SUBJECT NAME: Computer Organization & Architecture

SUBJECT CODE: 203105253

UNIT 1

Computer Organization:

is about the study of the function and design of the various units of computers that store and process information.

- Encompasses all physical aspects of computer systems.
- E.g., circuit design, control signals, memory types.
- *How does a computer work?*

Computer Architecture:

is a set of rules and methods that describe the functionality, organization, and implementation of computer systems.

Some definitions of architecture define it as describing the capabilities and programming model of a computer.

Why study computer organization and architecture?

- Design better programs, including system software such as compilers, operating systems, and device drivers.
- Optimize program behavior.
- Evaluate (benchmark) computer system performance.
- Understand time, space, and price tradeoffs.

FUNCTIONAL BLOCKS OF A COMPUTER

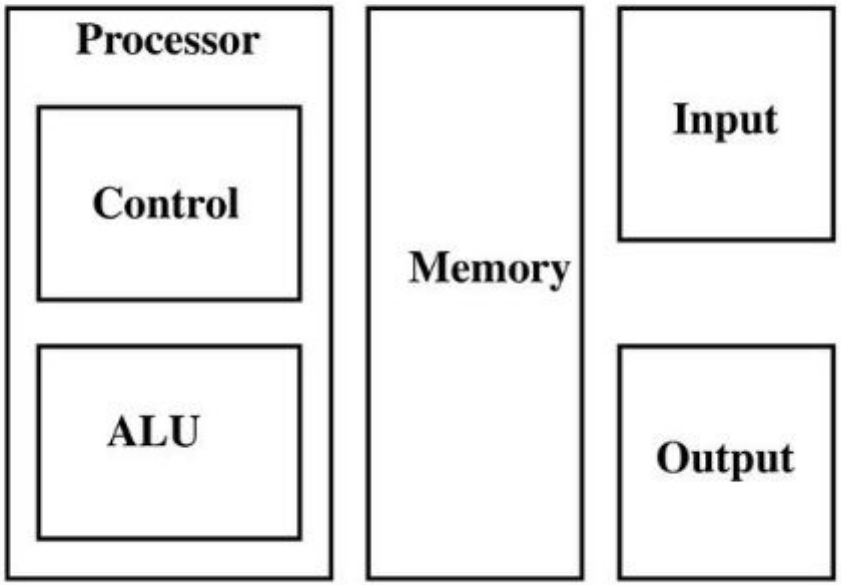
CPU, memory, input-output subsystems, control unit. Instruction set
architecture of a CPU-registers, instruction execution cycle, RTL
Interpretation of instructions, addressing modes, instruction set.
Case study- instruction set of some common CPUs

- The components from which computers are built, i.e., computer organization.
- In contrast, computer architecture is the science of integrating those components to achieve a level of functionality and performance.
- It is as if computer organization examines the bricks, nails, and other building material
- While computer architecture looks at the design of the house.

FUNCTIONAL UNITS OF COMPUTER

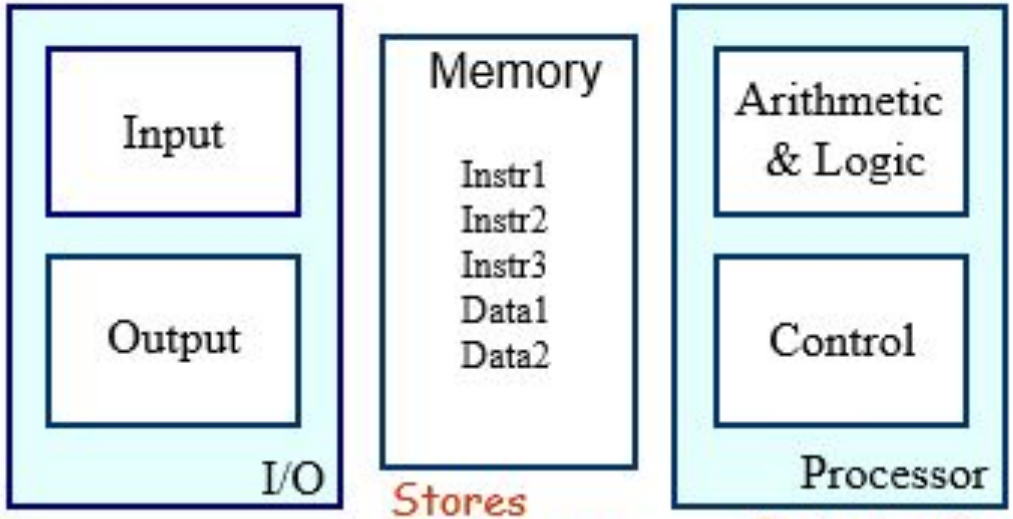
- Input Unit
- Output Unit
- Central processing Unit (ALU and Control Units)
- Memory
- Bus Structure

The Big Picture



Input unit accepts information:
- Human operators,
- Electromechanical devices
- Other computers

Arithmetic and logic unit(ALU):
- Performs the desired operations on the input information as determined by instructions in the memory



Output unit sends results of processing:
- To a monitor display,
- To a printer

Stores information:
- Instructions,
- Data

Control unit coordinates various actions
- Input,
- Output
- Processing

Function

IMPORTANT SLIDE !

- ALL computer functions are:
  - Data PROCESSING
  - Data STORAGE
  - Data MOVEMENT
  - CONTROL

Data = Information

- NOTHING ELSE!

Coordinates How Information is Used

INPUT UNIT:

- Converts the external world data to a binary format, which can be understood by CPU

- Eg: Keyboard, Mouse, Joystick etc

OUTPUT UNIT:

- Converts the binary format data to a format that a common man can understand

- Eg: Monitor, Printer, LCD, LED etc

CENTRAL PROCESSING UNIT

- The "brain" of the machine
- Responsible for carrying out computational task
- Contains ALU, CU, Registers
- ALU Performs Arithmetic and logical operations
- CU Provides control signals in accordance with some timings which in turn controls the execution process
- Register Stores data and result and speeds up the operation

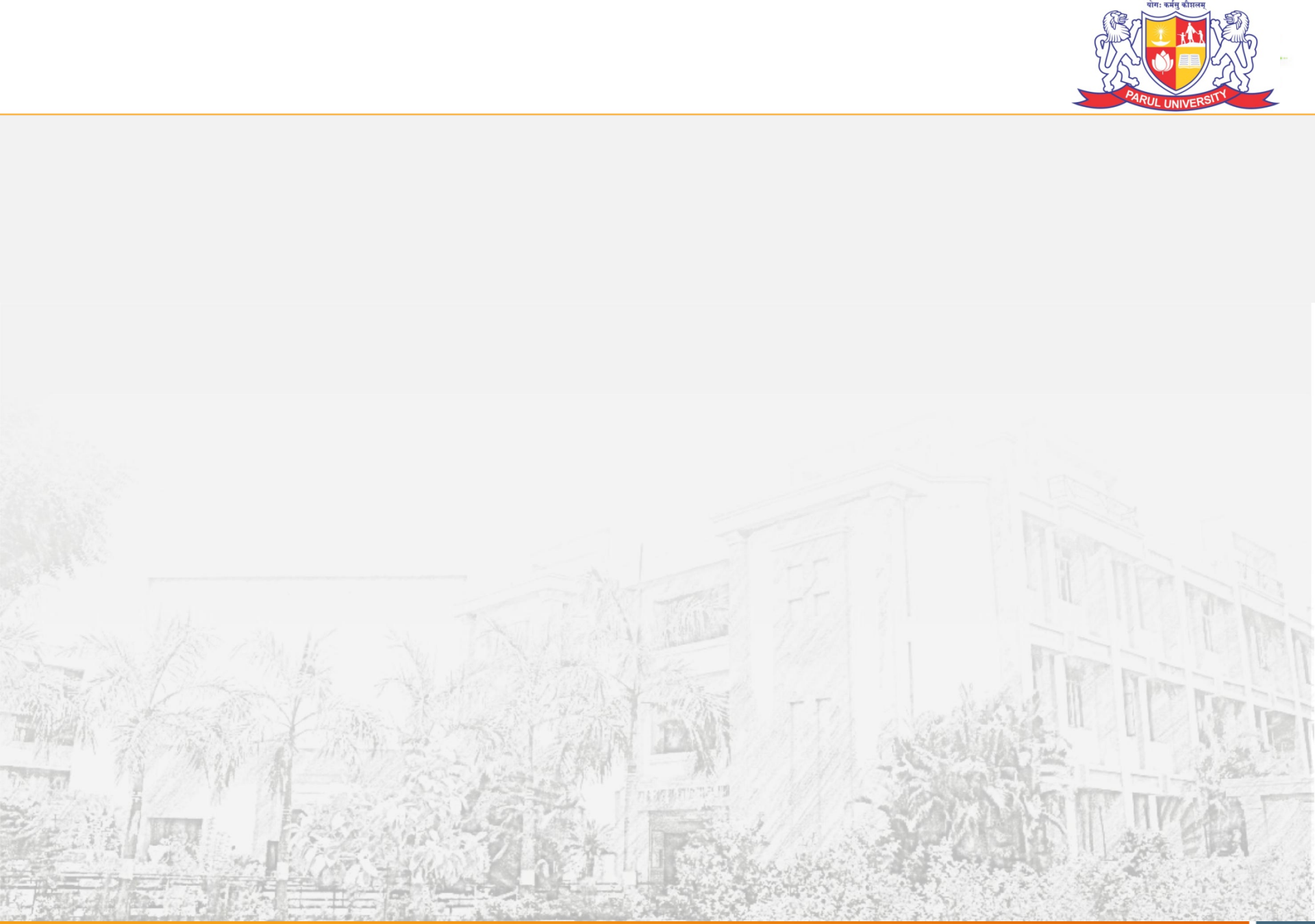
Instruction Set Architecture (ISA)

The *Instruction Set Architecture* (ISA) is the part of the processor that is visible to the programmer or compiler writer.

The ISA serves as the boundary between software and hardware

The 3 most common types of ISAs are:

1. ***Stack*** - The operands are implicitly on top of the stack.
2. ***Accumulator*** - One operand is implicitly the accumulator.
3. ***General Purpose Register (GPR)*** - All operands are explicitly mentioned, they are either registers or memory locations.



Instruction code

Instruction Codes

- Program
  - A program is a set of instructions that specify the operations, operands and the sequence by which processing has to occur.
- Computer Instruction
  - A computer instruction is a binary code that specifies a sequence of microoperations for the computer.
  - The computer reads each instruction from memory and places it in a control register.
  - The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.

Instruction Codes

- Instruction Code
  - An instruction code is a group of bits that instruct the computer to perform a specific operation.
  - Example

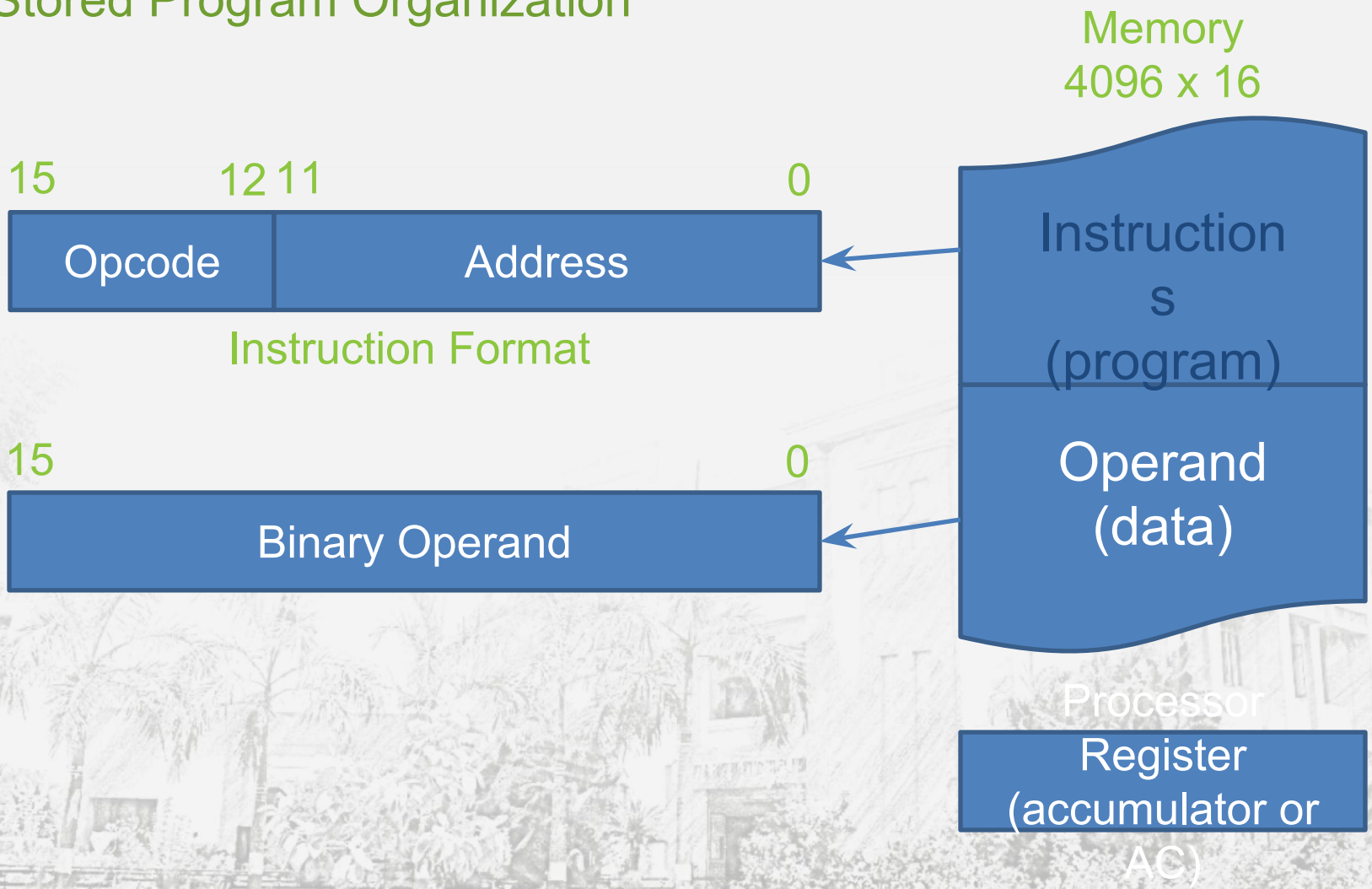
Unique Binary code is assigned to every OpCode

- Operation Code (Opcode)
  - The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.
  - The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.
  - The operation code must consist of at least n bits for a given 2^n (or less) distinct operations.

Stored Program Organization

- The simplest way to organize a computer is to have one processor register(AC) and an instruction code format with two parts.
- The first part specifies the operation (opcode) to be performed and the second specifies an address (operand).
- 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



Instruction format of basic computer

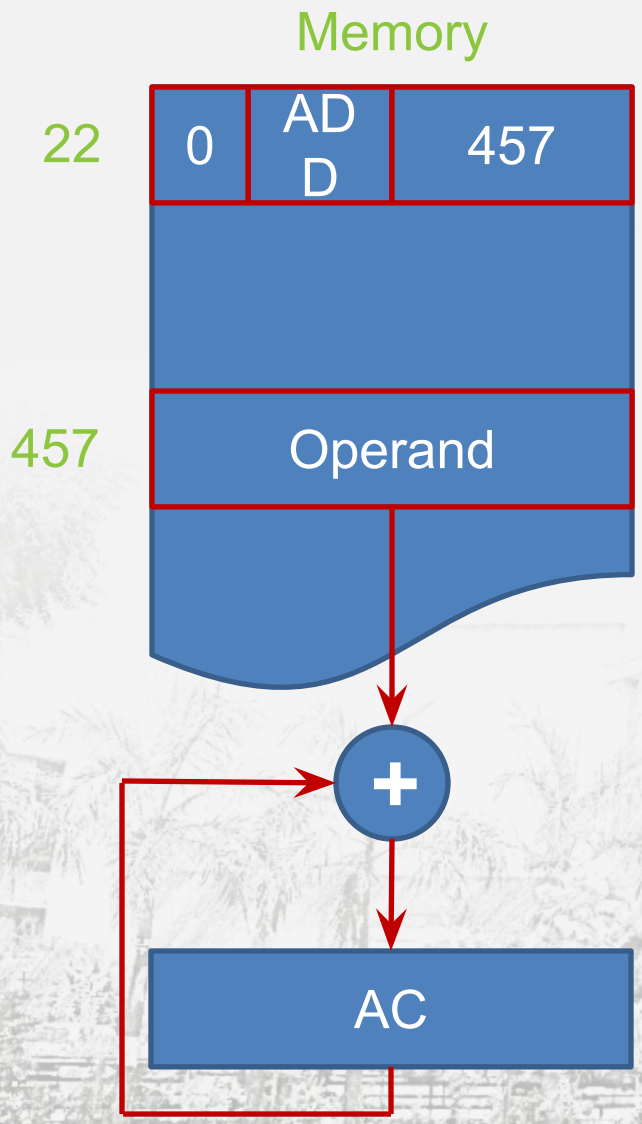
Instruction Format

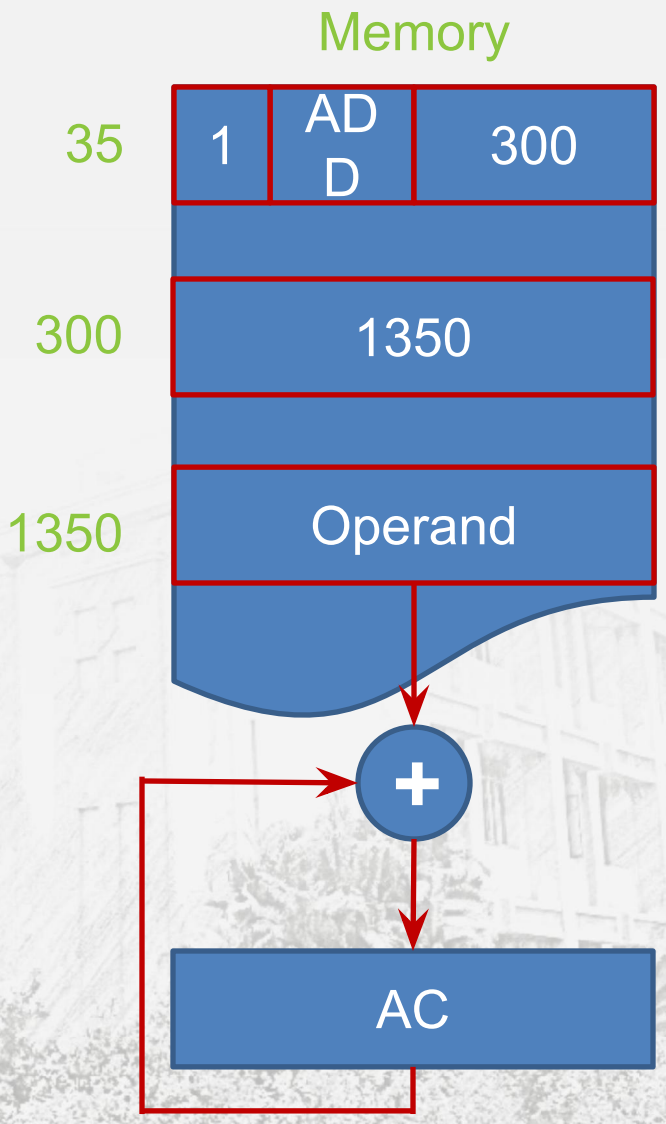
Add Instruction – ADD 457

Direct & Indirect Addressing of Memory

- If the second part of an instruction format specifies the address of an operand, the instruction is said to have a **direct address**.
- In **Indirect address**, 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.

Direct & Indirect Addressing of Memory

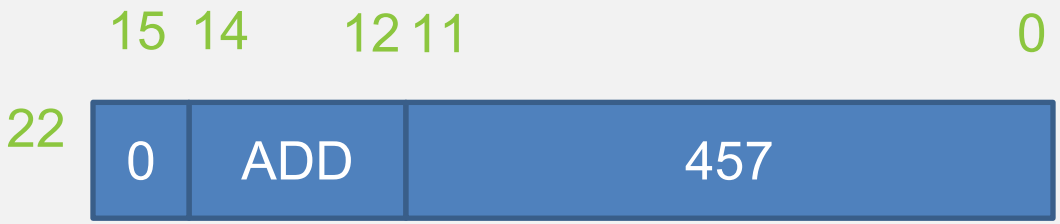




Direct & Indirect Addressing of Memory

- One bit of the instruction code can be used to distinguish between a direct and an indirect address.
- It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I.
- The mode bit is 0 for a direct address and 1 for an indirect address.

Direct & Indirect Addressing of Memory



- A direct address instruction is placed at address 22 in memory.
- The I bit is 0, so the instruction is recognized as a direct address instruction.
- The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
- The control finds the operand in memory at address 457 and adds it to the content of AC.

Direct & Indirect Addressing of Memory

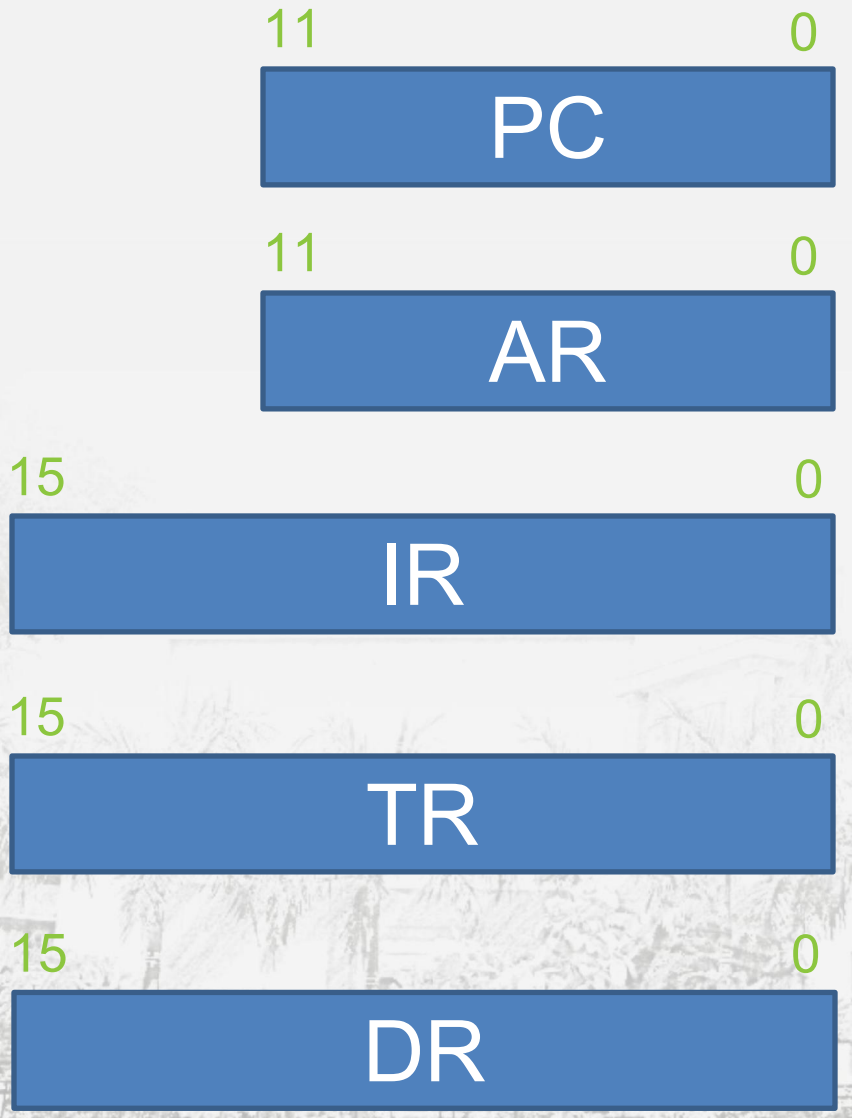
- The instruction in address 35 has a mode bit I = 1, recognized as an indirect address instruction.
- The address part is the binary equivalent of 300.
- The control goes to address 300 to find the address of the operand.
- The address of the operand in this case is 1350.
- The operand found in address 1350 is then added to the content of AC.

Direct & Indirect Addressing of Memory

- The indirect address instruction needs two references to memory to fetch an operand.
- The first reference is needed to read the address of the operand.
- Second reference is for the operand itself.
- The memory word that holds the address of the operand in an indirect address instruction is used as a pointer to an array of data.

Computer Registers

Computer Registers



Program Counter(12)

Holds address of instruction

Address Register(12)

Holds address for memory

Instruction Register(16)

Holds instruction code

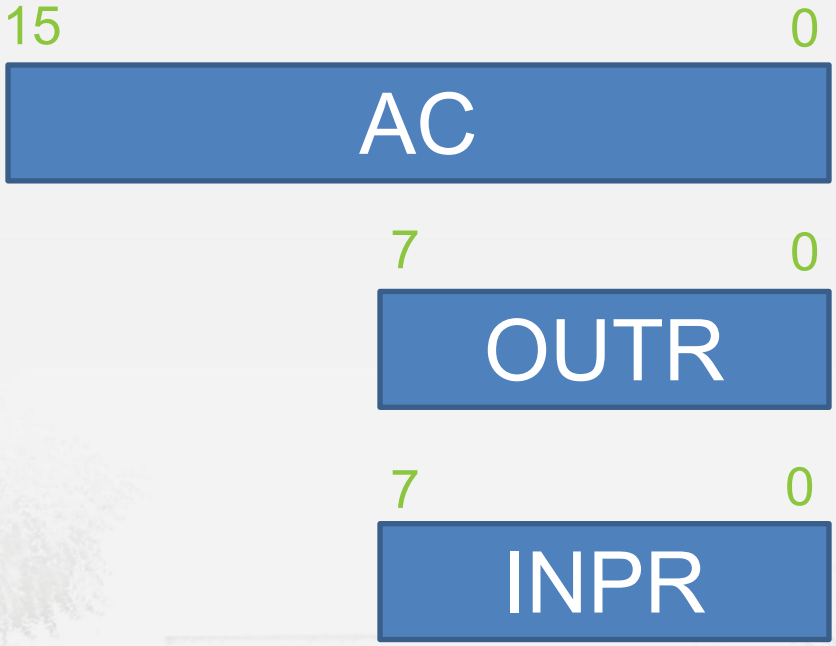
Temporary Register(16)

Holds temporary data

Data Register(16)

Holds memory operand

Computer Registers



Accumulator(16)

Processor Register

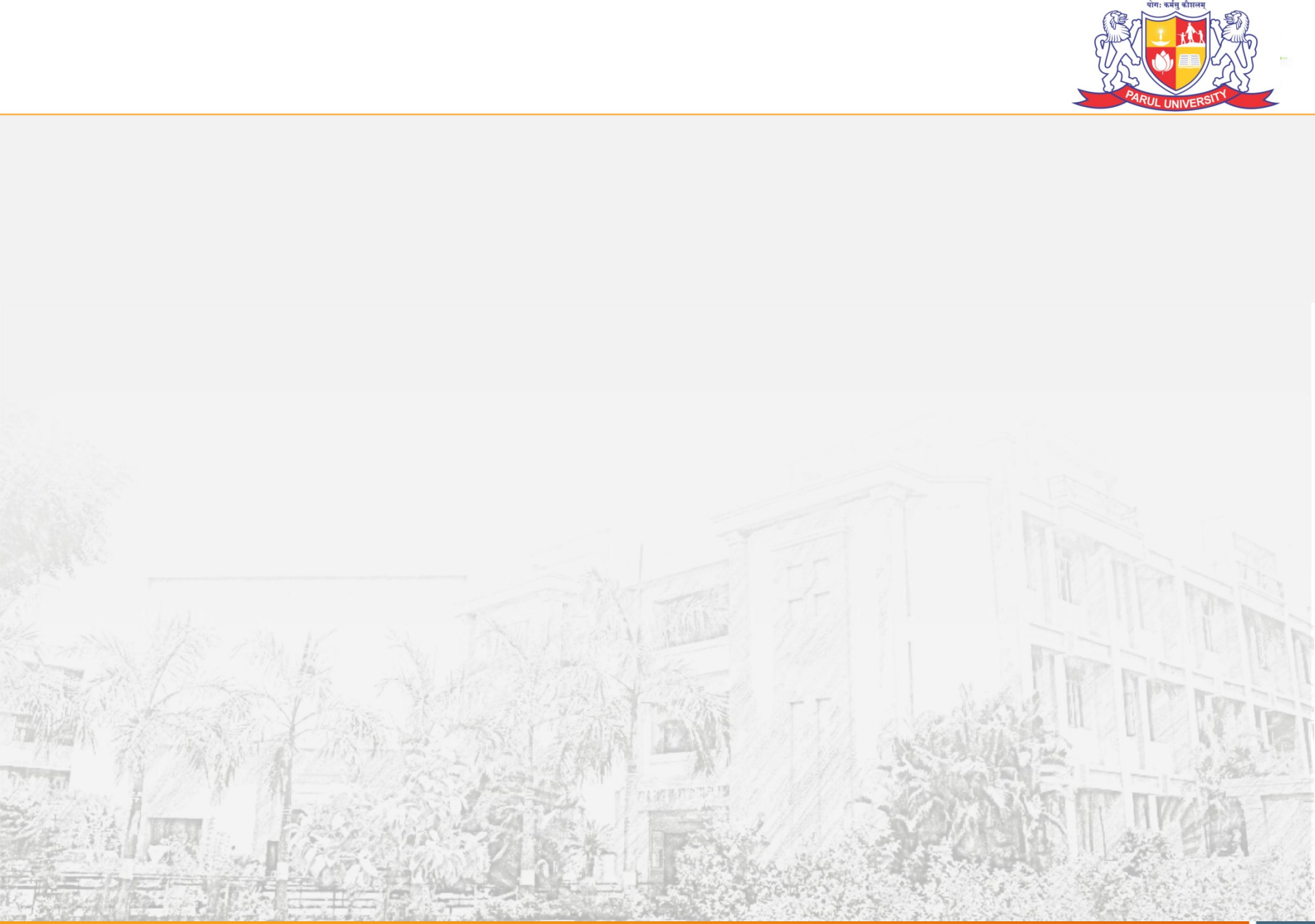
Output Register(8)

Holds output character

Input Register(8)

Holds input character

Memory
4096 words
16 bits per word

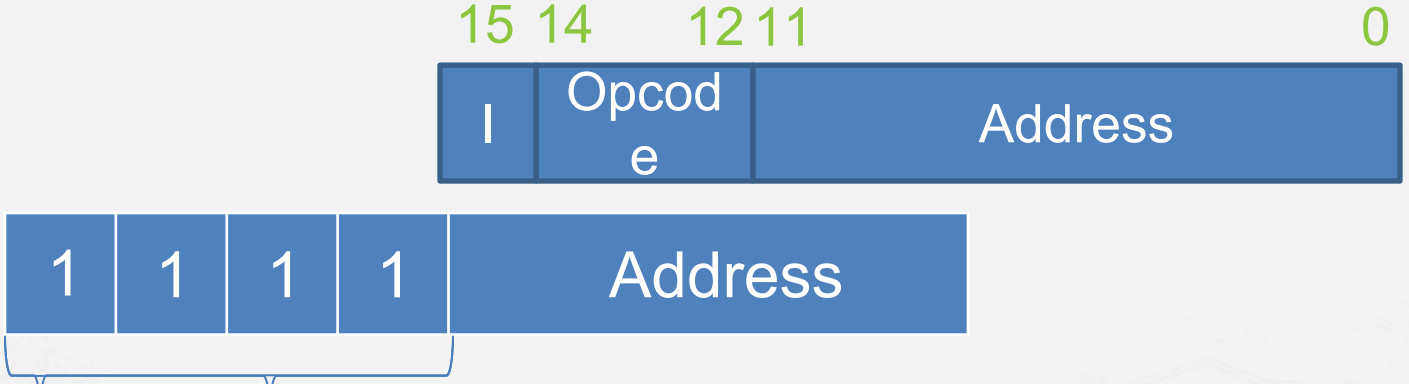


Computer Instructions

Types of Computer Instructions

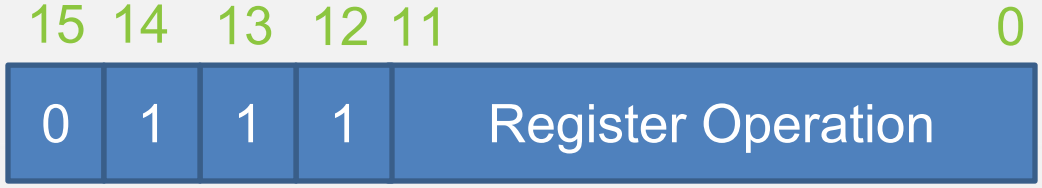
1. Memory Reference Instruction
2. Register Reference Instruction
3. Input – Output Instruction

1. Memory Reference Instruction



0xx	8xx	AN	AND the content of memory to
1xx	9xx	AD	Add the conte of memory to
2xx	Axx	LD	Load memory word to
3xx	Bxx	ST	Store co tent of AC in
4xx	Cxx	BU	Bran memory
5xx	Dxx	BS	Bran ha ve return
6xx	Exx	IS	Increme skip if
x	x	Z	zero

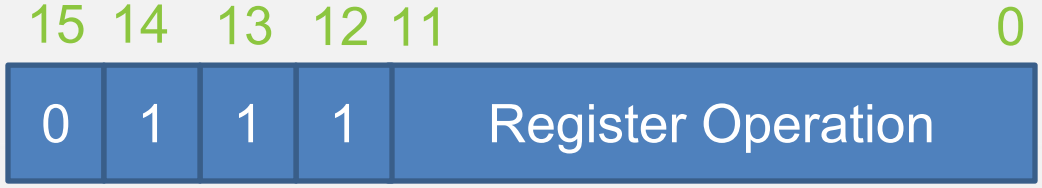
2. Register Reference Instruction



0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0

7800	CLA	Clear AC
7400	CLE	Clear E
7200	CMA	Complement AC
7100	CME	Complement E
7080	CIR	Circulate right AC and E
7040	CIL	Circulate left AC and E
7020	INC	Increment AC
7010	SPA	Skip next if AC positive
7008	SNA	Skip next if AC negative
7004	SZA	Skip next if AC zero
7002	SZE	Skip next if E zero
7001	HLT	Halt computer

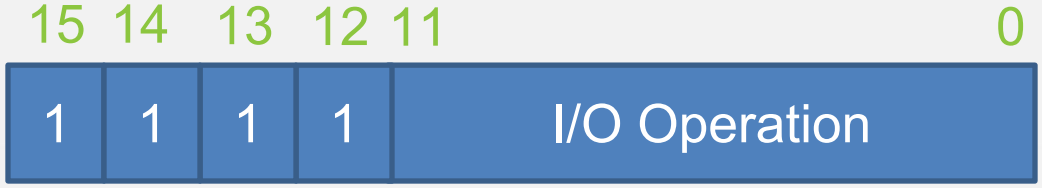
2. Register Reference Instruction



0	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1

7010	SPA	Skip next instruction if AC is positive
7008	SNA	Skip next instruction if AC is negative
7004	SZA	Skip next instruction if AC is zero
7002	SZE	Skip next instruction if E is zero
7001	HLT	Halt computer

3. Input – Output Instruction

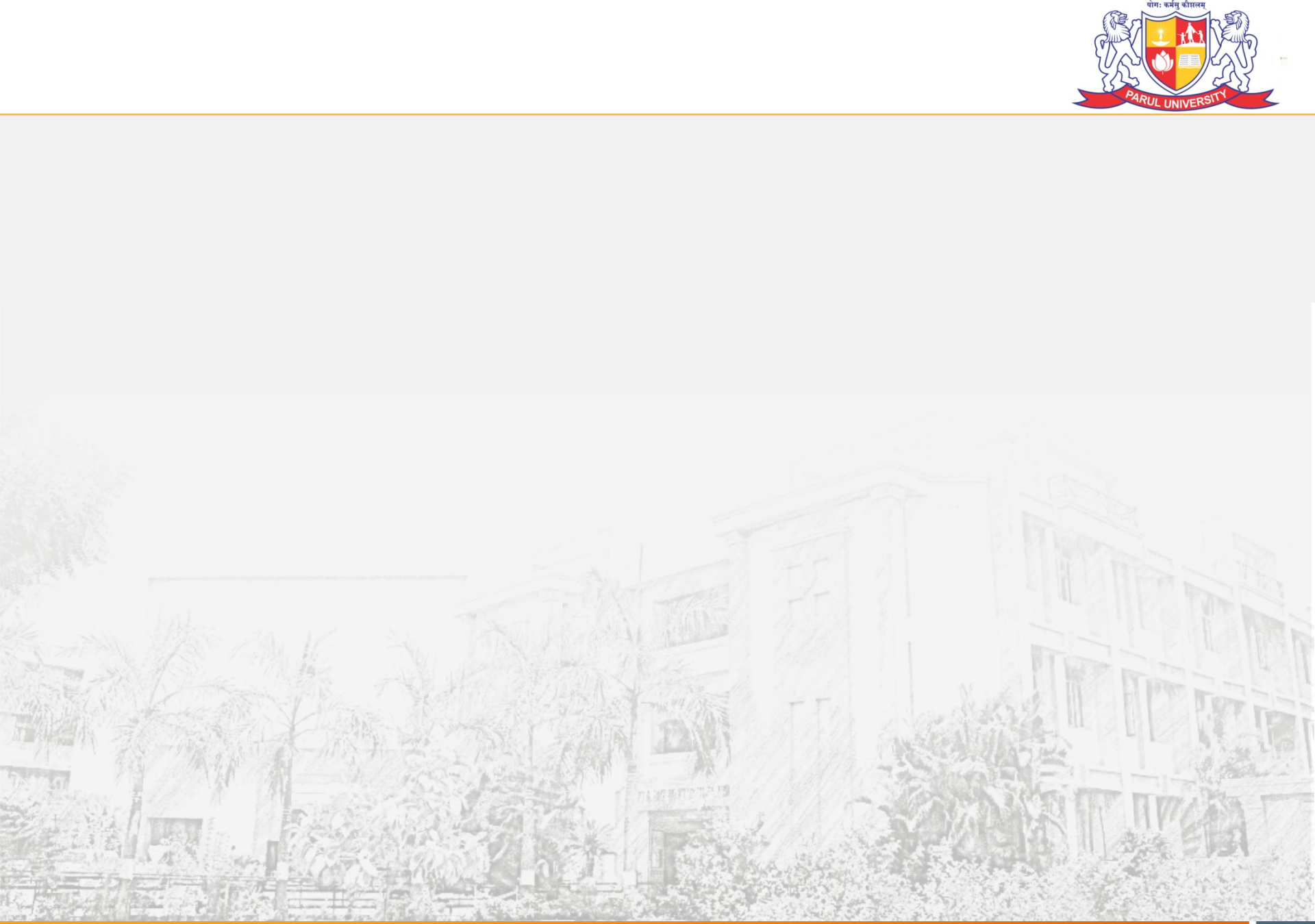


1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

F80 IN Input character to
F40 OUT Output character from AC
F20 SKI Skip on input flag
F10 SKO Skip on output flag
F08 ION Interrupt on
F04 IOF Interrupt off

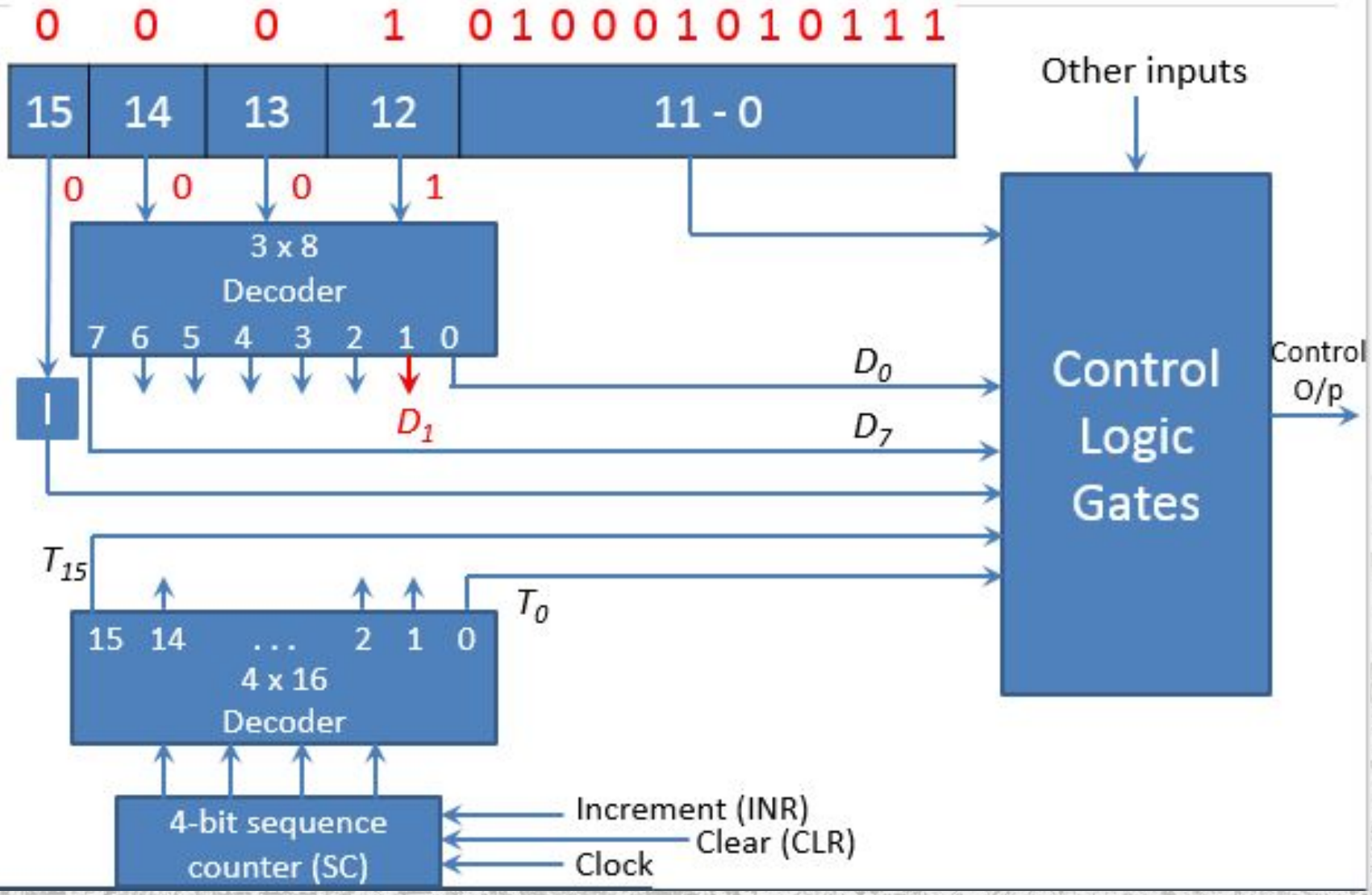
Instruction Set Completeness

- Instruction set is said to be complete if it includes sufficient number of instructions in each of the following categories:
  1. Arithmetic, logical and shift instructions
  2. Instructions for moving information to and from memory and processor registers
  3. Program control instructions together with instructions that check status conditions
  4. Input and output instructions



Timing & Control

Control Unit of Basic Computer



Control Unit

- Components of Control unit are
  1. Two decoders
  2. A sequence counter
  3. Control logic gates
- An instruction read from memory is placed in the instruction register (IR).
- In control unit the IR is divided into three parts: I bit, the operation code (12-14)bit, and bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder.
- Bit-15 of the instruction is transferred to a flip-flop designated by the symbol I.

Control Unit

- The eight outputs of the decoder are designated by the symbols D_0 through D_7.
- Bits 0 through 11 are applied to the control logic gates.
- The 4-bit sequence counter can count in binary from 0 through 15. The outputs of counter are decoded into 16 timing signals T_0 through T_{15}.
- The sequence counter SC can be incremented or cleared synchronously.
- Most of the time, the counter is incremented to provide the sequence of timing signals out of 4 X 16 decoder.
- Once in awhile, the counter is cleared to 0, causing the next timing signal to be T_0.

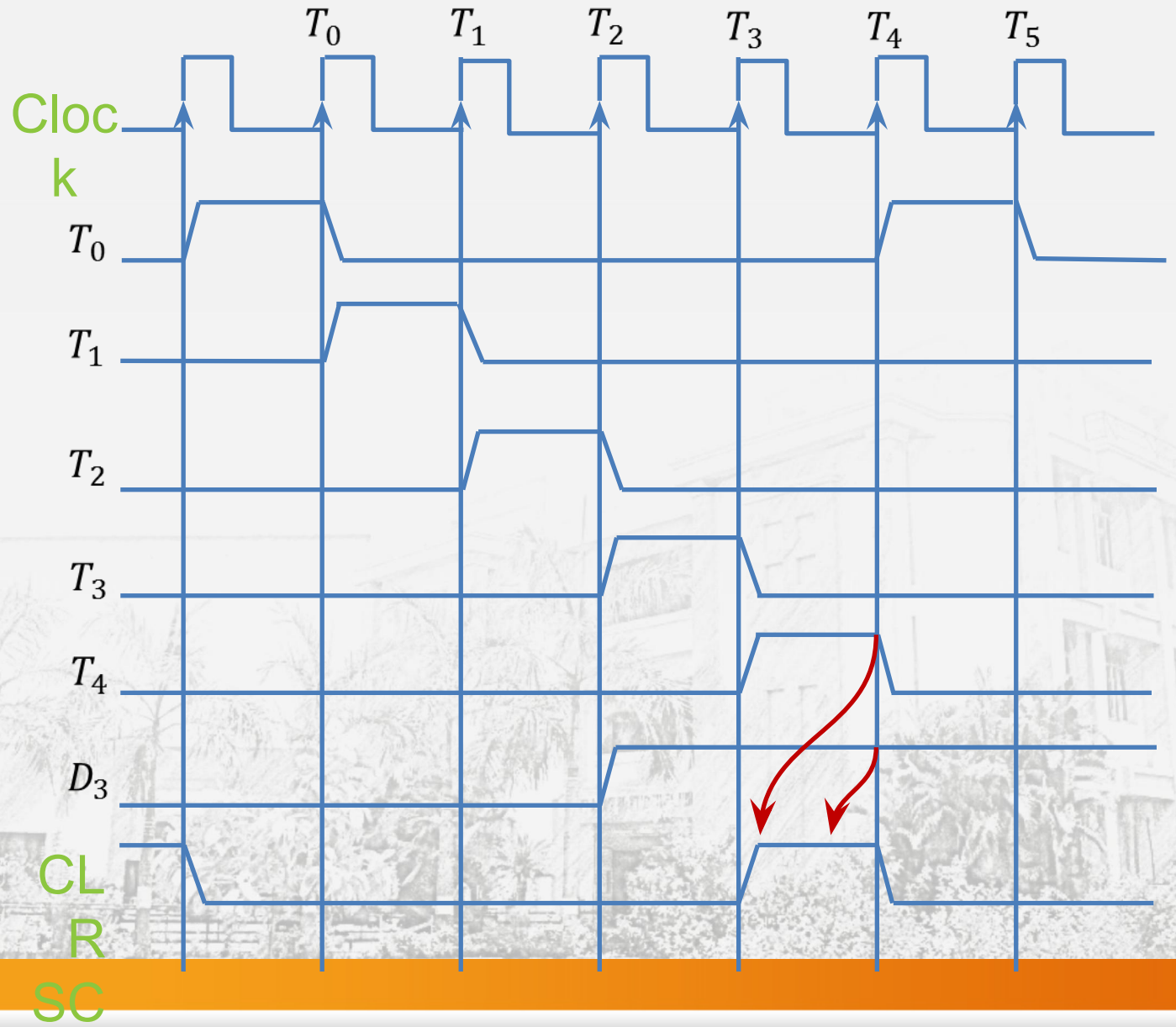
Control Unit

- As an example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3 and T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically by the statement

$$D_3T_4: \ SC \leftarrow 0$$

- Initially, the CLR input of SC is active.
- The first positive transition of the clock clears SC to 0, which in turn activates the timing T_0 out of the decoder.
- T_0 is active during one clock cycle.
- The positive clock transition labeled T_0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T_0.
- SC is incremented with every positive clock transition, unless its CLR input is active.
- This procedures the sequence of timing signals T_0, T_1, T_2, T_3 and T_4, and so on. If SC is not cleared, the timing signals will continue with T_5, T_6, up to T_{15} and back to T_0.

Timing Cycle for D_3T_4: SC ← 0



Control Unit

- The last three waveforms shows how SC is cleared when $D_3T_4 = 1$.
- Output D_3 from the operation decoder becomes active at the end of timing signal T_2.
- When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 becomes active.
- This signal is applied to the CLR input of SC.
- On the next positive clock transition the counter is cleared to 0.
- This causes the timing signal T_0 to become active instead of T_5 that would have been active if SC were incremented instead of cleared.

Control Organization

- Hardwired Control
  - The control logic is implemented with gates, flips-flops, decoders and other digital circuits.
  - It can be optimized to produce a fast mode of operation.
  - It requires changes in the wiring among the various components if the design has to be modified or changed.
- Microprogrammed Control
  - The control information is stored in a control memory.
  - The control memory is programmed to initiate the required sequence of microoperations.
  - Any required changes or modifications can be done by updating the microprogram in control memory.

Instruction Cycle

A program residing in the memory unit of a computer consists of a sequence of instructions. These instructions are executed by the processor by going through a cycle for each instruction.

In a basic computer, each instruction cycle consists of the following phases:

1. Fetch instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory.
4. Execute the instruction.

Instruction Cycle

- Fetch & Decode
  - PC is loaded with the address of the first instruction in the program.
  - The microoperations for fetch and decode phases are as follows:

$$T_0 : AR \leftarrow PC$$

$$T_1 : IR \leftarrow M[AR], PC \leftarrow PC + 1$$

$$T_2 : D_0, \ldots, D_7 \leftarrow Decode\ IR(12-14), AR \leftarrow IR(0-11), I \leftarrow IR(15)$$

Instruction Cycle

- Determine the type of instruction
  - During time T_3, the control unit determines the type of instruction i.e. Memory reference, Register reference or Input-Output instruction.
  - If $D_7 = 1$ then instruction must be register reference or input-output else memory reference instruction.
- Instruction Cycle Flowchart

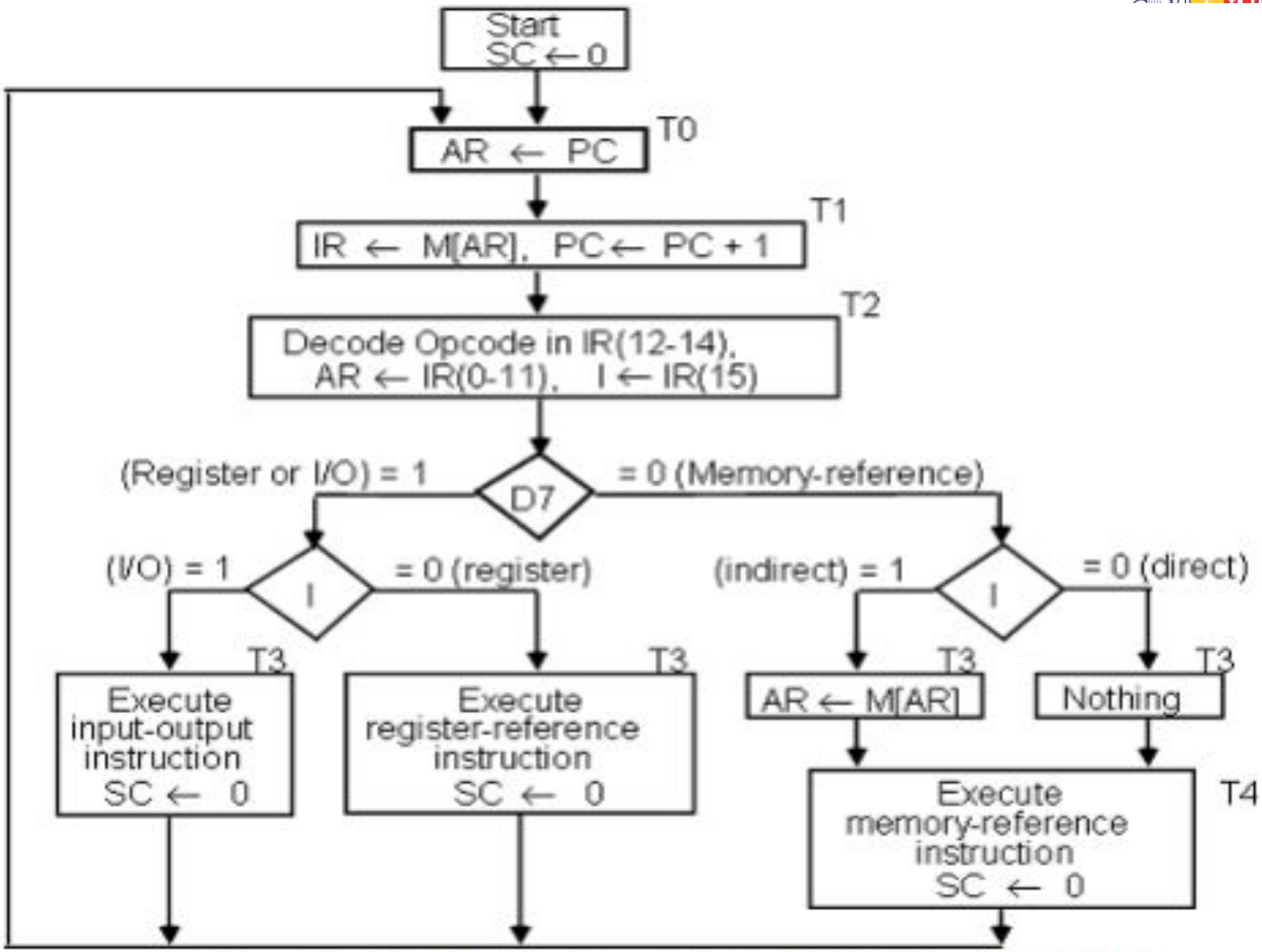


Figure 2.9: Flowchart for instruction cycle (initial configuration)

The flowchart represents an initial configuration for the instruction cycle and show how the control determines the instruction type after decoding.

If D7=1, the instruction must be register reference or input-output type. If D7=0, the operation code must be one of the other seven values 110, specifying a memory reference instruction. Control then inspects the value of the first bit of the instruction, which now available in flip flop.

If D7 = 0 and I = 1 we have memory reference instruction with indirect address. It is then necessary to read the effective address from memory.

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T3. this can be symbolized as follows:

D'7 | T3: AR<-M[AR]

D'7 |' T3: Nothing

D7 |' T3 Exe Register Ref instruction

D7 | T3: Exe input output instruction

encountered. It is not necessary to do anything since the effective address is already in AR

- However the sequence counter SC must be incremented when D'7|T3=1, so that the execution of the memory-reference instruction can be continued with timing Variable T4.
- A Reference register or input-output instruction can be executed with the click associated with timing signal T3. after the instruction is executed. Sc is cleared to 0 and control returns to the fetch phase with T0=1. SC is either incremented or cleared to 0 with every positive clock transition

Addressing Modes

Addressing Modes

- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
- Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:
  1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
  2. To reduce the number of bits in the addressing field of the instruction.
- There are basic 10 addressing modes supported by the computer.

Addressing Modes

1. Implied Mode
2. Immediate Mode
3. Register Mode
4. Register Indirect Mode
5. Autoincrement or Autodecrement Mode
6. Direct Address Mode
7. Indirect Address Mode
8. Relative Addressing Mode
9. Indexed Addressing Mode
10. Base Register Addressing Mode

1. Implied Mode

- Operands are specified *implicitly* in the definition of the instruction.
- For example, the instruction "complement accumulator (CMA)" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
- In fact, all register reference instructions that use an accumulator and zero address instructions are implied mode instructions.

2. Immediate Mode

- Operand is specified in the instruction itself.
- In other words, an immediate-mode instruction has an operand field rather than an address field.
- The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- Immediate mode of instructions is useful for initializing register to constant value.
- E.g. MOV R1, 05H

  instruction copies immediate number 05H to R1 register.

3. Register Mode

- Operands are in registers that reside within the CPU.
- The particular register is selected from a register field in the instruction.
- E.g. MOV AX,BX

  move value from BX to AX register

4. Register Indirect Mode

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.
- The advantage of this mode is that address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.
- E.g. MOV [R1], R2

  value of R2 is moved to the memory location specified in R1.

5. Autoincrement or Autodecrement Mode

- This is similar to the register indirect mode expect that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction.

6. Direct Address Mode

- In this mode the effective address is equal to the address part of the instruction.
- The operand resides in memory and its address is given directly by the address field of the instruction.
- E.g. ADD 457

7. Indirect Address Mode

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.
- The effective address in this mode is obtained from the following computational:

Effective address = address part of instruction + content of CPU register

8. Relative Address Mode

- In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number which can be either positive or negative.

Effective address = address part of instruction + content of PC

9. Indexed Addressing Mode

- In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- The indexed register is a special CPU register that contain an index value.
- The address field of the instruction defines the beginning address of a data array in memory.
- Each operand in the array is stored in memory relative to the begging address.

Effective address = address part of instruction + content of index register

10. Base Register Addressing Mode

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory.

Effective address = address part of instruction + content of base register

INSTRUCTION SET

Instruction sets are instruction codes to perform some task. It is classified into five categories

S.No.	Instruction & Description
1	Control Instructions
2	Logical Instructions
3	Branching Instructions
4	Arithmetic Instructions
5	Data Transfer Instructions

Control Instructions

Opcode	Operand	Meaning	Explanation
NOP	None	No operation	No operation is performed, i.e., the instruction is fetched and decoded.
HLT	None	Halt and enter wait state	The CPU finishes executing the current instruction and stops further execution. An interrupt or reset is necessary to exit from the halt state.
DI	None	Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts are disabled except TRAP.
EI	None	Enable interrupts	The interrupt enable flip-flop is set and all the interrupts are enabled.
RIM	None	Read interrupt mask	This instruction is used to read the status of interrupts
SIM	None	Set interrupt mask	This instruction is used to implement the interrupts

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.
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.
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.

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.
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.**
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

Branching instructions

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

Opcode	Description	Flag Status			
JC	Jump on Carry	CY=1	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.
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			

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 K.
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 K
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 55K
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 55K
LXI	Reg. pair, 16bit data	Load the register pair immediate	The instruction stores 16-bit data into the register pair designated in the operand. Example − LXI K, 3025M

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 K**
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 K
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 55K
SBI	8-bit data	Subtract the immediate from the accumulator with borrow	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
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 K

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 K**
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 K
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 K

DATA TRANSFER INSTRUCTION

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 K, L
MVI	Rd, data M, data	Move immediate 8-bit	The 8-bit data is stored in the destination register or memory. Example − MVI K, 55L
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 2034K
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 K

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 K, 3225L
STA	16-bit address	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example − STA 325K

OUT	**8-bit port address**	**Output the data from the accumulator to a port with 8bit address**	**The contents of the accumulator are copied into the I/O port specified by the operand.** **Example − OUT K9L**
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 − IN5KL