| | |
|---|---|
| **SUBJECT: MICROPROCESSOR LAB (MPL)** | |
| **NAME:** | |
| **CLASS:SE COMP** | **ROLL NO.:** |
| **SEMESTER:SEM-II** | **YEAR:2023-24** |
| **DATE OF PERFORMANCE:** | **DATE OF SUBMISSION:** |
| **EXAMINED:** | |

## Assignment No-02

**Title:-**String length calculation
**Assignment Name:** - Write an X86/64 ALP to accept a string and to display its length.

**Objective-**

- To study various string instruction
- To understand how to define string in data segment.
- To calculate string length.

**Outcome-**

- Students will be able to write code to accept str and display string length.

**Prerequisite-**

System call of Unix for Assembly language Program.

**Hardware Requirement-**

Desktop PC

**Software Requirement-**

Ubuntu 14.04,

Assembler: NASM version 2.10.07Linker: ld

**Introduction:-**

**Guidelines for the algorithm:**

1) Initialize Data section.
2) Declare string and other required variables.
3) Accept string from user.
4) Count of entered string including enter character is available with RAX register.
5) Display length as value available in RAX.

**Conclusion:-**Hence we implemented an ALP to calculate string length.

**Questions:-**

1. Explain string instruction of 80386?
2. Explain direction flag?

| | |
|---|---|
| **SUBJECT: MICROPROCESSOR LAB (MPL)** | |
| **NAME:** | |
| **CLASS:SE COMP** | **ROLL NO.:** |
| **SEMESTER:SEM-II** | **YEAR:2023-24** |
| **DATE OF PERFORMANCE:** | **DATE OF SUBMISSION:** |
| **EXAMINED:** | |

## Assignment No-05

**Title:-** Program to switch from real mode to protected mode and display the values of GDTR, LDTR, IDTR, TR and MSW Registers.

**Assignment Name: -**Write X86/64 ALP to switch from real mode to protected mode and display the values of GDTR, LDTR, IDTR, TR and MSW Registers.

**Objective-**

- To understand the assembly language program
- To understand 64 bit interrupt.
- To study GDTR, LDTR and IDTR and MSW

**Outcome-**

- Students will be able to understand different assembly language instruction.
- Students will be able to write code for how to display the values of GDTR, LDTR, IDTR, TR and MSW Registers
- Students will be able to switch Processor Mode.

**Prerequisite -**

System call of Unix for Assembly language Program.

**Hardware Requirement-**

Desktop PC

**Software Requirement-**

Ubuntu 14.04,
Assembler: NASM version 2.10.07
Linker: ld

**Introduction:-**

Four registers of the 80386 locate the data structures that control segmented memory

management called as memory management registers:

**1. GDTR :Global Descriptor Table Register**

These register point to the segment descriptor tables GDT. Before any segment register is changed in protected mode, the GDT register must point to a valid GDT. Initialization of the GDT and GDTR may be done in real-address mode. The GDT (as well as LDTs) should reside inRAM, because the processor modifies the accessed bit of descriptors. The instructions LGDT and SGDT give access to the GDTR.

**2. LDTR :Local Descriptor Table Register**

These register point to the segment descriptor tables LDT. The LLDT instruction loads a linear base address and limit value from a six-byte data operand in memory into the LDTR. The SLDT instruction always store into all 48 bits of the six-byte data operand.

**3. IDTR Interrupt Descriptor Table Register**

This register points to a table of entry points for interrupt handlers (the IDT). The LIDTinstruction loads a linear base address and limit value from a six-byte data operand in memory into the IDTR. The SIDT instruction always store into all 48 bits of the six-byte data operand.

**4. TR Task Register**

This register points to the information needed by the processor to define the current task.,These registers store the base addresses of the descriptor tables (A descriptor table is simply a memory array of 8-byte entries that contain Descriptors and descriptor stores all the information about segment) in the linear address space and store the segment limits.

**SLDT: Store Local Descriptor Table Register**

**Operation:** DEST ← 48-bit BASE/LIMIT register contents;

**Description:** SLDT stores the Local Descriptor Table Register (LDTR) in the two-byte register or memory location indicated by the effective address operand. This register is a selector that points into the Global Descriptor Table. SLDT is used only in operating system software. It is notused in application programs.

**Flags Affected:** None

**SGDT: Store Global Descriptor Table Register**

**Operation:** DEST ← 48-bit BASE/LIMIT register contents;

**Description:** SGDT copies the contents of the descriptor table register the six bytes of memory indicated by the operand. The LIMIT field of the register is assigned to the first word at the effective address. If the operand-size attribute is 32 bits, the next three bytes are assigned the BASE field of the register, and the fourth byte is written with zero. The last byte is undefined. Otherwise, if the operand-size attribute is 16 bits, the next 4 bytes are assigned the 32-bit BASE field of the register. SGDT and SIDT are used only in operating system software; they are not used in application programs.

**Flags Affected:** None

**SIDT: Store Interrupt Descriptor Table Register**

**Operation:** DEST ← 48-bit BASE/LIMIT register contents;

**Description:** SIDT copies the contents of the descriptor table register the six bytes of memory indicated by the operand. The LIMIT field of the register is assigned to the first word at the effective address. If the operand-size attribute is 32 bits, the next three bytes are assigned the BASE field of the register, and the fourth byte is written with zero. The last byte is undefined. Otherwise, if the operand-size attribute is 16 bits, the next 4 bytes are assigned the 32-bit BASE field of the register. SGDT and SIDT are used only in operating system software; they are not used in application programs.

**Flags Affected:** None

**ALGORITHM:**

1. Display welcome message on terminal using macro disp.
2. Store most significant bit of CR0 in eax register.
3. Check the PE bit of CR0.
4. If PE=1 then display message "Processor is in Protected mode".
5. And if PE=0 then display message "Processor is in Real mode".
6. Then copies/stores the contents of GDT, IDT, LDT using sgdt, sidt, sldt instruction.
7. Display their contents using macro

**Questions-**

1. Explain System Address registers
2. Explain Segment selecters-LDTR and TR
3. Explain CR0 register

| | |
|---|---|
| **SUBJECT: MICROPROCESSOR LAB (MPL)** | |
| **NAME:** | |
| **CLASS:SE COMP** | **ROLL NO.:** |
| **SEMESTER:SEM-II** | **YEAR:2023-24** |
| **DATE OF PERFORMANCE:** | **DATE OF SUBMISSION:** |
| **EXAMINED:** | |

## Assignment No-06

**Title:-** Non-Overlapped block data transfer.

**Assignment Name: -** Write X86/64 ALP to perform non-overlapped block transfer without string specific instructions Block containing data can be defined in the data segment.

**Objective-**

- To study various instruction related to
  a) Arithmetic operations.
  b) Data transfer operations.
  c) Branch operations.
  d) String operations.
- To understand how to define block in data segment.

**Outcome-**

- Students will be able to write code for block data transfer.
- Students will be able to understand different assembly language instruction.

**Prerequisite -**

System call of Unix for Assembly language Program.

**Hardware Requirement-**

Desktop PC

**Software Requirement-**

Ubuntu 14.04,
Assembler: NASM version 2.10.07
Linker: ld

**Introduction:-**

**Guidelines for the algorithm:**

## NON-OVERLAPPED BLOCK DATA TRANSFER

1) Initialize Data section.
2) Define 2 arrays (5 members) for source and destination with different memory locations.
3) Initialize destination array with all zeros.
4) Take count N=5
5) Move the first element of source array to destination array.
6) Decrement count N.
7) Repeat step 5,6 till count N=0
8) Display both source and destination arrays.

**Conclusion:-** Hence we implemented an ALP to Non-overlapped block data transfer.

**Questions:-**

1. Explain Assembler directives
2. Explain E-Flag register

| | |
|---|---|
| **SUBJECT: MICROPROCESSOR LAB (MPL)** | |
| **NAME:** | |
| **CLASS:SE COMP** | **ROLL NO.:** |
| **SEMESTER:SEM-II** | **YEAR:2023-24** |
| **DATE OF PERFORMANCE:** | **DATE OF SUBMISSION:** |
| **EXAMINED:** | |

## Assignment No-07

**Title:-** Overlapped block data transfer.

**Assignment Name:** Write X86/64 ALP to perform overlapped block transfer with string specific instructions, Block containing data can be defined in the data segment.

**Objective-**

- To study various instruction related to
  a) Arithmetic operations.
  b) Data transfer operations.
  c) Branch operations.
  d) String operations.
- To understand how to define block in data segment.

**Outcome-**

- Students will be able to write code for block data transfer.
- Students will be able to understand different assembly language instruction.

**Prerequisite -**

System call of Unix for Assembly language Program.

**Hardware Requirement-**

Desktop PC

**Software Requirement-**

Ubuntu 14.04,
Assembler: NASM version 2.10.07
Linker: ld

**Introduction:-**

**Guidelines for the algorithm:**

OVERLAPPED BLOCK DATA TRANSFER

1) Initialize Data section.
2) Define source array & destination array with 5 numbers..
3) Initialize destination array with all zeros.
4) Copy source array to destination array as it is.
5) Take index from destination array from where you want to do overlapping.
6) Find value N.
7) Move the first element of source array to index location mention in previous step to destination array.
8) Decrement count N.
9) Repeat step 7,8 till count N=0
10) Display both source and destination arrays.

**Conclusion:-** Hence we implemented an ALP to overlapped and nonoverlapped block data transfer.

**Questions:-**

3. Explain String Specific Instruction?
4. Explain Stack manipulation instructions?

| SUBJECT: MICROPROCESSOR LAB (MPL) | |
|---|---|
| NAME: | |
| CLASS:SE COMP | ROLL NO.: |
| SEMESTER:SEM-II | YEAR:2023-24 |
| DATE OF PERFORMANCE: | DATE OF SUBMISSION: |
| EXAMINED: | |

## Assignment No-08

**Title:-**Multiplication

**Assignment Name: -** Write X86/64 ALP to perform multiplication of two 8-bit hexadecimal numbers. Use successive addition and add and shift method. Accept input from the user.

**Objective-**

- To understand the different algorithm for multiplication.
- To understand how to write procedure.

**Outcome-**

- Students will be able to write code for doing multiplication.

**Prerequisite -**

System call of Unix for Assembly language Program.

**Hardware Requirement-**

Desktop PC

**Software Requirement-**

Ubuntu 14.04,

Assembler: NASM version 2.10.07
Linker: ld

**Introduction:-**

**Guidelines for the algorithm:**

1) Display the menu.
   Enter "1" – "ADD AND SHIFT METHOD."
   Enter "2" – "SUCCESSIVE ADDITION METHOD".
   Enter "3" – EXIT

2) Take choice from user then go to the respective subroutines.

## ADD AND SHIFT METHOD

1) Initialize code and bss sections.
2) Accept multiplier and multiplicand variables in data segment.
3) Initialize product variable to zero.
4) Set count as number of bits in operand, which is 8.
5) Shift product to left by 1 bit and insert zero as LSB.
6) Transfer MSB of multiplier to carry flag by rotating it to left.
7) Check if carry flag is set or not. If yes add multiplicand to product.
8) Decrement count by 1.
9) Check count=0 else repeat step 5 through step 9 till count=0.
10) Display the final product.

## SUCCESSIVE ADDITION METHOD

1) Define product=0.
2) Set count=multiplicand.
3) Add product=product + multiplier.
4) Decrement count.
5) Repeat step 3 and 4 till count=0
6) Display product variable value as final product.

**Conclusion: -** Hence we implemented an ALP to do multiplication.

**Questions:-**

1) Explain ADD and SHIFT algorithm with example?
2) Explain what is Interrupt?

| SUBJECT: MICROPROCESSOR LAB (MPL) | |
|---|---|
| NAME: | |
| CLASS:SE COMP | ROLL NO.: |
| SEMESTER:SEM-II | YEAR:2023-24 |
| DATE OF PERFORMANCE: | DATE OF SUBMISSION: |
| EXAMINED: | |

## Assignment No-09

**Title:-** Find factorial of a given integer number on a command line by using recursion

**Assignment Name:-**Write x86 ALP to find the factorial of a given integer number on a command line by using recursion. Explicit stack manipulation is expected in the code.

**Objective-**

- To understand the assembly language program
- To understand the concept of recursion
- Able to Implement factorial of a integer number using recursive method

**Outcome-**

- Students will be able to understand different assembly language instruction.
- Students will be familiar with the format of assembly language program and able to Apply the concept of recursion to find factorial of a number

**Prerequisite -**

System call of Unix for Assembly language Program.

**Hardware Requirement-**

Desktop PC

**Software Requirement-**

Ubuntu 14.04,
Assembler: NASM version 2.10.07
Linker: ld

**Introduction:-**

**THEORY:**

A recursive procedure is one that calls itself. There are two kind of recursion: direct and indirect. In direct recursion, the procedure calls itself and in indirect recursion, the first procedure calls a

second procedure, which in turn calls the first procedure.

Recursion could be observed in numerous mathematical algorithms. For example, consider the case of calculating the factorial of a number. Factorial of a number is given by the equation −

```
Fact (n) = n * fact (n-1) for n > 0
```

For example: factorial of 5 is 1 x 2 x 3 x 4 x 5 = 5 x factorial of 4 and this can be a good example of showing a recursive procedure. Every recursive algorithm must have an ending condition, i.e., the recursive calling of the program should be stopped when a condition is fulfilled. In the case of factorial algorithm, the end condition is reached when n is 0.

**Instructions needed:**

1. AND-AND each bit in a byte or word with corresponding bit in another byte or word
2. INC-Increments specified byte/word by1
3. DEC-Decrements specified byte/word by1
4. JG - The command JG simply means: Jump if Greater.
5. CMP-Compares to specified bytes or words
6. MUL - The MUL (Multiply) instruction handles unsigned data
7. CALL-Transfers the control from callingprogramto procedure.
8. ADD- ADD instructions are used for performing simple addition of binary data in byte, word and doubleword size, i.e., for adding 8-bit, 16-bit or 32-bit operands, respectively.
9. RET-Return from where call is made

**Algorithm:-**

This algorithm use recursive approach to find factorial of N.

1. Start
2. Read: Take input N
3. Retrieve parameter and put it into Register-PUSH

4. Check for base case if n==0
5. move the first argument to %rax
6. If the number is 1, that is our base case, and we simply return.
7. multiply by the result of the last call to factorial.
8. return to the function

**Conclusion:-**

**Questions:-**

1. What is Control transfer instructions. Explain in details
2. What different conditions used to find factorial of an integer number.
3. Explain CALL, JG, ADD instructions
4. Explain Pop and Push instruction in detail.

| | |
|---|---|
| **SUBJECT: MICROPROCESSOR LAB (MPL)** | |
| **NAME:** | |
| **CLASS:SE COMP** | **ROLL NO.:** |
| **SEMESTER:SEM-II** | **YEAR:2023-24** |
| **DATE OF PERFORMANCE:** | **DATE OF SUBMISSION:** |
| **EXAMINED:** | |

## Assignment No-10

**Title:**- Study Assignment

**Assignment Name:-**

Motherboards are complex. Break them down, component by component, and Understand how they work. Choosing a motherboard is a hugely important part of building a PC. Study- Block diagram, Processor Socket, Expansion Slots, SATA, RAM, Form Factor, BIOS, Internal Connectors, External Ports, Peripherals and Data Transfer, Display, Audio

**Objective-**

- To understand the Form Factor of Motherboard
- To understand different components of Motherboard

**Outcome-**

- Students will be able to identify components of Motherboard
- Students will be able to distinguished south bridge and north bridge and its working

**Prerequisite -**

Computer Fundamental.

**Hardware Requirement-**

Desktop PC,Motherboard,SMPS

**Introduction:-**

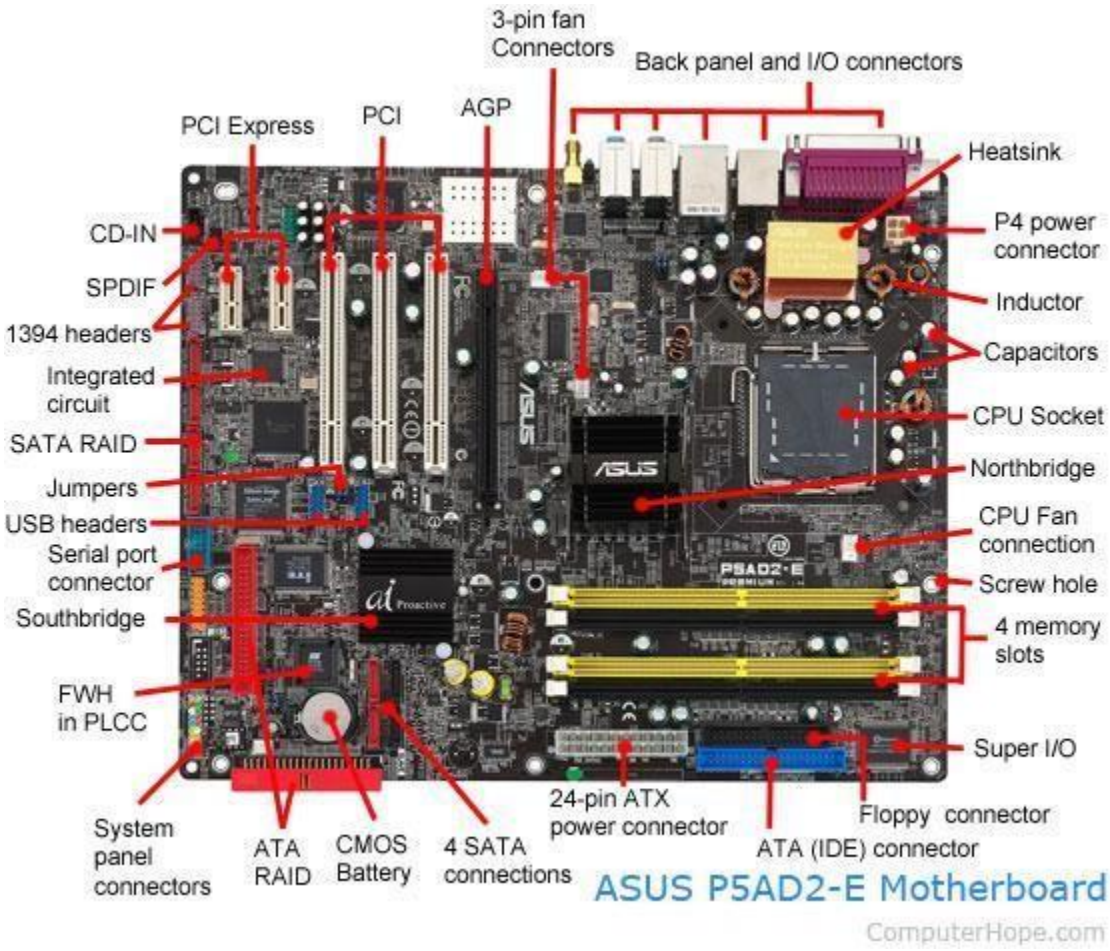
**THEORY:**

### Motherboard overview

A motherboard provides connectivity between the hardware components of a computer, like the processor (CPU), memory (RAM), hard drive, and video card. There are multiple types of motherboards, designed to fit different types and sizes of computers.

Each type of motherboard is designed to work with specific types of processors and memory, so they don't work with every processor and type of memory. However, hard drives are mostly universal and work with the majority of motherboards, regardless of the type or brand.

Below is a picture of the ASUS P5AD2-E motherboard with labels next to each of its major components. Clicking the image directs you to a larger and more detailed version.




## Where is the motherboard located?

A computer motherboard is located inside the computer case and is where most of the parts and computer peripherals connect. With tower computers, the motherboard is on the left or right side of the tower and is the biggest circuit board.

## Motherboard components

Below are links to pages with more details for each of the motherboard components mentioned in the previous section. The links are listed in clockwise order starting from the top-left corner of the image. Components not labeled on the image above are found in sections later on this page.

- Expansion slots (PCI Express, PCI, and AGP)
- 3-pin case fan connectors
- Back pane connectors
- Heat sink
- 4-pin (P4) power connector

- Inductor
- Capacitor
- CPU socket
- Northbridge
- Screw hole
- Memory slot
- Super I/O
- ATA / IDE disk drive primary connection
- 24-pin ATX power supply connector
- Serial ATA connections
- Coin cell battery (CMOS backup battery)
- RAID
- System panel connectors
- FWH
- Southbridge
- Serial port connector
- USB headers
- Jumpers
- Integrated circuit
- 1394 headers
- SPDIF
- CD-IN

### Older motherboard components

The following list contains links to components that are not shown in the picture above or were part of older computer motherboards.

- BIOS
- Bus
- Cache memory
- Chipset
- Diode
- Dip switches
- Electrolytic
- Floppy connection

**Questions**-List the different components of Motherboard