MODULE 1: BASIC STRUCTURE OF COMPUTERS

1.1 COMPUTER TYPES

- **Computer Architecture (CA)** is concerned with the structure and behaviour of the computer.
- CA includes the information formats, the instruction set and techniques for addressing memory.
- In general covers, CA covers 3 aspects of computer-design namely: 1) Computer Hardware, 2) Instruction set Architecture and 3) Computer Organization.

1. Computer Hardware

➢ It consists of electronic circuits, displays, magnetic and optical storage media and communication facilities.

2. Instruction Set Architecture

➢ It is programmer visible machine interface such as instruction set, registers, memory organization and exception handling.

➢ Two main approaches are 1) CISC and 2) RISC.

(CISC□Complex Instruction Set Computer,
RISC□Reduced Instruction Set Computer)

3. Computer Organization

➢ It includes the high level aspects of a design, such as

→ memory-system
→ bus-structure &
→ design of the internal CPU.

➢ 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.

1.2 FUNCTIONAL UNITS

- A computer consists of 5 functionally independent main parts:

1) Input
2) Memory
3) ALU
4) Output &
5) Control units.

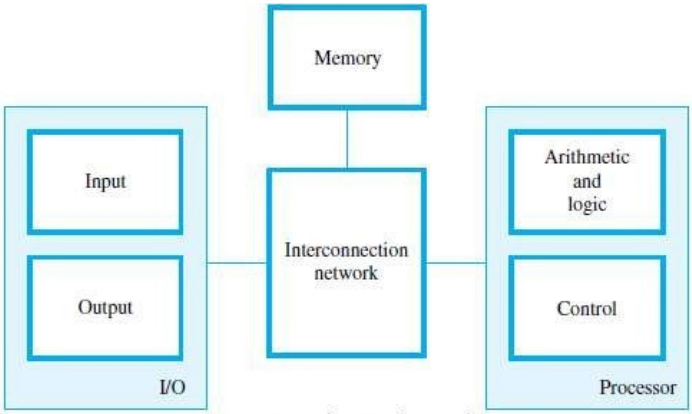


Figure 1.1 Basic functional units of a computer.

1.3 BASIC OPERATIONAL CONCEPTS

- An Instruction consists of 2 parts, 1) Operation code (Opcode) and 2) Operands.

- The data/operands are stored in memory.
- The individual instruction are brought from the memory to the processor.
- Then, the processor performs the specified operation.
- Let us see a typical instruction

ADD LOCA, R0

- This instruction is an addition operation. The following are the steps to execute the instruction: Step 1: Fetch the instruction from main-memory into the processor.

Step 2: Fetch the operand at location LOCA from main-memory into the processor.

Step 3: Add the memory operand (i.e. fetched contents of LOCA) to the contents of register R0. Step 4: Store the result (sum) in R0.

- The same instruction can be realized using 2 instructions as:

Load LOCA, R1 Add R1, R0

- The following are the steps to execute the instruction:

Step 1: Fetch the instruction from main-memory into the processor.

Step 2: Fetch the operand at location LOCA from main-memory into the register R1.

Step 3: Add the content of Register R1 and the contents of register R0.

Step 4: Store the result (sum) in R0.

MAIN PARTS OF PROCESSOR

- The processor contains ALU, control-circuitry and many registers.
- The processor contains „n" general-purpose registers R0 through Rn-1.
- The IR holds the instruction that is currently being executed.
- The control-unit generates the timing-signals that determine when a given action is to take place.
- The PC contains the memory-address of the next-instruction to be fetched & executed.
- During the execution of an instruction, the contents of PC are updated to point to next instruction.
- The MAR holds the address of the memory-location to be accessed.
- The MDR contains the data to be written into or read out of the addressed location.
- MAR and MDR facilitates the communication with memory. (IR Instruction-Register, PC Program Counter)

(MAR Memory Address Register, MDR Memory Data Register)

STEPS TO EXECUTE AN INSTRUCTION

1) The address of first instruction (to be executed) gets loaded into PC.

2) The contents of PC (i.e. address) are transferred to the MAR & control-unit issues Read signal to memory.

3) After certain amount of elapsed time, the first instruction is read out of memory and placed into MDR.

4) Next, the contents of MDR are transferred to IR. At this point, the instruction can be decoded & executed.

5) To fetch an operand, it's address is placed into MAR & control-unit issues Read signal. As a result, the operand is transferred from memory into MDR, and then it is transferred from MDR to ALU.

6) Likewise required number of operands is fetched into processor.

7) Finally, ALU performs the desired operation.

8) If the result of this operation is to be stored in the memory, then the result is sent to the MDR.

9) The address of the location where the result is to be stored is sent to the MAR and a Write cycle is initiated.

10) At some point during execution, contents of PC are incremented to point to next instruction in the program

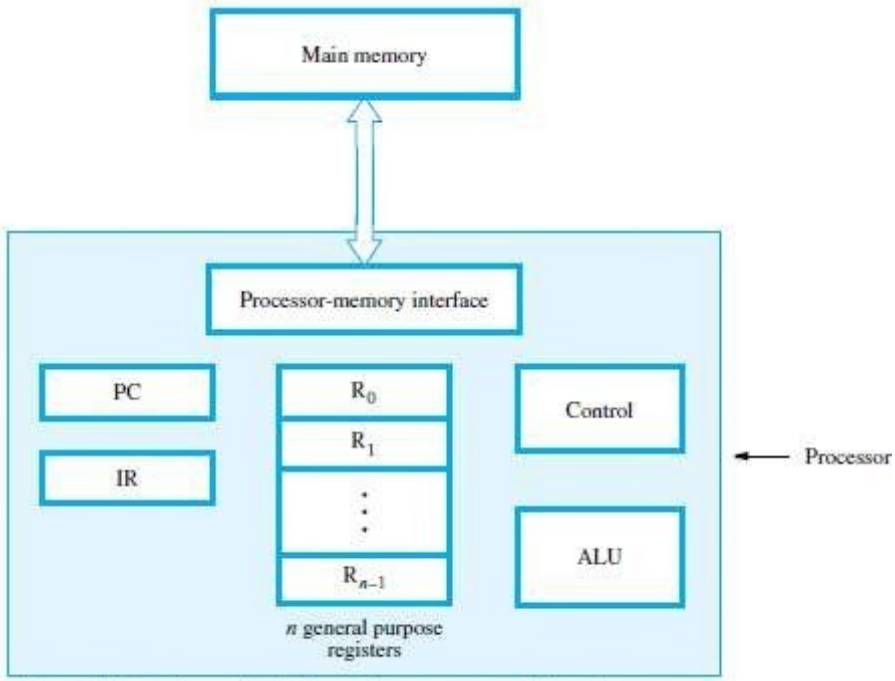


Figure 1.2 Connection between the processor and the main memory.

1.4 BUS STRUCTURE

- A bus is a group of lines that serves as a connecting path for several devices.
- A bus may be lines or wires.
- The lines carry data or address or control signal.
- There are 2 types of Bus structures: 1) Single Bus Structure and 2) Multiple Bus Structure.

Single Bus Structure

- Because the bus can be used for only one transfer at a time, only 2 units can actively use the bus at any given time.
- Bus control lines are used to arbitrate multiple requests for use of the bus.
- **Advantages:**
    1) Low cost &
    2) Flexibility for attaching peripheral devices.

2) Multiple Bus Structure

- Systems that contain multiple buses achieve more concurrency in operations.
- Two or more transfers can be carried out at the same time.
- **Advantage:** Better performance.
- **Disadvantage:** Increased cost.

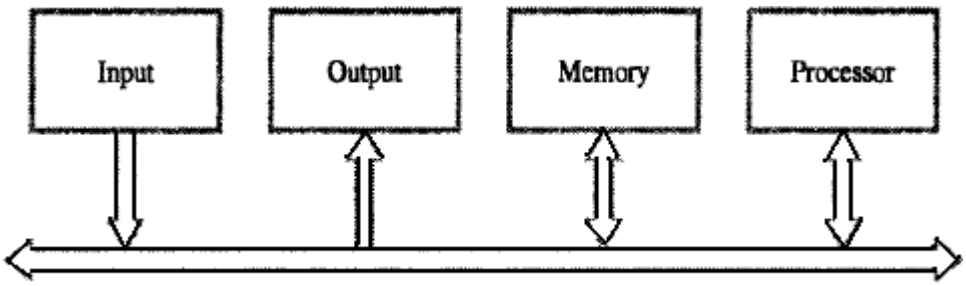


Figure 1.3 Single-bus structure.

- The devices connected to a bus vary widely in their speed of operation.
- To synchronize their operational-speed, buffer-registers can be used.
- **Buffer Registers**

→ are included with the devices to hold the information during transfers.

→ prevent a high-speed processor from being locked to a slow I/O device during data transfers.

1.5 PERFORMANCE

- The most important measure of performance of a computer is how quickly it can execute programs.
- The speed of a computer is affected by the design of

    1) Instruction-set.
    2) Hardware & the technology in which the hardware is implemented.
    3) Software including the operating system.

- Because programs are usually written in a HLL, performance is also affected by the compiler that translates programs into machine language. (HLL High Level Language).
- For best performance, it is necessary to design the compiler, machine instruction set and hardware in a co-ordinated way.

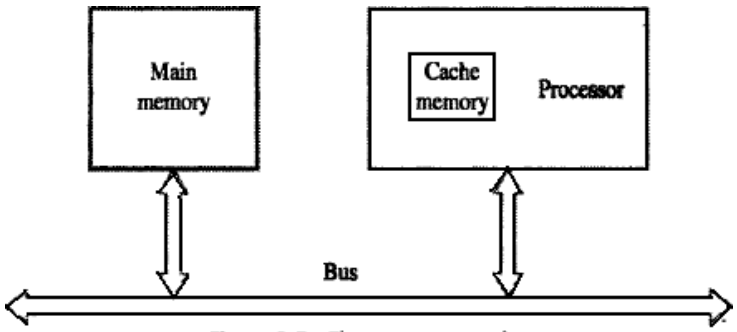


Figure 1.5 The processor cache.

examine the flow of program instructions and data between the memory & the processor.

- At the start of execution, all program instructions are stored in the main-memory.
- As execution proceeds, instructions are fetched into the processor, and a copy is placed in the cache.
- Later, if the same instruction is needed a second time, it is read directly from the cache.
- A program will be executed faster

if movement of instruction/data between the main-memory and the processor is minimized which is achieved by using the cache.

1.5.1 PROCESSOR CLOCK

- Processor circuits are controlled by a timing signal called a **Clock**.
- The clock defines 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 such that each step can be completed in one clock cycle.
- Let P = Length of one clock cycle R = Clock rate.
- Relation between P and R is given by

$$R = \frac{1}{P}$$

- R is measured in cycles per second.
- Cycles per second is also called Hertz (Hz)

1.5.2 BASIC PERFORMANCE EQUATION

- Let T = Processor time required to executed a program. N = Actual number of instruction executions.

S = Average number of basic steps needed to execute one machine instruction. R = Clock rate in cycles per second.

- The program execution time is given by

$$T = \frac{N \times S}{R} \qquad \text{------(1)}$$

- Equ1 is referred to as the basic performance equation.

- To achieve high performance, the computer designer must reduce the value of T, which means CLOCK RATE
- There are 2 possibilities for increasing the clock rate R:

1) Improving the IC technology makes logic-circuits faster.

This reduces the time needed to compute a basic step. (IC □ integrated circuits).

This allows the clock period P to be reduced and the clock rate R to be increased.

2) Reducing the amount of processing done in one basic step also reduces the clock period P.

- In presence of a cache, the percentage of accesses to the main-memory is small.

Hence, much of performance-gain expected from the use of faster technology can be realized. The value of T will be reduced by same factor as R is increased „." S & N are not affected.

1.5.3 PERFORMANCE MEASUREMENT

- Benchmark refers to standard task used to measure how well a processor operates.
- The Performance Measure is the time taken by a computer to execute a given benchmark.
- SPEC selects & publishes the standard programs along with their test results for different application domains. (SPEC System Performance Evaluation Corporation).

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

- SPEC Rating is given by
- SPEC rating = 50 The computer under test is 50 times as fast as reference-computer.
- The test is repeated for all the programs in the SPEC suite. Then, the geometric mean of the results is computed.
- Let SPECi = Rating for program „i' in the suite.

Overall SPEC rating for the computer is given by

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i\right)^{\frac{1}{n}}$$

where n = no. of programs in the suite.

INSTRUCTION SET: CISC AND RISC

RISC	CISC
Simple instructions taking one cycle.	Complex instructions taking multiple cycle.
Instructions are executed by hardwired control unit.	Instructions are executed by microprogrammed control unit.
Few instructions.	Many instructions.
Fixed format instructions.	Variable format instructions.
Few addressing modes, and most instructions have register to register addressing mode.	Many addressing modes.
Multiple register set.	Single register set.
Highly pipelined.	No pipelined or less pipelined.

Problem 1:

List the steps needed to execute the machine instruction:

Load R2, LOC

in terms of transfers between the components of processor and some simple control commands. Assume that the address of the memory-location containing this instruction is initially in register PC. **Solution:**

1. Transfer the contents of register PC to register MAR.
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 and wait until MDR is loaded.
6. Transfer contents of MDR to the ALU.
7. Transfer contents of R0 to the ALU.
8. Perform addition of the two operands in the ALU and transfer result into R0.
9. Transfer contents of PC to ALU.

10. Add 1 to operand in ALU and transfer incremented address to PC.

Problem 2:

List the steps needed to execute the machine instruction:

Add R4, R2, R3

in terms of transfers between the components of processor and some simple control commands. Assume that the address of the memory-location containing this instruction is initially in register PC. **Solution:**

1. Transfer the contents of register PC to register MAR.
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 contents of R1 and R2 to the ALU.
5. Perform addition of two operands in the ALU and transfer answer into R3.
6. Transfer contents of PC to ALU.
7. Add 1 to operand in ALU and transfer incremented address to PC.

Problem 3:

(a) Give a short sequence of machine instructions for the task "Add the contents of memory-location A to those of location B, and place the answer in location C". Instructions:

Load R*i*, LOC and

Store R*i*, LOC

are the only instructions available to transfer data between memory and the general purpose registers. Add instructions are described in Section 1.3. Do not change contents of either location A or B.

(b) Suppose that Move and Add instructions are available with the formats:

Move Location1, Location2 and

Add Location1, Location2

These instructions move or add a copy of the operand at the second location to the first location, overwriting the original operand at the first location. Either or both of the operands can be in the memory or the general-purpose registers. Is it possible to use fewer instructions of these types to accomplish the task in part (a)? If yes, give the sequence.

Solution:

(a)

Load A, R0 Load B, R1 Add R0, R1

Store R1, C

(b) Yes;

Move B, C Add A, C

Problem 4:

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

Solution:

N = 1000

25% of N= 250 instructions require 4 clock cycles.

40% of N =400 instructions require 5 clock cycles. 35% of N=350 instructions require 3 clock cycles.

T = (N*S)/R= (250*4+400*5+350*3)/1X109 =(1000+2000+1050)/1*109= 4.05 μs.

Problem 5:

For the following processor, obtain the performance.

Clock rate = 800 MHz

No. of instructions executed = 1000

Average no of steps needed / machine instruction = 20

$T = \frac{N \times S}{R}$ = (1000*20)/800 * 10^6=25 micro sec or 25*10^{-6} sec

Solution:

Problem 6:

(a) Program execution time T is to be examined for a certain high-level language program. The program can be run on a RISC or a CISC computer. Both computers use pipelined instruction execution, but pipelining in the RISC machine is more effective than in the CISC machine. Specifically, the effective value of S in the T expression for the RISC machine is 1.2, bit it is only 1.5 for the CISC machine. Both machines have the same clock rate R. What is the largest allowable value for N, the number of instructions executed on the CISC machine, expressed as a percentage of the N value for the RISC machine, if time for execution on the CISC machine is to be longer than on the RISC machine?

(b) Repeat Part (a) if the clock rate R for the RISC machine is 15 percent higher than that for the CISC machine.

Solution:

(a) Let TR = (NR X SR)/RR & TC = (NC X SC)/RC be execution times on RISC and CISC processors. Equating execution times and clock rates, we have

1.2NR = 1.5NC

Then

NC/NR = 1.2/1.5 = 0.8

Therefore, the largest allowable value for NC is 80% of NR.

(b) In this case,

1.2NR/1.15 = 1.5NC/1.00

Then

NC/NR =1.2/(1.15 X 1.5) = 0.696

Therefore, the largest allowable value for NC is 69.6% of NR.

Problem 7:

(a) Suppose that execution time for a program is proportional to instruction fetch time. Assume that fetching an instruction from the cache takes 1 time unit, but fetching it from the main-memory takes 10 time units. Also, assume that a requested instruction is found in the cache with probability 0.96. Finally, assume that if an instruction is not found in the cache it must first be fetched from the main- memory into the cache and then fetched from the cache to be executed. Compute the ratio of program execution time without the cache to program execution time with the cache. This ratio is called the speedup resulting from the presence of the cache.

(b) If the size of the cache is doubled, assume that the probability of not finding a requested instruction there is cut in half. Repeat part (a) for a doubled cache size.

Solution:

(a) Let cache access time be 1 and main-memory access time be 20. Every instruction that is executed must be fetched from the cache, and an additional fetch from the main-memory must be performed for 4% of these cache accesses.

Therefore,

(b) (b)

$$\text{Speedup factor} = \frac{1.0 \times 20}{(1.0 \times 1) + (0.04 \times 20)} = 11.1$$

$$\text{Speedup factor} = \frac{1.0 \times 20}{(1.0 \times 1) + (0.02 \times 20)} = 16.7$$

MODULE 1 (CONT.): MACHINE INSTRUCTIONS & PROGRAMS

1.6 1.6 NUMBERS, ARITHMETIC OPERATIONS AND CHARACTERS NUMBER REPRESENTATION

- Numbers can be represented in 3 formats:

1) Sign and magnitude

2) 1's complement

3) 2's complement

- In all three formats, MSB=0 for +ve numbers & MSB=1 for -ve numbers.
- In **sign-and-magnitude system**,

negative value is obtained by changing the MSB from 0 to 1 of the corresponding positive value.

For ex, +5 is represented by 0101 &

-5 is represented by 1101.

- In **1's complement system**,

negative values are obtained by complementing each bit of the corresponding positive number.

For ex, -5 is obtained by complementing each bit in 0101 to yield 1010.

(In other words, the operation of forming the 1's complement of a given number is equivalent to subtracting that number from 2n-1).

- In **2's complement system**,

forming the 2's complement of a number is done by subtracting that number from 2n.

For ex, -5 is obtained by complementing each bit in 0101 & then adding 1 to yield 1011. (In other words, the 2's complement of a number is obtained by adding 1 to the 1's complement of that number).

B	Values represented		
$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

Figure 1.3 Binary, signed-integer representations.

- 2's complement system yields the most efficient way to carry out addition/subtraction operations.

ADDITION OF POSITIVE NUMBERS

- Consider adding two 1-bit numbers.

The sum of 1 & 1 requires the 2-bit vector 10 to represent the value 2. We say that sum is 0

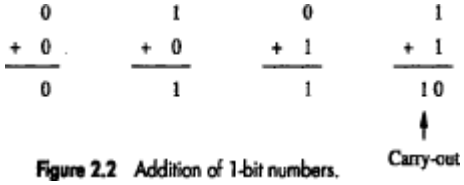


Figure 2.2 Addition of 1-bit numbers.

and the carry-

- out is 1.

ADDITION & SUBTRACTION OF SIGNED NUMBERS

- Following are the two rules for addition and subtraction of n-bit signed numbers using the 2's complement representation system (Figure 1.6).

Rule 1:

- **To Add** two numbers, add their n-bits and ignore the carry-out signal from the MSB position.
- Result will be algebraically correct, if it lies in the range -2n-1 to +2n-1-1.

Rule 2:

- **To Subtract** two numbers X and Y (that is to perform X-Y), take the 2's complement of Y and then add it to X as in rule 1.
- Result will be algebraically correct, if it lies in the range (2n-1) to +(2n-1-1).

- When the result of an arithmetic operation is outside the representable-range, an arithmetic overflow is said to occur.
- To represent a signed in 2's complement form using a larger number of bits, repeat the sign bit as many times as needed to the left. This operation is called **sign extension**.
- In 1's complement representation, the result obtained after an addition operation is not always correct. The carry-out(cn) cannot be ignored. If cn=0, the result obtained is correct. If cn=1, then a 1 must be added to the result to make it correct.

OVERFLOW IN INTEGER ARITHMETIC

- When result of an arithmetic operation is outside the representable-range, an **arithmetic overflow**

is said to occur.

- For example: If we add two numbers +7 and +4, then the output sum S is 1011(□0111+0100), which is the code for -5, an incorrect result.
- An overflow occurs in following 2 cases

1) Overflow can occur only when adding two numbers that have the same sign.

The carry-out signal from the sign-bit position is not a sufficient indicator of overflow when

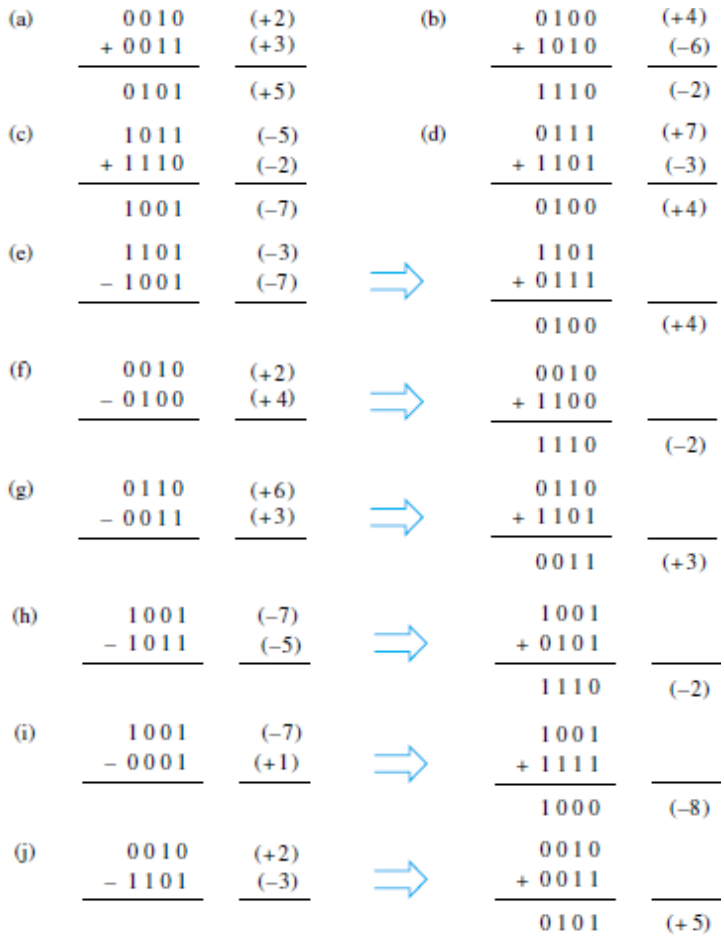


Figure 1.6 2's-complement Add and Subtract operations.

2) adding signed numbers.

1.7 FLOATING-POINT NUMBERS & OPERATIONS

IEEE STANDARD FOR FLOATING POINT NUMBERS

- Single precision representation occupies a single 32-bit word.

The scale factor has a range of 2-126 to 2+127 (which is approximately equal to 10+38).

- The 32 bit word is divided into 3 fields: sign(1 bit), exponent(8 bits) and mantissa(23 bits).
- Signed exponent=E.

Unsigned exponent E'=E+127. Thus, E' is in the range 0<E'<255.

- The last 23 bits represent the mantissa. Since binary normalization is used, the MSB of the mantissa is always equal to 1. (M represents fractional-part).
- The 24-bit mantissa provides a precision equivalent to about 7 decimal-digits (Figure 9.24).
- Double precision representation occupies a single 64-bit word. And E' is in the range 1<E'<2046.
- The 53-bit mantissa provides a precision equivalent to about 16 decimal-digits.

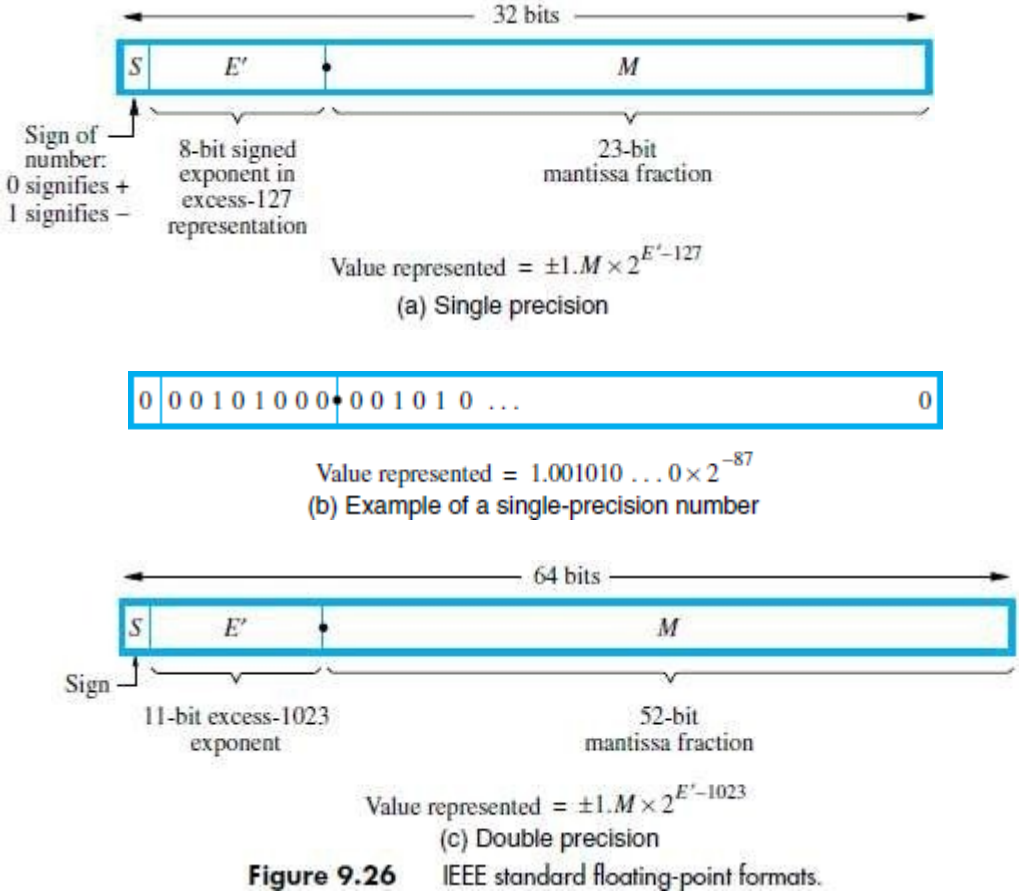


Figure 9.26 IEEE standard floating-point formats.

NORMALIZATION

- When the decimal point is placed to the right of the first(non zero) significant digit, the number is said to be normalized.

- If a number is not normalized, it can always be put in normalized form by shifting the fraction and adjusting the exponent. As computations proceed, a number that does not fall in the representable range of normal numbers might be generated.
- In single precision, it requires an exponent less than -126 (underflow) or greater than +127 (overflow). Both are exceptions that need to be considered.

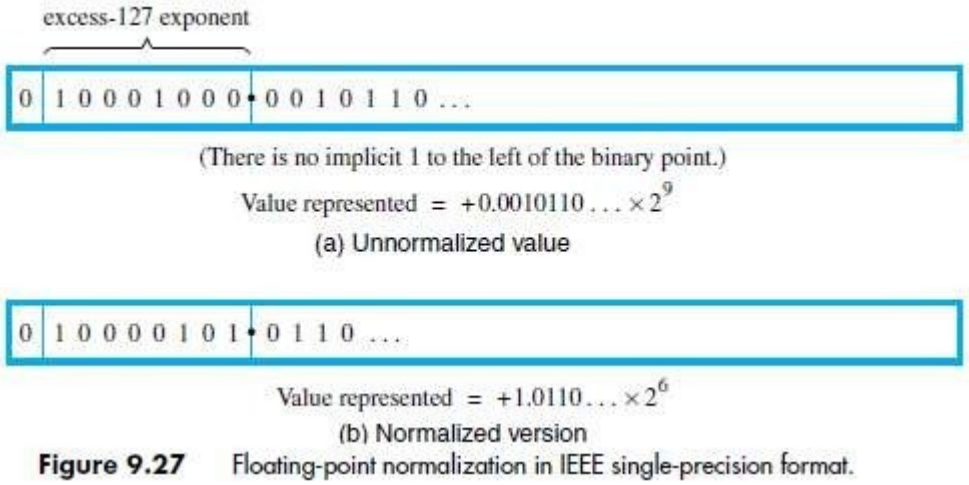


Figure 9.27 Floating-point normalization in IEEE single-precision format.

SPECIAL VALUES

- The end values 0 and 255 of the excess-127 exponent E' are used to represent special values.
- When E'=0 and the mantissa fraction m is zero, the value exact 0 is represented.
- When E'=255 and M=0, the value ∞ is represented, where ∞ is the result of dividing a normal number by zero.
- when E'=0 and M!=-, denormal numbers are represented. Their value is $\pm 0.M2^{-126}$
- When E'=255 and M!=0, the value represented is called not a number(NaN). A NaN is the result of performing an invalied operation such as 0/0 or $\sqrt{0}$.

1.8 MEMORY-LOCATIONS & ADDRESSES

- **Memory** consists of many millions of storage cells (flip-flops).
- Each cell can store a bit of information i.e. 0 or 1 (Figure 2.1).
- Each group of n bits is referred to as a **word** of information, and n is called the **word length**.

- The word length can vary from 8 to 64 bits.
- A unit of 8 bits is called a **byte**.
- Accessing the memory to store or retrieve a single item of information (word/byte) requires distinct addresses for each item location. (It is customary to use numbers from 0 through 2k-1 as the addresses of successive-locations in the memory).
- If 2k = no. of addressable locations;

then 2k addresses constitute the address-space of the computer.

For example, a 24-bit address generates an address-space of 224 locations (16 MB).

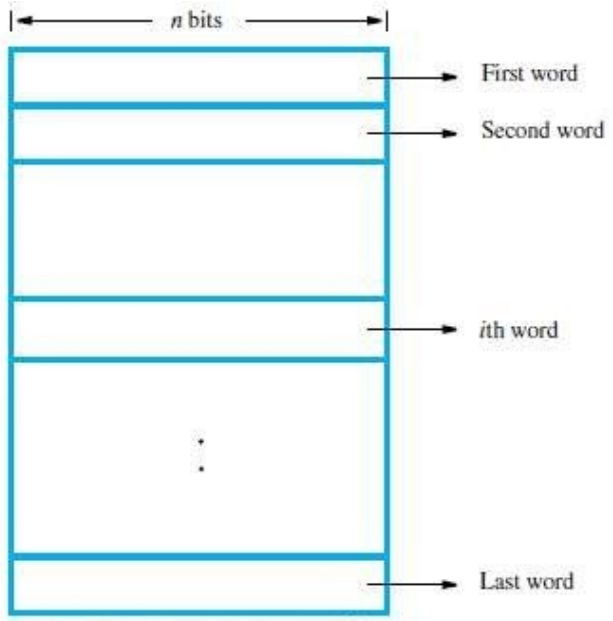


Figure 2.1 Memory words.

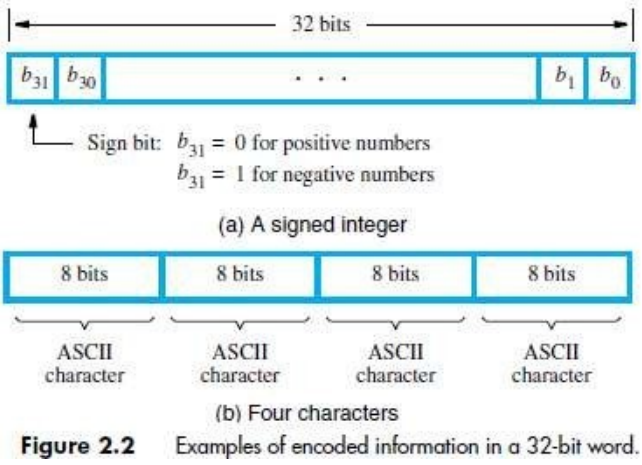


Figure 2.2 Examples of encoded information in a 32-bit word.

1.8.1 BYTE-ADDRESSABILITY

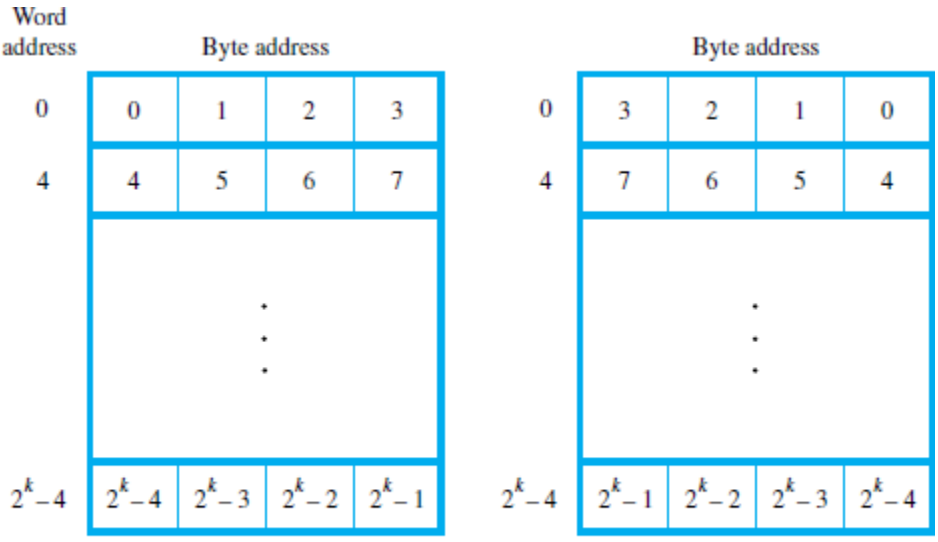
- In byte-addressable memory, successive addresses refer to successive byte locations in the memory.
- Byte locations have addresses 0, 1, 2. . . . .
- If the word-length is 32 bits, successive words are located at addresses 0, 4, 8. . with each word having 4 bytes.

BIG-ENDIAN & LITTLE-ENDIAN ASSIGNMENTS

- There are two ways in which byte-addresses are arranged (Figure 2.3).

    1) **Big-Endian:** Lower byte-addresses are used for the more significant bytes of the word.
    2) **Little-Endian:** Lower byte-addresses are used for the less significant bytes of the word

- In both cases, byte-addresses 0, 4, 8. are taken as the addresses of successive words in the memory.



(a) Big-endian assignment (b) Little-endian assignment

Figure 2.3 Byte and word addressing.

Consider a 32-bit integer (in hex): 0x12345678 which consists of 4 bytes: 12, 34, 56, and 78.

- Hence this integer will occupy 4 bytes in memory.
- Assume, we store it at memory address starting 1000.
- On little-endian, memory will look like

Address	Value
1000	78
1001	56
1002	34
1003	12

- On big-endian, memory will look like

Address	Value
1000	12
1001	34
1002	56
1003	78

WORD ALIGNMENT

- 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.
- For example,
  - If the word length is 16(2 bytes), aligned words begin at byte-addresses 0, 2, 4 . . . . .
  - If the word length is 64(2 bytes), aligned words begin at byte-addresses 0, 8, 16 . . . . .

- Words are said to have **Unaligned Addresses**, if they begin at an arbitrary byte-address.

ACCESSING NUMBERS, CHARACTERS & CHARACTERS 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.
- There are two ways to indicate the length of the string:
    1) A special control character with the meaning "end of string" can be used as the last character in the string.
    2) A separate memory word location or register can contain a number indicating the length of the string in bytes.

1.9 MEMORY OPERATIONS

- Two memory operations are:
    1) Load (Read/Fetch) &
    2) Store (Write).
- The **Load** operation transfers a copy of the contents of a specific memory-location to the processor. The memory contents remain unchanged.
- Steps for Load operation:
    1) Processor sends the address of the desired location to the memory.
    2) Processor issues „read“ signal to memory to fetch the data.
    3) Memory reads the data stored at that address.
    4) Memory sends the read data to the processor.
- The **Store** operation transfers the information from the register to the specified memory-location. This will destroy the original contents of that memory-location.
- Steps for Store operation are:
    1) Processor sends the address of the memory-location where it wants to store data.
    2) Processor issues „write“ signal to memory to store the data.
    3) Content of register(MDR) is written into the specified memory-location.

1.10 INSTRUCTIONS & INSTRUCTION SEQUENCING

- A computer must have instructions capable of performing 4 types of operations:

1) Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).
2) Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).
3) Program sequencing and control (CALL.RET, LOOP, INT).
4) I/0 transfers (IN, OUT).

REGISTER TRANSFER NOTATION (RTN)

- The possible locations in which transfer of information occurs are: 1) Memory-location 2) Processor register & 3) Registers in I/O device.

Location	Hardware Binary Address	Example	Description
Memory	LOC, PLACE, NUM	R1 □ [LOC]	Contents of memory-location LOC are transferred into register R1.
Processor	R0, R1 ,R2	[R3] □ [R1]+[R2]	Add the contents of register R1 &R2 and places their sum into R3.
I/O Registers	DATAIN, DATAOUT	R1 □ DATAIN	Contents of I/O register DATAIN are transferred into register R1.

ASSEMBLY LANGUAGE NOTATION

- To represent machine instructions and programs, assembly language format is used.

Assembly Language Format	Description

Move LOC, R1	Transfer data from memory-location LOC to register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.
Add R1, R2, R3	Add the contents of registers R1 and R2, and places their sum into register R3.

BASIC INSTRUCTION TYPES

Instruction Type	**Syntax**	**Example**	**Description**	**Instructions for Operation C<-[A]+[B]**
Three Address	Opcode Source1,Source2,Destination	Add A,B,C	Add the contents of memory-locations A & B. Then, place the result into location C.	
Two Address	Opcode Source, Destination	Add A,B	Add the contents of memory-locations A & B. Then, place the result into location B, replacing the original contents of this location. Operand B is both a source and a destination.	Move B, C Add A, C
One Address	Opcode Source/Destination	Load A	Copy contents of memory-location A into accumulator.	Load A Add B Store C
		Add B	Add contents of memory-location B to contents of accumulator register & place sum back into accumulator.	
		Store C	Copy the contents of the accumulator into location C.	

Zero Address	Opcode [no Source/Destination]	Push	Locations of all operands are defined implicitly. The operands are stored in a pushdown stack.	Not possible

- Access to data in the registers is much faster than to data stored in memory-locations.
- Let Ri represent a general-purpose register. The instructions: *Load A,Ri*

Store Ri,A 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.

- In processors, where arithmetic operations as allowed only on operands that are in registers, the task C<-[A]+[B] can be performed by the instruction sequence:

Move A,Ri
Move B,Rj
Add Ri,Rj
Move Rj,C

1.10.1 INSTRUCTION EXECUTION & STRAIGHT LINE SEQUENCING

- The program is executed as follows:

1) Initially, the address of the first instruction is loaded into PC (Figure 2.8).

2) 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***.**

3) During the execution of each instruction, PC is incremented by 4 to point to next instruction.

- There are 2 phases for Instruction Execution:

1) Fetch Phase: The instruction is fetched from the memory-location and placed in the IR.

2) Execute Phase: The contents of IR is examined to determine which operation is to be performed. The specified-operation is then performed by the processor.

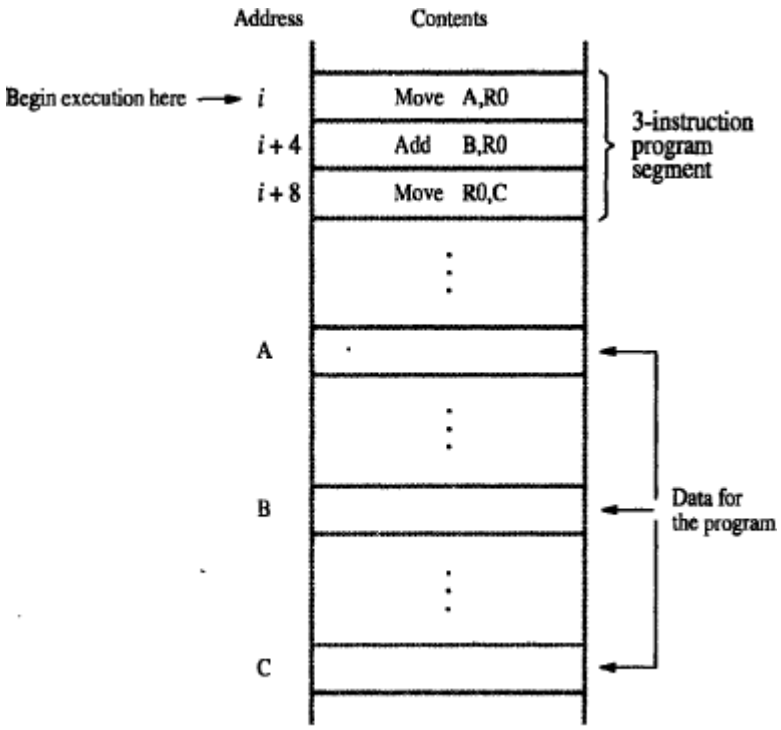


Figure 2.8 A program for C ← [A] + [B].

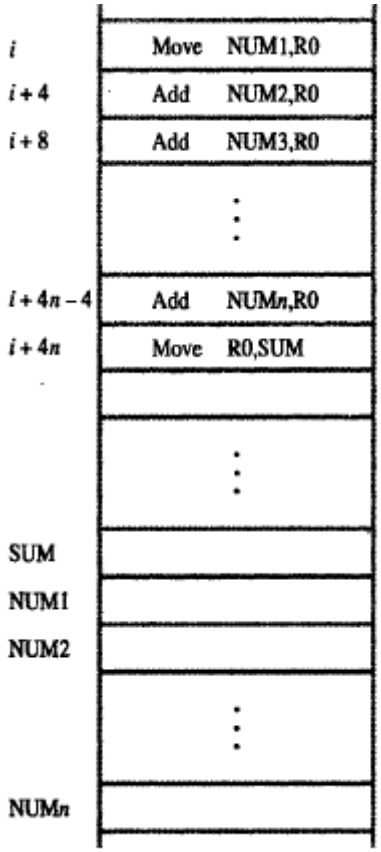


Figure 2.9 A straight-line program for adding *n* numbers.

Program Explanation

- Consider the program for adding a list of n numbers (Figure 2.9).
- 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.

BRANCHING

- Consider the task of adding a list of „n“ numbers (Figure 2.10).
- Number of entries in the list „n“ is stored in memory-location **N**.
- Register **R1** is used as a counter to determine the number of times the loop is executed.

- Content-location N is loaded into register R1 at the beginning of the program.
- The **Loop** is a straight line sequence of instructions executed as many times as needed. The loop starts at location LOOP and ends at the instruction Branch>0.
- During each pass,

→ address of the next list entry is determined and

→ that entry is fetched and added to R0.

- The instruction *Decrement R1* reduces the contents of R1 by 1 each time through the loop.
- Then **Branch 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**.
- 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.

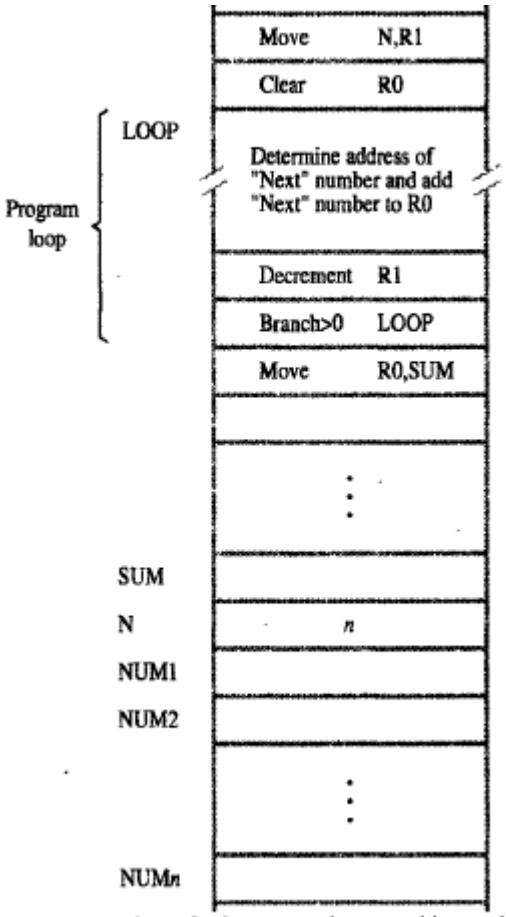


Figure 2.10 Using a loop to add *n* numbers.

1.10.2 CONDITION CODES

- The processor keeps track of information about the results of various operations. This is accomplished by recording the required information in individual bits, called **Condition Code Flags**.
- These flags are grouped together in a special processor-register called the condition code register (or statue register).
- Four commonly used flags are:

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