

Computer System Architecture

Mr. A. Victor Benevent Raj

B.Sc., B.Ed., M.C.A., M.Phil., (Ph.D)

Assistant Professor & Head

Department of Computer Applications

Ananda College, Devakottai

Tamil Nadu, India.

Lt. Dr. S. Kevin Andrews

B. Tech, M. Tech, Ph.D

Associate Professor, Deputy Dean - Activities, &

Associate NCC Officer (Army wing)

Department of Computer Applications

Dr. M. G. R Educational and Research Institute

Chennai. Tamil Nadu, India.



Computer System Architecture

Authors:

Mr. A. Victor Benevent Raj

Lt. Dr. S. Kevin Andrews

@ All rights reserved with the publisher.

First Published: January 2023

ISBN 978-93-92090-10-3



ISBN: 978-93-92090-10-3

DOI: <https://doi.org/10.47716/978-93-92090-10-3>

Pages: 178 (Front pages 12 & Inner pages 166)

Price: 350/-

Publisher:

Magestic Technology Solutions (P) Ltd.

544, Anna Street, Kathirvedu

Chennai, Tamil Nadu, India.

Website: www.magesticts.com

Email: info@magesticts.com

Imprint:

Magestic Technology Solutions (P) Ltd.

Chennai, Tamil Nadu, India.

Title Verso

Title of the Book:

Computer System Architecture

Author's Name:

Mr. A. Victor Benevent Raj

Lt. Dr. S. Kevin Andrews

Published By:

Magestic Technology Solutions (P) Ltd.

Publisher's Address:

544, Anna Street, Kathirvedu

Chennai 600 066. Tamil Nadu, India.

Printer's Details:

Magestic Technology Solutions (P) Ltd.

Edition Details: First Edition

ISBN: 978-93-92090-10-3

Copyright: @ Magestic Technology Solutions (P) Ltd.

Magestic Technology Solutions (P) Ltd

544, Anna Street, Kathirvedu

Chennai 600 066. Tamil Nadu. India

@ 2023, Magestic Technology Solutions (P) Ltd

Imprint, Magestic Technology Solutions (P) Ltd

Printed on acid-free paper

International Standard Book Number (ISBN): 978-93-92090-10-3

(Paperback)

Digital Object Identifier (DOI): 10.47716/978-93-92090-10-3

This book is a comprehensive guide that provides in-depth information on computer system architecture, drawn from reliable and authoritative sources. The author and publisher have made every effort to ensure the accuracy of the facts and information presented in this book, but cannot be held responsible for any errors or omissions.

Every effort has been made to trace and acknowledge the copyright holders of all material used in this book. If any copyrighted work has been inadvertently overlooked, please notify the publisher via email so that the necessary corrections can be made in future editions.

This book is protected by copyright laws and no part of it may be reproduced, transmitted, or used in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without the publisher's written permission.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks and are used only for identification and explanation without intent to infringe.

Visit the Magestic Technology Solutions (P) Ltd Web site at

<http://www.magesticts.com>

Acknowledgement

I express my heartfelt gratitude to my parents, **Mr. N. Amala Rajan** (Late), a National and State Awarded Teacher, and **Mrs. S. Catherine**, a retired teacher, my elder brother **Major A. Stanislaus Stephen**, and my lovable sisters, **Mrs. A. Cesiliya Sahayarani** and **Mrs. A. Teresa Selvarani**, for their constant support and inspiration in successfully completing this impressive book. I am also immensely grateful to my ever-loving wife, **Er. N. Angeline Matharasi**, for her unwavering support.

I sincerely thank my research supervisor, **Lt. Dr. S. Kevin Andrews**, Deputy Dean, and Associate Professor, Dr. M. G. R Educational & Research Institute, Chennai, for giving me this golden opportunity and his continuous support and encouragement throughout the book's completion.

I would also like to thank **Rev. Fr. Jesuraj. K Christy**, Secretary, **Rev. Dr. S. John Vasantha Kumar**, Principal, **Rev. Fr. C. George Fernandes**, Vice-Principal, **Rev. Fr. J. Densingh Rajan**, IQAC Head and Coordinator, **Rev. Fr. G Alexandar Narkunam**, Campus Minister of our ANANDA College, Devakottai, and my department colleagues for their assistance in completing the book efficiently within the limited time frame.

Finally, I am grateful to my friends who have been with me throughout my life. Thank you for your support and encouragement.

This Page Intentionally Left Blank

Preface

"Computer System Architecture" is an essential guide for anyone interested in understanding the inner workings of computer systems. The book is designed to provide a comprehensive and in-depth examination of the key concepts and principles of computer system architecture, from data representation and basic computer organization to memory organization and multiprocessors.

The book is divided into five units, each of which covers a different aspect of computer system architecture. Unit I introduces the reader to the basics of data representation, including data types, complements, and register transfer language. Unit II delves deeper into memory reference instructions and input-output and interrupt systems. Unit III covers the central processing unit, including its organization, instruction formats, addressing modes, and data transfer and manipulation. Unit IV explores the intricacies of computer arithmetic, including addition and subtraction, multiplication algorithms, and division algorithms. And Unit V concludes the book with a detailed examination of memory organization, including memory hierarchy, main memory, auxiliary memory, associative memory, cache memory, virtual memory, and the characteristics of multiprocessors.

This book is intended for students and professionals in the field of computer science and engineering, as well as anyone interested in understanding the inner workings of computer systems. The book provides a thorough and in-depth examination of the key concepts and principles of computer system architecture, making it an essential reference guide for anyone looking to gain a deep understanding of this complex and ever-evolving field.

We hope that this book will serve as a valuable resource for anyone interested in computer system architecture and will help readers to gain a greater understanding of the inner workings of computer systems.

Authors

Mr. A. Victor Benevent Raj

Lt. Dr. S. Kevin Andrews

This Page Intentionally Left Blank

SYLLABUS
III YEAR – V SEMESTER
COURSE CODE: 7BCA5C2

CORE COURSE - XI – COMPUTER SYSTEM ARCHITECTURE

Unit I

Data Representation: Data types, Complements, Register Transfer Language, Register Transfer Bus and Memory Transfers, Arithmetic, Logic and Shift unit. Introduction to Basic computer organization and design: Instruction codes, computer registers, Computer Instructions, Timing and control, Instruction cycle.

Unit II

Memory reference instructions: Input – Output and Interrupt. Introduction to programming the basic computer: Machine Language, Assembly Language, The assembler, Program Loops, Programming Arithmetic and Logic operations and Subroutines.

Unit III

Central Processing Unit: Introduction, General register Organization, Stack Organization, Instruction formats, addressing modes, data transfer and Manipulation, and Reduced Instruction Set Computer (RISC).

Unit IV

Introduction to computer Arithmetic: Addition and Subtraction, Multiplication algorithms, Division Algorithms, Input – Output Interface, priority Interrupt – Direct Memory Access, Input-Output Processor.

Unit V

Memory Organization: Memory Hierarchy, Main memory, Auxiliary memory, Associative memory, Cache memory, Virtual memory. Characteristics of multiprocessors.

Text Book:

1. Computer System Architecture, M.Morris Mano, PHI Pvt. Ltd. Third Edition, 2013.

Book for Reference:

1. Modern Computer Architecture, Mohammed Rafiquzzaman, and Rajan Chandra, Galgotia Publications Pvt. Ltd.

Table of Contents

	Page. No
I. Data Representation	
A. Data Types.....	3
B. Complements.....	4
C. Register Transfer Language.....	6
D. Register Transfer Bus and Memory Transfers	7
E. Arithmetic, Logic and Shift unit.....	12
F. Basic computer organization and design.....	13
1. Instruction codes.....	15
2. Computer registers.....	16
3. Computer Instructions.....	18
4. Timing and control.....	20
5. Instruction cycle.....	21
II. Memory reference instructions	
A. Input-Output and Interrupt.....	31
B. Programming the basic computer.....	38
1. Machine Language.....	38
2. Assembly Language.....	39
3. The Assembler.....	46
4. Program Loops.....	54
5. Programming Arithmetic and Logic operations.....	57
6. Subroutines.....	64
III. Central Processing Unit	
A. Introduction.....	77
B. Stack Organization.....	78
C. General register Organization.....	85
D. Instruction formats.....	83
E. Addressing modes.....	87
F. Data transfer and Manipulation.....	90

IV. Computer Arithmetic	
A. Addition and Subtraction.....	102
B. Multiplication algorithms.....	109
C. Division Algorithms.....	116
D. Input-Output Interface.....	121
E. Priority Interrupt-Direct Memory Access.....	127
F. Input-Output Processor.....	140
V. Memory Organization	
A. Memory Hierarchy.....	147
B. Main memory.....	149
C. Auxiliary memory.....	152
D. Associative memory.....	155
E. Cache memory.....	157
F. Virtual memory.....	159
G. Characteristics of multiprocessors.....	161
References	
Bibliography.....	165
Webliography.....	165

Unit I - Data Representation

Data types, Complements, Register Transfer Language, BUS and Memory Transfers, Arithmetic, Logic and Shift unit. Introduction to Basic computer organization and design: Instruction codes, computer registers, Computer Instructions, Timing and control, Instruction cycle.

This Page Intentionally Left Blank

1 Data Types

1.1. Computer data types

Computer programs or application may use different types of data based on the problem or requirement.

Given below is different types of data that computer uses:

Numeric data – Integer and Real numbers

Non-numeric data – Character data, address data, logical data

Let's study about each with further sub-categories.

1.1.1 Numeric data

It can be of the following two types:

Integers

1.1.2. Real Numbers

Real numbers can be represented as:

Fixed point representation

Floating point representation

1.1.3. Character data

A sequence of character is called character data.

A character may be alphabetic (A-Z or a-z), numeric (0-9), special character (+, #, *, @, etc.) or combination of all of these. A character is represented by group of bits.

When set of multiple character are combined together, they form a meaningful data. A character is represented in standard ASCII format.

Another popular format is EBCDIC used in large computer systems.

Example of character data

Rajneesh1#

229/3, xyZ

Mission Milap – X/10

1.1.4. Logical data

Computer systems employ logical data to make logical judgements.

Logical data differs from numeric or alphabetic data in that numeric and alphabetic data may be connected with numbers or letters, but logical data is represented by either true (T) or false (F).

Logic gates provide an illustration of logical data via the development of truth tables.

A statement including numeric or character data with relational symbols (>, =, etc.) may also be considered logical data.

1.1.5 Character set

In computers, character sets may be any one of the following types:

Characters from the alphabet It is made up of characters from the alphabet, either A-Z or a-z.

Characters that represent numbers It is made up of digits ranging from 0 to 9.

Characters with special meanings +, *, /, -, .., <, >, =, @, %, #, etc.+ , * , / , - , .. , < , > , = , @ , % , # , etc.etc. are examples of symbols with special meanings.

1.2 Complements

In digital computers, complements are employed for the purpose of making the subtraction operation more straightforward and for logical manipulation. Each base r system may be complemented using either the r 's complement or the $(r - 1)$'s complement. These are the two different techniques.

($r - 1$)'s Complement

1.2.1. 9's complement

If we have a number written in base r with n digits, then the complement of that number, written as $(r^n - 1)$ minus N , may be expressed as follows: Since $r = 10$ for decimal numbers and $r - 1 = 9$, the complement of N in base 9 is denoted by the expression $(10^n - 1) - N$.

Now, a number is said to have the value 10^n if it begins with a single 1 and is then followed by n 0s. n 9s are used to define the number 10^n minus 1. When n is equal to four, for instance, we get that $10^4 = 10000$ and $10^4 - 1 = 9999$. The conclusion that can be drawn from this is that the 9's complement of a decimal number may be obtained by first 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 $99999 - 12389 = 87610$.

1.2.2. 1's complement

Since $r = 2$ for binary integers and $r - 1 = 1$, the complement of N is $(2^n - 1)$ minus N since $r = 2$ and $r - 1 = 1$. 2^n is represented as a binary number

that begins with a 1, then continues with n 0s after it. $2^n - 1$ is a binary number that may be represented by n 1s. For instance, if we have n equal to 4, we get $2^4 = (10000)_2$ and $2^4 - 1 = (1111)_2$ in our calculations. Therefore, to get the 1's complement of a binary integer, one must first subtract each digit from 1, as shown. The bit that causes the transition from 0 to 1 or from 1 to 0 is produced when a binary digit is subtracted from the value 1. Because of this, the structure of the 1's complement of a binary number is constructed by changing the 1's into 0's and the 0's into 1's respectively. The 1's complement of the number 1011001, for instance, is 0100110, whereas the 1's complement of the number 0001111 is 1110000. When working with octal or hexadecimal integers, the $(r - 1)$'s complement may be derived by first subtracting each digit from 7, or F (decimal 15), whichever comes first.

1.2.3. 10's complement

The representation of the r 's complement of an n -digit integer in base r is $r^n - N$ for $N > 0$ and 0 for $N = 0$. This applies only when N is not equal to 0. When compared to the complement of $(r - 1)$, we can see that the complement of r is obtained by adding 1 to the complement of $(r - 1)$ since $r^n - N = [(r^n - 1) - N] + 1$. This is due to the fact that $r^n - N = [(r^n - 1) - N] + 1$.

Therefore, the value of the 10's complement of the decimal 2389 is 7610 plus 1 (which equals 7611), and this value may be obtained by adding 1 to the value of the 9's complement. The 2's complement of the binary number 101100 is 010100, which can be obtained by adding 1 to the 1's complement value, which results in the value 010011.

1.2.4. 2's complement

It is possible to make the 2's complement by first preserving the least significant bits, which consist of all 0's and the initial 1, and then restoring the most significant bits, which consist of all 1's and 0's, respectively. 1101100 has a 2's complement of 0010100, which can be obtained by maintaining the original values of the first bit, the two lowest-order 0's, and the first bit, and then replacing all of the 1's in the other four most important bits with 0's and all of the 1's with 0's.

1.3 Register Transfer Language

It is possible to describe the order of the microoperations that take place in a computer by describing each and every operation in words. On the other hand, we don't employ this approach since it involves a lot of steps. As a result, we make use of certain symbols in order to express the order in which registers are transferred from one another and the microoperations that are linked with the transfers. The micro-operation sequences in registers and the control functions that begin them may be represented in a clear and succinct manner using symbols, which is a simple and straightforward method. A register transfer language is the name given to this kind of symbolic notation. The listing of micro-operation sequences among the registers of a digital computer system is accomplished with the help of this language. In addition to this, it plays the role of a facilitator throughout the design process.

In order to indicate the purpose of a computer register, we give the register a name that begins with a capital letter. For instance, the register that stores an address for the memory unit is often referred to as a memory address register (or MAR for short), and it is represented by the symbol PC, which stands for "programme counter," IR, which stands for "instruction register," and R1 are some more examples (for processor register). To demonstrate the individual flip-flops included inside an n-bit register, we assign them numbers in the order 0 through n minus 1, beginning with 0 at the position farthest to the right and progressing leftward as we go toward the register's left side.

A 16-bit register is split in two equal halves called halves. The low byte, which consists of bits 0 through 7, is represented by the symbol L, while the high byte, which consists of bits 8 through 15, is represented by the symbol H. PC is the name of a register that has a capacity of 16 bits. The low order byte is represented by the symbol PC(L), whereas the high order byte is designated by the symbol PC(H).

When you see the expression "R2 \leftarrow R1," what it means is that the information from register R1 has been moved into register R2. It is important to notice that the information that is stored in the source register R1 does not alter after the transfer has taken place. In practical situations,

the flow of information can only take place when certain control conditions are met. One way to demonstrate this is by the use of a "if-then" statement:

If $P = 1$, then $R2 \leftarrow R1$

where P is a control signal that is produced in the component of the system that is responsible for control. By declaring a control function, we make it easier for ourselves to keep track of the control variables independently from the register transfer process. A Boolean variable that may take on the values 1 or 0 is an example of a control function. The control function reads as follows when typed out: $P: R2 \leftarrow R1$

1.4 Bus

Because a computer has so many registers, there has to be a way to get information from one register to another. If separate lines are utilised between each register and all of the other registers in the system, the total number of wires in the system will be excessive. A common bus system is recommended as the method of choice for transmitting information across registers in multiple-register configurations because of its superior efficacy. A bus architecture is made up of a collection of shared lines, one for each bit of a register. It is along these lines that binary information is moved one bit at a time. During each given register transfer, the register that is used for the transaction is decided by the bus based on the control signals.

It is possible to construct a network that is based on a shared bus infrastructure by utilising multiplexers. The source register is selected by these multiplexers, and once that decision has been made, the binary data stored in that register is then transferred onto the bus. A bus system would multiplex registers, each of which only carries a single bit, in order to create an n -line common bus. This would allow the system to store more information. When building the bus, the number of multiplexers required is equal to n , where n is the number of bits that are included in each register. This indicates that the bus can be constructed with no more than that number of multiplexers. Because it multiplexes k data lines, the size of each multiplexer needs to be at least k over 1 in order to meet the

bare minimum requirements. The development of a bus system may also make use of "three-state gates" rather than multiplexers as an alternative to the use of multiplexers. The name "three-state gate" refers to the name of a certain type of digital circuit that may display three distinct states. The logic values one and zero can be thought of as equivalent to two of the states.

A third potential scenario involves a state characterised by a high impedance. The situation of high impedance operates in the same manner as an open circuit, which shows that the output is not linked and does not have any significance from a logical standpoint. This is because high impedance behaves in the same way as an open circuit. Throughout the process of designing a bus system, the component known as the buffer gate is the one that is utilised the most of the time. An illustration of a visual symbol for a three-state buffer gate shown in Figure 1.1. You can view this illustration here. The control input has a decisive impact on the output in every possible way.

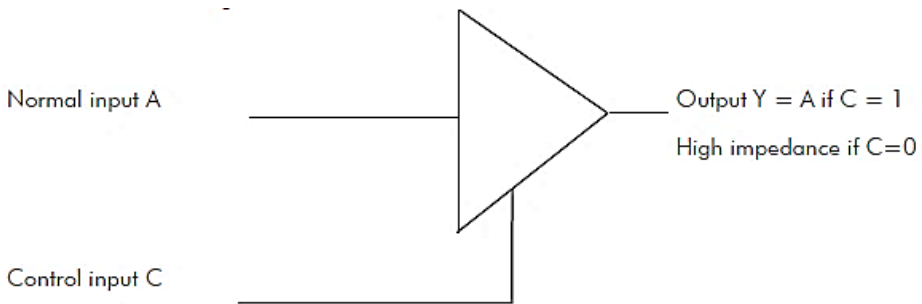


Fig.1.1 An illustration of a visual symbol for a three-state buffer gate

The construction of a bus system for four registers is shown below in Figure 1.2.

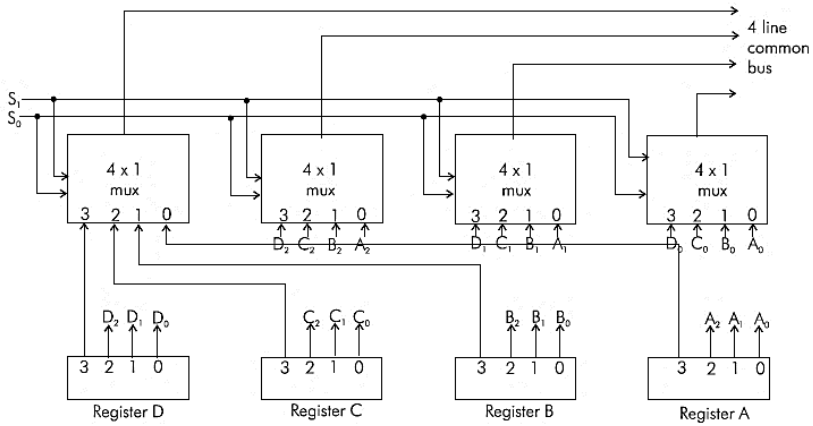


Fig. 1.2 Construction of a bus system for four registers

The function table of the above bus system is

S ₁	S ₀	Register collected
0	0	A
0	1	B
1	0	C
1	1	D

The construction of a bus system with three state table buffers is shown in the following figure 1.3:

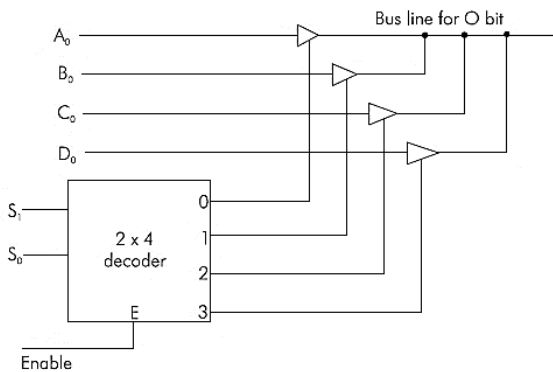


Fig. 1.3 Construction of a bus system with three state table buffers

1.5 Memory Transfer

A write operation is described as the process of storing information in memory, whereas a read operation refers to the process of extracting information from a memory word and transferring it to the external environment. A memory word is represented by the letter M, and it is chosen by the memory address while the transfer is taking place. The memory address is a specification for transfer operations. After the letter M, the address is supplied by enclosing it in square brackets as seen in the example.

For instance, an illustration of the read operation for the transfer of a memory unit M from an address register AR to another data register DR could look something like this:

Read: $DR \leftarrow M[AR]$

A micro-operation is a fundamental operation that is carried out on the information that has been saved in registers. The micro-operations can be broken down into the following four categories:

1. Register transfer is the process of moving binary information from one register to another.
2. Arithmetic: Carry out arithmetic operations on the numeric data that has been saved in registers.
3. Perform a bit manipulation operation on non-numerical data that is stored in registers using logic.
4. Shift — This operation allows you to perform shift operations on the data that is stored in registers.

The information is moved from one register to another by means of these micro-operations. During the course of this micro-operation, it is important to remember that the information does not undergo any changes. One possible design for a register transferred micro-operation is the following:

$R1 \leftarrow R2$

This suggests that the information found in register R2 should be moved to register R1. Because we anticipate that the register transfer will take place in a situation of control that has already been determined, the destination register ought to be capable of parallel loading.

Connecting all of the processor's registers is accomplished through the use of its standard internal data bus. The number of bits included within a general register should, as a rule, be proportional to the size of this data bus. There are certain transfers that are performed, not through the internal data bus, but rather through the system bus. These transfers take place. Memory and input/output modules are connected to these data flows in some way. In addition, the input/output process is handled as its own distinct activity, which is generally comprised of the execution of a programme and its associated instructions. Memory transfer is regarded as the most significant transfer for the execution of instructions because it occurs at least once for each and every instruction.

A system bus is capable of facilitating the transfer of memory. Because the main memory is a random-access memory, the address of the location that is to be used must be provided. This must be done because the address cannot be changed once it has been set. On the address bus, the central processing unit (CPU) provides this address. Read and write are the two operations that can be performed while transferring memory.

1.5.1 Memory Read

1. Enter the memory address into the register for the memory address (MAR).
2. Examine the information pertaining to the place. In most cases, this is accomplished by putting the data in MAR on the address bus in conjunction with a memory read control signal that is placed on the control bus. The information that was read from memory is then sent to the data bus, which, in turn, puts the information in the data register (DR). This whole process has the potential to be shown as:

$$DR \leftarrow M[MAR]$$

1.5.2 Memory Write

1. Place the desired memory address in the memory address database (MAR) and the data written in the data database (DR).
2. Write data in a location: MAR sends addresses to address buses, and DR sends data to data buses to write them to the memory address of MAR.

$$M [\text{MAR}] \leftarrow \text{DR}$$

Normally, memory reading and writing operations require more clock cycles than registering operations.

1.6 Arithmetic Micro operations

These micro-operations carry out certain fundamental arithmetic operations on the numeric data that has been saved in the registers. These fundamental procedures might include things like adding, subtracting, incrementing, or decrementing a number, or even performing shift operations in arithmetic. A micro-operation that can be expressed as a "add" can be as follows:

$$R3 \leftarrow R1 + R2$$

It suggests that the contents of registers R1 and R2 should be added together, and the result should be saved in register R3.

In order to perform the addition operation, the ALU needs to be equipped with three registers in addition to the addition circuit.

The operation of subtraction is carried out using the complement and the addition processes as:

$$R3 \leftarrow R1 - R2 \text{ is implemented as}$$

$$R3 \leftarrow R1 + (2\text{'s complement of } R2)$$

$$R3 \leftarrow R1 + (1\text{'s complement of } R2 + 1)$$

$$R3 \leftarrow R1 + R2 + 1$$

An increase operation can be represented as follows.

$$R1 \leftarrow R1 + 1$$

while a decrement operation can be symbolized as:

$$R1 \leftarrow R1 - 1$$

By utilising a combinational circuit or binary up/down counters, we are able to perform operations such as incrementing and decrementing. Multiplication and division are typically implemented in computers through the use of add/subtract and shift micro-operations. If a digital system has implemented division and multiplication using combinational circuits, then we can refer to these as the micro-operations for that system. This is because combinational circuits are a subset of combinational logic. The implementation of an arithmetic circuit typically involves the use of

parallel adder circuits. The presented circuit has multiplexers (MUX), and each one of those MUXs has two select inputs. This 4-bit circuit receives as input two 4-bit data values and a carry-in bit, and it produces as output the four data bits that come from combining those values along with a carry-out bit. We are able to obtain a wide variety of microoperations by using a variety of input values.

Equivalent micro-operation Micro-operation name

$R \leftarrow R1 + R2$ Add

$R \leftarrow R1 + R2 + 1$ Add with carry

$R \leftarrow R1 - R2$ Subtract with borrow

$R \leftarrow R1 - 2$'s Subtract

$R \leftarrow R1$ Transfer

$R \leftarrow R1 + 1$ Increment

$R \leftarrow R1 - 1$ Decrement

These operations are performed with binary data stored in the database. In logic micro operations, each register bit is treated as an independent variable.

For example, if R1 and R2 are 8 bits registers and

R1 contains 10010011 and

R2 contains 01010101

R1 AND R2 00010001

AND, OR, NOT, and complements are examples of common microoperations that are used in logic. Only OR, NOR, and NAND are allowed. There are four distinct permutations of the input for two variables that are feasible. The numbers in question are 00, 01, 10, and 11. Now, for each and every one of these four-input combination, a function can have $2^4 = 16$ different output combinations. This suggests that for every two variables, we have the potential for 16 different logical procedures.

1.7 Shift Microoperations

For the purpose of serial data transfer, shift microoperation can be utilised. In most cases, arithmetic, logic, and several other data processing activities will require their utilisation. A register's contents have the capability of being moved to either the left or the right. Whenever a shift-

right operation is being performed, the serial input will move one bit to the place that is furthest to the left. Whenever a shift-left operation is performed, the serial input will move one bit to the most extreme right position. Logical, circular, and arithmetic shifts are the three categories of shifts that exist.

Logical Shift

The logical transfer operation transfers 0 to the serial input. We use `shl` and `shr` symbols to move logic left and right microprocesses., e.g.

`R1 ← shl R1`

`R2 ← shr R2`

The two microoperations specified are the left-side one-bit shift of the R1 registry and the right-side one-bit shift of the R2 registry.

1.7.1 Circular Shift

The term "rotation operation" can also be used to refer to the circular shift. The bits of the register are passed around the two ends of the device in such a way that there is no loss of information. This can be done by making a connection from the shift register's serial output to the serial input of the register itself. The circular shift left and circular shift right are represented by the symbols `cil` and `cir`, respectively. For example, if the value 01101101 is stored in the Q1 register. After the `cir` operation, the value will change to 0110110, and after the `cil` operation, it will contain 11011010.

1.7.2 Arithmetic Shift

A signed binary number can be shifted to the left or right using a micro-operation known as arithmetic shift. The result of performing an arithmetic shift left operation on a binary value is the same as multiplying it by 2. In a same manner, an arithmetic shift to the right will divide the integer by 2. When the number is multiplied or divided by 2, the sign bit must be left unaltered by the arithmetic shift-right operation since the sign of the number must remain the same. The bit that is furthest to the left in a register is known as the sign bit, and the bits that follow it are used to store the number. For positive values, the sign bit is set to 0, and for negative values, it is set to 1. The form of negative numbers is known as

2's complement. The following diagram illustrates an ordinary register that contains n bits.

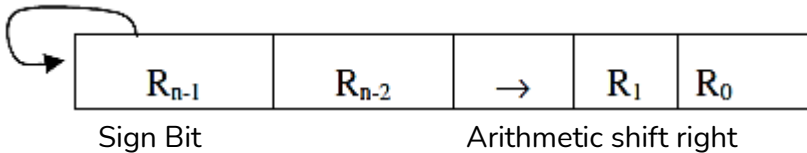


Fig. 1.4 Shift Microoperations

1.7.3 Instruction Codes

An instruction code is a set of bits that gives the computer instructions to carry out a certain action. The operation element of an instruction code is the most fundamental component of the code. An addition, a subtraction, a multiplication, a shift, or a complement are examples of the kinds of operations that are defined by the operation code of an instruction. The entire number of operations that can be performed by the computer is what defines the number of bits that are necessary for an instruction's operation code. For a set of 2^n or fewer different operations, the operation code needs to have at least n bits in order to function properly. The term "operation" refers to a binary code that gives the computer instructions to carry out a certain operation. The instruction is read from memory by the control unit, which also provides an interpretation of the operation code bits. After then, it sends out a series of control signals to kick off some micro-operations in the computer's internal registers. A series of microoperations will be issued by the control for each operation code. These microoperations are necessary for the hardware execution of the operation that has been described.

Either the data that is stored in the processor registers or the data that is stored in the memory should be used as the basis for this operation. As a result, the operation and the registers or memory words where the operands are to be found, as well as the registers or memory words where the operands are to be saved, must be specified in an instruction code in order for it to be valid. Instruction codes allow for the addressing of memory words to be used for this purpose. It is possible to specify the registers of a processor by providing additional binary code consisting of

K bits to the instruction. This code can specify any one of 2K registers. There are a lot of different ways to organise the binary code that instructions are written in. Each type of computer has what is known as its "Instruction Set," which is a unique instruction code format.

1.8 Computer Registers

In most cases, the instructions for a computer are saved in memory regions that are consecutive. These instructions are carried out in the order given, one after the other. The control reads an instruction from a certain location in memory during one operation, then puts that instruction into action, and the procedure is then repeated. A counter is required for this instruction sequencing so that the address of the subsequent instruction can be calculated after each execution. After the instruction code is read from memory, the control unit must also have a register for storing it after it has been read. This register is required to be present. For the computer to be able to manipulate data, it requires a processor register, as well as a register that can hold an address of memory. The following table provides a summary of these criteria, along with a concise explanation of their respective roles and the number of bits that they comprise.

Table: 1 List of Registers for the Basic Computer

Register Symbol	Number of bits	Register name	Function
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

The breadth of a memory address is 12 bits, which is why the memory address register (AR) consists of that many bits. In addition, there are 12 bits in the programme counter (PC). The address of the next instruction

that will be read from memory is stored in the PC. This address is used after the current instruction has been carried out. There is a counting sequence that the PC goes through. This causes the computer to read sequence, and it also causes the computer to read instructions that were previously stored in memory that were sequential.

Two registers are utilised for input and output respectively. An 8-bit character is delivered by an input device to the Input register, abbreviated as INPR. An 8-bit character for an output device is stored in the Output register (OUTR) of the computer.

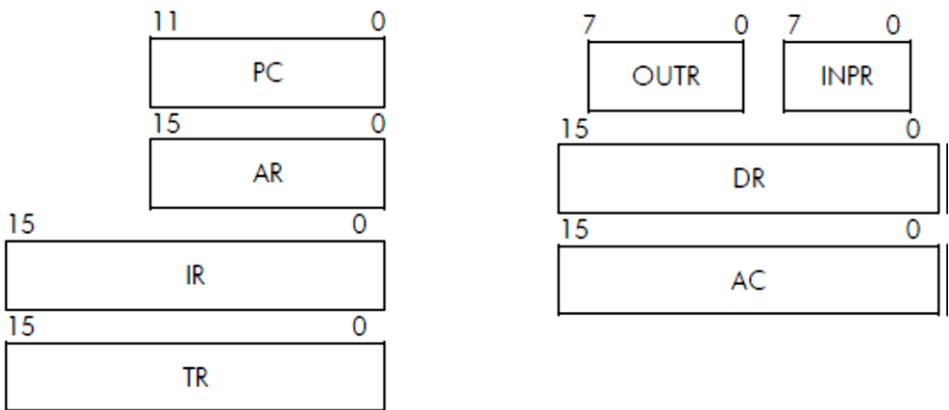


Fig.1.5 Computer Register Operations

1.9 Common Bus System

The most fundamental type of computer has eight registers, in addition to a memory unit and a control unit. Paths need to be made available in order for information to be moved from one register to another as well as between registers and memory. If connections are formed between the outputs of each register and the inputs of the other registers, then the total number of wires that are needed will become excessive. When there are a lot of registers in a system, using a common bus to transport information is the most effective and efficient method to utilise. In a previous chapter, we demonstrated how to design a bus system by making use of multiplexers (MUXs) or three-state buffer gates.

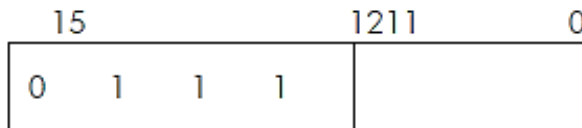
1.9.1 Computer Instructions

As may be seen in the image that follows, the basic computer makes use of three different instruction code forms. Every format consists of 16 bits. There are three bits that make up the operation code, also known as the opcode. The meaning of the remaining 13 bits is determined by the operation code. A memory reference instruction will use 12 bits to define an address, and it will use the remaining bit, known as I, to declare the addressing mode, which will be either 0 or 1, depending on whether the address will be direct or indirect. The register reference instructions are identified by the operation code 111, which contains a zero in the bit that is furthest to the left (bit15).

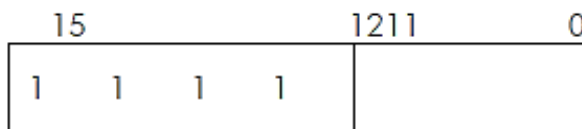
An operation or a test that is performed on the AC register can be specified using a register reference instruction.



Memory-reference instruction



(b) Register-reference instruction



(c) Input Output instruction

Fig. 1.6 Basic Computer Instruction Format

Internal Memory or Primary Memory –

Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

We can infer the following characteristics of Memory Hierarchy Design from above figure:

5.1.3 Capacity:

It is a global amount of information stored in memory. As we move from top to bottom in the Hierarchy, the capacity increases.

5.1. Access Time:

It is a time interval between a reading/writing request and data availability. As you move from top to bottom of the hierarchy, the time you have to access increases.

5.1.5 Performance:

When the computer system was first designed, it did not have a Memory Hierarchy design. As a result, there was a big disparity in the amount of time it took to access the CPU registers and the main memory. This caused a speed gap. Because of this, the performance of the system is decreased, and as a result, augmentation was necessary. This improvement was implemented in the form of a Memory Hierarchy Design, and as a result, the overall performance of the system has been improved. Minimizing the depth to which one must travel in the memory hierarchy in order to manipulate data is one of the most important things that can be done to improve the speed of a system.

5.1.6 Cost per bit:

The cost per bit rises as we progress up the Hierarchy from lowest level to highest level; hence, the cost of Internal Memory is higher than the cost of External Memory.

The memory hierarchy system includes all of the storage devices that are used in a computer system, beginning with the auxiliary memory, which is relatively slow but has a large capacity, and progressing to the main memory, which is relatively faster, and finally to the cache memory, which is even more compact but even more quickly accessible to the high-speed processing logic.

- Main Memory is a type of memory that is in direct communication with the central processing unit (RAM)
 - Auxiliary Memory, a term for any device that offers secondary storage (Disk Drives)
 - Cache Memory is a specialised form of memory that operates at a very fast speed and is used to accelerate the processing speed (Cache RAM).
- The comparatively sluggish magnetic tapes that are used to store removable files are located at the very bottom of the hierarchy. The next type of storage medium is magnetic discs, which are utilised as backups. Because it is able to interface directly with both the central processing unit (CPU) and with auxiliary memory devices through an input/output process, the main memory holds a pivotal place in the system. A programme that is not now required in main memory is moved to an auxiliary memory location in order to make room for programmes and data that are currently being used. The cache memory is responsible for storing sections of the programmes that are being run by the CPU at any given moment. The I/O processor is responsible for managing the flow of data between the main memory and auxiliary memory. When compared to main memory, the access speed of the auxiliary memory is much slower; yet, it has a big storage capacity and is relatively inexpensive. The cache memory takes up very little space, is very expensive, and provides exceptionally quick access. The central processing unit (CPU) has direct access to the cache as well as the main memory, but it does not have access to the auxiliary memory.

5.2 Main Memory

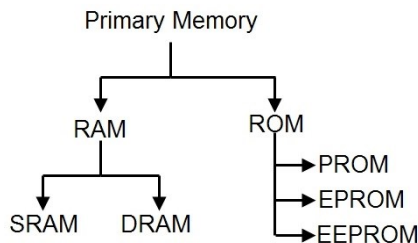


Fig.5.2 Main Memory Hierarchy

Memory that is directly accessible by the central processing unit is referred to as main memory. This memory is sometimes referred to as primary memory or RAM (Random Access Memory) (Central Processing Unit). Its primary function is to store data and instructions that the central processing unit (CPU) must have quick access to in order to carry out its activities.

In terms of how the memory is structured, the main memory is divided up into a number of memory cells, and each memory cell has the capacity to hold a single byte of information. These cells are numbered in order, beginning at a base address, and the central processing unit is able to access them by utilising their respective memory addresses. The amount of main memory that a computer has is measured in bytes, and most current computers have several gigabytes of memory. Bytes are the unit of measurement for computer memory.

The computer's main memory is volatile, which means that any data stored in it will be lost if the power is turned off. In contrast to primary memory, secondary memory, such as a hard drive, can keep data even when the power is turned off. Because of this, the main memory is used to store the data that the computer is presently working with, while the secondary memory is used to store the data that is not immediately required.

The speed of the main memory is an important property to consider. The data that is stored in main memory can be accessed by the CPU in just a few nanoseconds, which is far faster than accessing data that is stored in secondary memory. Because the central processing unit (CPU) needs to be able to swiftly access data in order to carry out its activities, this is absolutely necessary for the effective running of a computer.

In a nutshell, main memory is the primary memory that is directly accessible by the central processing unit (CPU) of a computer. Its primary function is to store data and instructions that the central processing unit (CPU) must have quick access to in order to carry out its activities. It can change quickly, it is measured in bytes most of the time, and it is volatile. **RAM** (Random Access Memory) is a type of computer memory that is used to store data and instructions that the CPU needs to access quickly in

order to perform its operations. RAM is volatile, meaning that it is wiped clean when the computer is powered off. There are two types of RAM:

- **SRAM** (Static Random Access Memory) is a type of RAM that retains its data as long as power is supplied. It is faster and more expensive than DRAM.
- **DRAM** (Dynamic Random Access Memory) is a type of RAM that requires a constant refresh to maintain its data. It is slower and less expensive than SRAM.

ROM (Read-Only Memory) is a type of computer memory that is used to store data that is not intended to be modified. ROM is non-volatile, meaning that it retains its data even when the computer is powered off.

There are several types of ROM:

- **PROM** (Programmable Read-Only Memory) is a type of ROM that can be programmed only once by the user and then becomes read-only.
- **EPROM** (Erasable Programmable Read-Only Memory) is a type of ROM that can be reprogrammed by exposing it to ultraviolet light.
- **EEPROM** (Electrically Erasable Programmable Read-Only Memory) is a type of ROM that can be reprogrammed by applying an electrical charge.

In summary, RAM is a volatile memory that is used to store data and instructions that the CPU needs to access quickly, there are two types of RAM: SRAM which is faster and more expensive and DRAM which is slower and less expensive. ROM is a non-volatile memory that is used to store data that is not intended to be modified, there are several types of ROM: PROM which can be programmed only once by the user and then becomes read-only, EPROM which can be reprogrammed by exposing it to ultraviolet light and EEPROM which can be reprogrammed by applying an electrical charge.

5.3 Auxiliary memory

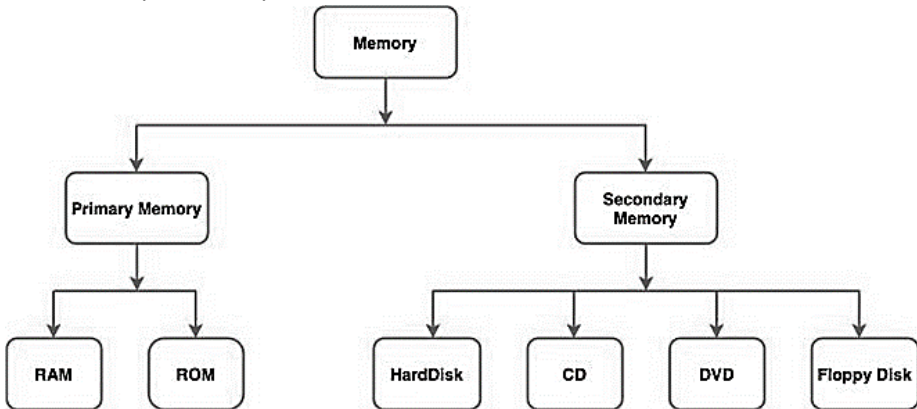


Fig. Auxiliary Memory Hierarchy

- Auxiliary memory, also known as secondary memory or non-volatile memory, is a type of computer memory that is used to store data and instructions that are not currently being used by the CPU. It is distinguished from main memory by the fact that it does not lose its contents when the computer is turned off. It is separate from main memory, which is used to store data and instructions that the CPU is actively utilising. Main memory is used to store data and instructions that the CPU is actively using.
- Auxiliary memory is normally organised in the same way as main memory, which is to say that it is comprised of a succession of memory cells. On the other hand, the cells that make up auxiliary memory are often much larger than those that make up main memory, which allows them to store significantly more information. The cells that make up auxiliary memory are often arranged in blocks as well, which allows for the cells to be read from and written to collectively.
- Auxiliary memory is non-volatile, which means that it stores data even after the computer has been turned off and the power is turned back on. This is in contrast to the computer's main memory, which is cleared once the power is turned off to the device.

Auxiliary memory is utilised as a result of this so that data can be saved in the event that the computer needs to retrieve it at a later time, even if the computer is shut off.

- Hard drives, solid-state drives, and flash drives are a few examples of the types of memory that fall under the category of auxiliary memory. It can take the central processing unit tens of milliseconds or even longer to retrieve data that is stored on these forms of memory, which is significantly slower than main memory. This is due to the fact that the data that is kept in auxiliary memory is saved on physical media, such as magnetic discs or flash memory chips, both of which require some amount of time in order to access.
- In conclusion, Auxiliary memory is a sort of computer memory that is used to store data and instructions that are not currently being used by the CPU. It is also known as secondary memory and non-volatile memory. Auxiliary memory can go by several other names. Although it operates more slowly than main memory, it is nevertheless capable of storing data even when the computer is turned off. Storage devices such as hard drives, solid-state drives, and flash drives are all examples of auxiliary memory. The information that is kept in auxiliary memory is kept on physical media, such as magnetic discs or chips containing flash memory.
- Auxiliary memory, also known as secondary memory or non-volatile memory, is one of several types of memory that can be used to store data and instructions that the central processing unit (CPU) is not actively using at the moment. There are numerous different types of auxiliary memory. These varieties of memory access data at a slower rate than main memory, but they keep their contents even when the computer is turned off. The following are some instances of auxiliary memory.
- **Hard Disk Drive (HDD):** This is the traditional type of auxiliary memory, which stores data on rapidly spinning disks. They are relatively cheap, but also relatively slow and can be affected by mechanical failure.

- **Solid State Drive (SSD):** This is a newer type of auxiliary memory that stores data on flash memory chips. They are faster and more reliable than HDDs, but also more expensive.
- **Flash Drive:** Also known as USB drive, thumb drive or pen drive, this type of auxiliary memory stores data on flash memory chips and can be easily plugged into a computer's USB port. They are small, portable, and relatively inexpensive, but also have limited storage capacity.
- **CD-ROM, DVD-ROM:** These are types of auxiliary memory that store data on a compact disk, they can be read by the computer's CD/DVD drive. They are relatively cheap and can store a large amount of data, but also relatively slow and can be easily damaged.
- **Tape Drive:** This type of auxiliary memory stores data on a magnetic tape, it is mainly used for backup and archiving of large amount of data. They are relatively inexpensive and can store a large amount of data, but also relatively slow and can be affected by mechanical failure.
- **Cloud storage:** This type of auxiliary memory stores data on remote servers that can be accessed over the internet. They are relatively inexpensive, have large storage capacity, and can be accessed from anywhere with an internet connection, but may have security and privacy concerns.

In summary, Auxiliary memory, also known as secondary memory or non-volatile memory, are slower than main memory but retain data even when the computer is powered off. There are several types of auxiliary memory such as Hard Disk Drive, Solid State Drive, Flash Drive, CD-ROM, DVD-ROM, Tape Drive, and Cloud storage. Each of them has their own advantages and disadvantages such as storage capacity, cost, speed, and reliability.

5.4 Associative Memory

Associative memory, which is also known as content-addressable memory (CAM), is a sort of computer memory that is used to locate data based on its content rather than its memory address. This type of memory allows for the data to be located fast and effectively. In contrast to this is conventional memory, which, in order to be accessed, must have associated memory addresses.

When it comes to the way in which memories are structured, associative memory is often arranged into a series of memory cells, similar to the way that main memory and auxiliary memory are. On the other hand, each cell in associative memory stores not just a value but also a key that may be used to locate the value. The value can then be retrieved. When a search is carried out, the key is used to do a comparison against the keys that are stored in the memory cells, and if a match is discovered, the value that corresponds to that match is returned.

The usage of associative memory is especially beneficial in circumstances in which it is necessary to rapidly retrieve data based on the data's content rather than on the data's memory address. This has potential utility in a diverse range of applications, including data compression and networking, for example. For instance, in the field of networking, associative memory can be used to do a rapid lookup based on the source address of a packet in order to determine its destination address. When compressing data, associative memory can be utilised to locate matches between data blocks more quickly, which in turn enables the data to be compressed more effectively.

The ability to recall information quickly is one of the primary benefits associated with associative memory. Because information is retrieved according to the data's content rather than its memory location, the central processing unit (CPU) does not have to search through memory cells one at a time, which can save a lot of time. Instead, it can conduct the search in parallel, which can significantly increase the rate at which it is completed.

Associative memory, also known as content-addressable memory (CAM), is a sort of computer memory that is used to locate data quickly and

effectively based on the data's content, as opposed to the data's memory address. In a nutshell, CAM stands for content-addressable memory. It is structured in the form of a sequence of memory cells, each of which stores a value as well as a key that may be utilised to determine the value. It is particularly helpful in circumstances in which data needs to be accessible rapidly based on its content, such as networking and data compression; in addition, it is quicker than standard memory.

Direct-mapped cache: This is a type of cache memory that uses a hash function to map memory addresses to cache lines. It is a simple and efficient way to implement cache memory, but it can lead to a high number of conflicts when multiple memory addresses map to the same cache line.

Set-associative cache: This is a type of cache memory that uses a hash function to map memory addresses to sets of cache lines. It is more complex than direct-mapped cache, but it reduces the number

Ternary Content-Addressable Memory (TCAM): This type of associative memory can store multiple values for each key, and returns the value that matches the key most closely. It is often used in networking for tasks such as routing tables and in data compression.

Hash table: This is a data structure that is used to implement an associative array, which maps keys to values. It uses a hash function to convert the key into an index in an array, where the value is stored. It is used in several applications such as database indexing, caching, and cryptography.

Bloom filter: This is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. It uses a hash function to map elements to a bit array, and uses a set of hash functions to map elements to multiple positions in the array. It is used in several applications such as network packet filtering and spell checking.

Content-addressable storage (CAS): This type of storage allows data to be retrieved based on the content of the data, rather than the location of the data. CAS systems can be used to improve data retrieval and search performance, and also to provide data deduplication.

5.5 Cache Memory

- Cache memory, also known as CPU cache, is a sort of computer memory that is employed for the purpose of providing a temporary storage location for data that the central processing unit (CPU) is very likely to require in the not too distant future. To facilitate a more rapid rate of data access by the central processing unit (CPU), a speedy and compact memory that is positioned in close proximity to the CPU is utilised.
- Cache memory is often organised into many levels, with each level having a different size and performance than the previous level. This is because cache memory is used to store frequently used data. The L1 cache, which is the first level of cache and is located immediately on the CPU, is the cache with the least amount of space and the fastest access time. The second level of cache is known as L2 cache, and it is located on the motherboard. It is a little bit slower and slightly larger than the first level of cache. The third level of cache, also known as the L3 cache, is placed on the motherboard or in the memory controller. This cache is even larger and slower than the previous two levels combined.
- The most frequently accessed information from the primary memory is copied from cache and stored there so that the memory can function properly. When the central processing unit needs to access a piece of data, it first checks the cache to determine whether or not the item is already present in the cache. It is possible to access the data rapidly if it is stored in the cache, which speeds up the process by which the CPU can access the data. If the material is not already stored in the cache, it will have to be accessed from main memory, which will take significantly more time.
- The average amount of time required to access data from main memory can be cut down with the help of cache memory. Because the central processing unit (CPU) can access the data stored in the cache significantly more quickly than it can access the data stored in the main memory, the CPU is able to spend significantly less

time waiting for data and significantly more time engaging in productive work. The performance of a computer can be significantly boosted as a result of this.

- Cache memory, in a nutshell, is a relatively compact and speedy memory that is situated in close proximity to the central processing unit (CPU). Its primary function is to provide a temporary storage space for data that the CPU is very likely to want in the not-too-distant future. In most cases, it is segmented into multiple levels, each of which may be distinguished from the others by its unique dimensions and rates of progression. When the central processing unit (CPU) needs to access a piece of data, it first checks the cache to see if it already has the data, and if it does, it can access it quickly, which speeds up the CPU's access to the data. This is the goal of cache memory: to reduce the average time it takes to access data from main memory by storing a copy of the data that is used the most frequently in cache.
- There are various distinct varieties of cache memory, including:
- **L1 cache:** This is the smallest and fastest cache, and is located directly on the CPU. It is also known as primary cache or level 1 cache.
- **L2 cache:** This is slightly larger and slower than L1 cache, and is located on the motherboard. It is also known as secondary cache or level 2 cache.
- **L3 cache:** This is even larger and slower than L2 cache, and is located on the motherboard or in the memory controller. It is also known as tertiary cache or level 3 cache.
- **Instruction cache:** This type of cache is used to store instructions that the CPU is likely to need in the near future. This can help to speed up program execution by reducing the number of memory accesses required.
- **Data cache:** This type of cache is used to store data that the CPU is likely to need in the near future. This can help to speed up data access by reducing the number of memory accesses required.

- **Victim cache:** This type of cache is used to store data that has been replaced from another cache. It can be used to reduce the number of cache misses.
- **Write-back cache:** This type of cache is used to temporarily store data that is waiting to be written to a slower storage device, such as a hard disk. It can help to speed up data writes by reducing the number of accesses to the slower storage device.
- **Write-through cache:** This type of cache is used to temporarily store data that is waiting to be written to a slower storage device, such as a hard disk.

5.6 Virtual Memory

The management technique known as virtual memory is a form of computer memory management that enables a computer to use more memory than it actually possesses by temporarily shifting data from memory to disc storage. Virtual memory is a sort of computer memory management. By designating a portion of the available space on the hard drive as extra memory, it grants applications access to a greater quantity of memory than the actual RAM that is installed on a computer.

When it comes to the way in which memory is structured, virtual memory is implemented using a page-based architecture. The pages that make up the computer's physical memory are uniformly sized, and the pages that make up the computer's virtual memory are also the same size. Both types of memory in the computer are organised in this way. A virtual memory address is provided to a programme whenever it makes a request for memory; this address is equivalent to a specific page inside the virtual memory space. After that, the operating system creates a mapping that converts the address of the virtual memory to an address in the physical memory, which is equivalent to a page in the physical memory.

If the page that was requested is already there in the physical memory, then the data can be accessed very rapidly. In the event that the page is not present in the physical memory, the operating system will move it to a storage location on the disc and load a new page into the physical

memory. A page fault is the name given to this process, and it has the potential to slow down the execution of the programme.

Virtual memory gives programmes the ability to use more memory than the actual RAM that is installed on a computer, which is one of the primary benefits of using this type of memory. This is accomplished by utilising some of the space on the hard drive as additional memory. This type of memory is slower than random access memory (RAM), but it is non-volatile and can store more data.

Another benefit is that it enables the operating system to share memory between several applications. This is accomplished by storing different programme pages in the same physical memory but using separate memory locations for each program's page. This makes it possible for the operating system to use the available memory in an effective manner and helps prevent any one programme from consuming up all of the RAM that is available.

In conclusion, the management technique known as virtual memory is a form of computer memory management that enables a computer to use more memory than it actually possesses by temporarily shifting data from memory to disc storage. Virtual memory is a sort of computer memory management. It is implemented as a page-based system, which means that the computer's physical memory is split up into small blocks of a predetermined size that are referred to as pages, and the computer's virtual memory is similarly split up into pages of the same size. When a programme requests memory, the operating system replies by providing the programme with a virtual memory address that corresponds to a page in the virtual memory space. The operating system then maps this virtual memory address to a location in the physical memory. It enables programmes to use more memory than the actual RAM that has been installed on a computer, and it also enables the operating system to share memory with multiple applications. There are several types of virtual memory:

- **Paging:** This is a type of virtual memory management where the computer's physical memory is divided into small fixed-size blocks called pages, and the computer's virtual memory is also divided

into the same size of pages. When a program requests memory, it is given a virtual memory address, which corresponds to a page in the virtual memory space. The operating system then maps this virtual memory address to a physical memory address, which corresponds to a page in the physical memory.

- **Segmentation:** This is a type of virtual memory management where the computer's physical memory is divided into variable-size blocks called segments, and the computer's virtual memory is also divided into segments. When a program requests memory, it is given a virtual memory address, which corresponds to a segment in the virtual memory space. The operating system then maps this virtual memory address to a physical memory address, which corresponds to a segment in the physical memory.
- **Swapping:** This is a type of virtual memory management where the operating system temporarily transfers entire processes from main memory to disk storage, in order to free up memory for other processes. This is typically used in systems where the amount of physical memory is limited.
- **Demand paging:** This is a technique of virtual memory management in which pages are not loaded into memory until they are actually needed, this is done to minimize the use of memory resources.
- **Page replacement algorithms:** These are techniques used by the operating system to determine which pages to remove from memory when memory is full. Some common examples are: FIFO, LRU, LFU, and clock algorithm

5.6 Characteristics of multiprocessors

A sort of computer system known as a multiprocessor is one that makes use of more than one processor in order to carry out a number of different jobs at the same time. These processors can be of the same type or various types, and they can be connected in a variety of different ways, such as

through a common bus or a crossbar switch. Another possibility is that they are all of the same type.

When it comes to the organisation of memory, multiprocessor systems often make use of a primary memory that is shared by all of the processors and can be accessed by any processor. This type of shared memory is referred to as global memory, and it functions as a shared resource that all processors can access in a parallel fashion. Each CPU has its own local cache memory, which is a small and quick memory used to temporarily store data that the processor is likely to require in the not-too-distant future. This data is stored in the cache memory until the processor needs it.

The data that is shared between the processors, such as global variables, arrays, and other data structures, is stored in the shared memory. This memory can also be used to store other types of data structures. Data that is specific to each processor, such as local variables, and data that is needed by just one processor are stored in the local cache memory. This memory is also used to store data that is not shared between processors. The ability of multiprocessor systems to accomplish numerous tasks simultaneously is one of the primary benefits of having multiple processors, as this can significantly improve the pace at which a computer operates. This is especially helpful in circumstances in which the workload is split up into numerous distinct activities, such as the simulation of scientific phenomena, the encoding of video, and the modelling of financial transactions.

One other advantage of multiprocessor systems is that they may be utilised to make a computer system more reliable. This is one of the many benefits of using such systems. In the case that one of the processors fails, the remaining processors can continue to carry out their duties, so preventing the failure of the entire system.

In a nutshell, a multiprocessor is a certain kind of computer system that makes use of numerous processors in order to carry out a number of different tasks all at once. The connections between these processors can be made in a number of different methods, for as by using a shared bus or a crossbar switch. They often make use of something referred to as global

memory for their shared main memory, which all processors are able to access. Additionally, each processor comes equipped with its very own private cache memory. The shared memory is where data is stored that is accessible by all of the processors, whereas the local cache memory is where data is stored that is only accessible by one CPU at a time. The capacity to carry out many operations at the same time is one of the primary benefits of using a multiprocessor system. This can significantly improve the speed at which a computer performs its functions and also helps to ensure that the system is more reliable.

This Page Intentionally Left Blank

Bibliography

1. **Mano, M. M., & Kime, D. R. (2018).** Computer system architecture (3rd ed.). Pearson.
2. **Stallings, W. (2018).** Computer organization and architecture: designing for performance (10th ed.). Pearson.
3. **Tanenbaum, A. S., & Woodhull, A. S. (2016).** Structured computer organization (7th ed.). Prentice Hall.
4. **Hennessy, J. L., & Patterson, D. A. (2018).** Computer architecture: a quantitative approach (6th ed.). Morgan Kaufmann Publishers.
5. **Deitel, H. M., & Deitel, P. J. (2019).** Computer system architecture (3rd ed.). Prentice Hall.
6. **Tannenbaum, A. S. (2015).** Computer networks (5th ed.). Prentice Hall.
7. **Morris Mano, M. M. (2013).** Computer system architecture (3rd ed.). Prentice Hall.
8. **William Stallings, W. (2016).** Computer organization and architecture: designing for performance (10th ed.). Pearson.
9. **Mano, M. M. (2013).** Computer system architecture (3rd ed.). PHI Pvt. Ltd.
10. **Rafiquzzaman, M., & Chandra, R. (n.d.).** Modern computer architecture. Galgotia Publications Pvt. Ltd.

Webliography

1. Tanenbaum, A. S. (n.d.). Computer organization and architecture. Retrieved from <http://www.cs.vu.nl/~ast/books/CA/>
2. Computer Science Department. (n.d.). Computer system architecture. Retrieved from <http://www.cs.columbia.edu/~sedwards/classes/2002/w4118/>
3. Computer Science and Engineering. (n.d.). Computer organization and architecture. Retrieved from <http://www.cse.iitb.ac.in/~aritra/cs621/>

4. Sheehan, D. (n.d.). Computer organization and architecture. Retrieved from <https://www.cs.cmu.edu/~412-s08/>
5. Computer Architecture and Systems Group. (n.d.). Computer architecture. Retrieved from <https://www.cas.mcmaster.ca/cas/courses/csci-3104/>