18CSC203J - Computer Organization and Architecture

Unit - 1

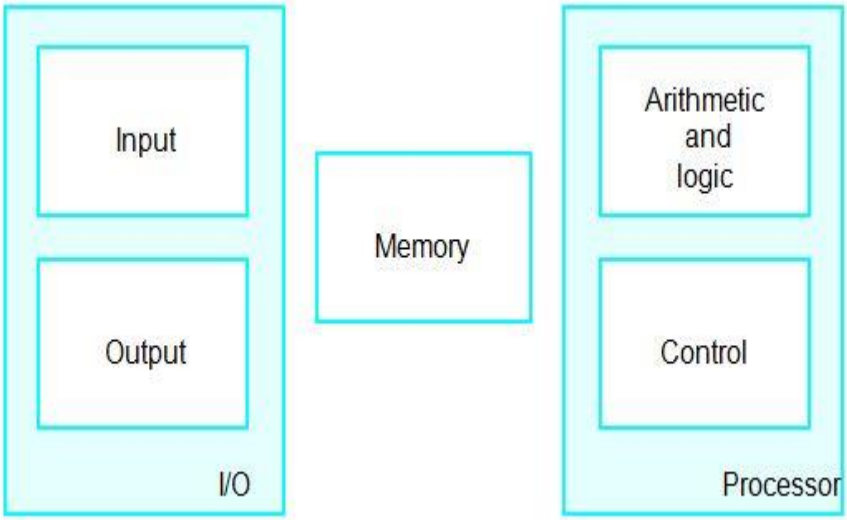
Functional units of a computer – Operational concepts – Bus structures – Memory locations and addresses – Memory operations – Instructions – Instruction Sequencing – Addressing modes – Introduction to Microprocessor – Introduction to Assembly language – Writing of Assembly Language programming – ARM processor – Thumb instruction set – Processor and CPU cores – Instruction encoding format – Load and Store instructions in ARM – Basics of IO operations.

COMPUTER

- A computer is a sophisticated electronic calculating machine that:
  - Accepts input information,
  - Processes the information according to a list of internally stored instructions and
  - Produces the resulting output information.
- Functions performed by a computer are:
  - Accepting information to be processed as input.
  - Storing a list of instructions to process the information.
  - Processing the information according to the list of instructions.
  - Providing the results of the processing as output.
- **Computer Architecture**
  - Computer Architecture is a functional description of requirements and design implementation for the various parts of a computer. It deals with functional behavior of computer system. It comes before the computer organization while designing a computer.
  - Architecture describes what the computer does.
- **Computer Organization**
  - Computer Organization comes after the decide of Computer Architecture first. Computer Organization is how operational attribute are linked together and contribute to realize the architectural specification. Computer Organization deals with structural relationship.
  - Organization describes how it does it.

FUNCTIONAL UNITS OF A COMPUTER

- A Computer consists of 5 functionally independent main parts:
  - Input
  - memory
  - Arithmetic and Logic
  - Output
  - Control unit
- Arithmetic and logic circuits in conjunction with the main control circuits are referred to as processor.
- Input and output equipment collectively referred to as Input Output(I/O) unit.
- Information handled by computer
  - Data
  - Instructions



- Instructions or machine instructions are explicit commands that
  - Govern transfer of information within computer or between computer and its I/O devices
  - Specify arithmetic and logical operations to be performed
  - List of instructions that performs a task is called a program
- Data
  - Numbers and encoded characters used as operands in instructions
  - Any digital information
  - Entire program considered as data if it is to be processed by another program
- **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

- **Memory Unit**
  - Stores data and programs
  - Two storage class – primary and secondary
- Primary Storage
  - Fast memory that operates at electronic speed
  - Programs stored here during execution
  - Groups of fixed size called word. Each word contains n bit called wordlength. Eg 16-bit word, 32-bit word..
  - One word can be stored or retrieved by one basic operation.
  - A distinct address is associated with each word location for easy access.
  - RAM, fixed amount of time to reach any word location
  - Memory hierarchy , cache (small, fast, closer to processor), main memory(large, slow unit)
- Secondary Storage
  - To store data and large programs accessed infrequently
  - Magnetic disks, tapes, optical disks(CD-ROMs)
- **Arithmetic and Logic Unit (ALU)**
  - ALU and CU forms the processor
  - Most operations are executed in ALU
  - Example – Addition, Subtraction, Multiplication, comparison of 2 numbers,etc.
  - Operands are brought into processor to perform required operation.
  - In processor, operands are stored in high-speed storage called registers.
  - Register can store one word and access time is fast.
- **Control Unit(CU)**
  - All units process information and perform operations.
  - The operation of these units are coordinated by CU.
  - Physically distributed throughout the machine. Control circuits carry signals for timing and synchronization of events in all units.

- The computer operation can be summarized as follows:
    - The computer accepts information in the form of data and program through an input unit and stores it in memory.
    - Information stored in memory is fetched and processed in ALU.
    - Processed information leaves the computer through an output unit.
    - All activities inside the machine are directed by control unit.

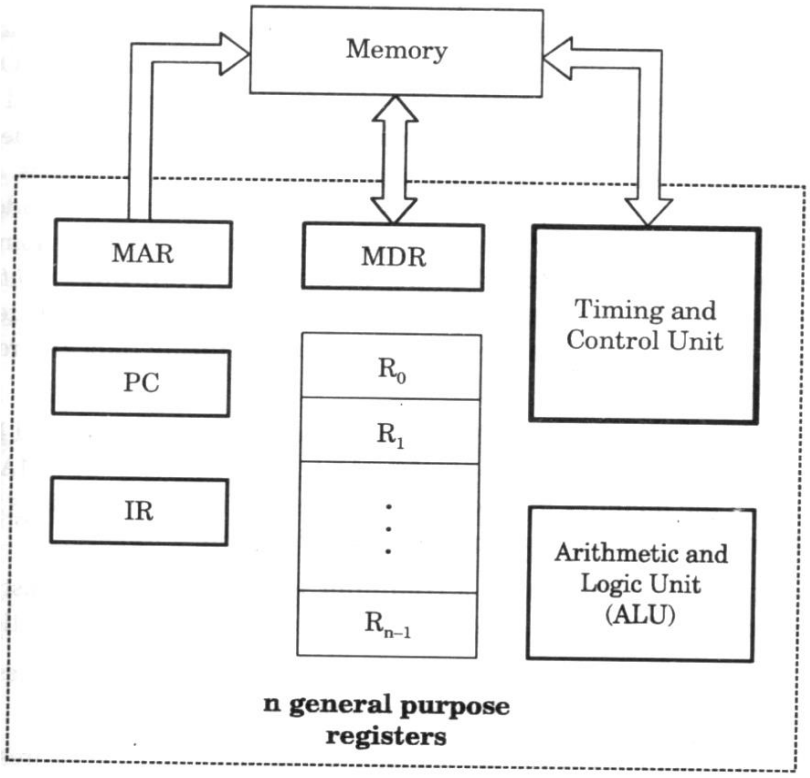
BASIC OPERATIONAL CONCEPTS

- Activity in a computer is governed by instructions. To perform a task, an appropriate program consisting of a list of instructions is stored in the memory.
- Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be used as operands are also stored in the memory.
- Example : Add LOCA, R0.
- Add the operand at memory location LOCA to the operand in a register R0 in the processor. Place the sum into register R0.
- The original contents of LOCA are preserved. The original contents of R0 is overwritten.
- Instruction is fetched from the memory into the processor – the operand at LOCA is fetched and added to the contents of R0 – the resulting sum is stored in register R0.
- Combines memory access operation with an ALU(add) operation.
- Separate Memory access and ALU operation.
- Load LOCA, R1
- Add R1, R0
- LOAD instruction – transfers the contents of memory location LOC A into register R1.
- ADD instruction – adds the contents of R1 with the contents in R0 and places the result in R0.
- Destroys contents in R0 and R1. Preserves the content in LOC A.

Registers in Processor

- Instruction Register(IR)
    - Holds the instruction that is being currently executed.
- Program Counter(PC)
    - Holds the memory address of the next instruction to be fetched and executed.
- Memory Address Register(MAR) and Memory Data Register(MDR)
    - Facilitate communication with memory.
    - MAR – holds the address of memory to be accessed.
    - MDR – contains the data to be written into or read out of the addressed location.
- N-general purpose registers R0 to Rn-1.
- **Execution of program**
    - PC set to point to the first instruction of the program. Contents of PC transferred to MAR.

- Read signal issued to the memory.
- Addressed word read from memory loaded into MDR. Contents of MDR transferred to IR. Instruction is ready to be decoded and executed.
- If instruction has operations to be performed by ALU then operands are needed.
- If operand is in register, perform ALU operation.
- If operand in memory, it has to be fetched by sending address to MAR and then initiating a read signal.
- Read operand is loaded in MDR. Then it is transferred to ALU.
- To store result in memory – result sent to MDR. Address specified in MAR. Write cycle initiated.
- During execution of current instruction, contents of PC incremented to point to the next instruction.

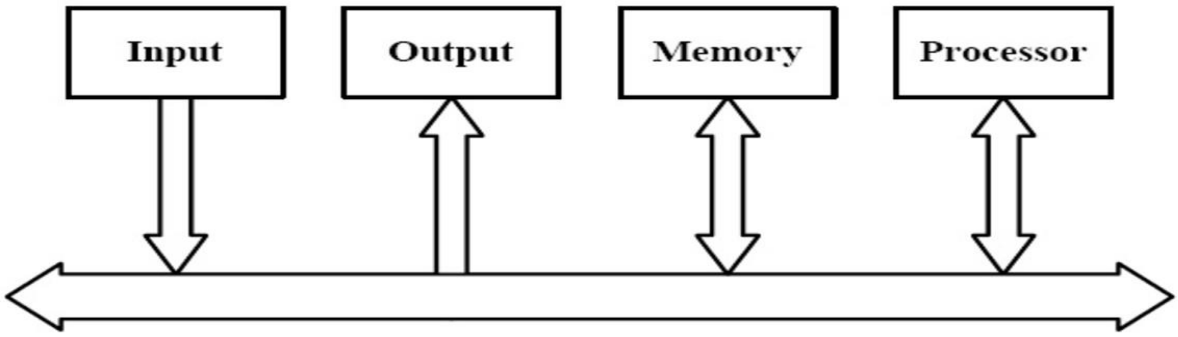


Interconnection between Processor and Memory

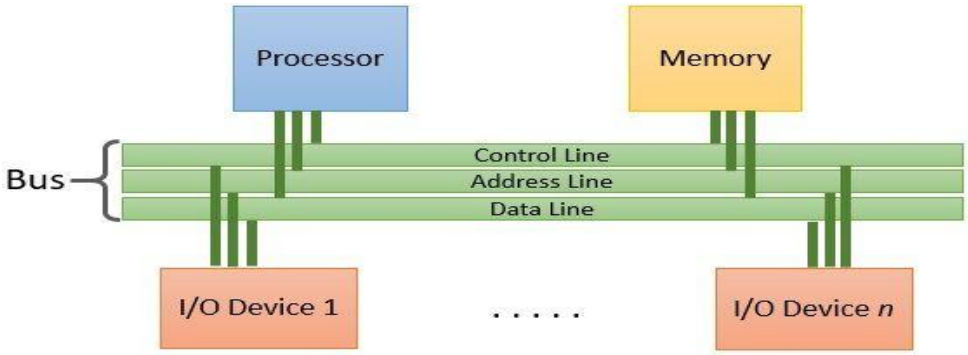
- **Interrupts**
  - Normal execution of program is preempted to provide urgent service to some device.
  - An interrupt is a request from I/O device for service by processor
  - Processor provides requested service by executing interrupt service routine (ISR)
  - Contents of PC, general registers, and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

BUS STRUCTURE

- Single bus – simplest way to connect all functional units.
- Only one transfer at a time, 2 units active at any given time.
- Advantage – low cost and flexibility to add peripheral devices.



- Multiple bus – achieve more concurrency in operations by allowing two or more transfers at the same time.
- Advantage – Improves performance
- Disadvantage – High cost
- Devices connected to a bus vary in their operation speed. Keyboard, printer have slow speed when compared to optical disk. Processors and memory operate at electronic speed.
- Solution – maintain a buffer register at the device to hold information during transfers.
- CPU and memory are normally connected by three groups of buses – data bus, address bus and control bus.

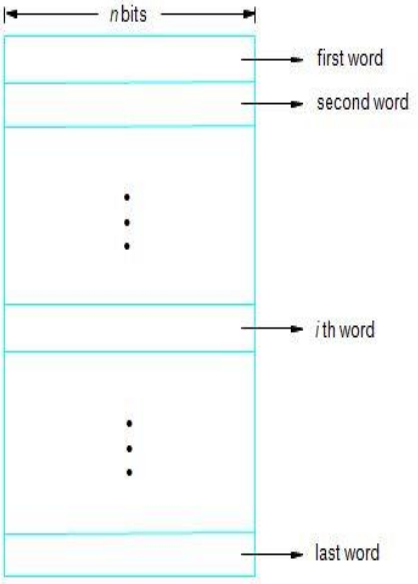


Bus Structure

MEMORY LOCATIONS AND ADDRESSES

- Number and character operands, as well as instructions, are stored in the memory of a computer.
- The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1.

- Because a single bit represents a very small amount of information the memory is organized so that a group of n bits can be stored or retrieved in a single, basic operation.
- Each group of n bits is referred to as a word of information, and n is called the word length.
- Modern computers have word lengths that typically range from 16 to 64 bits.
- Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or addresses for each item location.



- **Byte Addressability**
- There are three basic information quantities to deal with: the bit, byte and word. A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits.
- Address cannot be assigned to each bit. The most practical assignment is to have successive addresses refer to successive byte locations in the memory.
- The term byte-addressable memory is use for this assignment. Byte locations have addresses 0,1,2, ...
- Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0,4,8,...., with each word consisting of four bytes.

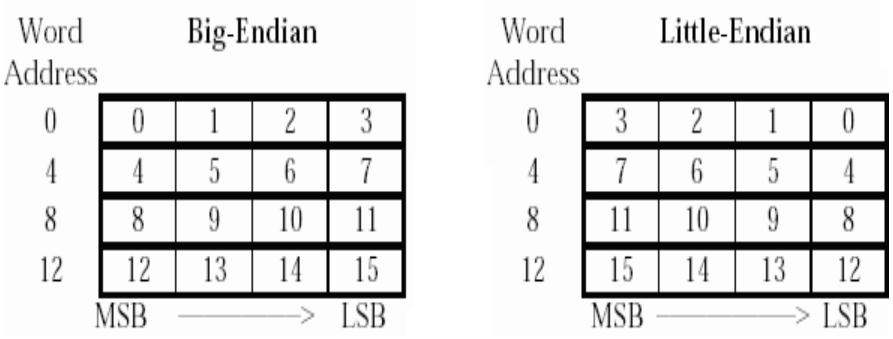
bit 31.....bit 0

Address 0	byte 3	byte 2	byte 1	byte 0
Address 4	byte 7	byte 6	byte 5	byte 4
Address 8	byte 11	byte 10	byte 9	byte 8
Address 12	byte 15	byte 14	byte 13	byte 12

BIG-ENDIAN AND LITTLE-ENDIAN ASSIGNMENTS

- There are two ways that byte addresses can be assigned across words,

- The name big-endian is used when lower byte addresses are used for the more significant bytes (the leftmost bytes) of the word.
- The name little-endian is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.



MEMORY OPERATIONS

- Both program instructions and data operands are stored in memory.
- For execution, the word(s) containing the instructions and data need to be transferred between the memory and processor.
- Two basic operations
  - Load(Read or Fetch)
  - Store (write)
- Load – transfers a copy of the contents of memory location to the processor. The memory contents remain unchanged.
- LOAD LOCA,R0
- Store- transfers information from the processor to a specific memory location, destroying the former content of that location.
- STORE R1,LOCB

INSTRUCTIONS AND INSTRUCTION SEQUENCING

- A computer must have instructions to perform the following:
  - Data transfer between memory and processor register
  - Arithmetic and logic operation
  - Program sequencing and control
  - I/O transfer
- Instructions
  - Instruction Notations
    - Register Transfer Notation
    - Assembly Language Notation
  - Instruction Types
    - Three address
    - Two address
    - One address
    - Zero Address

Register Transfer Notations

- Locations may be involved in data transfer. Eg, Memory locations, processor registers or registers in IO subsystem.
- Locations are specified in instructions by symbolic names.
- Examples:
  - LOC, A, PLACE – memory locations
  - R0,R1, …. – Processor registers
  - DATAIN, DATAOUT, INSTSTUS, OUTSTATUS – IO registers.
- The contents of a location are denoted by placing square brackets around the name of the location.
- 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 whereas Destination contents are overwritten
- Examples
  - R1←[LOC] – Copies the content in memory location LOC in register R1.
  - R3←[R1]+[R2] – Adds the content in registers R1 and R2 and stores the result in register R3.

Assembly Language Notation

- Another type of notation to represent machine instructions and program.
- Examples
  - Move LOC, R1 – Copies the content in memory location LOC in register R1.
  - Add R1,R2,R3 – Adds the content in registers R1 and R2 and stores the result in register R3.

Instruction Types

- Consider a high level language statement C=A+B
- In RTN , C←[A]+[B]
- On compilation, the variable A,B and C will be assigned with distinct memory locations. Programmers use the variables to refer the memory location addresses.
- **Three – address instruction**
- Syntax - Operation source1,source2,destination
- Example - Add A,B,C
- Disadvantage – Instruction is too large to fit in one word
- **Two-address instruction**
- Syntax - Operation source, destination
- Single two address instruction is not sufficient.
- Example – Add A,B this is B←[A]+[B] and not C←[A]+[B]
- Two-address instruction sequence
- Move B,C
- Add A,C
- **One-address instruction**
- Syntax – Operation source/destination
- Either the source or destination operand is mentioned. The other operand is implied to be a special processor register called Accumulator.

- Examples
- Add A #Acc ← [A] + [Acc]
- Load A # Acc ← [A]
- Store A # A ←[Acc]
- Example C=A+B
- Load A
- Add B
- Store C
- 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

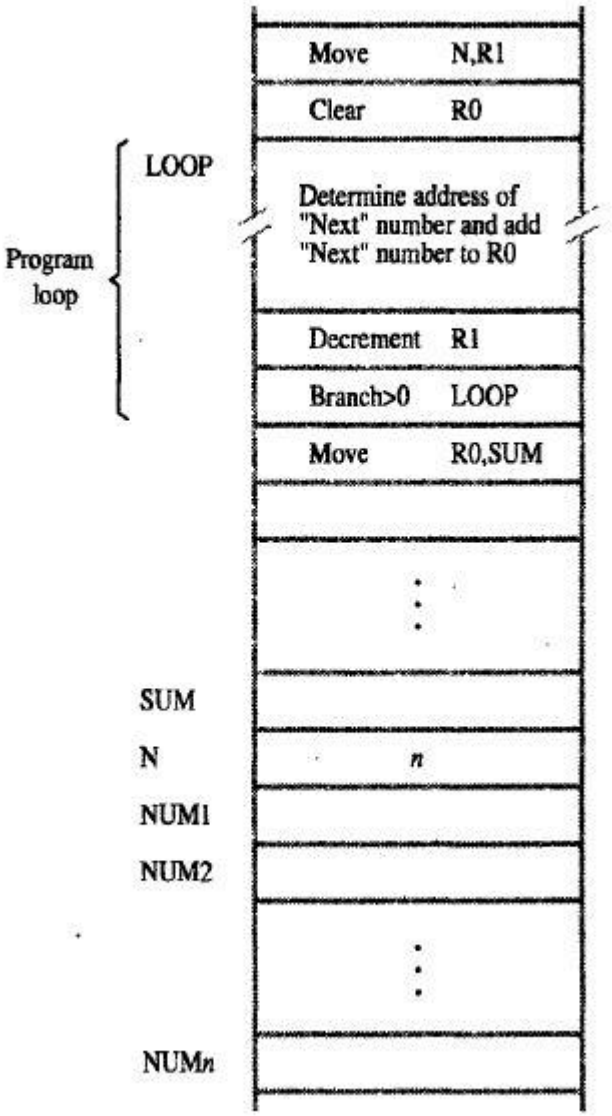
Instruction Execution and Straight-Line Sequencing

- Assume that the computer allows one memory operand per instruction and has a number of processor registers.
- The instructions of the program are in successive word locations, starting at location i. since each instruction is 4 bytes long, the second and third instructions start at addresses i + 4 and i + 8 and so on.
- Executing a given instruction is a two-phase procedure.
- In the first phase, called instruction fetch, the instruction is fetched from the memory location whose address is in the PC. This instruction is placed in the instruction register (IR) in the processor.
- In the second phase, called instruction execution, the instruction in IR is examined to determine which operation is to be performed. The specified operation is then performed by the processor. This often involves fetching operands from the memory or from processor registers, performing an arithmetic or logic operation, and storing the result in the destination location.

Branching

- Consider the task of adding a list of n numbers.
- Instead of using a long list of add instructions, it is possible to place a single add instruction in a program loop, as shown in figure.
- The loop is a straight-line sequence of instructions executed as many times as needed. It starts at location LOOP and ends at the instruction Branch > 0.
- During each pass through this loop, the address of the next list entry is determined, and that entry is fetched and added to R0.

- Assume that the number of entries in the list, n, is stored in memory location N, as shown.



- Register R1 is used as a counter to determine the number of time the loop is executed. Hence, the contents of location N are loaded into register R1 at the beginning of the program.
- Then, within the body of the loop, the instruction, Decrement R1, reduces the contents of R1 by 1 each time through the loop.
- This type of instruction loads a new value into the program counter.
- As a result, the processor fetches and executes the instruction at this new address, called the branch target, instead of the instruction at the location that follows the branch instruction in sequential address order.
- A conditional branch instruction causes a branch only if a specified condition is satisfied.
- If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.

- Branch > 0 LOOP (branch if greater that 0) is a conditional branch instruction that causes a branch to location LOOP if the result of the immediately preceding instruction, which is the decremented value in register R1, is greater than zero.
- This means that the loop is repeated, as long as there are entries in the list that are yet to be added to R0.
- At the end of the nth pass through the loop, the Decrement instruction produces a value of zero, and hence, branching does not occur.

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, often called condition code flags.
- These flags are usually grouped together in a special processor register called the condition code register or status register.
- Individual condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation performed.
- 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 ot1 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

- In general, a program operates on data that reside in the computer's memory.
- Programs are normally written in a high-level language, which enables the programmer to use constants, local and global variables, pointers, and arrays.
- The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

1. **Immediate Addressing mode –** The operand is given explicitly in the instruction. For example, the instruction

   Move $200_{immediate}$, R0, places the value 200 in register R0.

   - Clearly, the Immediate mode is only used to specify the value of a source operand.
   - Using a subscript to denote the Immediate mode is not appropriate in assembly languages.
   - A common convention is to use the sharp sign (#) in front of the value to indicate that this value is to be used as an immediate operand. Hence, we write the instruction above in the form

     Move #200, R0

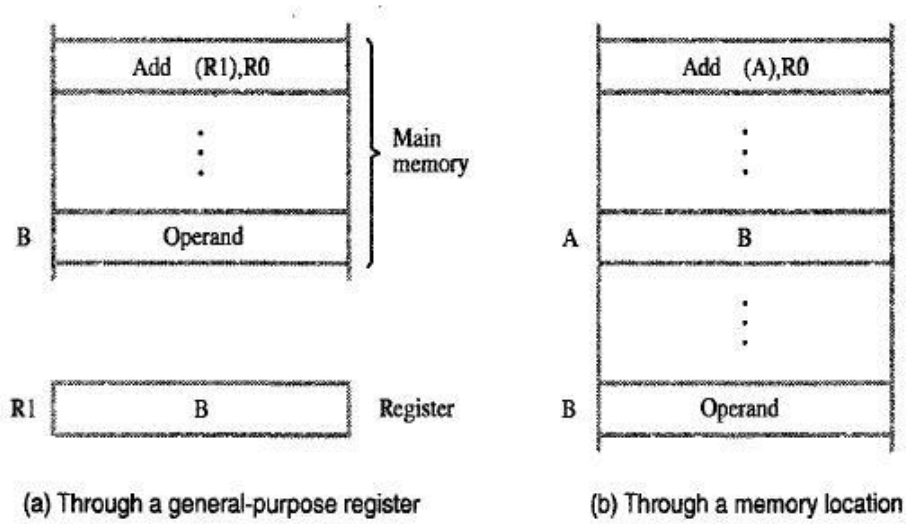
2. **Register Addressing mode -** The operand is the contents of a processor register; the name (address) of the register is given in the instruction.
3. **Absolute or Direct Addressing mode –** The operand is in a memory location; the address of this location is given explicitly in the instruction. (In some assembly languages, this mode is called Direct).

- The instruction Move LOC, R2 is an example with both Register and Absolute Addressing mode, where R2 represents register and LOC represents Absolute addressing mode.
- The above three addressing modes are used in implementation of constants and variables.

- In the addressing modes that follow, the instruction does not give the operand or its address explicitly. Instead, it provides information from which the memory address of the operand can be determined. We refer to this address as the effective address (EA) of the operand.

4. **Indirect Addressing mode**

- The effective address of the operand is the contents of a register or memory location whose address appears in the instruction
- Indirect addressing mode through registers and memory locations are possible.
- To execute the Add instruction in fig (a), the processor uses the value which is in register R1, as the effective address of the operand.

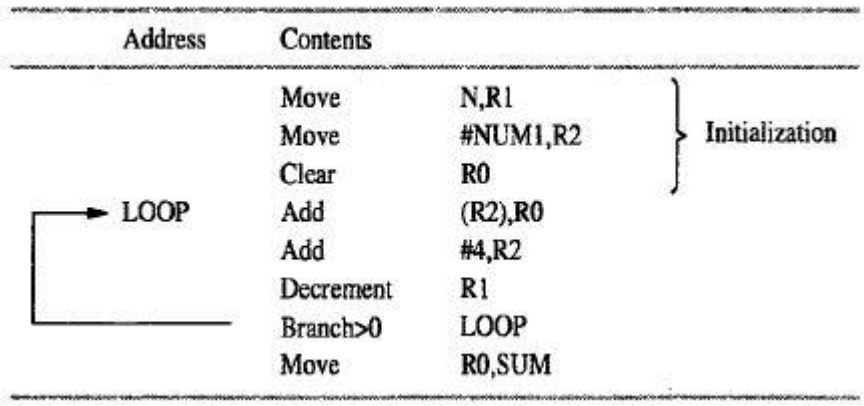


(a) Through a general-purpose register (b) Through a memory location

- It requests a read operation from the memory to read the contents of location B. the value read is the desired operand, which the processor adds to the contents of register R0.
- Indirect addressing through a memory location is also possible as shown in fig (b). In this case, the processor first reads the contents of memory location A,

then requests a second read operation using the value B as an address to obtain the operand

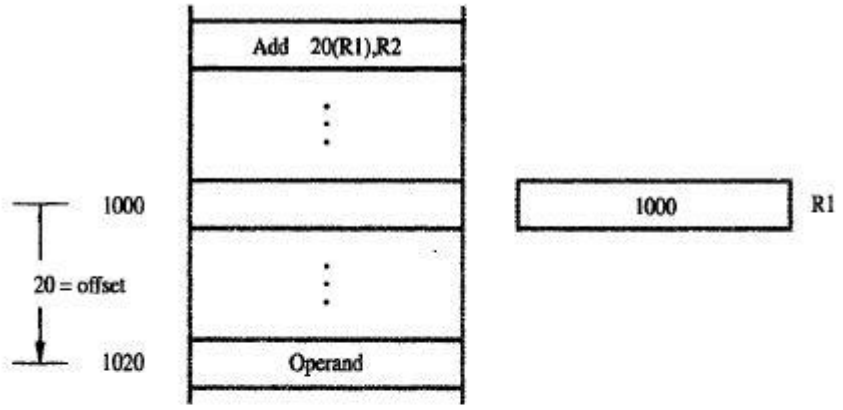
- The register or memory location that contains the address of an operand is called a pointer. This addressing mode implements pointer variables.
- **Example – Addition of list of numbers**



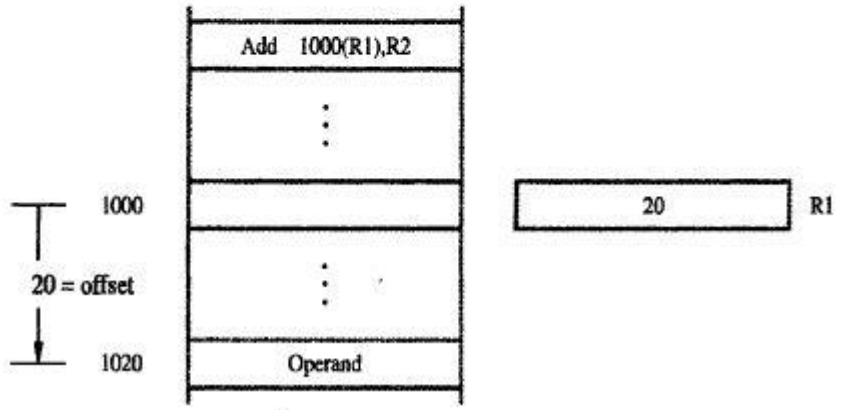
- In the program shown Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2.
- The initialization section of the program loads the counter value n from memory location N into R1 and uses the immediate addressing mode to place the address value NUM1, which is the address of the first number in the list, into R2.
- Then it clears R0 to 0.
- The first two instructions in the loop implement the unspecified instruction block starting at LOOP.
- The first time through the loop, the instruction **Add (R2), R0** fetches the operand at location NUM1 and adds it to R0.
- The second Add instruction adds 4 to the contents of the pointer R2, so that it will contain the address value NUM2 when the above instruction is executed in the second pass through the loop.

5. Indexed Addressing mode

- The effective address of the operand is generated by adding a constant value to the contents of a register. It is useful in accessing the operands in lists and arrays.
- The register use may be either a special register provided for this purpose or, may be any one of a set of general-purpose registers in the processor. In either case, it is referred to as index register.
- The Index mode is symbolically represented as X (Ri) where X denotes the constant value contained in the instruction and Ri is the name of the register involved.
- The effective address of the operand is given by EA = X + [Rj]



(a) Offset is given as a constant



(b) Offset is in the index register

- Figure illustrates two ways of using the Index mode. In fig a, the index register, R1, contains the address of a memory location, and the value X defines an offset (also called a displacement) from this address to the location where the operand is found.
- An alternative use is illustrated in fig b. Here, the constant X corresponds to a memory address, and the contents of the index register define the offset to the operand. In either case, the effective address is the sum of two values; one is given explicitly in the instruction, and the other is stored in a register.
- **Different versions of index addressing mode**
- Two registers can be used, one to store the base address and the other to store the offset. In this case, index addressing mode is represented as (Ri, Rj).
- The effective address is the sum of the contents of registers Ri and Rj. The second register is usually called the base register. This form of indexed addressing provides more flexibility in accessing operands, because both components of the effective address can be changed.
- Another version of the Index mode uses two registers plus a constant, which can be denoted as X(Ri, Rj). In this case, the effective address is the sum of the constant X and the contents of registers Ri and Rj.
- Example – To add the marks of three subjects for a list of n students.

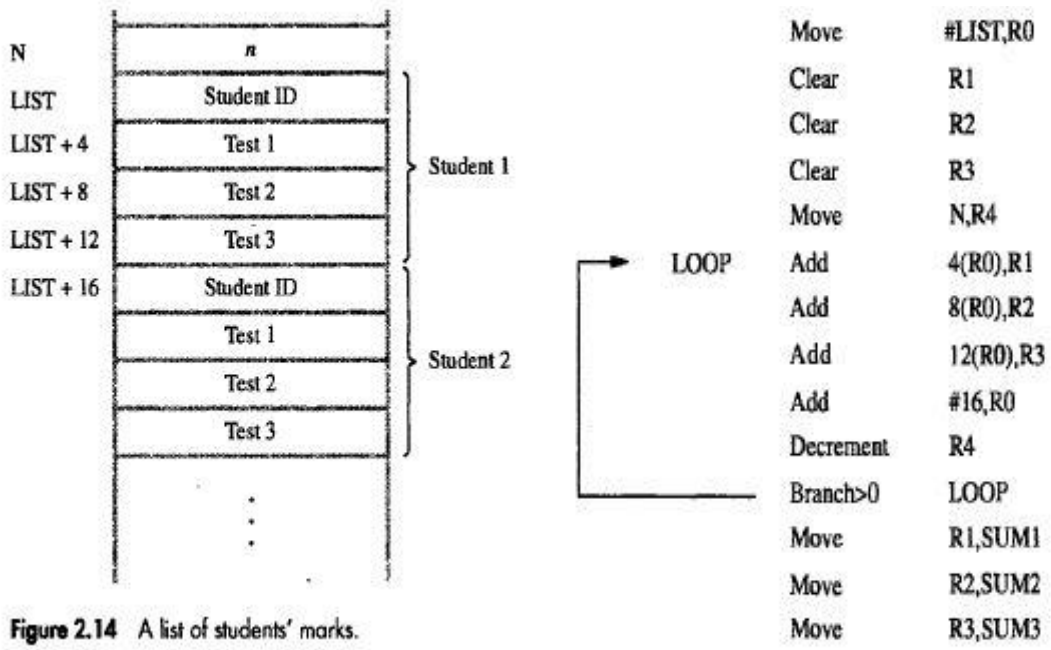


Figure 2.14 A list of students' marks.

- The program is to compute the sum of all scores obtained on each of the tests and to store the result in SUM1, SUM2 and SUM3.
- Register R0 is the index register. R0 is set to point to the ID location of the first student. Hence it contains the address LIST.
- On the first pass through the loop, the test scores of the first student are added to the running sums of registers R1,R2 and R3, which are initially cleared to 0.
- These scores are accessed through the Index addressing mode 4(R0), 8(R0) and 12(R0).
- The index register is incremented by 16 to point to ID location of second student.
- Register R4, initialized with n, is decremented by 1 at the end of each pass of the loop. When the contents of R4 reach 0, all student records have been accessed and the loop terminates.
- The last three instructions transfers the accumulated sums from register R1,R2 and R3 to memory locations SUM1,SUM2 and SUM3.

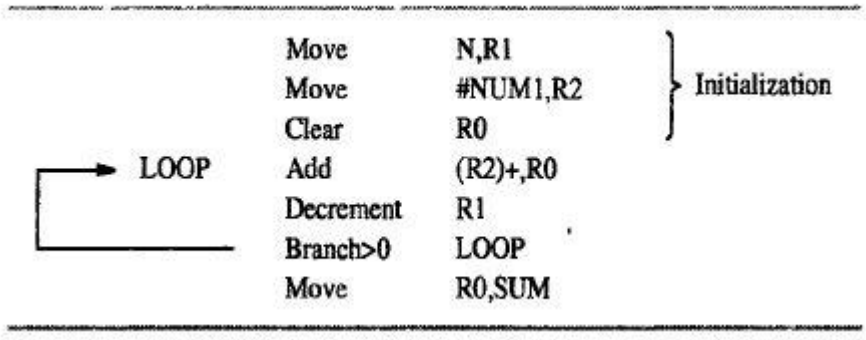
6. Relative Addressing mode

- The effective address is determined by the Index mode using the program counter in place of the general-purpose register Ri.
- Then, X(PC) can be used to address a memory location that is X bytes away from the location presently pointed to by the program counter.
- This mode can be used to access data operands. But, its most common use is to specify the target address in branch instructions.
- An instruction such as Branch > 0 LOOP causes program execution to go to the branch target location identified by the name LOOP if the branch condition is satisfied.

- This location can be computed by specifying it as an offset from the current value of the program counter. Since the branch target may be either before or after the branch instruction, the offset is given as a signed number.

7. Autoincrement and Autodecrement Addressing mode

- **Autoincrement 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 to point to the next item in a list. It is denoted as (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. It is denoted as -(Ri).
- **Example –** Addition of n numbers

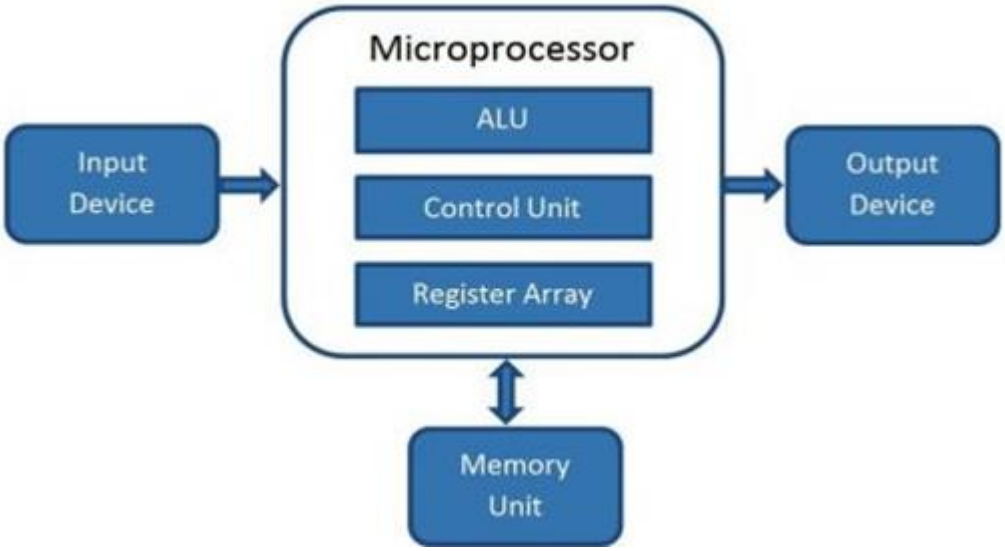


INTRODUCTION TO MICROPROCESSOR

- Computer's Central Processing Unit (CPU) built on a **single Integrated Circuit (IC)** is called a **microprocessor**. A digital computer with one microprocessor which acts as a CPU is called microcomputer.
- It is a programmable, multipurpose, clock -driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as output.
- The microprocessor contains millions of tiny components like transistors, registers, and diodes that work together.

Block Diagram of a Microcomputer

- A microprocessor consists of an ALU, control unit and register array. Where **ALU** performs arithmetic and logical operations on the data received from an input device or memory. Control unit controls the instructions and flow of data within the computer. And, **register array** consists of registers identified by letters like B, C, D, E, H, L, and accumulator.



Evolution of Microprocessors

We can categorize the microprocessor according to the generations or according to the size of the microprocessor:

First Generation (4 - bit Microprocessors)

- The first generation microprocessors were introduced in the year 1971-1972 by Intel Corporation. It was named **Intel 4004** since it was a 4-bit processor.
- It was a processor on a single chip. It could perform simple arithmetic and logical operations such as addition, subtraction, Boolean OR and Boolean AND.
- I 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. It was a first 8 - bit microprocessor which could perform arithmetic and logic operations on 8-bit words. It was Intel 8008, and another improved version was Intel 8088.

Third Generation (16 - bit Microprocessor)

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

Fourth Generation (32 - bit Microprocessors)

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

Fifth Generation (64 - bit Microprocessors)

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

Classification of Microprocessors

- Microprocessors are classified based on the word length, Instruction set Architecture and applications.

- **Word Length**
  - According to the word length, microprocessors are classified as
    - 4 - bit microprocessor
    - 8 - bit microprocessor
    - 16 - bit microprocessor
    - 32 - bit microprocessor
    - 64 - bit microprocessor
- **Instruction Set Architecture**
  - **Complex Instruction Set Computer (CISC) –** CISC or Complex Instruction Set Computer is a computer architecture where instructions are such that a single instruction can execute multiple low-level operations like loading from memory, storing into memory, or an arithmetic operation, etc. It has multiple addressing nodes within a single instruction. CISC makes use of very few registers.
  - **Reduced Instruction Set Computer (RISC) –** RISC or Reduced Instruction Set Computer is a computer architecture where instruction is simple and designed to get executed quickly. Instructions get completed in one clock cycle this is because of the optimization of instructions and pipelining (a technique that allows for simultaneous execution of parts, or stages, of instructions more efficiently process instructions). RISC makes use of multiple registers to avoid large interactions with memory. It has few addressing nodes.
- **Application**
  - General pupose Microprocessor
  - Application specific microprocessor (ASIC)

INTRODUCTION TO ASSEMBLY LANGUAGE

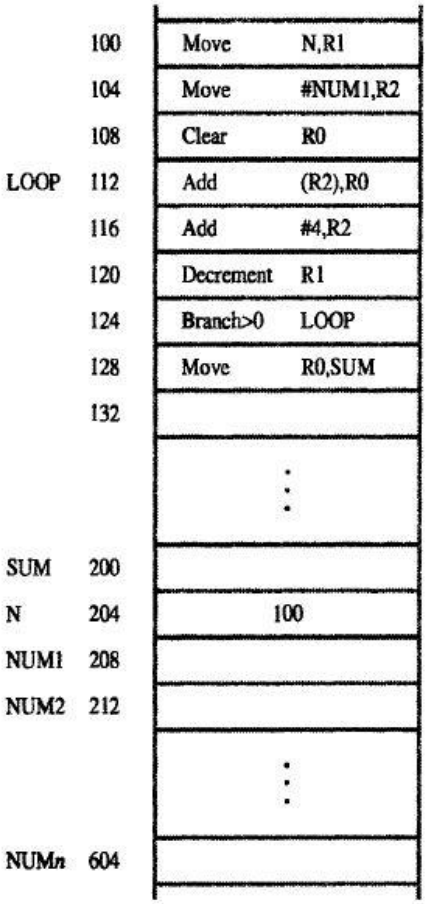
- Machine instructions are represented by patterns of 0s and 1s. Such patterns are awkward to deal with when discussing or preparing programs.

- Therefore, symbolic names are used to represent the pattern. So far, we have used normal words, such as Move, Add, Increment, and Branch, for the instruction operations to represent the corresponding binary code patterns.
- When writing programs for a specific computer, such words are normally replaced by acronyms called mnemonics, such as MOV, ADD, INC, and BR.
- Similarly, we use the notation R3 to refer to register 3, and LOC to refer to a memory location.
- A complete set of such symbolic names and rules for their use constitute a programming language, generally referred to as an assembly language.
- Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an assembler.
- When the assembler program is executed, it reads the user program, analyzes it, and then generates the desired machine language program. The latter contains patterns of 0s and 1s specifying instructions that will be executed by the computer.
- The user program in its original alphanumeric text format is called a source program, and the assembled machine language program is called an object program.
- Most assembly language require statements in a source program to be written in the form

  Label Operation Operand(s) Comment
- These 4 fields are separated by an appropriate delimiter, typically one or more blank characters.
- The Label is an optional name associated with the memory address where the machine language instruction produced from the statement will be loaded.
- The Operation field contains the opcode mnemonic of the desired instruction or assembler directive.
- The Operand field contains the addressing information for accessing one or more operands, depending on the type of instruction.
- The Comment field is ignored by assembler program. It is used for documentation purpose.
- **Assembler Directives**
- In addition to providing a mechanism for representing instructions in a program, the assembly language allows the programmer to specify other information needed to translate the source program into the object program.
- We have already mentioned that we need to assign numerical values to any names used in a program.
- Suppose that the name SUM is used to represent the value 200. This fact may be conveyed to the assembler program through a statement such as SUM EQU 200
- This statement does not denote an instruction that will be executed when the object program is run; in fact, it will not even appear in the object program.
- It simply informs the assembler that the name SUM should be replaced by the value 200 wherever it appears in the program. Such statements, called assembler directives (or commands), are used by the assembler while it translates a source program into an object program.

WRITING OF ASSEMBLY LANGUAGE PROGRAM

- To illustrate the assembly language program, let us consider the example program to add n numbers.
- The first figure shows the memory addresses where the machine instructions and the required data items are to be found after the program is loaded for execution.
- If the assembler is to produce an object program according to this arrangement, it has to know
  - How to interpret the names
  - Where to place the instructions in the memory
  - Where to place the data operands in the memory



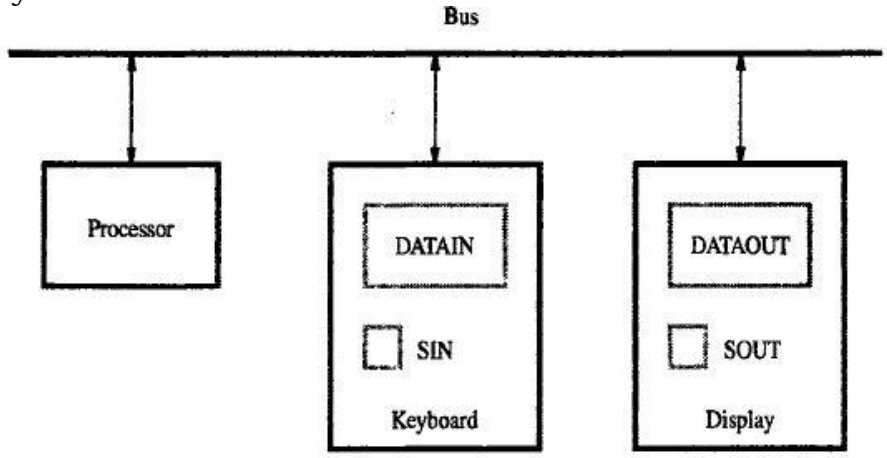
	Memory address label	Operation	Addressing or data information
Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
		ORIGIN	100
Statements that generate machine instructions	START	MOVE	N,R1
		MOVE	#NUM1,R2
		CLR	R0
	LOOP	ADD	(R2),R0
		ADD	#4,R2
		DEC	R1
		BGTZ	LOOP
		MOVE	R0,SUM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

- To provide the entire needed information source program may be given as in second figure.
- The program begins with assembler directives. The equate directive, EQU informs the assembler about the value sum.
- The second directive, ORIGIN, tells the assembler program where in the memory to place the data blocks that follows. In this case, the location specified has the address 204.

- Since this location is to be loaded with value 100, a DATAWORD directive is used to inform the assembler of this requirement. It states that the data value 100 is to be placed in the memory word at address 204.
- The RESERVE directive declares that a memory block of 400 bytes is to be reserved for data, and that the name NUM1 is to be associates with address 208.
- The second ORIGIN specifies that the instructions of the object program are to be loaded in the memory starting at address 100.
- The last statement is the assembler directive END, which tells the assembler that this is the end of source program text. The END directive includes label START which is the address of the location at which the execution of the program is to begin.
- RETURN assembler directive identifies the point at which execution of the program should be terminated. It causes the assembler to insert an appropriate machine instruction that returns control to the operating system of the computer.

BASIC I/O OPERATIONS

- Input/Output(I/O) operations are essential for data transfer between the memory of a computer and the outside world. A simple way of performing such I/O task is to use a method known as program controlled I/O.
- The rate of data transfer and data processing is much less in input and output devices when compared to the processing speed of a processor. The difference in speed between the processor and I/O devices creates the need for mechanisms to synchronize the transfer of data between them.



- A solution to this problem is as follows:
- On output, the processor sends the first character and then waits for a signal from the display that the character has been received. It then sends the second character and so on.
- Input is sent from the keyboard in a similar way; the processor waits for a signal indicating that a character key has been struck and that its code is available in some buffer register associated with the keyboard. Then the processor proceeds to read the code.

- **Moving a character code from the keyboard to the processor.**
- Striking a key stores the corresponding character code in an 8-bit buffer register named DATAIN.
- To inform the processor that a valid character is in DATAIN, a status control flag, SIN, is set to 1.
- A program monitors SIN, and when SIN is set to 1, the processor reads the contents of DATAIN.
- When the character is transferred to processor, SIN is automatically cleared to 0.
- **Moving a character from the processor to the display**
- A buffer register DATAOUT and status flag SOUT is used for this transfer.
- When SOUT is 1, the display device is ready to receive a character. Under program control, the processor monitors SOUT, and when SOUT is set to 1, the processor transfers a character code to DATAOUT.
- The transfer of a character to DATAOUT clears SOUT to 0; when the display device is ready to receive next character, SOUT is again set to 1.
- The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry commonly known as device interface.
- In order to perform I/O transfers, machine instructions are needed that can check the state of the status flags and transfer data between the proessor and I/O device.
- Most of the computers use an arrangement called memory mapped I/O in which some memory address values are used to refer to peripheral device buffer registers such as DATAIN and DATAOUT. Instructions like move, store and load can be used for data transfer.
- The contents of the keyboard buffer DATAIN can be transferred to register R1 in the processor by the instruction

  MoveByte DATAIN, R1
- Similarly, the contents of R1 can be transferred to DATAOUT by the instruction

  MoveByte R1, DATAOUT
- The MoveByte operation code signifies that the operand size is a byte, to distinguish it from the operation code Move that is used for word operands.
- The status flags SIN and SOUT are included in the device status registers. Bit b3 in the registers INSTATUS and OUTSTATUS corresponds to SIN and SOUT respectively.
- The read operation may be implemented by the machine instruction sequence

```
READWAIT  Testbit     #3,INSTATUS
          Branch=0    READWAIT
          MoveByte    DATAIN,R1
```

- The write operation may be implemented as

```
WRITEWAIT  Testbit     #3,OUTSTATUS
           Branch=0    WRITEWAIT
           MoveByte    R1,DATAOUT
```

- The Testbit instruction tests the state of one bit in the destination location, where the bit position to be tested is indicated by the first operand.
- If the bit tested is equal to 0, then the condition of the branch instruction is true, and a branch is made to the beginning of the wait loop.
- When the device is ready, that is, when the bit tested becomes equal to 1, the data are read from the input buffer or written into the output buffer.