**LAB MANUAL**

**MICROPROCESSOR & MICROCONTROLLER LABORTAORY(CE470)**

**SEM IV-Computer Engineering**

**List of Experiments**

**Class & Scheme:** SE COMP RC19-20

**Course:** CE470 MICROPROCESSORS & MICROCONTROLLERS LAB

**Year:** Mar\_July 2022

**Faculty in-charge:** Ms.Gauri Barve, Ms.Cynara Fernandes

|  |  |  |  |
| --- | --- | --- | --- |
| **Expt No** | **Experiment Name** | **CO** | **CL** |
| 1 | a)Introduction to programming in 803868(Vi Editor Commands)  b)Write 80386 ALP to print the given statement (eg:HELLO WORLD) |  | CL2 |
| 2 | Write a program which illustrates the programming constructs of higher level language in 80386 assembly coding. |  | CL3 |
| 3 | Write a program to implement Linear Search. |  | CL3 |
| 4 | Write a program to implement Bubble Sort. |  | CL3 |
| 5 | Write a program to demonstrate block data transfer with or without string instructions. |  | CL3 |
| 6 | Write a program which contains the following macros:  a. For calculating the Fibonacci series for N  b. For entering the value of N.  c. For displaying the numbers. |  |  |
| 7 | Write a program which contains the following procedures:  a. BCD to HEX  b. HEX to BCD |  | CL3 |
| 8 | Write a program using 80387 NDP. |  | CL3 |
| 9 | Case study on 8051 micro controller. |  | CL3 |

**Introduction to 80386 Microprocessor**

**What is microprocessor ?**

* Microprocessor, any of a type of miniature electronic device that contains the arithmetic, logic, and control circuitry necessary to perform the functions of a digital computer's central processing unit.
* Microprocessor is a computer Central processing Unit (CPU)on a single chip that contains millions of transistors connected by wires.

**A picture containing text, electronics

Description automatically generated**

**Microprocessor: ALU+CONTROL UNIT+Register Array**

**Diagram

Description automatically generated**

**Address BUS,Data Bus and Control Bus**

* Address Bus: It carries the address, which is a unique binary pattern used to identify a memory location or an I/O port. For example, an eight bit address bus has eight lines and thus it can address 28 = 256 different locations. The locations in hexadecimal format can be written as 00H – FFH.

• Data Bus: The data bus is used to transfer data between memory and processor or between I/O device and processor. For example, an 8-bit processor will generally have an 8-bit data bus and a 16-bit processor will have 16-bit data bus.

• Control Bus: The control bus carry control signals, which consists of signals for selection of mem synchronization of data transfer in case of slow memory or I/O device from the given address, direction of data transfer.

**Evolution of Microprocessor**

* 8-bit Microprocessors The first 8 bit microprocessor which could perform arithmetic and logic operations on 8 bit words was introduced in 1973 again by Intel. This was Intel 8008 and was later followed by an improved version, Intel 8088. Some other 8 bit processors are Zilog-80 and Motorola M6800.
* 16-bit Microprocessors The 8-bit processors were followed by 16 bit processors. They are Intel 8086 and 80286. 32-bit Microprocessors
* The 32 bit microprocessors were introduced by several companies but the most popular one is Intel 80386.
* Pentium Series Instead of 80586, Intel came out with a new processor namely Pentium processor. Its performance is closer to RISC performance. Pentium was followed by Pentium Pro CPU. Pentium Pro allows multiple CPUs in a single system in order to achieve multiprocessing.
* The MMX extension was added to Pentium Pro and the result was Pentium II.
* The Pentium III provided high performance floating point operations for certain types of computations by using the SIMD extensions to the instruction set. These new instructions makes the Pentium III faster than high-end RISC CPUs.

**Evolution of intel’s Microprocessor**

**Table

Description automatically generated**

**Architecture of 80386**

**Diagram

Description automatically generated**

**Central processing Unit(CU):**

**1. Execution unit(EU)**

**2.Instruction unit(IU)**

**• Memory Management Unit(MMU):**

**1.Segmentation Unit**

**2.Paging Unit**

**• Bus Interface Unit(BIU):**

* Execution unit has 8 General purpose and 8 Special purpose registers which are either used for handling data or calculating offset addresses.

• The Instruction unit consist of a Prefetch and Decode unit.

• The Instruction Prefetch unit is used to fetch instructions in advance to implement pipelining.

• While the Execution unit(EU) is executing the current instruction the Prefetch unit fetches the next 16B of the program & stores it into the Prefetch Queue.

* The Prefetch unit requests the Bus unit to fetch for instructions & if at the same time the EU also requests then priority is given to EU.

• While the EU is busy , the Instruction Decode unit decodes the opcode bytes received from the 16-byte instruction code queue and arranges them in a 3- instruction decoded instruction queue.

• After decoding them pass it to the control section for deriving the necessary control signals. The 64-bit barrel shifter increases the speed of all shift and rotate operations.

* The multiply / divide logic implements the bit-shift-rotate algorithms to complete the operations in minimum time.

• Even 32- bit multiplications can be executed within one microsecond by the multiply / divide logic.

• Execution of any program needs Arithmetic and logic operations & this is performed by a 32-bit ALU .

• Operands for ALU can be taken from Register File which contains all general purpose register.

• Additionally it has a 32-bit flag register. These flags give status of the current result.

* A Protection Test Unit provides/gives protection to the programs or instruction based on their Privilege.
* (There are 4 privilege levels, Usually kernel of the OS code has highest privilege or protection followed by OS services [2nd & 3rd level] & the lowest for applications.
* For eg : Like for critical OS code & data can be protected by keeping them in more privilege segment than those that contains application code. This prevents application code from accessing the OS code & data . It is usually used to detect problems & bugs. Lets not go into detail.. )
* The Memory management unit consists of a Segmentation unit and a Paging unit. It is used to determine the Physical address from the Logical(virtual) address.

• 80386 microprocessor implements 64TB of virtual memory using Segmentation and Paging . Hence the Memory unit is subdivided into Segmentation unit and Paging unit.

• Segmentation unit allows the use of two address components, viz. segment and offset for relocability and sharing of code and data.

* Segmentation unit allows segments of size 4Gbytes at max.

• Paging unit works under the control of the segmentation unit, i.e. each segment is further divided into pages. The virtual memory is also organizes in terms of segments and pages by the memory management unit.

• The Segmentation unit provides a 4 level protection mechanism for protecting and isolating the system code and data from those of the application program.

* Segmentation unit allows segments of size 4Gbytes at max.

• Paging unit works under the control of the segmentation unit, i.e. each segment is further divided into pages. The virtual memory is also organizes in terms of segments and pages by the memory management unit.

• The Segmentation unit provides a 4 level protection mechanism for protecting and isolating the system code and data from those of the application program.

* Segmentation is compulsory , while Paging is optional.

• The Paging unit organizes the physical memory in terms of pages of 4kbytes size each.

• The Segmentation unit converts the 48-bit Logical Address(actually 46b,2b for protection) into 32-bit Physical address while the Paging unit converts the 32-bit linear address into 32-bit physical address.

• If paging is not used, then Linear address itself is the Physical address.

* The control and attribute PLA checks the privileges at the page level. Each of the pages maintains the paging information of the task. The limit and attribute PLA checks segment limits and attributes at segment level to avoid invalid accesses to code and data in the memory segments.

• The Bus control unit has a prioritizer to resolve the priority of the various bus requests. This controls the access of the bus. The address driver drives the bus enable and address signal A0 – A31. The pipeline and dynamic bus sizing unit handle the related control signals.

• The data buffers interface the internal data bus with the system bus.

**Register Organization of 80386**

A picture containing text, receipt

Description automatically generated

* **Eight 32 - bit general purpose registers which may be used as either 8 bit or 16 bit registers.**
* **A 32 - bit register known as an extended register, is represented by the register name with prefix E.**
* **Example : A 32 bit register corresponding to AX is EAX, similarly BX is EBX etc**

**The 16 bit registers BP, SP, SI and DI in 8086 are now available with their extended size of 32 bit and are names as EBP,ESP,ESI and EDI.**

* **AX represents the lower 16 bit of the 32 bit register EAX.**
* **BP, SP, SI, DI represents the lower 16 bit of their 32 bit counterparts, and can be used as independent 16 bit registers.**
* **The six segment registers available in 80386 are CS, SS, DS, ES, FS and GS. The CS and SS are the code and the stack segment registers respectively, while DS, ES, FS, GS are 4 data segment registers.**
* **A 16 bit instruction pointer IP is available along with 32 bit counterpart EIP.**

**FLAG REGISTER OF 80386**

**A picture containing diagram

Description automatically generated**

* **C (Carry) –It holds the carry after calculations.**
* **P (Parity) –Parity is a logic 0 for odd parity and a logic 1 for even parity. Parity is a count of ones in a number expressed as even or odd.**
* **A (Auxiliary carry) –Carry occurs bits positions 3 and 5 of the results.**
* **Z (Zero) – The zero flag shows that the result of an arithmetic or logical operation is zero. When Z = 1, the result is zero. When Z = 0, the result was non-zero.**
* **S (Sign) – The sign flag holds the arithmetic sign after an arithmetic or a logical operation. If S =1 the sign bit is set and the result is negative. If S = 0, the sign bit is not set and the result is positive.**
* **T (Trap) – Enables trapping through an on-chip debugging facility.**
* **I (Interrupt) – The interrupt flag controls the operations of the INTR(Interrupt request) input pin. If I =1, the INTR pin is enabled; if I =0, the INTR pin is disabled.**
* **D(Direction flag)-This flag is set and cleared by the STD and CLD instructions. its sole purpose is to determine whether string operations (LODS,STOS,CMPS etc)will automatically increment (DF=0)or decrement (DF=1)the index registers ESI and EDI after each iteration .**
* **OF(overflow flag)-Most arithmetic instructions set this flag to indicate that the result was too large to fit in the destination.**
* **IOPL (I/O Privilege level) – IOPL is used in protected mode operation to select the privilege level for I./O devices. IF the current privilege level is higher or more trusted than the IOPL, I/O executed without hindrance. If the IOPL is lower than the current privilege level, an interrupt occurs.**
* **NT (nested task) – The nested task flag is used to indicated that the current task is nested within another task in protected mode operation. This flag is when the task I nested by software.**
* **RF(resume) – The resume flag is used with debugging to control the resumption of execution after the next instruction.**
* **VM (virtual mode) – If this flag set, the 80386 enters the virtual mode within the protected mode. In this mode, if any privileged instruction is executed, an exception 13 is generated.**

**Introduction to Vi Editor**

BASIC VI EDITOR COMMANDS

**To start vi:**

*Example:***> vi letter***will open a new file called letter to edit, or if letter already exits, open the exiting file.*

|  |  |
| --- | --- |
| **Command** | **Effect** |
| ***vi filename*** | edit *filename* starting at line 1 |
| ***vi +n filename*** | edit *filename* beginning at line n |
| ***vi +filename*** | edit *filename* beginning at the last line |
| ***vi -r filename*** | recover *filename* after a system crash |
| ***vi +/patter filename*** | edit *filename* starting at the first line containing **pattern** |

**Command Mode vs. Insert Mode**

**Insert mode** is the mode to be in when inserting text into the file. **Command mode** is the mode to be in when giving commands which will move the cursor, delete text, copy and paste, save the file etc.

When entering a file, vi is in command mode. To enter text, you must enter insert mode. If in insert mode, enter command mode by hitting the escape, <esc>, key.

**To insert text:**

|  |  |
| --- | --- |
| **Command** | **Insert Text** |
| **i** | before cursor |
| **a** | after cursor |
| **A** | at the end of the line |
| **o** | open a line below the current line |
| **O** | open a line above the current line |
| **r** | replace the current character |
| **R** | replace characters until <ESC>, overwrite |

**To move the cursor:**

You must be in Command Mode to use commands that move the cursor. Each of these commands can be preceded with a Repeat Factor.

*Examples:*  
**8j** will move the cursor down 8 lines  
**3w** will move the cursor 3 words to the right.

|  |  |
| --- | --- |
| **Command** | **Moves the cursor** |
| **SPACE, l (el), or right arrow** | space to the right |
| **h or left arrow** | space to the left |
| **j or down arrow** | down one line |
| **k or up arrow** | up one line |
| **w** | word to the right |
| **b** | word to the left |
| **$** | end of the line |
| **0**(zero) | beginning of the line |
| **e** | end of the word to the right |
| **-** | beginning of previous line |
| **)** | end of the sentence |
| **(** | beginning of the sentence |
| **}** | end of paragraph |
| **{** | beginning of paragraph |

**To Delete Text:**

The **d** command removes text from the Work Buffer. The amount removed depends on the Repeat Factor and the Unit of Measure you enter after **d**. **If you delete by mistake:**give the command **u** (undo) immediately after you give the delete command.

*Examples:***3dd** will delete 3 lines beginning with the current line.  
**3dw** or **d3w** will delete 3 words

|  |  |
| --- | --- |
| **Command** | **Action** |
| **d0** | delete to beginning of line |
| **dw** | delete to end of word |
| **d3w** | delete to end of third word |
| **db** | delete to beginning of word |
| **dW** | delete to end of blank delimited word |
| **dB** | delete to beginning of blank delimited word |
| **dd** | delete current line |
| **5dd** | delete 5 lines starting with the current line |
| **dL** | delete through the last line on the screen |
| **dH** | delete through the first line on the screen |
| **d)** | delete through the end of the sentence |
| **d(** | delete through the beginning of the sentence |
| **x** | delete the current character |
| **nx** | delete the number of characters specified by **n.** |
| **nX** | delete **n** characters before the current character |

**Viewing Different Parts of the Work Buffer:**

^Character means that you should hold down the **Control** key while striking the indicated character key.

|  |  |
| --- | --- |
| **Command** | **Moves the cursor** |
| **^D** | forward one-half screenful |
| **^U** | backward one-half screenful |
| **^F** | forward one screenful |
| **^B** | backward one screenful |
| **nG** | to line **n**(Ex: **25G** moves the cursor to line #25) |
| **H** | to the top of the screen |
| **M** | to the middle of the screen |
| **L** | to the bottom of the screen |
| **^L** | refresh the screen |

***Yanking (copy) and Putting (paste) Text:***

*Example:* **3yy** will yank (copy) 3 lines  
**p** will put the 3 lines just yanked on the line below the current cursor.

In the following list **M** is a Unit of Measure that you can precede with a Repeat Factor, n.

|  |  |
| --- | --- |
| **Command** | **Effect** |
| **yM** | yank text specified by M |
| **y3w** | yank 3 words |
| **nyy** | yank **n** lines |
| **Y** | yank to the end of the line |
| **P** | put text above current line |
| **p** | put text below current line |

**Changing Text**

*Example:* **cw** allows you to change a word. The word may be replaced by as many word as needed. Stop the change by hitting < esc &gt.  
**c3w** allows you to change 3 words.

**Ending an Editing Session**

|  |  |
| --- | --- |
| **Command** | **Effect** |
| **:w** | writes the contents of the work buffer to the file |
| **:q** | quit |
| **:q!** | quit without saving changes |
| **ZZ** | save and quit |
| **:wq** | save and quit |
| **:w *filename*** | saves to *filename* (allows you to change the name of the file) |

**Miscellaneous commands**

|  |  |
| --- | --- |
| **Command** | **Effect** |
| **J** | join the current line and the following line |
| **:set number** | number the lines on the screen (not actually added to file) |
| **:set nonumber** | turns off numbering of lines |
| **:r *filename*** | reads *filename* into the current file at the location of the cursor |
| **:set showmode** | displays INPUT MODE at the lower right hand corner of screen |
| **~** | change uppercase to lowercase and vice-versa |

**Date:22/03/2022**

**Experiment No 1: Introduction to programming in 80386**

**Theory:**

1. Check if putty.exe is on your desktop. If not download it from 172.16.40.1/downloads2/

Putty 0.70(x64)

1. Open putty.exe
2. Under Hostname type 172.16.40.10 and click open
3. Your command prompt opens and asks for login id and password
4. Enter your login id as **pcce**
5. Enter your password as **pcce**
6. You will be logged in to the pcce account
7. At the command prompt type vi filename.asm
8. Type the following code

**segment .text**

**global \_start**

**\_start:**

**;display a message**

**mov edx,len ;size of message**

**mov ecx,msg ;pointer to char buffer**

**mov ebx,1 ;fd is file descriptor**

**;stdin=0,stdout=1,stderr=2**

**mov eax,4 ;syscall for write is 4**

**int 0x80 ;call interrupt 80h**

**;exit the program**

**mov eax,1 ;syscall for exit program is 1**

**int 0x80 ;call interrupt 80h**

**segment .data**

**msg db 'Hello,World!'**

**len equ $ -msg**

1. Enter :wq
2. Assemble the program type

nasm -f elf64 filename.asm -o filename.o

1. If the execution is error free, it implies that filename.o object file has been created.
2. To link and create the executable give the command

ld -o filename filename.o

1. To execute the program write at the prompt

./filename

1. “Hello, World” will be displayed at the prompt.

IMPORTANT link

<https://securityboulevard.com/2021/05/linux-x86-assembly-how-to-build-a-hello-world-program-in-nasm/>

<https://jameshfisher.com/2018/03/10/linux-assembly-hello-world/>

**Experiment No:2.a Adding,sub ,Mul and Div of two numbers**

**Aim :**Adding of two numbers using registers in Assembly language using Nasm on linux x64 and to analyze this code using GDB

**Theory:**

Write syntax for mov instruction

Add instruction

**Add.asm :**

global \_start

section .text

\_start:

mov rax,10 ;move data to rax register

mov rbx,5

add rax,rbx

mov rax,1

int 0x80

**steps for GDB**

gdb -q -tui ./add

set assembly.flavour intel

break \_start

run

layout asm

layout regs

stepi

**Experiment :2b Adding three numbers in 80386 and analyse the code through GDB.**

**Experiment: 2c To perform the addition of two numbers(hardcoded) and display the output on the console.**

section .text

global \_start ;must be declared for using gcc

\_start: ;tell linker entry point

mov eax,'3'

sub eax, '0'

mov ebx, '4'

sub ebx, '0'

add eax, ebx

add eax, '0'

mov [sum], eax

mov ecx,msg

mov edx, len

mov ebx,1 ;file descriptor (stdout)

mov eax,4 ;system call number (sys\_write)

int 0x80 ;call kernel

mov ecx,sum

mov edx, 1

mov ebx,1 ;file descriptor (stdout)

mov eax,4 ;system call number (sys\_write)

int 0x80 ;call kernel

mov eax,1 ;system call number (sys\_exit)

int 0x80 ;call kernel

section .data

msg db "The sum is:", 0xA,0xD

len equ $ - msg

segment .bss

sum resb 1

**Expt2d:Enter two numbers from user and finding its sum**

SYS\_EXIT equ 1

SYS\_READ equ 3

SYS\_WRITE equ 4

STDIN equ 0

STDOUT equ 1

segment .data

msg1 db "Enter a digit ", 0xA,0xD

len1 equ $- msg1

msg2 db "Please enter a second digit", 0xA,0xD

len2 equ $- msg2

msg3 db "The sum is: "

len3 equ $- msg3

segment .bss

num1 resb 2

num2 resb 2

res resb 1

section .text

global \_start ;must be declared for using gcc

\_start: ;tell linker entry point

mov eax, SYS\_WRITE

mov ebx, STDOUT

mov ecx, msg1

mov edx, len1

int 0x80

mov eax, SYS\_READ

mov ebx, STDIN

mov ecx, num1

mov edx, 2

int 0x80

mov eax, SYS\_WRITE

mov ebx, STDOUT

mov ecx, msg2

mov edx, len2

int 0x80

mov eax, SYS\_READ

mov ebx, STDIN

mov ecx, num2

mov edx, 2

int 0x80

mov eax, SYS\_WRITE

mov ebx, STDOUT

mov ecx, msg3

mov edx, len3

int 0x80

; moving the first number to eax register and second number to ebx

; and subtracting ascii '0' to convert it into a decimal number

mov eax, [num1]

sub eax, '0'

mov ebx, [num2]

sub ebx, '0'

; add eax and ebx

add eax, ebx

; add '0' to to convert the sum from decimal to ASCII

add eax, '0'

; storing the sum in memory location res

mov [res], eax

; print the sum

mov eax, SYS\_WRITE

mov ebx, STDOUT

mov ecx, res

mov edx, 1

int 0x80

exit:

mov eax, SYS\_EXIT

xor ebx, ebx

int 0x80

**Expt 26:Multiplication of two number and displaying result on screen**

section .text

global \_start ;must be declared for using gcc

\_start: ;tell linker entry point

mov al,'3'

sub al, '0'

mov bl, '2'

sub bl, '0'

mul bl

add al, '0'

mov [res], al

mov ecx,msg

mov edx, len

mov ebx,1 ;file descriptor (stdout)

mov eax,4 ;system call number (sys\_write)

int 0x80 ;call kernel

mov ecx,res

mov edx, 1

mov ebx,1 ;file descriptor (stdout)

mov eax,4 ;system call number (sys\_write)

int 0x80 ;call kernel

mov eax,1 ;system call number (sys\_exit)

int 0x80 ;call kernel

section .data

msg db "The result is:", 0xA,0xD

len equ $- msg

segment .bss

res resb 1

**Expt 2e:Division of two numbers (hardcoded)**

section .text

global \_start ;must be declared for using gcc

\_start: ;tell linker entry point

mov ax,'8'

sub ax, '0'

mov bl, '2'

sub bl, '0'

div bl

add ax, '0'

mov [res], ax

mov ecx,msg

mov edx, len

mov ebx,1 ;file descriptor (stdout)

mov eax,4 ;system call number (sys\_write)

int 0x80 ;call kernel

mov ecx,res

mov edx, 1

mov ebx,1 ;file descriptor (stdout)

mov eax,4 ;system call number (sys\_write)

int 0x80 ;call kernel

mov eax,1 ;system call number (sys\_exit)

int 0x80 ;call kernel

section .data

msg db "The result is:", 0xA,0xD

len equ $- msg

segment .bss

**Exp 2f:Enter two numbers from users and do the division**

**Experiment 2g:Finding the sum of numbers using loop**

section .text

global \_start ;must be declared for linker (ld)

\_start:

mov eax,3 ;number bytes to be summed

mov ebx,0 ;EBX will store the sum

mov ecx, x ;ECX will point to the current element to be summed

top: add ebx, [ecx]

add ecx,1 ;move pointer to next element

dec eax ;decrement counter

jnz top ;if counter not 0, then loop again

done:

add ebx, '0'

mov [sum], ebx ;done, store result in "sum"

display:

mov edx,1 ;message length

mov ecx, sum ;message to write

mov ebx, 1 ;file descriptor (stdout)

mov eax, 4 ;system call number (sys\_write)

int 0x80 ;call kernel

mov eax, 1 ;system call number (sys\_exit)

int 0x80 ;call kernel

section .data

global x

x:

db 2

db 4

db 3

sum:

db 0