****

**Department of Computer Engineering**

**LAB Manual**

**Class: SY BTech**

**Semester: III**

**Course Name: Computer Organization and Operating System**

**Academic Year: 2023-24**

**Prepared By: Mr. S. R. Pawar**

**Vision**

* To be known for imparting quality education in computer engineering to serve the changing needs of global industry.

**Mission**

* To create an environment that fosters technical and professional growth to make graduates globally competent.
* To develop industry ready professionals, researchers, and entrepreneurs to solve real world problems and societal issues.

**Program Education Objectives(PEOs)**

* **PEO1:** Work productively as successful computer professionals / entrepreneurs / researchers in global industry.
* **PEO2:** Adapt latest technological skills to face challenges of the modern computing industry.
* **PEO3:** Work in finance, healthcare, security and banking sectors as a team member or a leader with ethical values.

**Program Specific Outcomes (PSOs)**

**PSO1 Problem Solving and Programming Skills:** Graduates will be able to apply knowledge of procedural or object oriented programming to solve computing problems

**PSO2 Professional Skills:** Graduates will be able to design and develop software systems using front end and back end technologies by following standard software engineering principles

**PSO3 Professional Skills:** Graduates will be able to analyse the algorithms and implement optimized solutions in the domain of machine learning, natural language processing, security , cloud computing and Internet of Things

**Program Outcomes (POs)**

**Graduates will be able to**

1. Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. **[Engineering knowledge]**
2. Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. **[Problem analysis]**
3. Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. **[Design/development of solutions]**
4. Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. **[Conduct investigations of complex problems]**
5. Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations. **[Modern tool usage]**
6. Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice. **[The engineer and society]**
7. Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. **[Environment and sustainability]**
8. Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. **[Ethics]**
9. Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings. **[Individual and team work]**
10. Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. **[Communication]**
11. Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments. **[Project management and finance]**
12. Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change. **[Life-long learning]**

|  |  |
| --- | --- |
| Course Outcome statements and CO-PO-PSO Mapping | |
| COURSE NAME: Computer Organization and Operating System | COURSE CODE: **COPCC305** |
| CLASS: SY | NAME OF FACULTY MEMBER: Dr.S.N.Zaware/Mr.Pawar S.R. |
|  | |

|  |  |
| --- | --- |
| *Course Objectives* | |
|  | To explain the structure, function and characteristics of computer systems. |
|  | To illustrate the structure and function of I/O and memory Organization. |
|  | To acquaint the basic of 8086 microprocessor and assembly language. |
|  | To make aware of different services of operating system and Linux commands |

|  |  |
| --- | --- |
| *Course Outcomes:* Students will be able to | |
| 305.1 | Describe functions and components of computer system and compare the architecture of processors. |
| 305.2 | Demonstrate the memory partitioning through operating system installation. |
| 305.3 | Install input output peripherals. |
| 305.4 | Implement an ALP application using assembly language programming. |
| 305.5 | Implement process scheduling algorithm . |
| 305.6 | Make use of different Linux commands to perform various operations. |

**CO-PO-PSO Mapping:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CO** | **Course Outcome Statement** | **POs/**  **PSOs** | **Cognitive/Bloom’s Level** | **Knowledge Category** | **Theory Sessions (hrs.)** | **Practical (if applicable)**  **(hrs.)** | **Tutorial (if applicable)**  **(hrs.)** |
| **305.1** | Describe functions and components of computer system and compare the architecture of processors. | PO1 | Understand | Conceptual | 6 | 2 | NA |
| **305.2** | Demonstrate the memory partitioning through operating system installation. | PO1, PO5 | Understand  Apply | Conceptual  Procedural | 6 | 8 | NA |
| **305.3** | Install input output peripherals. | PO1 | Understand  Apply | Conceptual  Procedural | 6 | 4 | NA |
| **305.4** | Implement an ALP application using assembly language programming. | PO1,PO5, PSO1 | Understand  Apply | Conceptual  Procedural | 6 | 6 | NA |
| **305.5** | Implement process scheduling algorithm . | PO1, PSO1 | Understand  Apply | Conceptual  Procedural | 6 | 4 | NA |
| **305.6** | Make use of different Linux commands to perform various operations. | PO1 | Understand  Apply | Conceptual  Procedural | 6 | 2 | NA |
|  | **Total Hours of instruction** | | | | **36** | **26** |  |

**Strength of CO-PO-PSO Mapping:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PO/**  **PSO** | **COs** | **Total Number of Sessions** | **Percentage of Sessions Devoted** | **Strength of Mapping**  **3 – 66 to 100 %**  **2 – 34 to 65 %**  **1 – 1 to 33%**  **“-“ i.e. (no mapping) <1%** |
| PO1 | 305.1, 305.2,305.3,305.4,305.5,305.6 | 62 | 100% | 3 |
| PO5 | 305.2,305.4 | 26 | 41.93% | 2 |
| PSO1 | 305.4,305.5 | 22 | 35.48% | 1 |

**Mapping of course with PO/PSO**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PO  Course | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 | PSO3 |
| C305.1 | 3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| C305.2 | 3 | - | - | - | 2 | - | - | - | - | - | - | - | - | - | - |
| C305.3 | 3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| C305.4 | 3 | - | - | - | 2 | - | - | - | - | - | - | - | 1 | - | - |
| C305.5 | 3 | - | - | - | - | - | - | - | - | - | - | - | 1 | - | - |
| C305.6 | 3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| C305 | 3 | - | - | - | 2 | - | - | - | - | - | - | - | 1 | - | - |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **List of Experiments** | | | | | |
| Subject**: Computer Organization and Operating System** | | | | | |
| CLASS: SYBTech A & B | | SEMESTER: III | | | |
| **Academic Year : 2023-2024** | | | | | |
| **Sr. No.** | **Title of Experiment** | | **CO** | **PO** | **PSO** |
|  | Assemble the computer system using various hardware components. Study the general structure of computer system. | | 305.1 | 1,12 | 1 |
|  | Study of memory hierarchy with respect to capacity, access time, performance and cost per bit. | | 305.2 | 1 | 1 |
|  | Single boot OS installation (Linux OS installation) | | 305.2 | 1,5,12 | 1 |
|  | Dual boot OS installation (Linux / Windows OS installation) | | 305.2 | 1,5,12 | 1 |
|  | Multi OS installation | | 305.2 | 1,5,12 | 1 |
|  | Demonstrate installation of I/O Peripherals- Printer/Scanner | | 305.3 | 1,5,12 | 1 |
|  | Study of the architecture of 8086 microprocessor | | 305.4 | 1 | 1 |
|  | Implement 8/16/32/64 bit ALP to accept two digit numbers and display same on command prompt | | 305.4 | 1,5 | 1 |
|  | Implement 8/16/32/64 bit ALP to accept two digit numbers perform addition and display result on command prompt. | | 305.4 | 1,5 | 1 |
|  | Write a program to implement Scheduling Algorithms: FCFS, SJF and Round Robin. | | 305.5 | 1,5 | 1 |
| 11. | Study of various Linux commands. | | 305.6 | 1,5,12 | 1 |

**Experiment No. 1**

**Aim:**

Assemble the computer system using various hardware components. Study the general structure of computer system.

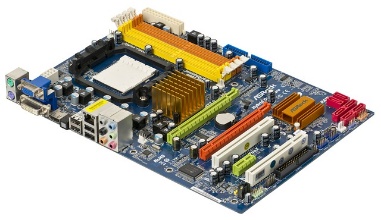
**Objective:**

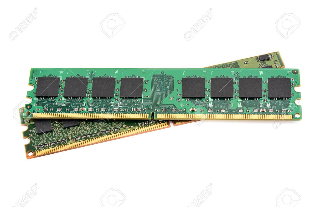
To demonstrate different hardware components of computer.

**Theory:**

A computer system is a complex interplay of various hardware components that work together to execute tasks and process data. Understanding the general Structure of a computer system is crucial for both enthusiasts and professionals in the field of computing. Here, we dive into the theory behind computer systems, their components, and how they interact:

1. Central Processing Unit (CPU): The CPU is the brain of the computer, responsible for executing instructions. It performs arithmetic and logical operations, fetches data from memory, and controls the flow of data within the System.

2. Motherboard: The motherboard serves as the central hub of the computer, connecting all major components. It contains the CPU socket, memory slots, expansion Slots, and various connectors.

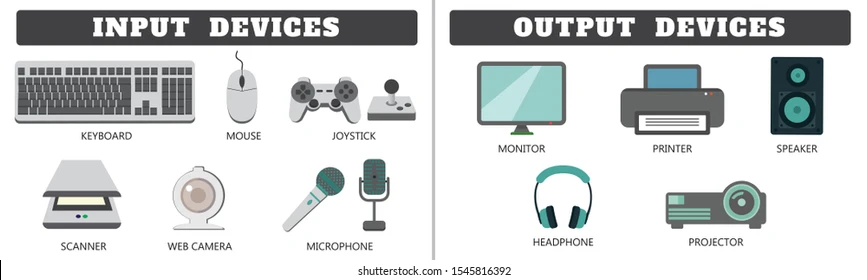
3.Memory Modules: Random Access Memory (RAM) modules provide temporary Storage for data that the CPU is the actively processing. They are essential for quick data access and retrieval.

4.Storage Devices: Computer Systems use various storage devices, Such as hard dish drives (HDDs) and Solid State drives (SDDs), to Store data and software for long term use. These devices differ in forms of speed, capacity, and reliability.



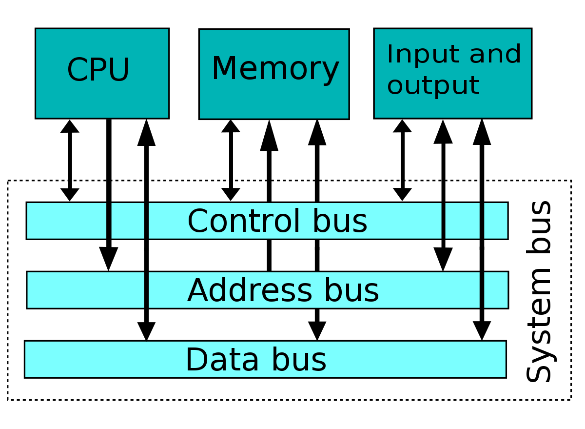
Power Supply Unit (PSU): The PSU provides electrical power to all components within the computer. It converts external power into voltages suitable for internal components.

Input and Output (I/O) Devices: These include peripherals like keyboards, mice, monitors, and central storage devices. I/o devices allow user to interact with the computer and receive output.

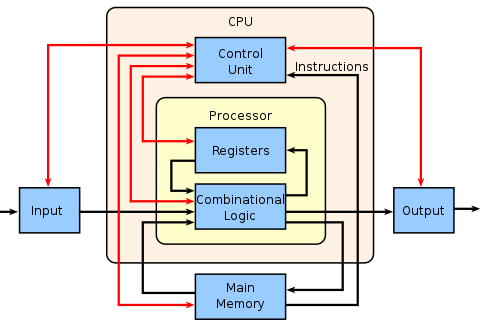


Expansion Cards: Expansion slots on the motherboard accommodate add in cords, such as graphics cards, Sound cards, and network, adapters, to enhance system functionality.

System Bus: The systems bus consists of data, address, and control buses that facilitates user interactions, and crosslines the execution of programs. Il acts as an intermediary between users and hardware.



System Architecture: Computer stung follow specific architectural designs, such as the von Neumann architecture, separates memory for data and instructions. Understanding these architectural is fundamental to comprehending how computers process information. By assembling hardware components and studying their interactions this project aims to provide hands on experience with the practical aspects of a computer system. It enables individuals to grasp the significance of each component, their roles in data processing and the architectural principles that underlie modern computing systems.



**Conclusion:**

In conclusion, the proceed of assembling a computer system using diverse hardware components and exploring the general structure of a computer system offers valuable insights into the intricate workings of modern computing.

**Experiment No. 2**

**Aim:**

Study of memory hierarchy with respect to capacity, access time, performance and cost per bit.

**Objective:**

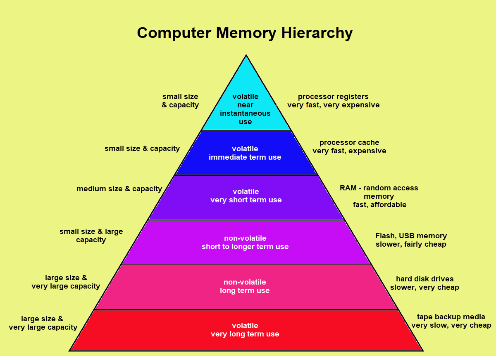
To study memory hierarchy.

**Theory:**

**1. Memory Hierarchy:**

Memory hierarchy is a structure that organizes computer memory into various levels, each with different characteristics. The primary purpose of memory hierarchy is to provide a trade-off between memory speed, capacity, and cost. The typical levels of memory hierarchy include:

* **Registers:** Fastest, smallest, and most expensive. Located in the CPU.
* **Cache Memory:** Faster than main memory but smaller in capacity.
* **Main Memory (RAM):** Slower than cache but larger in capacity.
* **Secondary Storage (Hard Drives, SSDs):** Larger capacity but slower than RAM.
* **Tertiary Storage (Tape Drives):** Very large capacity but even slower access.



**2. Capacity vs. Access Time:**

* **Registers:** Very small capacity (few kilobytes) but extremely fast access time (nanoseconds).
* **Cache Memory:** Larger capacity (several megabytes) with faster access time than RAM (few nanoseconds to tens of nanoseconds).
* **Main Memory (RAM):** Moderate capacity (several gigabytes to terabytes) with access time in the range of tens of nanoseconds to hundreds of nanoseconds.
* **Secondary Storage:** Large capacity (hundreds of gigabytes to petabytes) with slower access time (milliseconds or more).
* **Tertiary Storage:** Extremely large capacity (petabytes or more) with very slow access time (seconds or more).

**3. Performance Impact:**

The use of memory hierarchy significantly impacts system performance. By placing frequently accessed data in faster but smaller memory levels (e.g., cache), the overall system speed can be increased.

**4. Cost per Bit:**

* **Registers and Cache:** Expensive per bit due to their high-speed nature.
* **Main Memory (RAM):** Less expensive than cache but more costly than secondary storage.
* **Secondary Storage:** Least expensive per bit among volatile memories.

**Conclusion:**

This lab provided insights into the memory hierarchy and its impact on capacity, access time, performance, and cost per bit. Understanding these trade-offs is crucial for designing computer systems that balance the need for speed and capacity within budget constraints.

**Experiment No. 3**

**Aim:**

Single boot OS installation (Windows OS installation)

**Objective:**

To install windows operating system (single boot OS).

**Theory:**

**Before you begin:**

1. **Backup:** Ensure you have a backup of your important data before proceeding with the installation.
2. **System Requirements:** Make sure your computer meets the minimum system requirements for the Windows version you are installing.

**Installation Steps:**

1. **Insert the Installation Media:**

Insert the Windows installation DVD or USB flash drive into your computer. 

**2. Boot from Installation Media:**

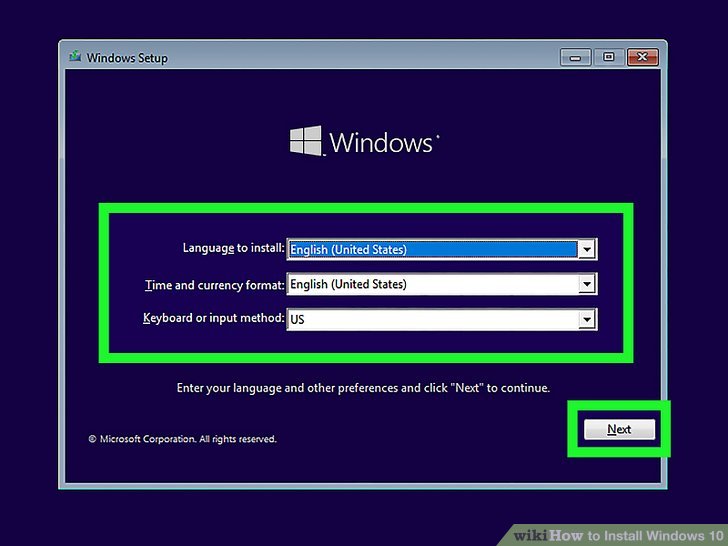
Restart your computer and make sure it boots from the installation media.

You may need to change the boot order in the BIOS/UEFI settings. This is usually done by pressing a key (like F8, F12, or Del) during startup to access the BIOS/UEFI. 

**1. Windows Setup:**

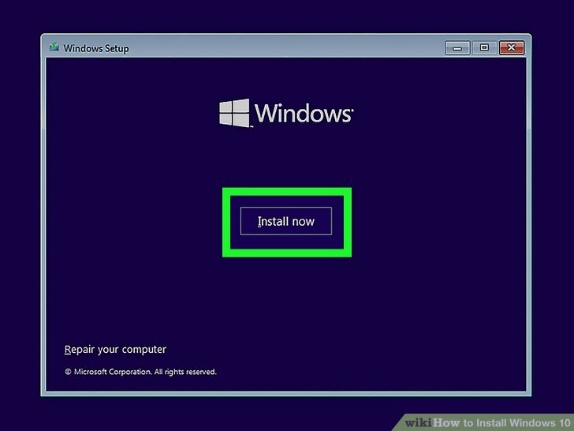
Once the computer boots from the installation media, you'll see the Windows Setup screen.

Select your language, time, and keyboard input preferences, then click "Next."



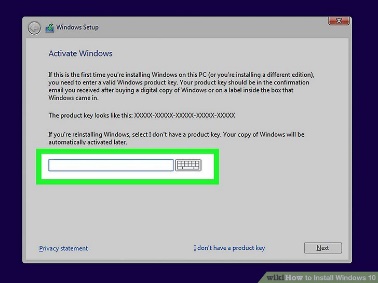
**2.Install Now:**

Click on "Install Now" to start the installation process.



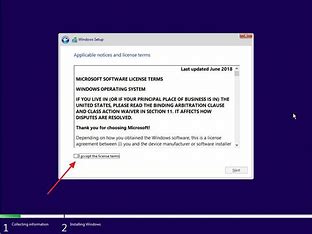
**3.Enter Product Key:**

Enter your Windows product key when prompted. Click "Next."



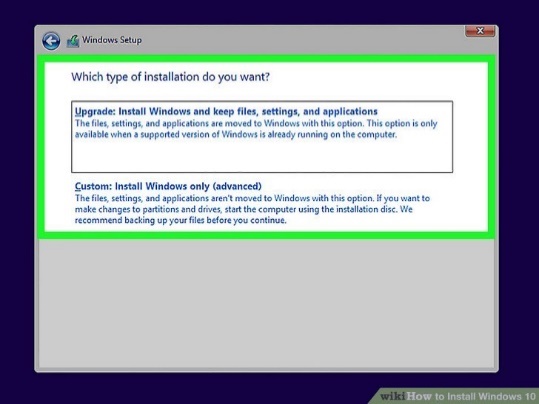
**1.Accept License Terms:**

Read and accept the license terms, then click "Next."



**2.Custom Installation:**

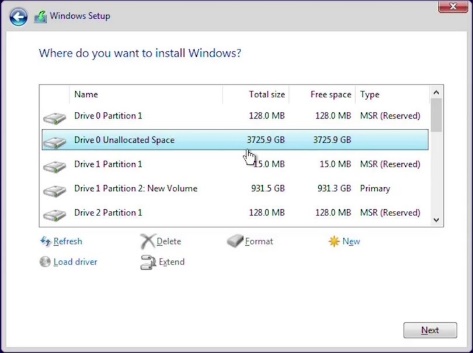
Choose the "Custom: Install Windows only (advanced)" option.



**3.Partition Selection:**

Select the partition where you want to install Windows. You can create a new partition or use an existing one.

Click "Next" to start the installation.



**4.Install Windows:**

The installation process will begin. Your computer may restart several times during this process.

**5.Set up Windows:**

After the installation is complete, you'll be prompted to set up Windows. Follow the on-screen instructions to choose your region, language, and other preferences.

**6. Restart:**

Restart your computer to ensure all changes take effect.

After completing these steps, you should have a single-boot Windows installation on your computer. Remember that specific details may vary, and it's important to follow on-screen instructions and refer to the official documentation for the version of Windows you are installing.

**Conclusion:**

This lab guided users through the process of installing a single-boot operating system (Windows). Understanding the steps involved in installation, partitioning, and configuring the boot loader is crucial for successfully deploying an operating system on a computer.

**Experiment No 4**

**Aim:**

Dual boot OS installation (Linux Os installation)

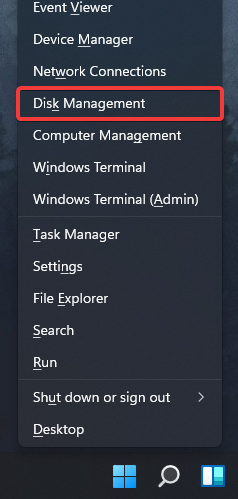
**Objective:**

To install ubuntu operating system (Dual boot OS).

**Theory:**

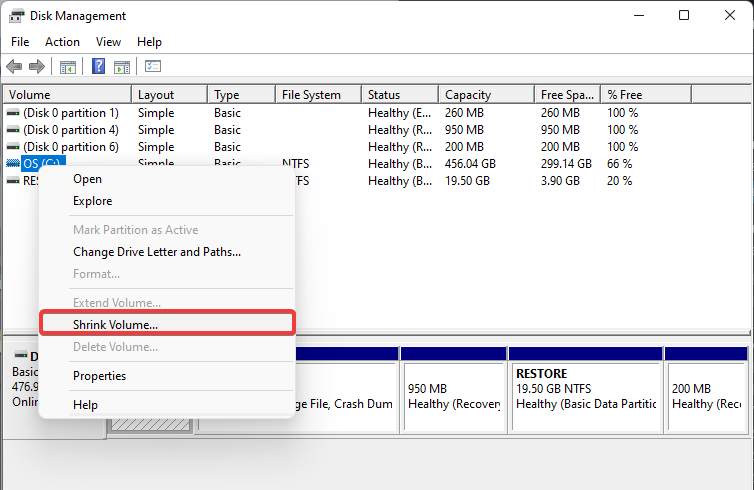
**Ubuntu Installation Steps:**

1. Create a dedicated Ubuntu partition : Open **Disk Management**. You can do that by pressing Windows Key + X and selecting *Disk Management* from thelist.

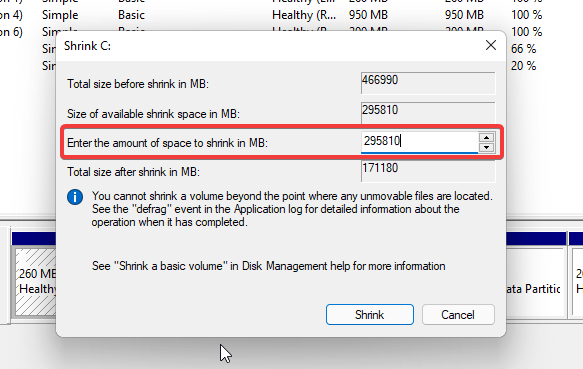


2. Now, you need to select a drive that you want to shrink. Choose one with at least 50GB of free space, but if you plan to use Ubuntu extensively, you might need even more space.

3. Right-click the desired drive and choose **Shrink Volume** from the context menu.



4. Set the Enter the amount of space to shrink in MB to **50000** or more and click **Shrink**.

[](https://cdn.windowsreport.com/wp-content/uploads/2021/07/shrink-properties.png)

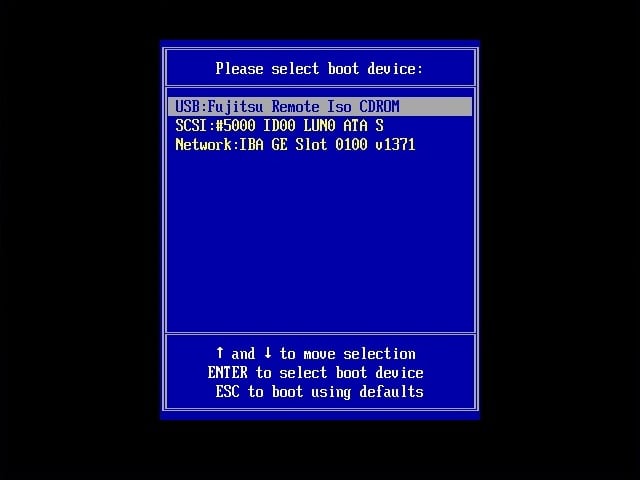
5. Next, click the Unallocated space that you just created and choose **New Simple Volume**.

6. Set the *Simple volume size in MB* to the maximum value and click **Next**.

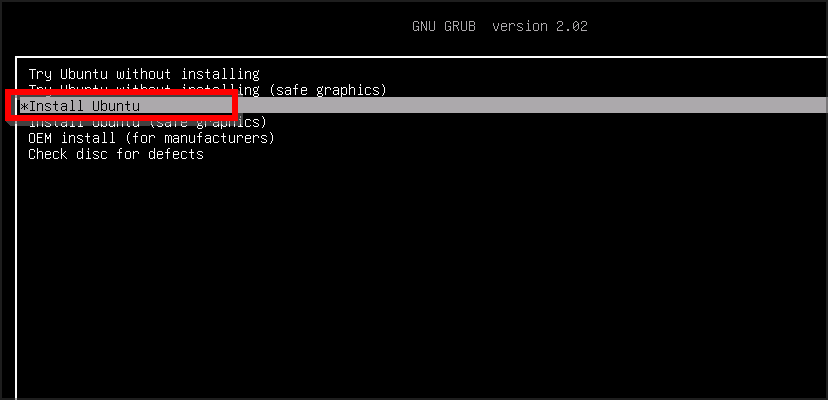
7. Finally, set the desired drive letter and label and click **Finish** once you’re done.

8. Make sure that the Ubuntu flash drive is connected to your PC.

9. While your PC boots, keep pressing F2,F10,F12 to access the boot menu and select your USB drive. The key might be different depending on your motherboard.



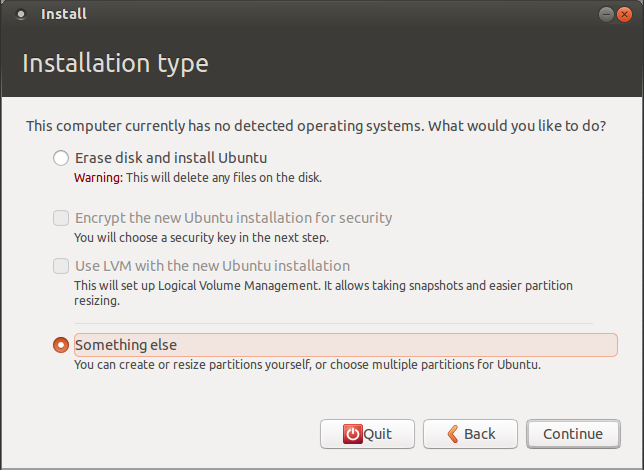
10. When your PC boots from the flash drive, choose the option to **Install Ubuntu**.

[](https://cdn.windowsreport.com/wp-content/uploads/2021/08/install-ubuntu-menu.png)

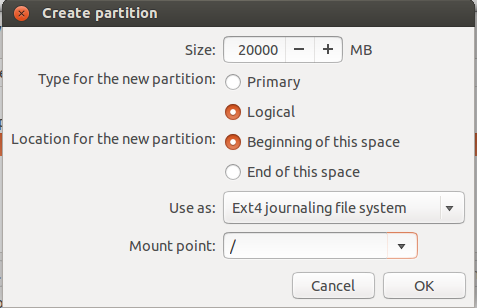
11.Select the language and the keyboard layout that you want to use.

12. Then, select **Normal installation** and click **Continue**. You can also use Minimal installation and download the necessary updates later.

13.Select **Something else**.

[](https://cdn.windowsreport.com/wp-content/uploads/2021/07/ubuntu-something-else.png)

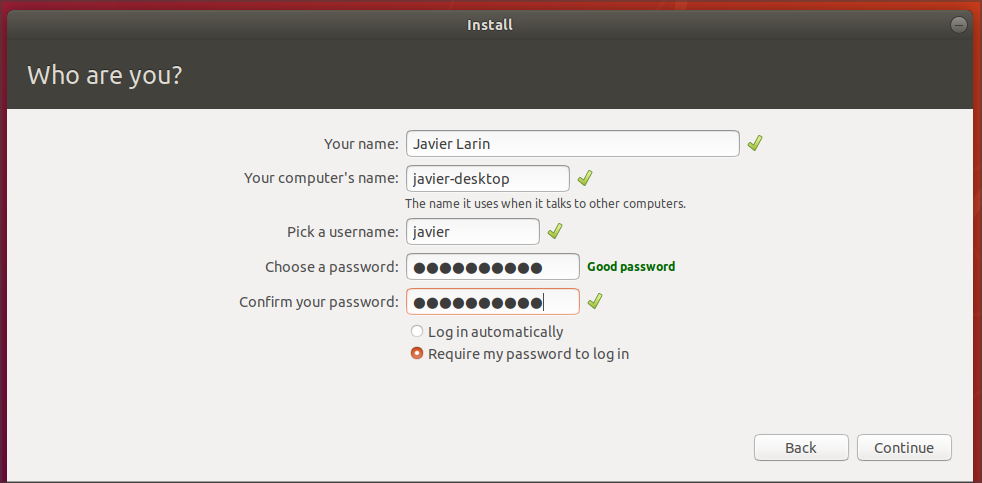
14. Up next, select the partition that you created for Ubuntu and format it to the **ext4** file system.

[](https://cdn.windowsreport.com/wp-content/uploads/2021/07/format-new-partition-ubuntu.png)

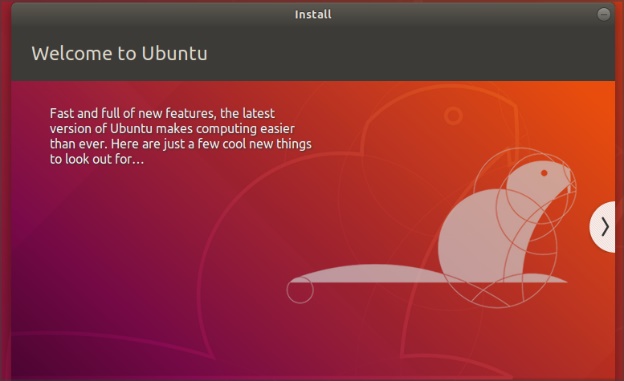
15. Click on **Install Now**.

16. Select your location.

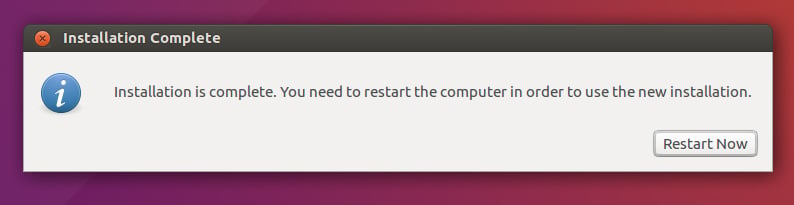
17.Enter the login details for Ubuntu and click on **Continue**.

[](https://cdn.windowsreport.com/wp-content/uploads/2021/07/ubuntu-credentials.png)

18. Wait for the installation to finish.

[](https://cdn.windowsreport.com/wp-content/uploads/2021/07/installing-ubuntu.png)

19. Click the **Restart Now** button.

[](https://cdn.windowsreport.com/wp-content/uploads/2021/07/installation-complete-ubuntu.jpg)

**Conclusion:**

This lab provided guidance on setting up a dual-boot configuration with Linux alongside an existing Windows installation. Understanding the partitioning and boot loader configuration is crucial for a successful and seamless dual-boot experience.

**Experiment No. 5**

**Aim:**

Multi OS installation

**Objectives:**

To install multi operating system.

**Theory:**

**1. Prerequisites for Multi-OS Installation:**

* **Existing Windows Installation:** Ensure that Windows is already installed on the system.
* **Free Disk Space:** Allocate free space on the disk for Linux and macOS installations.
* **Backup:** Backup important data to prevent loss during the installation process.
* **Bootable Media:** Create bootable USBs or DVDs with the installation media for Windows, Linux, and macOS.

**2. Operating System Installation Steps:**

a. Windows Installation:

* Insert the Windows installation media and install it on a designated partition.
* Complete the initial setup, including creating a user account.

b. Linux Installation:

* Insert the Linux installation media and install it on a separate partition.
* Configure partitions for root (/), home, and swap.
* Install the Grub boot loader to the Linux partition.

c. macOS Installation (if applicable):

* Insert the macOS installation media (only if allowed by licensing).
* Install macOS on a dedicated partition (this may involve additional steps depending on hardware compatibility).

**3. Multi-Boot Configuration:**

* Grub, installed during the Linux installation, will act as the primary boot loader.
* Grub will automatically detect Windows and macOS installations (if applicable) and add them to the boot menu.

**4. Verification:**

* Reboot the system and verify that the Grub menu appears.
* Test booting into Windows, Linux, and macOS (if applicable) to ensure all operating systems are functional.

**Conclusion:**

This lab provided guidance on setting up a multi-boot configuration with Windows, Linux, and macOS (if applicable). Understanding the partitioning and boot loader configuration is crucial for a successful and seamless multi-OS experience. Ensure compliance with licensing agreements, especially when installing macOS on non-Apple hardware.

**Experiment No. 6**

**Aim:**

Demonstrate installation of I/O Peripherals- Printer/Scanner

**Objectives:**

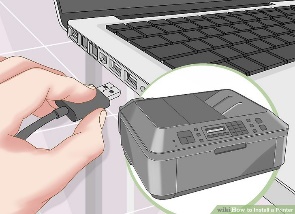
To install input and output peripherals.

**Theory:**

**1. Printer Installation:**

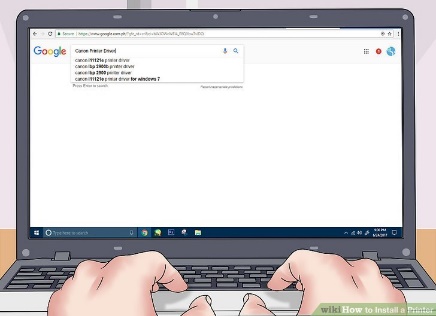
a. Physical Connection:

* Connect the printer to the computer using the provided USB cable or via a network connection.



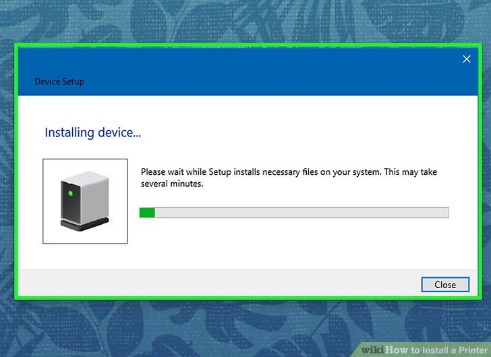
b. Driver Installation:

* Insert the printer installation CD or download the latest drivers from the manufacturer's website.
* Follow the on-screen instructions to install the printer drivers.



c. Configuration:

* Wait for your operating system to detect and install the printer

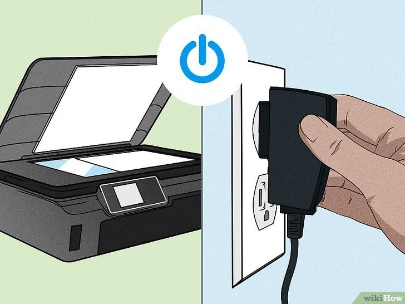
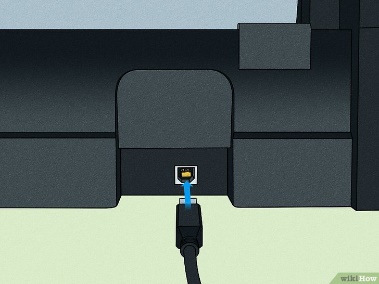


* Set the installed printer as the default printer in the operating system settings.
* Configure additional settings such as paper size, print quality, and preferences as needed.

**2. Scanner Installation:**

a. Physical Connection:

* Connect the scanner to the computer using the provided USB cable or via a network connection.

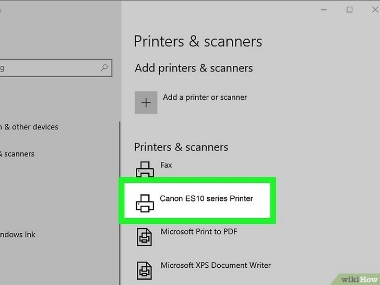
 

b. Driver Installation:

* Insert the scanner installation CD or download the latest drivers from the manufacturer's website.
* Follow the on-screen instructions to install the scanner drivers.

c. Configuration:

* Verify that the scanner is recognized by the operating system.
* Set up any additional scanning preferences, such as resolution and file format.



**Conclusion:**

This lab demonstrated the installation of I/O peripherals, specifically a Printer and a Scanner. Understanding the physical connections and driver installation processes is crucial for efficient and trouble-free operation of these peripherals.

**Experiment No. 8**

Aim:

Implement 8/16/32/64 bit ALP to accept two digit numbers and display same on command prompt

**Working Environment :**

1. **CPU – Core 2 Duo, 64bit with 2.3 GHz clock frequency**
2. **OS – Ubuntu, 64bit**

**Tools :**

* **Editor – gedit, a GNU editor**
* **Assembler – NASM (Netwide Assembler)**
* **LINKER – LD , a GNU linker**

**Theory:**

In this program five 64-bit hexadecimal number has to accept from user and display accepted numbers, for this we have to perform following steps:

1. Declare array to store numbers
2. Accept numbers
3. Display numbers

**Instructions Used:**

1. **MOV: Move byte or Word**
2. The MOV instruction Copies a word or byte of data from fixed/specified source to a fixed/specified destination.
3. **Flags Affected:** No flags are affected

**Syntax:** MOV Destination , Source

**Eg-1:** MOV [SI] , AL

1. **ADD: ADD Byte or Word**
2. This instruction adds a contents of source to contents of destination and store the result to specified destination. Both operand can not be memory locations. Source and destination both have same type i.e. byte or word.
3. **Flags Affected:** OF, SF, ZF, AF, CF and PF

**Syntax:**   ADD destination , source

**Eg-1:**   ADD BL , CL

1. **DEC:** **Decrement Byte or word by 1**
2. This instruction subtract 1 from destination operand. The operand may be byte or word. Segment registers can not be decremented by this instruction.
3. **Flags Affected:** OF, SF, ZF, AF and PF

**Syntax:**   DEC destination

**Eg-1:**   DEC BX // subtract 1 from contents of BX register

1. **JNZ: JUMP if Not Zero**
2. JNZ is an unsigned conditional jump instructions. If condition is true, then control is transferred to the target specified in the instruction. If condition is false, then control passes to the instruction that follows the conditional jump.
3. **Flags Affected:** ZF

**Algorithm:**

1. Start
2. Initialize the .data segment
3. Display message 1 “Enter 5 64-bit numbers from user:”
4. Initialize the .bss segment
5. Reserve byte for array and counter variable
6. Initialize .text segment
7. Initialize counter to 05 and rbx to 00
8. Store element in array by moving pointer to start of the array
9. Add rsi to rbx
10. Move rdx to 17
11. Call syscall function
12. Add 17 to rbx
13. Decrement counter
14. Jump to step 8 until counter value is not zero.
15. Display message 2 “Entered 5 64 bit numbers are:”
16. Initialize counter to 05 and rbx to 00
17. Display array element
18. Add rsi to rbx
19. Move rdx to 17
20. Call syscall function
21. Add 17 to rbx
22. Decrement counter
23. Jump to step 16 until counter value is not zero.
24. End (Terminate the program)

**Conclusion:**

Hence we have successfully implement X86/64 ALP to accept five 64 bit Hexadecimal number from user and store them in an array and display the accepted number.

**Experiment No. 9**

Aim:

Implement 8/16/32/64 bit ALP to accept two digit numbers perform addition and display result on command prompt.

**ALGORITHM:-**

1. **Addition:-**
   1. Start the process
   2. Initialize the count value
   3. Get the two data.
   4. Add the two data values
   5. If carry exists increment the count value.
   6. Store the result.
   7. Stop the process.

**PROGRAM**

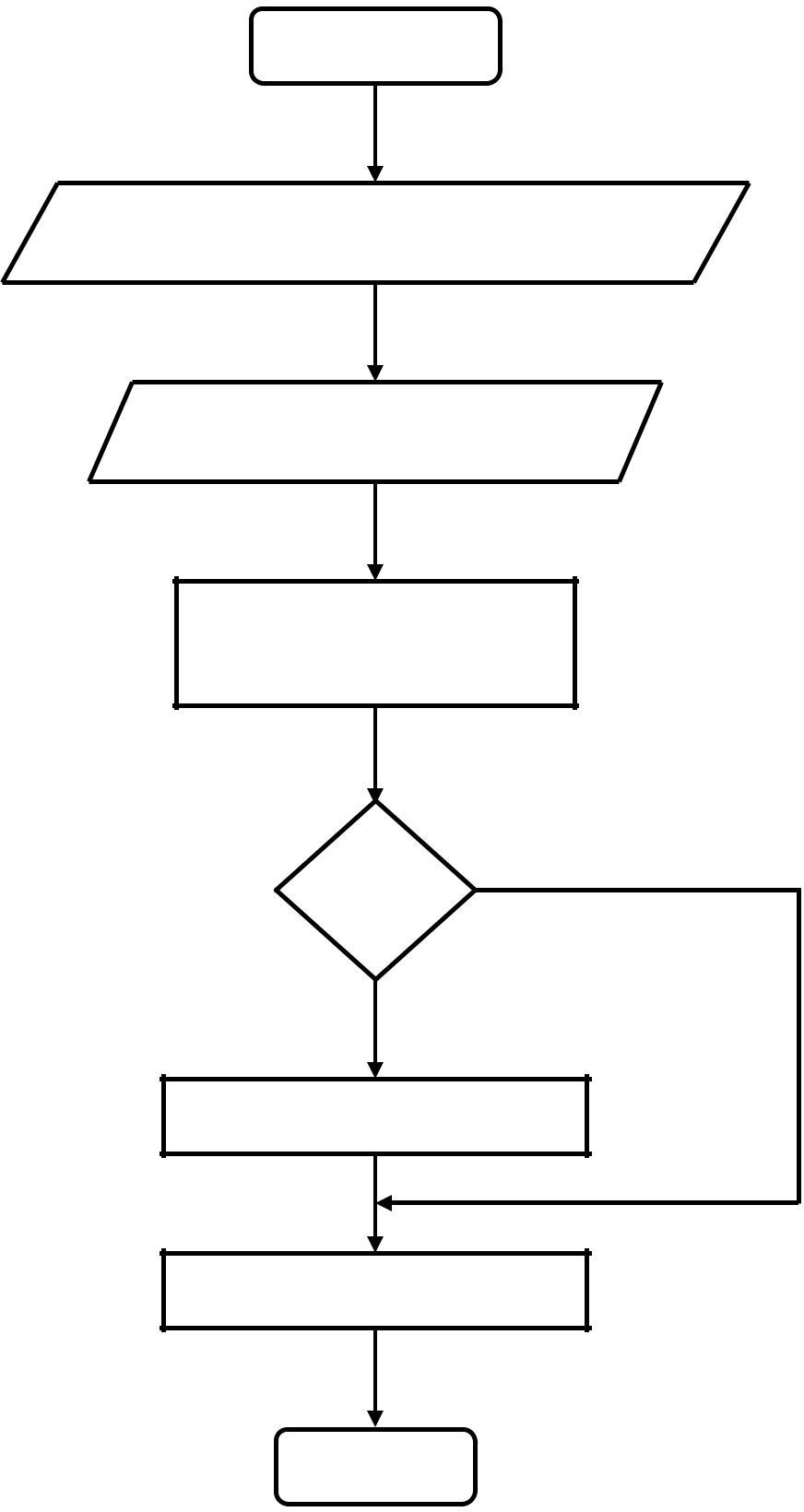
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Labe l | Address | Mnemonics | | Hex code | Comments |
| Opcode | Operand |  |  |
| LOOP1: | 1000 | MOV | CL , 00 | C6, C1, 00 | ; Initialize the count |
| 1003 | MOV | AX, 0F0C | C7, C0, 0C, 0F | ; Move1st data to accumulator |
| 1007 | MOV | BX, 111F | C7, C3, 1F, 11 | ; Move2nd data to register |
| 100B | ADD | BX, AX | 01, C3 | ; Add the two data |
| 100D | JNC | LOOP1 | 73, 02 | ; Jump on no carry |
| 100F | INC | CL | FE, C1 | ; Increment the counter |
| 1011 | MOV | [1100], BX | 89,1E, 00, 11 | ; Store the result |
| 1015 | MOV | [1102], CL | 88, 0E, 02,11 | ; Store the carry |
| 1019 | HLT |  | F4 | ; Stop the process |

**OUTPUT**

**16 – BIT ADDITION**

|  |  |
| --- | --- |
| **Address** | **Output** |
| **1100** | **2B** |
| **1101** | **20** |
| **1102** | **00** |

**16 BIT ADDITION**



Start

Initialize count as zero for carry

Get the two data

Add the datas

If

Carry

No

Yes

Increment the count

Store the result & carry

Stop

**1 B) 16 BIT SUBTRACTION**

**ALGORITHM:-**

1. Start the process
2. Initialize the count value
3. Get the two data and subtract it.
4. If carry exists, get the 2‟s complement of the value.
5. Store the result and carry value.
6. Stop the process.

**PROGRAM**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Labe l | Address | Mnemonics | | Hex code | Comments |
|  |  | Opcode | Operand |  |  |
| LOOP1: | 1000 | MOV | CL, 00 | C6, C1, 00 | ; Initialize the count |
| 1003 | MOV | AX, [1100] | 8B, 06, 00, 11 | ; Move1st data to accumulator |
| 1007 | MOV | BX, [1102] | 8B, 1E, 02, 11 | ; Move 2nd data to „B‟ register |
| 100B | SUB | BX, AX | 29, C3 | ; Subtract the two datas |
| 100D | JNC | LOOP1 | 73, 05 | ; Jump on no carry |
| 100F | INC | CL | FE, C1 | ; Increment the counter |
| 1011 | NOT | BX | F7, D3 | ; Get the complement value |
| 1013 | INC | BX | 43 | ; Increment the value |
| 1014 | MOV | [1104], BX | 89, 1E, 04, 11 | ; Store the result |
| 1018 | MOV | [1106], CL | 88, 0E, 06, 11 | ; Store the carry |
| 101C | HLT |  | F4 | ; Stop the process |

**OUTPUT**

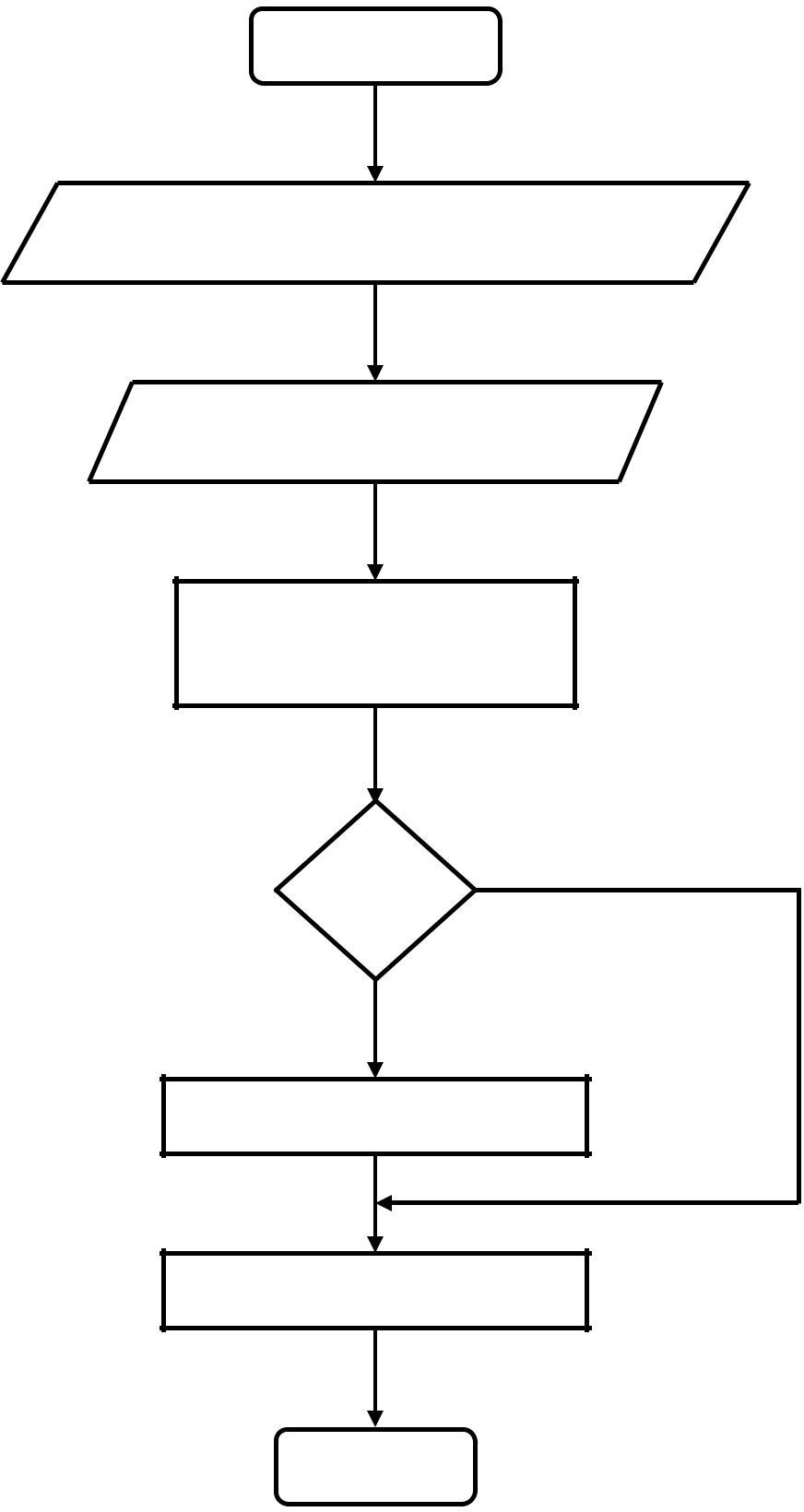
**16 – BIT SUBTRACTION**

|  |  |
| --- | --- |
| **Address** | **Input** |
| **1100** | **76** |
| **1101** | **86** |
| **1102** | **45** |
| **1103** | **81** |

|  |  |
| --- | --- |
| **Address** | **Output** |
| **1104** | **31** |
| **1105** | **65** |
| **1106** | **00** |

**FLOWCHAR T:-**

**Subtraction:-**



Start

Initialize count as zero for borrow

Get the two data

Subtract the datas

If Carry exists

No

Yes

Take 2‟s complement

Store the result & carry

Stop

**1.C) 16 BIT MULTIPLICATION**

**ALGORITHM:-**

1. Start the process
2. Get the two values
3. Multiply the two values.
4. Store the result and overflow
5. Stop the process.

**PROGRAM**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Label | Address | Mnemonics | | Hex code | Comments |
|  |  | Opcode | Operand |  |  |
|  | 1000 | MOV | SI, 1100 | C7, C6,00, 11 | ; Move the source index |
| 1004 | MOV | AX, [SI] | 8B, 04 | value |
| 1006 | MOV | BX, [SI + 02] | 8B, 54, 02 | ; Move the first data |
| 1009 | MUL | BX | F7, E3 | ; Get the second data |
| 100B | MOV | [SI + 04], | 89, 44, 04 | ; Multiply the data |
| 100E | MOV | AX | 89, 54, 0b | ; Store the result |
| 1011 | HLT | [SI + 06], | F4 | ; Store the over flow |
|  |  | DX |  | ; Stop the process |

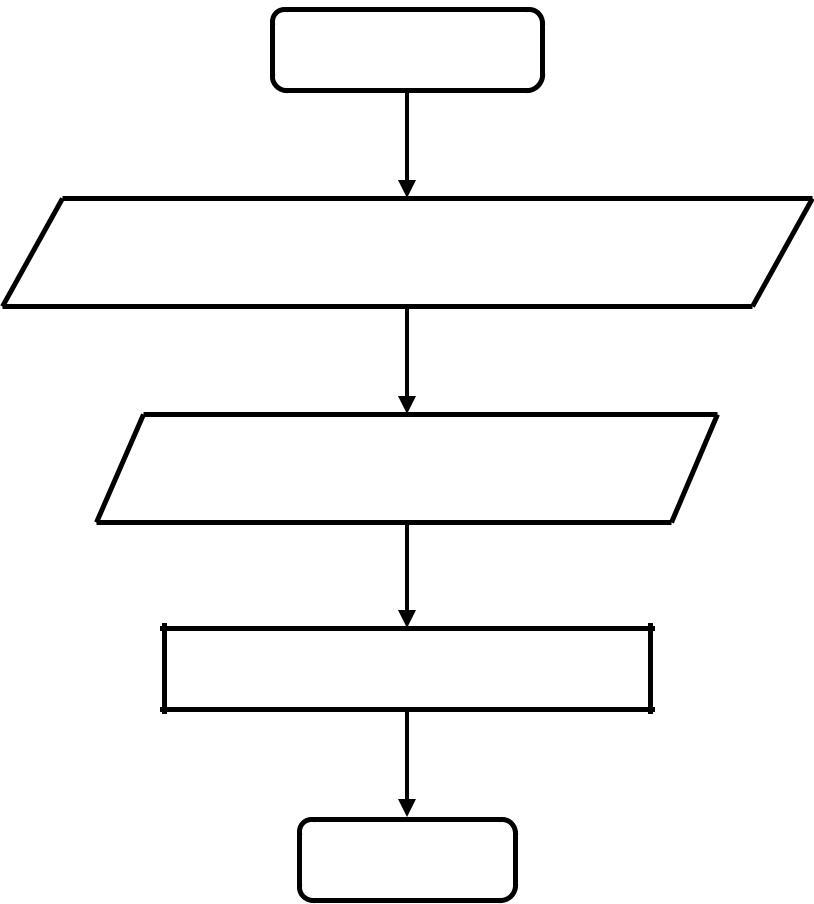
**INPUT OUTPUT**

|  |  |
| --- | --- |
| **Address** | **Input** |
| **1100** | **11** |
| **1101** | **11** |
| **1102** | **00** |
| **1103** | **11** |

|  |  |
| --- | --- |
| **Address** | **Output** |
| **1104** | **00** |
| **1105** | **21** |
| **1106** | **22** |
| **1107** | **01** |

**FLOW CHART:-**

**Multiplication:-**



Start

Get the two values

Multiply the values

Store the result & overflow

Stop

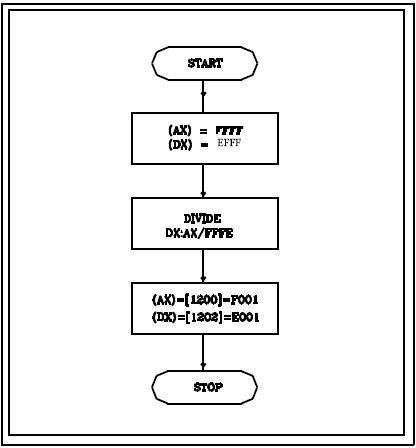
**D) 16 BIT DIVISION**

AIM:

To perform division of a 32 bit number by a 16 bit number and store the quotient and remainder in memory

**ALGORITHM:-**

1. Start the process
2. Get the two values
3. Initialize „DX‟ register as zero
4. Divide the values
5. Store the quotient and remainder
6. Stop the process.

**FLOWCHART:**

**D) 16 BIT DIVISION PROGRAM**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Label | Address | Mnemonics | | Hex code | Comments |
|  |  | Opcode | Operand |  |  |
|  | 1000 | MOV | SI, 1100 | C7, C6,00, 11 | ; Get the source index value |
| 1004 | MOV | Ax, [SI] | 8B, 04 | ; Get the first data |
| 1006 | MOV | DX, [SI + | 8B, 54, 02 | ; Initialize „DX‟ register |
| 1009 | MOV | 02] | 8B, 5C, 04 | value |
| 100C | DIV | BX, [SI + 04] | F7, E3 | ; Get the dividend value |
| 100E | MOV | BX | 89, 44, 06 | ; Divide the value |
| 1011 | MOV | [SI + 06], | 89, 54, 08 | ; Move the quotient |
| 1014 | HLT | AX | F4 | ; Move the remainder & store |
|  |  | [SI + 08], |  | ; Stop the process |
|  |  | DX |  |  |

**16 – BIT DIVISIION**

|  |  |
| --- | --- |
| **Address** | **Input** |
| **1100** | 42 (DIVIDEND) |
| **1101** | 24 |
| **1102** | 00 |
| **1103** | 00 |
| **1104** | 02(DIVISOR) |
| **1105** | 00 |

|  |  |
| --- | --- |
| **Address** | **Output** |
| **1106** | 21(QUOTIENT) |
| **1107** | 12 |
| **1108** | 00(REMAINDER) |
| **1109** | 00 |

**RESULT:-**

Thus the assembly language program for 16 Bit Arithmetic and Logical operations has been done and verified.

**VIVA QUES TIONS AND ANSWERS**

1. **What is a Microprocessor?**

Microprocessor is a CPU fabricated on a single chip, program-controlled device, which fetches the instructions from memory, decodes and executes the instructions.

1. **What is Instruction Set?**

It is the set of the instructions that the Microprocessor can execute.

1. **What is Clock Speed?**

Clock speed is measured in the MHz and it determines that how many instructions a processor can processed. The speed of the microprocessor is measured in the MHz or GHz.

1. **What are the features of Intel 8086?**

Features:

Released by Intel in 1978 Produced from 1978 to 1990s A 16-bit microprocessor chip.

Max. CPU clock rate:5 MHz to 10 MHz Instruction set: x86-16

1. **What are the flags in 8086?**

In 8086 carry flag, Parity flag, Auxiliary carry flag, Zero flag, Overflow flag, Trace flag, Interrupt flag, Direction flag, and Sign flag.

1. **What is assembly language?**

The language in which the mnemonics (short -hand form of instructions) are used to write a program is called assembly language. The manufacturers of microprocessor give the mnemonics.

1. **What are machine language and assembly language programs?**

The software developed using 1's and 0's are called machine language, programs. The software developed using mnemonics are called assembly language programs.

1. **What is the drawback in machine language and assembly language, programs?**

The machine language and assembly language programs are machine dependent. The programs developed using these languages for a particular machine cannot be directly run on another machine.

1. **Define bit, byte and word.**

A digit of the binary number or code is called bit. Also, the bit is the fundamental storage unit of computer memory. The 8-bit (8-digit) binary number or code is called byte and 16-bit binary number or code is called word. (Some microprocessor manufactures refer the basic data size operated by the processor as word).

1. **What is a bus?**

Bus is a group of conducting lines that carries data, address and control signals.

**Experiment No. 10**

**Aim:**

Write a program to implement Scheduling Algorithms: FCFS, SJF and Round Robin.

## OBJECTIVE

Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the above problem.

a) FCFS b) SJF c) Round Robin d) Priority

## DESCRIPTION

Assume all the processes arrive at the same time.

## FCFS CPU SCHEDULING ALGORITHM

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

## SJF CPU SCHEDULING ALGORITHM

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

## ROUND ROBIN CPU SCHEDULINGALGORITHM

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly.

## PRIORITY CPU SCHEDULING ALGORITHM

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

## PROGRAM

## FCFS CPU SCHEDULING ALGORITHM

#include<stdio.h> #include<conio.h> main()

{

int bt[20], wt[20], tat[20], i, n; float wtavg, tatavg;

clrscr();

printf("\nEnter the number of processes -- "); scanf("%d", &n);

for(i=0;i<n;i++)

{

printf("\nEnter Burst Time for Process %d -- ", i); scanf("%d", &bt[i]);

}

wt[0] = wtavg = 0; tat[0] = tatavg = bt[0]; for(i=1;i<n;i++)

{

wt[i] = wt[i-1] +bt[i-1];

tat[i] = tat[i-1] +bt[i]; wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];

}

printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

for(i=0;i<n;i++)

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]); printf("\nAverage Waiting Time -- %f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n); getch();

}

|  |  |  |  |
| --- | --- | --- | --- |
| ***INPUT*** |  |  |  |
| Enter the number of processes -- | | 3 |  |
| Enter Burst Time for Process 0 -- | | 24 |  |
| Enter Burst Time for Process 1 -- | | 3 |  |
| Enter Burst Time for Process 2 -- | | 3 |  |
| ***OUTPUT*** |  |  |  |
| PROCESS | BURST TIME | WAITING TIME | TURNAROUND TIME |
| P0 | 24 | 0 | 24 |
| P1 | 3 | 24 | 27 |
| P2 | 3 | 27 | 30 |

Average Waiting Time-- 17.000000

Average Turnaround Time -- 27.000000

## SJF CPU SCHEDULING ALGORITHM

#include<stdio.h> #include<conio.h> main()

{

int p[20], bt[20], wt[20], tat[20], i, k, n, temp; float wtavg, tatavg;

clrscr();

printf("\nEnter the number of processes -- "); scanf("%d", &n);

for(i=0;i<n;i++)

{

p[i]=i;

printf("Enter Burst Time for Process %d -- ", i); scanf("%d", &bt[i]);

}

for(i=0;i<n;i++)

for(k=i+1;k<n;k++)

if(bt[i]>bt[k])

{

temp=bt[i]; bt[i]=bt[k]; bt[k]=temp;

}

wt[0] = wtavg = 0;

temp=p[i]; p[i]=p[k]; p[k]=temp;

tat[0] = tatavg = bt[0]; for(i=1;i<n;i++)

{

wt[i] = wt[i-1] +bt[i-1];

tat[i] = tat[i-1] +bt[i]; wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];

}

printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

for(i=0;i<n;i++)

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]); printf("\nAverage Waiting Time -- %f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n); getch();

}

|  |  |  |  |
| --- | --- | --- | --- |
| ***INPUT*** |  |  |  |
| Enter the number of processes -- | | 4 |  |
| Enter Burst Time for Process 0 -- | | 6 |  |
| Enter Burst Time for Process 1 -- | | 8 |  |
| Enter Burst Time for Process 2 -- | | 7 |  |
| Enter Burst Time for Process 3 -- | | 3 |  |
| ***OUTPUT*** |  |  |  |
| PROCESS | BURST TIME | WAITING TIME | TURNAROUND TIME |
| P3 | 3 | 0 | 3 |
| P0 | 6 | 3 | 9 |
| P2 | 7 | 9 | 16 |
| P1 | 8 | 16 | 24 |

Average Waiting Time -- 7.000000

Average Turnaround Time -- 13.000000

## ROUND ROBIN CPU SCHEDULING ALGORITHM

#include<stdio.h> main()

{

int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max; float awt=0,att=0,temp=0;

clrscr();

printf("Enter the no of processes -- "); scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\nEnter Burst Time for process %d -- ", i+1); scanf("%d",&bu[i]);

ct[i]=bu[i];

}

printf("\nEnter the size of time slice -- "); scanf("%d",&t);

max=bu[0]; for(i=1;i<n;i++)

if(max<bu[i])

max=bu[i]; for(j=0;j<(max/t)+1;j++)

for(i=0;i<n;i++)

if(bu[i]!=0)

if(bu[i]<=t)

{

for(i=0;i<n;i++)

{

}

else

{

}

tat[i]=temp+bu[i]; temp=temp+bu[i]; bu[i]=0;

bu[i]=bu[i]-t; temp=temp+t;

wa[i]=tat[i]-ct[i]; att+=tat[i];

awt+=wa[i];

}

printf("\nThe Average Turnaround time is -- %f",att/n); printf("\nThe Average Waiting time is -- %f ",awt/n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");

for(i=0;i<n;i++)

printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);

getch();

}

### INPUT

Enter the no of processes – 3

Enter Burst Time for process 1 – 24 Enter Burst Time for process 2 -- 3 Enter Burst Time for process 3 -- 3

Enter the size of time slice – 3

## OUTPUT

The Average Turnaround time is – 15.666667 The Average Waiting time is -- 5.666667

|  |  |  |  |
| --- | --- | --- | --- |
| PROCESS | BURST TIME | WAITING TIME | TURNAROUND TIME |
| 1 | 24 | 6 | 30 |
| 2 | 3 | 4 | 7 |
| 3 | 3 | 7 | 10 |

## PRIORITY CPU SCHEDULING ALGORITHM

#include<stdio.h> main()

{

int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp; float wtavg, tatavg;

clrscr();

printf("Enter the number of processes --- "); scanf("%d",&n);

for(i=0;i<n;i++)

{

p[i] = i;

printf("Enter the Burst Time & Priority of Process %d --- ",i); scanf("%d %d",&bt[i], &pri[i]);

}

for(i=0;i<n;i++)

for(k=i+1;k<n;k++)

if(pri[i] > pri[k])

{

temp=p[i]; p[i]=p[k]; p[k]=temp;

temp=bt[i]; bt[i]=bt[k]; bt[k]=temp;

temp=pri[i]; pri[i]=pri[k]; pri[k]=temp;

}

wtavg = wt[0] = 0;

tatavg = tat[0] = bt[0];

for(i=1;i<n;i++)

{

wt[i] = wt[i-1] + bt[i-1];

tat[i] = tat[i-1] + bt[i];

wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

for(i=0;i<n;i++)

printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);

printf("\nAverage Waiting Time is --- %f",wtavg/n); printf("\nAverage Turnaround Time is --- %f",tatavg/n); getch();

}

|  |  |
| --- | --- |
| ***INPUT*** |  |
| Enter the number of processes -- 5  Enter the Burst Time & Priority of Process 0 --- 10 | 3 |
| Enter the Burst Time & Priority of Process 1 --- 1 | 1 |
| Enter the Burst Time & Priority of Process 2 --- 2 | 4 |
| Enter the Burst Time & Priority of Process 3 --- 1 | 5 |
| Enter the Burst Time & Priority of Process 4 --- 5 | 2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***OUTPUT***  PROCESS | PRIORITY | BURST TIME | WAITING TIME | TURNAROUND TIME |
| 1 | 1 | 1 | 0 | 1 |
| 4 | 2 | 5 | 1 | 6 |
| 0 | 3 | 10 | 6 | 16 |
| 2 | 4 | 2 | 16 | 18 |
| 3 | 5 | 1 | 18 | 19 |

Average Waiting Time is --- 8.200000 Average Turnaround Time is --- 12.000000

1.1 OBJECTIVE

Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the above problem.

a) FCFS b) SJF c) Round Robin d) Priority

1.2 DESCRIPTION

Assume all the processes arrive at the same time.

1.2.1 FCFS CPU SCHEDULING ALGORITHM

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

1.2.2 SJF CPU SCHEDULING ALGORITHM

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

1.2.3 ROUND ROBIN CPU SCHEDULINGALGORITHM

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly.

1.2.4 PRIORITY CPU SCHEDULING ALGORITHM

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

1.3 PROGRAM

1.3.1 FCFS CPU SCHEDULING ALGORITHM

#include<stdio.h> #include<conio.h> main()

{

int bt[20], wt[20], tat[20], i, n; float wtavg, tatavg;

clrscr();

printf("\nEnter the number of processes -- "); scanf("%d", &n);

for(i=0;i<n;i++)

{

printf("\nEnter Burst Time for Process %d -- ", i); scanf("%d", &bt[i]);

}

wt[0] = wtavg = 0; tat[0] = tatavg = bt[0]; for(i=1;i<n;i++)

{

wt[i] = wt[i-1] +bt[i-1];

tat[i] = tat[i-1] +bt[i]; wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];

}

printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

for(i=0;i<n;i++)

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]); printf("\nAverage Waiting Time -- %f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n); getch();

}

INPUT

Enter the number of processes -- 3

Enter Burst Time for Process 0 -- 24

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 -- 3

OUTPUT

PROCESS BURST TIME WAITING TIME TURNAROUND TIME

P0 24 0 24

P1 3 24 27

P2 3 27 30

Average Waiting Time-- 17.000000

Average Turnaround Time -- 27.000000

1.3.2 SJF CPU SCHEDULING ALGORITHM

#include<stdio.h> #include<conio.h> main()

{

int p[20], bt[20], wt[20], tat[20], i, k, n, temp; float wtavg, tatavg;

clrscr();

printf("\nEnter the number of processes -- "); scanf("%d", &n);

for(i=0;i<n;i++)

{

p[i]=i;

printf("Enter Burst Time for Process %d -- ", i); scanf("%d", &bt[i]);

}

for(i=0;i<n;i++)

for(k=i+1;k<n;k++)

if(bt[i]>bt[k])

{

temp=bt[i]; bt[i]=bt[k]; bt[k]=temp;

}

wt[0] = wtavg = 0;

temp=p[i]; p[i]=p[k]; p[k]=temp;

tat[0] = tatavg = bt[0]; for(i=1;i<n;i++)

{

wt[i] = wt[i-1] +bt[i-1];

tat[i] = tat[i-1] +bt[i]; wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];

}

printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

for(i=0;i<n;i++)

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]); printf("\nAverage Waiting Time -- %f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n); getch();

}

INPUT

Enter the number of processes -- 4

Enter Burst Time for Process 0 -- 6

Enter Burst Time for Process 1 -- 8

Enter Burst Time for Process 2 -- 7

Enter Burst Time for Process 3 -- 3

OUTPUT

PROCESS BURST TIME WAITING TIME TURNAROUND TIME

P3 3 0 3

P0 6 3 9

P2 7 9 16

P1 8 16 24

Average Waiting Time -- 7.000000

Average Turnaround Time -- 13.000000

1.3.3 ROUND ROBIN CPU SCHEDULING ALGORITHM

#include<stdio.h> main()

{

int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max; float awt=0,att=0,temp=0;

clrscr();

printf("Enter the no of processes -- "); scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\nEnter Burst Time for process %d -- ", i+1); scanf("%d",&bu[i]);

ct[i]=bu[i];

}

printf("\nEnter the size of time slice -- "); scanf("%d",&t);

max=bu[0]; for(i=1;i<n;i++)

if(max<bu[i])

max=bu[i]; for(j=0;j<(max/t)+1;j++)

for(i=0;i<n;i++)

if(bu[i]!=0)

if(bu[i]<=t)

{

for(i=0;i<n;i++)

{

}

else

{

}

tat[i]=temp+bu[i]; temp=temp+bu[i]; bu[i]=0;

bu[i]=bu[i]-t; temp=temp+t;

wa[i]=tat[i]-ct[i]; att+=tat[i];

awt+=wa[i];

}

printf("\nThe Average Turnaround time is -- %f",att/n); printf("\nThe Average Waiting time is -- %f ",awt/n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");

for(i=0;i<n;i++)

printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);

getch();

}

INPUT

Enter the no of processes – 3

Enter Burst Time for process 1 – 24 Enter Burst Time for process 2 -- 3 Enter Burst Time for process 3 -- 3

Enter the size of time slice – 3

OUTPUT

The Average Turnaround time is – 15.666667 The Average Waiting time is -- 5.666667

PROCESS BURST TIME WAITING TIME TURNAROUND TIME

1 24 6 30

2 3 4 7

3 3 7 10

1.3.4 PRIORITY CPU SCHEDULING ALGORITHM

#include<stdio.h> main()

{

int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp; float wtavg, tatavg;

clrscr();

printf("Enter the number of processes --- "); scanf("%d",&n);

for(i=0;i<n;i++)

{

p[i] = i;

printf("Enter the Burst Time & Priority of Process %d --- ",i); scanf("%d %d",&bt[i], &pri[i]);

}

for(i=0;i<n;i++)

for(k=i+1;k<n;k++)

if(pri[i] > pri[k])

{

temp=p[i]; p[i]=p[k]; p[k]=temp;

temp=bt[i]; bt[i]=bt[k]; bt[k]=temp;

temp=pri[i]; pri[i]=pri[k]; pri[k]=temp;

}

wtavg = wt[0] = 0;

tatavg = tat[0] = bt[0];

for(i=1;i<n;i++)

{

wt[i] = wt[i-1] + bt[i-1];

tat[i] = tat[i-1] + bt[i];

wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

for(i=0;i<n;i++)

printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);

printf("\nAverage Waiting Time is --- %f",wtavg/n); printf("\nAverage Turnaround Time is --- %f",tatavg/n); getch();

}

INPUT

Enter the number of processes -- 5

Enter the Burst Time & Priority of Process 0 --- 10 3

Enter the Burst Time & Priority of Process 1 --- 1 1

Enter the Burst Time & Priority of Process 2 --- 2 4

Enter the Burst Time & Priority of Process 3 --- 1 5

Enter the Burst Time & Priority of Process 4 --- 5 2

OUTPUT

PROCESS PRIORITY BURST TIME WAITING TIME TURNAROUND TIME

1 1 1 0 1

4 2 5 1 6

0 3 10 6 16

2 4 2 16 18

3 5 1 18 19

Average Waiting Time is --- 8.200000 Average Turnaround Time is --- 12.000000

**Experiment No. 11**

Aim:

Study of various Linux commands.

Objective: To study and execute the shell commands.

Outcome: work with basic shell commands.

Pre-request/Theme: Students should know the basic commands of DOS.

Description:

UNIX is security conscious, and can be used only by those persons who have an account. Telnet (Telephone Network) is a Terminal emulator program for TCP/IP networks that enables users to log on to remote servers.

To logon, type telnet server\_ipaddress in run window.

User has to authenticate himself by providing username and password. Once verified, a greeting and $ prompt appears. The shell is now ready to receive commands from the user. Options suffixed with a hyphen (–) and arguments are separated by space.

UNIX Shell:

A UNIX shell is a command-line interpreter that provides a user interface for the UNIX operating system . Users direct the operation of the computer by entering commands as text for a command line interpreter to execute or by creating text scripts of one or more such commands.

The commands can be combined using the pipeline (|) operator.

For example, number of users logged in can be obtained as  who | wc –l

COMMAND SHELL: It is a program that interprets commands. It allows a user to execute commands by typing them manually at a terminal, or automatically in programs called shell scripts. A shell is not an operating system. It is a way to interface with the operating system and run commands.

BASH = Bourne Again Shell.

Bash is a shell written as a free replacement to the standard Bourne Shell (/bin/sh) originally written by Steve Bourne for UNIX systems. It has all of the features of the original Bourne Shell, plus additions that make it easier to program with and use from the command line. Since it is Free Software, it has been adopted as the default shell on most Linux systems.

General commands:

|  |  |
| --- | --- |
| **Command** | **Function** |
| date | Used to display the current system date and time. |
| date + %D | Displays date only |
| date + %T | Displays time only |
| date + %Y | Displays the year part of date |
| date +%H | Displays the hour part of time |
| cal | Calendar of the current month |
| cal *year* | Displays calendar for all months of the specified year |
| cal *month year* | Displays calendar for the specified month of the year |
| who | Get the information about all the users currently working in the system |
| who am i | It is used to know in which terminal the user is currently logged on |
| id | It is used to display the login name. |
| Tty | Used to display the terminal name |
| uname | Displays the Operating System |
| uname–r | Shows version number of the OS (kernel). |
| uname–n | Displays domain name of the server |
| echo "*txt*" | Displays the given text on the screen |
| echo $HOME | Displays the user's home directory |
| Bc | Basic calculator. Press **Ctrl+d** to quit |
| Lp *file* | Allows the user to spool a job along with others in a print queue. |
| man *cmdname* | Manual for the given command. Press **q** to exit |
| history | To display the commands used by the user since logon. |
| Exit | Exit from a process. If shell is the only process then logs out |

Directory commands:

|  |  |
| --- | --- |
| **Command** | **Function** |
| Pwd | Path of the present working directory |
| mkdir *dir* | A directory is created in the given name under the current directory |
| mkdir *dir1dir2* | A number of sub-directories can be created under one stroke |
| cd *subdir* | Change Directory. If the *subdir* starts with**/**then path starts from  **root**(absolute)otherwise from current working directory. |
| cd | To switch to the home directory. |
| cd/ | To switch to the root directory. |
| cd .. | To move back to the parent directory |
| rmdir subdir | Removes an empty sub-directory. |

File commands:

|  |  |
| --- | --- |
| **Command** | **Function** |
| cat >*filename* | To create a file with some contents. To end typing press **Ctrl+d**. The  **>**symbol means redirecting output to a file.(**<** for input) |
| cat *filename* | Displays the file contents. |
| cat >>*filename* | Used to append contents to a file |
| cp *src des* | Copy files to given location. If already exists, it will be overwritten |
| cp–i*src des* | Warns the user prior to overwriting the destination file |
| cp–r*src des* | Copies the entire directory, all its sub-directories and files. |
| mv *old new* | To rename an existing file or directory. –i option can also be used |
| mv *f1 f2 f3 dir* | To move a group of files to a directory. |
| mv–v *old new* | Display name of each file as it is moved. |
| file *filename* | Determine the type of file |
| rm *file* | Used to delete a file or group of files. –i option can also be used |
| spell *filename* | Find the spelling errors in the file |
| rm\* | To delete all the files in the directory. |
| rm–r\* | Deletes all files and sub-directories |
| rm–f\* | To forcibly remove even write-protected files |
| ls | Lists all files and subdirectories (blue colored) in sorted manner. |
| ls *name* | To check whether a file or directory exists. |
| ls *name***\*** | Short-hand notation to list out file names of a specific pattern. |
| ls–a | Lists all files including hidden files (files beginning with**.**) |
| ls–x *dir name* | To have specific listing of a directory. |
| ls–R | Recursive listing of all files in the subdirectories |
| ls–l | Long listing showing file access rights (read/write/execute-**rwx** for user/group/others**-ugo**). |
| cmp *file1file2* | Used to compare two files. Displays nothing if files are identical. |
| wc *file* | It produces a statistics of lines (**l**), words (**w**), and characters(**c**). |
| chmod *perm file* | Changes permission for the specified file.(r=4,w=2,x=1) chmod740*file*setsallrightsforuser,readonlyforgroups and no rights for others |

CONCLUSION:

Thus the study and execution of basic shell commands has been completed successfully.

VIVA QUESTIONS:

1. What is the difference between soft and hard links?

2. What is the difference between $\* and $@?

3. Write the command to display the date in the form dd/mm/yyyy

4. How will you record your login session in the file session.lst?

5. How will you ensure that bc displays the result of all divisions using 3 decimal places?

6. State True or False: “/” can be used in a filename

7. In how many ways you can find out what your home directory is ?

8. What does cd do when used without arguments?

9. If rmdirc\_progs fails, what could be the possible reasons ?