Computer Organization Architecture

What is Computer Architecture and Organization?

Architecture describes what the computer does.

The organization describes how it does it.

- The computer architecture is concerned with the structure and behavior of various functional modules of a computer and how they interact to provide the processing needs of user. It can be considered as a catalogue of tools or attributes that are visible to the user such as instruction sets, number of bits used for data, addressing techniques, etc.
- Computer Organization is concerned with the way the hardware components are connected together to form a computer system. It defines the way system is structured so that all those catalogued tools can be used. The significant components of Computer organization are ALU, CPU, memory and memory organization. Computer Organization deals with a structural relationship.

Syllabus

Sr. No.	Content	Total Hrs
1	Computer Data Representation Basic computer data types, Complements, Fixed point representation, Register Transfer and Micro-operations: Floating point representation, Register Transfer language, Register Transfer, Bus and Memory Transfers (Tree-State Bus Buffers, Memory Transfer), Arithmetic Micro-Operations, Logic Micro-Operations, Shift Micro-Operations, Arithmetic logical shift unit	4
2	Basic Computer Organization and Design Instruction codes, Computer registers, computer instructions, Timing and Control, Instruction cycle, Memory-Reference Instructions, Input-output and interrupt, Complete computer description, Design of Basic computer, Design of Accumulator Unit.	4
3	Assembly Language Programming Introduction, Machine Language, Assembly Language Programming: Arithmetic and logic operations, looping constructs, Subroutines, I-O Programming.	8
4	Micro programmed Control Organization: Control Memory, Address sequencing, Micro program example, Design of Control Unit	4
5	Central Processing Unit Introduction, General Register Organization, Stack Organization, Instruction format, Addressing Modes, Data transfer and manipulation, Program control, Reduced Instruction Set Computer (RISC) & Complex Instruction Set Computer (CISC)	5

6	Pipeline And Vector Processing Flynn's taxonomy, Parallel Processing, Pipelining, Arithmetic Pipeline, Instruction, Pipeline, RISC Pipeline, Vector Processing, Array Processors	5
7	Computer Arithmetic Introduction, Addition and subtraction, Multiplication Algorithms (Booth Multiplication Algorithm), Division Algorithms, Floating Point Arithmetic operations, Decimal Arithmetic Unit.	4
8	Input-Output Organization Input-Output Interface, Asynchronous Data Transfer, Modes Of Transfer, Priority Interrupt, DMA, Input-Output Processor (IOP), CPUIOP Communication, Serial communication.	4
9	Memory Organization Memory Hierarchy, Main Memory, Auxiliary Memory, Associative Memory, Cache Memory, Virtual Memory.	6
10	Multiprocessors Characteristics of Multiprocessors, Interconnection Structures, Inter-processor Arbitration, Inter-processor Communication and Synchronization, Cache Coherence, Shared Memory Multiprocessors.	4

Reference Books

1. M. Morris Mano, “Computer System Architecture”, Pearson Education
2. Yale N. Patt, Sanjay J. Patel, “Introduction to Computing Systems” McGraw Hill.
3. Hamacher, Vranesic, Zaky, “Computer Organization”, McGraw Hill.
4. Andrew S. Tanenbaum and Todd Austin, “Structured Computer Organization”, Pearson Education
5. N. D. Jotwani, “Computer system organization”, McGraw Hill
6. R.S.Gaonkar, “Microprocessor Architecture, Programming and Applications with 8085A”, Penram International 7. Douglas Hall, Microprocessors and Interfacing, TMH.

Course Objective

- The main objective of the computer organization is to understand the various computer hardware components and the interaction between these components. In computer organization and architecture , the computer system can be classified into number of functional units.

Course Outcomes

CO-1 Identify and explain the basic structure and functional units of a digital computer.

CO-2 Write assembly language programs and identify the role and working of various functional units of a computer for executing an instructions.

CO-3 Design processing unit using the concepts of ALU and control logic design.

CO-4 Design circuits for interfacing memory and I/O with processor.

CO-5 Comprehend the features and performance parameters of different types of computer architectures

Applications

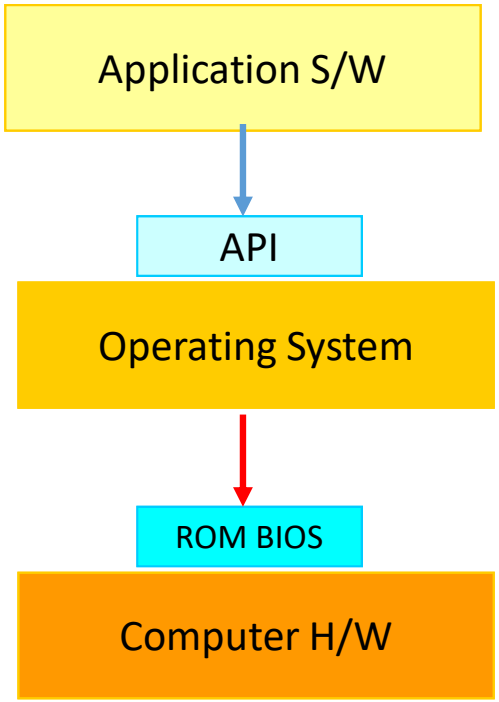
- Computer architecture involves the broad infrastructure of modern PCs. All modern computers, mobile devices and similar technology rely on this architectural knowledge.
- Computer architects help companies maintain functioning hardware, software and networks. They may work in a variety of industries with other computer technology professionals.
- Earlier, computer architects designed computer architecture on paper. It was then directly built into a final hardware form.

Unit I-Computer Data Representation

- Objectives
  - How the various data types found in digital computers are represented in binary form in computer registers. Emphasis is on the representation of numbers employed in arithmetic operations and on the binary coding of symbols used in data processing.
  - To introduce a register transfer language and shows how it is used to express micro-operations in symbolic form. Symbols are defined for arithmetic, logic and shift micro-operations. A composite arithmetic unit is developed to show the hardware design of most common micro-operations.

Digital Computers

- Computer = H/W + S/W
- Program(S/W)
  - A sequence of instruction
  - S/W = Program + Data
    - The data that are manipulated by the program constitute the data base
  - Application S/W
    - DB, word processor, Spread Sheet
  - System S/W
    - OS, Firmware, Compiler, Device Driver



- Computer Hardware
  - CPU
  - Memory
    - Program Memory(ROM)
    - Data Memory(RAM)
  - I/O Device
    - Interface: 8251 SIO, 8255 PIO, 6845 CRTC, 8272 FDC, 8237 DMAC, 8279 KDI
    - Input Device: Keyboard, Mouse, Scanner
    - Output Device: Printer, Plotter, Display
    - Storage Device(I/O): FDD, HDD, MOD
      - HDD is also known as Hard disk, Hard drive, or Fixed disk. HDD is a data storage device, which uses magnetic storage to store digital data and fetch it using rotating platters.
      - **floppy disk drive**, also called **FDD** or **FD** for short, is a computer disk drive that enables a user to save data to removable diskettes
      - MOD-Magneto-Optical device

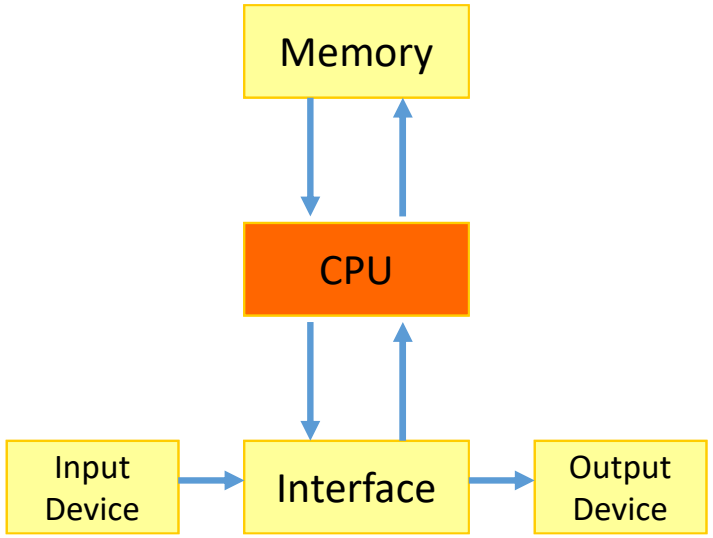


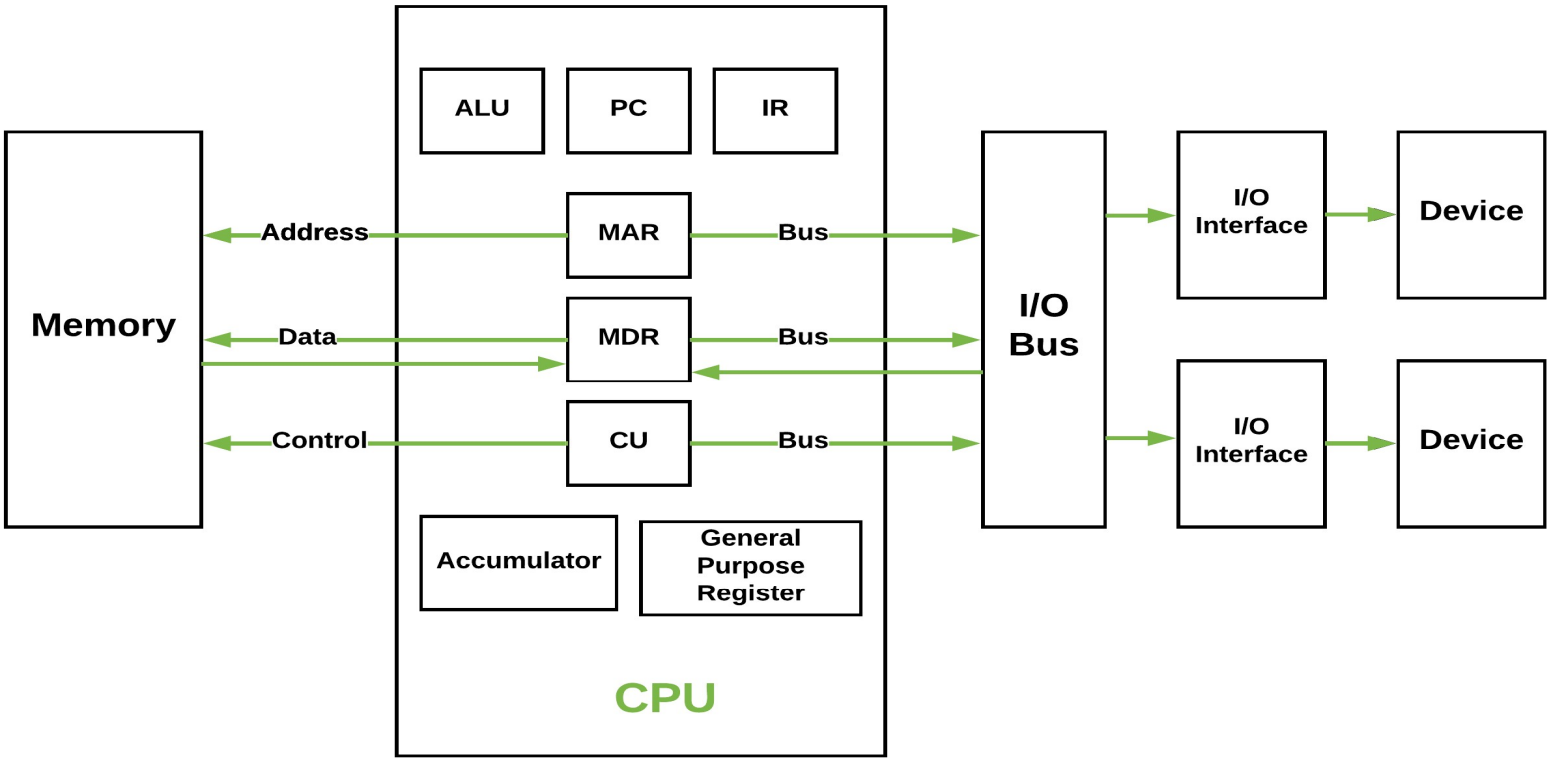
Figure 1-1 Block Diagram of a digital Computer

- 3 different point of view(Computer Hardware)
  - Computer Organization
    - H/W components operation/connection
  - Computer Design
    - H/W Design/Implementation
  - Computer Architecture
    - Structure and behavior of the computer as seen by the user
    - Information format, Instruction set, memory addressing, CPU, I/O, Memory
- ISA(Instruction Set Architecture)
  - the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls, the logic design, and the physical implementation. Instructions, Addressing modes, Instruction and data formats, Register

What is Computer Organization?

- Computer Organization is realization of what is specified by the computer architecture.
- It deals with how operational attributes are linked together to meet the requirements specified by computer architecture.
- Some organizational attributes are hardware details, control signals, peripherals.
- The computer organization is concerned with the structure and behavior of digital computers.
- The main objective of this subject to understand the overall basic computer hardware structure, including the peripheral devices.

Basic CPU structure



Basic Terminology

- **Control Unit:-** A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches the code for instructions and controlling how data moves around the system.
- **Arithmetic and Logic Unit (ALU) –:** The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic Operation.

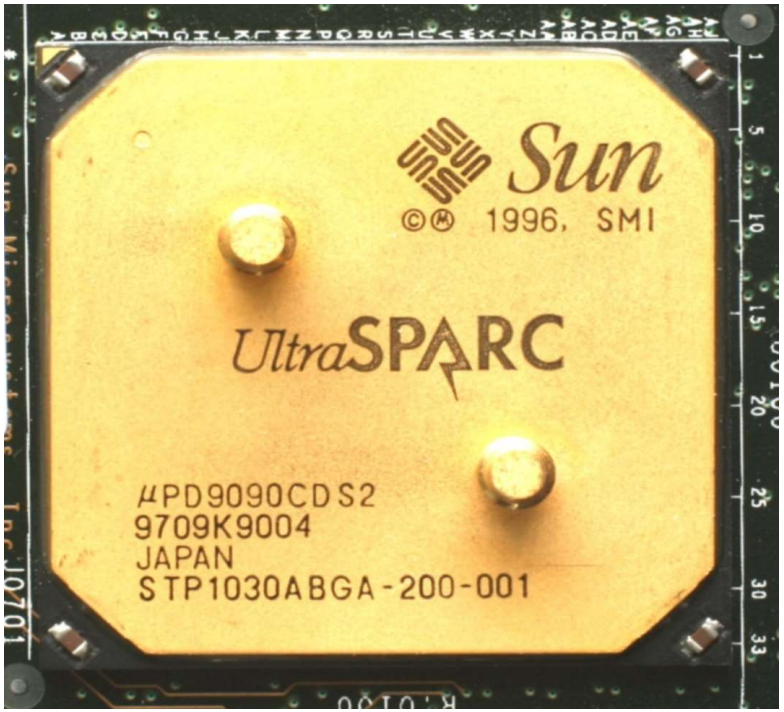
Main Memory Unit (Registers) –

- **Accumulator:** Stores the results of calculations made by ALU.
- **Program Counter (PC):** Keeps track of the memory location of the next instructions to be dealt with. The PC then passes this next address to Memory Address Register (MAR).
- **Memory Address Register (MAR):** It stores the memory locations of instructions that need to be fetched from memory or stored into memory.
- **Memory Data Register (MDR):** It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.
- **Current Instruction Register (CIR):** It stores the most recently fetched instructions while it is waiting to be coded and executed.
- **Instruction Buffer Register (IBR):** The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.

- **Input/Output Devices** –Program or data is read into main memory from the *input device* or secondary storage under the control of CPU input instruction. *Output devices* are used to output the information from a computer.
- **Buses –** Data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory, by the means of Buses. Types:
  - **Data Bus:** It carries data among the memory unit, the I/O devices, and the processor.
  - **Address Bus:** It carries the address of data (not the actual data) between memory and processor.
  - **Control Bus:** It carries control commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer.

What is RISC Architecture?

- A Reduced Instruction Set Computer is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions rather than the highly-specialized set of instructions typically found in other architectures.
- Example: ARM7, ARM9



The Sun Microsystems UltraSPARC processor is a type of RISC microprocessor.

Reduced Instruction Set Computer (RISC) architecture

Characteristics of RISC:-

- Simpler instruction, hence simple instruction decoding.
- Instruction come under size of one word.
- Instruction take single clock cycle to get executed.
- More number of general purpose register.
- Simple Addressing Modes.
- Less Data types.
- Pipelining can be achieved.

What is CISC Architecture?

A complex instruction set computer (CISC) is a computer architecture in which single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instructions.

Example: Pentium

MCU 8 Bit CISC 5 V 44 Pin

Complex Instruction Set Computer (CISC) architecture

Characteristic of CISC –

- Complex instruction, hence complex instruction decoding.
- Instruction are larger than one word size.
- Instruction may take more than single clock cycle to get executed.
- Less number of general purpose register as operation get performed in memory itself.
- Complex Addressing Modes.
- More Data types.

Data types

Binary information in digital computers is stored in memory or processor registers. Registers contain either data or control information. Control information is a bit or a group of bits used to specify the sequence of command signals needed for manipulation of the data in other registers. Data are numbers and other binary-coded information that are operated on to achieve required computational results .

The data types found in the registers of digital computers may be classified as being one of the following categories:

1) numbers used in arithmetic computations,

2) letters of the alphabet used in data processing. and

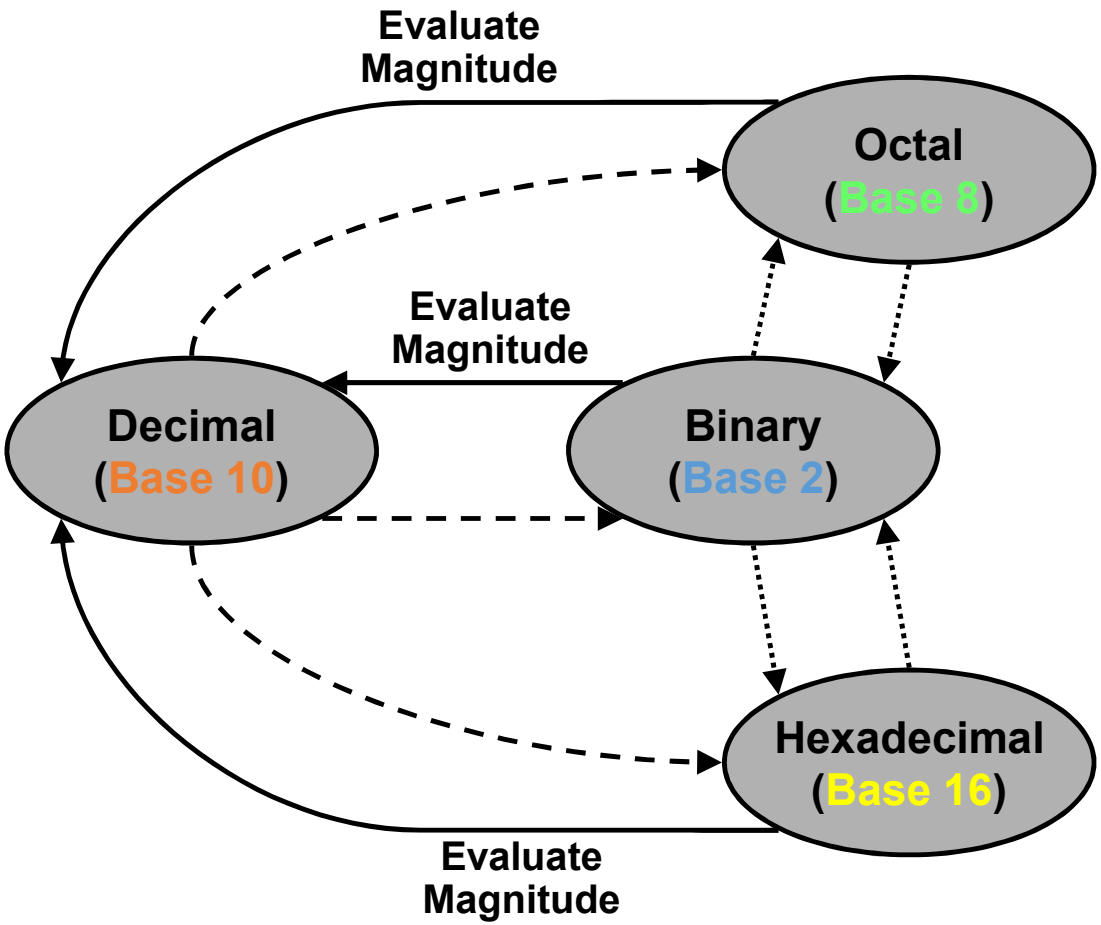
3) other discrete symbols used for specific purposes.

All types of data, except binary numbers, are represented in computer registers in binary-coded form. because registers are made up of flip-flops and flip-flops are two-state devices that can store only l's and O's. The binary number system is the most natural system to differ use in a digital computer. But sometimes it is convenient to employ diffrent number systems, especially the decimal number system, since it is used by people to perform arithmetic computations.

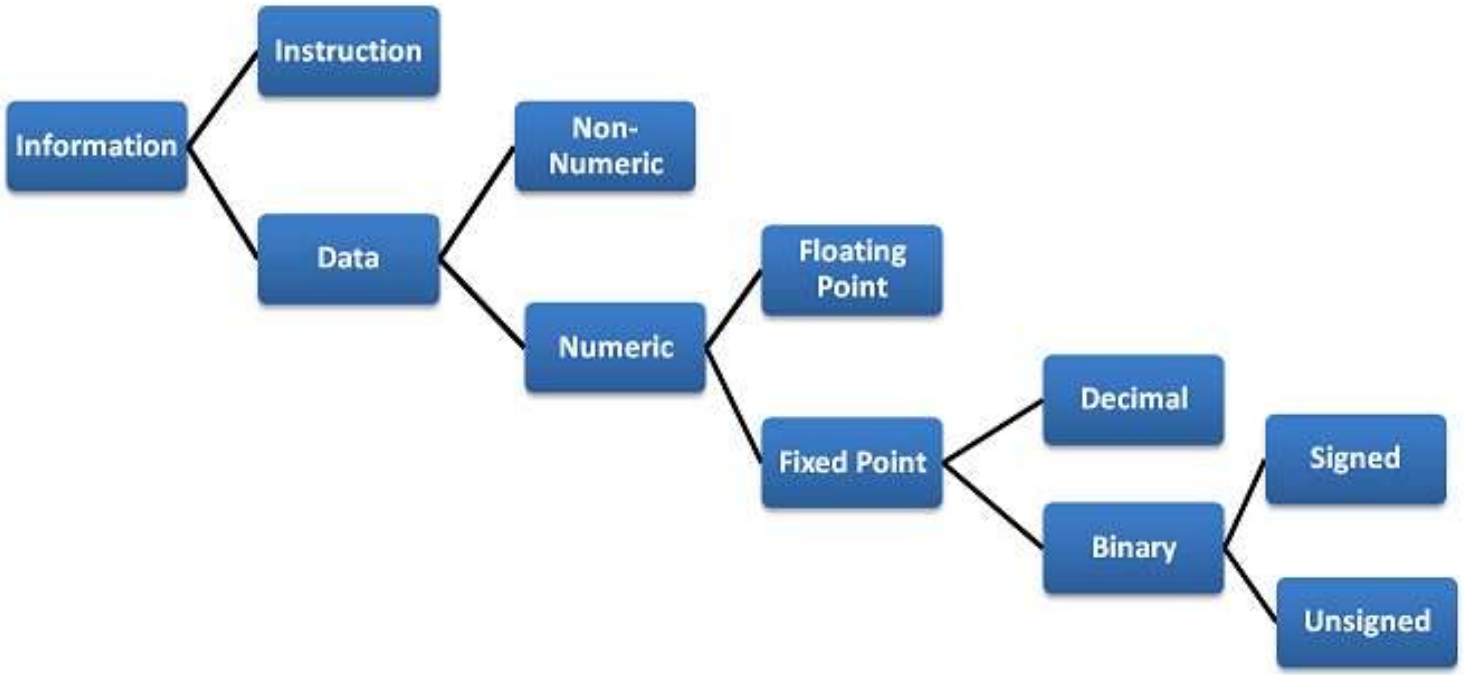
Data Representation (Data Types)

- Number Systems
  - Binary Number System (base-2)
  - Decimal Number System (base-10)
  - Octal Number System (base-8)
  - Hexadecimal Number System (base-16)

Number Base Conversions



Typical Internal data representation types



Number Systems

radix

A number system of *base*, or *radix*, r is a system that uses distinct symbols for r digits. Numbers are represented by a string of digit symbols. To determine the quantity that the number represents, it is necessary to multiply each digit by an integer power of r and then form the sum of all weighted digits. For example, the decimal number system in everyday use employs the radix 10 system. The 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The string of digits 724.5 is interpreted to represent the quantity

decimal

$$7 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1}$$

that is, 7 hundreds, plus 2 tens, plus 4 units, plus 5 tenths. Every decimal number can be similarly interpreted to find the quantity it represents.

binary

The *binary* number system uses the radix 2. The two digit symbols used are 0 and 1. The string of digits 101101 is interpreted to represent the quantity

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45$$

To distinguish between different radix numbers, the digits will be enclosed in parentheses and the radix of the number inserted as a subscript. For example, to show the equality between decimal and binary forty-five we will write $(101101)_2 = (45)_{10}$.

octal

octal
hexademical

Besides the decimal and binary number systems, the *octal* (radix 8) and *hexadecimal* (radix 16) are important in digital computer work. The eight symbols of the octal system are 0, 1, 2, 3, 4, 5, 6, and 7. The 16 symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The last six symbols are, unfortunately, identical to the letters of the alphabet and can cause confusion at times. However, this is the convention that has been adopted. When used to represent hexadecimal digits, the symbols A, B, C, D, E, F correspond to the decimal numbers 10, 11, 12, 13, 14, 15, respectively.

A number in radix r can be converted to the familiar decimal system by forming the sum of the weighted digits. For example, octal 736.4 is converted to decimal as follows:

$$(736.4)_8 = 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1}$$
$$= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 = (478.5)_{10}$$

The equivalent decimal number of hexadecimal F3 is obtained from the following calculation:

$$(F3)_{16} = F \times 16 + 3 = 15 \times 16 + 3 = (243)_{10}$$

Conversion of decimal 41.6875 into binary.

Integer = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

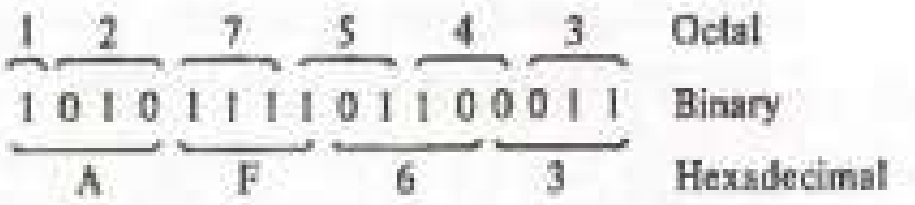
$(41)_{10} = (101001)_2$

Fraction = 0.6875

0.6875
2
1.3750
x 2
0.7500
x 2
1.5000
x 2
1.0000

$(0.6875)_{10} = (0.1011)_2$

$(41.6875)_{10} = (101001.1011)_2$



Binary, octal, and hexadecimal conversion.

TABLE 3-1 Binary-Coded Octal Numbers

Octal number	Binary-coded octal	Decimal equivalent	
0	000	0	↑
1	001	1	|
2	010	2	Code
3	011	3	for one
4	100	4	octal
5	101	5	digit
6	110	6	|
7	111	7	↓
10	001 000	8	
11	001 001	9	
12	001 010	10	
24	010 100	20	
62	110 010	50	
143	001 100 011	99	
370	011 111 000	248	

TABLE 3-2 Binary-Coded Hexadecimal Numbers

Hexadecimal number	Binary-coded hexadecimal	Decimal equivalent	
0	0000	0	
1	0001	1	
2	0010	2	
3	0011	3	
4	0100	4	
5	0101	5	
6	0110	6	Code
7	0111	7	for one
8	1000	8	hexadecimal
9	1001	9	digit
A	1010	10	
B	1011	11	
C	1100	12	
D	1101	13	
E	1110	14	
F	1111	15	
14	0001 0100	20	
32	0011 0010	50	
63	0110 0011	99	
F8	1111 1000	248	

TABLE 3-3 Binary-Coded Decimal (BCD) Numbers

Decimal number	Binary-coded decimal (BCD) number	
0	0000	
1	0001	
2	0010	
3	0011	Code for one decimal digit
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	
10	0001 0000	
20	0010 0000	
50	0101 0000	
99	1001 1001	
248	0010 0100 1000	

TABLE 3-4 American Standard Code for Information Interchange (ASCII)

Character	Binary code	Character	Binary code
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	space	010 0000
N	100 1110	.	010 1110
O	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0001	$	010 0100
R	101 0010	*	010 1010
S	101 0011	)	010 1001
T	101 0100	−	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
W	101 0111	=	011 1101
X	101 1000		
Y	101 1001		
Z	101 1010		

Different Types of Binary Codes

- In the digital logic circuits, each decimal or piece of information is represented through an equivalent combination of binary digits. A complete group of such combinations, which represents numbers, layers, or symbols, is known as a digital or binary code.
- Codes are used for security purposes so that in case the message has been detected, it may not be deciphered or decoded.

Further codes being used in the digital systems are of the following two types

- Numeric weighted code
- Special binary codes or non-weighted code

Weighted Codes

- The values assigned to consecutive places in the decimal system which is a place value system are 10^4, 10^3, 10^2, 10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}… and so on from left to right. It is easily can be understood that the weight of digit of the decimal system is '10'.

For example:

$$(3546.25)_{10} = 3 \times 10^3 + 5 \times 10^2 + 4 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$$

The weights in a binary system are 2^4, 2^3, 2^2, 2^1, 2^0, 2^{-1}, 2^{-2}, 2^{-3}… from left to right. It is easily can be understood that the weight of digit of the binary system is '2'.

For example:

$$(1110110)_2 = 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

8421 BCD Code

- 8421 code A weighted code in which each decimal digit 0 through 9 is represented by a four-bit codeword
- These are the codes, wherein every digit has got some particular weight.
- It is less efficient than the pure binary, in the sense that it requires more bits.

Ex: $(14)_2 \rightarrow 1110$ In case of 8421 code → 0001 0100

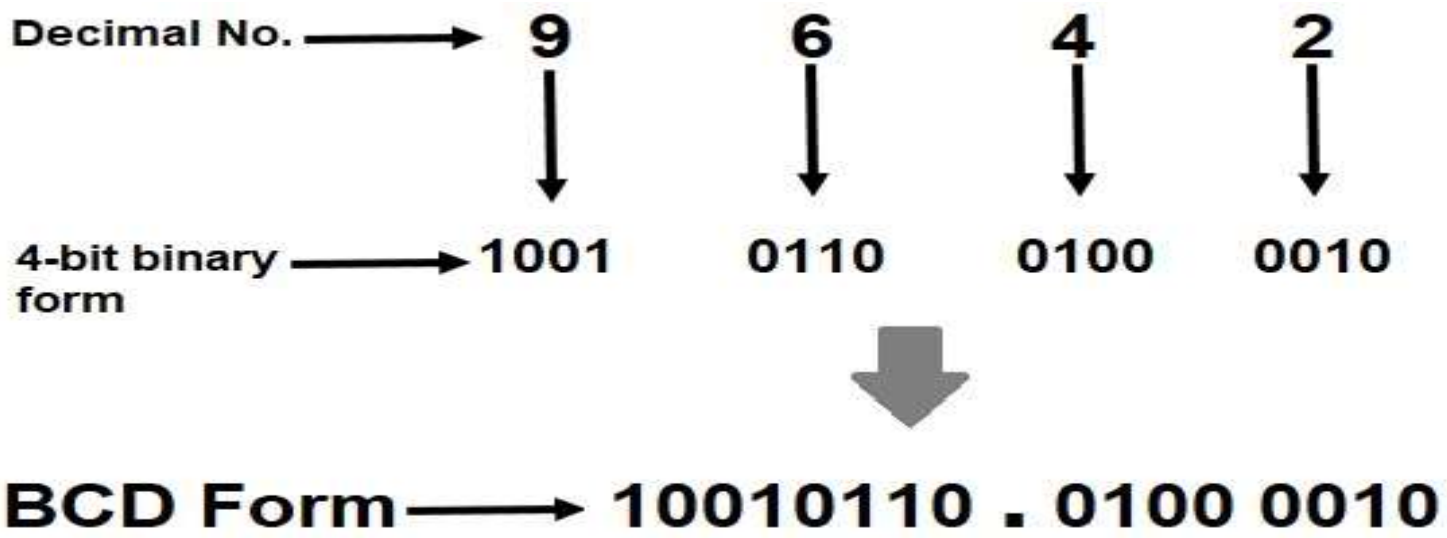
In this code, there are six illegal combinations: 1010, 1011, 1100, 1101, 1110, 1111

- For example, 8421 BCD (binary coded decimal) is a weighted code.

Example:

$$23 \rightarrow 2\ 3$$
$$code \rightarrow 0010\ 0011$$

Example



- 1 (94)
- 2 (24)
- 3 (29)

DECIMAL NUMBER	BINARY NUMBER	4 BIT EXPRESSION(8421)
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111
8	1000	1000
9	1001	1001

Decimal digit	*(BCD) 8421*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD Addition

Example

										1	1	1				
	2	5			0	0	1	0	0	1	0	1				
+	1	3		+	0	0	0	1	0	0	1	1				
	3	8			0	0	1	1	1	0	0	0				

If there is an illegal code or carry is generated as a result of addition, then add 0110 to particular that 4 bits of result.

BCD Addition

Example-2

		1	111		
679.6		0110	0111	1001	.0110
+ 536.8	+	0101	0011	0110	.1000
1216.4		1011	1010	1111	.1110
	+	0110	0110	0110	.0110
		0001	0000	0101	.0100
		1	1	1	1
	0001	0010	0001	0110	.0100

All are illegal codes

Add 0110 to each

Propagate carry

BCD Subtraction

Example-1

	3	8		0	0	1	1	1	0	0	0
-	1	5	-	0	0	0	1	0	1	0	1
	2	3		0	0	1	0	0	0	1	1

If there is an illegal code or carry is generated as a result of addition, then add 0110 to particular that 4 bits of result.

BCD Subtraction

Example 2

206.7	0010	0000	0110	.0111	
147.8	0001	0100	0111	.1000	
58.9	0000	1011	1110	.1111	Borrows are present
		0110	0110	0110	Subtract 0110
		0101	1000	.1001	corrected Difference

2421 Code

- This code also a 4 bit application code where the binary weights carry 2, 4, 2, 1 from left to right.

DECIMAL NUMBER	BINARY NUMBER	2421 CODE
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	1011
6	110	1100
7	111	1101
8	1000	1110
9	1001	1111

5211 Code

- This code is also a 4 bit application code where the binary weights carry 5, 2, 1, 1 from left to right.

DECIMAL NUMBER	BINARY NUMBER	5211 CODE
0	0	0000
1	1	0001
2	10	0011
3	11	0101
4	100	0111
5	101	1000
6	110	1010
7	111	1100
8	1000	1110
9	1001	1111

Non-Weighted Codes

- Some of the codes will not follow the weights of the sequence binary numbers these are called as non-weighted codes. ASCII code, Excess 3 and Grey code are some of the examples where they are coded for some special purpose applications and they do not follow the weighted binary number calculations.

Excess-3 Code

- Excess-3 code is a non-weighted code used to represent decimal numbers.
- Excess-3 binary code is an unweighted self-complementary code BCD code.
- The XS-Code has six invalid states 0000, 0001, 0010, 1101, 1110, and 1111.
- Self-Complementary property means that the 1's complement of an Excess-3 number is the Excess-3 code of the 9's complement of the corresponding decimal number.
- The Excess-3 codewords are derived from the 8421 BCD code words by adding "3" to each codeword in 8421. The Excess-3 code is self-complementing code. The excess-3 codes can be obtained as follows:

$48 \rightarrow$	4	8
	$+3$	$+3$
Excess-3	7	11
	0111	1011

Example

- Find the excess-3 code of $(237.75)_{10}$

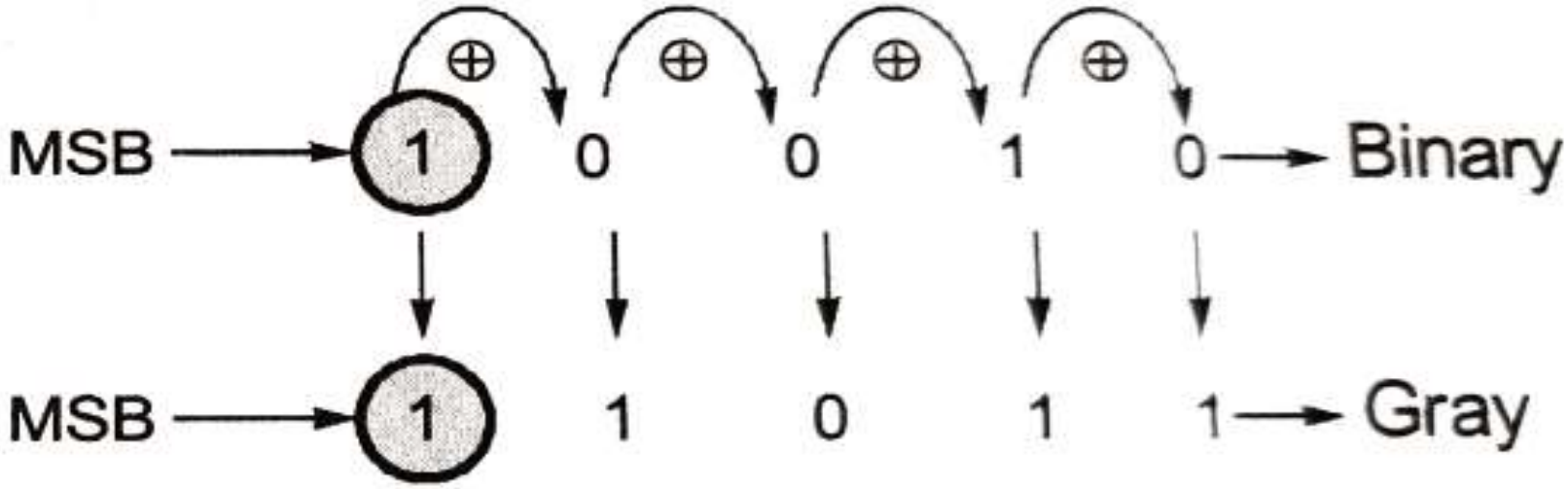
Solution

- The excess-3 code for $(237)_{10}$ is obtained by adding 3 to all the digits individually, that is 2, 3 and 7 will become 5, 6 and 10 respectively. These 5, 6 and 10 decimals have to be converted into binary form and the result is 010101101010.
- The excess-3 code for $(.75)_{10}$ is obtained by replacing 7 and 5 with 10 and 8 respectively by adding 3 to each digit. That is, the excess-3 code for $(.75)_{10}$ is $.10101000$.
- Combining the results of the integral and fractional parts, the excess-3 code for
- $(237.75)_{10}$ is 010101101010.10101000

Gray Code

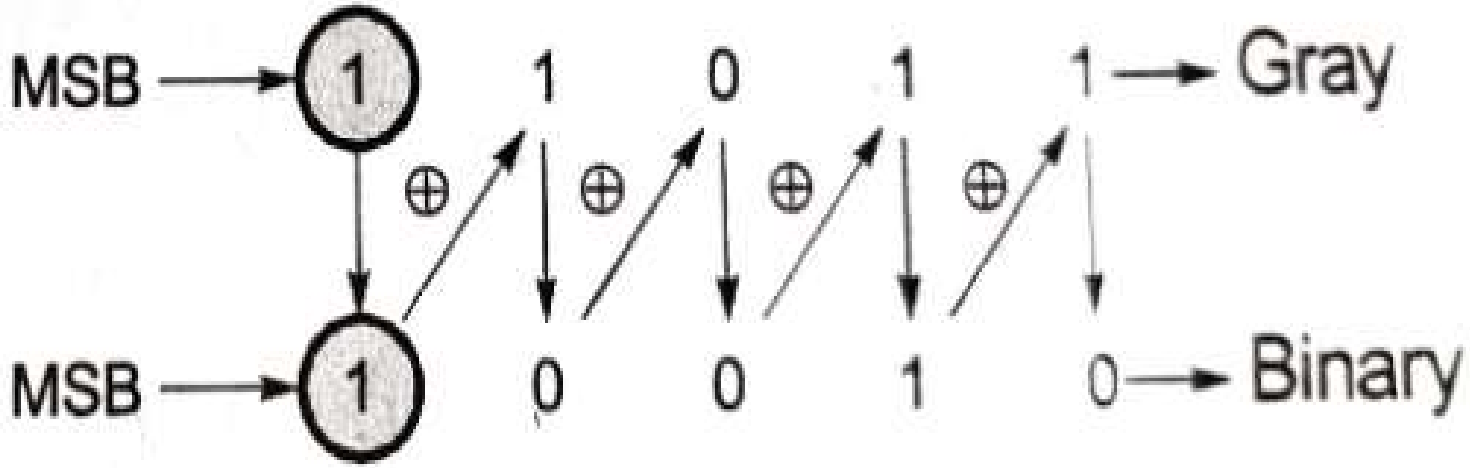
- It is the non-weighted code that implies there are no specific weights allocated to the bit position. It has a very unique feature that only one bit will alter each time the decimal number is incremented.
- These types of codes are also called minimum change codes.
- As only one bit turns at a time, the gray code is termed a unit distance code. The code is cyclic.

Binary to Gray codes Conversion



$(10010)_2 = (11011)_{Gray}$

Gray to Binary Code Conversion



$$(11011)_{Gray} = (10010)_2$$

$(r-1)$'s Complement

9's complement

Given a number N in base r having n digits, the $(r-1)$'s complement of N is defined as $(r^n-1)-N$. For decimal numbers $r=10$ and $r-1=9$, so the 9's complement of N is $(10^n-1)-N$. Now, 10^n represents a number that consists of a single 1 followed by n 0's. 10^n-1 is a number represented by n 9's. For example, with $n=4$ we have $10^4=10000$ and $10^4-1=9999$. It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9. For example, the 9's complement of 546700 is $999999-546700=453299$ and the 9's complement of 12389 is $99999-12389=87610$.

1's complement

For binary numbers, $r=2$ and $r-1=1$, so the 1's complement of N is $(2^n-1)-N$. Again, 2^n is represented by a binary number that consists of a 1 followed by n 0's. 2^n-1 is a binary number represented by n 1's. For example, with $n=4$, we have $2^4=(10000)_2$ and $2^4-1=(1111)_2$. Thus the 1's complement of a binary number is obtained by subtracting each digit from 1. However, the subtraction of a binary digit from 1 causes the bit to change from 0 to 1 or from 1 to 0. Therefore, the 1's complement of a binary number is formed by changing 1's into 0's and 0's into 1's. For example, the 1's complement of 1011001 is 0100110 and the 1's complement of 0001111 is 1110000.

The $(r-1)$'s complement of octal or hexadecimal numbers are obtained by subtracting each digit from 7 or F (decimal 15) respectively.

(r's) Complement

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and 0 for $N = 0$. Comparing with the $(r - 1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r - 1)$'s complement since $r^n - N = [(r^n - 1) - N] + 1$. Thus the 10's complement of the decimal 2389 is 7610 + 1 = 7611 and is obtained by adding 1 to the 9's complement value. The 2's complement of binary 101100 is 010011 + 1 = 010100 and is obtained by adding 1 to the 1's complement value.

10's complement

Since 10^n is a number represented by a 1 followed by n 0's, then $10^n - N$, which is the 10's complement of N, can be formed also be leaving all least significant 0's unchanged, subtracting the first nonzero least significant digit from 10, and then subtracting all higher significant digits from 9. The 10's complement of 246700 is 753300 and is obtained by leaving the two zeros unchanged, subtracting 7 from 10, and subtracting the other three digits from 9. Similarly, the 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged, and then replacing 1's by 0's and 0's by 1's in all other higher significant bits. The 2's complement of 1101100 is 0010100 and is obtained by leaving the two low-order 0's and the first 1 unchanged, and then replacing 1's by 0's and 0's by 1's in the other four most significant bits.

2's complement

Subtraction of Unsigned Numbers

The direct method of subtraction taught in elementary schools uses the borrow concept. In this method we borrow a 1 from a higher significant position when the minuend digit is smaller than the corresponding subtrahend digit. This seems to be easiest when people perform subtraction with paper and pencil. When subtraction is implemented with digital hardware, this method is found to be less efficient than the method that uses complements.

subtraction

The subtraction of two n-digit unsigned numbers $M - N (N \neq 0)$ in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. This performs $M + (r^n - N) = M - N + r^n$.

end carry

2. If $M \geq N$, the sum will produce an end carry r^n which is discarded, and what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Consider, for example, the subtraction 72532 − 13250 = 59282. The 10's complement of 13250 is 86750. Therefore:

$$\begin{aligned} M &= 72532 \\ \text{10's complement of } N &= +86750 \\ \text{Sum} &= 159282 \\ \text{Discard end carry } 10^5 &= -100000 \\ \text{Answer} &= 59282 \end{aligned}$$

Now consider an example with $M < N$. The subtraction 13250 − 72532 produces negative 59282. Using the procedure with complements, we have

$$\begin{aligned} M &= 13250 \\ \text{10's complement of } N &= +27468 \\ \text{Sum} &= 40718 \end{aligned}$$

There is no end carry

Answer is negative 59282 = 10's complement of 40718

$$
\begin{aligned}
X &= 1010100 \\
\text{2's complement of } Y &= +0111101 \\
\text{Sum} &= 10010001 \\
\text{Discard end carry } 2^7 &= -10000000 \\
\text{Answer: } X - Y &= 0010001
\end{aligned}
$$

$$
\begin{aligned}
Y &= 1000011 \\
\text{2's complement of } X &= +0101100 \\
\text{Sum} &= 1101111
\end{aligned}
$$

There is no end carry

Answer is negative 0010001 = 2's complement of 1101111

3-3 Fixed-Point Representation

Positive integers, including zero, can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values. In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations, computers must represent everything with 1's and 0's, including the sign of a number. As a consequence, it is customary to represent the sign with a bit placed in the leftmost position of the number. The convention is to make the sign bit equal to 0 for positive and to 1 for negative.

binary point

In addition to the sign, a number may have a binary (or decimal) point. The position of the binary point is needed to represent fractions, integers, or mixed integer–fraction numbers. The representation of the binary point in a register is complicated by the fact that it is characterized by a position in the register. There are two ways of specifying the position of the binary point in a register: by giving it a fixed position or by employing a floating-point representation. The fixed-point method assumes that the binary point is always fixed in one position. The two positions most widely used are (1) a binary point in the extreme left of the register to make the stored number a fraction, and (2) a binary point in the extreme right of the register to make the stored number an integer. In either case, the binary point is not actually present, but its presence is assumed from the fact that the number stored in the register is treated as a fraction or as an integer. The floating-point representation uses a second register to store a number that designates the position of the decimal point in the first register. Floating-point representation is discussed further in the next section.

Fixed point Integer Number Representation

- Fixed point numbers are also known as whole numbers or Integers.
- Unsigned Integer: A positive number including zero can be represented in this format. All the allotted bits are utilised in defining the number. So if one is using 8 bits to represent the unsigned integer, the range of values that can be represented is 28 i.e. "0" to "255". If 16 bits are used for representing then the range is 216 i.e. "0 to 65535".
- Signed Integer: In this format negative numbers, zero, and positive numbers can be represented. A sign bit indicates the magnitude direction as positive or negative. There are three possible representations for signed integer and these are Sign Magnitude format, 1's Compliment format and 2's Complement format.
- *Sign Magnitude format:* Most Significant Bit (MSB) is reserved for indicating the direction of the magnitude (value). A "0" on MSB means a positive number and a "1" on MSB means a negative number. If n bits are used for representation, n-1 bits indicate the absolute value of the number.

Examples for n=8:

0010 1111 = + 47 Decimal (Positive number)

1010 1111 = - 47 Decimal (Negative Number)

0111 1110 = +126 (Positive number)

1111 1110 = -126 (Negative Number)

0000 0000 = + 0 (Postive Number)

1000 0000 = - 0 (Negative Number)

Sign Magnitude representation shortcomings

- Zero can be represented in two ways causing redundancy and confusion.
- The total range for magnitude representation is limited to 2n-1, although n bits were accounted.
- The separate sign bit makes the addition and subtraction more complicated. Also, comparing two numbers is not straightforward.

1’s Complement format:

- In this format too, MSB is reserved as the sign bit. But the difference is in representing the Magnitude part of the value for negative numbers (magnitude) is inversed and hence called 1’s Complement form. The positive numbers are represented as it is in binary.

Examples for n=8:

0010 1111 = + 47 Decimal (Positive number)

1101 0000 = - 47 Decimal (Negative Number)

0111 1110 = +126 (Positive number)

1000 0001 = -126 (Negative Number)

0000 0000 = + 0 (Postive Number)

1111 1111 = - 0 (Negative Number)

Converting a given binary number to its 2's complement form

Step 1. -x = x' + 1 where x' is the one's complement of x.

Step 2 Extend the data width of the number, fill up with sign extension i.e. MSB bit is used to fill the bits.

Example: **-47 decimal over 8bit representation**

```
Binary equivalent of + 47 is      0010 1111
Binary equivalent of - 47 is      1010 1111 (Sign Magnitude Form)
1's complement equivalent is      1101 0000
2's complement equivalent is      1101 0001
```

As you can see zero is not getting represented with redundancy. There is only one way of representing zero. The other problem of the complexity of the arithmetic operation is also eliminated in 2's complement representation. Subtraction is done as Addition.

3-4 Floating-Point Representation

mantissa

exponent

The floating-point representation of a number has two parts. The first part represents a signed, fixed-point number called the mantissa. The second part designates the position of the decimal (or binary) point and is called the exponent. The fixed-point mantissa may be a fraction or an integer. For example, the decimal number +6132.789 is represented in floating-point with a fraction and an exponent as follows:

Fraction	*Exponent*
+0.6132789	+04

The value of the exponent indicates that the actual position of the decimal point is four positions to the right of the indicated decimal point in the fraction. This representation is equivalent to the scientific notation $+0.6132789 \times 10^{+4}$.

Floating-point is always interpreted to represent a number in the following form:

$$m \times r^e$$

Only the mantissa m and the exponent e are physically represented in the register (including their signs). The radix r and the radix-point position of the mantissa are always assumed. The circuits that manipulate the floating-point numbers in registers conform with these two assumptions in order to provide the correct computational results.

A floating-point binary number is represented in a similar manner except that it uses base 2 for the exponent. For example, the binary number +1001.11 is represented with an 8-bit fraction and 6-bit exponent as follows:

Fraction	*Exponent*
01001110	000100

fraction

The fraction has a 0 in the leftmost position to denote positive. The binary point of the fraction follows the sign bit but is not shown in the register. The exponent has the equivalent binary number +4. The floating-point number is equivalent to

$$m \times 2^{e} = +(.1001110)_2 \times 2^{+4}$$

normalization

A floating-point number is said to be *normalized* if the most significant digit of the mantissa is nonzero. For example, the decimal number 350 is normalized but 00035 is not. Regardless of where the position of the radix point is assumed to be in the mantissa, the number is normalized only if its leftmost digit is nonzero.