

Number System

In any electronic system, data is stored and manipulated using numbers. Numbers are used to represent the values of integers, reals, characters etc. Every number system has a *base*. For example, the decimal number system has base 10; binary system has base 2; octal base 8 etc. If the base of a number system is b , it will have b different symbols corresponding to the integer values $0, 1, 2, \dots, b-1$. An n -digit number $(x_1x_2\dots x_n)_b$ in a base b number system, with each x_i corresponding to one of the integer values in the set $\{0, 1, 2, \dots, b-1\}$, represents the following integer value in the decimal (base 10) number system.

$$x_1 \times b^{n-1} + x_2 \times b^{n-2} + \dots + x_{n-1} \times b + x_n$$

Example 1: $(678)_{16} = 6 \times 16^2 + 7 \times 16^1 + 8 = (1586)_{10}$
 $(110011)_2 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1 + 1 = (51)_{10}$ □

A real number $(x_1x_2\dots x_n \cdot x_{n+1}x_{n+2}\dots x_{n+m})_b$ represents the following real number in base 10 system.
 $x_1 \times b^{n-1} + x_2 \times b^{n-2} + \dots + x_{n-1} \times b + x_n + x_{n+1} \times b^{-1} + x_{n+2} \times b^{-2} + \dots + x_{n+m} \times b^{-m}$

Example 2: $(323.12)_4 = 3 \times 4^3 + 2 \times 4^2 + 3 + 1 \times 4^{-1} + 2 \times 4^{-2} = (59.375)_{10}$ □

Converting decimal to base b : To convert a decimal number to base b system, the number is divided repetitively by b . The remainders are noted. The first remainder gives the least significant digit, whereas, the successive remainders give the more significant digits. The process continues till the quotient becomes zero.

Example 3: Convert $(723)_{10}$ to binary system.

	Quotient	Remainder
$723/2 =$	361	1
$361/2 =$	180	1
$180/2 =$	90	0
$90/2 =$	45	0
$45/2 =$	22	1
$22/2 =$	11	0
$11/2 =$	5	1
$5/2 =$	2	1
$2/2 =$	1	0
$1/2 =$	0	1

Hence, $(723)_{10} = (1011010011)_2$. □

Example 4: Convert $(557)_{10}$ to hexadecimal number system.

	Quotient	Remainder
$557/16 =$	34	13 (D)
$34/16 =$	2	2
$2/16 =$	0	2

Thus, $(557)_{10} = (22D)_{16}$. □

For real numbers, the integer position is converted as noted earlier. For the fractional part, it is repetitively multiplied by the base value b . The integer portion of the first product is taken as the most significant digit of fractional part. The fractional part of the product is again multiplied by b . The process continues till the product becomes zero, or desired level of accuracy reached.

Example 5: Convert $(5.75)_{10}$ to binary system.

	Quotient	Remainder		Integer	Fraction
$5/2 =$	2	1	$0.75 \times 2 =$	1	0.5
$2/2 =$	1	0	$0.5 \times 2 =$	1	0
$1/2 =$	0	1	$0.0 \times 2 =$	0	0

Thus, $(5.75)_{10} = (101.11)_2$.

Converting binary numbers to octal and hexadecimal: This can be accomplished in two different ways.

1. Binary $\rightarrow$ Decimal $\rightarrow$ Octal / Hexadecimal.
2. Since octal system has base 8, three successive bits of a binary number represents a single octal digit. Similarly, four consecutive bits represent a hexadecimal number. For example, $(1001101.101101)_{10}$ corresponds to $(115.55)_8$ and $(4D.B4)_{16}$.

Addition of binary numbers: Addition of two n -bit numbers results into a $(n+1)$ -bit number. The least significant n bits constitute the sum, while the $(n+1)^{\text{th}}$ bit is the carry.

Example 6:

$$\begin{array}{r}
 101101 \\
 + 011010 \\
 \hline
 1000111
 \end{array}
 \quad
 \begin{array}{r}
 101010 \\
 + 001101 \\
 \hline
 0110111
 \end{array}$$

$\swarrow$ carry $\searrow$

$\rightarrow$ Led

2's complement representation: It is used to represent both positive and negative numbers uniformly. In 1's complement representation, a negative number $-m$ is represented by complementing each bit of the binary representation of $+m$. This leads to dual representation of 0: $+0 = 000\dots 0$ and $-0 = 111\dots 1$. 2's complement representation is obtained by adding 1 to the 1's complement of the positive number representation.

Example 7: 4-bit representation of -5.

$$\begin{array}{rcl}
 +5 & = & 0101 \\
 \text{1's complement} & = & 1010 \\
 & & +1 \\
 \text{2's complement} & = & 1011
 \end{array}$$

- An n -bit representation in 2's complement form can represent numbers -2^{n-1} to $2^{n-1} - 1$.
- Most significant bit of a number represented in 2's complement is zero for positive numbers and one for negative numbers.
- 2's complement representation in n -bit of a negative number $-m$ is same as the binary representation of the number $2^n - m$ in n bits.
- The subtraction $(m_1 - m_2)$ can be carried out by first taking the 2's complement of m_2 and adding it to m_1 .

Example 8: Compute $55_{10} - 44_{10}$ using binary arithmetic.

$$\begin{array}{lcl}
 55 = & 0110111 & \\
 44 = & 0101100 & \\
 \text{1's complement} = & 1010011 & \\
 \text{2's complement} = & 101010 &
 \end{array}
 \quad
 \begin{array}{rcl}
 55 - 44 = & & \\
 & \begin{array}{r}
 0110111 \\
 1010100 \\
 \hline
 10001011
 \end{array} & = (11)_{10} \\
 \text{discard} & \swarrow &
 \end{array}$$

Problems on Number Systems

1. Determine minimum suitable values for x and y , such that, $(49)_{(x-1)} = (35)_y$.

Soln.:

$$\begin{aligned}(49)_{(x-1)} &= (35)_y \\ \Rightarrow 4(x-1) + 9 &= 3y + 5 \\ \Rightarrow 4x + 5 &= 3y + 5 \\ \Rightarrow 4x &= 3y \\ \Rightarrow x &\geq 3y/4\end{aligned}$$

Also, $x \geq 11$, $y \geq 6$. The minimum values of x and y , satisfying these constraints are, $x = 12$, $y = 16$.

2. Perform the subtraction $(45.50 - 22.25)$ using 2's complement arithmetic.

Soln.: To add or subtract two fixed-point numbers, it is sufficient to add or subtract the underlying integers. Now, $45.50 = 101101.10_2$, $22.25 = 010110.01_2$. Thus, $-22.25 = 101001.11_2$. Hence $45.50 - 22.25 = 010111.01_2$, which is equal to 23.25.

Overview of Digital Logic

Though the signals of real-world are analog in nature, quantizing them into distinct levels and using digital values to represent them often helps to design the corresponding electronic circuitry that show better noise immunity and easier implementations. This leads to digital design of systems and correspond directly to the binary number system with two logic values. Several logic families are available, differing in characteristics, voltage levels and noise margins. For example, TTL considers input $< 0.5V$ as logic 0, while input $> 2.4V$ is taken as logic 1. On the other hand, CMOS considers voltage below $0.3 \times (V_{DD} - V_{SS})$ as logic 0 and voltages above $0.7 \times (V_{DD} - V_{SS})$ as logic 1.

Digital logic functions can broadly be classified into two categories – *combinational logic* and *sequential logic*. In a combinational logic circuit, the present output depends only on the present inputs (state less), while in a sequential logic circuit, output depends not only on the present input, but also on the past inputs as well (state oriented). In a combinational logic circuit, the basic components are logic gates.

Example 1 Figure 1 shows some logic gates with their truth tables. □

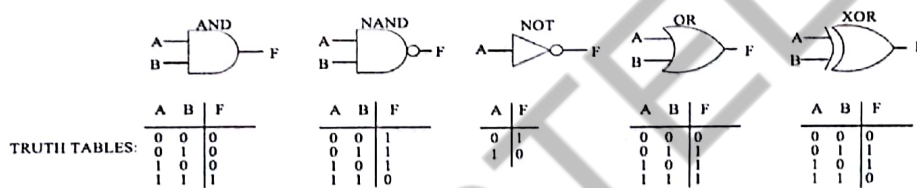


Figure 1: Some basic logic gates

A logic gate network realizes a given combinational function. The function is specified as a truth table. Karnaugh Map is used to minimize the function. This minimized form may be implemented using different gates. *ver*

Example 2 Three-input majority gate: Figure 2 shows a three-input majority gate design along with truth table, Karnaugh map and schematic. □

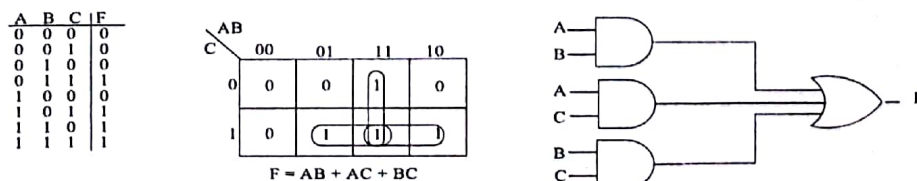


Figure 2: Three-input majority function

A very common purely combinational logic block present in processors is the adder circuitry. A *half adder* adds two bits generating *sum* and *carry* as output. Two such half adders can be connected together to design a full adder adding 3 bits. A number of full adders are utilized

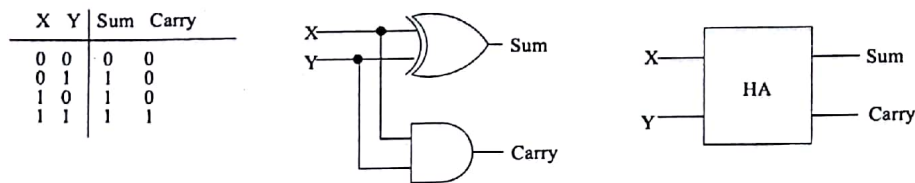


Figure 3: Half adder

to add numbers of larger sizes.

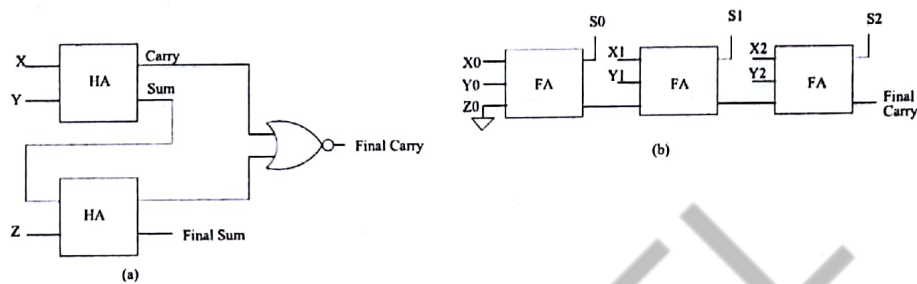


Figure 4: (a) A full adder (b) 3-bit adder

Sequential circuit contains one or more memory elements – commonly D flip-flops. A

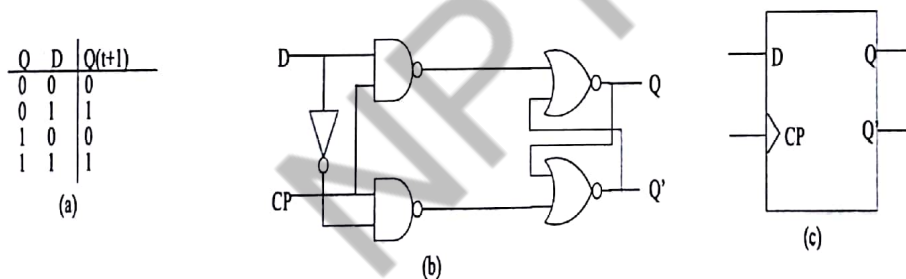


Figure 5: (a) Transition Table (b) Schematic (c) Block for D-flipflop

typical processor component using sequential logic is a register, that contains a collection of flip-flops that can be written into or read from parallelly.

Problems on Overview of Digital Logic

- Design a 4-bit magnitude comparator comparing the numbers $A_3A_2A_1A_0$ with $B_3B_2B_1B_0$ (A_3, B_3 are the MSBs). There are three outputs GT (to be asserted if $A > B$), LT (if $A < B$), EQ (if $A = B$).

Soln.: Though we can make the truth table having 256 entries in it, we also can directly write the logic equations driving the three outputs.

$$\begin{aligned}
 GT &= A_3 \text{ AND NOT}(B_3) \text{ OR } (A_3 \text{ XNOR } B_3) \text{ AND } (\\
 &\quad A_2 \text{ AND NOT}(B_2) \text{ OR } (A_2 \text{ XNOR } B_2) \text{ AND } (\\
 &\quad A_1 \text{ AND NOT}(B_1) \text{ OR } (A_1 \text{ XNOR } B_1) \text{ AND } (\\
 &\quad A_0 \text{ AND NOT}(B_0)) \\
 EQ &= (A_3 \text{ XNOR } B_3) \text{ AND } (A_2 \text{ XNOR } B_2) \text{ AND } (A_1 \text{ XNOR } B_1) \text{ AND } \\
 &\quad (A_0 \text{ XNOR } B_0) \\
 LT &= GT \text{ NOR } EQ
 \end{aligned}$$

The corresponding logic circuit is shown below.

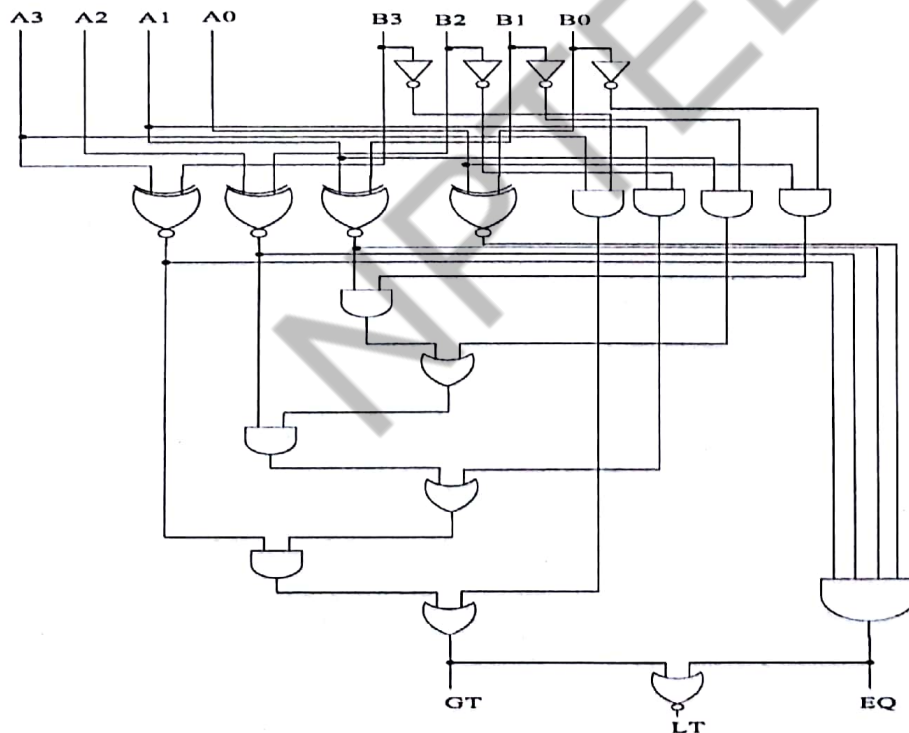


Figure 1: 4-bit magnitude comparator

- Design a 4-bit register with facility to parallelly load 4-bits, serially load content, shift the content, read the content parallelly and serially.

Soln.:

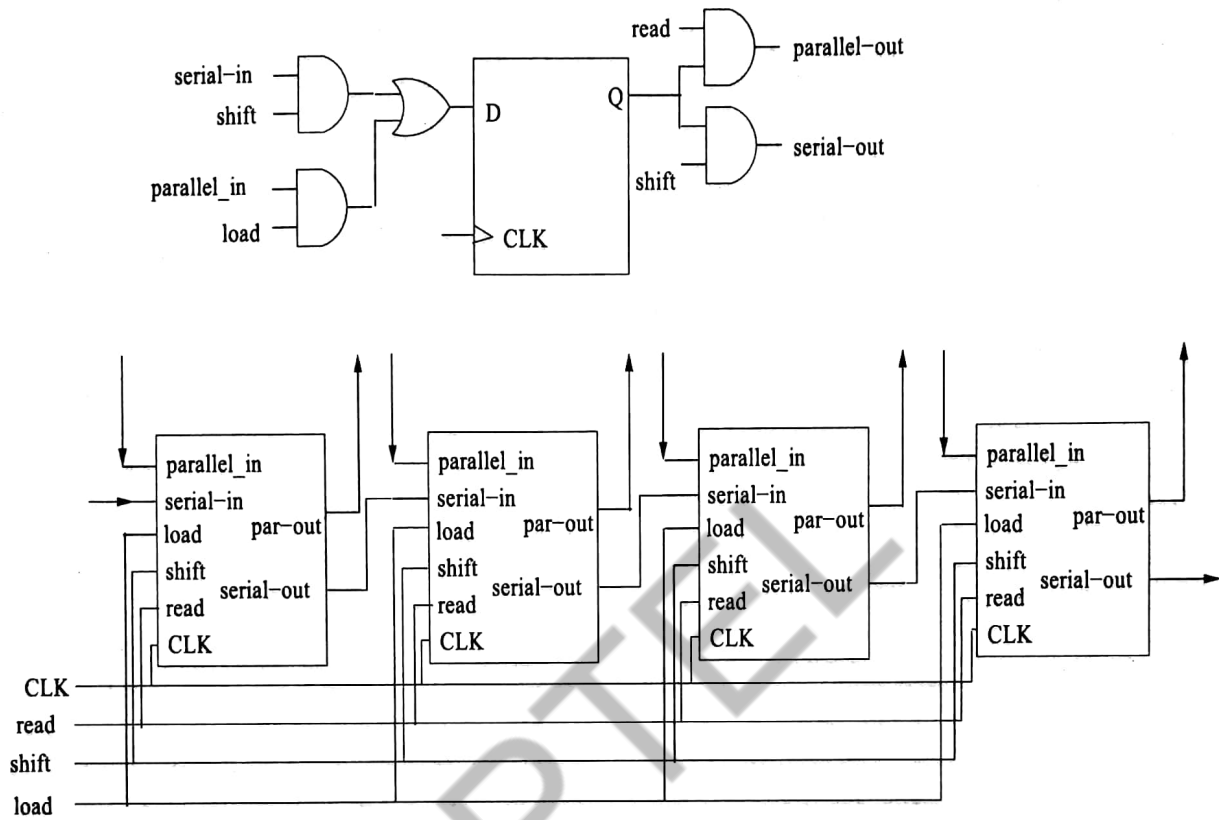
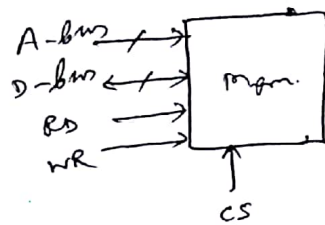


Figure 2: 4-bit register

Lex-3



64KB mem. with 16Kx8-bit (5)
 16Kx4-bit
 Folding/Expandable
 ↓
 decoder simplicity

Basic Computer Organization

A computer system consists of three different components – the *central processing unit (CPU)*, the *memory* and the *peripheral devices*. These are connected via *buses* – *address bus* (holding the address generated by the CPU), *data bus* (holding the data being transferred), and *control bus* (carrying the control information).

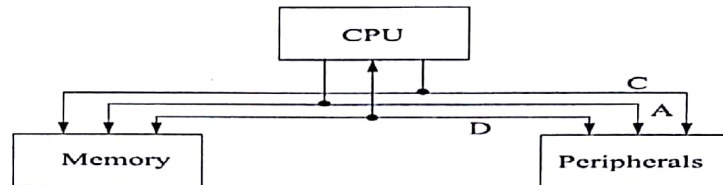


Figure 1: A typical computer system

The size of the data bus determines, in one access, how many bits of data can be transferred between CPU and memory/peripherals. The address bus size determines the number of different locations that can be accessed by the CPU. The individual memory chips may have different number of address and data pins. Decoders are used to interface them properly with the CPU.

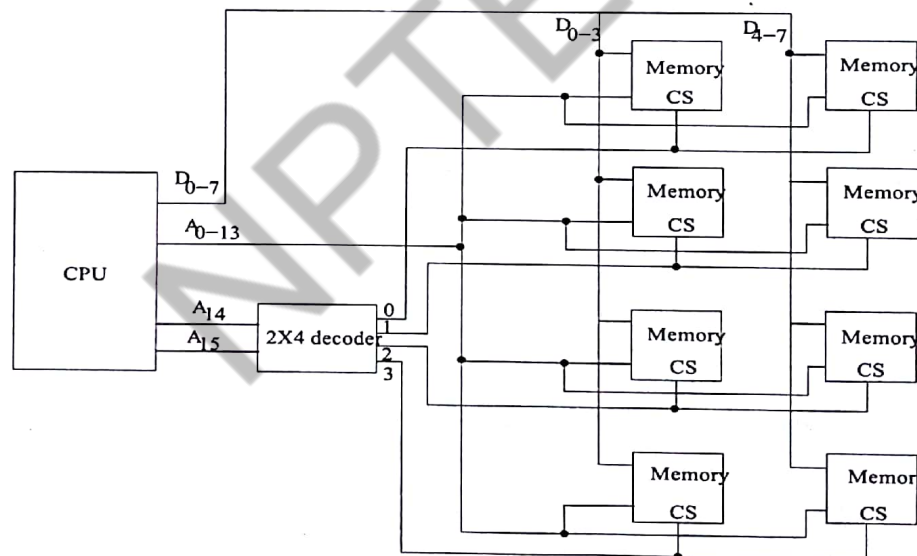


Figure 2: Interfacing ⁶⁴KB memory using ¹⁶Kx⁸/₄-bit memory chips

Internally, a CPU contains,

- a set of registers to store data temporarily.
- an *Arithmetic and Logic Unit (ALU)* to perform arithmetic and logic operations.
- few special purpose registers, such as,

local

- PC – *program counter* to point to the next instruction in memory.
- Flags – to hold the status information about the previous operation.
- IR – *instruction register* to decode the individual instruction opcodes and generate control information.

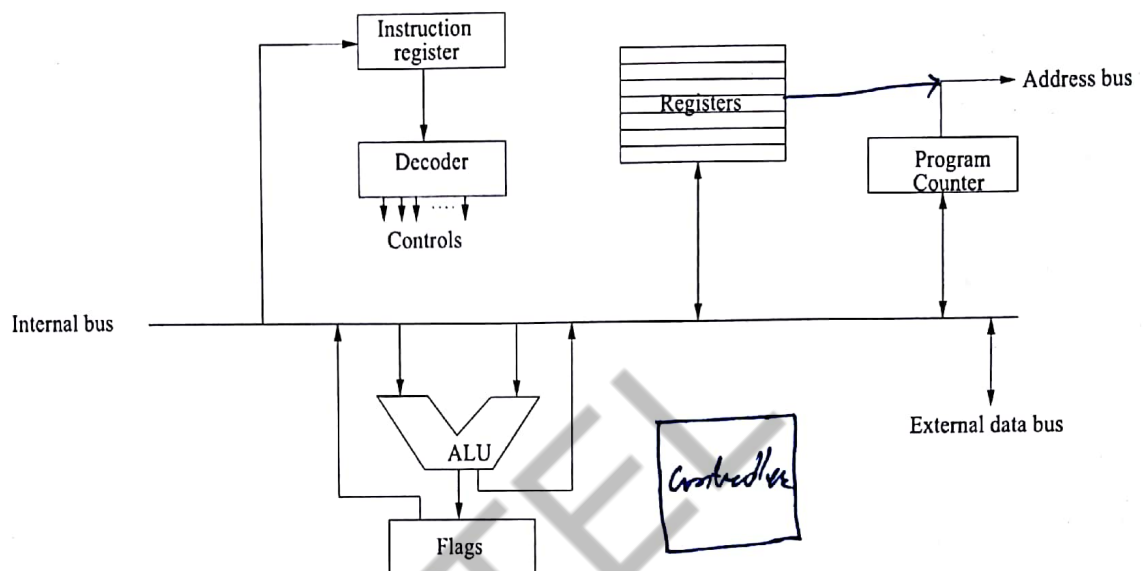


Figure 3: Structure of a typical CPU

The following may be noted regarding the operation of the CPU.

- CPU interfaces with the outside world via address and data bus.
- The address of the next instruction pointed to by PC is put on the address bus.
- Instruction reaches instruction register.
- Decoder decodes the opcode and decides the sequence in which various control signals are to be activated to accomplish the instruction.
- Instruction may need one/more operands to be brought from memory to the CPU.
- The operands are fetched from memory/register.
- Operation is carried out in the ALU, and result is stored back in register/memory location.
- PC is incremented to point to the next instruction.

In general, the components within the CPU can be realized using distinct integrated circuit chips. If all components are clubbed together into a single chip, the resulting chip is called a *microprocessor*.

There are different levels of memories present in a computer. The registers are the fastest, while the secondary storages are the slowest one.

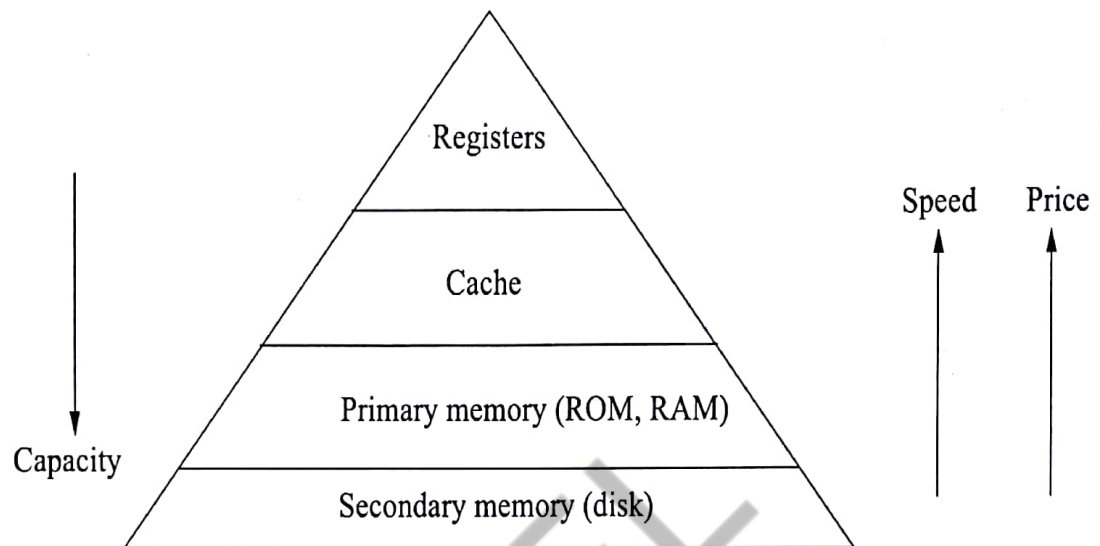


Figure 4: Memory hierarchy

Problems on Basic Computer Organization

- Design a memory system consisting of 8 Kbyte ROM and 16 Kbyte RAM spaces. The ROM starts at address 0000H, while the RAM starts at 8000H. If the CPU address is 16-bit, list all possible ways in which ROM and RAM spaces can be accessed. Assuming individual ROM and RAM chips to be of size 4 Kbytes each, design the full memory system.

Soln.:

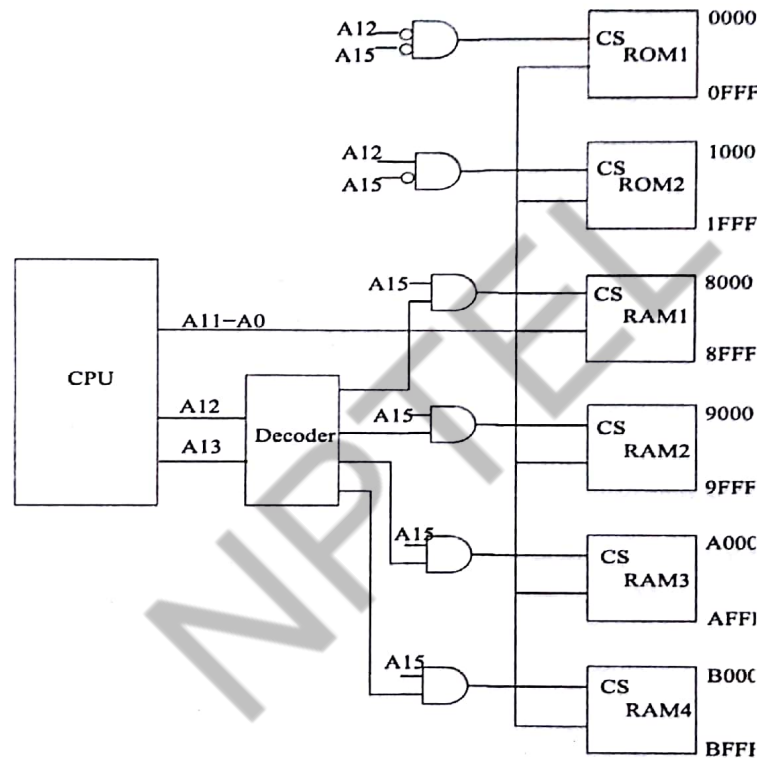


Figure 1: Memory interface

The address spaces are as follows.

ROM: 0000-1FFF, 2000-3FFF, 4000-5FFF, 6000-7FFF

RAM: 8000-BFFF, C000-FFFF

- For the datapath shown in the following figure, identify the control signals to be activated at different clock cycles to execute the following instructions. Assume control signals load and enable for each register, add for ALU, memory read and write etc. to be available from the decoder. The register *W* can hold an address, it is copied from the address field of an instruction in instruction register. The instructions are:

LOAD R0, 1000; 1000 is a memory location

ADD R0, R1, R2; R2 gets R0 + R1
STORE R0, 3000;

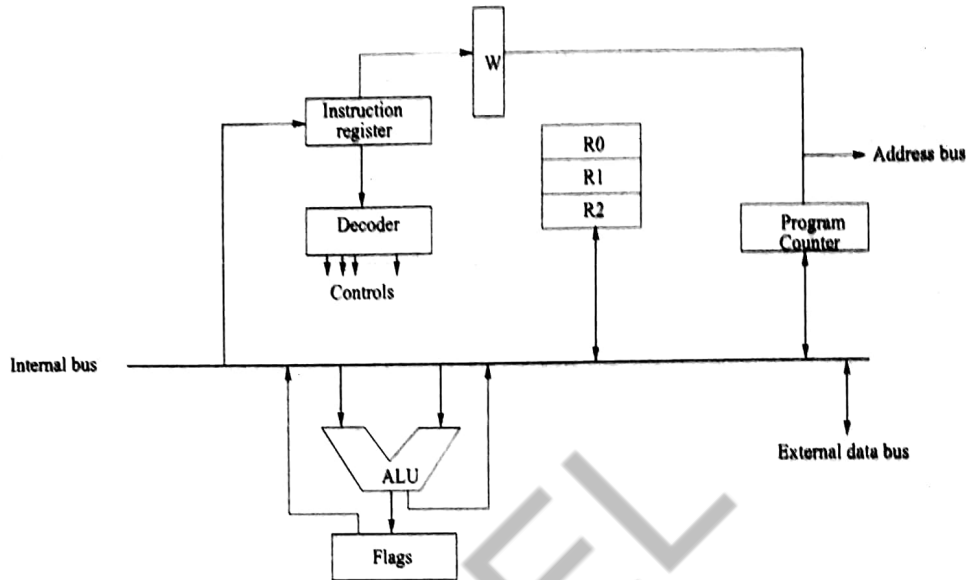


Figure 2: A datapath

Soln.:

LOAD R0, 1000: t_1 : Enable_{PC}, Memory_{read}
 t_2 : Load_{IR}
 t_3 : decode, Increment_{PC}
 t_4 : Load_W, Enable_{IR}, Enable_W, Memory_{read}
 t_5 : Enable_W, Memory_{read}, Load_{R0}

ADD R0, R1, R2: t_1 : Enable_{PC}, Memory_{read}
 t_2 : Load_{IR}
 t_3 : decode, Increment_{PC}
 t_4 : Enable_{R0}, Enable_{R1}, Add_{ALU}, Load_{R2}

STORE R2, 3000: t_1 : Enable_{PC}, Memory_{read}
 t_2 : Load_{IR}
 t_3 : decode, Increment_{PC}
 t_4 : Load_W, Enable_{IR}, Enable_W, Memory_{write}
 t_5 : Enable_W, Memory_{write}, Enable_{R2}