PUNE INSTITUTE OF COMPUTER TECHNOLOGY

Department of Computer Engineering

Academic Year: 2019-20

CLASS: S.E. **SEMESTER: II**

LAB MANUAL

SUBJECT: MICROPROCESSOR LAB (210257)

ASSIGNMENT NUMBER: 1

ASSINGMENT NO.	1
TITLE	To find positive and negative numbers.
PROBLEM STATEMENT /DEFINITION	Write X86/64 ALP to count number of positive and negative numbers from the array
OBJECTIVE	To understand how to identify a positive number or negative number.
OUTCOME	Students will be e able to use different index registers and find positive and negative numbers from array.
S/W PACKAGES AND HARDWARE APPARATUS USED	Core 2 duo/i3/i5/i7 - 64bit processor OS – Linux 64bit OS Editor- gedit /vi Assembler –NASM Debugger- GDB/TD
REFERENCES	1. "Microprocessor and Interfacing Techniques", Douglas Hall 2. "IBM PC Assembly language and programming", Peter Able 3. "Advances MS-DOS programming ", Ray Duncan

STEPS	Start.
	2. Initialize an array of 10 numbers or accept 10 numbers from user and store them in one array.
	3. Initialize pos_counter=0, neg_counter=0, index_reg=array address, counter=10
	2. Read the number from index_reg into a register.
	3. Perform addition with 00H and check sign bit
	4. If sign bit==1 then
	increment neg_counter=neg_counter+1
	else
	increment pos_counter=pos_counter+1
	end if
	5. Increment index_reg= index_reg+1
	6. Decrement counter=counter-1
	7. If counter!=0 then goto step number 4 else
	continue
	8. Print message "Positive numbers are:" and print pos_counter.
	9. Print message "Negative numbers are:" and print neg_counter.
	10. Exit.

INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 10. Conclusion/Analysis

Prerequisites: COA

Concepts related Theory:

Any number above zero is a positive number. Positive numbers are written with no sign or a + sign in front of them and they are counted from zero to the right. Any number **below zero** is a negative number. Negative numbers are written with a - sign in front of them and they are counted from zero to the left. Always look at the **sign** in front of a number to see if it is positive or negative.

In Computer Systems, the negative number is represented by different ways:

1. Sign-Magnitude representation: -

Negative numbers are essential and any computer not capable of dealing with them would not be particularly useful. But how can such numbers be represented? There are several methods which can be used to represent negative numbers in Binary. One of them is called the Sign-Magnitude Method.

The Sign-Magnitude Method is quite easy to understand. In fact, it is simply an ordinary binary number with one extra digit placed in front to represent the sign. If this extra digit is a '1', it means that the rest of the digits represent a negative number. However if the same set of digits are used but the extra digit is a '0', it means that the number is a positive one. The following examples explain the Sign-Magnitude method better. Let us assume that we have an 8-bit register. This means that we have 7 bits which represent a number and the other bit to represent the sign of the number (the **Sign Bit**). This is how numbers are represented:

0	0	1	0	0	1	0	1

The red digit means that the number is positive. The rest of the digits represent 37. Thus, the above number in sign-magnitude representation means +37. And this is how -37 is represented:

1	0	1	0	0	1	0	1

Binary	Value
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

Assign the leftmost (most significant) bit to be the sign bit. If the sign bit is 0, this means the number is positive. If the sign bit is 1, then the number is negative. The remaining m-1 bits are used to represent the magnitude of the binary number in the unsigned binary notation.

2. 1's Complement representation: -

The 1's complement of a binary number is the number that results when we change all 1s to zeros and the zeros to ones.

3. 2's Complement representation: -

Another common method of representing negative numbers in binary is the Twos Complement. The 2's complement is the binary number that results when we add 1 to the 1's complement. It is given as

2's complement = 1's complement + 1

e.g. 2's complement of $(11000100)_2$

1's complement of number = 00111011

Add 1 = 1

2's complement of number = 00111100

A complete binary table for four bits is shown below:

Binary	Two's Comp Value	Binary	Two' Comp Value
0000	0	1111	-1
0001	+1	1110	-2
0010	+2	1101	-3
0011	+3	1100	-4
0100	+4	1011	-5
0101	+5	1010	-6
0110	+6	1001	-7
0111	+7	1000	-8

- **Advantages of 2's complement representation: -**

1. There are distinct +0 and -0 representations in both the sign-magnitude and 1's complement systems, but the 2's complement system has only a +0 representation.

2. For 4 bit numbers, the value -8 is represent able only in the 2's complement system and not in other systems.

3. It is more efficient for logic circuit implementation, and one often used in computers for addition and subtraction operations.

Bit Test Instruction

Opcode	Mnemonic	Description
0F B3	BTR r/m16, r16	Store selected bit in CF flag and clear
0F B3	BTR r/m32, r32	Store selected bit in CF flag and clear
0F BA /6 ib	BTR r/m16, imm8	Store selected bit in CF flag and clear
0F BA /6 ib	BTR r/m32, imm8	Store selected bit in CF flag and clear

Description

Selects the bit in a bit string (specified with the first operand, called the bit base) at the bitposition designated by the bit offset operand (second operand), stores the value of the bit in the CF flag, and clears the selected bit in the bit string to 0. The bit base operand can be a register or a memory location; the bit offset operand can be a register or an immediate value. If the bit base operand specifies a register, the instruction takes the modulo 16 or 32 (depending on the register size) of the bit offset operand, allowing any bit position to be selected in a 16- or 32-bit register, respectively. If the bit base operand specifies a memory location, it represents the address of the byte in memory that contains the bit base (bit 0 of the specified byte) of the bit string. The offset operand then selects a bit position within the range -2^31 to 2^31 - 1 for a register offset and 0 to 31 for an immediate offset.

Some assemblers support immediate bit offsets larger than 31 by using the immediate bit offset field in combination with the displacement field of the memory operand. See "BT-Bit Test" in this chapter for more information on this addressing mechanism.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

Algorithm:

1. Start.
2. Initialize an array of 10 numbers or accept 10 numbers from user and store them in one array.
3. Initialize pos_counter=0, neg_counter=0, index_reg=array address, counter=10
4. Read the number from index_reg into a register.
5. Perform addition with 00H and check sign bit
6. If sign bit==1 then
   increment neg_counter=neg_counter+1
   else
   increment pos_counter=pos_counter+1
   end if
7. Increment index_reg= index_reg+1
8. Decrement counter=counter-1
9. If counter!=0 then goto step number 4 else

continue
10. Print message “Positive numbers are:” and print pos_counter.
11. Print message “Negative numbers are:” and print neg_counter.
12. Exit.

Conclusion:

We are able to use different index registers and find positive and negative numbers from array.

Review Questions:

1. Defined various signed data types in 80386.
2. Explain BT and BTR instruction with example.
3. Explain basic structure of ALP.
4. Write '-93' in sign-magnitude representation, using an 8-bit register.
5. What is the largest positive number which can be represented in an 8-bit register, using the sign-magnitude method of representation?
6. What is the largest negative number which can be represented in an 8-bit register, using the sign-magnitude method of representation?
7. Give the range of possible numbers which can be represented in a 16-bit register, using the sign-magnitude method.

ASSIGNMENT NUMBER:2

ASSINGMENT NO.	2
TITLE	Data block Transfer
PROBLEM STATEMENT /DEFINITION	Write x86 ALP to perform non-overlapped and overlapped block transfer (with and without string specific instructions). Block containing data can be defined in the data segment.
OBJECTIVE	To learn • Overlapped / Non – overlapped data transfer in segments • Block transfer instruction of 8086 • Data storage in the memory and segments
OUTCOME	Students will study different block transfer instructions and also understood block transfer within different segments.
S/W PACKAGES AND HARDWARE APPARATUS USED	Core 2 duo/i3/i5/i7 - 64bit processor OS – Linux 64bit OS Editor- gedit /vi Assembler –NASM Debugger- GDB/TD
REFERENCES	1. "Microprocessor and Interfacing Techniques", Douglas Hall 2. "IBM PC Assembly language and programming", Peter Able **3.** "Advances MS-DOS programming ", Ray Duncan
STEPS	**A] Overlapping** 1. Study system call to read and display character on the screen. 2. Accept the Value of 'N' i.e. how many numbers to add 3. initialize Sum =0 4. Read a number (two digit) 5. add it to sum

	6. repeat the steps 4 and 5 to add all N numbers 7. Print the result /Sum **8.** End. **B] Non Overlapping** 1. Declare a source array. 2. Load the address of source array in one of the registers. (Index register) 3. Read the first byte from the source array. 4. Increment the pointer/SI by length of array which becomes the starting Address of destination array. 5. Move the source element to the destination address. 6. In case of overlapping mode based on the destination address either move the first Element or last element in beginning of transfer operation.
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 10. Conclusion/Analysis

Prerequisites: Instruction set of 80386

Concepts related Theory: One of the frequent operations used in programming is shifting/

transferring the data form one memory location to another memory location. These operations can be with simple mov instructions which may result in more number of operations. We can make use of instructions like MOVSB to transfer the data. The relevant instructions are LOOP / MOVSB.

The data can be transferred either in overlapped fashion or non overlapped Fashion. In case of overlapped address the two possible situations are for the Source address to be greater than the destination address in which case the first element in the source is to be moved first or for the source address to be less than the destination address which requires the last element of the source to be moved first.

Algorithm: A] Overlapping

1. Study system call to read and display character on the screen.
2. Accept the Value of 'N' i.e. how many numbers to add
3. initialize Sum =0
4. Read a number (two digit)
5. add it to sum
6. repeat the steps 4 and 5 to add all N numbers
7. Print the result /Sum
8. End.

B] Non Overlapping

1. Declare a source array.

2. Load the address of source array in one of the registers. (Index register)

3. Read the first byte from the source array.

4. Increment the pointer/SI by length of array which becomes the starting Address of destination array.

5. Move the source element to the destination address.

6. In case of overlapping mode based on the destination address either move the first Element or last element in beginning of transfer operation.

Conclusion:

We have studied different block transfer instructions and also understood block transfer within different segments.

Review Questions:

1. Explain various block transfer instruction with examples.
2. Explain use of index registers in ALP.
3. Explain different string instructions in 80386.
4. What are the different addressing modes?
5. Explain the addressing mode with suitable example.

ASSIGNMENT NUMBER: 3

ASSINGMENT NO.	3
TITLE	Number Conversion
PROBLEM STATEMENT /DEFINITION	Write 64 bit ALP to convert 4-digit Hex number into its equivalent BCD number and 5-digit BCD number into its equivalent HEX number. Make your program user friendly to accept the choice from user for: a) HEX to BCD b) BCD to HEX c) EXIT
OBJECTIVE	To learn • Data representation and conversion • Understand the stack operations
OUTCOME	Students will studuy use of stack operations and number conversion in ALP.
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **Editor:** gedit/vi **Assembler:** NASM **Debugger:** GDB
REFERENCES	1. Douglas Hall, "Microprocessors & Interfacing", McGraw Hill, Revised 2nd Edition, 2006 ISBN 0-07-100462-9 **2.** A.Ray, K.Bhurchandi "Advanced Microprocessors and peripherals: Arch, Programming & Interfacing"
STEPS	1. Start. 2. Take i/p as hex to bcd or bcd to hex. 3. Convert input to ASCII. 4. Convert i/p to hex or bcd as required using function. 5. Convert answer to ASCII to display

	6. Display answer 7. Exit
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 10. Conclusion/Analysis

Prerequisites: COA, DEL

Concepts related Theory:

The different numbers systems can be used with the computer such as Hexadecimal, Octal, Binary and BCD. The number can be converted to any number system from any source other number system.

For example, in applications such as frequency counters, digital voltmeters or calculator where the output is a decimal display, a Binary Coded Decimal (BCD) is often used. BCD uses bit binary code to individually represent each decimal digit in a number.

Decimal	5	2	9
BCD	0101	0010	1001

Example:

HEX number: 1234

0A) 1234(1D2	0A) 1D2 (2E	0A) 24(3	0A) 4(0
1234	1CC	1E	0
------	------	----	--
0	6	6	4
Push (0)	Push (6)	Push (6)	Push (4)

4
6
6
0

Pop from stack and display result = 4660

BCD to HEX

- Initialize sum = 0
- Accept the first digit multiplicand
- Multiply the number by 10,000 multiplier
- Add result to sum
- Divide the multiplier by 10
- Repeat the step 2 to 4 till multiplier becomes zero
- Display the sum which is result of converting BCD number to HEX

Example

BCD number =4660

4*1000 + 6*100 + 6*10 + 6*0 = 1234H

FA0 + 258 +3C + 0 =1234H

Algorithm:

1. Start.
2. Take i/p as hex to bcd or bcd to hex.
3. Convert input to ASCII.
4. Convert i/p to hex or bcd as required using function.
5. Convert answer to ASCII to display
6. Display answer
7. Exit

Conclusion:

We have studied use of stack operations and number conversion in ALP.

Review Questions:

1. What is ASCII format?
2. Explain Hex to BCD and BCD to Hex conversion with example.
3. Explain MUL/IMUL instruction
4. Explain DIV/IDIV instruction

ASSIGNMENT NUMBER: 4

ASSINGMENT NO.	4
TITLE	Multiplication of two digit Numbers.
PROBLEM STATEMENT /DEFINITION	Write X86/64 ALP to perform multiplication of two 8-bit hexadecimal numbers. Use successive addition and add and shift method. (use of 64-bit registers is expected)
OBJECTIVE	To understand following multiplication techniques in ALP. 1. Successive Addition 2. Add and Shift Method.
OUTCOME	Students will be able to do multiplication in ALP.
S/W PACKAGES AND HARDWARE APPARATUS USED	Core 2 duo/i3/i5/i7 - 64bit processor OS – Linux 64bit OS Editor- gedit /vi Assembler –NASM Debugger- GDB/TD
REFERENCES	4. "Microprocessor and Interfacing Techniques", Douglas Hall 5. "IBM PC Assembly language and programming", Peter Able 6. "Advances MS-DOS programming ", Ray Duncan
STEPS	Successive Addition: 1. Start 2. Accept two 2-digit numbers. (Multiplier and Multiplicand) 3. Set Multiplicand value as a counter value. 4. Add Multiplier with itself "Counter-1" number of times. 5. Print The answer. Add and Shift Method

	1. Start 2. Accept two 2-digit numbers. (Multiplier and Multiplicand) 3. Store multiplier to BL and Multiplicand to CL, Initialize AX with 00. 4. Shift BL to left by 1 bit. (Shifted bit will be stored to carry flag) 5. If carry flag is set, Add CL to AL, and shift AL to left by 1 bit. 6. If carry flag is reset, shift AL to left by 1 bit. 7. Repeat step 4 to 6 for 8 times. (As 2 digit numbers contains 8 bits) 8. Print the result from AX. 9. Stop
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 10. Conclusion/Analysis

Prerequisites: COA

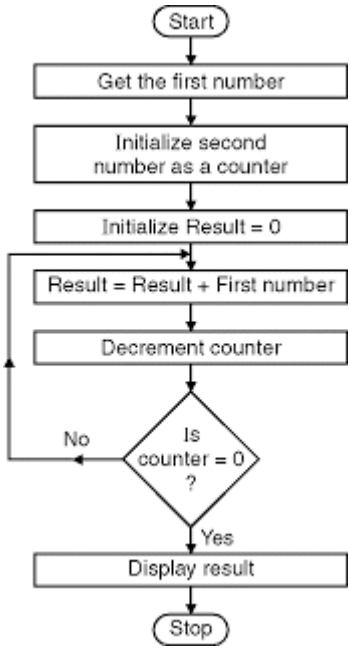
Concepts related Theory:

Successive Addition Method:

Consider that a byte is present in the AL register and second byte is present in the BL register.

- We have to multiply the byte in AL with the byte in BL.
- We will multiply the numbers using successive addition method.
- In successive addition method, one number is accepted and other number is taken as a counter. The first number is added with itself, till the counter decrements to zero.
- Result is stored in DX register. Display the result, using display routine.

- For example : AL = 12 H, BL = 10 H
  - Result = 12H + 12H + 12H + 12H + 12H + 12H + 12H + 12H + 12H + 12H
  - Result = 0120 H



Add and Shift Method:

Consider that one byte is present in the AL register and another byte is present in the BL register. We have to multiply the byte in AL with the byte in BL.
We will multiply the numbers using add and shift method. In this method, you add number with itself and rotate the other number each time and shift it by one bit to left alongwith carry. If carry is present add the two numbers.
Initialize the count to 4 as we are scanning for 4 digits. Decrement counter each time the bits are added. The result is stored in AX. Display the result.

For example : AL = 11 H, BL = 10 H, Count = 4

Step I :

AX = 11
\+ 11
22 H

Rotate BL by one bit to left along with carry.

BL = 10 H 0 ¬ 0 0 0 1 0 0 0 0

CY
BL = 0 0 0 1 0 0 0 0 0
CY 2 0

Step II : Now decrement counter count = 3.

Check for carry, carry is not there so add number with itself.

AX = 22
\+ 22

44 H

Rotate BL to left,

BL = 0 ¬0	100	0000
CY	4	0

Carry is not there.

Decrement count, count=2

Step III : Add number with itself

AX = 44
+ 44
88 H

Rotate BL to left,

BL = 0 1	000	0000
CY	8	0

Carry is not there.

Step IV : Decrement counter count = 1.

Add number with itself as carry is not there.

AX = 88
+ 88
110 H

Rotate BL to left,

BL = 1	0000	0000
CY	0	0

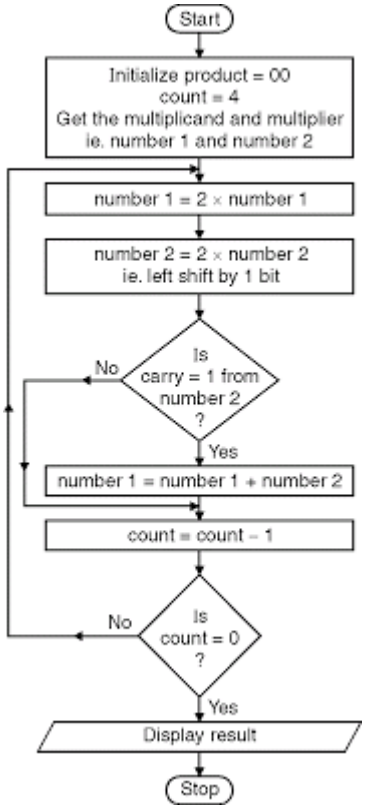
Carry is there.

Step V : Decrement counter = 0.

Carry is present.

\ add AX, BX

\ 0110 i.e. 11 H
+ 0000 ´ 10 H
0110 H 0110 H



Algorithm:

Successive Addition:

1. Start
2. Accept two 2-digit numbers. (Multiplier and Multiplicand)
3. Set Multiplicand value as a counter value.
4. Add Multiplier with itself “Counter-1” number of times.
5. Print The answer.

Add and Shift Method

1. Start
2. Accept two 2-digit numbers. (Multiplier and Multiplicand)
3. Store multiplier to BL and Multiplicand to CL, Initialize AX with 00.
4. Shift BL to left by 1 bit. (Shifted bit will be stored to carry flag)
5. If carry flag is set, Add CL to AL, and shift AL to left by 1 bit.
6. If carry flag is reset, shift AL to left by 1 bit.
7. Repeat step 4 to 6 for 8 times. (As 2 digit numbers contains 8

bits)

8. Print the result from AX.
9. Stop

Conclusion:

We have studied following multiplication techniques in ALP.

1. Successive Addition
2. Add and Shift Method.

Review Questions:

1. **Explain different multiplication techniques in ALP.**
2. **Explain booth's algorithm.**

ASSIGNMENT NUMBER: 5

ASSINGMENT NO.	5
TITLE	Write X86 ALP to find, a) Number of Blank spaces b) Number of lines c) Occurrence of a particular character
PROBLEM STATEMENT /DEFINITION	Write X86 ALP to find, a) Number of Blank spaces b) Number of lines c) Occurrence of a particular character. Accept the data from the text file. The text file has to be accessed during Program_1 execution and write FAR PROCEDURES in Program_2 for the rest of the processing. Use of GLOBAL and EXTERN directives is mandatory.
OBJECTIVE	To understand how to implement NEAR and FAR procedure.
OUTCOME	Students will study NEAR and FAR procedure and there application.
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **Editor:** gedit/vi **Assembler:** NASM **Debugger:** GDB
REFERENCES	➢ "The Intel microprocessor", Barry B. Brey. ➢ "Introduction to 64 bit Intel Assembly Language Programming for Linux", 2nd Edition, Ray Seyfarth, ➢ "80386 Microprocessor Handbook", Chris H. Papas.

STEPS	**Create a one Folder which contain 2 program file and 1 text file.** Write all input text to the text file. Define all parameter required to execute above procedure in 1st program file. Define all the procedure (required to count and print the number of blank space,procedure required to count and print the number of ENTER in text file and procedure required to count number of occurrences of given character in text file) in 2nd program file Using GLOBAL keyword. Declare a file name in section .data Open a file using OPEN system call. Check the descriptor value of open file.(if value is negative means file is not open and if value is positive means file is open) Save the file descriptor for further use. Read the content of text file using read system call To find the Number of Spaces from the text file compare each entry from the text with 20H (ASCII Value of space) ,count the value and print. To find the Number of Enter from the text file compare each entry from the text with 0x0A (ASCII Value of enter) ,count the value and print. Accept the letter whose number of occurrences find in the text file. To find the Number of occurrences from the text file compare each entry from the text with ASCII Value of entered text,count the value and print. Close the file. Exit.

INSTRUCTIONS FOR WRITING JOURNAL	1. Title 2. Problem Definition 3. Objective: Intention behind study 4. Software & Hardware requirements 5. Explanation of the assignment 6. Algorithm 7. State Diagram 8. Test cases for program. 9. Print outs 10. Conclusion.

Assignment No. 5

- **Aim:** Write X-86 ALP to find, a) Number of Blank spaces b) Number of lines c) Occurrence of a particular character. Accept the data from the text file. The text file has to be accessed during Program_1 execution and write FAR PROCEDURES in Program_2 for the rest of the processing. Use of GLOBAL and EXTERN directives is mandatory.
- **Prerequisites :** Program Transfer control instruction, File system
- **Concept related Theory:**
    - **OPEN File**

```
mov rax, 2          ; 'open' syscall
mov rdi, fname1   ; file name
mov rsi, 2          ;  File access mode
mov rdx, 0777      ; permissions set
Syscall
mov [fd_in], rax
```

    - **READ File**

```
mov rax, 0          ; ‘Read' syscall
```

mov rdi, [fd_in] ; file Pointer

mov rsi, Buffer ; Buffer for read

mov rdx, length ; len of data want to read

Syscall

- **WRITE File**

mov rax, 01 ; 'Write' syscall

mov rdi, [fd_in] ; file Pointer

mov rsi, Buffer ; Buffer for write

mov rdx, length ; len of data want to read

Syscall

- **CLOSE File**

mov rax,3

mov rdi,[fd_in]

syscall

- **Conclusion**

  We have studied Near and Far process and there application.

- **Review Questions:**

  1. Difference between Near and Far procedure.
  2. Explain GLOBAL and Extern Assembler Directives.
  3. What is difference between procedure and Macro?
  4. What is difference between procedure and interrupt?
  5. How to call a procedure?

ASSIGNMENT NUMBER: 6

TITLE	**To read and display contents pointed by GDTR, LDTR and IDTR.**
PROBLEM STATEMENT /DEFINITION	**Write an ALP to read and display the table content pointed by GDTR/LDTR and IDTR**
OBJECTIVE	To understand how to read and display contents of GDTR, LDTR and IDTR registers.
OUTCOME	Students will study different Descriptor tables in system also different registers associated with it.
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **Editor:** gedit/vi **Assembler:** NASM **Debugger:** GDB
REFERENCES	➢ "The Intel microprocessor", Barry B. Brey. ➢ "Introduction to 64 bit Intel Assembly Language Programming for Linux", 2nd Edition, Ray Seyfarth, ➢ "80386 Microprocessor Handbook", Chris H. Papas.
STEPS	**1.** Start **2.** Store the value of GDTR,IDTR and LDTR in respective variable. 3. Display the result 4. Stop

INSTRUCTIONS FOR WRITING JOURNAL	1. Title 2. Problem Definition 3. Objective: Intention behind study 4. Software & Hardware requirements 5. Explanation of the assignment 6. Algorithm 7. State Diagram 8. Test cases for program. **9.** Print outs **10.** Conclusion.

Assignment No. 6

- **Aim: To display the contents pointed by System Address Registers**
- **Prerequisites :** 80386 Architecture
- **Concept related Theory:**

GDTR and IDTR

These registers hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

The GDT and IDT segments, since they are global to all tasks in the system, are defined by 32-bit linear addresses (subject to page translation if paging is enabled) and 16-bit limit values.

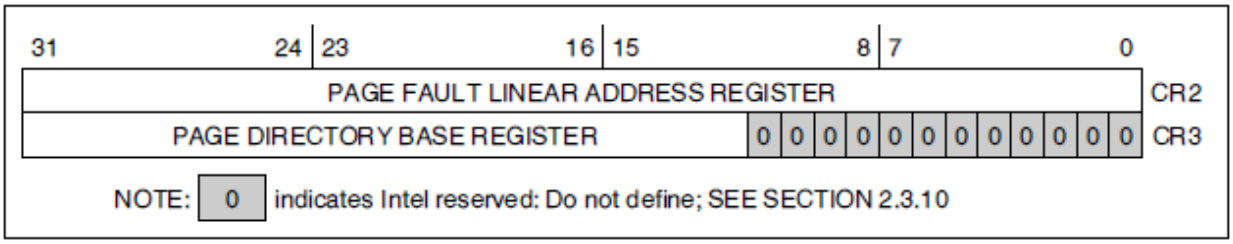


Figure 1. Control Registers 2 and 3

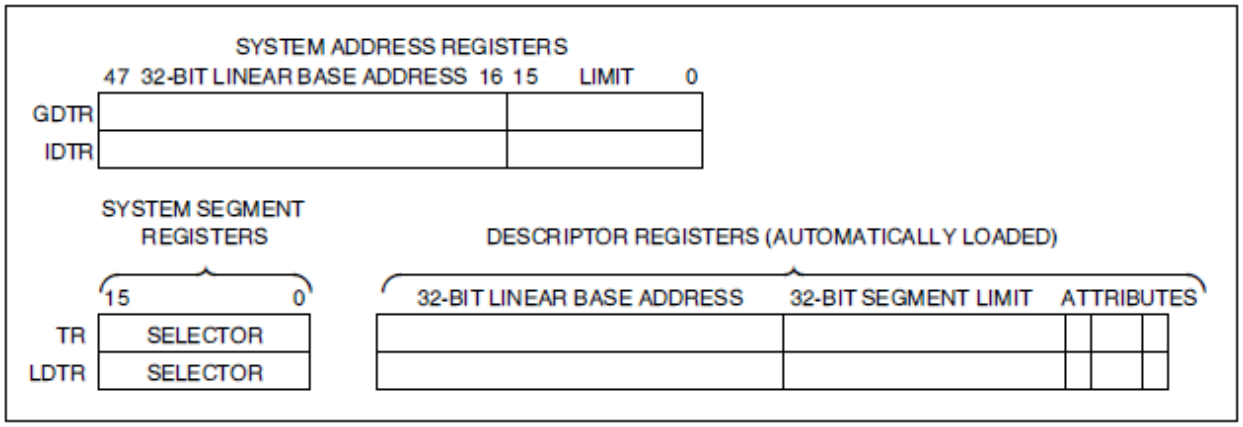


Figure2. System Address and System Segment Registers

LDTR and TR

These registers hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

The LDT and TSS segments, since they are task specific segments, are defined by selector values stored in the system segment registers. Note that a segment descriptor register (programmer-invisible) is associated with each system segment register.

- **Conclusion**

  We have studied different Descriptor tables in system also different registers associated with it.

- **Review Questions:**

  1. What are the different descriptor tables of 80386 processors?
  2. What is Selector?
  3. Explain the descriptor format.
  4. Explain descriptor cache.
  5. What are the types of Descriptors?
  6. What is size of GDTR, LDTR IDTR,TR?
  7. How to get the base address of GDT/LDT/IDT?

ASSIGNMENT NUMBER : 7

TITLE	Write X86 program to sort the list of integers in ascending/descending order. Read the input from the text file and write the sorted data back to the same text file using bubble sort
PROBLEM STATEMENT /DEFINITION	Write X86 program to sort the list of integers in ascending/descending order. Read the input from the text file and write the sorted data back to the same text file using bubble sort .
OBJECTIVE	To understand how to implement Bubble sort using ALP.
OUTCOME	Students will study sorting of number in ALP.
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **Editor:** gedit/vi **Assembler:** NASM **Debugger:** GDB
REFERENCES	➢ "The Intel microprocessor", Barry B. Brey. ➢ "Introduction to 64 bit Intel Assembly Language Programming for Linux", 2nd Edition, Ray Seyfarth, ➢ "80386 Microprocessor Handbook", Chris H. Papas.
STEPS	1. Create a text file which contain the single digit number. 2. Create one assembly language program file which contain the following step. 3. Open the text file and check that is it successfully opened or not. 4. Read the content of file and store it in buffer. 5. Sort the contain of buffer using bubble sort method. 6. write the content of buffer in the text file. 7. close the file.

INSTRUCTIONS FOR **WRITING JOURNAL**	1. Title 2. Problem Definition 3. Objective: Intention behind study 4. Software & Hardware requirements 5. Explanation of the assignment 6. Algorithm 7. State Diagram 8. Test cases for program. 9. Print outs 10. Conclusion.

Assignment No. 7

- **Aim:** Write X86 program to sort the list of integers in ascending/descending order. Read the input from the text file and write the sorted data back to the same text file using bubble sort

- **Prerequisites : Data Structure, Instruction set 80386,File System**
- **Concept related Theory:**
    - **OPEN File**

```
mov rax, 2          ; 'open' syscall
mov rdi, fname1   ; file name
mov rsi, 2          ;  File access mode
mov rdx, 0777      ; permissions set
Syscall
mov [fd_in], rax
```

- **READ File**

```
mov rax, 0          ; ‘Read' syscall
```

mov rdi, [fd_in] ; file Pointer

mov rsi, Buffer ; Buffer for read

mov rdx, length ; len of data want to read

Syscall

- **WRITE File**

mov rax, 01 ; 'Write' syscall

mov rdi, [fd_in] ; file Pointer

mov rsi, Buffer ; Buffer for write

mov rdx, length ; len of data want to read

Syscall

- **CLOSE File**

mov rax,3

mov rdi,[fd_in]

syscall

- **Conclusion**

  We have studied how to implement bubble sort using ALP.

- **Review Questions**

ASSIGNMENT NUMBER : 8

TITLE	**Write X86 menu driven Assembly Language Program (ALP) to implement OS (DOS) commands TYPE, COPY and DELETE using file operations. User is supposed to provide command line arguments in all cases.**
PROBLEM STATEMENT /DEFINITION	**Write X86 menu driven Assembly Language Program (ALP) to implement OS (DOS) commands TYPE, COPY and DELETE using file operations. User is supposed to provide command line arguments in all cases.**
OBJECTIVE	To understand how to implement OS (DOS) commands using file operations.
OUTCOME	Students will study different DOS commands and file operations.
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **Editor:** gedit/vi **Assembler:** NASM **Debugger:** GDB
REFERENCES	➢ "The Intel microprocessor", Barry B. Brey. ➢ "Introduction to 64 bit Intel Assembly Language Programming for Linux", 2nd Edition, Ray Seyfarth, ➢ "80386 Microprocessor Handbook", Chris H. Papas.

STEPS	**DOS COPY Command** **1.** Start **2.** pop no. of arguments, exec arguments and source filename. **3.** Save source file name. **4.** Pop destination filename. **5.** Save destination filename. **6.** Open the file for reading. **7.** Open or create the file for writing. **8.** Close source file. **9.** Close destination file. 10. Exit. **DOS DELETE Command** 1. Start 2. pop no. of arguments, exec arguments and source filename. 3. Save source file name. 4. Delete file 5. Exit. **DOS TYPE Command** 1. Start 2. pop no. of arguments, exec arguments and source filename. 3. Save source file name. 4. Open the file for reading. 5. Write to the terminal 6. Close source file 7. Exit.

INSTRUCTIONS FOR WRITING JOURNAL	1. Title 2. Problem Definition 3. Objective: Intention behind study 4. Software & Hardware requirements 5. Explanation of the assignment 6. Algorithm 7. State Diagram 8. Test cases for program. **9.** Print outs **10.** Conclusion.

Assignment No. 8

- **Aim:** Write X86 menu driven Assembly Language Program (ALP) to implement OS (DOS) commands TYPE, COPY and DELETE using file operations. User is supposed to provide command line arguments in all cases.
- **Prerequisites :** Linux commands, Command Line
- **Concept related Theory:**
    - OPEN File

```
mov rax, 2          ; 'open' syscall
mov rdi, fname1   ; file name
mov rsi, 0          ;
mov rdx, 0777       ; permissions set
Syscall
mov [fd_in], rax
```

- OPEN File/Create file

```
mov rax, 2          ; 'open' syscall
mov rdi, fname1   ; file name
mov rsi, 0102o      ; read and write mode, create if not
```

```
mov rdx, 0666o       ; permissions set
Syscall
mov [fd_in], rax
```

- READ File

```
mov rax, 0           ; ‘Read' syscall
mov rdi, [fd_in]     ; file Pointer
mov rsi, Buffer      ; Buffer for read
mov rdx, length      ; len of data want to read
Syscall
```

- WRITE File

```
mov rax, 01          ; ‘Write' syscall
mov rdi, [fd_in]     ; file Pointer
mov rsi, Buffer      ; Buffer for write
mov rdx, length      ; len of data want to read
Syscall
```

- DELETE File

```
mov rax,87
mov rdi,Fname
syscall
```

- CLOSE File

```
mov rax,3
mov rdi,[fd_in]
syscall
```

TYPE Command:

- Open file in read mode using open interrupt.

- Read contents of file using read interrupt.
- Display contents of file using write interrupt.
- **Close file using close interrupt**

COPY Command:

- Open file in read mode using open interrupt.
- Read contents of file using read interrupt.
- Create another file using read interrupt change only attributes.
- Open another file using open interrupt.
- Write contents of buffer into opened file.
- Close both files using close interrupt.

DELETE Command:

- DELETE file using delete interrupt.

- **Conclusion**

  We have studied different DOS commands and file operations.

- **Review Questions:**

  1. What are the different DOS commands?
  2. How to provide read, write and execute permission?
  3. How to write to a file?
  4. How to read from a file?
  5. What is the return value of Open system call?
  6. What is the return value of Read system call?

ASSIGNMENT NUMBER: 9

TITLE	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.
PROBLEM STATEMENT /DEFINITION	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 how to use stack segment for recursion.
OUTCOME	Students will study recursion using stacki n ALP.
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **Editor:** gedit/vi **Assembler:** NASM **Debugger:** GDB
REFERENCES	• "The Intel microprocessor", Barry B. Brey. • "Introduction to 64 bit Intel Assembly Language Programming for Linux", 2nd Edition, Ray Seyfarth, • "80386 Microprocessor Handbook", Chris H. Papas.

STEPS	➢ Start. ➢ Accept the number from user. ➢ Convert that number into Hexadecimal(ASCII TO HEX). ➢ Compare accepted number with 1,If it is equal to 1 go to step 5,else push the number on stack and decrement the number and go to step 4 ➢ pop the content of stack and multiply with number ➢ repeat the step until stack become empty. ➢ Convert the number from HEX to ASCII ➢ Print the number. ➢ End.
INSTRUCTIONS FOR WRITING JOURNAL	• Title • Problem Definition • Objective: Intention behind study • Software & Hardware requirements • Explanation of the assignment • Algorithm • State Diagram • Test cases for program. • Print outs • Conclusion.

Assignment No. 9

- **Aim:** 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.
- **Prerequisites : COA**
- **Concept related Theory:**

PUSH -- Push Operand onto the Stack

PUSH decrements the stack pointer by 2 if the operand-size attribute of the instruction is 16 bits; otherwise, it decrements the stack pointer by 4. PUSH then places the operand on the new top of stack, which is pointed to by the stack pointer.

The 80386 PUSH eSP instruction pushes the value of eSP as it existed before the instruction. This differs from the 8086, where PUSH SP pushes the new value (decremented by 2).

Opcode	Instruction	Clocks	Description
FF /6	PUSH m16	5	Push memory word
FF /6	PUSH m32	5	Push memory dword
50 + /r	PUSH r16	2	Push register word
50 + /r	PUSH r32	2	Push register dword
6A	PUSH imm8	2	Push immediate byte
68	PUSH imm16	2	Push immediate word
68	PUSH imm32	2	Push immediate dword
0E	PUSH CS	2	Push CS
16	PUSH SS	2	Push SS
1E	PUSH DS	2	Push DS
06	PUSH ES	2	Push ES
0F A0	PUSH FS	2	Push FS
OF A8	PUSH GS	2	Push GS

POP -- Pop a Word from the Stack

POP replaces the previous contents of the memory, the register, or the segment register operand with the word on the top of the 80386 stack, addressed by SS:SP (address-size attribute of 16 bits) or SS:ESP (addresssize attribute of 32 bits). The stack pointer SP is incremented by 2 for an operand-size of 16 bits or by 4 for an operand-size of 32 bits. It then points to the new top of stack.

Opcode	Instruction	Clocks	Description
8F /0	POP m16	5	Pop top of stack into memory word
8F /0	POP m32	5	Pop top of stack into memory dword
58 + rw	POP r16	4	Pop top of stack into word register
58 + rd	POP r32	4	Pop top of stack into dword register
1F	POP DS	7,pm=21	Pop top of stack into DS
07	POP ES	7,pm=21	Pop top of stack into ES
17	POP SS	7,pm=21	Pop top of stack into SS
0F A1	POP FS	7,pm=21	Pop top of stack into FS
0F A9	POP GS	7,pm=21	Pop top of stack into GS

11. **Conclusion: I**n this way we studied to use stack segment for recursion.

ASSIGNMENT NUMBER: 10

ASSINGMENT NO.	10
TITLE	Write 80387 ALP to find the roots of the quadratic equation. All the possible cases must be considered in calculating the roots.
PROBLEM STATEMENT /DEFINITION	Write 80387 ALP to find the roots of the quadratic equation. All the possible cases must be considered in calculating the roots.
OBJECTIVE	To be able to solve mathematical problems in Assembly language programming
OUTCOME	Students will be efficient in handling and solving mathematical problems using ALP
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **Editor:** gedit/vi **Assembler:** NASM **Debugger:** GDB
REFERENCES	➢ "The Intel microprocessor", Barry B. Brey. ➢ "Introduction to 64 bit Intel Assembly Language Programming for Linux", 2nd Edition, Ray Seyfarth, "80386 Microprocessor Handbook", Chris H. Papas.
STEPS	1. Write msg to enter the Quadratic equation 2. Write msg to enter first coefficient a 3. Read first coefficient 4. Convert the no from ascii to hex 5. Write msg to enter second coefficient b 6. Read second coefficient 7. Convert the no from ascii to hex 8. Write msg to enter third coefficient c 9. Read third coefficient 10. Convert the no from ascii to hex 11. Write down the quadratic formula $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ 12. Identify the values of a, b, and c in the quadratic equation.

	The variable *a* is the coefficient of the x^2 term, *b* is the coefficient of the x term, and *c* is the constant. For the equation $3x^2$ -5x - 8 = 0, a = 3, b = -5, and c = -8. Write this down. 13. Substitute the values of a, b, and c into the equation. Now that you know the values of the three variables, you can just plug them into the equation like this: a. {-b +/-√ (b^2 - 4ac)}/2 b. {-(-5) +/-√ ($(-5)^2$ - 4(3)(-8))}/2(3) = c. {-(-5) +/-√ ($(-5)^2$ - (-96))}/2(3) 14. Do the math. After you've plugged in the a. numbers, do the remaining math to simplify b. positive or negative signs, multiply, or square the c. remaining terms. Here's how you do it: d. {-(-5) +/-√ ($(-5)^2$ - (-96))}/2(3) = e. {5 +/-√(25 + 96)}/6 f. {5 +/-√(121)}/6 15. Simplify the square root**.** If the number under the radical symbol is a perfect square, you will get a whole number. If the number is not a perfect square, then simplify to its simplest radical version. If the number is negative, *and you're sure it's supposed to be negative*, then the roots will be complex. In this example, √(121) = 11. You can write that x = (5 +/- 11)/6. 16. Convert the final roots from hex to ascii 17. Print the final roots. Real and imaginary.
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 10. Conclusion/Analysis

Assignment No. 10

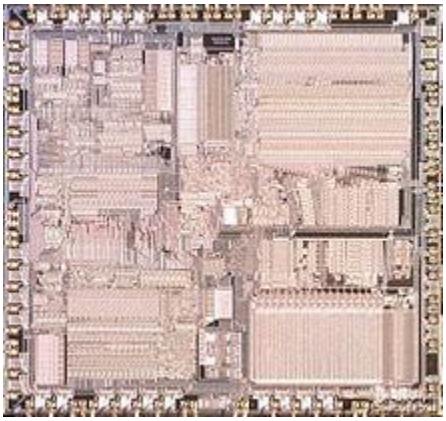
Aim: Write 80387 ALP to find the roots of the quadratic equation. All the possible cases must be considered in calculating the roots.

Prerequisites: FPU instruction set

Concepts related Theory:

80387 Microprocessor:

The **80387** (**387** or **i387**) is the first Intel coprocessor to be fully compliant with the IEEE 754-1985 standard. Released in 1987, a full two years after the 386 chip, the i387 includes much improved speed over Intel's previous 8087/80287 coprocessors, and improved characteristics of its trigonometric functions. The 8087 and 80287's FPTAN and FPATAN instructions are limited to an argument in the range $\pm\pi/4$ ($\pm 45°$), and the 8087 and 80287 have no *direct* instructions for the sin and cos functions.



Intel 80387 CPU Die Image

Without a coprocessor, the 386 normally performs floating-point arithmetic through (slow) software routines, implemented at runtime through a software exception-handler. When a math coprocessor is paired with the 386, the coprocessor performs the floating point arithmetic in hardware, returning results much faster than an (emulating) software library call.

The i387 is compatible only with the standard i386 chip, which has a 32-bit processor bus. The later cost-reduced i386SX, which has a narrower 16-bit data bus, can not interface with the i387's 32-bit bus. The i386SX requires its own coprocessor, the 80387SX, which is compatible with the SX's narrower 16-bit data bus.

Instruction set:

All instructions of 80386 MOV, JC,JNC,JG,JZ,JNZ,PUSH POP,INC DEC,CMP, ADD ETC are used in this program.

Algorithm:

1. Write msg to enter the Quadratic equation
2. Write msg to enter first coefficient a
3. Read first coefficient
4. Convert the no from ascii to hex
5. Write msg to enter second coefficient b
6. Read second coefficient
7. Convert the no from ascii to hex
8. Write msg to enter third coefficient c
9. Read third coefficient
10. Convert the no from ascii to hex
11. Write down the quadratic formula $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$
12. Identify the values of a, b, and c in the quadratic equation. The variable *a* is the coefficient of the x^2 term, *b* is the coefficient of the x term, and *c* is the constant. For the equation $3x^2$ -5x - 8 = 0, a = 3, b = -5, and c = -8. Write this down.
13. Substitute the values of a, b, and c into the equation. Now that you know the values of the three variables, you can just plug them into the equation like this:

    {-b +/-√ (b^2 - 4ac)}/2

    {-(-5) +/-√ ($(-5)^2$ - 4(3)(-8))}/2(3) =

    {-(-5) +/-√ ($(-5)^2$ - (-96))}/2(3)

14. Do the math. After you've plugged in the numbers, do the remaining math to simplify positive or negative signs, multiply, or square the remaining terms. Here's how you do it:

    {-(-5) +/-√ ($(-5)^2$ - (-96))}/2(3) =

    {5 +/-√(25 + 96)}/6

    {5 +/-√(121)}/6

14. Simplify the square root**.** If the number under the radical symbol is a perfect square, you will get a whole number. If the number is not a perfect square, then simplify to its simplest radical version. If the number is negative, *and you're sure it's supposed to be negative,* then the roots will be complex. In this example, √(121) = 11. You can write that x = (5 +/- 11)/6.
15. Convert the final roots from hex to ascii
16. Print the final roots. Real and imaginary.

Conclusion: Thus Roots of Quadratic equations are calculated following above steps.

Review Questions: Difference between 80386 and 80387 microprocessor?

ASSIGNMENT NUMBER: 11

TITLE	**Write 80387 ALP to plot Sine Wave, Cosine Wave and Sinc function. Access video memory directly for plotting.**
PROBLEM STATEMENT /DEFINITION	**Write 80387 ALP to plot Sine Wave, Cosine Wave and Sinc function. Access video memory directly for plotting.**
OBJECTIVE	To understand how to how to access video memory for plotting.
OUTCOME	Students will study video memory plotting and regen buffer concepts..
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **Editor:** gedit/vi **Assembler:** NASM **Debugger:** GDB
REFERENCES	➢ "The Intel microprocessor", Barry B. Brey. ➢ "Introduction to 64 bit Intel Assembly Language Programming for Linux", 2nd Edition, Ray Seyfarth, ➢ "80386 Microprocessor Handbook", Chris H. Papas.
STEPS	

INSTRUCTIONS FOR **WRITING JOURNAL**	1. Title 2. Problem Definition 3. Objective: Intention behind study 4. Software & Hardware requirements 5. Explanation of the assignment 6. Algorithm 7. State Diagram 8. Test cases for program. **9.** Print outs **10.** Conclusion.

Assignment No. 11

- **Aim: Write 80387 ALP to plot Sine Wave, Cosine Wave and Sinc function. Access video memory directly for plotting.**
- **Prerequisites : Dos Interrupts, FPU instruction**
- **Concept related Theory:**
    - The overall display characteristics, such as vertical and horizontal resolution, background color, and palette, are controlled by values written to I/O ports whose addresses are hardwired on the adapter,
    - whereas the appearance of each individual character or graphics pixel on the display is controlled by a specific location within an area of memory called the regen buffer or refresh buffer.
    - Both the CPU and the video controller access this memory;
    - The software updates the display by simply writing character codes or bit patterns directly into the regen buffer. (This is called memory-mapped I/O.)

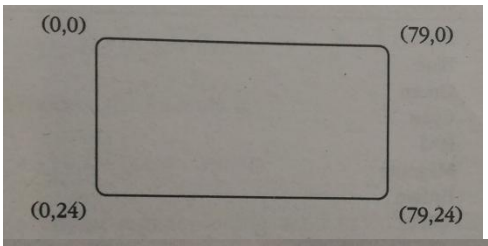
      E.g. MDA, CGA, EGA, MCGA, VGA etc.

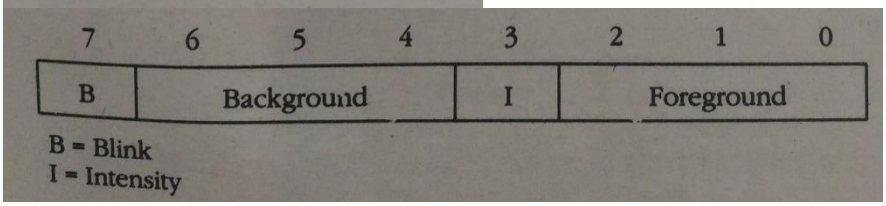
Text Mode:

- Sometimes also called as alphanumeric display mode
- Use 16KB of memory starting at segment B800H

- 2 Bytes per character (ASCII Code & Attribute)
- For 80-by-25 text mode

offset = (y*80 + x) *2





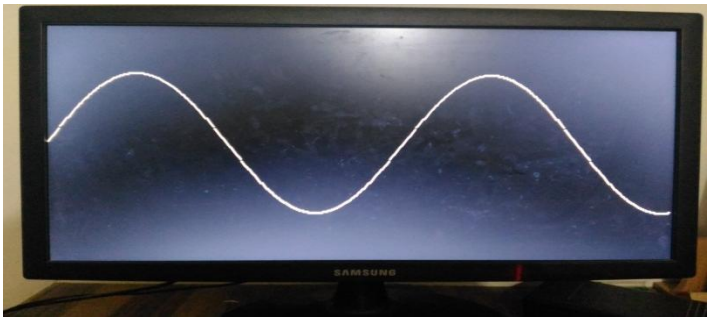
Graphics Mode:

- More complicated
- Each bit or group of bits in regen buffer corresponds to an addressable point or pixel on the screen
- E.g. 640-by-200, 2-color graphics display mode of the CGA
- Each pixel is represented by a single bit
- (x,y) range (0,0) through (639,199)
- The memory map is set up so that all the even y coordinates are scanned as a set and all the odd y coordinates are scanned as a set; this mapping is referred to as the memory interlace.
- Offset = ((y AND 1)*2000H) + (y/2*50H)+ (x/8)
- Bit position = 7 – (x MOD 8)

  (8 byte table or array of bit masks and operation ‘x AND 7’)

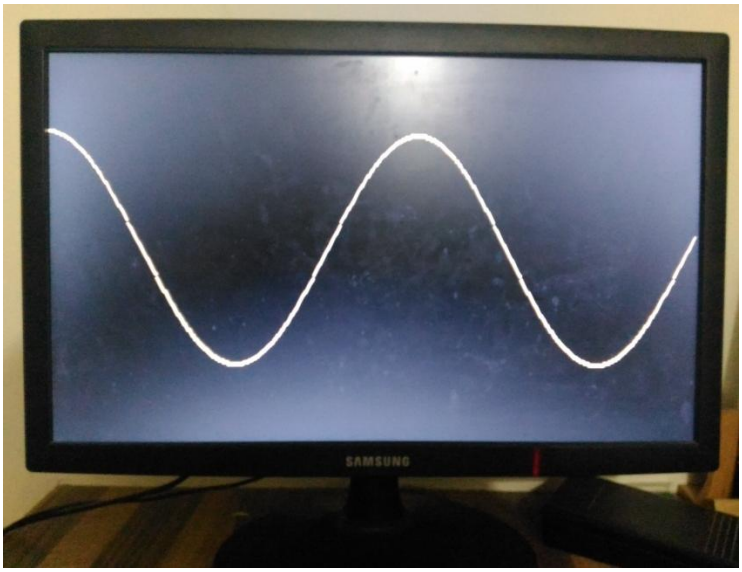
Sine Wave:

- Y = 100 – 60 sin ({pi/180}*x)



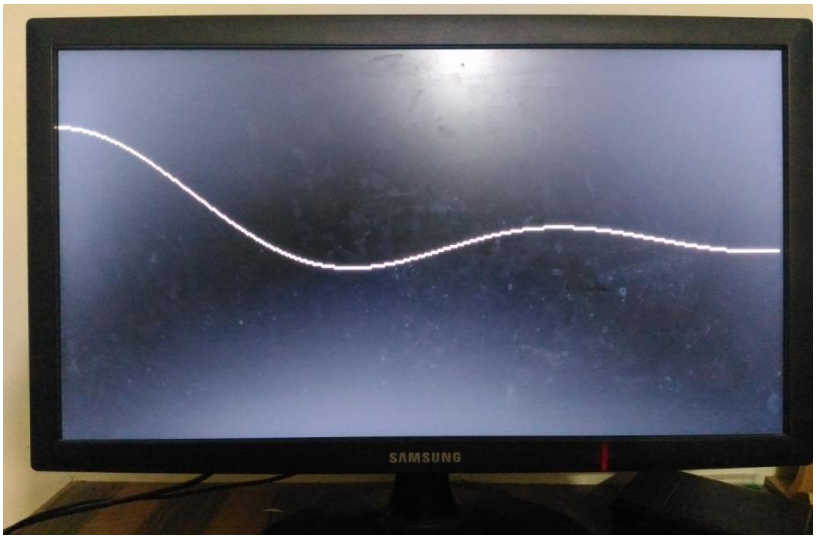
Cosine Wave:

- **Y = 100 – 60 cos ({pi/180}*x)**



Sinc Function:

- **Sinc (x) = sin (x)/x**



- **Conclusion**

  We have studied video memory plotting to plot sine wave, cosine wave and sin function.

- **Review Questions:**

  1. What is the starting address of video memory buffer?
  2. What is the size of regen buffer? It is divided in how many parts?
  3. Calculate offset for 80 by 25 text mode.
  4. Explain alphanumeric display mode.

ASSIGNMENT NUMBER: 12

ASSINGMENT NO.	12
TITLE	Write 80387 ALP to obtain: i) Mean ii) Variance iii) Standard Deviation Also plot the histogram for the data set. The data elements are available in a text file.
PROBLEM STATEMENT /DEFINITION	Write 80387 ALP to obtain: i) Mean ii) Variance iii) Standard Deviation Also plot the histogram for the data set. The data elements are available in a text file.
OBJECTIVE	To be able to solve mathematical problems in Assembly language programming To be able to handle file and data set from file in ALP, To be able to include mathematical histogram using ALP
OUTCOME	Students will be efficient in handling and solving mathematical problems through file using ALP
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **Editor:** gedit/vi **Assembler:** NASM **Debugger:** GDB
REFERENCES	➢ "The Intel microprocessor", Barry B. Brey. ➢ "Introduction to 64 bit Intel Assembly Language Programming for Linux", 2nd Edition, Ray Seyfarth, "80386 Microprocessor Handbook", Chris H. Papas.
STEPS	1. Create a one Folder which contain 1 program file and 1 text file. 2. Write all input text to the text file. 3. Define all parameter required to execute above procedure in 1st program file.

	4. Define all the procedures a. Ascii to Hex b. Hex to ascii c. Find Mean d. Find Variance e. Find Standard Deviation 5. Accept /Read the data set 6. Perform Ascii to Hex 7. Calculate mean 8. Print hex Mean value 9. Use mean value to calculate variance 10. Print hex Variance 11. Calculate Standard Deviation 12. Print hex Deviation.
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 10. Conclusion/Analysis

Assignment No. 12

Aim: Write 80387 ALP to obtain: i) Mean ii) Variance iii) Standard Deviation Also plot the histogram for the data set. The data elements are available in a text file.

Prerequisites: Basic instructions of Assembly Language programming

Concepts related Theory:

- **OPEN File**

mov rax, 2 ; 'open' syscall

mov rdi, fname1 ; file name

mov rsi, 2 ; File access mode

mov rdx, 0777 ; permissions set

Syscall

mov [fd_in], rax

- **READ File**

mov rax, 0 ; 'Read' syscall

mov rdi, [fd_in] ; file Pointer

mov rsi, Buffer ; Buffer for read

mov rdx, length ; len of data want to read

Syscall

- **WRITE File**

mov rax, 01 ; 'Write' syscall

mov rdi, [fd_in] ; file Pointer

mov rsi, Buffer ; Buffer for write

mov rdx, length ; len of data want to read

Syscall

- **CLOSE File**

mov rax,3

mov rdi,[fd_in]

syscall

Algorithm:

Data set: Measured height of the dogs

The heights (at the shoulders) are: 600mm, 470mm, 170mm, 430mm and 300mm.

Mean = Addition of all the values /no of values

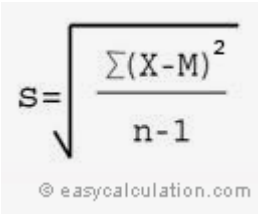
Mean = $\frac{600 + 470 + 170 + 430 + 300}{5} = \frac{1970}{5} = 394$

Variance : Variance = s^2

To calculate the Variance, take each difference, square it, and then average the result:

$$\text{Variance: } \sigma^2 = \frac{206^2 + 76^2 + (-224)^2 + 36^2 + (-94)^2}{5}$$
$$= \frac{42,436 + 5,776 + 50,176 + 1,296 + 8,836}{5}$$
$$= \frac{108,520}{5} = 21,704$$

Sample Standard Deviation :



And the Standard Deviation is just the square root of Variance, so:

Standard Deviation

σ = √21,704
= 147.32...
= **147** *(to the nearest mm)*

Conclusion:. Thus we have calculated Mean ,Variance and Standard deviation using ALP

Review Questions:

1. How to find Mean , Variance, and Standard deviations?
2. What are the different Floating point Data type

ASSIGNMENT NUMBER: 13

ASSIGNMENT NO	**13**
TITLE	**Write a Terminate but Stay Resident (TSR) program for a key-logger. The key-presses during the stipulated time need to be displayed at the center of the screen**
PROBLEM STATEMENT /DEFINITION	**Write a Terminate but Stay Resident (TSR) program for a key-logger. The key-presses during the stipulated time need to be displayed at the center of the screen**
OBJECTIVE	To understand Terminate but Stay Resident program.
OUTCOME	Students will study Terminate but Stay Program for a key-logger..
S/W PACKAGES AND HARDWARE APPARATUS USED	**Processor:** Core 2 duo/i3/i5/i7 **OS:** Linux 32bit/64bit OS **S/W:** key-logger **Assembler:** NASM **Debugger:** GDB
REFERENCES	➢ "The Intel microprocessor", Barry B. Brey. ➢ "Introduction to 64 bit Intel Assembly Language Programming for Linux", 2nd Edition, Ray Seyfarth, ➢ "80386 Microprocessor Handbook", Chris H. Papas.

STEPS	1. Far JUMP to transient portion of memory 2. Clear IF 3. Get vector address 4. Save vector address 5. Set new vector address 6. Make program resident
INSTRUCTIONS FOR WRITING JOURNAL	Title 2. Problem Definition 3. Objective: Intention behind study 4. Software & Hardware requirements 5. Explanation of the assignment 6. Algorithm 7. State Diagram 8. Test cases for program. 9. Print outs 10. Conclusion.

Assignment No. 13

Aim: Write a Terminate but Stay Resident (TSR) program for a key-logger. The key-presses during the stipulated time need to be displayed at the center of the screen

- **Prerequisites :** MS DOS, DOS interrupts
- **Concept related Theory:**
    - Terminate and Stay Resident (TSR) generally refers to a special class of programs for PC-compatible computers running DOS.
    - When a user exits a normal program running in DOS, the memory that the program used is usually freed for other programs and tasks; therefore, the program must be reloaded from a disk back into memory for it to be used again. When you run a TSR program, however, it loads itself into the computer's memory and remains there for later use. You may run other programs while the TSR is still alive in memory, and these programs may

invoke the TSR or be affected by the behavior of the TSR program. For this reason, TSR programs may give DOS the appearance of multitasking (the ability to perform several tasks at once) which is built into many other operating systems.

- You may use TSR programs for a wide variety of tasks. Some of these programs are active only when you press a hot key (a special key or combination of keystrokes that activates the TSR). An example would be a pop-up calculator program that appears on the screen whenever you press Alt-Shift-c, even from within a separate word processing program. Other TSR programs run continuously in the background and may normally be invisible. Examples of such programs are some network and communications programs and special virus scanner programs that monitor the use of a computer's memory and disk drives. Still other TSR programs may operate in both ways. A visually impaired user might call up a TSR that intercepts text information sent to the screen and displays it in a larger, easier-to-read format.

- TSR programs may be quite complex, and are often difficult to program reliably. TSR programmers must make sure that their programs do not conflict with other programs active in the computer's memory. The TSR must also not interfere with other programs' use of the disk drives or other hardware. Whenever a TSR becomes active, it must carefully record information in memory being used by another program, and restore this information exactly when it transfers control.

- Because of this complexity, TSR programs are often likely culprits when a PC is behaving strangely or crashing. Some TSR programs may not be compatible with other programs because of the way they use memory, or they may conflict with other TSR programs that are also active. When installing TSR programs, it is a good idea to install them one at a time, and make sure that other programs such as word processors and spreadsheets are behaving normally despite the presence of the TSR.

- Another drawback to TSR programs is their consumption of memory. Because a TSR retains a block of the computer's memory as its own, less space is then available to other programs. If several TSR programs are present in memory, there may not be enough space left over to load other large, demanding programs such as spreadsheets. Beginning with version 5.0, however, DOS supports the ability to load high most of a TSR program's contents into expanded and extended memory, which have fewer demands on their use than the computer's main memory, where most programs are executed and consumption of space is critical.

- Key-logger:

key-logger is a type of surveillance software (considered to be either software or spyware) that has the capability to record every keystroke you make to a log file, usually encrypted. A key-logger recorder can record instant message, e-mail, and any information you type at any time using your keyboard. The log file created by the key-logger can then be sent to a specified receiver. Some key-

logger programs will also record any e-mail addresses you use and Web Site URL's you visit.

- **Conclusion**

  We have studied Terminate but Stay Resident program for key-logger

- **Review Questions:**

  1. What is Terminate but Stay Resident program?
  2. What is the key-logger?
  3. What are the limitations of Terminate but Stay Resident program?
  4. What are the advantages of Terminate but Stay Resident program?.
  5. How to make your program TSR?
  6. How to get vector address?
  7. Hoe to set vector address?
  8. What are the types of model?
  9. What do you mean by TINY model?