Practical 1

**Aim:** Introduction to 8086 Microprocessor & Assembly Language Programming. Write an assembly language programs which prints "Hello World" on screen.

**Hardware:** PC

**Software:** Emulator 8086

**Implementation:**

name "hi-world"

; this example prints out "hello world!"

; by writing directly to video memory.

; in vga memory: first byte is ascii character, byte that follows is character attribute.

; if you change the second byte, you can change the color of

; the character even after it is printed.

; character attribute is 8 bit value,

; high 4 bits set background color and low 4 bits set foreground color.

; hex bin color

;

; 0 0000 black

; 1 0001 blue

; 2 0010 green

; 3 0011 cyan

; 4 0100 red

; 5 0101 magenta

; 6 0110 brown

; 7 0111 light gray

; 8 1000 dark gray

; 9 1001 light blue

; a 1010 light green

; b 1011 light cyan

; c 1100 light red

; d 1101 light magenta

; e 1110 yellow

; f 1111 white

org 100h

; set video mode

mov ax, 3 ; text mode 80x25, 16 colors, 8 pages (ah=0, al=3)

int 10h ; do it!

; cancel blinking and enable all 16 colors:

mov ax, 1003h

mov bx, 0

int 10h

; set segment register:

mov ax, 0b800h

mov ds, ax

; print "hello world"

; first byte is ascii code, second byte is color code.

mov [02h], 'H'

mov [04h], 'e'

mov [06h], 'l'

mov [08h], 'l'

mov [0ah], 'o'

mov [0ch], ','

mov [0eh], 'W'

mov [10h], 'o'

mov [12h], 'r'

mov [14h], 'l'

mov [16h], 'd'

mov [18h], '!'

; color all characters:

mov cx, 12 ; number of characters.

mov di, 03h ; start from byte after 'h'

c: mov [di], 11101100b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

loop c

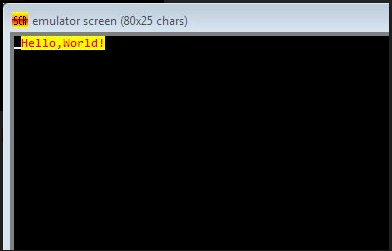
; wait for any key press:

mov ah, 0

int 16h

ret

**Output:**

****

Practical 2

Aim:

**Program-1**: Change the Foreground color of each letter of "Hello world"

**Hardware:** PC

**Software:** Emulator 8086

**Implementation :**

name "hi-world"

; this example prints out "hello world!"

; by writing directly to video memory.

; in vga memory: first byte is ascii character, byte that follows is character attribute.

; if you change the second byte, you can change the color of

; the character even after it is printed.

; character attribute is 8 bit value,

; high 4 bits set background color and low 4 bits set foreground color.

; hex bin color

;

; 0 0000 black

; 1 0001 blue

; 2 0010 green

; 3 0011 cyan

; 4 0100 red

; 5 0101 magenta

; 6 0110 brown

; 7 0111 light gray

; 8 1000 dark gray

; 9 1001 light blue

; a 1010 light green

; b 1011 light cyan

; c 1100 light red

; d 1101 light magenta

; e 1110 yellow

; f 1111 white

org 100h

; set video mode

mov ax, 3 ; text mode 80x25, 16 colors, 8 pages (ah=0, al=3)

int 10h ; do it!

; cancel blinking and enable all 16 colors:

mov ax, 1003h

mov bx, 0

int 10h

; set segment register:

mov ax, 0b800h

mov ds, ax

; print "hello world"

; first byte is ascii code, second byte is color code.

mov [02h], 'H'

mov [04h], 'e'

mov [06h], 'l'

mov [08h], 'l'

mov [0ah], 'o'

mov [0ch], ','

mov [0eh], 'W'

mov [10h], 'o'

mov [12h], 'r'

mov [14h], 'l'

mov [16h], 'd'

mov [18h], '!'

; color all characters:

mov cx, 12 ; number of characters.

mov di, 03h ; start from byte after 'h'

c: mov [di], 11100000b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11100001b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11100010b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11100011b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11100100b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11100101b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11100110b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11100111b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11101000b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11101001b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11101010b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11101011b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

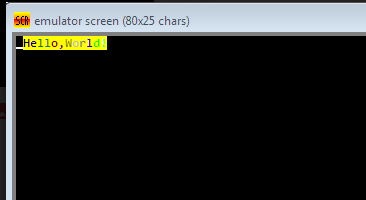
; wait for any key press:

mov ah, 0

int 16h

ret

**Output:**

****

**Program-2:**Change the background color of each letter of "Hello world"

**Implementation:**

name "hi-world"

; this example prints out "hello world!"

; by writing directly to video memory.

; in vga memory: first byte is ascii character, byte that follows is character attribute.

; if you change the second byte, you can change the color of

; the character even after it is printed.

; character attribute is 8 bit value,

; high 4 bits set background color and low 4 bits set foreground color.

; hex bin color

;

; 0 0000 black

; 1 0001 blue

; 2 0010 green

; 3 0011 cyan

; 4 0100 red

; 5 0101 magenta

; 6 0110 brown

; 7 0111 light gray

; 8 1000 dark gray

; 9 1001 light blue

; a 1010 light green

; b 1011 light cyan

; c 1100 light red

; d 1101 light magenta

; e 1110 yellow

; f 1111 white

org 100h

; set video mode

mov ax, 3 ; text mode 80x25, 16 colors, 8 pages (ah=0, al=3)

int 10h ; do it!

; cancel blinking and enable all 16 colors:

mov ax, 1003h

mov bx, 0

int 10h

; set segment register:

mov ax, 0b800h

mov ds, ax

; print "hello world"

; first byte is ascii code, second byte is color code.

mov [02h], 'H'

mov [04h], 'e'

mov [06h], 'l'

mov [08h], 'l'

mov [0ah], 'o'

mov [0ch], ','

mov [0eh], 'W'

mov [10h], 'o'

mov [12h], 'r'

mov [14h], 'l'

mov [16h], 'd'

mov [18h], '!'

; color all characters:

mov cx, 12 ; number of characters.

mov di, 03h ; start from byte after 'h'

c: mov [di], 11110000b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11110001b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11110010b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11110011b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11110100b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11110101b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11110110b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11110111b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11111000b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11111001b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11111010b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

mov [di], 11111011b ; light red(1100) on yellow(1110)

add di, 2 ; skip over next ascii code in vga memory.

;loop c

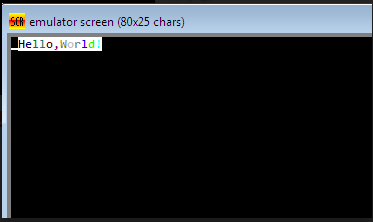
; wait for any key press:

mov ah, 0

int 16h

ret

**Output:**

****

**Practical 3**

**Aim:**

**Program-1:** Write a program to perform addition of 5 elements. Elements are 5,4,5,2,1

**Hardware:** PC

**Software:** Emulator 8086

**Implementation:**

name "calc-sum"

org 100h ; directive make tiny com file.

; calculate the sum of elements in vector,

; store result in m and print it in binary code.

; number of elements:

mov cx, 5

; al will store the sum:

mov al, 0

; bx is an index:

mov bx, 0

; sum elements:

next: add al, vector[bx]

; next byte:

inc bx

; loop until cx=0:

loop next

; store result in m:

mov m, al

; print result in binary:

mov bl, m

mov cx, 8

print: mov ah, 2 ; print function.

mov dl, '0'

test bl, 10000000b ; test first bit.

jz zero

mov dl, '1'

zero: int 21h

shl bl, 1

loop print

; print binary suffix:

mov dl, 'b'

int 21h

mov dl, 0ah ; new line.

int 21h

mov dl, 0dh ; carrige return.

int 21h

; print result in decimal:

mov al, m

call print\_al

; wait for any key press:

mov ah, 0

int 16h

ret

; variables:

vector db 5, 4, 5, 2, 1

m db 0

print\_al proc

cmp al, 0

jne print\_al\_r

push ax

mov al, '0'

mov ah, 0eh

int 10h

pop ax

ret

print\_al\_r:

pusha

mov ah, 0

cmp ax, 0

je pn\_done

mov dl, 10

div dl

call print\_al\_r

mov al, ah

add al, 30h

mov ah, 0eh

int 10h

jmp pn\_done

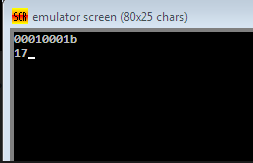
pn\_done:

popa

ret

endp

**Output:**

****

**Program-2:**Write a program to perform addition of 20 elements. Elements are 20 times 1.

**Implementation:**

name "calc-sum"

org 100h ; directive make tiny com file.

; calculate the sum of elements in vector,

; store result in m and print it in binary code.

; number of elements:

mov cx, 20

; al will store the sum:

mov al, 0

; bx is an index:

mov bx, 0

; sum elements:

next: add al, vector[bx]

; next byte:

inc bx

; loop until cx=0:

loop next

; store result in m:

mov m, al

; print result in binary:

mov bl, m

mov cx, 8

print: mov ah, 2 ; print function.

mov dl, '0'

test bl, 10000000b ; test first bit.

jz zero

mov dl, '1'

zero: int 21h

shl bl, 1

loop print

; print binary suffix:

mov dl, 'b'

int 21h

mov dl, 0ah ; new line.

int 21h

mov dl, 0dh ; carrige return.

int 21h

; print result in decimal:

mov al, m

call print\_al

; wait for any key press:

mov ah, 0

int 16h

ret

; variables:

vector db 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

m db 0

print\_al proc

cmp al, 0

jne print\_al\_r

push ax

mov al, '0'

mov ah, 0eh

int 10h

pop ax

ret

print\_al\_r:

pusha

mov ah, 0

cmp ax, 0

je pn\_done

mov dl, 10

div dl

call print\_al\_r

mov al, ah

add al, 30h

mov ah, 0eh

int 10h

jmp pn\_done

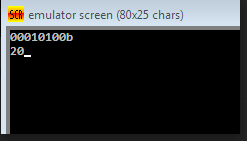
pn\_done:

popa

ret

endp

**Output:**

****

**Practical 4**

**Aim:** Observe the values of flags for Example program: Compare Numbers. Write down Arithmetic and Logic Instructions from Tutorial.

**Hardware:** PC

**Software:** Emulator 8086

**Program-1:** Print result (above/equal/below) on screen for first comparison

**Implementation:**name "flags"

org 100h

; this sample shows how cmp instruction sets the flags.

; usually cmp instruction is followed by any relative

; jump instruction such as: je, ja, jl, jae...

; it is recommended to click "flags" and "analyze"

; for better visual expirience before stepping through this code.

; (signed/unsigned)

; 4 is equal to 4

mov ah, 4

mov al, 4

cmp ah, al

nop

; (signed/unsigned)

; 4 is above and greater then 3

mov ah, 4

mov al, 3

cmp ah, al

nop

; -5 = 251 = 0fbh

; (signed)

; 1 is greater then -5

mov ah, 1

mov al, -5

cmp ah, al

nop

; (unsigned)

; 1 is below 251

mov ah, 1

mov al, 251

cmp ah, al

nop

; (signed)

; -3 is less then -2

mov ah, -3

mov al, -2

cmp ah, al

nop

; (signed)

; -2 is greater then -3

mov ah, -2

mov al, -3

cmp ah, al

nop

; (unsigned)

; 255 is above 1

mov ah, 255

mov al, 1

cmp ah, al

nop

; now a little game:

game: mov dx, offset msg1

mov ah, 9

int 21h

; read character in al:

mov ah, 1

int 21h

cmp al, '0'

jb stop

cmp al, '9'

ja stop

cmp al, '5'

jb below

ja above

mov dx, offset equal\_5

jmp print

below: mov dx, offset below\_5

jmp print

above: mov dx, offset above\_5

print: mov ah, 9

int 21h

jmp game ; loop.

stop: ret ; stop

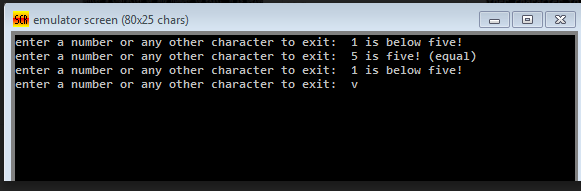
msg1 db "enter a number or any other character to exit: $"

equal\_5 db " is five! (equal)", 0Dh,0Ah, "$"

below\_5 db " is below five!" , 0Dh,0Ah, "$"

above\_5 db " is above five!" , 0Dh,0Ah, "$"

**Output:**

****

**Program-3:** Rewrite program to compare alphabet a to z. l act as a eqator. it prints above and below with respect to l.

**Implementation:**

name "flags"

org 100h

; this sample shows how cmp instruction sets the flags.

; usually cmp instruction is followed by any relative

; jump instruction such as: je, ja, jl, jae...

; it is recommended to click "flags" and "analyze"

; for better visual expirience before stepping through this code.

; (signed/unsigned)

; 4 is equal to 4

mov ah, 4

mov al, 4

cmp ah, al

nop

; (signed/unsigned)

; 4 is above and greater then 3

mov ah, 4

mov al, 3

cmp ah, al

nop

; -5 = 251 = 0fbh

; (signed)

; 1 is greater then -5

mov ah, 1

mov al, -5

cmp ah, al

nop

; (unsigned)

; 1 is below 251

mov ah, 1

mov al, 251

cmp ah, al

nop

; (signed)

; -3 is less then -2

mov ah, -3

mov al, -2

cmp ah, al

nop

; (signed)

; -2 is greater then -3

mov ah, -2

mov al, -3

cmp ah, al

nop

; (unsigned)

; 255 is above 1

mov ah, 255

mov al, 1

cmp ah, al

nop

; now a little game:

game: mov dx, offset msg1

mov ah, 9

int 21h

; read character in al:

mov ah, 1

int 21h

cmp al, 'a'

jb stop

cmp al, 'z'

ja stop

cmp al, 'l'

jb below

ja above

mov dx, offset equal\_5

jmp print

below: mov dx, offset below\_5

jmp print

above: mov dx, offset above\_5

print: mov ah, 9

int 21h

jmp game ; loop.

stop: ret ; stop

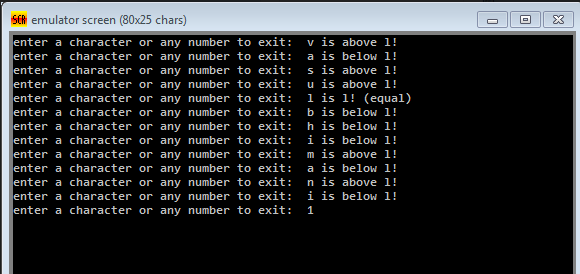
msg1 db "enter a character or any number to exit: $"

equal\_5 db " is l! (equal)", 0Dh,0Ah, "$"

below\_5 db " is below l!" , 0Dh,0Ah, "$"

above\_5 db " is above l!" , 0Dh,0Ah, "$"

**Output:**

****

**Practical 5**

**Aim:** AIM: 8086 Procedures and dealy.

**Program-1:** Traffic control System. Review program of traffic control system available in simulator.

**Hardware:** PC

**Software:** Emulator 8086

**Implementation:**

; controlling external device with 8086 microprocessor.

; realistic test for c:\emu8086\devices\Traffic\_Lights.exe

#start=Traffic\_Lights.exe#

name "traffic"

mov ax, all\_red

out 4, ax

mov si, offset situation

next:

mov ax, [si]

out 4, ax

; wait 5 seconds (5 million microseconds)

mov cx, 4Ch ; 004C4B40h = 5,000,000

mov dx, 4B40h

mov ah, 86h

int 15h

add si, 2 ; next situation

cmp si, sit\_end

jb next

mov si, offset situation

jmp next

; FEDC\_BA98\_7654\_3210

situation dw 0000\_0011\_0000\_1100b

s1 dw 0000\_0110\_1001\_1010b

s2 dw 0000\_1000\_0110\_0001b

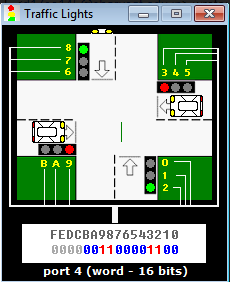
s3 dw 0000\_1000\_0110\_0001b

s4 dw 0000\_0100\_1101\_0011b

sit\_end = $

all\_red equ 0000\_0010\_0100\_1001b

**Output:**

****

**Program-2:** Add procedure into Traffic Control system program.

**Implementation:**

; controlling external device with 8086 microprocessor.

; realistic test for c:\emu8086\devices\Traffic\_Lights.exe

#start=Traffic\_Lights.exe#

name "traffic"

mov ax, all\_red

out 4, ax

mov si, offset situation

CALL m2

m2 PROC

next:

mov ax, [si]

out 4, ax

; wait 5 seconds (5 million microseconds)

mov cx, 4Ch ; 004C4B40h = 5,000,000

mov dx, 4B40h

mov ah, 86h

int 15h

add si, 2 ; next situation

cmp si, sit\_end

jb next

mov si, offset situation

jmp next

m2 ENDP

; FEDC\_BA98\_7654\_3210

situation dw 0000\_0011\_0000\_1100b

s1 dw 0000\_0110\_1001\_1010b

s2 dw 0000\_1000\_0110\_0001b

s3 dw 0000\_1000\_0110\_0001b

s4 dw 0000\_0100\_1101\_0011b

sit\_end = $

all\_red equ 0000\_0010\_0100\_1001b

; controlling external device with 8086 microprocessor.

; realistic test for c:\emu8086\devices\Traffic\_Lights.exe

#start=Traffic\_Lights.exe#

name "traffic"

mov ax, all\_red

out 4, ax

mov si, offset situation

m1 PROC

next:

mov ax, [si]

out 4, ax

; wait 5 seconds (5 million microseconds)

mov cx, 4Ch ; 004C4B40h = 5,000,000

mov dx, 4B40h

mov ah, 86h

int 15h

ret

m1 ENDP

CALL m1

add si, 2 ; next situation

cmp si, sit\_end

jb next

mov si, offset situation

jmp next

; FEDC\_BA98\_7654\_3210

situation dw 0000\_0011\_0000\_1100b

s1 dw 0000\_0110\_1001\_1010b

s2 dw 0000\_1000\_0110\_0001b

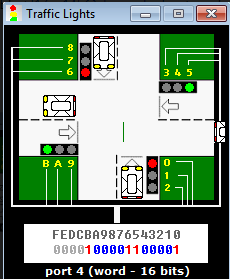
s3 dw 0000\_1000\_0110\_0001b

s4 dw 0000\_0100\_1101\_0011b

sit\_end = $

all\_red equ 0000\_0010\_0100\_1001b

**Output:**

****

**Practical 6**

**Aim:**

**Hardware:** PC

**Software:** Emulator 8086

**Program-1:** Write an assembly program to print string in reverse order.

**Implementation:**

; reverse string

name "reverse"

org 100h

jmp start

; when reversed it will be a readable string,

; '$' marks the end of the string:

string1 db '! D17ce151$'

start: lea bx, string1

mov si, bx

next\_byte: cmp [si], '$'

je found\_the\_end

inc si

jmp next\_byte

found\_the\_end: dec si

; now bx points to beginning,

; and si points to the end of string.

; do the swapping:

do\_reverse: cmp bx, si

jae done

mov al, [bx]

mov ah, [si]

mov [si], al

mov [bx], ah

inc bx

dec si

jmp do\_reverse

; reverse complete, print out:

done: lea dx, string1

mov ah, 09h

int 21h

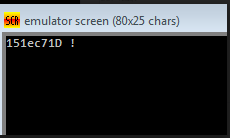
; wait for any key press....

mov ah, 0

int 16h

ret

**Output:**

****

**Program-2:** Write an assembly program to check whether string is a palindrome or not.

**Implementation:**

; this sample checks if string is a palindrome or not.

; palindrome is a text that can be read backwards

; and give the same meaning as if it was read forward.

; for example: "abba" is polindrome.

; note: this program is case sensitive, "abba" is not "abba".

name "pali"

org 100h

jmp start

m1:

s db 'able was ere ere saw elba'

s\_size = $ - m1

db 0Dh,0Ah,'$'

start:

; first let's print it:

mov ah, 9

mov dx, offset s

int 21h

lea di, s

mov si, di

add si, s\_size

dec si ; point to last char!

mov cx, s\_size

cmp cx, 1

je is\_palindrome ; single char is always palindrome!

shr cx, 1 ; divide by 2!

next\_char:

mov al, [di]

mov bl, [si]

cmp al, bl

jne not\_palindrome

inc di

dec si

loop next\_char

is\_palindrome:

; the string is "palindrome!"

mov ah, 9

mov dx, offset msg1

int 21h

jmp stop

not\_palindrome:

; the string is "not palindrome!"

mov ah, 9

mov dx, offset msg2

int 21h

stop:

; wait for any key press:

mov ah, 0

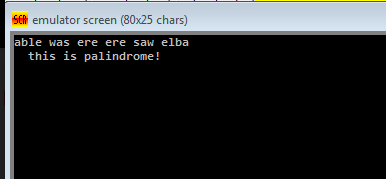
int 16h

ret

msg1 db " this is palindrome!$"

msg2 db " this is not a palindrome!$"

**Output:**

****

**Program-3:** Write an assembly program which converts upped case to lower and vice versa for a given string.

**Implementation:**

; this is a program in 8086 assembly language that

; accepts a character string from the keyboard and

; stores it in the string array. the program then converts

; all the lower case characters of the string to upper case.

; if the string is empty (null), it doesn't do anything.

name "upper"

org 100h

jmp start

; first byte is buffer size,

; second byte will hold number

; of used bytes for string,

; all other bytes are for characters:

string db 20, 22 dup('?')

new\_line db 0Dh,0Ah, '$' ; new line code.

start:

; int 21h / ah=0ah - input of a string to ds:dx,

; fist byte is buffer size, second byte is number

; of chars actually read. does not add '$' in the

; end of string. to print using int 21h / ah=09h

; you must set dollar sign at the end of it and

; start printing from address ds:dx + 2.

lea dx, string

mov ah, 0ah

int 21h

mov bx, dx

mov ah, 0

mov al, ds:[bx+1]

add bx, ax ; point to end of string.

mov byte ptr [bx+2], '$' ; put dollar to the end.

; int 21h / ah=09h - output of a string at ds:dx.

; string must be terminated by '$' sign.

lea dx, new\_line

mov ah, 09h

int 21h

lea bx, string

mov ch, 0

mov cl, [bx+1] ; get string size.

jcxz null ; is string is empty?

add bx, 2 ; skip control chars.

upper\_case:

; check if it's a lower case letter:

cmp byte ptr [bx], 'a'

jb ok

cmp byte ptr [bx], 'z'

ja ok

; convert to uppercase:

; upper case letter do not have

; third bit set, for example:

; 'a' : 01100001b

; 'a' : 01000001b

; upper case mask : 11011111b

; clear third bit:

and byte ptr [bx], 11011111b

ok:

inc bx ; next char.

loop upper\_case

; int 21h / ah=09h - output of a string at ds:dx.

; string must be terminated by '$' sign.

lea dx, string+2

mov ah, 09h

int 21h

; wait for any key press....

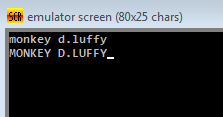
mov ah, 0

int 16h

null:

ret ; return to operating system.

**Output:**



**Practical 7**

**Aim:**

**Program-1:**Write an assembly program which pritns ASCII value of given string letters on LED display. (Dealy between each letter must be 2 sec)

**Implementation:**

#start=led\_display.exe#

name "paliled"

org 100h

jmp start

m1:

s db 'abcdef'

s\_size = $ - m1

db 0Dh,0Ah,'$'

start:

mov ah, 9

mov dx, offset s

int 21h

lea di, s

mov cx, s\_size

cmp cx, 1

mov ax, 0

next\_char:

mov al, [di]

out 199,ax

inc di

loop next\_char

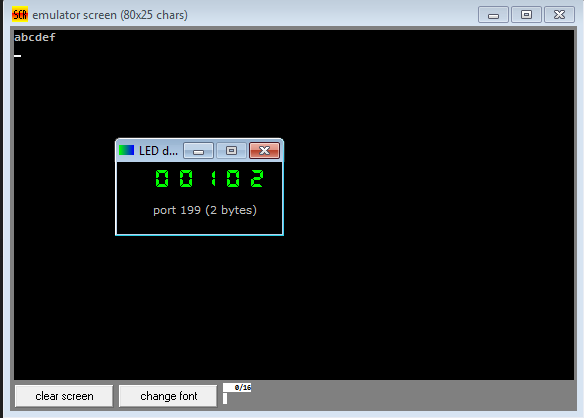
; wait for any key press:

mov ah, 0

int 16h

ret

**Output:**

****

**Program-2:**Write an assembly program which rotates stepper moter in clock wise direction. (Delay between each roatation must be 1 sec)

**Implementation:**

; this is an example of out instruction.

; it writes values to virtual i/o port

; that controls the stepper-motor.

; c:\emu8086\devices\stepper\_motor.exe is on port 7

#start=stepper\_motor.exe#

name "stepper"

#make\_bin#

steps\_before\_direction\_change = 20h ; 32 (decimal)

jmp start

; ========= data ===============

; bin data for clock-wise

; half-step rotation:

datcw db 0000\_0110b

db 0000\_0100b

db 0000\_0011b

db 0000\_0010b

; bin data for counter-clock-wise

; half-step rotation:

datccw db 0000\_0011b

db 0000\_0001b

db 0000\_0110b

db 0000\_0010b

; bin data for clock-wise

; full-step rotation:

datcw\_fs db 0000\_0001b

db 0000\_0011b

db 0000\_0110b

db 0000\_0000b

; bin data for counter-clock-wise

; full-step rotation:

datccw\_fs db 0000\_0100b

db 0000\_0110b

db 0000\_0011b

db 0000\_0000b

start:

mov bx, offset datcw ; start from clock-wise half-step.

mov si, 0

mov cx, 0 ; step counter

next\_step:

; motor sets top bit when it's ready to accept new command

wait: in al, 7

test al, 10000000b

jz wait

mov al, [bx][si]

out 7, al

inc si

cmp si, 4

jb next\_step

mov si, 0

inc cx

cmp cx, steps\_before\_direction\_change

jb next\_step

mov cx, 0

add bx, 4 ; next bin data

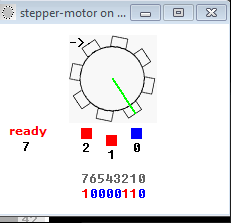
cmp bx, offset datccw\_fs

jbe next\_step

mov bx, offset datcw ; return to clock-wise half-step.

jmp next\_step

**Output:**

****

**Program-3:**Write an assembly program which counts number of key presses.

**Implementation:**

; Count number of key presses. the result is in bx register.

;

; You must type into the emulator's screen,

; if it closes, press screen button to re-open it.

name "keycount"

org 100h

; print welcome message:

mov dx, offset msg

mov ah, 9

int 21h

xor bx, bx ; zero bx register.

wait: mov ah, 0 ; wait for any key....

int 16h

cmp al, 27 ; if key is 'esc' then exit.

je stop

mov ah, 0eh ; print it.

int 10h

inc bx ; increase bx on every key press.

jmp wait

; print result message:

stop: mov dx, offset msg2

mov ah, 9

int 21h

mov ax, bx

call print\_ax

; wait for any key press:

mov ah, 0

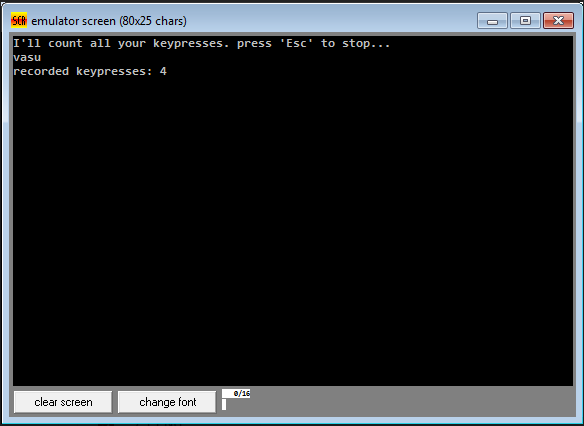
int 16h

ret ; exit to operating system.

msg db "I'll count all your keypresses. press 'Esc' to stop...", 0Dh,0Ah, "$"

msg2 db 0Dh,0Ah, "recorded keypresses: $"

**Output:**

****

**Practical 8**

**Aim:**

**Program-1:** Write an assembly program which gets the number from the user and calculates factorial for it.

**Implementation:**

; You may customize this and other start-up templates;

; The location of this template is c:\emu8086\inc\0\_com\_template.txt

org 100h

mov ax,05h

mov bx,04h

mov cx,01h

loop:

mul cx

inc cx

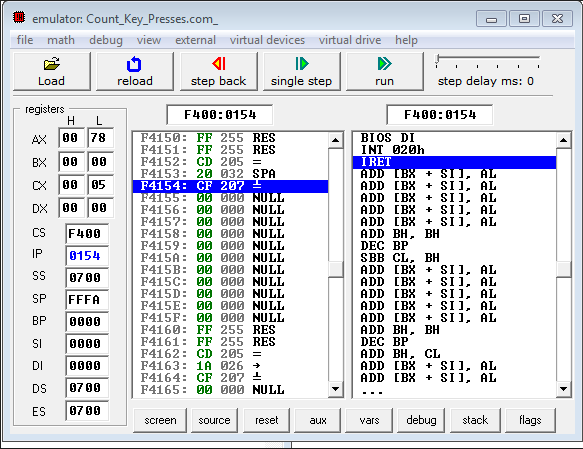
sub bx,01h

cmp bx,00h

jnz loop

ret

**Output:**

****

**Program-2:**Write an assembly program which converts a 2 digit hexadecimal number into binary.

**Implementation:**

; hex convertor.

; this example converts a 2 digit hexadecimal number

; into a numeric value and then into decimal/ascii string representation,

; and finally it prints out the result in binary code.

; to see decimal string:

; 1. click "vars"

; 2. click "result" variable

; 3. enter "3" for the elements and "ascii" for show as.

name "hex"

org 100h

jmp start

; source hex value is 2 char string.

; numeric value is stored into temp,

; and string decimal value is stored into result.

source db '8f', 0 ; 1bh is converted to 27 (decimal) 00011011b (binary)

result db '000', 0

temp dw ?

start:

; convert first digit to value 0..15 from ascii:

mov al, source[0]

cmp al, '0'

jae f1

f1:

cmp al, '9'

ja f2 ; jumps only if not '0' to '9'.

sub al, 30h ; convert char '0' to '9' to numeric value.

jmp num1\_ready

f2:

; gets here if it's 'a' to 'f' case:

or al, 00100000b ; remove upper case (if any).

sub al, 57h ; convert char 'a' to 'f' to numeric value.

num1\_ready:

mov bl, 16

mul bl ; ax = al \* bl

mov temp, ax

; convert second digit to value 0..15 from ascii:

mov al, source[1]

cmp al, '0'

jae g1

g1:

cmp al, '9'

ja g2 ; jumps only if not '0' to '9'.

sub al, 30h ; convert char '0' to '9' to numeric value.

jmp num2\_ready

g2:

; gets here if it's 'a' to 'f' case:

or al, 00100000b ; remove upper case (if any).

sub al, 57h ; convert char 'a' to 'f' to numeric value.

num2\_ready:

xor ah, ah

add temp, ax

; convertion from hex string complete!

push temp ; store original temp value.

; convert to decimal string,

; it has to be 3 decimal digits or less:

mov di, 2 ; point to top of the string.

next\_digit:

cmp temp, 0

je stop

mov ax, temp

mov bl, 10

div bl ; al = ax / operand, ah = remainder.

mov result[di], ah

add result[di], 30h ; convert to ascii.

xor ah, ah

mov temp, ax

dec di ; next digit in string.

jmp next\_digit

stop:

pop temp ; re-store original temp value.

; print result in binary:

mov bl, b.temp

mov cx, 8

print: mov ah, 2 ; print function.

mov dl, '0'

test bl, 10000000b ; test first bit.

jz zero

mov dl, '1'

zero: int 21h

shl bl, 1

loop print

; print binary suffix:

mov dl, 'b'

int 21h

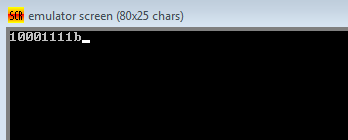
; wait for any key press:

mov ah, 0

int 16h

ret ; return to operating system.

**Output:**

****

Practical 10

**Aim**: Multicore programming, L1, L2 and L3 cache rebuilt and performance measurement. Comparison between Intel and AMD architecture.

**L1, L2 and L3 cache rebuilt:**

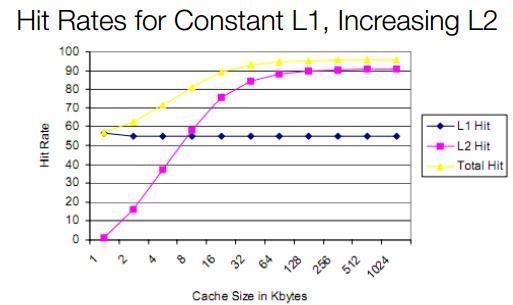
Caching was invented to solve a significant problem. In the early decades of computing, main memory was extremely slow and incredibly expensive — but CPUs weren’t particularly fast, either. Starting in the 1980s, the gap began to widen quickly. Microprocessor clock speeds took off, but memory access times improved far less dramatically. As this gap grew, it became increasingly clear that a new type of fast memory was needed to bridge the gap.

How caching works: CPU caches are small pools of memory that store information the CPU is most likely to need next. Which information is loaded into cache depends on sophisticated algorithms and certain assumptions about programming code. The goal of the cache system is to ensure that the CPU has the next bit of data it will need already loaded into cache by the time it goes looking for it (also called a cache hit).

A cache miss, on the other hand, means the CPU has to go scampering off to find the data elsewhere. This is where the L2 cache comes into play — while it’s slower, it’s also much larger. Some processors use an inclusive cache design (meaning data stored in the L1 cache is also duplicated in the L2 cache) while

others are exclusive (meaning the two caches never share data). If data can’t be

found in the L2 cache, the CPU continues down the chain to L3 (typically still on-die), then L4 (if it exists) and main memory (DRAM).



This chart shows the relationship between an L1 cache with a constant hit rate, and a larger L2 cache. Note that the total hit rate goes up sharply as the size of the L2 increases. A larger, slower, cheaper L2 can provide all the benefits of a large L1 — but without the die size and power consumption penalty. Most modern L1 cache rates have hit rates far above the theoretical 50 percent shown here — Intel and AMD both typically field cache hit rates of 95 percent or higher.

**Multicore Programming:**

In a multiprogramming system there are one or more programs loaded in main memory which are ready to execute. Only one program at a time is able to get the CPU for executing its instructions (i.e., there is at most one process running on the system) while all the others are waiting their turn. The main idea of multiprogramming is to maximize the use of CPU time. Indeed, suppose the currently running process is performing an I/O task (which, by definition, does not need the CPU to be accomplished). Then, the OS may interrupt that process and give the control to one of the other in-main-memory programs that are ready to execute (i.e. process context switching). In this way, no CPU time is wasted by the system waiting for the I/O task to be completed, and a running process keeps executing until either it voluntarily releases the CPU or when it blocks for an I/O operation. Therefore, the ultimate goal of multiprogramming is to keep the CPU busy as long as there are processes ready to execute. Note that in order for such a system to function properly, the OS must be able to load multiple programs into separate areas of the main memory and provide the required protection to avoid the chance of one process being modified by another one. Other problems that need to be addressed when having multiple programs in memory is fragmentation as programs enter or leave the main memory. Another issue that needs to be handled as well is that large programs may not fit at once in memory which can be solved by using pagination and virtual memory.

**Comparison of Intel and AMD architecture:**

AMD (Advanced Micro Devices) is an American multinational company that creates semiconductors, processors and other hardware for computers. It was founded in 1969 and has been in a constant battle for market share with Intel. Intel Corporation is also an American multinational company that produces semiconductors and other hardware for computing systems. It was founded in 1968, one year before AMD and has managed to capture approximately 70% of the market.

In an attempt to capture the market it is often under legal disputes as it has been known to employ a lot of ‘underhand’ techniques to maintain the leader in the market. In 1975, AMD and Intel had a short partnership working on Intel’s 8080

microprocessor for IBM. However, that partnership quickly dissolved with each company starting its own endeavors. While, Intel’s objective is to provide the most technologically advanced systems, AMD offers the customers’ value for their money. AMD processors are usually cheaper compared to Intel processors of the same architecture. The main rivalry between the companies is often boiled down to motherboards and processors.

The primary difference between AMD and Intel motherboards is that they only accept the same kind of processor. Hence, an AMD motherboard would only work with an AMD processor, and likewise, an Intel motherboard will only work with an Intel processor, and not the other way around. The reason for this is the fact that each processor requires a different socket type. Intel motherboards have LGA 1156 and LGA 1366 sockets, while AMD motherboards have AM2 and AM3 sockets.

However, the fact that an AMD motherboard would only work with an AMD processor, and likewise for Intel, the sales and market share of Intel and AMD motherboards directly correspond to the sales and market share of Intel and AMD processors. The number of slots and the amount of RAM that the motherboard can accommodate is mainly dependent on the make and model of the motherboard. This of course has an effect on pricing. Hence, a motherboard with more SATA ports and/or with more RAM compatibility will cost more.

Pentium is the brand of processors that belong to Intel, while AMD sells the processors under the AMD name itself. The Pentium processor is a consumer-level product. It is placed higher than the low-end Atom and Celeron products but below the faster Core i3, i5, i7 and the Xeon processors. As compared to Intel Core, Pentium has a lower clock frequency, a partially disabled L3 cache, in addition to disabled hyper-threading and virtualization. One of the main difference between the processors is the fact that Intel is often more expensive than its AMD equivalents. This generalization also applies to Pentium. This price difference is often driven by Intel, as it often holds the top spot when comparing the performance of microprocessors.

Also, Intel processors, such as Pentium have longer pipelines than AMD processors. This allows them to have a much higher clock speed than what could be normally achieved. However, AMD has found another way to compete with the increased clock speed, in the means of storing and accessing the CPU memory.

Intel Pentium processors store their memory in an L2 (level 2) cache. This is almost double the size of AMD Athlon processors’ cache. The L2 cache is a memory bank that stores and transmits data to the L1 (level 1) cache. The L1 (level 1) cache in turn, stores and transmits data to the processor itself. Hence, the larger the L2 cache, the faster the processing speed.

AMD Athlon processors have roughly half the L2 cache space of a Pentium processor; however its L2 cache space is integrated directly into the processor itself. This allows AMD Athlon processors to access their cache data much

quicker than Intel Pentium processors, providing a faster processing speed despite its size. So, even though technically AMD Athlon processors’ clock rate and cache space are listed as less on paper, it provides comparable performance to Intel Pentium and all at a relatively lower price.

Both the companies aim to come out with the next best thing and stay a step ahead of each other. Hence, their products are always closely related with minor differences that each of the company thinks will make their product better. Due to this, the processors of the two companies are virtually the same.

|  |  |  |
| --- | --- | --- |
|  | AMD | Intel |
| Industry | Semiconductors | Semiconductors |
| Founder | May 1, 1969 | July 18, 1968 |
| Founder(s) | Jerry Sanders, Edwin Turney | Gordon Moore, Robert Noyce |
| Products | Microprocessors, Motherboard chipset, Graphics processors, Random-access memory and TV tuner-cards | Bluetooth chipsets, flash memory, microprocessors motherboard chipsets and network interface cards |
| Stands for | Advanced Micro Devices | Intel Corporation |
| Price | AMD processor cheaper as compared to Intel processors | Expensive as compared to AMD processors |
| Technology | AMD | As technology wise, Intel is said to have superior processors that outperform the AMD processors |