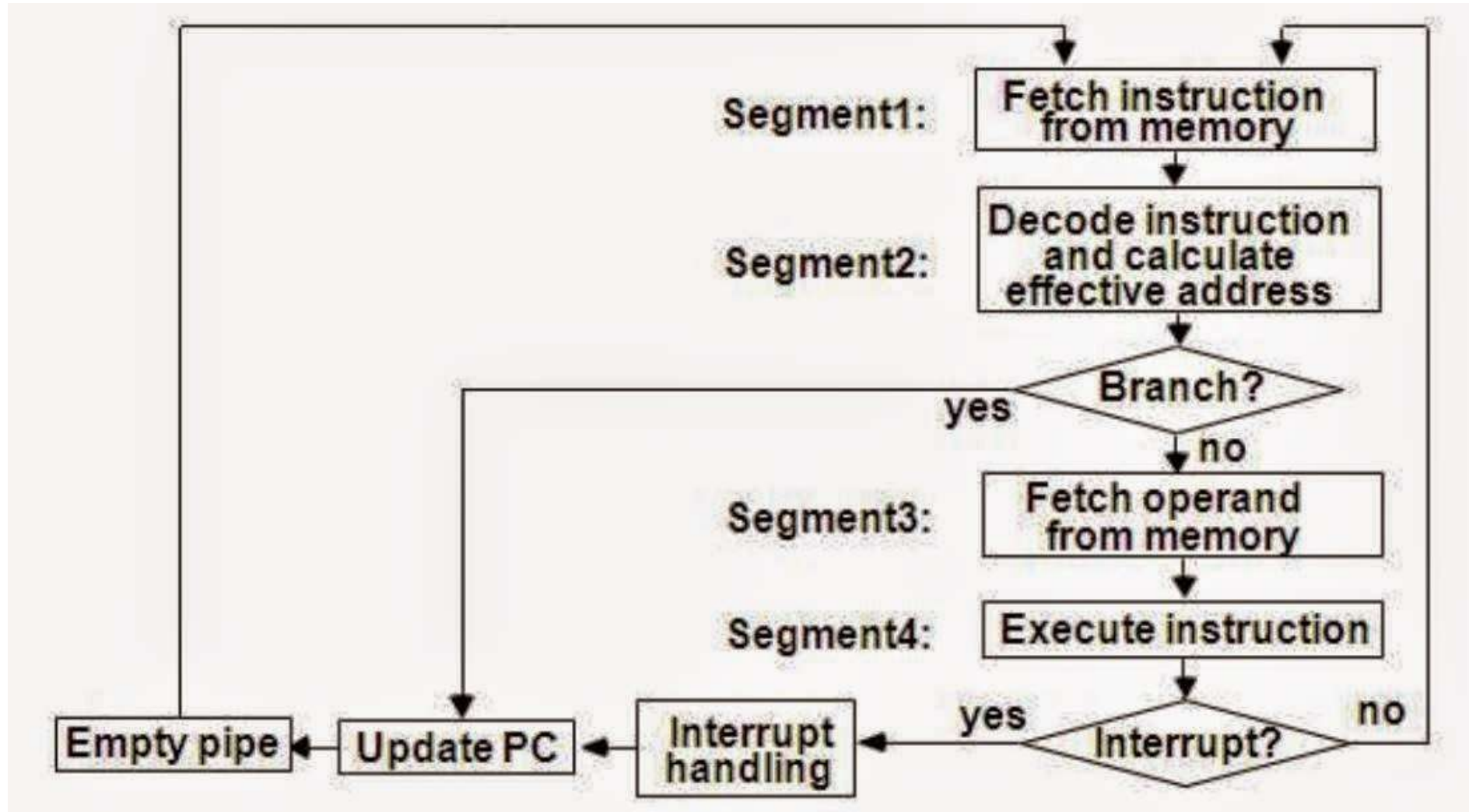
INSTRUCTION PIPELINE

- Pipeline processing can not only be implemented in data but also in instructions.
- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- This causes the instruction fetch and execute phases to overlap and perform simultaneous operations.

INSTRUCTION PIPELINE

- For CISC computers the tentative sequence of steps are:
 - 1. Fetch the instruction from memory
 - 2. Decode the instruction.
 - 3. Calculate the effective address.
 - 4. Fetch the operands from the memory.
 - 5. Execute the instruction.
 - 6. Store the result in proper place.

EG: FOUR SEGMENT INSTRUCTION PIPELINE



EG: FOUR SEGMENT INSTRUCTION PIPELINE

- Four segments are represented as:
 1. FI: segment 1 that fetches the instruction
 2. DA: segment 2 that decodes the instruction and calculate the effective address.
 3. FO: segment 3 that fetches the operands
 4. EX: segment 4 that executes the instruction.

Space time diagram for 4 seg instr pipeline

Step	1	2	3	4	5	6	7	8	9
1	FI	DA	FO	EX					
2		FI	DA	FO	EX				
3			FI	DA	FO	EX			
4				FI	DA	FO	EX		
5					FI	DA	FO	EX	
6						FI	DA	FO	EX

Fig: timing diagram for 4-segment instruction pipeline

PIPELINE HAZARDS

- Pipeline hazard occurs when the instruction pipeline deviates at some phases, some operational condition that do not permit the continued execution.
- Three major difficulties:
 - Structural Hazards:
 - Arise from resource conflicts.
 - HW cannot support all possible combinations of instructions.
 - Data Hazards:
 - Occur when given instruction depends on data from an instruction ahead of it in pipeline.
 - Control Hazards:
 - Result from branch, other instructions that change flow of program (i.e. change PC)

Solution to Structural Hazard

- Avoid structural hazards by duplicating resources.
- EG:
 - an ALU to perform an arithmetic operation and an adder to increment PC
- If not all possible combinations of instructions can be executed, structural hazards occur.
- Pipelines stall result of hazards, CPI increased from the usual “1”
 - $\text{CPI} = \text{Cycles per Instructions}$

STRUCTURAL HAZARD EXAMPLE

Step	1	2	3	4	5	6	7	8	9
1	FI	DA	FO	EX					
2		FI	DA	FO	EX				
3			FI	DA	FO	EX			
4				FI	DA	FO	EX		
5					FI	DA	FO	EX	
6						FI	DA	FO	EX

Fig: timing diagram for 4-segment instruction pipeline

STRUCTURAL HAZARD EXAMPLE

STEP	1	2	3	4	5	6	7	8	9
1	FI	DA	FO	EX					
2		FI	DA	FO	EX				
3			X	X	FI	DA	FO	EX	
4						FI	DA	FO	EX
5							X	X	FI
6									

DATA HAZARDS

- **Why do they exist???**
 - **Pipelining changes when data operands are read, written**
 - **Order differs from order seen by sequentially executing instructions on un-pipelined machine**

- **Consider this example:**

- **ADD R1, R2, R3**
- **SUB R4, R1, R5**
- **AND R6, R1, R7**
- **OR R8, R1, R9**
- **XOR R10, R1, R11**

All instructions after ADD use result of ADD

ADD writes the register in WB but SUB needs it in ID.

This is a data hazard

RESOLVING DATA HAZARDS

- Hardware Interlock
- Operand Forwarding
- Delayed Load

Hardware Interlock

- A circuit that detects instructions whose source operands are destinations of instructions farther up in pipeline.
- It then inserts enough number of clock cycles to delay the execution of such instructions.

Operand Forwarding

- Problem illustrated on previous slide can actually be solved relatively easily – **with forwarding**
- Can we move the result from EX/MEM register to the beginning of ALU (where SUB needs it)?
 - Yes!
- Generally speaking:
 - Forwarding occurs when a result is passed directly to functional unit that requires it.
 - Result goes from output of one unit to input of another

Delayed Load

- Software solution where a compiler is designed in such a way that detect the conflicts; reorder the instructions to delay the loading of conflicting data by inserting no operation instruction.

CONTROL HAZARDS

- Arises from branch and other instructions that change the value of the program counter.
- The conditional branch provides plenty of instruction branch line and it is difficult to determine which branches will be taken or not.
- Approaches for dealing with control hazards:
 - Multiple Streaming
 - Prefetch branch target
 - Branch prediction
 - Loop buffer
 - Delayed Branch

H/W

- Describe all the methods for dealing with Control Hazards
 - Multiple Streaming
 - Prefetch branch target
 - Branch prediction
 - Loop buffer
 - Delayed Branch

VECTOR PROCESSING

- Is a procedure for speeding the processing of information by a computer in which the pipelined units perform arithmetic operations on uniform, linear arrays of data values and a single instruction involves the execution of the same operation on every element of data.
- There is a class of computational problems that are beyond the capabilities of the conventional computer.
- These are characterized by the fact that they require vast number of computation and it take a conventional computer days or even weeks to complete.

Application areas of vector processing

- Long range weather forecasting.
- Petroleum explorations.
- Seismic data analysis
- Medical diagnosis
- Aerodynamic and flight simulations
- AI and expert system
- Human genome mapping
- Image processing

Vector Operations

- A vector V of length n is represented as row vector by

$$V = [V_1 \quad V_2 \quad V_3 \quad \cdots \quad V_n]$$

- The element V_i of vector V is written as $V(I)$ and the index I refers to a memory address or register where the number is stored.

Vector Operations

- Let us consider the program in assembly language that two vectors A and B of length 100 and put the result in vector C.

```
Initialize I = 0  
20 Read A(I)  
   Read B(I)  
   Store C(I) = A(I) + B(I)  
   Increment I = I + 1  
   If I ≤ 100 go to 20  
Continue
```

Vector Operations

- A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop.
- It allows operations to be specified with a single vector instruction of the form:

$$C(1:100) = A(1:100) + B(1:100)$$

Instruction format for vector processor

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length
-------------------	--------------------------	--------------------------	-----------------------------	------------------

Matrix Multiplication

- Let us consider the multiplication of two 3*3 matrix A and B.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

The product matrix C is a 3×3 matrix whose elements are related to the elements of A and B by the inner product:

$$c_{ij} = \sum_{k=1}^3 a_{ik} \times b_{kj}$$

Matrix Multiplication

For example, the number in the first row and first column of matrix C is calculated by letting $i = 1, j = 1$, to obtain

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$

- This requires three multiplication and (after initializing c_{11} to 0) three addition.
- Total number of addition or multiplication required is 3×9 .

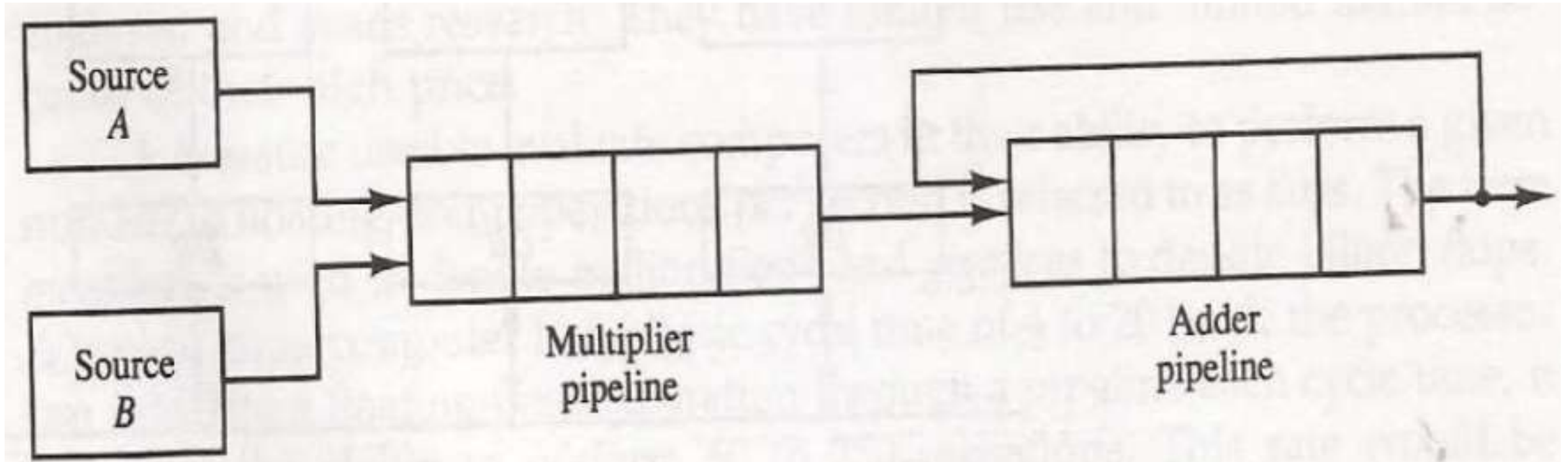
Matrix Multiplication

- In general inner product consists of the sum of k product terms of the form:

$$C = A_1 B_1 + A_2 B_2 + A_3 B_3 + A_4 B_4 + \dots + A_k B_k$$

- In typical application value of k may be 100 or even 1000.
- The inner product calculation on a pipeline vector processor is shown below.
- Floating point adder and multiplier are assumed to have four segments each.

Pipeline for calculating an inner product

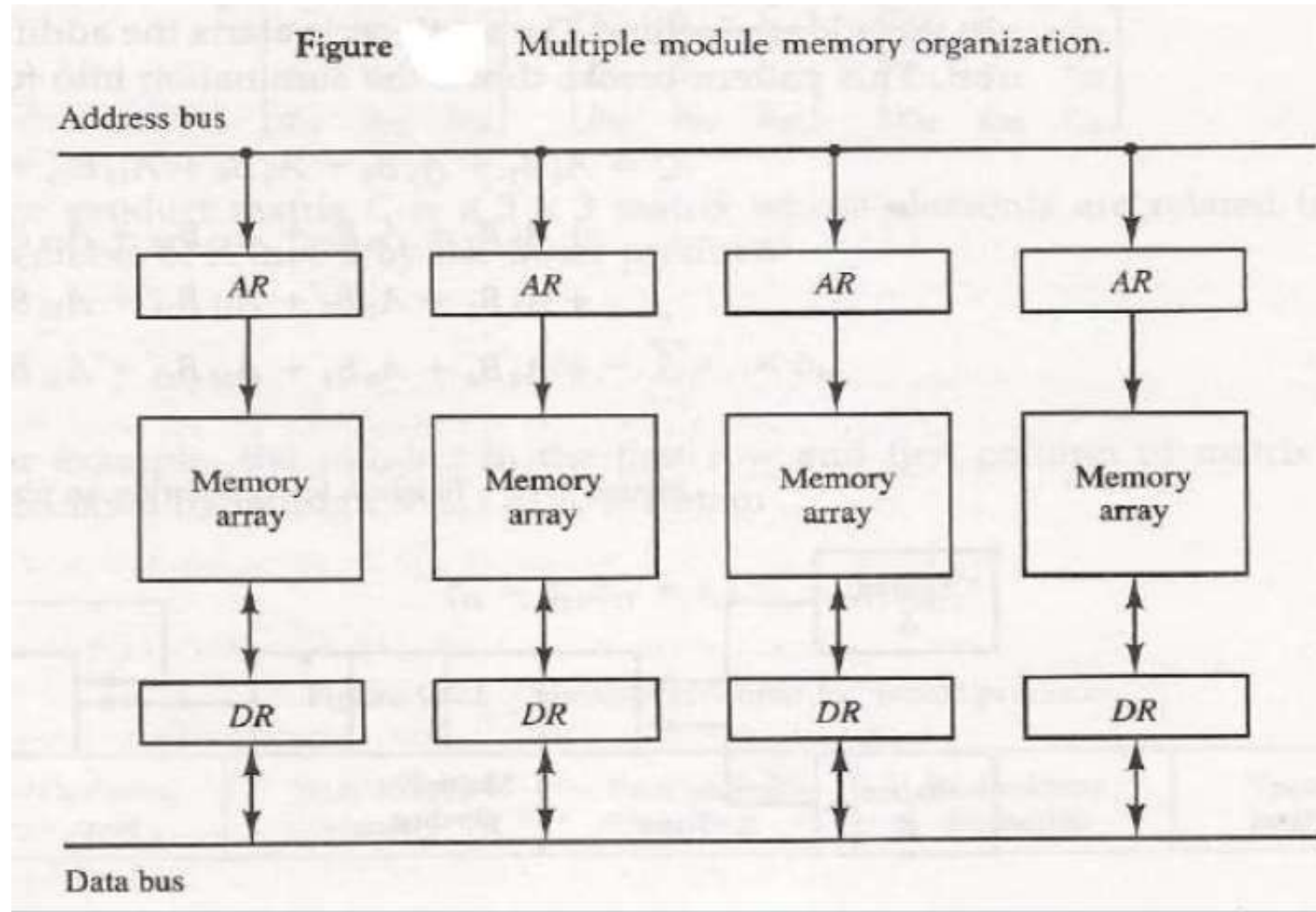


Pipeline for calculating an inner product

$$\begin{aligned} C = & A_1 B_1 + A_5 B_5 + A_9 B_9 + A_{13} B_{13} + \dots \\ & + A_2 B_2 + A_6 B_6 + A_{10} B_{10} + A_{14} B_{14} + \dots \\ & + A_3 B_3 + A_7 B_7 + A_{11} B_{11} + A_{15} B_{15} + \dots \\ & + A_4 B_4 + A_8 B_8 + A_{12} B_{12} + A_{16} B_{16} + \dots \end{aligned}$$

- The four partial sum are added to form the final sum

Memory Interleaving



Multiprocessor System

- A computer system with two or more CPUs, with each one sharing the common main memory as well as peripherals is called a multiprocessor system.
- The key objective of using multiprocessor is to boost the system's execution speed, with other objective being fault tolerance and application matching.
- A multiprocessor is regarded as a means to improve computing speeds, performance and cost-effectiveness, as well as to provide enhanced availability and reliability.

Characteristics of Multiprocessor System

- Consist of more than one CPU.
- Fast Processing.
- Reliability.
- Cost Effective
- Simultaneous processing of programs.

Interconnection Structure of MP system

- Components that form a multiprocessor system are CPUs, IOPs connected to the IO device and memory units.
- There are several physical forms available for establishing an interconnection network.
 - Time shared common bus
 - Multiport memory
 - Crossbar switch
 - Multistage switching network

Time shared Common bus

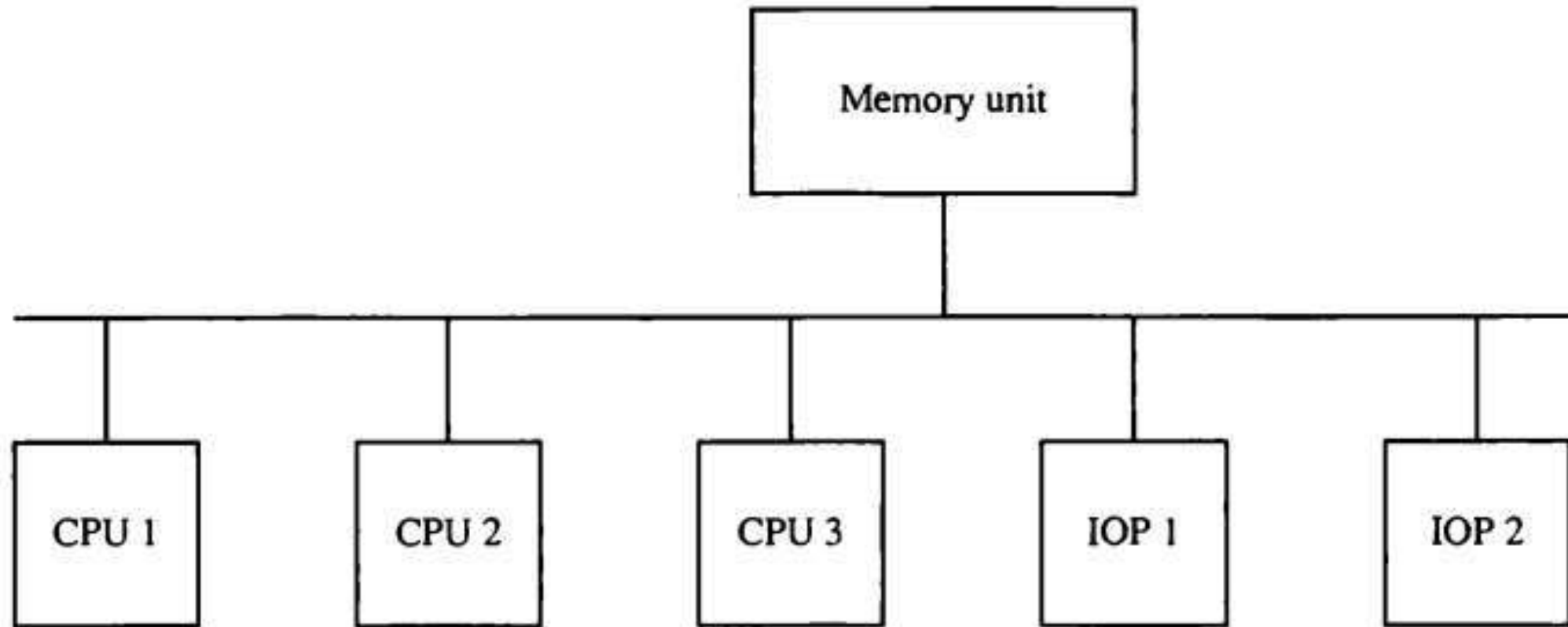
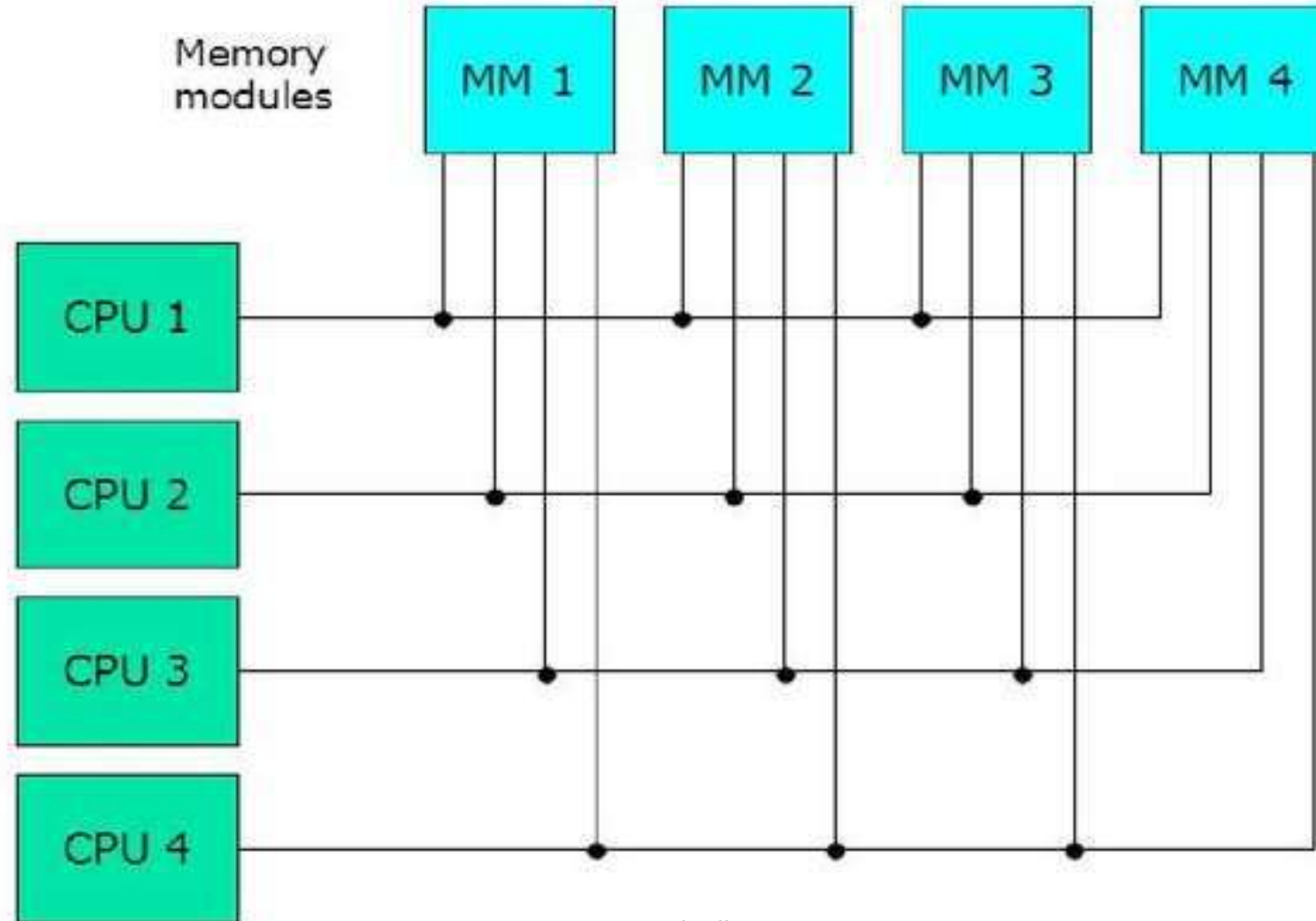
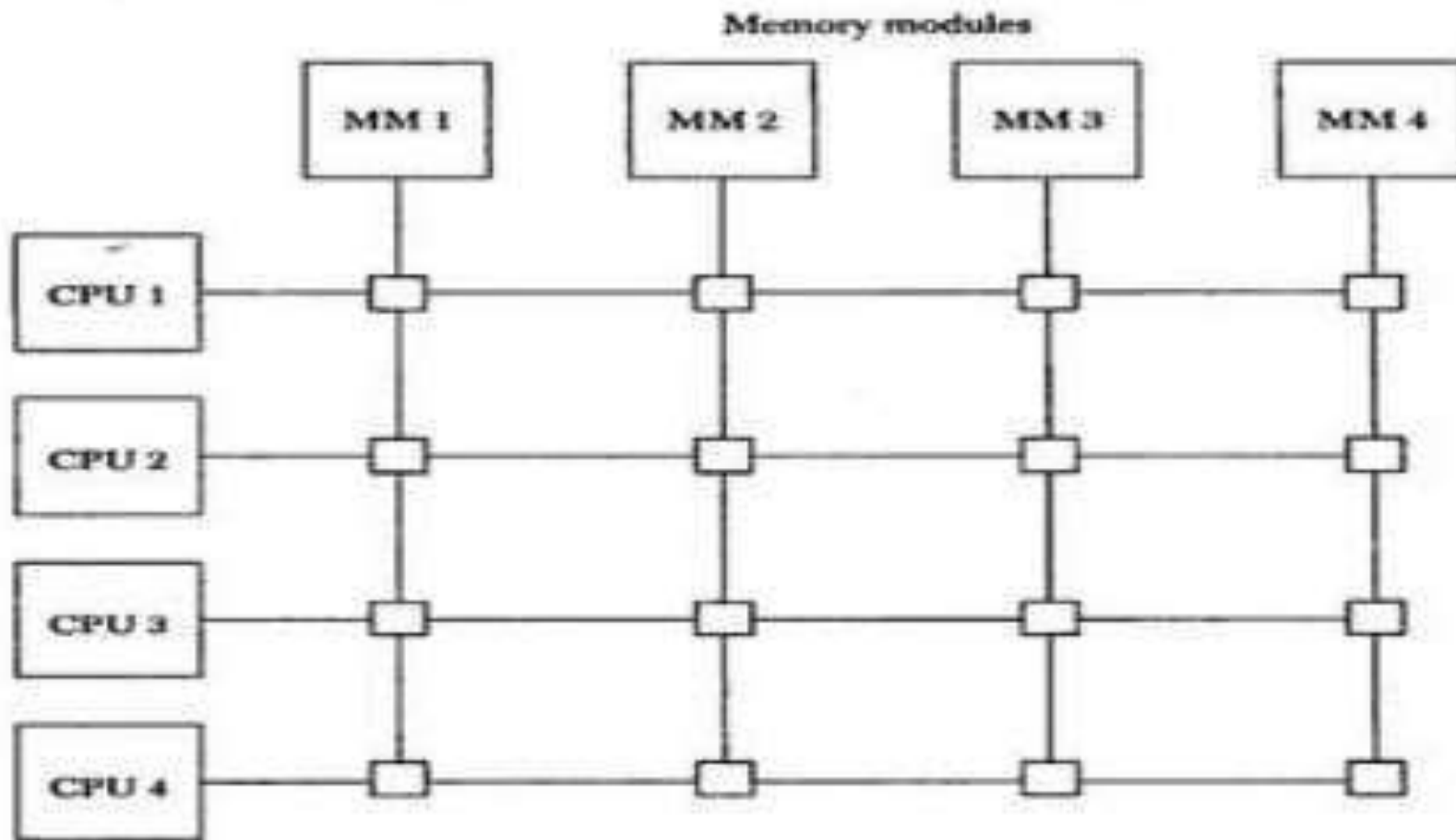


Figure 1 Time-shared common bus organization.

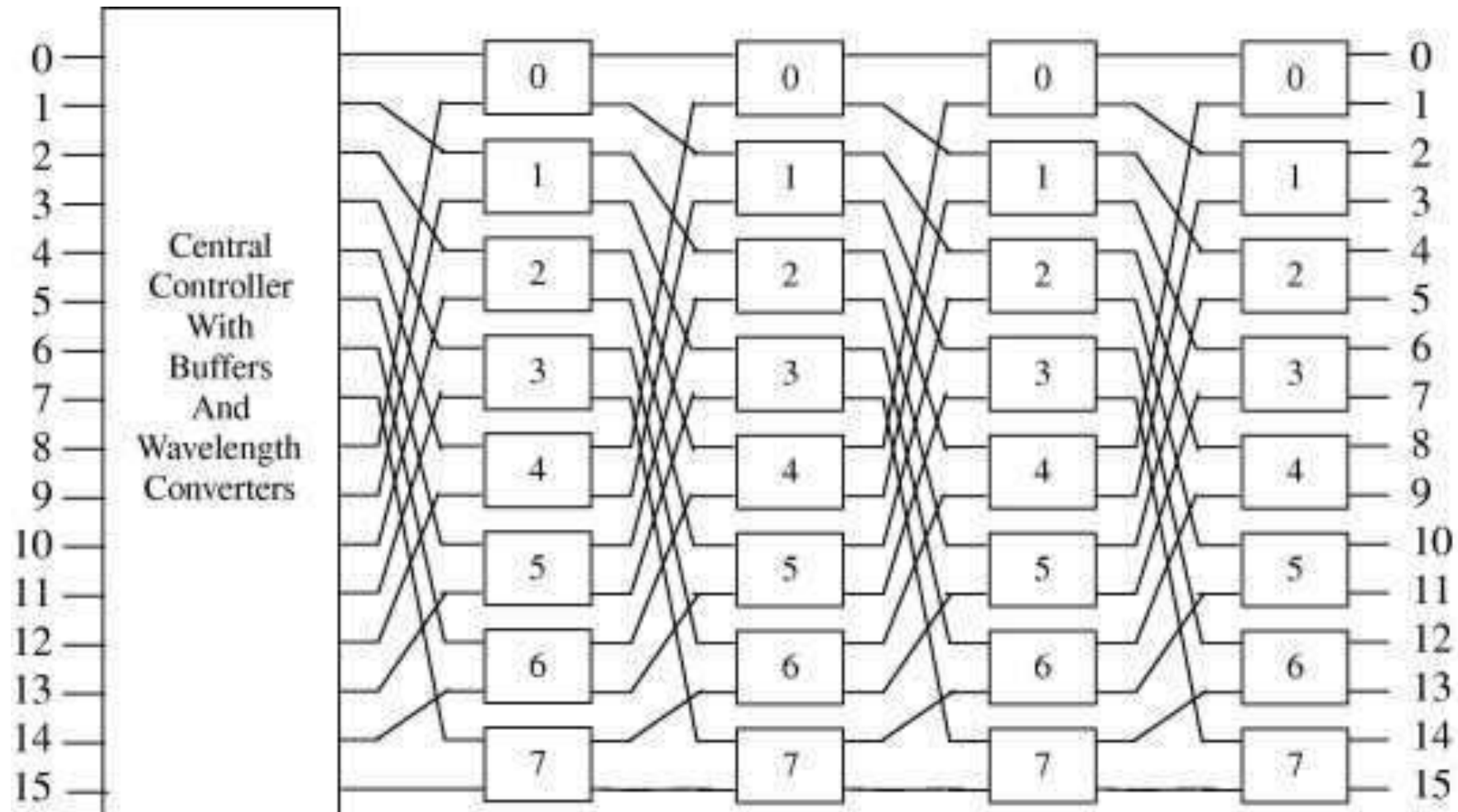
Multiport Memory



Crossbar Switch



Multistage switching network



LAB

1. Identify the 8085 microprocessor pinouts and signals.
2. WAP to add two 8 bit numbers stored in memory location 2020H and 3030H and then output the result on port 8FH.
3. WAP to find the 8 bit data stored in the memory location FFFFH is either odd or even. If the stored data is odd output 01H to port 80H else output 10H to port 80H.
4. WAP to add two 16 bit hex numbers 586FH and 3219H and then store them on the memory location 5050H.
5. Draw the architecture diagram of 8086 microprocessor and explain the EU and BIU in detail.

THE END
THANK YOU !!!

Niraj Khadka