SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

COURSE NAME : 19IT301 COMPUTER ORGANIZATION AND ARCHITECTURE

II YEAR /III SEMESTER

Unit 1- BASIC STRUCTURES OF COMPUTER

Topic 1: FUNCTIONAL UNITS

COMPUTER ORGANISATION AND ARCHITECTURE

- 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 lumber, bricks, nails, and other building material
- While computer architecture looks at the design of the house.

UNIT-I INTRODUCTION

- Evolution of Computer Systems
- Computer Types
- Functional units
- Basic operational concepts
- Bus structures
- Memory location and addresses
- Memory operations
- Addressing modes
- Design of a computer system
- Instruction and instruction sequencing,
- RISC versus CISC.

INTRODUCTION

This chapter discusses the computer hardware, software and their interconnection, and it also discusses concepts like computer types, evolution of computers, functional units, basic operations, RISC and CISC systems.

Brief History of Computer Evolution

Two phases:

1. before VLSI 1945 – 1978
   - ENIAC
   - IAS
   - IBM
   - PDP-8
2. VLSI 1978 → present day
   - microprocessors !

VLSI = Very Large Scale Integration

Evolution of Computers

FIRST GENERATION (1945 – 1955)

- Program and data reside in the same memory (stored program concepts – John von Neumann)
- ALP was made used to write programs
- Vacuum tubes were used to implement the functions (ALU & CU design)
- Magnetic core and magnetic tape storage devices are used
- Using electronic vacuum tubes, as the switching components

SECOND GENERATION (1955 – 1965)

- Transistor were used to design ALU & CU
- HLL is used (FORTRAN)
- To convert HLL to MLL compiler were used
- Separate I/O processor were developed to operate in parallel with CPU, thus improving the performance
- Invention of the transistor which was faster, smaller and required considerably less power to operate

THIRD GENERATION (1965-1975)

- IC technology improved
- Improved IC technology helped in designing low cost, high speed processor and memory modules
- Multiprogramming, pipelining concepts were incorporated
- DOS allowed efficient and coordinate operation of computer system with multiple users
- Cache and virtual memory concepts were developed
- More than one circuit on a single silicon chip became available

FOURTH GENERATION (1975-1985)

- CPU – Termed as microprocessor
- INTEL, MOTOROLA, TEXAS,NATIONAL semiconductors started developing microprocessor
- Workstations, microprocessor (PC) & Notebook computers were developed
- Interconnection of different computer for better communication LAN,MAN,WAN
- Computational speed increased by 1000 times
- Specialized processors like Digital Signal Processor were also developed

BEYOND THE FOURTH GENERATION (1985 – TILL DATE)

- E-Commerce, E- banking, home office
- ARM, AMD, INTEL, MOTOROLA
- High speed processor - GHz speed
- Because of submicron IC technology lot of added features in small size

COMPUTER TYPES

Computers are classified based on the parameters like

- Speed of operation
- Cost
- Computational power
- Type of application

DESK TOP COMPUTER

- Processing &storage units, visual display &audio uits, keyboards
- Storage media-Hard disks, CD-ROMs
- Eg: Personal computers which is used in homes and offices
- Advantage: Cost effective, easy to operate, suitable for general purpose educational or business application

NOTEBOOK COMPUTER

- Compact form of personal computer (laptop)
- Advantage is portability

WORK STATIONS

- More computational power than PC
- Costlier
- Used to solve complex problems which arises in engineering application (graphics, CAD/CAM etc)

ENTERPRISE SYSTEM (MAINFRAME)

- More computational power
- Larger storage capacity
- Used for business data processing in large organization
- Commonly referred as servers or super computers

SERVER SYSTEM

- Supports large volumes of data which frequently need to be accessed or to be modified
- Supports request response operation

SUPER COMPUTERS

- Faster than mainframes
- Helps in calculating large scale numerical and algorithm calculation in short span of time
- Used for aircraft design and testing, military application and weather forecasting

HANDHELD

- Also called a PDA (Personal Digital Assistant).
- A computer that fits into a pocket, runs on batteries, and is used while holding the unit in your hand.
- Typically used as an appointment book, address book, calculator, and notepad.
- Can be synchronized with a personal microcomputer as a backup.

Basic Terminology

- Computer
  - A device that accepts input, processes data, stores data, and produces output, all according to a series of stored instructions.
- Hardware
  - Includes the electronic and mechanical devices that process the data; refers to the computer as well as peripheral devices.
- Software
  - A computer program that tells the computer how to perform particular tasks.
- Network
  - Two or more computers and other devices that are connected, for the purpose of sharing data and programs.
- Peripheral devices
  - Used to expand the computer’s input, output and storage capabilities.

Basic Terminology

- Input
  - Whatever is put into a computer system.
- Data
  - Refers to the symbols that represent facts, objects, or ideas.
- Information
  - The results of the computer storing data as bits and bytes; the words, numbers, sounds, and graphics.
- Output
  - Consists of the processing results produced by a computer.
- Processing
  - Manipulation of the data in many ways.
- Memory
  - Area of the computer that temporarily holds data waiting to be processed, stored, or output.
- Storage
  - Area of the computer that holds data on a permanent basis when it is not immediately needed for processing.

Basic Terminology

- Assembly language program (ALP) – Programs are written using mnemonics

- Mnemonic – Instruction will be in the form of English like form

- Assembler – is a software which converts ALP to MLL (Machine Level Language)

- HLL (High Level Language) – Programs are written using English like statements

- Compiler - Convert HLL to MLL, does this job by reading source program at once

Basic Terminology

- Interpreter – Converts HLL to MLL, does this job statement by statement

- System software – Program routines which aid the user in the execution of programs eg: Assemblers, Compilers

- Operating system – Collection of routines responsible for controlling and coordinating all the activities in a computer system

Computing Systems

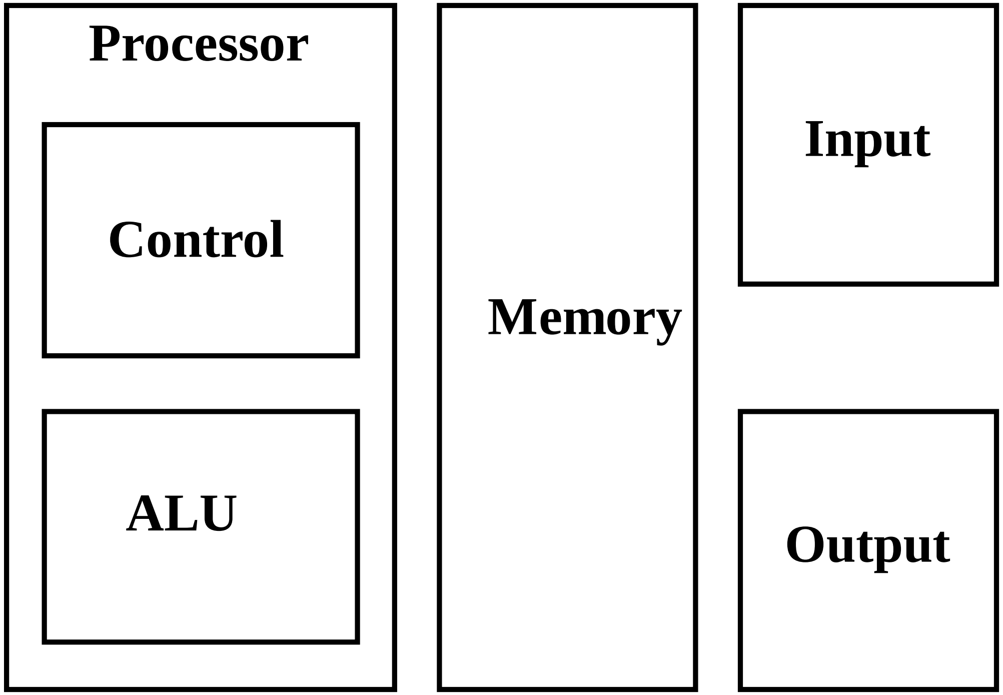
Computers have two kinds of components:

- *Hardware*, consisting of its physical devices (CPU, memory, bus, storage devices, ...)
- *Software*, consisting of the programs it has (Operating system, applications, utilities, ...)

FUNCTIONAL UNITS OF COMPUTER

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

The Big Picture

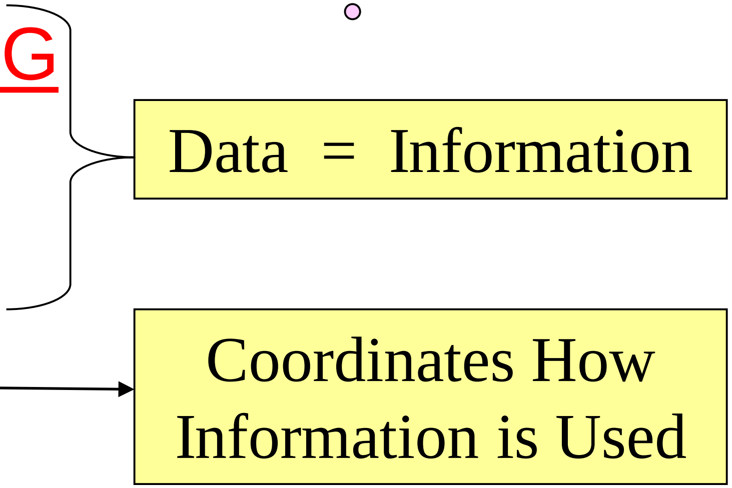


Since 1946 all computers have had 5 components!!!

Function

IMPORTANT SLIDE !

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



Data = Information

Coordinates How Information is Used

- NOTHING ELSE!

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

CPU

- 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

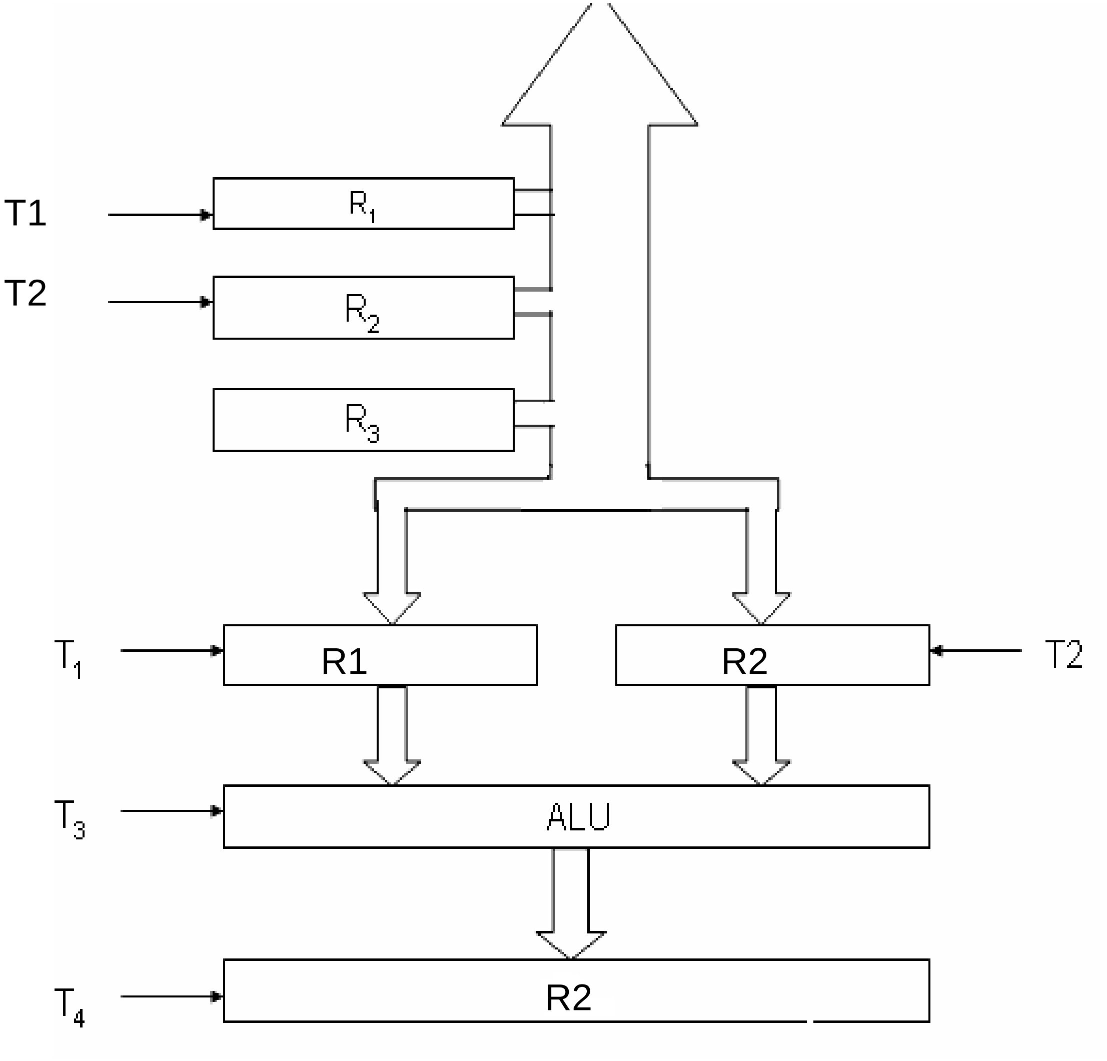
Example
Add R1, R2

T1 → Enable R1

T2 → Enable R2

T3 → Enable ALU for addition operation

T4 → Enable out put of ALU to store result of the operation



- Control unit works with a reference signal called processor clock

- Processor divides the operations into basic steps

- Each basic step is executed in one clock cycle

MEMORY

- Stores data, results, programs

- Two class of storage
(i) Primary (ii) Secondary

- Two types are RAM or R/W memory and ROM read only memory

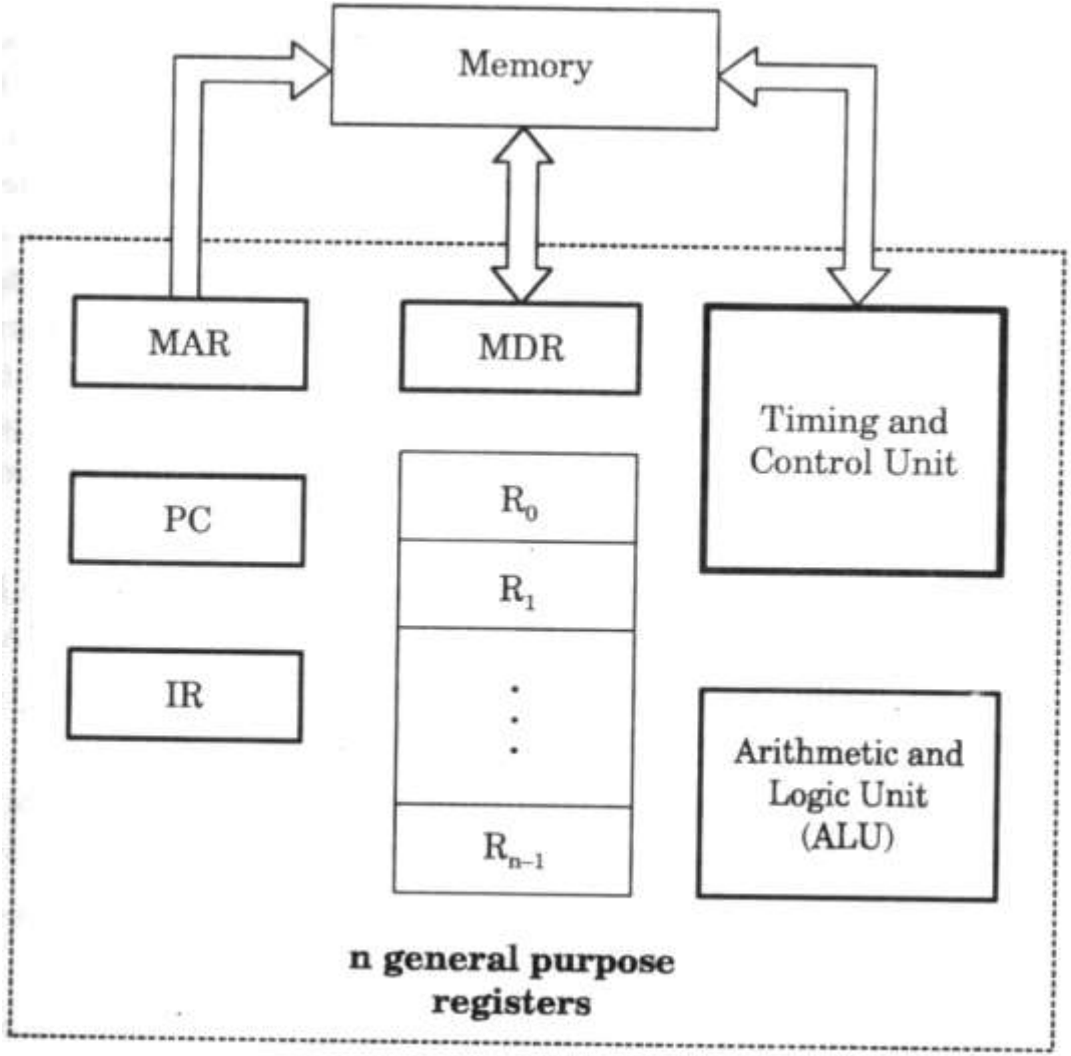
- ROM is used to store data and program which is not going to change.

- Secondary storage is used for bulk storage or mass storage

Basic Operational Concepts

Basic Function of Computer

- To Execute a given task as per the appropriate program
- Program consists of list of instructions stored in memory



Interconnection between Processor and Memory

Registers

Registers are fast stand-alone storage locations that hold data temporarily. Multiple registers are needed to facilitate the operation of the CPU. Some of these registers are

❑ **Two registers-MAR (Memory Address Register) and MDR (Memory Data Register) : To handle the data transfer between main memory and processor. MAR-Holds addresses, MDR-Holds data**

❑ **Instruction register (IR) : Hold the Instructions that is currently being executed**

❑ **Program counter: Points to the next instructions that is to be fetched from memory**

- (PC) → (MAR)(the contents of PC transferred to MAR)

- (MAR) → (Address bus) Select a particular memory location

- Issues RD control signals

- Reads instruction present in memory and loaded into MDR

- Will be placed in IR (Contents transferred from MDR to IR)

- Instruction present in IR will be decoded by which processor understand what operation it has to perform

- Increments the contents of PC by 1, so that it points to the next instruction address

- If data required for operation is available in register, it performs the operation

- If data is present in memory following sequence is performed

- Address of the data → MAR

- MAR → Address bus → select memory location where is issued RD signal

- Reads data via data bus → MDR

- From MDR data can be directly routed to ALU or it can be placed in register and then operation can be performed

- Results of the operation can be directed towards output device, memory or register

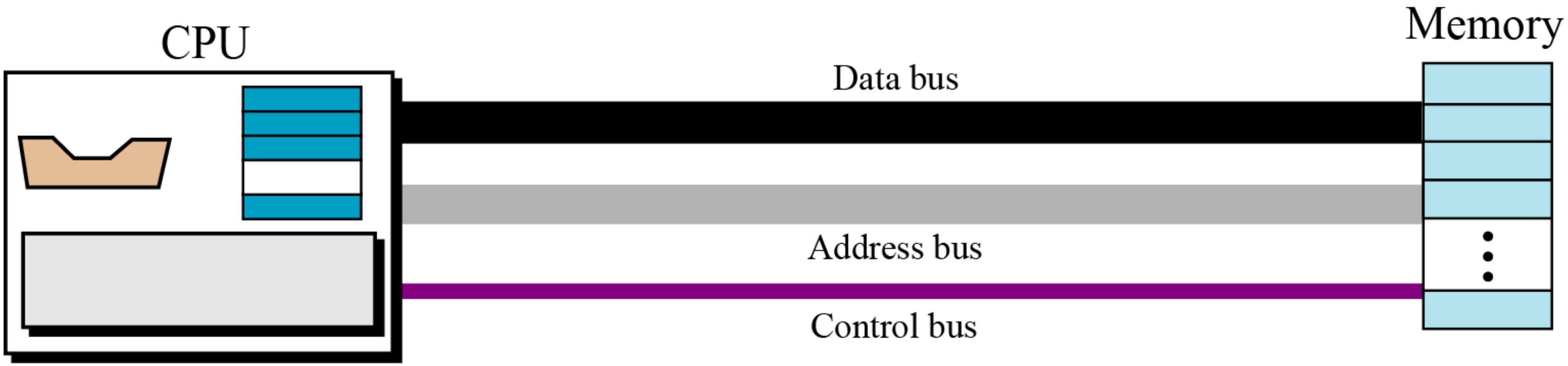
- Normal execution preempted (interrupt)

Interrupt

- An interrupt is a request from I/O device for service by processor
- Processor provides requested service by executing interrupt service routine (ISR)
- Contents of PC, general registers, and some control information are stored in memory .
- When ISR completed, processor restored, so that interrupted program may continue

BUS STRUCTURE
Connecting CPU and memory

The CPU and memory are normally connected by three groups of connections, each called a **bus**: *data bus*, *address bus* and *control bus*

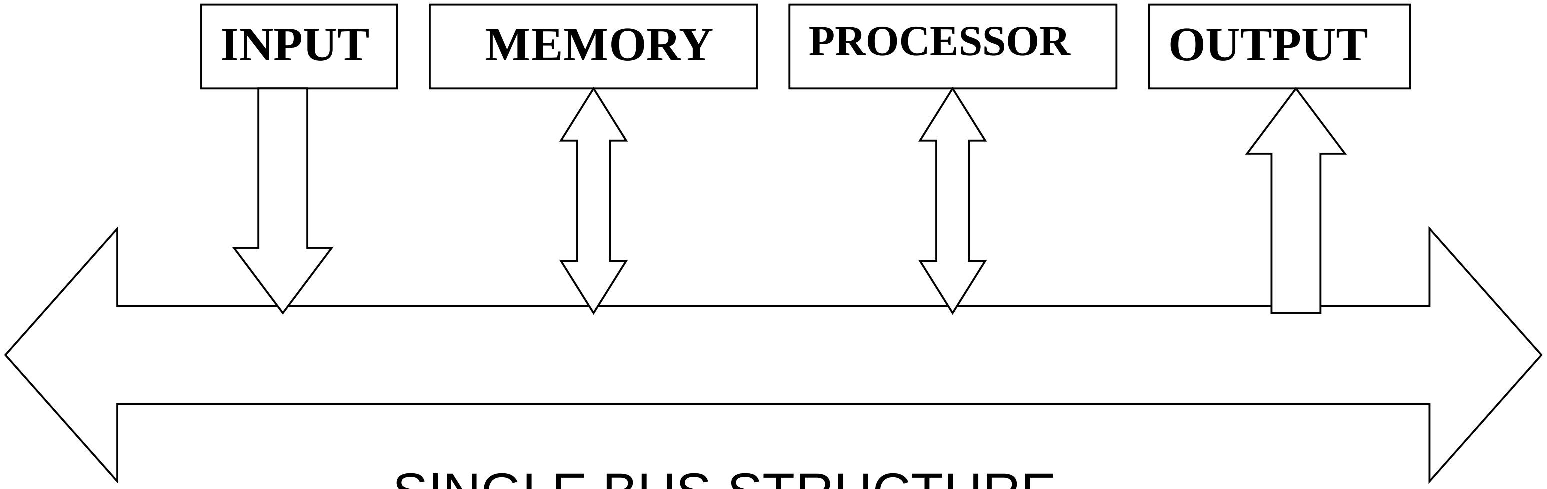


Connecting CPU and memory using three buses

BUS STRUCTURE

•Group of wires which carries information form CPU to peripherals or vice – versa

•**Single bus structure**: Common bus used to communicate between peripherals and microprocessor



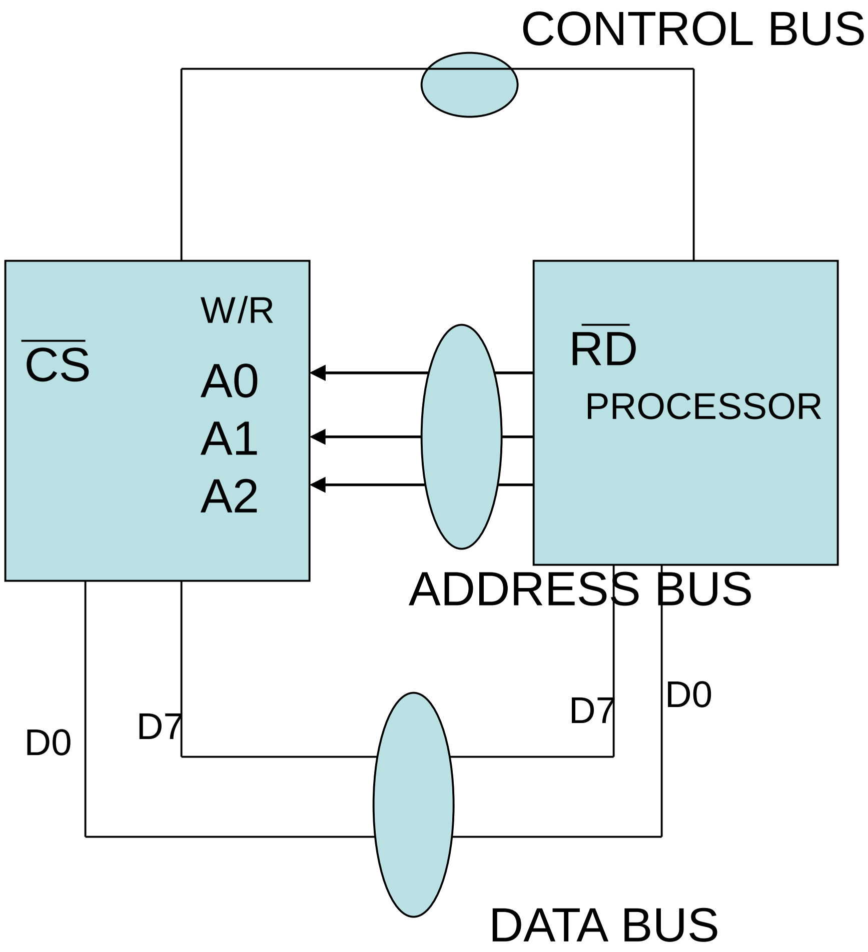
SINGLE BUS STRUCTURE

Continued:-

- To improve performance **multi-bus** structure can be used

- In two – bus structure : One bus can be used to fetch instruction other can be used to fetch data, required for execution.

- Thus improving the performance ,but cost increases



A_2	A_1	A_0	Selected location
0	0	0	0th Location
0	0	1	1st Location
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Cont:-

•2^3 = 8 i.e. 3 address line is required to select 8 location

•In general 2^x = n where x number of address lines (address bit) and n is number of location

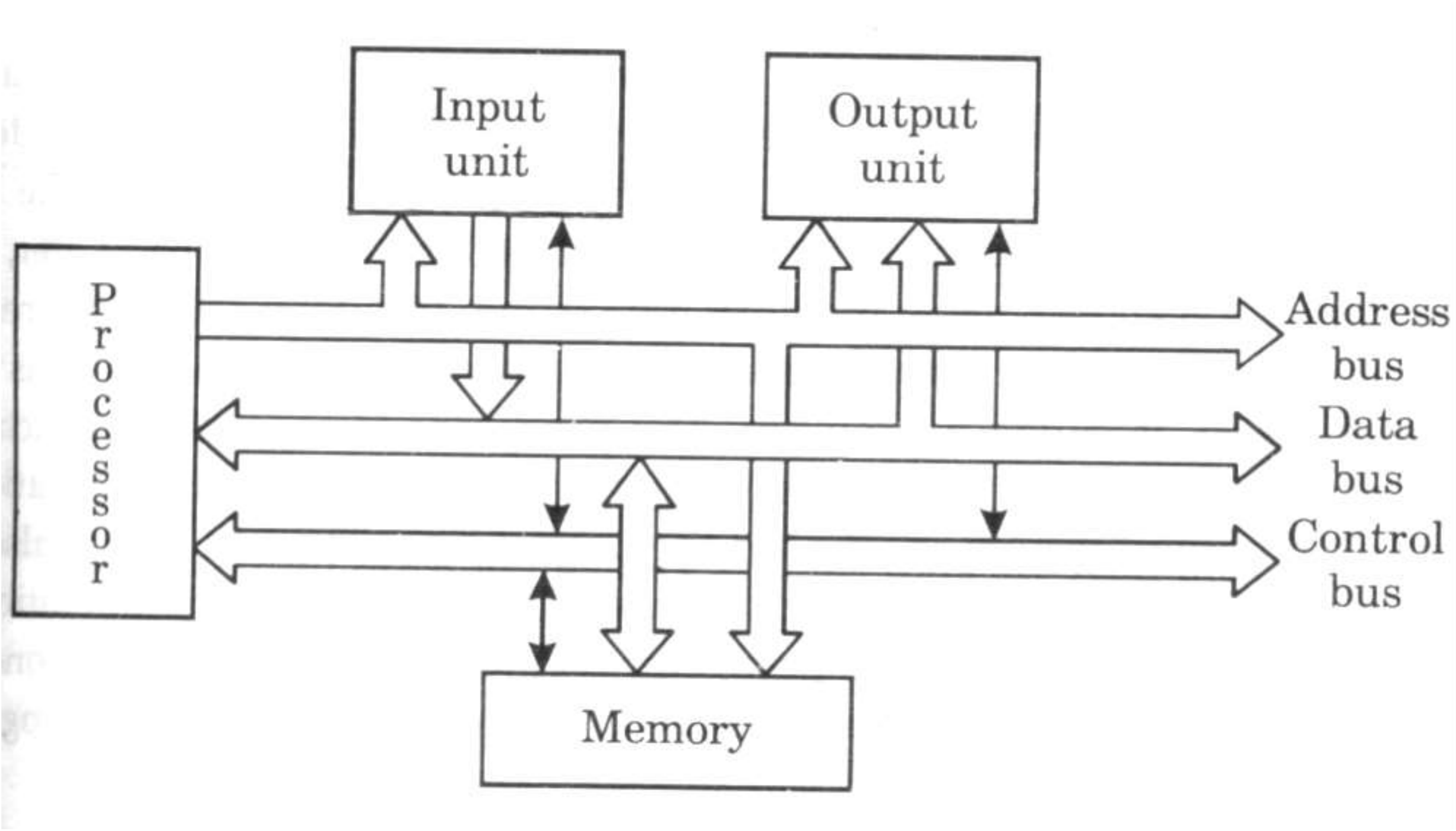
•**Address bus** : unidirectional : group of wires which carries address information bits form processor to peripherals (16,20,24 or more parallel signal lines)

Cont:-

- **Databus**: bidirectional : group of wires which carries data information bit form processor to peripherals and vice – versa

- **Controlbus**: bidirectional: group of wires which carries control signals form processor to peripherals and vice – versa

- Figure below shows address, data and control bus and their connection with peripheral and microprocessor



Single bus structure showing the details of connection

PERFORMANCE

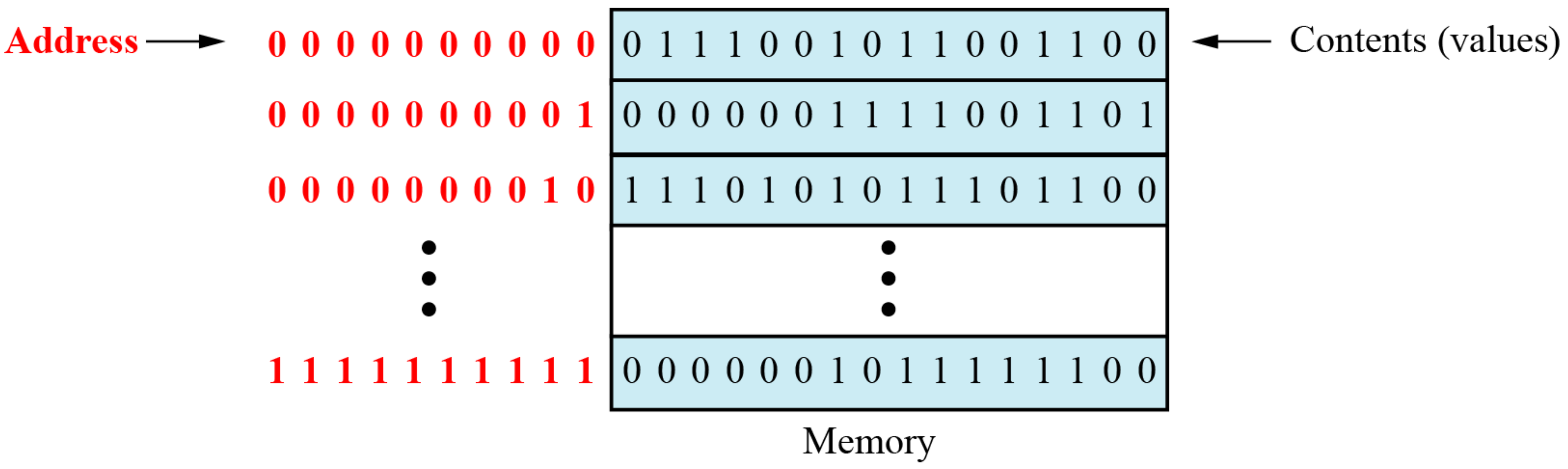
- Time taken by the system to execute a program
- Parameters which influence the performance are
  - Clock speed
  - Type and number of instructions available
  - Average time required to execute an instruction
  - Memory access time
  - Power dissipation in the system
  - Number of I/O devices and types of I/O devices connected
  - The data transfer capacity of the bus

MEMORY LOCATIONS AND ADDRESSES

- **Main memory** is the second major subsystem in a computer. It consists of a collection of storage locations, each with a unique identifier, called an **address**.

- Data is transferred to and from memory in groups of bits called **words**. A word can be a group of 8 bits, 16 bits, 32 bits or 64 bits (and growing).

- If the word is 8 bits, it is referred to as a **byte**. The term “byte” is so common in computer science that sometimes a 16-bit word is referred to as a 2-byte word, or a 32-bit word is referred to as a 4-byte word.



Address →	← Contents (values)
0 0 0 0 0 0 0 0 0 0	0 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 1 1 1 1 0 0 1 1 0 1
0 0 0 0 0 0 0 0 1 0	1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 0
⋮	⋮
1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0

Memory

Figure 5.3 Main memory

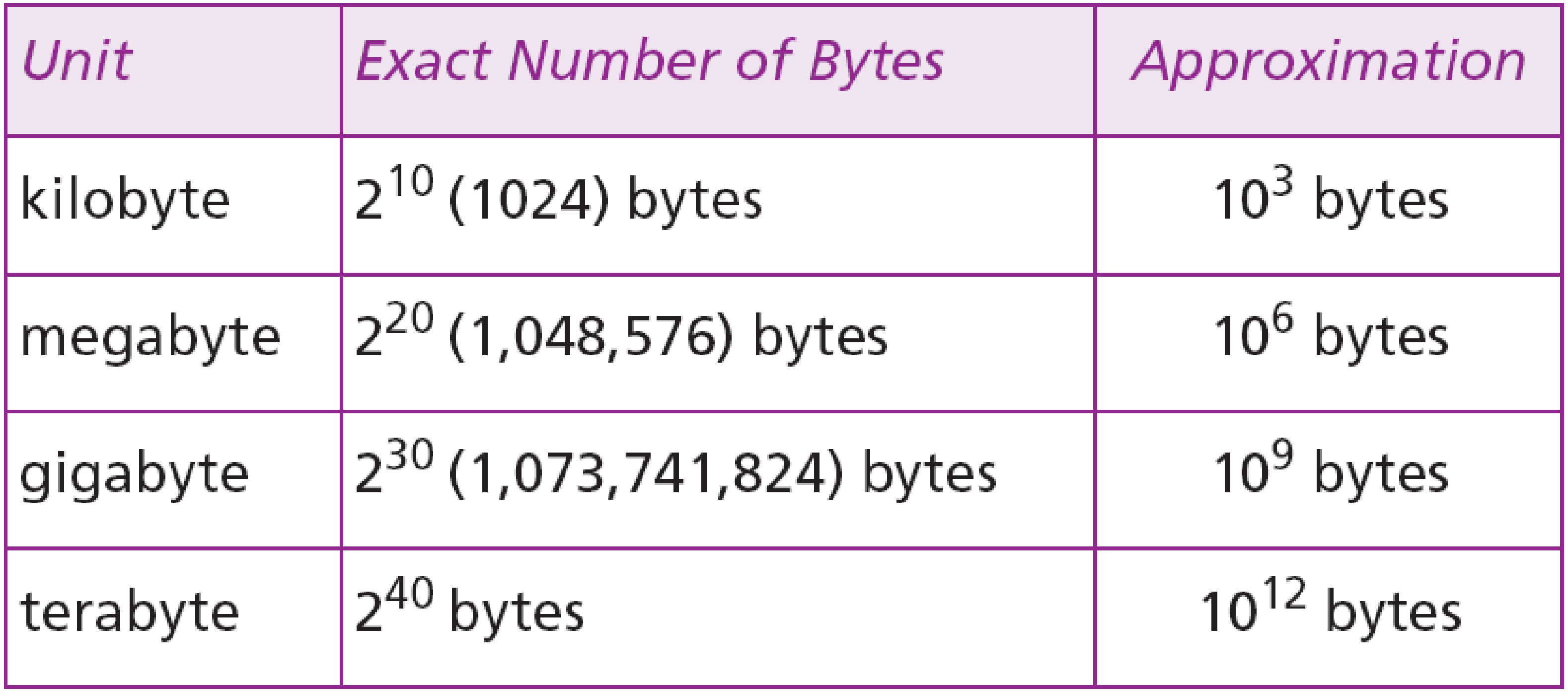
Address space

•To access a word in memory requires an identifier. Although programmers use a name to identify a word (or a collection of words), at the hardware level each word is identified by an address.

•The total number of uniquely identifiable locations in memory is called the **address space**.

•For example, a memory with 64 kilobytes (16 address line required) and a word size of 1 byte has an address space that ranges from 0 to 65,535.

Table 5.1 Memory units



Unit	*Exact Number of Bytes*	*Approximation*
kilobyte	2^{10} (1024) bytes	10^3 bytes
megabyte	2^{20} (1,048,576) bytes	10^6 bytes
gigabyte	2^{30} (1,073,741,824) bytes	10^9 bytes
terabyte	2^{40} bytes	10^{12} bytes

Memory addresses are defined using unsigned binary integers.

Example 1

A computer has 32 MB (megabytes) of memory. How many bits are needed to address any single byte in memory?

Solution

The memory address space is 32 MB, or 2^{25} ($2^5 \times 2^{20}$). This means that we need $\log_2 2^{25}$, or **25 bits**, to address each byte.

Example 2

A computer has 128 MB of memory. Each word in this computer is eight bytes. How many bits are needed to address any single word in memory?

Solution

The memory address space is 128 MB, which means 2^{27}. However, each word is eight (2^3) bytes, which means that we have 2^{24} words. This means that we need $\log_2 2^{24}$, or **24 bits**, to address each word.

Assignment of byte addresses

- Little Endian (e.g., in DEC, Intel)
  - » low order byte stored at lowest address
  - » byte0 byte1 byte2 byte3

- Eg: 46,78,96,54 (32-bit data)
- H BYTE ← L BYTE

- 8000
- 8001
- 8002
- 8003
- 8004

54

96

78

46

Big Endian

- Big Endian (e.g., in IBM, Motorolla, Sun, HP)
  - » high order byte stored at lowest address
  - » byte3 byte2 byte1 byte0

- Programmers/protocols should be careful when transferring binary data between Big Endian and Little Endian machines

- In case of 16-bit data, aligned words begin at byte addresses of 0,2,4,…………………………………….
- In case of 32-bit data, aligned words begin at byte address of 0,4,8,…………………………………….
- In case of 64-bit data, aligned words begin at byte addresses of 0,8,16,…………………………………..
- In some cases, words can start at an arbitrary byte address also then, we say that word locations are unaligned

MEMORY OPERATIONS

- Today, **general-purpose computers** use a set of instructions called a **program** to process data.

- A computer executes the program to create output data from input data

- Both program instructions and data operands are stored in memory

- Two basic operations requires in memory access
    - Load operation (Read or Fetch)-Contents of specified memory location are read by processor
    - Store operation (Write)- Data from the processor is stored in specified memory location

- INSTRUCTION SET ARCHITECTURE:- Complete instruction set of the processor

- BASIC 4 TYPES OF OPERATION:-
    - Data transfer between memory and processor register
    - Arithmetic and logic operation
    - Program sequencing and control
    - I/O transfer

Register transfer notation (RTN)

Transfer between processor registers & memory, between processor register & I/O devices

Memory locations, registers and I/O register names are identified by a symbolic name in uppercase alphabets

- LOC,PLACE,MEM are the address of memory location
- R1 , R2,… are processor registers
- DATA_IN, DATA_OUT are I/O registers

- Contents of location is indicated by using square brackets [ ]

- RHS of RTN always denotes a values, and is called Source

- LHS of RTN always denotes a symbolic name where value is to be stored and is called destination

- Source contents are not modified

- Destination contents are overwritten

Examples of RTN statements

- $R2 \leftarrow [LOCN]$

- $R4 \leftarrow [R3] +[R2]$

ASSEMBLY LANGUAGE NOTATION (ALN)

- RTN is easy to understand and but cannot be used to represent machine instructions
- Mnemonics can be converted to machine language, which processor understands using assembler

Eg:

1. MOVE LOCN, R2
2. ADD   R3, R2, R4

TYPE OF INSTRUCTION

- Three address instruction

  - Syntax: Operation source 1, source 2, destination
  - Eg: ADD D,E,F   where D,E,F are memory location
  - Advantage: Single instruction can perform the complete operation
  - Disadvantage : Instruction code will be too large to fit in one word location in memory

TWO ADDRESS INSTRUCTION

- Syntax : Operation source, destination

- Eg: MOVE E,F MOVE D,F

  ADD D,F OR ADD E,F

Perform ADD A,B,C using 2 instructions
MOVE B,C
ADD A,C

❖Disadvantage: Single instruction is not sufficient to perform the entire operation.

ONE ADDRESS INSTRUCTION

- Syntax- Operation source/destination
- In this type either a source or destination operand is mentioned in the instruction
- Other operand is implied to be a processor register called Accumulator
- Eg: ADD B (general)
- Load D; ACC ← [memlocation _D]
- ADD E; ACC ← (ACC) +(E)
- STORE F; memlocation_ F ← (ACC)

Zero address instruction

- Location of all operands are defined implicitly
- Operands are stored in a structure called pushdown stack

Continued

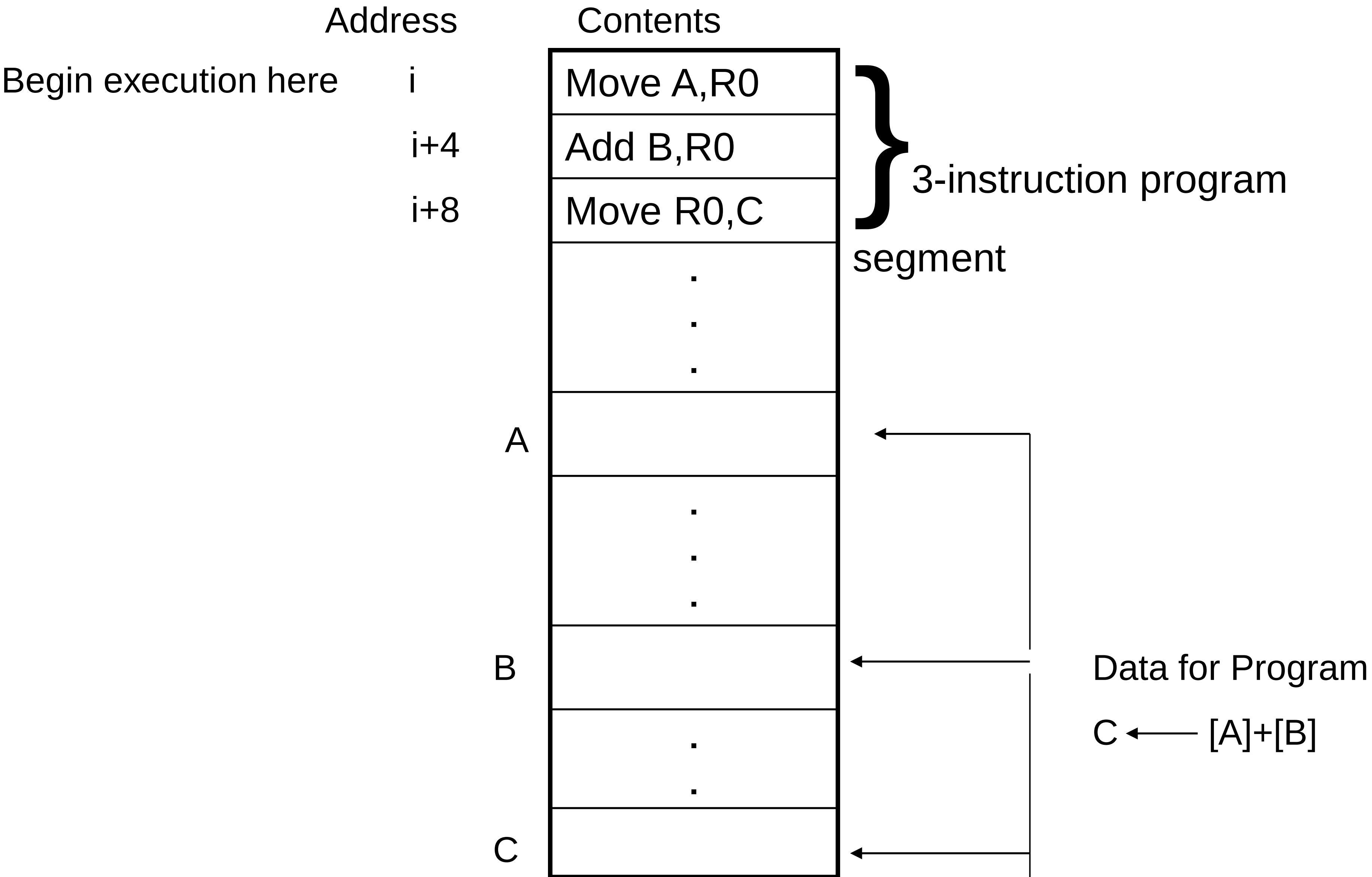
- If processor supports ALU operations one data in memory and other in register, then the instruction sequence is
- MOVE D, Ri
- ADD E, Ri
- MOVE Ri, F

- If processor supports ALU operations only with registers, then one has to follow the instruction sequence given below
- LOAD D, Ri
- LOAD E, Rj
- ADD Ri, Rj
- MOVE Rj, F

Basic Instruction Cycle

- Basic computer operation cycle
  - Fetch the instruction from memory into a control register (PC)
  - Decode the instruction
  - Locate the operands used by the instruction
  - Fetch operands from memory (if necessary)
  - Execute the operation in processor registers
  - Store the results in the proper place
  - Go back to step 1 to fetch the next instruction

INSTRUCTION EXECUTION & STRIAGHT LINE SEQUENCING



	Address	Contents
Begin execution here	i	Move A,R0
	i+4	Add B,R0
	i+8	Move R0,C
		. . .
	A	
		. . .
	B	
		. .
	C	

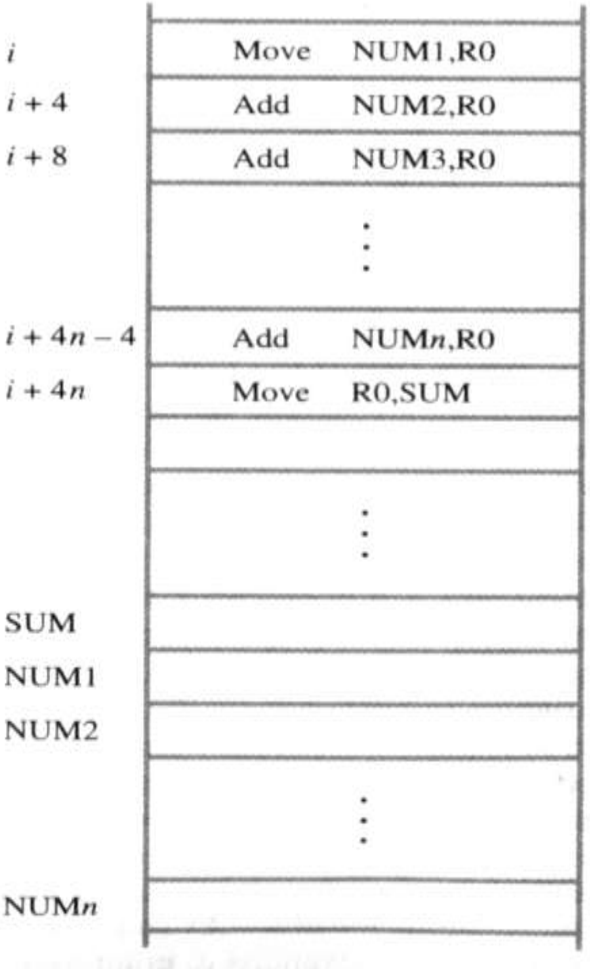
3-instruction program segment

Data for Program

C ← [A]+[B]

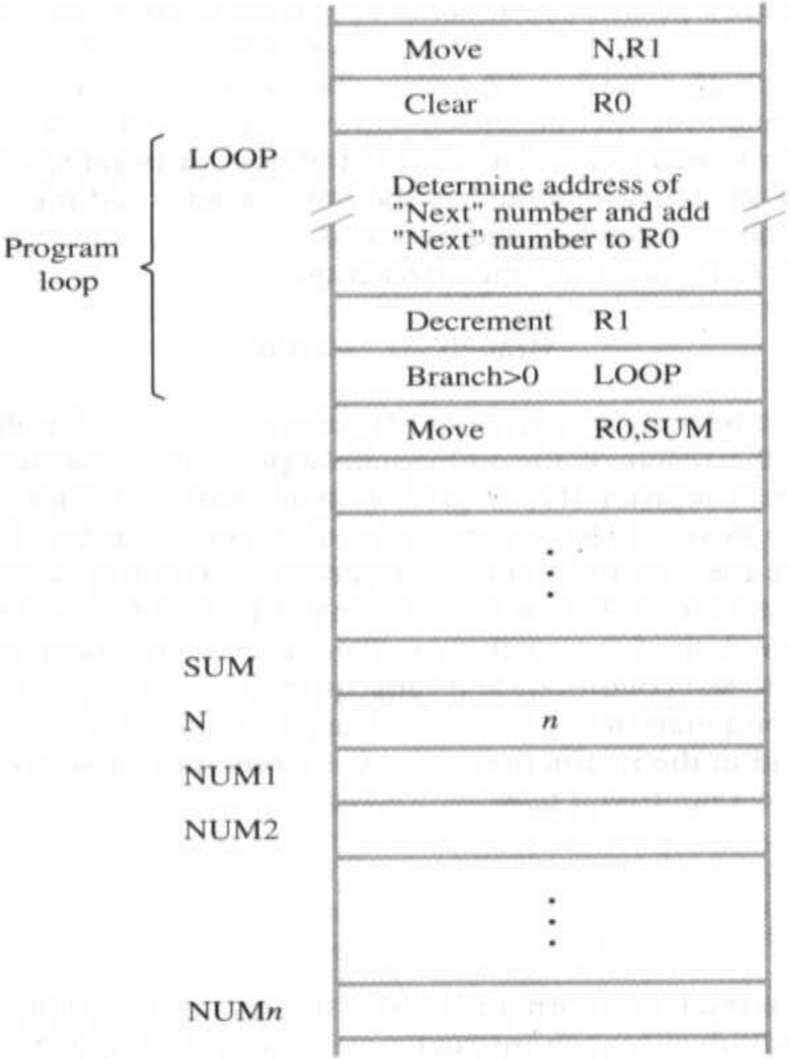
- PC – Program counter: hold the address of the next instruction to be executed
- Straight line sequencing: If fetching and executing of instructions is carried out one by one from successive addresses of memory, it is called straight line sequencing.
- Major two phase of instruction execution
- Instruction fetch phase: Instruction is fetched form memory and is placed in instruction register IR
- Instruction execute phase: Contents of IR is decoded and processor carries out the operation either by reading data from memory or registers.

BRANCHING



i	Move	NUM1,R0
i + 4	Add	NUM2,R0
i + 8	Add	NUM3,R0
	⋮	
i + 4*n* − 4	Add	NUM*n*,R0
i + 4*n*	Move	R0,SUM
	⋮	
SUM		
NUM1		
NUM2		
	⋮	
NUM*n*		

A straight line program for adding n numbers



		Move	N,R1
		Clear	R0
Program loop	LOOP	Determine address of "Next" number and add "Next" number to R0	
		Decrement	R1
		Branch>0	LOOP
		Move	R0,SUM
		⋮	
	SUM		
	N	*n*	
	NUM1		
	NUM2		
		⋮	
	NUM*n*		

Using a loop to add n numbers

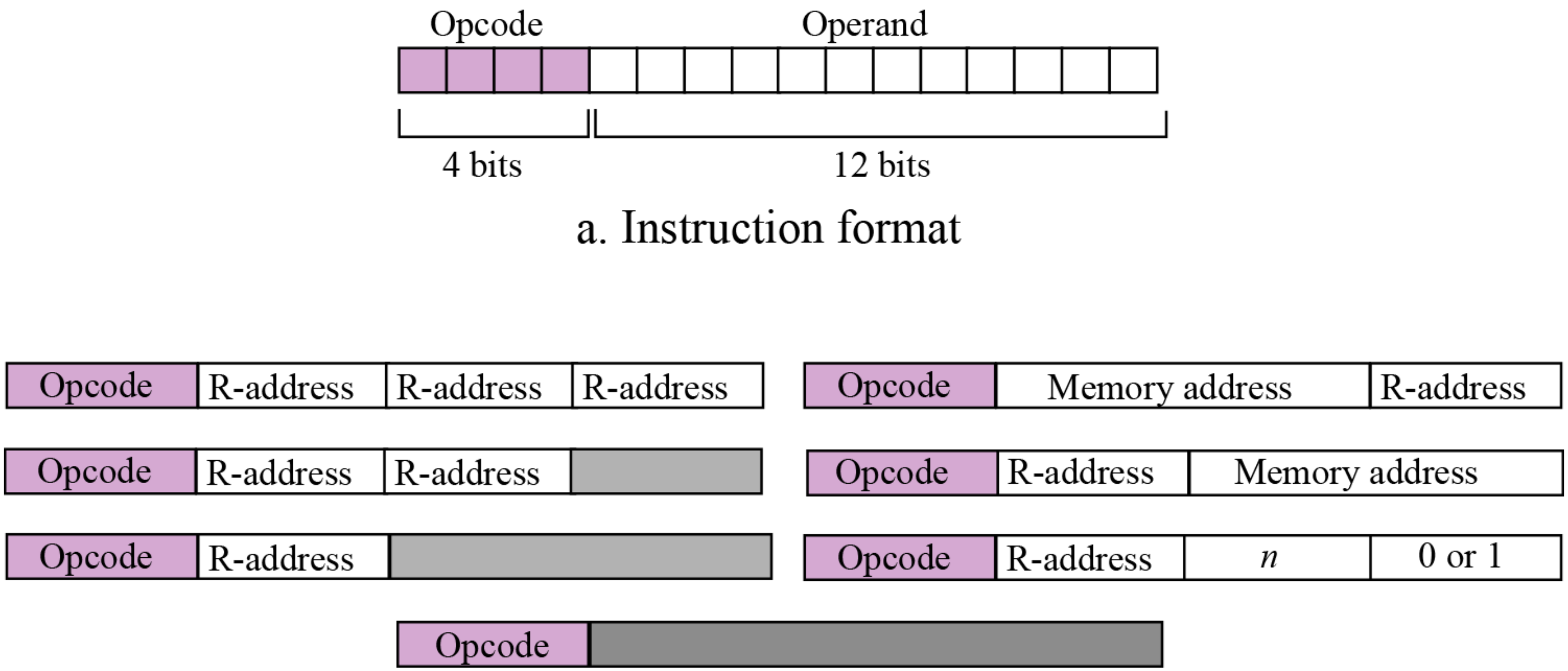
BRANCHING

- Branch instruction are those which changes the normal sequence of execution.
- Sequence can be changed either conditionally or unconditionally.
- Accordingly we have conditional branch instructions and unconditional branch instruction.
- Conditional branch instruction changes the sequence only when certain conditions are met.
- Unconditional branch instruction changes the sequence of execution irrespective of condition of the results.

CONDITION CODES

- CONDITIONAL CODE FLAGS: The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions

• N – Negative	1 if results are Negative 0 if results are Positive
• Z – Zero	1 if results are Zero 0 if results are Non zero
• V – Overflow	1 if arithmetic overflow occurs 0 non overflow occurs
• C – Carry	1 if carry and from MSB bit 0 if there is no carry from MSB bit



a. Instruction format

b. Instruction types

Figure **Format and different instruction types**

Processing the instructions

Simple computer, like most computers, uses machine cycles.

A cycle is made of three phases: fetch, decode and execute.

During the fetch phase, the instruction whose address is determined by the PC is obtained from the memory and loaded into the IR. The PC is then incremented to point to the next instruction.

During the decode phase, the instruction in IR is decoded and the required operands are fetched from the register or from memory.

During the execute phase, the instruction is executed, and the results are placed in the appropriate memory location or the register.

Once the third phase is completed, the control unit starts the cycle again, but now the PC is pointing to the next instruction.

The process continues until the CPU reaches a HALT instruction.

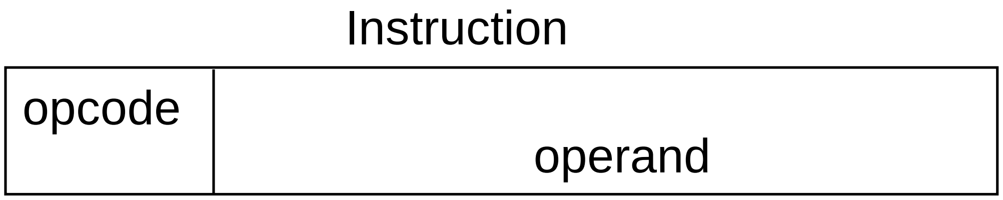
Types of Addressing Modes

The different ways in which the location of the operand is specified in an instruction are referred to as addressing modes

- Immediate Addressing
- Direct Addressing
- Indirect Addressing
- Register Addressing
- Register Indirect Addressing
- Relative Addressing
- Indexed Addressing

Immediate Addressing

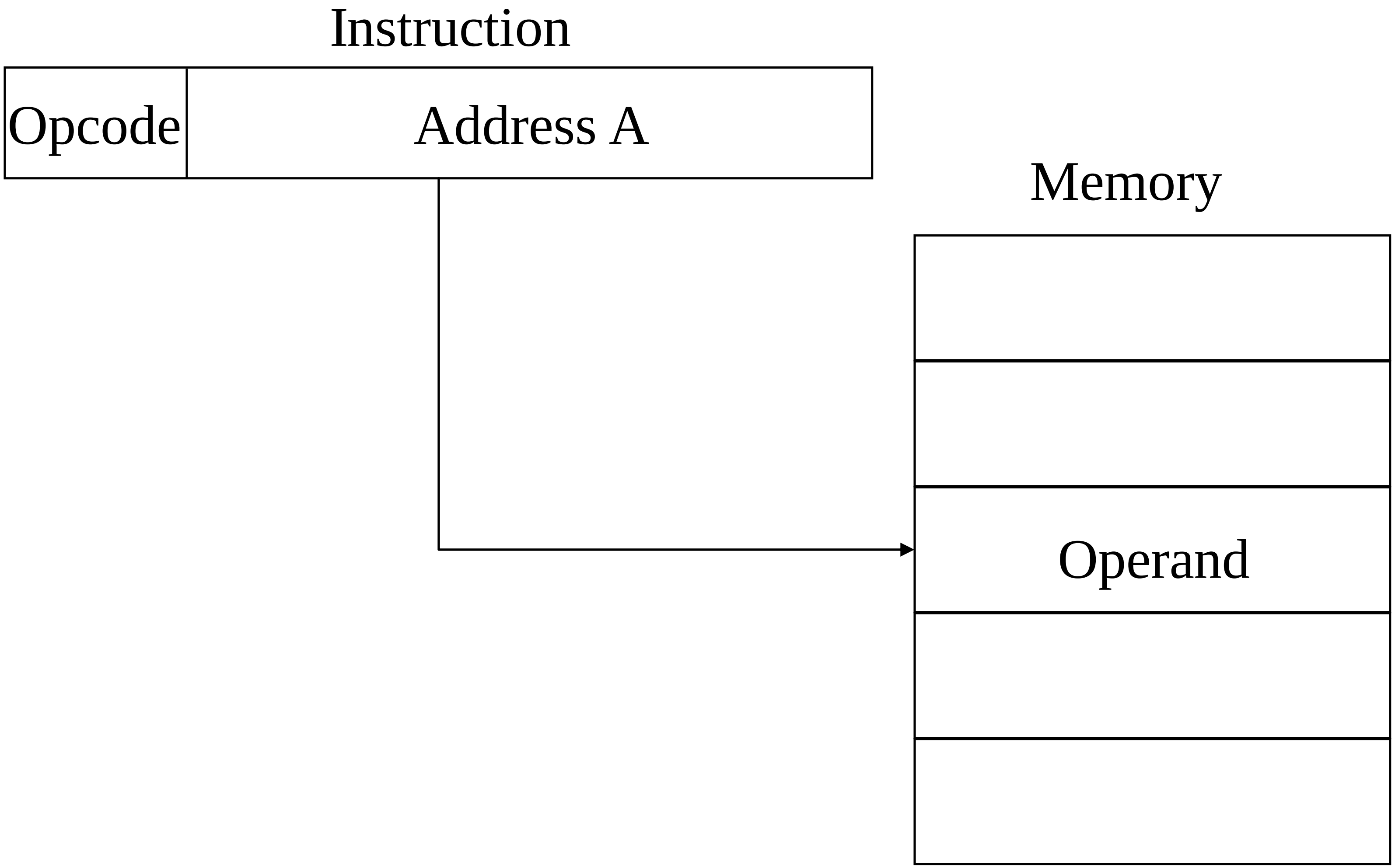
- Operand is given explicitly in the instruction
- Operand = Value
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



Direct Addressing

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

Direct Addressing Diagram



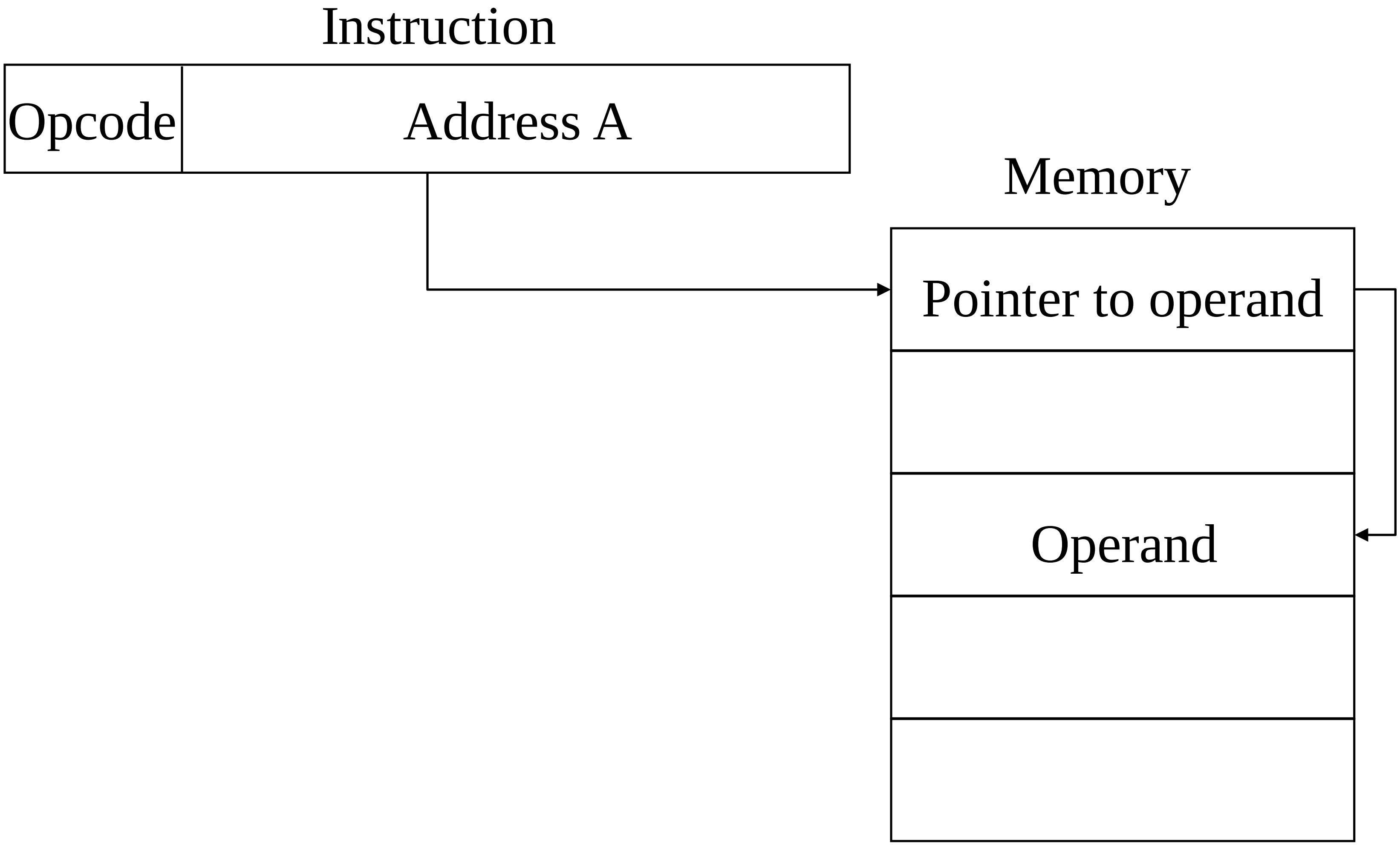
Indirect Addressing (1)

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA = [A]
  - Look in A, find address (A) and look there for operand
- e.g. ADD (A)
  - Add contents of cell pointed to by contents of A to accumulator

Indirect Addressing (2)

- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
    - Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower

Indirect Addressing Diagram



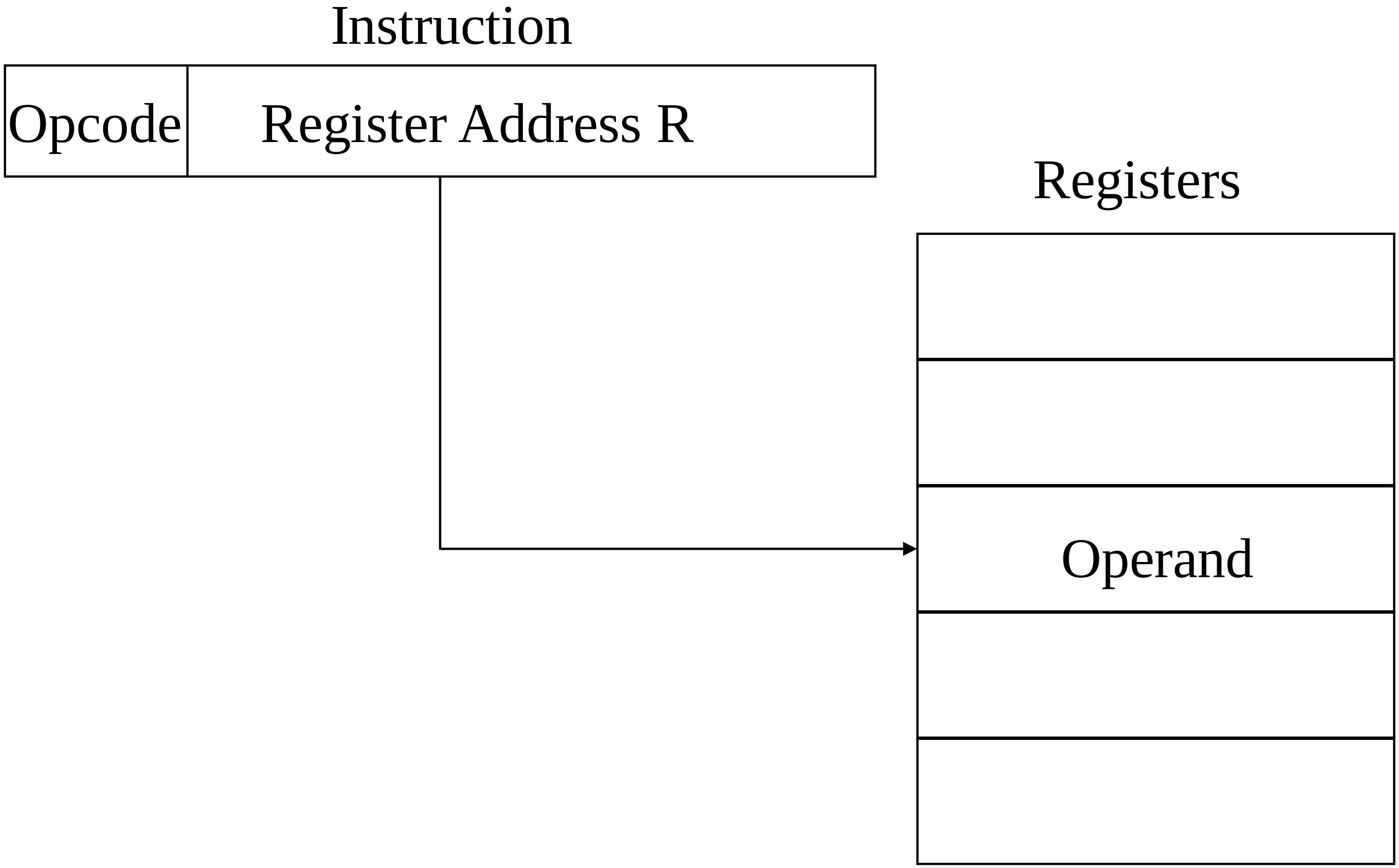
Register Addressing (1)

- Operand is held in register named in address field
- EA = R
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch

Register Addressing (2)

- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing

Register Addressing Diagram



Instruction

Opcode

Register Address R

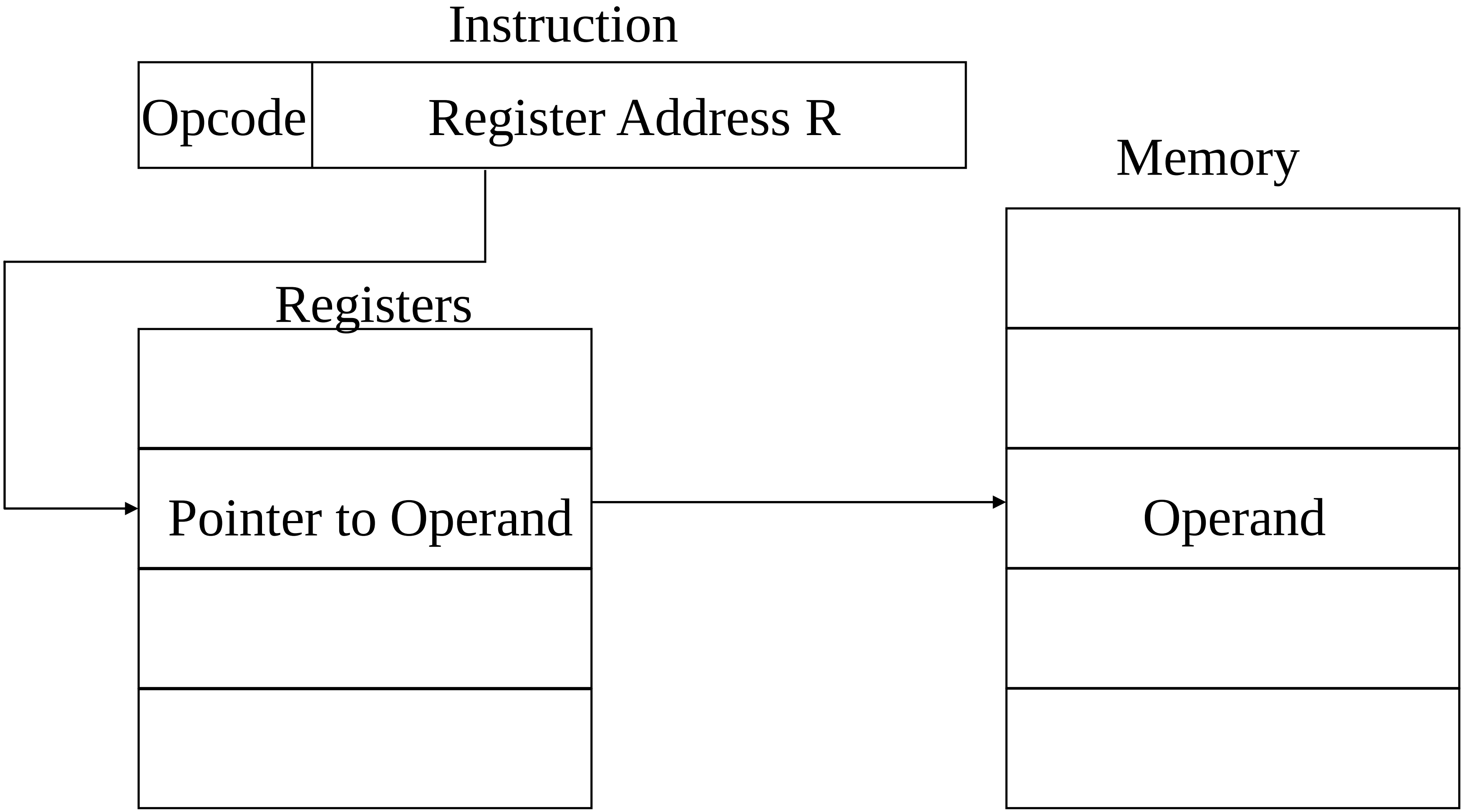
Registers

Operand

Register Indirect Addressing

- C.f. indirect addressing
- EA = [R]
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing

Register Indirect Addressing Diagram



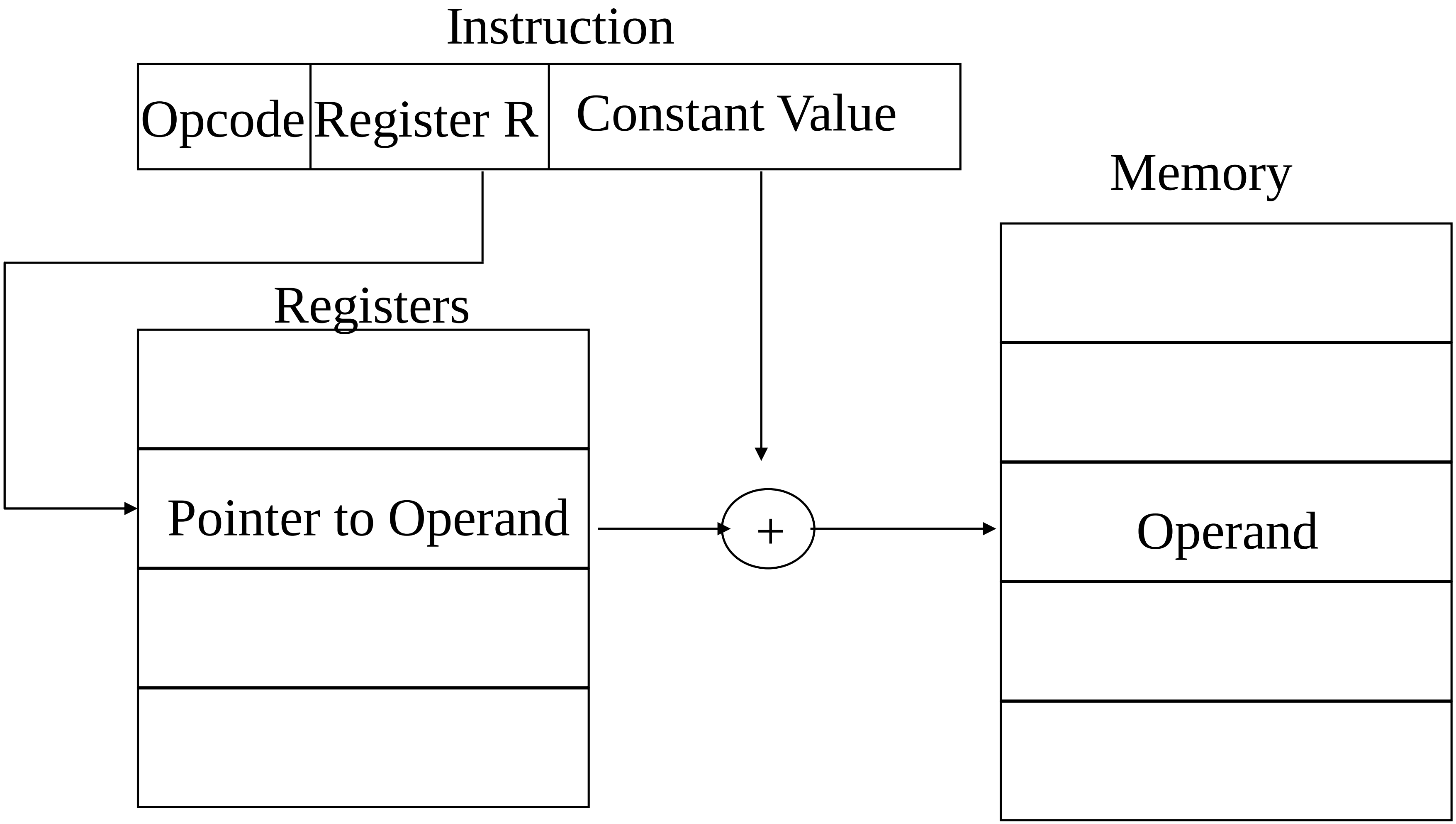
Indexed Addressing

- EA = X + [R]
- Address field hold two values
  - X = constant value (offset)
  - R = register that holds address of memory locations
  - or vice versa

□(Offset given as constant or in the index register)

Add 20(R1),R2 or Add 1000(R1),R2

Indexed Addressing Diagram



Relative Addressing

- A version of displacement addressing
- R = Program counter, PC
- EA = X + (PC)
- i.e. get operand from X bytes away from current location pointed to by PC
- c.f locality of reference & cache usage

Auto increment mode

- The effective address of the operand is the contents of a register specified in the instruction.
- After accessing the operand, the contents of this register are automatically incremented to point to the next item in the list
- EA=[Ri]; Increment Ri ---- (Ri)+

Eg: Add (R2)+,R0

Auto decrement mode

- The contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand

- Decrement Ri; EA= [Ri] ----- -(Ri)

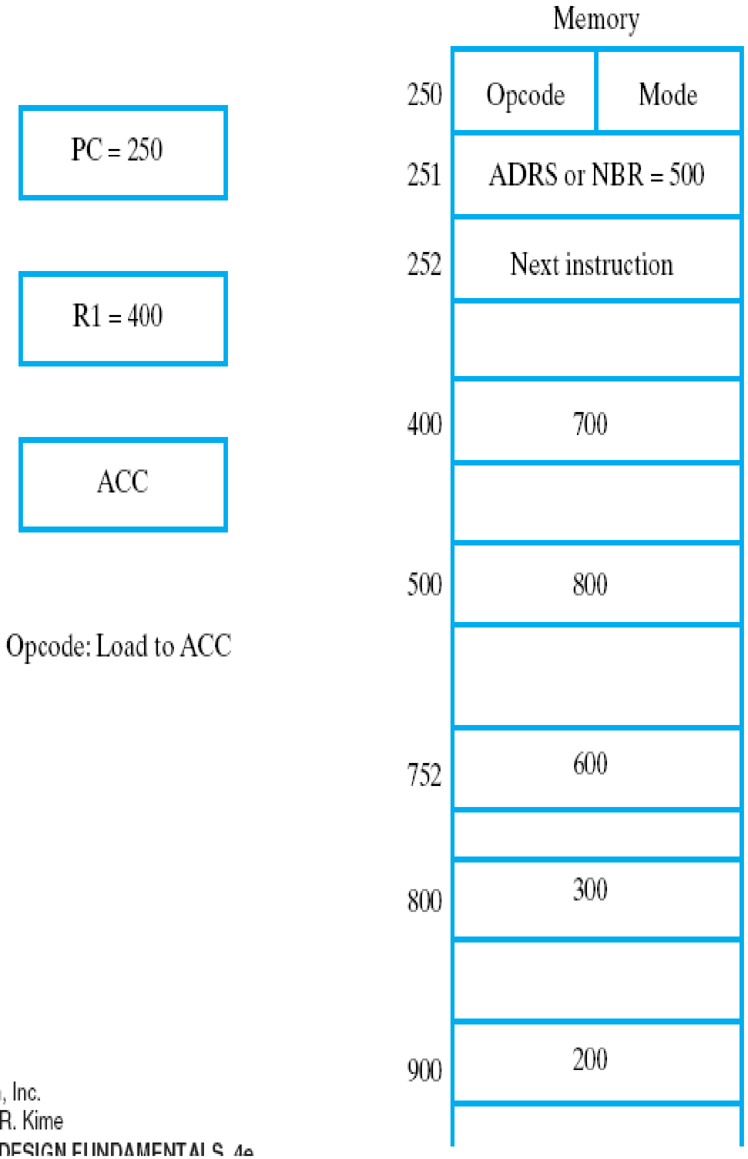
Addressing Architecture

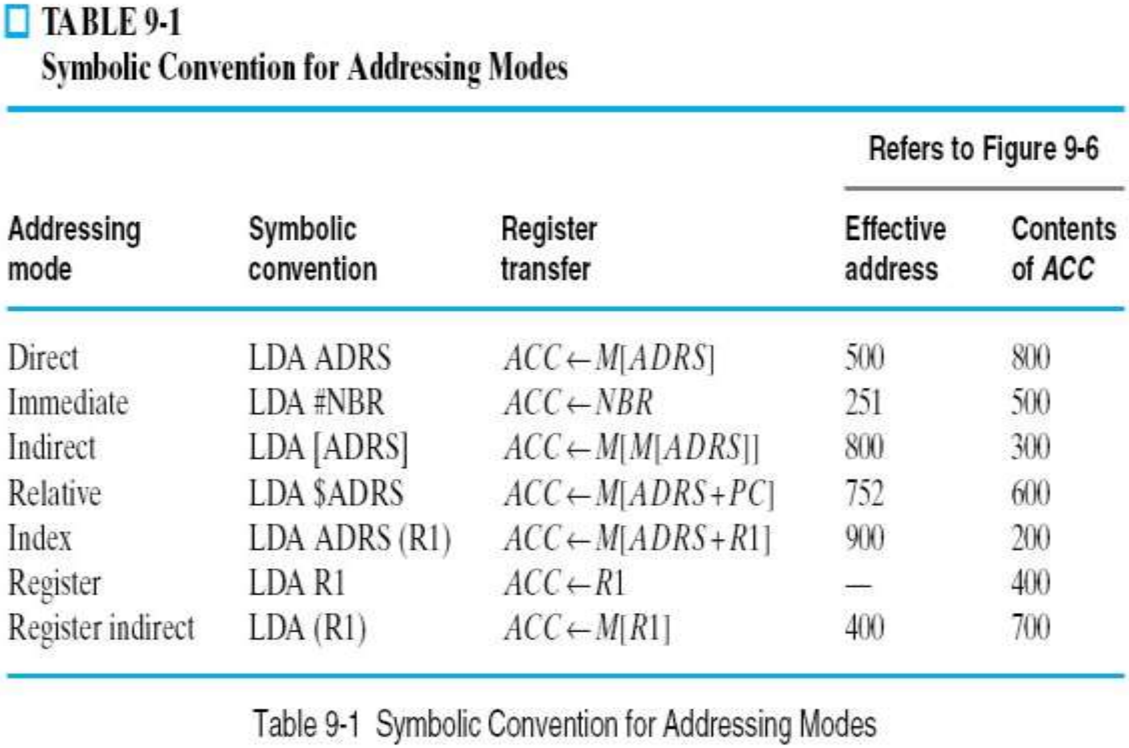
- Memory-to-Memory architecture
  - All of the access of addressing -> Memory
  - Have only control registers such PC
  - Too many memory accesses
- Register-to-Register architecture
  - Allow only one memory address
    - “load”, “store” instructions
- Register-to-Memory architecture
  - Program lengths and # of memory accesses tend to be intermediate between the above two architectures
- Single accumulator architecture
  - Have no register profile
  - Too many memory accesses
- Stack architecture
  - Data manipulation instructions use no address.
  - Too many memory (stack) accesses
  - Useful for rapid interpretation of high-level lang. programs in which the intermediate code representation uses stack operations.

Addressing Modes

- Implied mode
  - The operand is specified implicitly in the definition of the opcode.
- Immediate mode
  - The actual operand is specified in the instruction itself.

Addressing Modes (Summary)





□ **TABLE 9-1**
Symbolic Convention for Addressing Modes

			Refers to Figure 9-6	
Addressing mode	Symbolic convention	Register transfer	Effective address	Contents of *ACC*
Direct	LDA ADRS	$ACC \leftarrow M[ADRS]$	500	800
Immediate	LDA #NBR	$ACC \leftarrow NBR$	251	500
Indirect	LDA [ADRS]	$ACC \leftarrow M[M[ADRS]]$	800	300
Relative	LDA $ADRS	$ACC \leftarrow M[ADRS+PC]$	752	600
Index	LDA ADRS (R1)	$ACC \leftarrow M[ADRS+R1]$	900	200
Register	LDA R1	$ACC \leftarrow R1$	—	400
Register indirect	LDA (R1)	$ACC \leftarrow M[R1]$	400	700

Table 9-1 Symbolic Convention for Addressing Modes

Base register LDA #ADRS(R1) ACC <- M[R1+ADRS]

Instruction Set Architecture

- RISC (Reduced Instruction Set Computer) Architectures
  - **Memory accesses are restricted to load and store instruction, and data manipulation instructions are register to register.**
  - **Addressing modes are limited in number.**
  - **Instruction formats are all of the same length.**
  - **Instructions perform elementary operations**

- CISC (Complex Instruction Set Computer) Architectures
  - **Memory access is directly available to most types of instruction.**
  - **Addressing mode are substantial in number.**
  - **Instruction formats are of different lengths.**
  - **Instructions perform both elementary and complex operations.**

Instruction Set Architecture

- RISC (Reduced Instruction Set Computer) Architectures
  - **Large register file**
  - **Control unit: simple and hardwired**
  - **pipelining**

- CISC (Complex Instruction Set Computer) Architectures
  - **Register file: smaller than in a RISC**
  - **Control unit: often micro-programmed**
  - **Current trend**
    - **CISC operation → a sequence of RISC-like operations**

CISC Examples

- Examples of CISC processors are the
  - System/360(excluding the 'scientific' Model 44),
  - VAX,
  - PDP-11,
  - Motorola 68000 family
  - Intel x86 architecture-based processors.

Pro’s

- Emphasis on hardware
- Includes multi-clock complex instructions
- Memory-to-memory: "LOAD" and "STORE" incorporated in instructions
- Small code sizes, high cycles per second
- Transistors used for storing complex instructions

Con’s

- That is, the incorporation of older instruction sets into new generations of processors tended to force growing complexity.
- Many specialized CISC instructions were not used frequently enough to justify their existence.
- Because each CISC command must be translated by the processor into tens or even hundreds of lines of microcode, it tends to run slower than an equivalent series of simpler **commands** that do not require so much translation.

RISC Examples

- Apple iPods (custom ARM7TDMI SoC)
- Apple iPhone (Samsung ARM1176JZF)
- Palm and PocketPC PDAs and smartphones (Intel XScale family, Samsung SC32442 - ARM9)
- Nintendo Game Boy Advance (ARM7)
- Nintendo DS (ARM7, ARM9)
- Sony Network Walkman (Sony in-house ARM based chip)
- Some Nokia and Sony Ericsson mobile phones

Pro’s

- Emphasis on software
- Single-clock,
  reduced instruction only
- Register to register:
  "LOAD" and "STORE"
  are independent instructions
- Low cycles per second,
  large code sizes
- Spends more transistors
  on memory registers

Performance

- The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program.

Characteristics of RISC Vs CISC processors

No	RISC	CISC
1	Simple instructions taking one cycle	Complex instructions taking multiple cycles
2	Instructions are executed by hardwired control unit	Instructions are executed by microprogramed control unit
3	Few instructions	Many instructions
4	Fixed format instructions	Variable format instructions
5	Few addressing mode, and most instructions have register to register addressing mode	Many addressing modes
6	Multiple register set	Single register set
7	Highly pipelined	Not pipelined or less pipelined

SUMMARY

Computer components and its function

Evolution and types of computer

Instruction and instruction sequencing

Addressing modes

RISC Vs CISC

REFERENCES

- Carl Hammacher,”Computer Organization,”Fifth Edition,McGrawHill International Edition,2002
- P.Pal Chaudhuri,”Compter Organization and Design”,2nd Edition ,PHI,2003
- William Stallings,”Computer organization and Architecture-Designing for Performance”,PHI,2004