UNIT 2

Dr. S. Anbukkarasi
AP/CSE

Basic structure of computers: Functional Units of a computer, Operational concepts, Bus structures, Memory addresses and operations, assembly language , Instructions, Instruction sequencing, Addressing modes. Case study: 8086.

Basic structure of computers:

- Describes function and design of various units of digital computer that store and process the information
- Computer Hardware => electronic circuits, display, magnetic and optical storage media,
- Architecture = > Instruction set and hardware which implements instruction set

What is computer?

Let us first define the term *digital computer*, or simply *computer*. In the simplest terms, a contemporary computer is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions, and produces the resulting output information. The list of instructions is called a computer *program*, and the internal storage is called computer *memory*.

Computer Types

- Personal or Desktop
  - Processing,storage, visual display, audio output,keyboard
- Portable notebooks
- Workstations
  - Engineering Applications
  - High resolution graphics ip/op
- Enterprise systems (Mainframes)
  - More computing power and storage
- Servers - Over Internet
- Super Computers (Large Scale numeric calculations)

Mainframe Computers

Functional Units

5 independent main parts:

- Input
- Memory
- Arithmetic and Logic
- Output
- Control

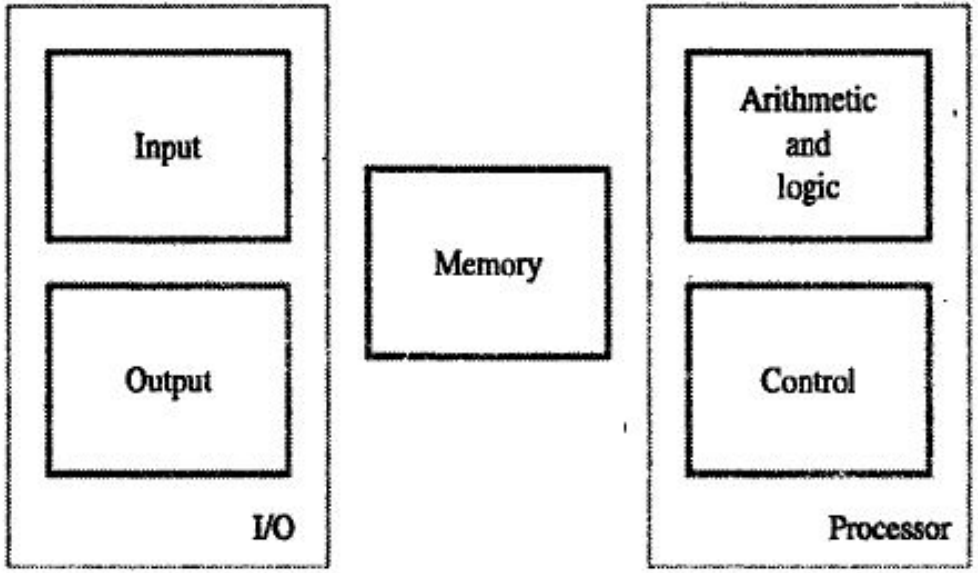


Figure 1.1 Basic functional units of a computer.

- Information to computer is passed as either **data or instruction.**
- Instructions or machine instructions are commands that
    - Governs the transfer of info within computer as well as between the computers and its I/O devices.
    - Specify arithmetic and logic operations to be performed
- A list of instructions that performs a task is known as **program**.
- Computer is completely controlled by **stored programs.**
- Program is stored in memory fetched by processor one after the other and perform a task.

Question Time….

What is source code?

What is object code?

What is compiler?

What is interpreter?

ASCII? No of bits??

EBCDIC

Input units

Memory Unit

- To store data and program
- Two categories:
  - Primary
  - Secondary

Primary

- Fast memory operates at **electronic speed**
- Large number **semiconductor storage cells** capable of storing one bit of info
- **Cells** grouped into fixed size called words
- One word with n bits - be stored and retrieved in one basic operation
- Words are accessed by **addresses**
- No. of bits in each word is known as word length - usually 16 to 64 bits
- RAM - Random Access Memory - any location can be accessed in a short and fixed amount of time.
- Time required to access one word - memory access time (about 100 ns)
- Small , fastest memory - cache
- Largest and slow - main memory

Primary	Secondary
Primary memory is temporary.	Secondary memory is permanent.
Primary memory is directly accessible by Processor/CPU.	Secondary memory is not directly accessible by the CPU.
Nature of Parts of Primary memory varies, RAM- volatile in nature. ROM- Non-volatile.	It's always Non-volatile in nature.
The memory devices used for primary memory are semiconductor memories.	The secondary memory devices are magnetic and optical memories.
Primary memory is also known as Main memory or Internal memory.	Secondary memory is also known as External memory or Auxiliary memory.
Examples: RAM, ROM, Cache memory,	Examples: Hard Disk, Floppy Disk, Magnetic Tapes

Secondary storage

Magnetic Disks and Tapes

Optical disks

ALU

- During arithmetic operations, **operands** are brought into processor
- Stored in high speed storage elements called **registers**

Output unit

Control Unit

Coordinates input and output activities

Uses Time signals to control the operations

Timing signals are signals which determine when a given action is to take place.

Basic Operational Concepts

- To perform task , appropriate program with list of instructions stored in memory.
- Individual instructions brought from memory to processor
- Data act as operands are also stored in memory

  Ex:

  Add LOCA, R0

Add LOCA, R0

Can be done in two steps:

Load LOCA, R1

Add R1, R0

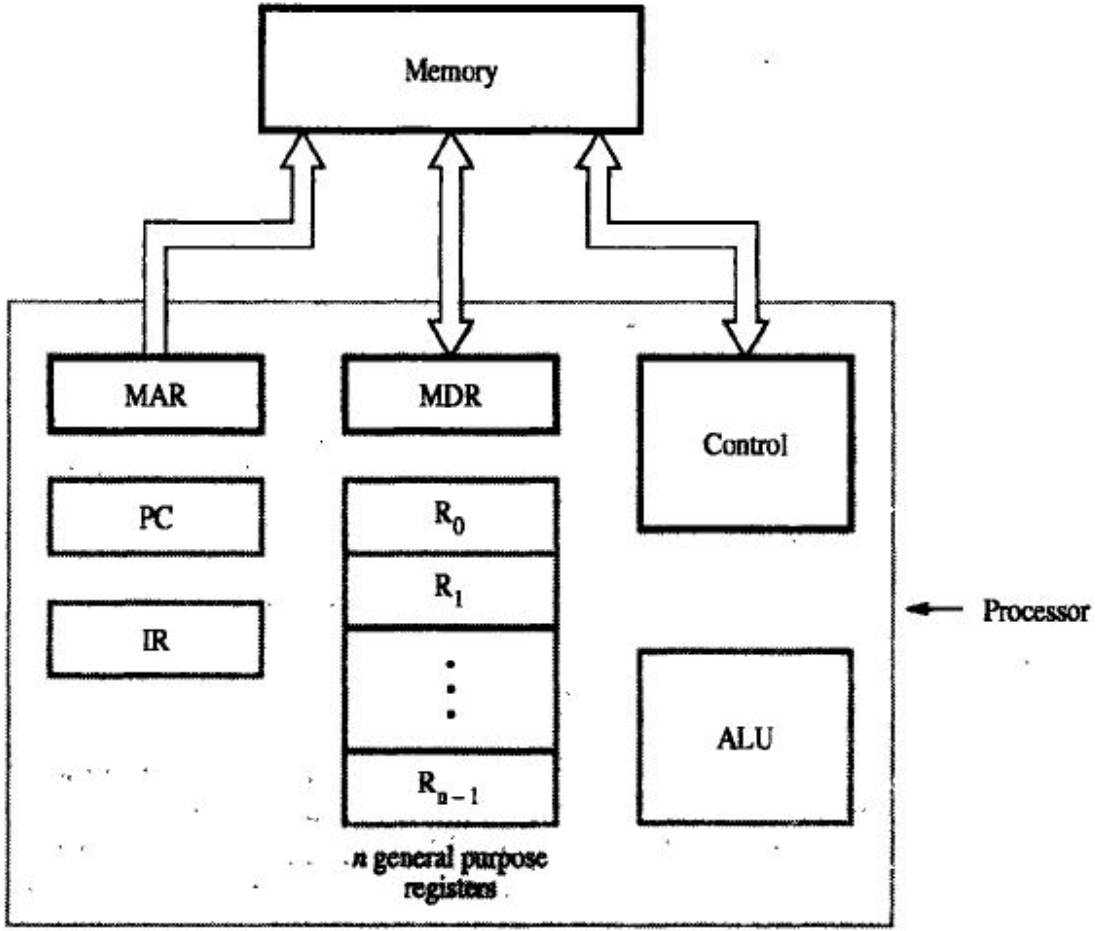


Figure 1.2 Connections between the processor and the memory.

Instruction Register - Holds the instruction is currently being executed.

Program counter - next instruction to be executed.

R_0 to R_{n-1} => n general purpose registers

Memory Address Register => holds the address of the location to be accessed

Memory Data Register => holds data to be written/read to/from the memory

- Operating steps:

Example:

LOAD LOCA, R1

ADD R1, R0

1. Execution of the program starts when the PC is set to point to the first instruction of a program.
2. The contents of the PC are transferred to the MAR and a Read control signal is sent to the memory.
3. After some time, addressed word is read out of the memory and loaded into the MDR.
4. Next the contents of the MDR are transferred to the IR. At this point, the instructions is ready to be decoded and executed.

5. If the instruction involves an operation to be performed by ALU, then its necessary to obtain the required operand.

6. If an operand resides in the memory (it could also be in the GPR), it has to be fetched by setting its address to MAR and initiating Read signal.

7. When the operand has been read from the memory into the MDR, it is transferred from the MDR to ALU. After one or more operands are fetched in this way, the ALU can perform the desired operation.

8. If the result of this operation is to be stored in the memory, then the result is sent to MDR and the address of the result is to be stored in MAR and a write signal is initiated.

9. The contents of the PC are incremented so that PC points to the next instruction to be executed.

Bus Structure

- A group of lines serves as a connecting path for several devices is called bus.
- Addition to carry data, bus must have lines for address and control purposes.

Types of Bus structure:

- Address bus
- Data bus
- Control bus

Address Bus:

- Address bus carry the memory address while reading from writing into memory.
- Address bus carry I/O port address or device address from I/O port.

Data bus:

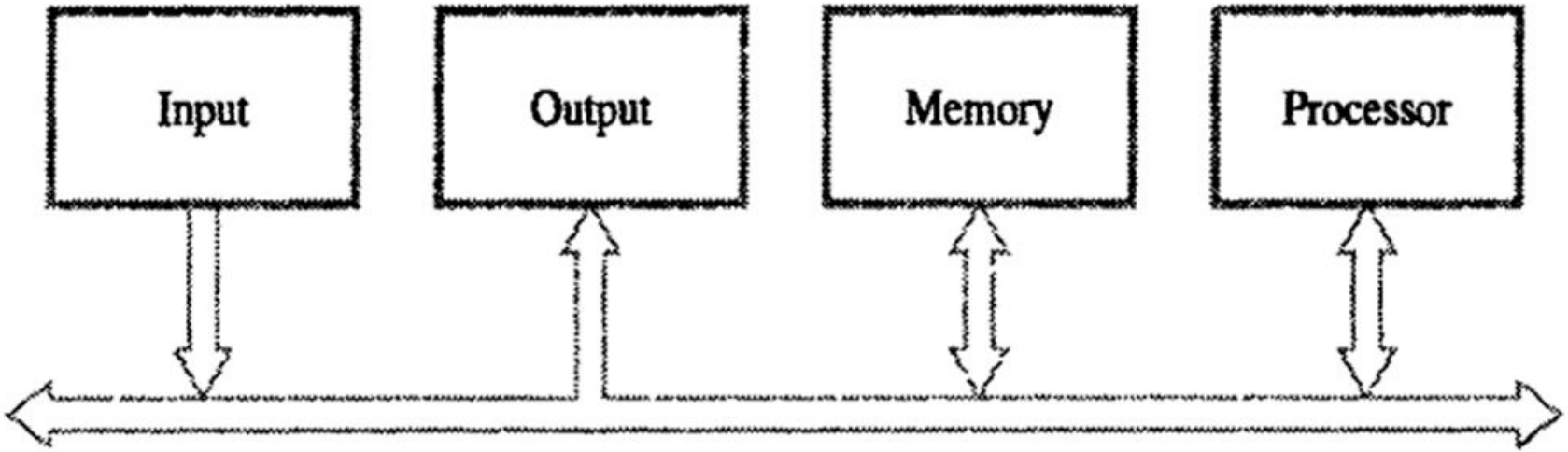
- Data bus carry the data. It is bidirectional.
- Data bus fetch the instructions from memory.

- Data bus used to store the result of an instruction into memory. It carry commands to an I/O device controller .

Control Bus:

- **Memory Read:** This signal, is issued by the CPU when performing a read operation with the memory.
- **Memory Write:** This signal is issued by the CPU when performing a write operation with the memory.
- **I/O Read:** This signal is issued by the CPU when it is reading from an input port.
- **I/O Write:** This signal is issued by the CPU when writing into an output port.

Single bus structure



Memory Locations and Addresses

- Group of n bits represent word => n is word length
- Rage from 16 to 64 bits
- If word length is 32b,

  A single word can store 32-bit 2's complement number

  Or

  4 ASCII Characters (Here each occupies 8 bits)

- 1 byte = ?

Memory Words

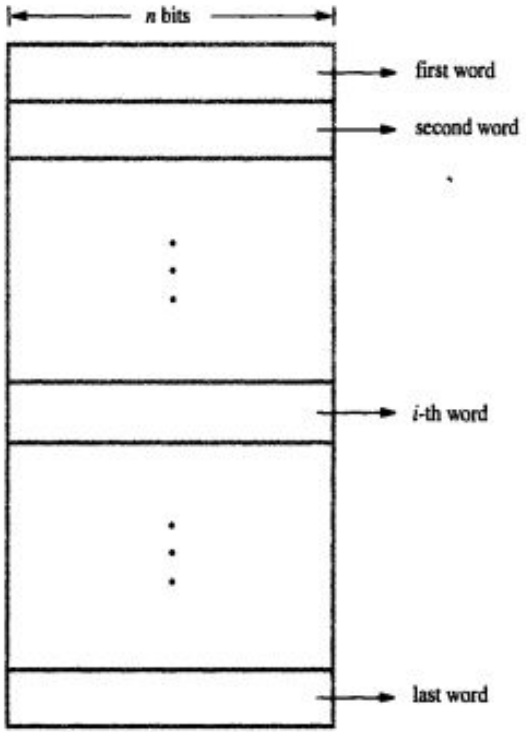
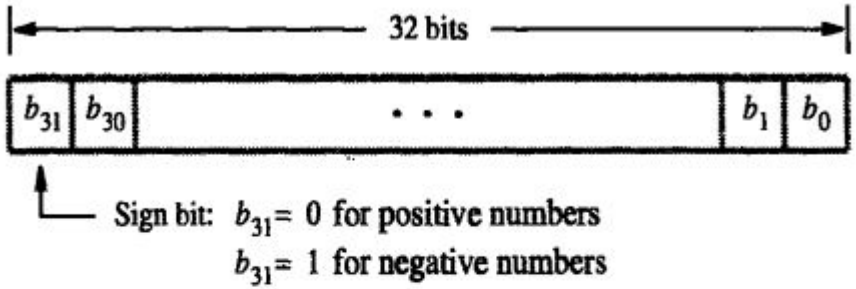
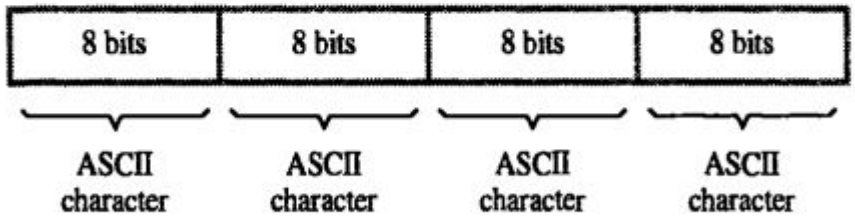


Figure 2.5 Memory words.



(a) A signed integer



(b) Four characters

Figure 2.6 Examples of encoded information in a 32-bit word.

- Distinct names or address helps to access memory to store or retrieve
- Represents from 0 to 2^k - 1.
- 24 bit address generates **address space** of 2^{24} locations (16,777,216)
- 1 K => 2^{10}
- 1M => 2^{20}
- 1G => 2^{30}
- 1 T => 2^{40}

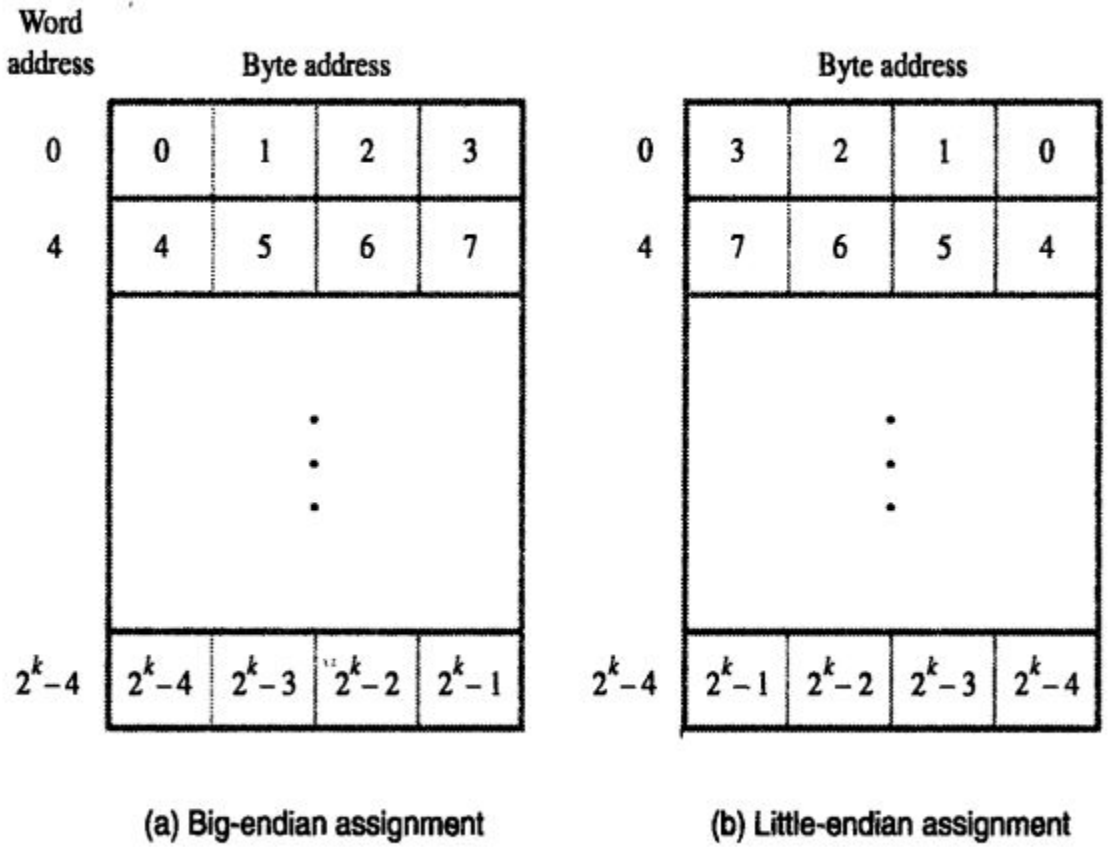


Figure 2.7 Byte and word addressing.

Big-endian => when lower byte addresses are used for the more significant bytes.

Little-endian => lower byte addresses are used for the less significant bytes.

Memory Operations

Two operations involving memory are:

Load (Read or Fetch)

- Transfers **copy** of contents of specific memory location to processor

Store (write)

- Transfer the information from the processor to specific memory location

Instructions and Instruction Sequencing

The tasks carried out by a computer program consist of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen. A computer must have instructions capable of performing four types of operations:

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

The **operations and operands** are represented by two types of notations:

- Register Transfer Notation (RTN)
- Assembly Language Notation (ALN)

Register Transfer Notation

- Transfer of information from one location to another
- Possible locations: memory locations, processor registers, registers in I/O subsystem
- Location identified by symbolic representation
- Memory location: LOC, PLACE, A, VAR2
- Processor Register: R0, R5
- I/O register names: DETAIN, OUTSTATUS
- Contents are represented using []
- R1<-[LOC]
- R3<- [R1] + [R2]

- Example 1: C= A+B, to represent this in RTN
- The left hand side contains the destination
- The right hand side contains the source.

  C<- [A]+[B]
- Example 2: move A to R0

  R0<-[A]
- Example 3 move contents from R2 to A

  A<-[R2]

Assembly Language Notation

Move LOC, R1

LOC => Unchanged

- Example 1: C=A+B

  Move A, R1

  Add B, R1

  Move R1,C

Add R1, R2, R3

Basic Instruction Type

Type of Instruction:

- One Address Instruction
- Two Address Instruction
- Three Address Instruction
- Zero Address Instruction

three-address instruction:

Format: Operation source1, source2, destination

Add A,B,C

A, B => Source operands

C => Destination operands

- If k-bits are needed to specify memory address of each operand , 3k bits needed for above addressing format.
- 32-bit address 3-address is too large

two-address instruction

Format : Operation source, destination

Add A,B

B<- [A] + [B]

It replaces the original content

Move B,C => Another instruction is needed

C <- [B]

The operation C <- [A] + [B] can be done as,

Move B,C => Contents of B is moved to C

Add A, C

one-address instruction

A processor register usually called accumulator can be used for this purpose.

Ex: Add A

=> Adds the contents of memory location A to the contents of accumulator register and place the sum back to the accumulator.

Load A => copy the contents of memory location A to accumulator

Store A => Copy the contents of accumulator to memory location

Load A

Add B

Store C

Zero Address Instruction: These instructions do not specify any operands or addresses. Instead the operands will be stored in stack.

General Purpose registers

- Most modern computers have 8-32 GPR
- Load A, Ri
- Store Ri, A
- Add A, Ri

Some performs direct operations using GPR.

Add Ri, Rj

Add Ri, Rj, Rk

Move instruction can be used for Load/Store

Move A, Ri

Move B, Rj

Add Ri,Rj

Move Rj, C

If one operand should be in memory then,

Move A, Ri => Here one operand in location A, and another operation in Register Ri

Add B,Ri

Move Ri,C

Instruction Execution and Straight line Sequencing

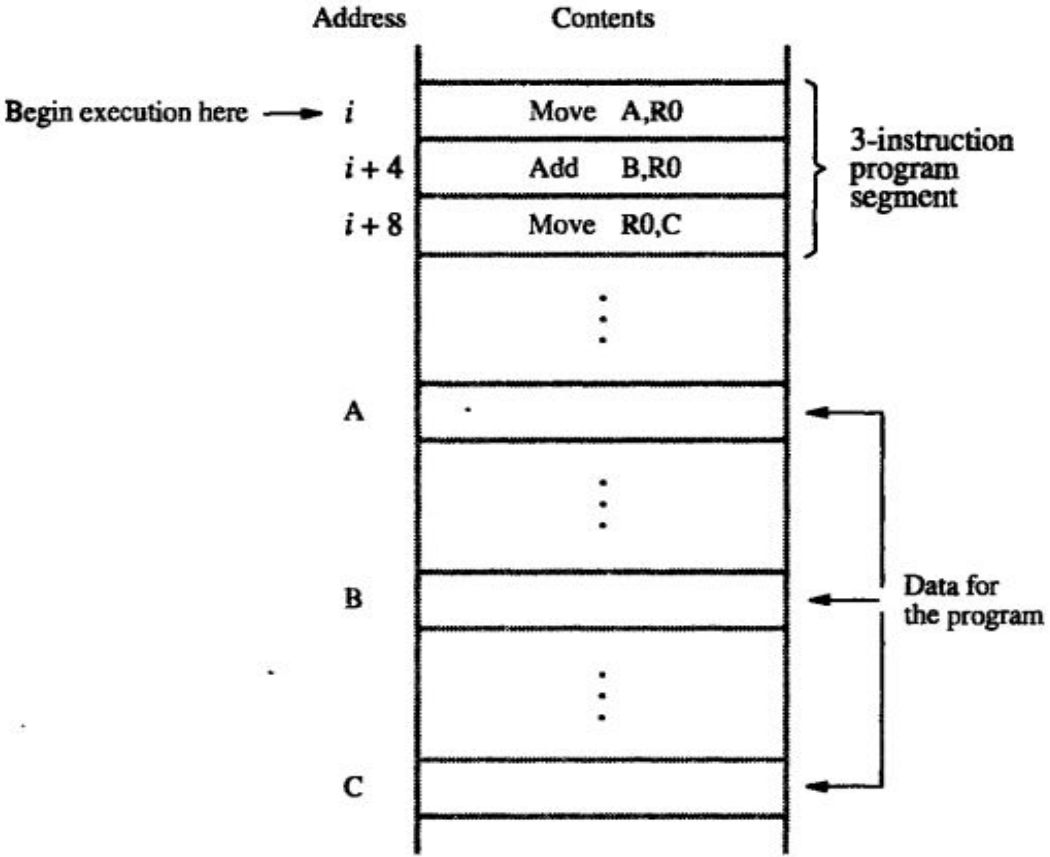


Figure 2.8 A program for C ← [A] + [B].

- The instructions are executed one at a time inorder to increase the addresses. Knows as **Straight-line sequencing.** During instruction PC is incremented by 4 to point to next instruction.

- The program counter(PC) holds the instructions to be executed next. To begin the execution, the address of the first instructions must be placed into the PC.

Execution of instruction is done in two phase: **instruction fetch , instruction execute(fetching operands, performing operation, storing result)**

- Lets consider the task of adding a list of n numbers. The addresses of the memory location containing the n numbers are represented as NUM1, NUM2,….NUMn and a separate Add instruction is used to add each number to the contents of register R0.
- Finally, after adding all the numbers, the result is placed in the memory location SUM.
- Instead of adding long list of Add instructions, it is possible to place a single Add instruction in a program loop. The loop is a straight line sequence of instructions executed as many times as needed.
- It starts at the location LOOP and ends at the instruction Branch>0. During each pass through the loop, address of the next list entry is determined and it is added to R0.

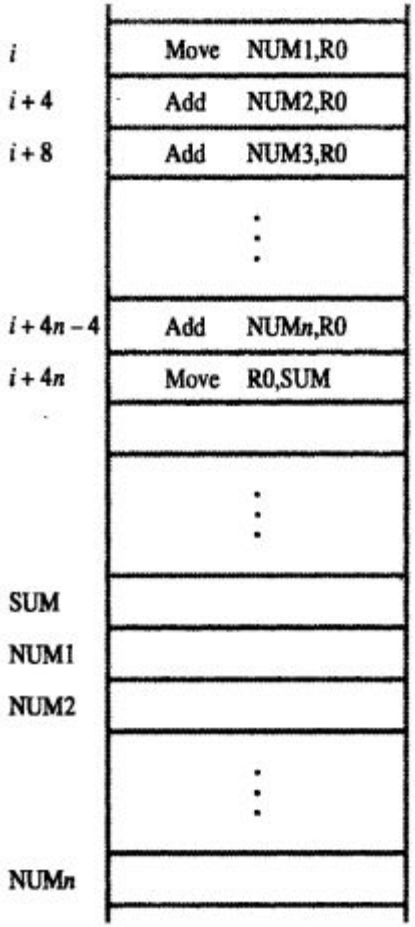


Figure 2.9 A straight-line program for adding n numbers.

Branching:

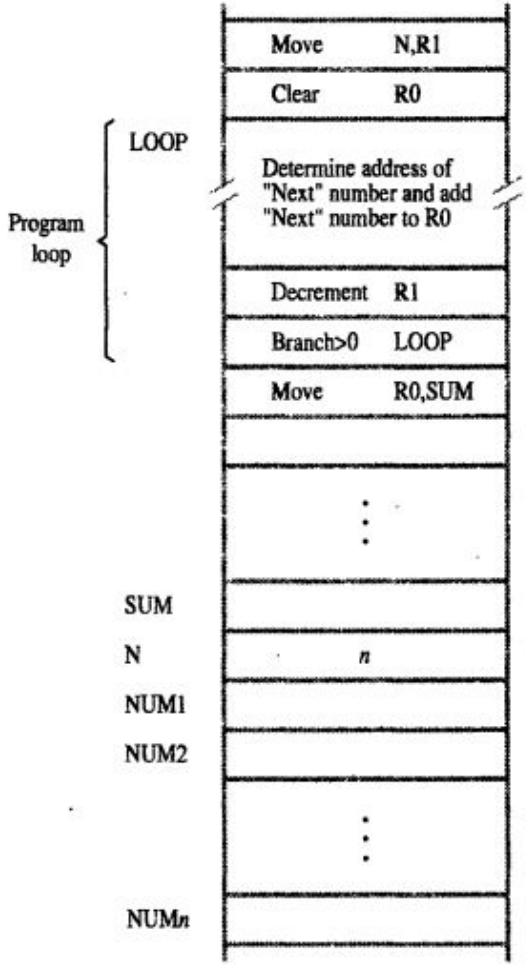


Figure 2.10 Using a loop to add *n* numbers.

- Branch instruction loads new value to the program counter
- Processor fetches and executes the instruction at the new address called **branch target**
- Conditional branch instruction causes a branch only if a specified condition is met.

Condition Codes

The Processor keeps track of information about the **results of various operations** for use by subsequent conditional branch instructions. This is accomplished by recording the required information in individual bits called as **condition code flags.**

Four commonly used flags are

N (negative)	Set to 1 if the result is negative; otherwise, cleared to 0
Z (zero)	Set to 1 if the result is 0; otherwise, cleared to 0
V (overflow)	Set to 1 if arithmetic overflow occurs; otherwise, cleared to 0
C (carry)	Set to 1 if a carry-out results from the operation; otherwise, cleared to 0

Addressing Modes

The **different ways in which the location of an operand** is specified in an instruction are called as Addressing modes.

What is Data Structure?

Example for DS?

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

Different types of addressing modes:

- Register mode
- Absolute mode
- Immediate mode
- Indirect mode
- Index mode
- Relative mode
- Auto increment mode
- Auto decrement mode

- **Register mode:** the operand is the contents of a processor register, the name of the register is given in the instruction.

  Move R1,R2

- **Absolute (direct) mode:** the operand is in the memory location, the address of this location is given explicitly in the instruction.

  Move LOCA, R2

- **Immediate mode**

The operand is given exactly in the instruction.

Move $200_{immediate,}$R0

Move #200, R0

A= B + 6

Move B, R1
Add #6, R1
Move R1, A

Indirect mode

- The instruction does not give the operand or its address explicitly
- It provides the information from which the memory address of the operand can be determined.
- We refer this address as effective address (EA)

The effective **address of the operand is the contents of a register** or memory location whose address appear in the instruction.

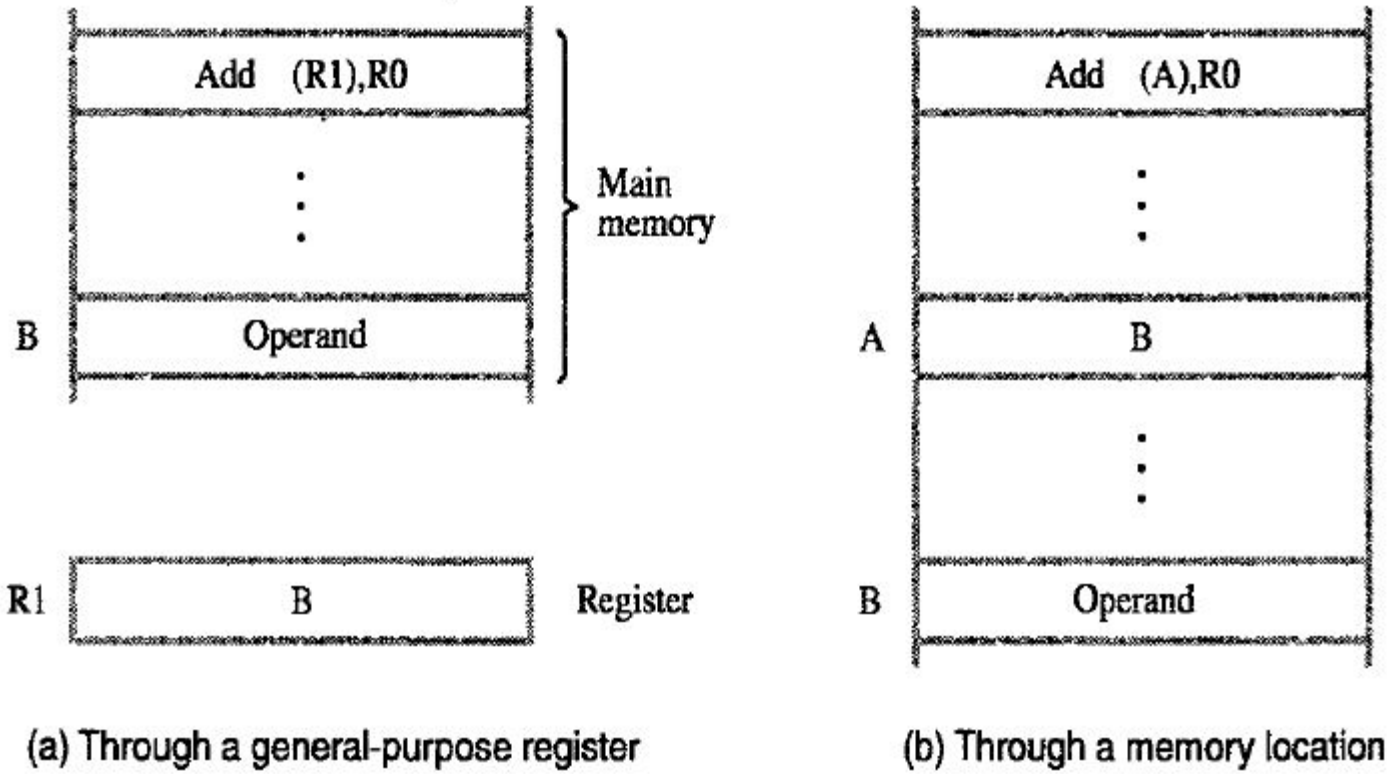


Figure 2.11 Indirect addressing.

Program

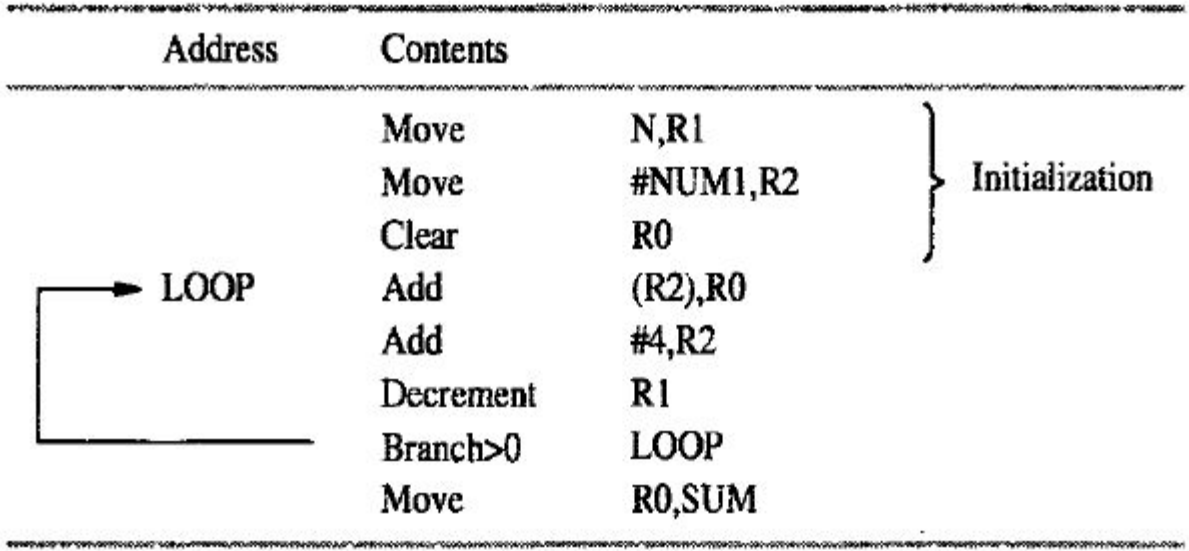


Figure 2.12 Use of indirect addressing in the program of Figure 2.10.

The register or memory location that contains the address of an operand is called **pointer.**

A = *B

Using indirect Address:

Move B, R1

Move (R1), A

Through Memory:

Move (B), A

Indexing

The effective address (EA) of the operand is generated by adding a constant value to the contents of a register. This register is known as **index register**

X(Ri)

X => constant value in register

Ri => name of the register

EA is given by,

EA = X + [Ri]

Variations in Index addressing modes:

- Base Index mode
- Base Index & offset

- Base Index mode:

  Effective Address (EA) = (Ri)+(Rj)

  where, Ri is the general purpose register

  Rj contains the contents of the X

- Base Index & offset:

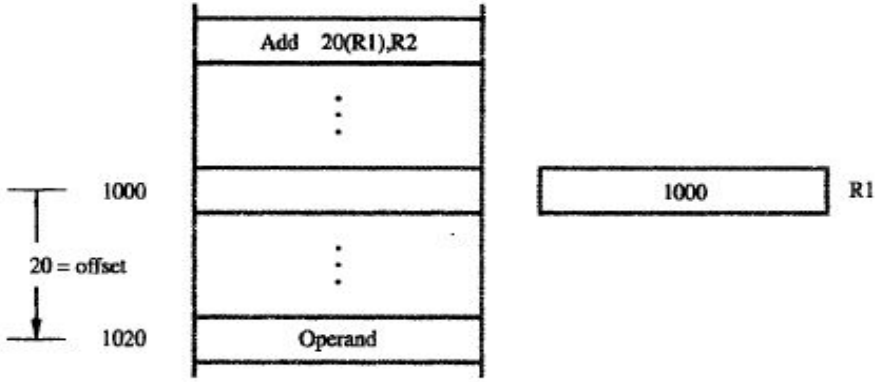
Effective Address (EA) = X+(Ri)+(Rj)

where, Ri, Rj- operands

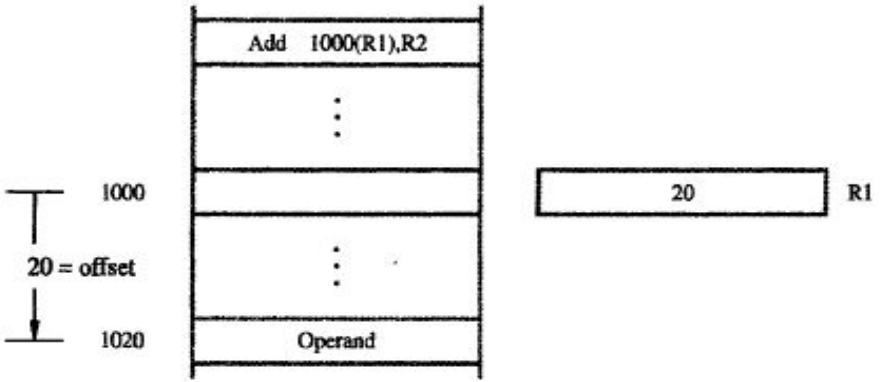
X- constant (offset)

Offset => called as

displacement



(a) Offset is given as a constant



(b) Offset is in the index register

Figure 2.13 Indexed addressing.

Example: Adding ‘N’ students Marks

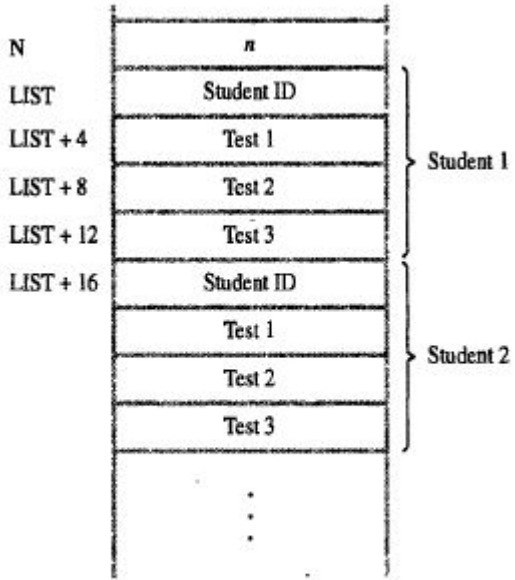


Figure 2.14 A list of students' marks.

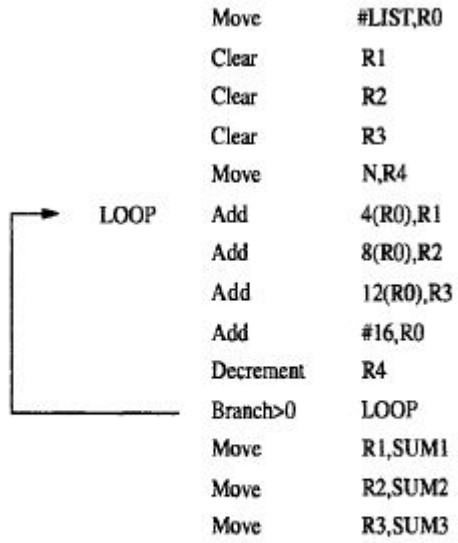


Figure 2.15 Indexed addressing used in accessing test scores in the list in Figure 2.14.

Relative Addressing

The effective address is determined by the index mode using the program counter in place of general-purpose register Ri.

This mode can be used to access **data operand.**

Branch>0 LOOP (Causes program execution to go to branch target location identified by name LOOP if condition is satisfied)

Additional Modes

Auto Increment Mode: The effective address of the operand is the contents of the register specified in the instruction.

A special case of Register Indirect Addressing Mode where

Effective Address of the Operand = Content of Register

After **accessing the operand**,**the** contents of this register are automatically **incremented to point** to the next item in a list.

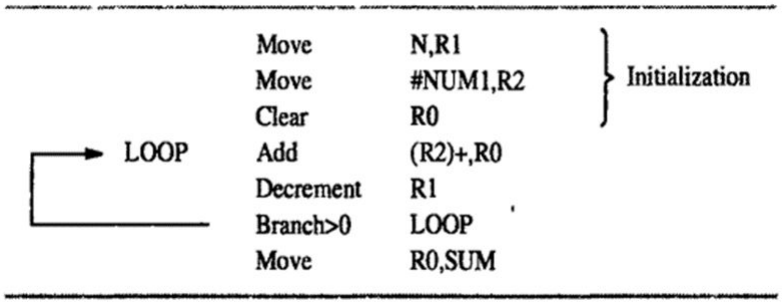
(Ri)++

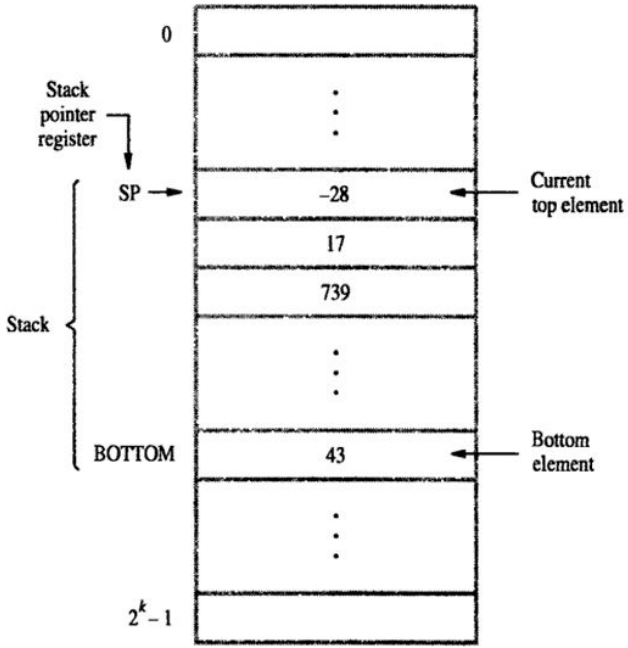
Autodecrement mode: the contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand.

-(Ri)

- A special case of Register Indirect Addressing Mode where

Effective Address of the Operand = Content of Register – Step Size





The push and pop operation can be implemented as

PUSH

Subtract #4, SP

Move NEWITEM, (SP)

POP

Move (SP),ITEM

Add #4, SP

If the processor has the Autoincrement and Autodecrement addressing modes, then the push and pop operation can be performed by the single instruction.

Move NEWITEM, -(SP) (PUSH)

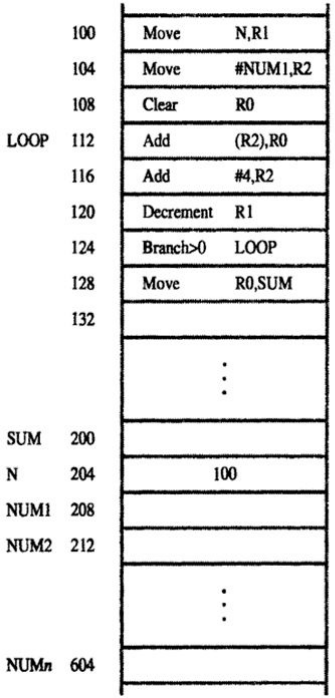
Move (SP)+, ITEM (POP)

Assembly Language

- Assembly Language is a low-level programming language. It helps in understanding the programming language of machine code.
- It is a series of instructions that provide the necessary information to a user's CPU (Central Processing Unit) to carry out a particular task (add, subtract, compare values, etc.).
- In computers, there is an **assembler** that helps in converting the assembly code into machine code executable.
- Assembly language uses a series of mnemonic codes to represent machine language instructions.
- The normal words such as move, add, increment, branch can be replaced by acronyms called mnemonics, such MOVE, ADD, INC and BR. Similarly, notation R3 is used to refer register 3 and LOC used to refer a memory location.
- The set of rules for using the mnemonics are called as the syntax of the language.
- The assembly language is translated into corresponding machine instructions using assembler.

- The user program is usually entered into a computer through a keyboard and stored either in the memory or on a magnetic disk.
- At this point, the user program is simply a set of lines of alphanumeric characters.
- The assembler then reads the user program, analyze it and then generated the machine language program.
- The original user program in alphanumeric characters are called a source program and the assembled machine language program is called as a object program.
- The assembly language for a given computer may or may not be case sensitive.
- Example: MOVE R0,SUM
- The mnemonics MOVE followed by one blank space character, then the information that specifies the operand is given.
- The source operand is register R0 followed by destination operands. Both the source and destination separated by comma with no intervening blanks.
- Since there are several possible addressing modes for specifying operand locations, the assembly language mush indicate which mode is being used.
    - MOVE R0,SUM
    - ADD #5,R3
    - ADDI 5,R3
    - MOVE #5,(R2)

Assembler directives (commands): the assembly language allows the programmer to specify other information needed to translate the program into the object program.



	Memory address label	Operation	Addressing or data information
Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
		ORIGIN	100
Statements that generate machine instructions	START	MOVE	N,R1
		MOVE	#NUM1,R2
		CLR	R0
	LOOP	ADD	(R2),R0
		ADD	#4,R2
		DEC	R1
		BGTZ	LOOP
		MOVE	R0,SUM
Assembler directives		RETURN	
		END	

Write Instruction for the expression

One Address	Two Address	Three Address	Zero Address

C = A + B
Move A
Add B
Store C
	C = A + B
	Move A, R0
	Add B, R0
	Store R0, C
		C = A + B
		Add A, B, R0
		Store R0, C
			C = A + B
			Push A
			Push B
			Add
			Store C

Write Instruction for the expression

E = (A + B) * (C + D)

Three Address Instruction:

ADD A, B, R0

ADD C, D, R1

MUL R0, R1, E

Two Address Instruction: E = (A + B) * (C + D)

Move A, R0

Add B, R0

Move C, R1

Add D, R1

Mul R0, R1

Store R1, E

One Address Instruction (Another oprand from Accumalator) : E = (A + B) * (C + D)

Move A

Add B

Store T

Move C

Add D

Mul T

Store T

Zero Address Instruction (Another operand from stack): E = (A + B) * (C + D)

Push A

Push B

Add

Push C

Push D

Add

Mul

Store E

Case Study: 8086
Introduction to Microprocessor

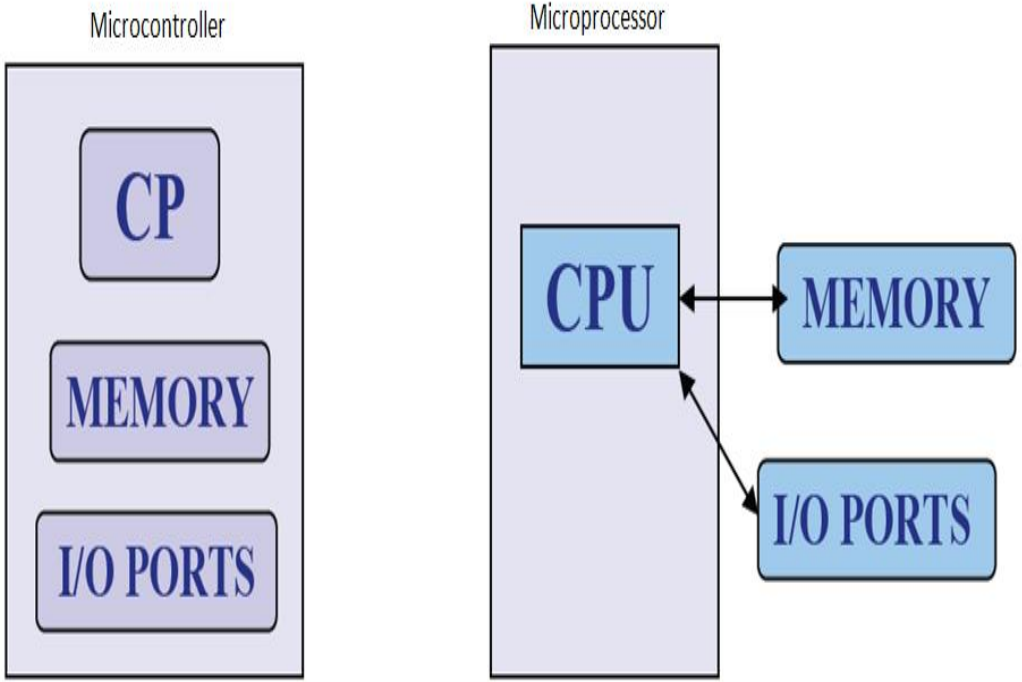
Microprocessor

Microprocessor : is a CPU on a single chip.

Microcontroller: If a microprocessor, its associated support circuitry, memory and peripheral I/O components are implemented on a single chip, it is a microcontroller.

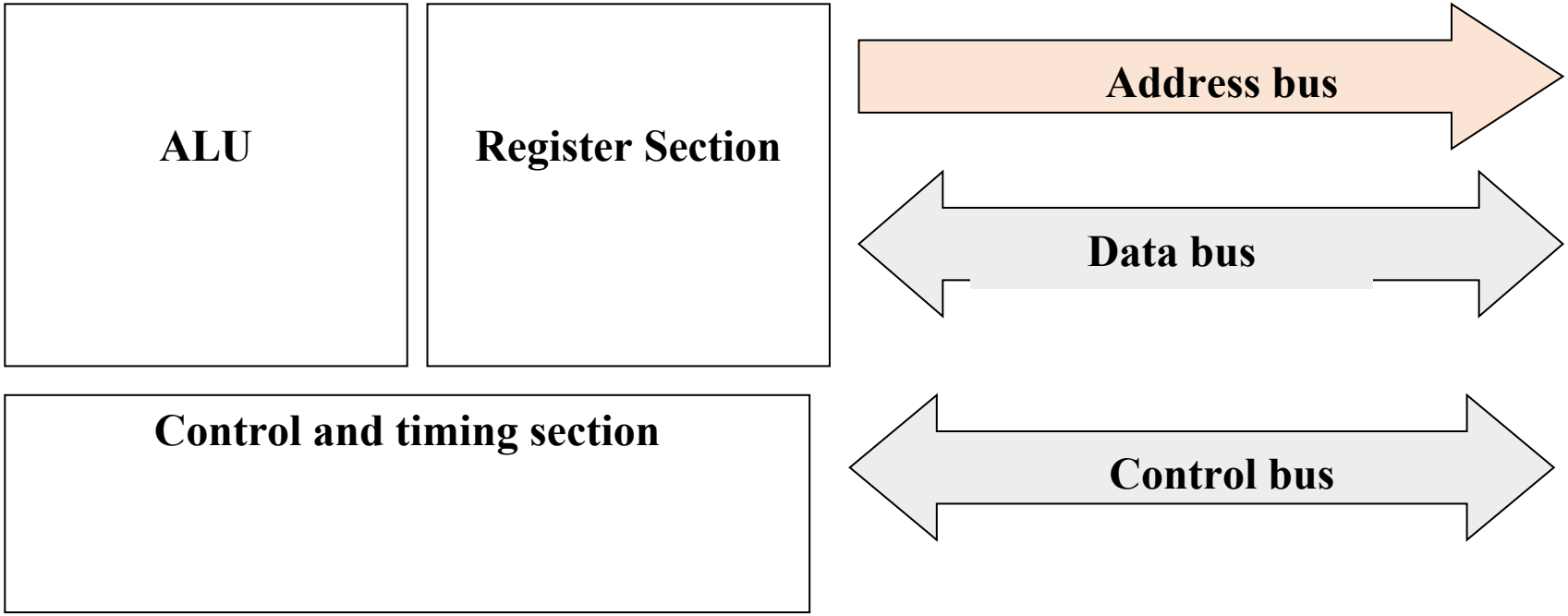
We use AVR microcontroller as the example in our course study

What is Microprocessor and Microcontroller?



Sl No	Microprocessor	Microcontroller
1	CPU is stand-alone, RAM, ROM, I/O, timer are separate	CP,RAM,ROM,I/O and timer are all on single chip
2	Designer can decide on the amount of ROM,RAM and I/O ports	Fix amount of on-chip ROM, RAM, I/O ports
3	Expansive	Not Expansive
4	General purpose	Single purpose
5	Microprocessor based system design is complex and expensive	Microcontroller based system design is rather simple and cost effective
6	The instruction set of microprocessor is complex with large number of instructions.	The instruction set of a Microcontroller is very simple with the less number of instructions.

Internal structure and basic operation of microprocessor



Block diagram of a Microprocessor

Microprocessor performs **three main tasks:**

- data transfer between itself and the memory or I/O systems
- simple arithmetic and logic operations
- program flow via simple decisions

Microprocessor types

Microprocessors can be characterized based on the word size

- 8 bit, 16 bit, 32 bit, etc. processors
- Instruction set structure
- RISC (Reduced Instruction Set Computer), CISC (Complex Instruction Set Computer)

Functions

- General purpose, special purpose such as **image processing, floating point calculations**
- And more …

Evolution of Microprocessors

- The first **microprocessor** was **introduced** in 1971 by Intel Corp.
- It was named Intel 4004 as it was a 4 bit processor.

Categories according to the generations or size

First Generation (4 - bit Microprocessors)

- could perform simple arithmetic such as addition, subtraction, and logical operations like Boolean OR and Boolean AND.
- had a control unit capable of performing control functions like
  - fetching an instruction from storage memory,
  - decoding it, and then
  - generating control pulses to execute it.

Second Generation (8 - bit Microprocessor)

- The second generation microprocessors were introduced in 1973 again by Intel.
- the first 8 - bit microprocessor which could perform arithmetic and logic operations on 8-bit words.

Third Generation (16 - bit Microprocessor)

- introduced in 1978
- represented by **Intel's 8086, Zilog Z800 and 80286**,
- 16 - bit processors with a performance like minicomputers.

Fourth Generation (32 - bit Microprocessors)

- Several different companies introduced the 32-bit microprocessors
- the most popular one is the **Intel 80386**

Fifth Generation (64 - bit Microprocessors)

- Introduced in 1995
- After 80856, Intel came out with a new processor namely Pentium processor followed by **Pentium Pro CPU**
- allows multiple CPUs in a single system to achieve multiprocessing.
- Other improved 64-bit processors are **Celeron, Dual, Quad, Octa Core processors**.

Typical microprocessors

Most commonly used

- 68K
  - Motorola
- X86
  - Intel
- IA-64
  - Intel
- MIPS
  - Microprocessor without interlocked pipeline stages
- ARM
  - Advanced RISC Machine
- PowerPC
  - Apple-IBM-Motorola alliance
- Atmel AVR

8086 Microprocessor

- designed by Intel in 1976
- 16-bit Microprocessor having
- 20 address lines
- 16 data lines
- provides up to 1MB storage
- consists of powerful instruction set, which provides operations like multiplication and division easily.

supports two modes of operation

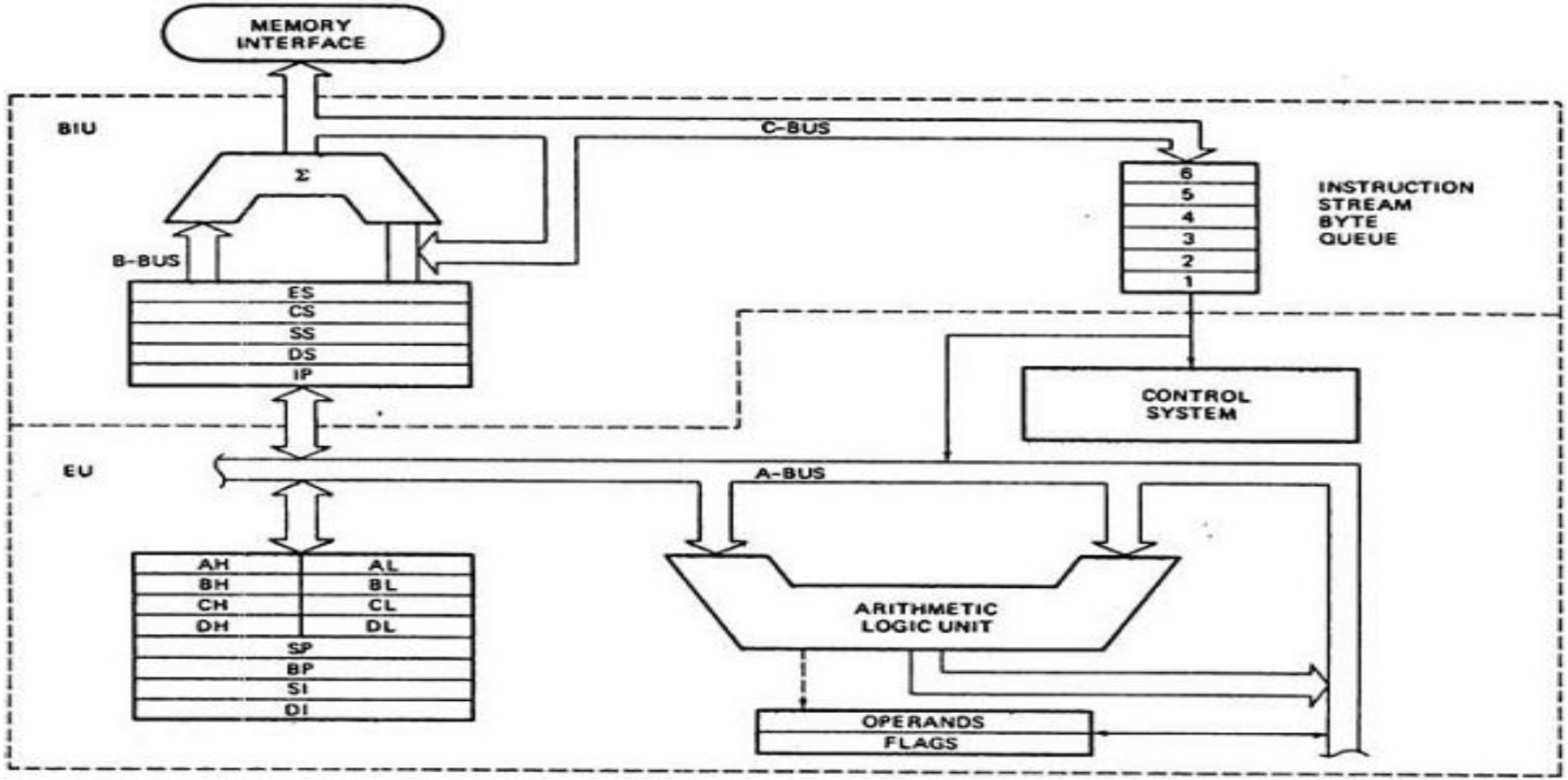
Maximum mode : suitable for system having multiple processors

Minimum mode : suitable for system having a single processor.

Features of 8086

- Has an instruction queue, which is capable of storing six instruction bytes
- First 16-bit processor having
  - 16-bit ALU
  - 16-bit registers
- internal data bus
- 16-bit external data bus
- uses two stages of pipelining
- 1. Fetch Stage and
  - 2. Execute Stage
- which improves performance.
- **Fetch stage** : can pre-fetch up to 6 bytes of instructions and stores them in the queue.
- **Execute stage** : executes these instructions.

Architecture of 8086



Segments in 8086

memory is divided into various sections called segments

Code segment : where you store the program.
Data segment : where the data is stored.
Extra segment : mostly used for string operations.
Stack segment : used to push/pop

General purpose registers

used to store temporary data within the microprocessor

AX – Accumulator

16 bit register

divided into two 8-bit registers AH and AL

to perform 8-bit instructions also

generally used for arithmetical and logical instructions

BX – Base register

16 bit register

divided into two 8-bit registers BH and BL

to perform 8-bit instructions also

Used to store the value of the offset.

CX – Counter register

16 bit register

divided into two 8-bit registers **CH and CL** to perform 8-bit instructions also used in looping and rotation

DX – Data register

16 bit register

divided into two 8-bit registers **DH and DL** to perform 8-bit instructions also used in multiplication an input/output port addressing

Pointers and Index Registers

SP – **S**tack pointer

- 16 bit register
- points to the topmost item of the stack
- If the stack is empty the stack pointer will be (FFFE)H
- It's offset address relative to stack segment

BP –**B**ase pointer

- 16 bit register
- used in accessing parameters passed by the stack
- It's offset address relative to stack segment

SI – Source index register

- 16 bit register
- used in the pointer addressing of data and
- as a source in some string related operations
- It's offset is relative to **data segment**

DI – Destination index register

- 16 bit register
  used in the pointer addressing of data and
- as a destination in string related operations
- It's offset is relative to **extra segment**.

IP - Instruction Pointer

- 16 bit register
- stores the address of the next instruction to be executed
- also acts as an offset for CS register.

Segment Registers

CS - Code Segment Register:

user cannot modify the content of these registers

Only the microprocessor's compiler can do this

DS - Data Segment Register:

The user can modify the content of the data segment.

SS - Stack Segment Registers:

used to store the information about the memory segment.

operations of the SS are mainly Push and Pop.

ES - Extra Segment Register:

By default, the control of the compiler remains in the DS where the user can add and modify the instructions

If there is less space in that segment, then ES is used

Also used for copying purpose.

Flag or Status Register

- 16-bit register
- contains 9 flags
- remaining 7 bits are idle in this register
- These flags tell about the status of the processor after any arithmetic or logical operation
- IF the flag value is 1, the flag is set, and if it is 0, it is said to be reset.

Microcomputer

- A digital computer with one microprocessor which acts as a CPU
- A complete computer on a small scale, designed for use by one person at a time
- called a personal computer (PC)
- a device based on a single-chip microprocessor
- includes laptops and desktops

Block Diagram