COMPUTER ORGANIZATION AND ARCHITECTURE

# MODULE 1

**(Syllabus)**

**Basic Structure of Computers:** Basic operational concepts, Bus Structures, Performance- Processor Clock, Basic performance equation, Clock rate, Performance Measurement

**Machine Instructions and Programs:** Memory Location and Addresses, Memory operations, Instructions and Instruction sequencing, Addressing Modes

## BASIC STRUCTURE OF COMPUTERS

### Define a computer. Explain the different types of computer with example.

A computer can be defined as a fast electronic calculating machine that accepts the (data) digitized input information, process it as per the list of internally stored instructions and produces the resulting information.

Types of Computers:

1. **Personal computers: -** This is the most common type found in homes, schools, Business offices etc., It is the most common type of desk top computers with processing and storage units along with various input and output devices that can all be located easily on a home or office desk.
2. **Note book computers: -** These are compact and portable versions of Personal computer with all of these components packaged into a single unit the size of a thin briefcase. ex: Laptop, Palm top pc etc.
3. **Work stations: -** These have high resolution input/output (I/O) graphics capability, but with same dimensions as that of desktop computer. These are used in engineering applications of interactive design work.
4. **Enterprise systems: -** These are used for business data processing in medium to large corporations that require much more computing power and storage capacity than work stations. Mainly used in Railway reservation systems, Insurance company.
5. **Server Systems:** - Servers contain sizable database storage units and are capable of handling large volumes of requests to access the data. In many cases servers are widely accessible to the education, banking system, business, and personal user communities. The request and responses are usually transported over Internet communication facilities. Internet associated with servers have become a dominant worldwide source of all types of information.
6. **Super computers: -** These are used for large scale numerical calculations required in the applications like weather forecasting , aircraft engineering etc.,

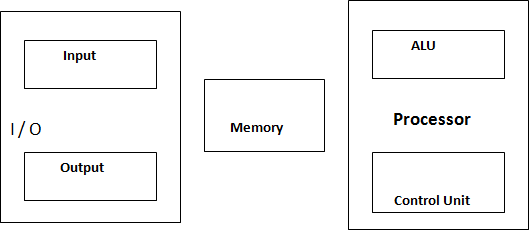
### Describe the basic functional units of a computer using a schematic.

**OR**

### Explain the different functional units of a digital computer.

A computer consists of five functionally independent main parts

1. Input Unit
2. Output Unit
3. Memory Unit
4. Arithmetic logic unit (ALU)
5. Control Unit.



### Input unit: -

The data is fed to a computer through input devices, keyboard is a most common type. Whenever a key is pressed, one corresponding word or number is translated into its equivalent binary code over a cable & fed either to memory or processor.

Example: Joysticks, trackballs, mouse, scanners etc are other input devices.

### Memory unit:

Its function is to store programs and data. It is basically of two types

* 1. Primary memory
  2. Secondary memory

*Primary memory***: -** Is the one exclusively associated with the processor and operates at the electronics speeds, programs must be stored in this memory while they are being executed. The memory contains a large number of semiconductors storage cells. Each capable of storing one bit of information. These are

processed in a group of fixed size called word. To provide easy access to a word in memory, a distinct address is associated with each word location. Addresses are numbers that identify memory location. Number of bits in each word is called word length of the computer. Programs must reside in the memory during execution. Instructions and data can be written into the memory or read out under the control of processor. Memory in which any location can be reached in a short and fixed amount of time after specifying its address is called random-access memory (RAM). The time required to access one word is called memory access time. Memory which is only readable by the user and contents of which can‟t be altered is called read only memory (ROM) and it contains operating system. Caches are the small fast RAM units, which are coupled with the processor and are often contained on the same IC chip to achieve high performance. Although primary storage is essential it tends to be expensive.

*Secondary memory:* Is used where large amounts of data & programs have to be stored, particularly information that is accessed infrequently.

Examples: **-** Magnetic disks & tapes, optical disks (ie CD-ROM‟s), floppies etc.,

### Arithmetic and logic unit (ALU):

Most of the computer operations are executed in ALU of the processor, like addition, subtraction, division, multiplication, etc. the operands are brought into the ALU from memory and stored in high speed storage elements called register. Then according to the instructions the operation is performed in the required sequence. The control and the ALU are many times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices such as key boards, displays, magnetic and optical disks, sensors and other mechanical controllers.

### Output unit:-

Output unit is the counterpart of input unit. Its basic function is to send the processed results to the outside world.

Examples:- Printer, speakers, monitor etc.

### Control unit:

Control unit is effectively the nerve center that sends signals to other units and senses their states. The actual timing signals that govern the transfer of data between input unit, processor, memory and output unit are generated by the control unit.

### BASIC OPERATIONAL CONCEPTS

The memory stores information relevant to a task to be performed. The information is stored as a binary pattern of 0‟s and 1‟s in every location sequentially. The information can be data (Operand) or instruction. For doing any operation, processor must get the instruction code form memory. Later, the processor may read data from memory if required. If data are already present in processor registers, memory access is not required. Processor fetches the instruction from memory one by one sequentially and performs the required operations. After computation the results are sent to the output unit or it may also be stored in memory. Thus the complete operation is determined by the stored program in memory.

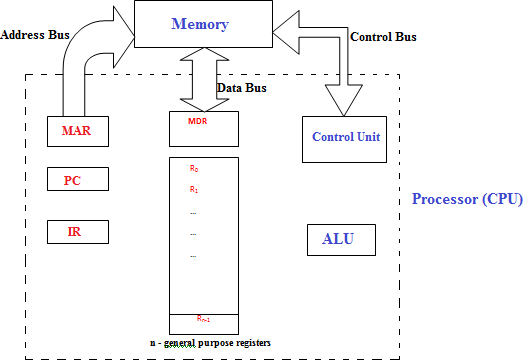
### \*\*\*\*\*With neat diagram explain the basic operational concepts of a computer

**OR**

### \*\*\*\*\*With neat diagram explain different processor registers.

**OR**

### \*\*\*\*\*Draw the connection between processor and memory and mention the functions of each component in the connection



Processor contains the following registers along with ALU and control unit.

1. Instruction register (**IR**) holds the instruction code (Opcode) that is currently being executed. This instruction code is passed to Instruction decoder in the control unit and then decoded to generate timing signals for different operations.
2. Program Counter (PC) contains the address of the next instruction to be fetched and executed. PC contents are automatically updated during the execution of an instruction. That means PC is always pointing to the next instruction.
3. n- General purpose registers: The processor contains R0, R1, R2, R3,……………….Rn-1 general purpose registers. These registers are used for temporarily storing the data and results of any computation.
4. Memory Address Register (MAR): holds the address of the memory location to be accessed.
5. Memory Data Register (MDR): contains the data to be written into or read from the addressed location.

The complete operation in computer is governed by the instructions stored in memory. ***The steps involved in execution of an instruction are:***

* The execution of the program starts by loading the address of the first instruction into PC
* The contents of PC are transferred to MAR and the address is sent out on the address bus. A

„Read‟ control signal is issued to the memory to read instruction code. The instruction code is read from memory and loaded into MDR. Next the contents of MDR are transferred to the IR. Simultaneously PC gets incremented.

* The instruction code is decoded to decide about the operation. If data (operands) needed for the operation is already available in general purpose registers, the operation is immediately carried out.
* If the operands reside in memory, then respective addresses are sent from MAR to memory and the data are read into MDR one by one. From MDR, they are passed directly to ALU or to any other general purpose register in the processor as needed. Then the operation is performed.
* If the result is to be stored in memory, then the result is sent to the MDR and address of the memory location where the result is to be stored is sent to the MAR. A ‟Write‟ control signal is issued and then the result is stored in memory.

List the steps needed to execute the machine instruction **ADD LOCA, R0** in terms of transfer between the processor and the memory along with some simple control commands. Assume that the instruction itself is stored in the memory at location INSTR and that this address is initially in register PC. The first two steps might be expressed as:

* + Transfer the contents of register PC to register MAR
  + Issue a read command to the memory and then wait until it has transferred the requested word into register MDR.

Remember to include the steps needed to update the contents of PC from INSTR to INSTR + 1 so that the next instruction can be fetched.

*Steps needed to execute the machine instruction* ***ADD LOCA, R0***

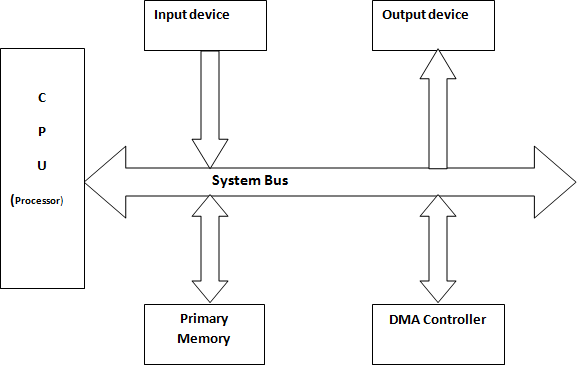
1. Transfer the contents of register PC (address INSTR) to MAR register and this address is sent out on the address bus.
2. Issue a read command to memory and then wait until it has transferred the requested word into register MDR.
3. Transfer the instruction from MDR into IR and decode it
4. Transfer the address LOCA from IR to MAR
5. Issue a read command to memory and wait until MDR is loaded.
6. Transfer the contents of MDR to the ALU
7. Transfer the contents of R0 to the ALU
8. Perform addition of the two operands in the ALU and then transfer the result into R0
9. Transfer contents of PC to ALU
10. Add **1** to operand in ALU and transfer incremented address to PC

### BUS STRUCTURES

Draw single bus structure. Discuss about memory mapped I/O

### OR

What is a Bus? Explain a single bus structure architecture



Bus is a group of wires which carry electrical signals from one end to the other. Generally, bus is used to connect all the functional units of a computer like CPU, Memory and I/O devices. Here electrical signal corresponds to binary information (0 or 1) sent from one device to the other. A bus which connects various functional units of a computer is referred to as “system bus”. To achieve a reasonable speed of operation, a computer must be organized so that all its units can handle one full word of data at a given time. In addition to the lines that carry the data, the bus must have lines for address and control purpose.

1. Address bus is unidirectional which carries address of the memory or any I/O device connected to processor.
2. Control Bus is unidirectional, which is used to arbitrate multiple requests for use of the bus.
3. Data bus is bidirectional, which is used to carry data from or to I/O device or memory to the processor.

The devices connected to a bus vary widely in their speed of operation. In order to synchronize the speed of the processor and I/O device, a common approach is to include buffer register with the devices to hold the information during transfers. For example processor sends the encoded character over the bus to the printer buffer. Once buffer is loaded, the printer can start printing without further

intervention by the processor. The bus and processor no longer needed and can be released for some other activity. Printer continues printing the character in its buffer and is not available for further transfers until this process is completed. This allows processor to switch rapidly from one device to another for data transfers involving several I/O devices.

Memory Mapped I/O:

I/O device is also considered as memory and the memory addresses are used to refer to the buffers and registers of the I/O device. The total address space of the computer is divided between memory and I/O device. Data transfer takes place through the use of normal instructions like MOVE, LOAD etc. and no special instructions required to access the contents of buffer registers.

Disadvantages of memory mapped I/O:

1. Allocating large space, is a wastage of memory using this method, since the number I/O devices connected are less.
2. There may be conflict between I/O and memory if the correct addresses are not specified.

### PERFORMANCE

The most important measure of the performance of a computer is how quickly it can execute programs. The speed with which a computer executes program is affected by the design of its hardware. For best performance, it is necessary to design the compilers, the machine instruction set, and the hardware in a coordinated way. The total time required to execute the program is elapsed time is a measure of the performance of the entire computer system. It is affected by the speed of the processor, the disk and the printer. The time needed to execute instruction is called the processor time.

### PROCESSOR CLOCK

Processor circuits are controlled by a timing signal called clock. The clock designer the regular time intervals called clock cycles. To execute a machine instruction the processor divides the action to be performed into a sequence of basic steps that each step can be completed in one clock cycle. The length P of one clock cycle is an important parameter that affects the processor performance.

Processors used in today‟s personal computer and work station have a clock rates that range from a few hundred million to over a billion cycles per second.

### CLOCK RATE

**Clock rate R = 1/P (cycles per second or Hz)**

The length of one clock cycle is P

What are the factors that affect processor performance? Explain any 4. Performance of computer system is given by the equation:



The factors that affect the performance of processor:

* 1. Processor clock rate (R)
  2. Number of instructions available. (N)
  3. Average time required to execute an instruction (T)
  4. Average number of basic steps required for each instruction (clock cycles per instruction)
  5. Memory access time
  6. Power dissipation in the system
  7. Number of I/O devices connected, its type, data transfer capacity of the bus etc.

Processor clock rate (R): By increasing the clock rate, execution speed can be reduced. Clock rate can be increased by improving IC technology, makes logic circuits faster, which reduces the time needed to complete a basic step. Also by reducing the amount of processing done in one basic step can increase the clock rate.

The value of N is reduced if the source program is compiled into less number of machine instructions. S value can be reduced if each instruction has small number of basic steps to be executed.

Write the basic performance equation. Explain the role of each of the parameters in the equation on the performance of the computer.

The basic Performance equation is given by

*T*  *N*  *S*

*R*

Here the complete execution of the program requires the execution of **N** number of machine language instruction.

**S** be the average number of basic steps required to execute one machine instruction, where each basic step is completed in one clock cycle.

Let **R** be the processor clock rate in cycles per second.

From the above equation it is clear that, to achieve greater performance, designer has to reduce the value of **T**. This can be done by reducing **N** and **S** and / or increasing the value of **R**

The value of **N** is reduced if the source program is compiled into less number of machine instructions.

**S** value will be reduced if each instruction has small number of basic steps to be executed. Using higher frequency clock, the value of **R** can be increased.

### A program contains 1000 instructions. Out of which 25% instructions requires 4 clock cycle, 40% instructions requires 5 clock cycles and remaining requires 3 clock cycles for execution. Find the total time required to execute the program running in a 1GHz machine.

Total time required to execute the program Number of instructions N = 1000

*T*  *N*  *S*

*R*

Processor clock rate R = 1GHz = 1 x 109, therefore the cycle time = **1**/ **R** = 1 x 10-9 sec

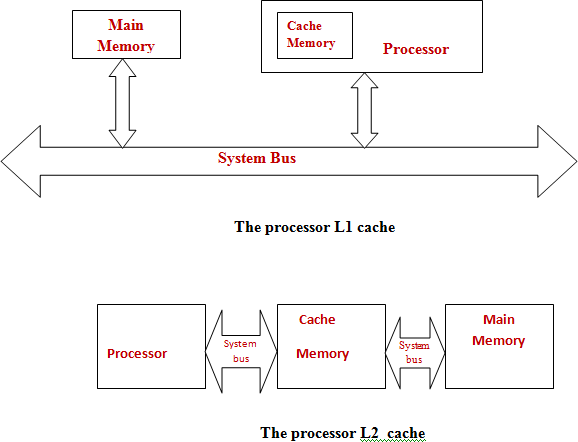
The number of clock cycles per instruction in each basic step S = 0.25 x 4 + 0.4 x 5 + 0.35 x 3 = **4.05**

Total time required to execute the program = 1000 x 4.05 x 1 x 10-9 = **4.05 µsec**

### PERFORMANCE IMPROVEMENT

Explain how performance of computer system can be improved.

An important measure of performance is the memory access time of the CPU. The access time is reduced if the high speed memories are designed to store data. Such a fast memory is called Cache which has lesser capacity than main memory. Cache can be built within the processor (**called L1 cache**) or can be placed between the main memory and the CPU (**called L2 Cache**) as shown in below figure.



Since the cache size is small, only those instructions and data which are required by the processor in the immediate future are stored in cache. As the execution proceeds, new set of programs can be loaded into the cache by swapping the previous contents back into main memory. That is, the CPU can fetch the instructions and data directly from cache instead of accessing main memory every time. This will reduce the memory access time and hence improve the performance.

Performance can also be improved by executing the instructions in overlapped fashion called Pipelining. Thus pipelining increases the throughput of the system.

### PERFORMANCE MEASUREMENT

What is performance measurement? Explain the overall SPEC rating for a computer in a program suite.

### OR

Explain how the performance of the computer can be measured

The performance measurement in a computer system is the time taken by the computer to execute a given standardized programs called bench mark programs. An organization called SPEC- System Performance Evaluation Corporation selects and publishes bench mark programs for different application domains such as game playing, compiler, data base applications, real time applications, numerically computations etc. To evaluate the performance of a computer, we need to run several programs each with different

degree of complexity, different memory requirements etc. The program is compiled for the computer under test, and running time on a real computer is measured. The same program is compiled and run on the reference computer maintained by SPEC such as UltraSPARC10 work station with 300MHz Ultra SPARC-IIi processor for SPEC 2000. Thus the SPEC rating is measured using the equation:

#### SPEC rating = Running time on the reference computer Running time on the computer under test

The higher the SPEC ratio gives the better performance of the system.

The test is repeated for all the benchmark programs in the SPEC suit, and the geometric mean of the result is computed. Thus the overall SPEC rating for comparing two computers using SPEC ratio is given by:

1

 *n*  *n*

Overall SPEC rating =   *SPECi* 

 *i*1 

Where „n‟ = number of benchmark programs in suite.

### Assume that the reference computer is ULTRA SPARCIO work station with 300 MHz ultra SPARC processor. A company has to purchase 1000 new computers, hence ordered testing of new computer with SPEC 2000. Following observations were made.

|  |  |  |
| --- | --- | --- |
| **Programs** | **Run time on reference computer** | **Run time in new computer** |
| 1 | 50 minutes | 5 minutes |
| 2 | 75 minutes | 4 minutes |
| 3 | 60 minutes | 6 minutes |
| 4 | 30 minutes | 3 minutes |

**The company system manager will place the order for purchasing new computers only if the overall SPEC rating is at least 12. After the said test, will the system manager place order for purchase of new computer**.

The overall SPEC rating is the Geometric mean of each result.

1

Overall SPEC rating =  *n SPECi*  *n*

 



 *i*1 

n = 4 programs.



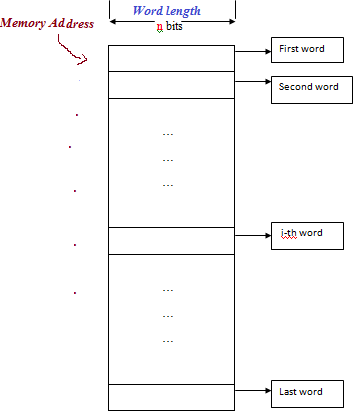
= (10 x 18.75 x 10 x 10)1/4

= **11.70**

Since the overall SPEC rating is less than benchmark rating 12, the purchase orders will not be issued.

### 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, bits are seldom handled individually. The usual approach is to deal with them in groups of fixed size. For this purpose, 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***. The memory of a computer can be schematically represented as a collection of words as shown in figure .



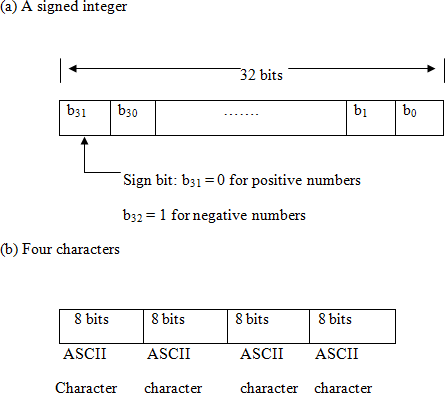
Modern computers have word lengths that typically range from 16 to 64 bits. If the word length of a computer is 32 bits, a single word can store a 32-bit 2‟s complement number or four ASCII characters, each occupying 8 bits. A unit of 8 bits is called a byte.

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. It is customary to use numbers from 0 through 2K-1, for some suitable values of k, as the addresses of successive locations in the memory. The 2k addresses constitute the address space of the computer, and the memory can have up to 2k addressable locations. 10-bit address generates an address space of 210 (1024 bytes) locations. A 30-bit addresses creates an address space of 230 or 1G (1 giga) locations.

Word length = 64bits, Successive memory addresses are: 0, 8, …… Word length = 32bits, Successive memory addresses are: 0, 4, 8,……

Word length = 16bits, Successive memory addresses are: 0, 2, 4, 6, 8,……

Word length = 8bits, Successive memory addresses are: 0, 1, 2, 3, 4,……



### BYTE ADDRESSABILITY

Write a short note on Byte addressability.

### OR

Discuss the two ways in which Byte addresses are assigned across the words.

A byte is always 8 bits, but the word length typically ranges from 8 to 64 bits. 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.

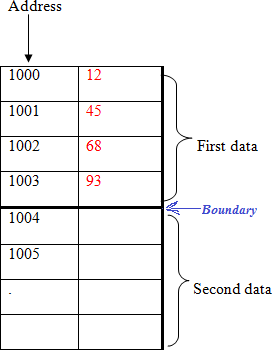
There are two ways that byte addresses can be assigned across words

* + 1. BIG-ENDIAN Assignments
    2. LITTLE –ENDIAN Assignments

### BIG-ENDIAN Assignments

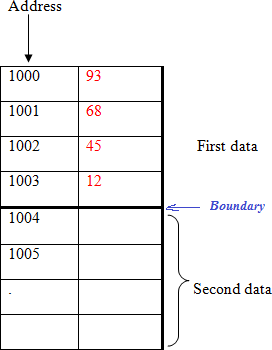
The name big-endian is used when lower byte addresses are used for the most significant bytes (the leftmost bytes) of the word.

*Example*: A 32bits word = 12456893 is represented in BIG-ENDIAN format as:



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

*Example*: A 32bits word = 12456893 is represented in LITTLE-ENDIAN format as:



### WORD ALIGNMENT

In the case of a 32-bit word length, natural word boundaries occur at addresses 0, 4, 8, …, as shown in above memory diagram. We say that the word locations have aligned addresses. in general, words are

said to be aligned in memory if they begin at a byte address that is a multiple of the number of bytes in a word. The memory of bytes in a word is a power of 2. Hence, if the word length is 16 (2 bytes), aligned words begin at byte addresses 0,2,4,…, and for a word length of 64 (23 bytes), aligned words begin at bytes addresses 0,8,16 …. There is no fundamental reason why words cannot begin at an arbitrary byte address. In that case, words are said to have unaligned addresses. While the most common case is to use aligned addresses, some computers allow the use of unaligned word addresses.

What is word alignment of a machine? What are the consecutive addresses of aligned words for 16, 32 and 64 bits word lengths of machine? Give two consecutive addresses for each case.

Generally the memory is byte addressable, **ie:** one location contains one byte of information. In general each information or word is said to be„ aligned‟, if they begin at a byte address that is a multiple of the number of bytes in a word. The consecutive addresses are as shown:

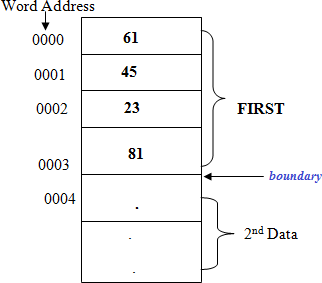
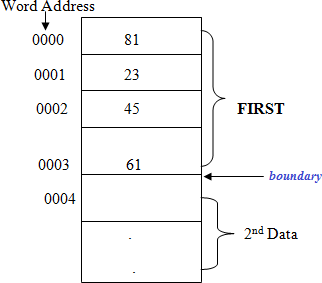
|  |  |  |
| --- | --- | --- |
| **Word length of the**  **machine** | **Consecutive word address**  **in terms of byte address** | **Example of word addresses (in decimal)**  **assuming starting address of 1000** |
| 16 bit | 0, 2, 4, 6, 8,…………… | 1000, 1002, 1004,……. |
| 32 bit | 0, 4, 8,16, ……………… | 1000, 1004, 1008,…….. |
| 64 bit | 0, 8. 16, …………… | 1000, 1008, 1016…………. |

### ACCESSING NUMBERS, CHARACTERS, AND CHARACTER STRINGS

A number usually occupies one word. It can be accessed in the memory by specifying its word address. Similarly, individual characters can be accessed by their byte address. In many applications, it is necessary to handle character strings of variable length. The beginning of the string is indicated by giving the address of the byte containing its first character. Successive byte locations contain successive characters of the string. There are two ways to indicate the length of the string. A special control character with the meaning “end of string” can be used as the last character in the string, or a separate memory word location or processor register can contain a number indicating the length of the string in bytes

Represent the number 81234561 in 32 bit Big-endian and little endian memory organization

### Big-endian Format: Little-endian Format:



**Consider a computer that has byte addressable memory organized in 32 bit words according to the big endian scheme. A program reads ASCII characters entered at keyboard and store them in successive bye location starting at location 1000. Show the contents of the memory words at location 1000 and 1004 after the name “Johnson’ has been entered. ( ASCII codes J= 4AH, o= 6FH, h=68H, n=6EH, s=73H)**

|  |  |
| --- | --- |
|  |  |
| **1000** | **4A** |
| **1001** | **6F** |
| **1002** | **68** |
| **1003** | **6E** |
| **1004** | **73** |
| **1005** | **6F** |
| **1006** | **6E** |
| **1007** | **XX** |

The content of memory location **1000 = 4A6F686E**

**1004 = 736F6EXX where XX** indicates any other data in memory

### INSTRUCTIONS AND INSTRUCTION SEQUENCING

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

Instruction operations and operands can be represented using:

* 1. Register Transfer Notation 2. Assembly language Notation

### REGISTER TRANSFER NOTATION

Transfer of information from one location in the computer to another. Possible locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem. Most of the time, we identify a location by a symbolic name standing for its hardware binary address.

***Example:*** names for the addresses of memory locations may be LOC, PLACE, A, VAR2; processor registers names may be R0, R5; and I/O register names may be DATAIN, OUTSTATUS, and so on. The

contents of a location are denoted by placing square brackets around the name of the location. Thus, the expression

R1  [ LOC ]

Means that the contents of memory location LOC are transferred into processor register R1. C = A + B can be represented in Register transfer notation as:

C  [A] + [B]

As another example, consider the operation that adds the contents of registers R1 and R2, and then places their sum into register R3. This action is indicated as

R3 [R1] + [R2]

This type of notation is known as Register Transfer Notation (RTN). Note that the right-hand side of an RTN expression always denotes a value, and the left-hand side is the name of a location where the value is to be places, overwriting the old contents of that location.

### ASSEMBLY LANGUAGE NOTATION

Another type of notation used to represent machine instructions and programs, is Assembly language notation. For this, we use an assembly language format. Here we use English like words called mnemonics to represent an operation uniquely. For example, an instruction that causes the transfer described above, from memory location LOC to processor register R1, is specified by the statement

Move LOC, R1

The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.

The second example of adding two numbers contained in processor registers R1 and R2 and placing their sum in R3 can be specified by the assembly language statement

Add R1, R2, R3

### BASIC INSTRUCTIONS

The operation of adding two numbers is a fundamental capability in any computer. The statement

C = A + B

In a high-level language program is a command to the computer to add the current values of the two variables called A and B, and to assign the sum to a third variable, C. When the program containing this statement is compiled, the three variables,

A, B, and C, are assigned to distinct locations in the memory. We will use the variable names to refer to the corresponding memory location addresses. The contents of these locations represent the values of the three variables. Hence, the above high-level language statement requires the action.

C [A] + [B]

To carry out this action, the contents of memory locations A and B are fetched from the memory and transferred into the processor where their sum is computed. This result is then sent back to the memory and stored in location C.

Explain the basic Instruction types with the help of example.

Mainly there are 4 types of instructions are available:

1. Three address Instruction
2. Two address instruction.
3. One address instruction.

### Three address Instruction:

A general instruction of this type has the format.

Operation Source1, Source2, Destination

This instruction contains two source operands and one destination operands Example: C = A + B

The above statement can be represented in three-address instruction symbolically as

Add A, B, C

Operands A and B are called the source operands, C is called the destination operand, and Add is the operation to be performed on the operands and the result is stored in C

### Two address Instruction:

Two-address instructions general format is:

Operation Source, Destination

Here each instruction has two operands, Source operand and destination operand. Example: C = A+ B

Add A, B ( sum is stored in B) which performs the operation B [A] + [B].

Move B, C which performs the operations C [B], leaving the contents of location B unchanged.

### One Address Instruction:

The general format is:

Operation Source/destination

Here the instruction specifies only memory operand which may be source or destination. The other operand is implied to be a processor register called accumulator. All operations are to be performed on the contents of accumulator

Example: C = A + B

Using only one-address instructions, the above operation C  [A] + [B] can be performed by executing the sequence of instructions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Load | A |  | **Load B** | ; copy the contents of mem location B into Accumulator |
| Add | B | **OR** | **Add A** | ; add the content of accumulator with mem content of locatn A |
| Store | C |  | **Store C** | ; will transfer the content of accumulator into mem locatn B |

Some early computers were designed around a single accumulator structure. Most modern computers have a number of general-purpose processor registers – typically 8 to 32, and even considerably more in some cases. Access to data in these registers is much faster than to data stored in memory locations because the registers are inside the processor.

Let Ri represent a general-purpose register. The instructions

Load A, Ri Store Ri, A and Add A, Ri

Are generalizations of the Load, Store, and Add instructions for the single-accumulator case, in which register Ri performs the function of the accumulator.

When a processor has several general-purpose registers, many instructions involve only operands that are in the register. In fact, in many modern processors, computations can be performed directly only on data held in processor registers. Instructions such as

Add Ri, Rj Or

Add Ri, Rj, Rk

In both of these instructions, the source operands are the contents of registers Ri and Rj. In the first instruction, Rj also serves as the destination register, whereas in the second instruction, a third register, Rk, is used as the destination.

It is often necessary to transfer data between different locations. This is achieved with the instruction Move Source, Destination

When data are moved to or from a processor register, the Move instruction can be used rather than the Load or Store instructions because the order of the source and destination operands determines which operation is intended. Thus,

Is the same as

and

is the same as

Move A, Ri

Load A, Ri

Move Ri, A

Store Ri, A

In processors where arithmetic operations are allowed only on operands that are processor registers, the C

= A + B task can be performed by the instruction sequence

Move A, Ri Move B, Rj

Add Ri, Rj Move Rj, C

In processors where one operand may be in the memory but the other must be in register, an instruction sequence for the required task would be

Move A, Ri

Add B, Ri

Move Ri, C

The speed with which a given task is carried out depends on the time it takes to transfer instructions from memory into the processor and to access the operands referenced by these instructions. Transfers that involve the memory are much slower than transfers within the processor.

### Write a program to evaluate the expression Y= AX B + C X D in a single address, Two address and Three address instruction formats.

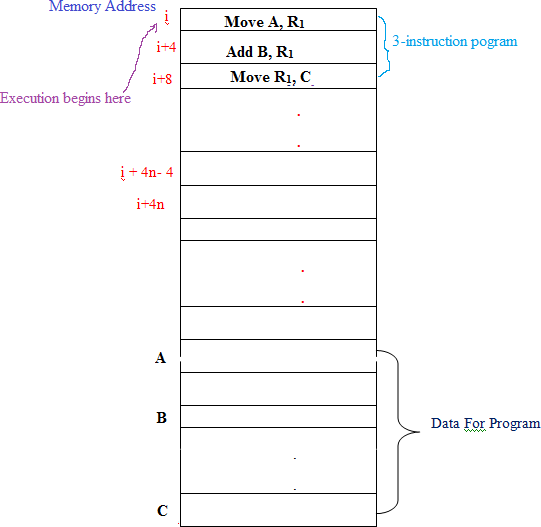
|  |  |  |
| --- | --- | --- |
| **Single Address**: | **Two Address** | **Three address** |
| LOAD A | MUL A, B | MUL A, B, X |
| MUL B | MUL C, D | MUL C, D, Z |
| STORE F | ADD B, D | ADD X, Z, Y |
| LOAD C | MOVE D, Y |  |
| MUL D |  |  |
| ADD F  STORE Y |  |  |

**Write a program to evaluate the expression S= A + BCy+ D in a single address, Two address and Three address instruction formats**

|  |  |  |
| --- | --- | --- |
| **Single Address**: | **Two Address** | **Three address** |
| LOAD B | MUL B, C | MUL B, #y, F |
| MUL #y | MUL #y, C | MUL C, F, C |
| ADD D | ADD C, D | ADD C, D, D |
| ADD A | ADD D, A | ADD D, A, S |
| STORE S | MOVE A, S |  |

### INSTRUCTION EXECUTION AND STRAIGHT-LINE SEQUENCING

In the preceding discussion of instruction formats, we used to task C [A] + [B]. The below figure shows a possible program segment for this task as it appears in the memory of a computer.



We have assumed that ***the computer allows one memory operand per instruction*** and has a number of processor registers. The three 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.

Let us consider how this program is executed. The processor contains a register called the program counter (PC), which holds the address of the instruction to be executed next. To begin executing a program, the address of its first instruction (i in our example) must be placed into the PC. Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called straight-line sequencing. During the execution of each instruction, the PC is incremented by 4 to point to the next instruction. Thus, after the Move instruction at location i + 8 is executed, the PC contains the value i + 12, which is the address of the first instruction of the next program segment.

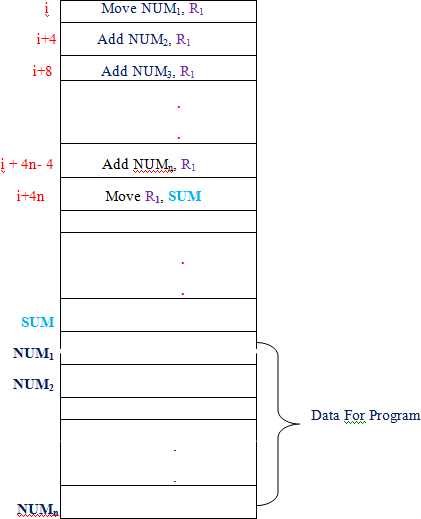
Executing a given instruction is a two-phase procedure. In the first phase, called instruction fetch, the instruction is fetched from the memory location whose address is in the PC. This instruction is placed in the instruction register (IR) in the processor. The instruction in IR is examined to determine 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.

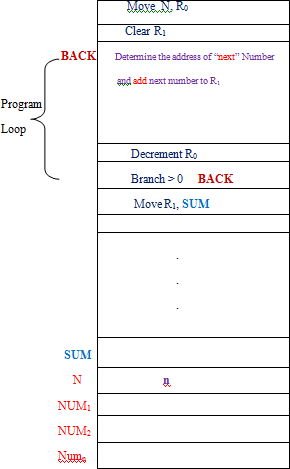
### BRANCHING

In respect of performing a set of operations on every element in an array, explain how branching helps in reducing the program size, but increases the execution time. Give an example to illustrate your answer using branching and without using branching for array handling.

Consider the task of adding a list of n numbers. The program structure shown below is a straight line program for adding n numbers. The addresses of the memory locations are symbolically given as NUM1, NUM2,…..NUMn and a separate Add instruction is used to add each number to the contents of register R1. After all the numbers have been added, the result is placed in memory location SUM.

Writing separate ADD instructions to perform addition is not only tedious but also requires more memory space to store the instructions.

It is possible to place a single ADD instruction in a program loop, as shown below.



The sequence of instructions inside the loop is executed as many times as needed. During each pass through this loop, the address of the next number in the list is determined, and that entry is fetched and added to R1.

Assume that the number of entries in the list, n, is stored in memory location N, as shown. Register R0 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 R0 at the beginning of the program and, within the body of the loop, the instruction.

### Decrement R0

reduces the contents of R0 by 1 each time through the loop. Execution of the loop is repeated as long as R0 is greater than 0.

The instruction BRANCH > 0 BACK

Is a conditional branch instruction that causes a branch to location BACK if the result of the immediately preceding instruction, which is the decremented value in register R0, 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 R1. at the end of the nth pass through the loop, the Decrement instruction produces a value of zero, and hence, branching does

not occur. When branching occurs, this 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.

### CONDITION CODES

**What are conditional code flags? Explain their use**

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

Based on the value of flags, a conditional branch instruction can be used in the program to alter the program sequence.

### GENERATING MEMORY ADDRESSES

Why do we need addressing mode?

We have a loop to be carried out „n‟ number of times, for adding „n‟ numbers. During each pass, through the loop, the address of the next number is to be computed. Individual address cannot be explicitly given in a single ADD instruction in the loop. Therefore a method for determining the address of an operand is needed. So we need an Addressing mode to specify the operands and their addresses in an instruction **ADDRESSING MODES**

Any constant value is given in the instruction by using **“#”** sign preceding the value. For example #34, #98 etc. The “$” is used to indicate hexadecimal numbers. If $ sign is present, the number is considered

as a decimal number. $16 = 16H = 22 decimal, $10 =10H = 16 decimal. Constant value cannot be specified in destination field of an instruction. A binary number identified by a prefix symbol such as a percent sign(%), as in ADD #%01011101, R1

\*\*\*\*\*\*Define addressing mode. Explain the following addressing modes with example.

1. Register addressing mode
2. Immediate addressing mode
3. Absolute (Direct) addressing mode
4. Indirect addressing mode
5. Indexed addressing mode
6. Relative addressing mode
7. Auto increment addressing mode
8. Auto decrement addressing mode

The different ways in which the operands and/ or their addresses is specified in an instruction are referred to as addressing modes.

### Register addressing Mode

The operand is the contents of a processor register; register name is specified in the instruction. Effective address **EA = Ri**

Example:

MOVE R1, R2 ADD R0, R1

### Immediate Addressing:

The operand is explicitly given in the instruction. EA = **Value** (signed integer)

Example:

MOVE **#17**, R1

ADD **#76**, R5

### Absolute (Direct) Addressing:

The operand is in a memory location. The direct (Absolute) address of this location is given in the instruction.

EA = **NUM** ( symbolic name given to the memory location of an operand) Example:

MOVE **NUM**, R1 ADD R6, **LOCA**

### Indirect Addressing:

Neither the operands nor their addresses are given explicitly. Instruction provides effective address (EA) of the operand using register or memory location. The indirect is denoted by **( )** sign around the register or memory location.

EA = [Ri] or EA = [NUM]

Example: ADD (**R1**), R0

Here the operand is in memory whose address (say **B**) is stored in register R1. So the data is read from location B and added to R0.

OR

ADD (**NUM**), R0

Here the operand is in memory whose address (say **B**) is stored in memory location NUM. So the data is read from location B and added to R0.

### Indexed Addressing mode

The effective address of the operand is generated by adding a constant value (Offset) to the contents of a index register specified in the instruction. X(Ri)

EA = [Ri] + X ;where X is a constant value, and **Ri** is a index register.

Example:

ADD 50 (R1), R2

Here the address of the operand is computed by adding the content of R1 with offset value 50. EA = [R1] +50

Two types of indexed addressing modes:

* 1. Base with index: (Ri, Rj) , EA = [Ri] +[Rj] ; Ri contains base address and offset is stored in register **Rj**
  2. Base with index and offset: X(Ri, Rj) , EA = X+ [Ri] +[Rj] ; Offset in X and base address is part of Ri and Rj

### Relative Addressing mode

Instead of using general purpose register Ri, we can use PC register to access an operand. The X(PC) denotes an EA = [PC] + X. The operand is X (constant value) location above or below the current contents of PC. This is used in conditional branch instructions.

Example: Branch > 0 BACK ; Here the target address BACK is calculated by the assembler as an

offset value X from the current value of PC.

### Auto increment addressing mode:

The effective address of the operand is the contents of pointer register specified in the instruction. After accessing the operand, the contents of the pointer register is incremented automatically to point to the next entity. EA = [Ri];

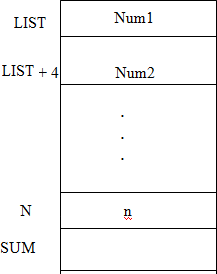
Increment Ri Example: ADD (R2)+, R0

### Auto decrement addressing mode:

The value of the pointer register specified in the instruction is decremented first and this value is used as the effective address of the operand. Decrement Ri

EA = [Ri]

Example: ADD -(R2) , R7

Write an Assembly Language Program to add „n‟ number of elements stored in an array LIST, using indirect addressing.

|  |  |  |  |
| --- | --- | --- | --- |
|  | MOVE | N, | R0 |
|  | MOVE | #LIST, | R1 |
|  | CLEAR | R2, |  |
| **BACK** | ADD | (R1), | R2 |
|  | ADD | #4, | R1 |
|  | DEC | R0 |  |
|  | BRANCH | > 0 | **BACK** |
|  | MOVE | R2, | SUM |

Write an Assembly Language Program to copy “N” number of elements stored in an array A to array B, using indirect addressing. (Assume A and B are the starting memory location of array.)

|  |  |  |  |
| --- | --- | --- | --- |
|  | MOVE | N, | R0 |
|  | MOVE | #A, | R1 |
|  | MOVE | #B, | R2 |
| **BACK** | MOVE | (R1), | R3 |
|  | MOVE | R3 | (R2) |
|  | ADD | #4, | R1 |
|  | ADD | #4, | R2 |
|  | DEC | R0 |  |
|  | BRANCH | > 0 | **BACK** |