UNIT I

BASIC STRUCTURE OF COMPUTERS

- Functional units
- Basic operational concepts
- Bus structures
- Performance and metrics
- Instructions and instruction sequencing
- Hardware
- Software interface
- Instruction set architecture
- Addressing modes
- RISC
- CISC
- ALU design
- Fixed point and floating point operations

BASIC STRUCTURE OF COMPUTERS:

Computer Organization:

It refers to the operational units and their interconnections that realize the architectural specifications.

It describes the function of and design of the various units of digital computer that store and process information.

Computer hardware:

Consists of electronic circuits, displays, magnetic and optical storage media, electromechanical equipment and communication facilities.

Computer Architecture:

- It is concerned with the structure and behaviour of the computer.
- It includes the information formats, the instruction set and techniques for addressing memory.

Functional Units

A computer consists of 5 main parts.

- Input
- Memory
- Arithmetic and logic
- Output
- Control Units

Functional units of a Computer

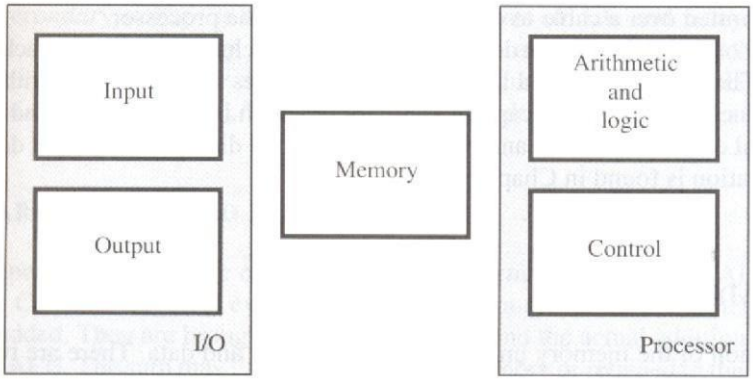


Figure 1.1 Basic functional units of a computer.

- Input unit accepts coded information from human operators, from electromechanical devices such as keyboards, or from other computers over digital communication lines.
- The information received is either stored in the computers memory for later reference or immediately used by the arithmetic and logic circuitry to perform the desired operations.
- The processing steps are determined by a program stored in the memory.
- Finally the results are sent back to the outside world through the output unit.
- All of these actions are coordinated by the control unit.
- The list of instructions that performs a task is called a program.
- Usually the program is stored in the memory.
- The processor then fetches the instruction that make up the program from the memory one after another and performs the desire operations.

Input Unit:

- Computers accept coded information through input units, which read the data.
- Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Some input devices are

- ✓ Joysticks
- ✓ Trackballs
- ✓ Mouses
- ✓ Microphones (Capture audio input and it is sampled & it is converted into digital codes for storage and processing).

Memory Unit:

It stores the programs and data.

There are 2 types of storage classes

- ✓ Primary
- ✓ Secondary

Primary Storage:

- It is a fast memory that operates at electronic speeds.
- Programs must be stored in the memory while they are being executed.
- The memory contains large no of semiconductor storage cells.
- Each cell carries 1 bit of information.
- The Cells are processed in a group of fixed size called Words.
- To provide easy access to any word in a memory,a distinct address is associated with each word location.

- Addresses are numbers that identify successive locations.
- The number of bits in each word is called the word length.
- The word length ranges from 16 to 64 bits.
- There are 3 types of memory.They are

  ✓ RAM(Random Access Memory)
  ✓ Cache memory
  ✓ Main Memory

RAM:

Memory in which any location can be reached in short and fixed amount of time after specifying its address is called RAM.

Time required to access 1 word is called Memory Access Time.

Cache Memory:

The small,fast,RAM units are called Cache. They are tightly coupled with processor to achieve high performance.

Main Memory:

The largest and the slowest unit is called the main memory.

ALU:

Most computer operations are executed in ALU. Consider a example,

Suppose 2 numbers located in memory are to be added. They are brought into the processor and the actual addition is carried out by the ALU. The sum may then be stored in the memory or retained in the processor for immediate use.

Access time to registers is faster than access time to the fastest cache unit in memory.

Output Unit:

Its function is to send the processed results to the outside world. eg.Printer

Printers are capable of printing 10000 lines per minute but its speed is comparatively slower than the processor.

Control Unit:

The operations of Input unit, output unit, ALU are co-ordinate by the control unit.

The control unit is the Nerve centre that sends control signals to other units and senses their states.

Data transfers between the processor and the memory are also controlled by the control unit through timing signals.

The operation of computers are,

- The computer accepts information in the form of programs and data through an input unit and stores it in the memory.

- Information stored in the memory is fetched, under program control into an arithmetic and logic unit, where it is processed.
- Processed information leaves the computer through an output unit.
- All activities inside the machine are directed by the control unit.

BASIC OPERATIONAL CONCEPTS:

The data/operands are stored in memory.

The individual instruction are brought from the memory to the processor, which executes the specified operation.

Eg:1

```
Add LOC A  ,R1
```

Instructions are fetched from memory and the operand at LOC A is fetched. It is then added to the contents of R0, the resulting sum is stored in Register R0.

Eg:2

```
Load LOC A, R1
```

Transfer the contents of memory location A to the register R1.

Eg:3

```
Add R1 ,R0
```

Add the contents of Register R1 & R0 and places the sum into R0.

Fig:Connection between Processor and Main Memory

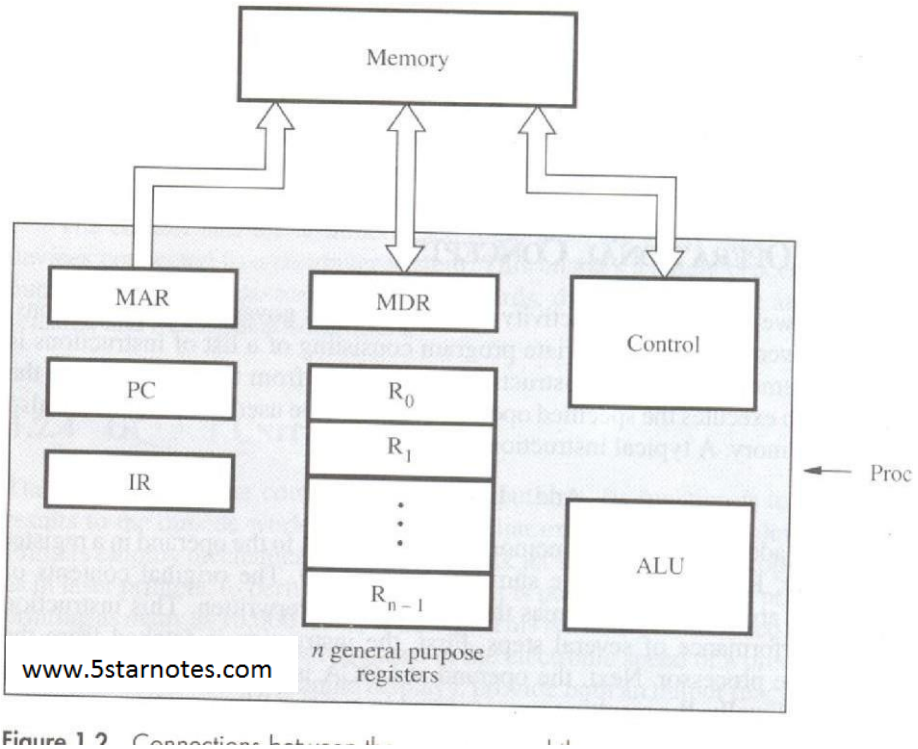


Figure 1.2 Connections between the ... unit

- Memory Address Register(MAR)
- Memory Data Register(MDR)

Instruction Register (IR):

- It holds the instruction that is currently being executed.
- It generates the timing signals.

Program Counter (PC):

It contains the memory address of the next instruction to be fetched for execution.

Memory Address Register (MAR):

It holds the address of the location to be accessed.

Memory Data Register (MDR):

- It contains the data to written into or read out of the address location.
- MAR and MDR facilitates the communication with memory.

Operation Steps:

- The program resides in memory. The execution starts when PC is point to the first instruction of the program.
- MAR read the control signal.
- The Memory loads the address word into MDR.The contents are transferred to Instruction register. The instruction is ready to be decoded & executed.

Interrupt:

- Normal execution of the program may be pre-empted if some device requires urgent servicing.
- Eg...Monitoring Device in a computer controlled industrial process may detect a dangerous condition.
- In order to deal with the situation immediately, the normal execution of the current program may be interrupted & the device raises an interrupt signal.
- The processor provides the requested service called the Interrupt Service Routine(ISR).

- ISR save the internal state of the processor in memory before servicing the interrupt because interrupt may alter the state of the processor.
- When ISR is completed, the state of the processor is restored and the interrupted program may continue its execution.

BUS STRUCTURES:

A group of lines that serves as the connection path to several devices is called a Bus. A Bus may be lines or wires or one bit per line.
The lines carry data or address or control signal.

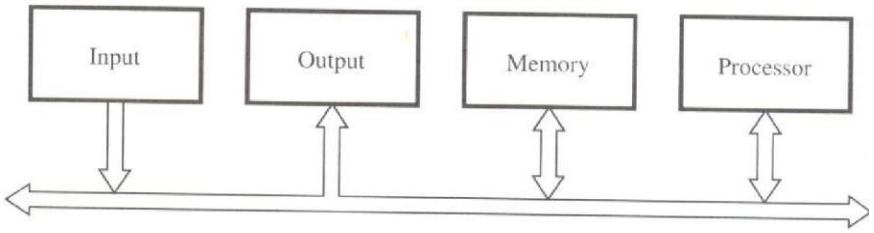


Figure 1.3 Single-bus structure.

There are 2 types of Bus structures. They are
Single Bus Structure
Multiple Bus Structure

Single Bus Structure:

It allows only one transfer at a time.
It costs low.
It is flexible for attaching peripheral devices.
Its Performance is low.

Multiple Bus Structure:

It allows two or more transfer at a time.
It costs high.
It provides concurrency in operation.
Its Performance is high.

Devices Connected with Bus	Speed
Electro-mechanical decvices (Keyboard,printer)	Slow
Magnetic / optical disk	High
Memory & processing units	Very High

The Buffer Register when connected with the bus, carries the information during transfer. The Buffer Register prevents the high speed processor from being locked to a slow I/O device during a sequence of data transfer.

SOFTWARE:

System Software is a collection of programs that are executed as needed to perform function such as,

- Receiving & Interpreting user commands.
- Entering & editing application program and storing them as files in secondary Storage devices.
- Managing the storage and retrieval of files in Secondary Storage devices.
- Running the standard application such as word processor, games, and spreadsheets with data supplied by the user.
- Controlling I/O units to receive input information and produce output results.
- Translating programs from source form prepared by the user into object form.
- Linking and running user-written application programs with existing standard library routines.

Software is of 2 types.They are

- Application program
- System program

Application Program:

It is written in high level programming language(C,C++,Java,Fortran)

The programmer using high level language need not know the details of machine program instruction.

System Program:(Compiler,Text Editor,File)

Compiler:

It translates the high level language program into the machine language program.

Text Editor:

It is used for entering & editing the application program.

System software Component ->OS(OPERATING SYSTEM)

Operating System :

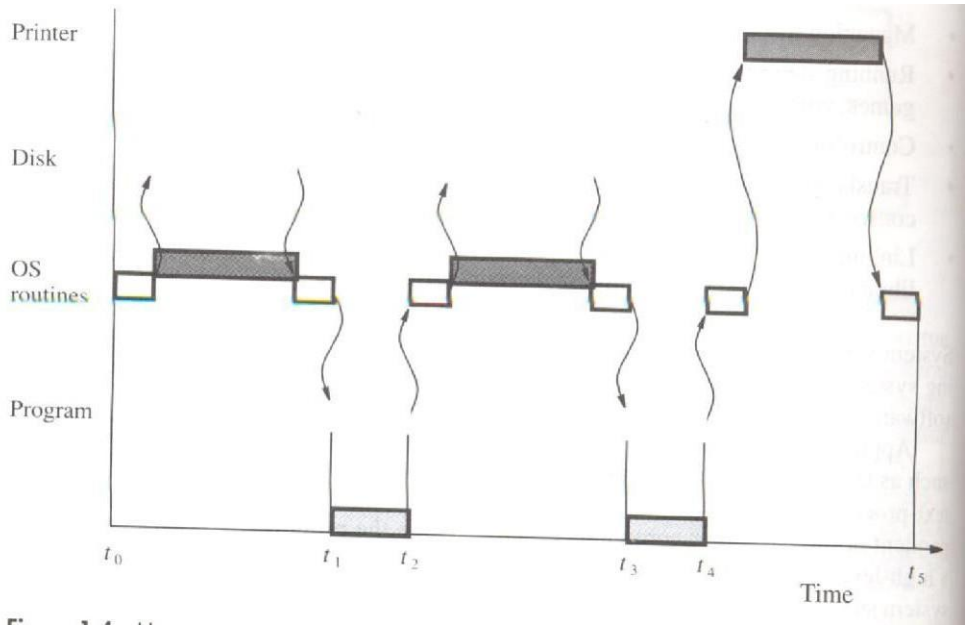
It is a large program or a collection of routines that is used to control the sharing of and interaction among various computer units.

Functions of OS:

- Assign resources to individual application program.
- Assign memory and magnetic disk space to program and data files.

- Move the data between the memory and disk units.
- Handles I/O operation.

Fig:User Program and OS routine sharing of the process



Steps:

1. The first step is to transfer the file into memory.
2. When the transfer is completed, the execution of the program starts.
3. During time period ‘t0’ to ‘t1’ , an OS routine initiates loading the application program from disk to memory, wait until the transfer is complete and then passes the execution control to the application program & print the results.
4. Similar action takes place during ‘t2’ to ‘t3’ and ‘t4’ to ‘t5’.
5. At ‘t5’, Operating System may load and execute another application program.
6. Similarly during ‘t0’ to ‘t1’ , the Operating System can arrange to print the previous program’s results while the current program is being executed.
7. The pattern of managing the concurrent execution of the several application programs to make the best possible use of computer resources is called the multi-programming or multi-tasking.

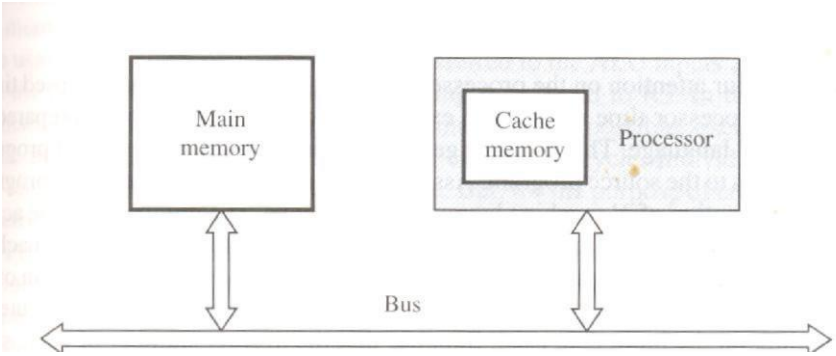
PERFORMANCE:

For best performance, it is necessary to design the compiler, machine instruction set and hardware in a co-ordinate way.

Elapsed Time→the total time required to execute the program is called the elapsed time. It depends on all the units in computer system.

Processor Time→The period in which the processor is active is called the processor time. It depends on hardware involved in the execution of the instruction.

Fig: The Processor Cache



A Program will be executed faster if the movement of instruction and data between the main memory and the processor is minimized, which is achieved by using the Cache.

Processor clock:

Clock→The Processor circuits are controlled by a timing signal called a clock.
Clock Cycle→The cycle defines a regular time interval called clock cycle.

$$\text{Clock Rate, } R = 1/P$$

Where, P→Length of one clock cycle.

Basic Performance Equation:

$$T = (N*S)/R$$

Where, T→Performance Parameter
R→Clock Rate in cycles/sec

N→Actual number of instruction execution

S→Average number of basic steps needed to execute one machine instruction.

To achieve high performance,

N,S<R

Pipelining and Superscalar operation:

Pipelining→A Substantial improvement in performance can be achieved by overlapping the execution of successive instruction using a technique called pipelining.

Superscalar Execution →It is possible to start the execution of several instruction in everey clock cycles (ie)several instruction can be executed in parallel by creating parallel paths.This mode of operation is called the Superscalar execution.

Clock Rate:

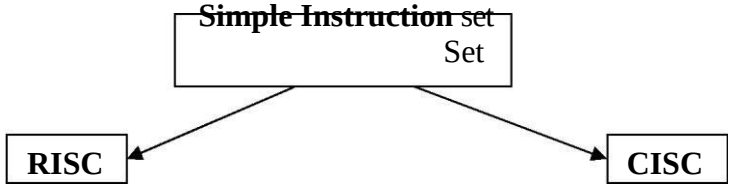
There are 2 possibilities to increase the clock rate(R).They are,

- Improving the integrated Chip(IC) technology makes logical circuits faster.
- Reduce the amount of processing done in one basic step also helps to reduce the clock period P.

Instruction Set:CISC AND RISC:

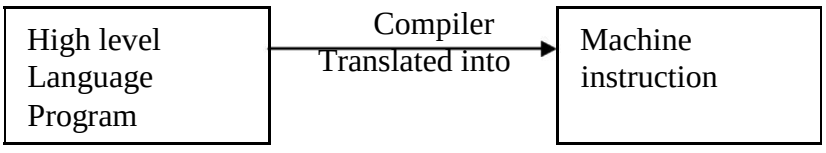
The Complex instruction combined with pipelining would achieve the best performance.

It is much easier to implement the efficient pipelining in processor with simple instruction set.



(Reduced Instruction Set Computer)
It is the design of the instruction set of a processor with simple instruction

(Complex Instruction Set Computer)
It is the design of the instruction set of a processor with complex instruction.



Functions of Compiler:

- The compiler re-arranges the program instruction to achieve better performance.
- The high quality compiler must be closely linked to the processor architecture to reduce the total number of clock cycles.

Performance Measurement:

- The Performance Measure is the time it takes a computer to execute a given benchmark.
- A non-profit organization called SPEC(System Performance Evaluation Corporation) selects and publishes representative application program.

$$\text{SPEC rating} = \frac{\text{Running time on reference computer}}{\text{Running time on computer under test}}$$

The Overall SPEC rating for the computer is given by,

$$\text{Where, SPEC rating} = \rightarrow \text{Number}\left(\prod_{i=1}^{n} \text{SPEC}_i \text{ofprograms}\right)^{1/n} \text{in the suite (SPEC)i} \rightarrow \text{rating i=1forprogram I in the suite.}$$

INSTRUCTION AND INSTRUCTION SEQUENCING

A computer must have instruction capable of performing the following operations. They are,

- Data transfer between memory and processor register.
- Arithmetic and logical operations on data.
- Program sequencing and control.
- I/O transfer.

Register Transfer Notation:

The possible locations in which transfer of information occurs are,

- Memory Location
- Processor register
- Registers in I/O sub-system.

Location	**Hardware Binary Address**	**Eg**	**Description**
Memory	LOC,PLACE,A,VAR2	R1←[LOC]	The contents of memory location are transferred to. the processor register.
Processor	R0,R1,….	[R3]←[R1]+[R2]	Add the contents of register R1 &R2 and places .their sum into register R3.It is .called Register Transfer Notation.
I/O Registers	DATAIN,DATAOUT		Provides Status information

Assembly Language Notation:

Assembly Language Format	**Description**
Move LOC,R1	Transfers the contents of memory location to the processor register R1.
Add R1,R2,R3	Add the contents of register R1 & R2 and places their sum into register R3.

Basic Instruction Types:

Instruction Type	**Syntax**	**Eg**	**Description**
Three Address	Operation Source1,Source2,Destination	Add A,B,C	Add values of variable A ,B & place the result into c.
Two Address	Operation Source,Destination	Add A,B	Add the values of A,B & place the result into B.
One Address	Operation Operand	Add B	Content of accumulator add with content of B.

Instruction Execution and Straight–line Sequencing:

Instruction Execution:

There are 2 phases for Instruction Execution. They are,

- Instruction Fetch
- Instruction Execution

Instruction Fetch:

The instruction is fetched from the memory location whose address is in PC.This is placed in IR.

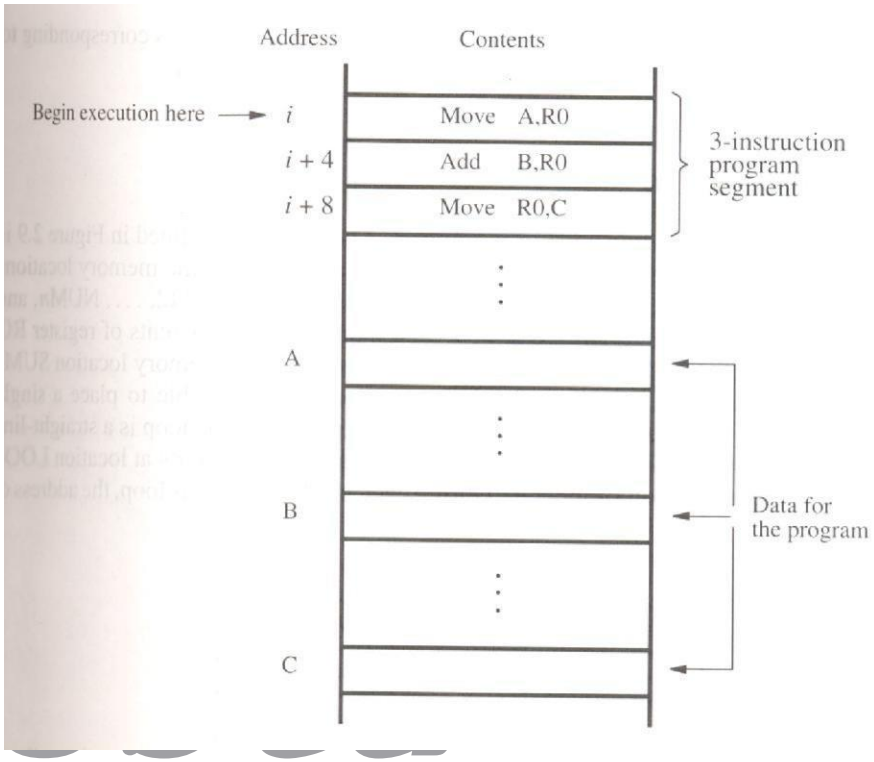
Instruction Execution:

Instruction in IR is examined to determine whose operation is to be performed.

Program execution Steps:

- To begin executing a program, the address of first instruction must be placed in PC.
- The processor control circuits use the information in the PC to fetch & execute instructions one at a time in the order of increasing order.
- This is called Straight line sequencing.During the execution of each instruction,the PC is incremented by 4 to point the address of next instruction.

Fig: Program Execution

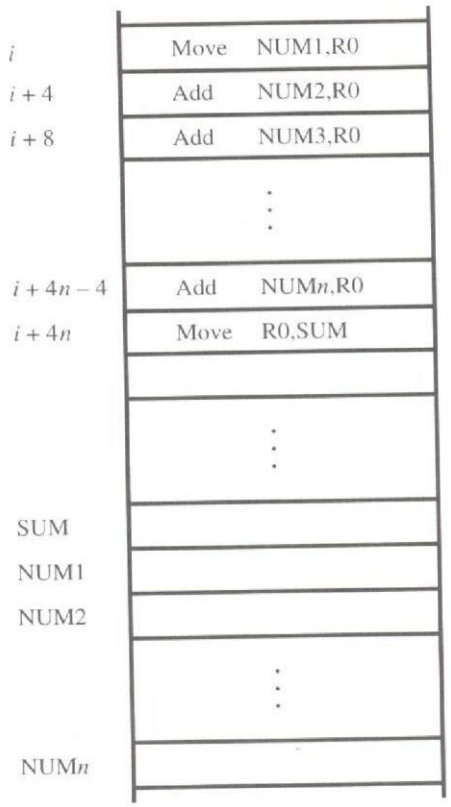


Branching:

- The Address of the memory locations containing the n numbers are symbolically given as NUM1,NUM2…..NUMn.

- Separate Add instruction is used to add each number to the contents of register R0.
- After all the numbers have been added,the result is placed in memory location SUM.

Fig:Straight Line Sequencing Program for adding ‘n’ numbers



Using loop to add ‘n’ numbers:

- Number of enteries in the list ‘n’ is stored in memory location M.Register R1 is used as a counter to determine the number of times the loop is executed.
- Content location M are loaded into register R1 at the beginning of the program.
- It starts at location Loop and ends at the instruction.Branch>0.During each pass,the address of the next list entry is determined and the entry is fetched and added to R0.

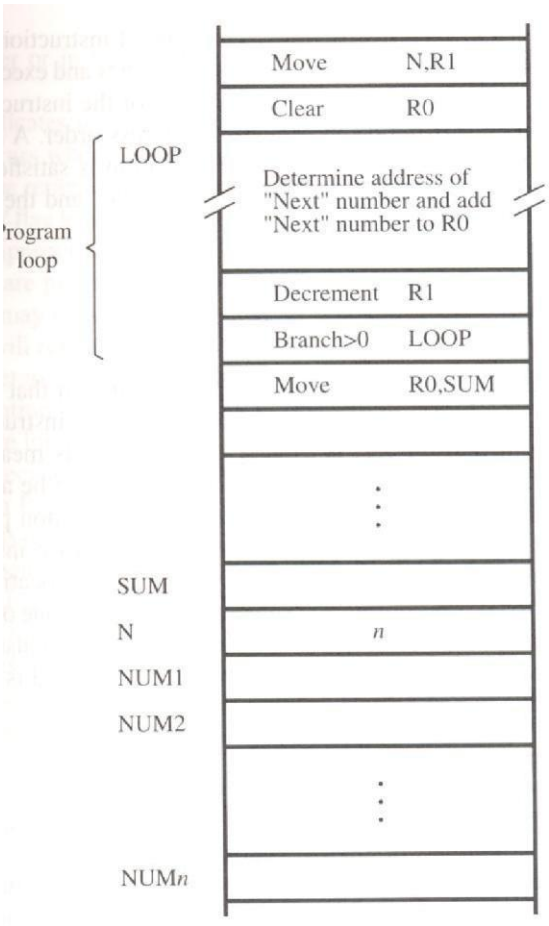
Decrement R1

- It reduces the contents of R1 by 1 each time through the loop.

Branch >0	Loop

- A conditional branch instruction causes a branch only if a specified condition is satisfied.

Fig:Using loop to add 'n' numbers:



Conditional Codes:

Result of various operation for user by subsequent conditional branch instruction is accomplished by recording the required information in individual bits often called **Condition code Flags.**

Commonly used flags:

- **N(Negative)**→set to 1 if the result is –ve ,otherwise cleared to 0.
- **Z(Zero)**→ set to 1 if the result is 0 ,otherwise cleared to 0.
- **V(Overflow)**→ set to 1 if arithmetic overflow occurs ,otherwise cleared to 0.
- **C(Carry)**set to 1 if carry and results from the operation ,otherwise cleared to 0.

ADDRESSING MODES

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

Generic Addressing Modes:

- Immediate mode
- Register mode
- Absolute mode
- Indirect mode
- Index mode
- Base with index
- Base with index and offset
- Relative mode
- Auto-increment mode
- Auto-decrement mode

Implementation of Variables and Constants:

Variables:

The value can be changed as needed using the appropriate instructions.
There are 2 accessing modes to access the variables. They are

- Register Mode
- Absolute Mode

Register Mode:

The operand is the contents of the processor register.
The name(address) of the register is given in the instruction.

Absolute Mode(Direct Mode):

- The operand is in new location.
- The address of this location is given explicitly in the instruction.

Eg: MOVE LOC,R2

The above instruction uses the register and absolute mode.
The processor register is the temporary storage where the data in the register are accessed using register mode.
The absolute mode can represent global variables in the program.

Mode	Assembler Syntax	Addressing Function
Register mode	Ri	EA=Ri
Absolute mode	LOC	EA=LOC

Where **EA**-Effective Address

Constants:

Address and data constants can be represented in assembly language using Immediate Mode.

Immediate Mode.

The operand is given explicitly in the instruction.

Eg: Move 200 immediate ,R0

It places the value 200 in the register R0.The immediate mode used to specify the value of source operand.

In assembly language, the immediate subscript is not appropriate so # symbol is used. It can be re-written as

Move #200,R0

Assembly Syntax:	**Addressing Function**
Immediate #value	Operand =value

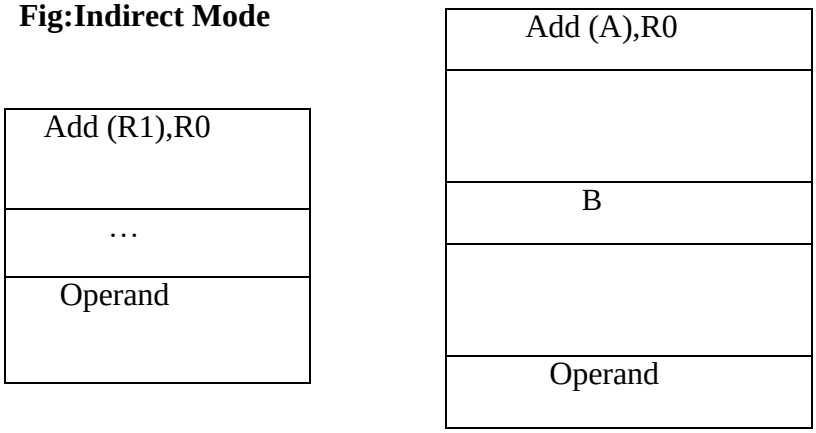
Indirection and Pointers:

Instruction does not give the operand or its address explicitly.Instead it provides information from which the new address of the operand can be determined.This address is called effective Address(EA) of the operand.

Indirect Mode:

- The effective address of the operand is the contents of a register .
- We denote the indirection by the name of the register or new address given in the instruction.

Fig:Indirect Mode



Address of an operand(B) is stored into R1 register.If we want this operand,we can get it through register R1(indirection).

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

Mode	**Assembler Syntax**	**Addressing Function**
Indirect	Ri , LOC	EA=[Ri] or EA=[LOC]

Indexing and Arrays:

Index Mode:

- The effective address of an operand is generated by adding a constant value to the contents of a register.
- The constant value uses either special purpose or general purpose register.
- We indicate the index mode symbolically as,

$$\mathbf{X(R_i)}$$

Where **X** – denotes the constant value contained in the instruction

$\mathbf{R_i}$ – It is the name of the register involved.

The Effective Address of the operand is,

$$\mathbf{EA=X + [Ri]}$$

The index register R1 contains the address of a new location and the value of X

defines an offset(also called a displacement).

To find operand,

- First go to Reg R1 (using address)-read the content from R1-1000
- Add the content 1000 with offset 20 get the result.
- 1000+20=1020
- Here the constant X refers to the new address and the contents of index register define the offset to the operand.
- The sum of two values is given explicitly in the instruction and the other is stored in register.

Eg: Add 20(R1) , R2 (or) EA=>1000+20=1020

Index Mode	**Assembler Syntax**	**Addressing Function**
Index	X(Ri)	EA=[Ri]+X
Base with Index	(Ri,Rj)	EA=[Ri]+[Rj]
Base with Index and offset	X(Ri,Rj)	EA=[Ri]+[Rj] +X

Relative Addressing:

It is same as index mode. The difference is, instead of general purpose register, here we can use program counter(PC).

Relative Mode:

- The Effective Address is determined by the Index mode using the PC in place of the general purpose register (gpr).
- This mode can be used to access the data operand. But its most common use is to specify the target address in branch instruction.Eg. Branch>0 Loop
- It causes the program execution to goto the branch target location. It is identified by the name loop if the branch condition is satisfied.

Mode	**Assembler Syntax**	**Addressing Function**
Relative	X(PC)	EA=[PC]+X

Additional Modes:

There are two additional modes. They are

- Auto-increment mode

- Auto-decrement mode

Auto-increment mode:

- The Effective Address of the operand is the contents of a register in the instruction.
- After accessing the operand, the contents of this register is automatically incremented to point to the next item in the list.

Mode	Assembler syntax	Addressing Function
Auto-increment	(Ri)+	EA=[Ri]; Increment Ri

Auto-decrement mode:

- The Effective Address of the operand is the contents of a register in the instruction.
- After accessing the operand, the contents of this register is automatically decremented to point to the next item in the list.

Mode	Assembler Syntax	Addressing Function
Auto-decrement	-(Ri)	EA=[Ri]; Decrement Ri

CISC

Pronounced *sisk,* and stands for **C**omplex **I**nstruction **S**et **C**omputer. Most PC's use CPU based on this architecture. For instance Intel and AMD CPU's are based on CISC architectures.

Typically CISC chips have a large amount of different and complex instructions. The philosophy behind it is that hardware is always faster than software, therefore one should make a powerful instruction set, which provides programmers with assembly instructions to do a lot with short programs.

In common CISC chips are relatively slow (compared to RISC chips) per instruction, but use little (less than RISC) instructions.

RISC

Pronounced *risk,* and stands for **R**educed **I**nstruction **S**et **C**omputer. RISC chips evolved around the mid-1980 as a reaction at CISC chips. The philosophy behind it is that almost no one uses complex assembly language instructions as used by CISC, and people mostly use compilers which never use complex instructions. Apple for instance uses RISC chips. Therefore fewer, simpler and faster instructions would be better, than the large, complex and slower CISC instructions. However, more instructions are needed to accomplish a task.

An other advantage of RISC is that - in theory - because of the more simple instructions,

RISC chips require fewer transistors, which makes them easier to design and cheaper

to produce.
Finally, it's easier to write powerful optimised compilers, since fewer instructions exist.

RISC vs CISC

There is still considerable controversy among experts about which architecture is better. Some say that RISC is cheaper and faster and therefor the architecture of the future. Others note that by making the hardware simpler, RISC puts a greater burden on the software. Software needs to become more complex. Software developers need to write more lines for the same tasks.

Therefore they argue that RISC is not the architecture of the future, since conventional CISC chips are becoming faster and cheaper anyway.

RISC has now existed more than 10 years and hasn't been able to kick CISC out of the market. If we forget about the embedded market and mainly look at the market for PC's, workstations and servers I guess a least 75% of the processors are based on the CISC architecture. Most of them the x86 standard (Intel, AMD, etc.), but even in the mainframe territory CISC is dominant via the IBM/390 chip. Looks like CISC is here to stay …

Is RISC than really not better? The answer isn't quite that simple. RISC and CISC architectures are becoming more and more alike. Many of today's RISC chips support just as many instructions as yesterday's CISC chips. The PowerPC 601, for example, supports *more* instructions than the Pentium. Yet the 601 is considered a RISC chip, while the Pentium is definitely CISC. Further more today's CISC chips use many techniques formerly associated with RISC chips.

ALU Design

In computing an **arithmetic logic unit** (**ALU**) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs.

Mathematician John von Neumann proposed the ALU concept in 1945, when he wrote a report on the foundations for a new computer called the EDVAC. Research into ALUs remains an important part of computer science, falling under **Arithmetic and logic structures** in the ACM Computing Classification System

Numerical systems

An ALU must process numbers using the same format as the rest of the digital circuit. The format of modern processors is almost always the two's complement binary number representation. Early computers used a wide variety of number systems, including ones' complement, Two's complement sign-magnitude format, and even true decimal systems, with ten tubes per digit.

ALUs for each one of these numeric systems had different designs, and that influenced

the current preference for two's complement, as this is the representation that makes it easier for the ALUs to calculate additions and subtractions.

The ones' complement and Two's complement number systems allow for subtraction to be accomplished by adding the negative of a number in a very simple way which negates the need for specialized circuits to do subtraction; however, calculating the negative in Two's complement requires adding a one to the low order bit and propagating the carry. An alternative way to do Two's complement subtraction of A-B is present a 1 to the carry input of the adder and use ~B rather than B as the second input.

Practical overview

Most of a processor's operations are performed by one or more ALUs. An ALU loads data from input registers, an external Control Unit then tells the ALU what operation to perform on that data, and then the ALU stores its result into an output register. Other mechanisms move data between these registers and memory.

Simple operations

A simple example arithmetic logic unit (2-bit ALU) that does AND, OR, XOR, and addition

Most ALUs can perform the following operations:

- Integer arithmetic operations (addition, subtraction, and sometimes multiplication and division, though this is more expensive)
- Bitwise logic operations (AND, NOT, OR, XOR)
- Bit-shifting operations (shifting or rotating a word by a specified number of bits to the left or right, with or without sign extension). Shifts can be interpreted as multiplications by 2 and divisions by 2.

Complex operations

Engineers can design an Arithmetic Logic Unit to calculate any operation. The more complex the operation, the more expensive the ALU is, the more space it uses in the processor, the more power it dissipates. Therefore, engineers compromise. They make the ALU powerful enough to make the processor fast, but yet not so complex as to become prohibitive. For example, computing the square root of a number might use:

1. **Calculation in a single clock** Design an extraordinarily complex ALU that calculates the square root of any number in a single step.
2. **Calculation pipeline** Design a very complex ALU that calculates the square root of any number in several steps. The intermediate results go through a series of circuits arranged like a factory production line. The ALU can accept new numbers to calculate even before having finished the previous ones. The ALU can now

produce numbers as fast as a single-clock ALU, although the results start to flow out of the ALU only after an initial delay.

3. **interactive calculation** Design a complex ALU that calculates the square root through several steps. This usually relies on control from a complex control unit with built-in microcode
4. **Co-processor** Design a simple ALU in the processor, and sell a separate specialized and costly processor that the customer can install just beside this one, and implements one of the options above.
5. **Software libraries** Tell the programmers that there is no co-processor and there is no emulation, so they will have to write their own algorithms to calculate square roots by software.
6. **Software emulation** Emulate the existence of the co-processor, that is, whenever a program attempts to perform the square root calculation, make the processor check if there is a co-processor present and use it if there is one; if there isn't one, interrupt the processing of the program and invoke the operating system to perform the square root calculation through some software algorithm.

The options above go from the fastest and most expensive one to the slowest and least expensive one. Therefore, while even the simplest computer can calculate the most complicated formula, the simplest computers will usually take a long time doing that because of the several steps for calculating the formula.

Powerful processors like the Intel Core and AMD64 implement option #1 for several simple operations, #2 for the most common complex operations and #3 for the extremely complex operations.

Inputs and outputs

The inputs to the ALU are the data to be operated on (called operands) and a code from the control unit indicating which operation to perform. Its output is the result of the computation.

In many designs the ALU also takes or generates as inputs or outputs a set of condition codes from or to a status register. These codes are used to indicate cases such as carry-in or carry-out, overflow, divide-by-zero, etc.

ALUs vs. FPUs

A Floating Point Unit also performs arithmetic operations between two values, but they do so for numbers in floating point representation, which is much more complicated than the two's complement representation used in a typical ALU. In order to do these calculations, a FPU has several complex circuits built-in, including some internal ALUs.

In modern practice, engineers typically refer to the ALU as the circuit that performs integer arithmetic operations (like two's complement and BCD). Circuits that calculate

more complex formats like floating point, complex numbers, etc. usually receive a more specific name such as FPU.

FIXED POINT NUMBER AND OPERATION

In computing, a **fixed-point number** representation is a real data type for a number that has a fixed number of digits after (and sometimes also before) the radix point (*e.g.*, after the decimal point '.' in English decimal notation). Fixed-point number representation can be compared to the more complicated (and more computationally demanding) floating point number representation.

Fixed-point numbers are useful for representing fractional values, usually in base 2 or base 10, when the executing processor has no floating point unit (FPU) or if fixed-point provides improved performance or accuracy for the application at hand. Most low-cost embedded microprocessors and microcontrollers do not have an FPU.

Representation

A value of a fixed-point data type is essentially an integer that is scaled by a specific factor determined by the type. For example, the value 1.23 can be represented as 1230 in a fixed-point data type with scaling factor of 1/1000, and the value 1230000 can be represented as 1230 with a scaling factor of 1000. Unlike floating-point data types, the scaling factor is the same for all values of the same type, and does not change during the entire computation.

The scaling factor is usually a power of 10 (for human convenience) or a power of 2 (for computational efficiency). However, other scaling factors may be used occasionally, e.g. a time value in hours may be represented as a fixed-point type with a scale factor of 1/3600 to obtain values with one-second accuracy.

The maximum value of a fixed-point type is simply the largest value that can be represented in the underlying integer type, multiplied by the scaling factor; and similarly for the minimum value. For example, consider a fixed-point type represented as a binary integer with b bits in two's complement format, with a scaling factor of $1/2^f$ (that is, the last f bits are fraction bits): the minimum representable value is $-2^{b-1}/2^f$ and the maximum value is $(2^{b-1}-1)/2^f$.

Operations

To convert a number from a fixed point type with scaling factor R to another type with scaling factor S, the underlying integer must be multiplied by R and divided by S; that is, multiplied by the ratio R/S. Thus, for example, to convert the value 1.23 = 123/100 from a type with scaling factor R=1/100 to one with scaling factor S=1/1000, the underlying integer 123 must be multiplied by (1/100)/(1/1000) = 10, yielding the representation 1230/1000. If S does not divide R (in particular, if the new scaling factor R is less than the

original *S*), the new integer will have to be rounded. The rounding rules and methods are usually part of the language's specification.

To add or subtract two values the same fixed-point type, it is sufficient to add or subtract the underlying integers, and keep their common scaling factor. The result can be exactly represented in the same type, as long as no overflow occurs (i.e. provided that the sum of the two integers fits in the underlying integer type.) If the numbers have different fixed-point types, with different scaling factors, then one of them must be converted to the other before the sum.

To multiply two fixed-point numbers, it suffices to multiply the two underlying integers, and assume that the scaling factor of the result is the product of their scaling factors. This operation involves no rounding. For example, multiplying the numbers 123 scaled by 1/1000 (0.123) and 25 scaled by 1/10 (2.5) yields the integer 123×25 = 3075 scaled by (1/1000)×(1/10) = 1/10000, that is 3075/10000 = 0.3075. If the two operands belong to the same fixed-point type, and the result is also to be represented in that type, then the product of the two integers must be explicitly multiplied by the common scaling factor; in this case the result may have to be rounded, and overflow may occur. For example, if the common scaling factor is 1/100, multiplying 1.23 by 0.25 entails multiplying 123 by 25 to yield 3075 with an intermediate scaling factor if 1/10000. This then must be multiplied by 1/100 to yield either 31 (0.31) or 30 (0.30), depending on the rounding method used, to result in a final scale factor of 1/100.

To divide two fixed-point numbers, one takes the integer quotient of their underlying integers, and assumes that the scaling factor is the quotient of their scaling factors. The first division involves rounding in general. For example, division of 3456 scaled by 1/100 (34.56) by 1234 scaled by 1/1000 (1.234) yields the integer 3456÷1234 = 3 (rounded) with scale factor (1/100)/(1/1000) = 10, that is, 30. One can obtain a more accurate result by first converting the dividend to a more precise type: in the same example, converting 3456 scaled by 1/100 (34.56) to 3456000 scaled by 1/100000, before dividing by 1234 scaled by 1/1000 (1.234), would yield 3456000÷1234 = 2801 (rounded) with scaling factor (1/100000)/(1/1000) = 1/100, that is 28.01 (instead of 290). If both operands and the desired result are represented in the same fixed-point type, then the quotient of the two integers must be explicitly divided by the common scaling factor.

Precision loss and overflow

Because fixed point operations can produce results that have more bits than the operands there is possibility for information loss. For instance, the result of fixed point multiplication could potentially have as many bits as the sum of the number of bits in the two operands. In order to fit the result into the same number of bits as the operands, the answer must be rounded or truncated. If this is the case, the choice of which bits to keep is very important. When multiplying two fixed point numbers with the same format, for instance with I integer bits, and Q fractional bits, the answer could have up to $2I$ integer bits, and $2Q$ fractional bits.

For simplicity, fixed-point multiply procedures use the same result format as the operands. This has the effect of keeping the middle bits; the *I*-number of least significant integer bits, and the *Q*-number of most significant fractional bits. Fractional bits lost below this value represent a precision loss which is common in fractional multiplication. If any integer bits are lost, however, the value will be radically inaccurate.

Some operations, like divide, often have built-in result limiting so that any positive overflow results in the largest possible number that can be represented by the current format. Likewise, negative overflow results in the largest negative number represented by the current format. This built in limiting is often referred to as *saturation*.

Some processors support a hardware overflow flag that can generate an exception on the occurrence of an overflow, but it is usually too late to salvage the proper result at this point.

FLOATING POINT NUMBERS & OPERATIONS

Floating point Representation:

To represent the fractional binary numbers, it is necessary to consider binary point.If binary point is assumed to the right of the sign bit ,we can represent the fractional binary numbers as given below,

$$B= (b_0 * 2^{0} + b_{-1} * 2^{-1} + b_{-2} * 2^{-2} + \ldots + b_{-(n-1)} * 2^{-(n-1)})$$

With this fractional number system,we can represent the fractional numbers in the following range,

$$-1 < F < 1 - 2^{-(n-1)}$$

- The binary point is said to be float and the numbers are called floating point numbers.

- The position of binary point in floating point numbers is variable and hence numbers must be represented in the specific manner is referred to as floating point representation.

- The floating point representation has three fields.They are,
  - Sign
  - Significant digits and
  - Exponent.

Eg: 111101.1000110→$1.111101100110 * 2^5$

Where ,

2^5 →Exponent and scaling factor