UNIT – III

Register Transfer Language and Micro operations: Register Transfer language, Register Transfer, Bus and memory transfers, Arithmetic Micro operations, logic micro operations, shift micro operations, Arithmetic logic shift unit.

Basic Computer Organization and Design: Instruction codes, Computer Registers Computer instructions, Timing and Control, Instruction cycle, Memory Reference Instructions, Input – Output and Interrupt.

Register Transfer Language (RTL)

Symbolic notation for internal data transfers and microoperations

Register Transfer Language describes transfers and microoperations inside a digital computer.

Registers are represented by symbolic names such as R1, R2, AC, PC and IR.

R2 ← R1 means the contents of R1 are transferred to R2.

A conditional transfer may be written as P: R2 ← R1; the operation occurs when P is active.

RTL provides a concise description of the internal operations used during instruction execution.

Basic Transfer

R2 ← R1
Destination receives source contents.

Conditional Transfer

P: R2 ← R1
Transfer occurs when P = 1.

Parallel Operations

R1 ← R2, R3 ← R4
Possible when hardware permits.

Register Transfer

Register-to-Register

Source register provides data; destination register loads it on the active clock edge.

Example: R2 ← R1

Control Signals

Control logic enables the source, selects a path, and loads the destination register.

Clocked Operation

Synchronous systems coordinate transfers with clock pulses.
The destination captures data at the specified clock event.

Register transfer is a fundamental microoperation used to move information within the CPU.

Transfers may use a common bus, multiplexer network, or dedicated connections.

Bus and Memory Transfers

Common Bus

A shared communication path among registers. Only the selected source drives the bus while the destination loads the value.

Memory Read

AR ← Address
DR ← M[AR]
Memory contents at the address in AR are loaded into DR.

Memory Write

AR ← Address
M[AR] ← DR
The contents of DR are stored in the addressed memory location.

Bus transfers reduce the number of direct connections required between registers and memory interfaces.

Arithmetic Microoperations

Arithmetic microoperations perform arithmetic on numeric data stored in registers.

Addition: R3 ← R1 + R2

Subtraction: R3 ← R1 − R2

Increment: R1 ← R1 + 1

Decrement: R1 ← R1 − 1

Other operations include add-with-carry and transfer through arithmetic circuits.

Add

R3 ← R1 + R2

Subtract

R3 ← R1 − R2

Increment

R1 ← R1 + 1

Logic Microoperations

AND

$R3 \leftarrow R1 \wedge R2$
Bit-by-bit AND.
Useful for masking.

OR

$R3 \leftarrow R1 \vee R2$
Sets selected bits.

XOR

$R3 \leftarrow R1 \oplus R2$
Detects differing bits or toggles bits.

Complement

$R2 \leftarrow \overline{R1}$
Inverts every bit.

Logic microoperations operate independently on corresponding bits.

Applications include masking, clearing, setting flags, and manipulating control fields.

Shift Microoperations

Logical Shift

Bits move left or right; vacated positions are filled with 0.

Example: 101101 → right → 010110

Arithmetic Shift

Preserves the sign bit during signed arithmetic shifts.
Useful for multiplication or division by powers of 2.

Circular Shift

Bits shifted out from one end re-enter at the other end.
Also called rotate.

Shift operations are useful for alignment, serial movement, bit extraction, and arithmetic scaling.

Left and right shifts can provide efficient multiplication or division by 2 for suitable representations.

Arithmetic Logic Shift Unit (ALSU)

Arithmetic Section

Addition, subtraction, increment, decrement and related arithmetic functions.

Logic Section

AND, OR, XOR, complement and other bitwise functions.

Shift Section

Logical, arithmetic and rotate operations; selected through control inputs.

Inputs

A, B / register data

Control

Operation select signals

ALSU

Selected operation

Output

Result to destination register

The ALSU combines arithmetic, logic and shift capabilities into one datapath unit controlled by operation-select signals.

Basic Computer Organization and Design

A basic computer consists of a processor, memory, input/output subsystem, registers, buses, and control circuitry.

The CPU executes instructions by transferring data between registers, memory and I/O devices.

The control unit generates timing and control signals to coordinate microoperations.

Registers provide fast temporary storage for addresses, instructions, operands and intermediate results.

A common bus system provides an organized path for transferring information.

CPU

ALU + Control Unit + Registers

Memory

Stores instructions and data

I/O

Communicates with external devices

Instruction Codes

Instruction Format

An instruction contains an opcode and, depending on the architecture, address or operand fields.

Opcode identifies the operation.

Basic Format

| I | Opcode | Address |

I: addressing mode indicator
Opcode: operation
Address: memory location / operand field

Addressing

Direct: address field gives effective address.

Indirect: address field points to a location containing the effective address.

Instruction codes provide the binary representation that the control unit decodes to determine the required sequence of microoperations.

Computer Registers

AR – Address Register

Holds the address of a memory location.

PC – Program Counter

Holds the address of the next instruction.

DR – Data Register

Holds data read from or written to memory.

AC – Accumulator

Stores operands and intermediate arithmetic/logic results.

IR – Instruction Register

Holds the instruction currently being decoded/executed.

TR – Temporary Register

Stores temporary intermediate information.

INPR / OUTR

Input and output registers interface with external devices.

Computer Instructions

Memory-Reference

Operate on data in memory.
Examples: AND, ADD, LDA, STA, BUN, BSA, ISZ.

Register-Reference

Operate directly on CPU registers.
Examples include clear AC, complement AC, increment AC and rotate operations.

Input–Output

Transfer data between CPU registers and I/O devices.
Examples include input, output, interrupt enable and disable.

The instruction decoder identifies the instruction class and the control unit generates the corresponding sequence of microoperations.

Timing, Control and Instruction Cycle

Fetch

AR ← PC
IR ← M[AR]
PC ← PC + 1
Fetch obtains the next instruction.

Decode

Decode opcode.
Determine addressing mode.
Generate required control signals.

Execute

Perform the instruction's microoperations. May access registers, memory or I/O.

Interrupt

If enabled and requested, save required state and service the interrupt before continuing.

Timing signals divide the instruction cycle into clocked steps such as T0, T1, T2 and later control states.

The control unit coordinates register transfers, memory operations, ALU functions and I/O actions.

Memory Reference Instructions

AND

AC ← AC ∧ M[EA]
AND accumulator with memory operand.

ADD

AC ← AC + M[EA]
Add memory operand to accumulator.

LDA

AC ← M[EA]
Load accumulator from memory.

STA

M[EA] ← AC
Store accumulator into memory.

BUN

PC ← EA
Unconditional branch.

BSA

Save return address and branch to a subroutine.

ISZ

Increment memory word; skip next instruction if result is zero.

Effective Address

EA is the effective address. Direct and indirect addressing determine how EA is obtained.

Input–Output, Interrupt and Unit–III Summary

Input–Output

Input register receives external data.
Output register holds data for an output device.
I/O instructions transfer information between CPU and external devices.

Interrupt

An interrupt requests CPU attention for an event. The CPU saves required state, services the interrupt, then resumes execution.

Interrupt Cycle

1. Complete current instruction
2. Save required state
3. Branch to service routine
4. Execute routine
5. Return to program

Unit summary: RTL → Register/Bus/Memory Transfers → Arithmetic/Logic/Shift Microoperations → ALSU → Basic Computer → Instruction Codes → Registers → Timing & Control → Instruction Cycle → Memory Reference → I/O & Interrupt.

Key idea: The control unit coordinates a sequence of microoperations to execute every machine instruction.