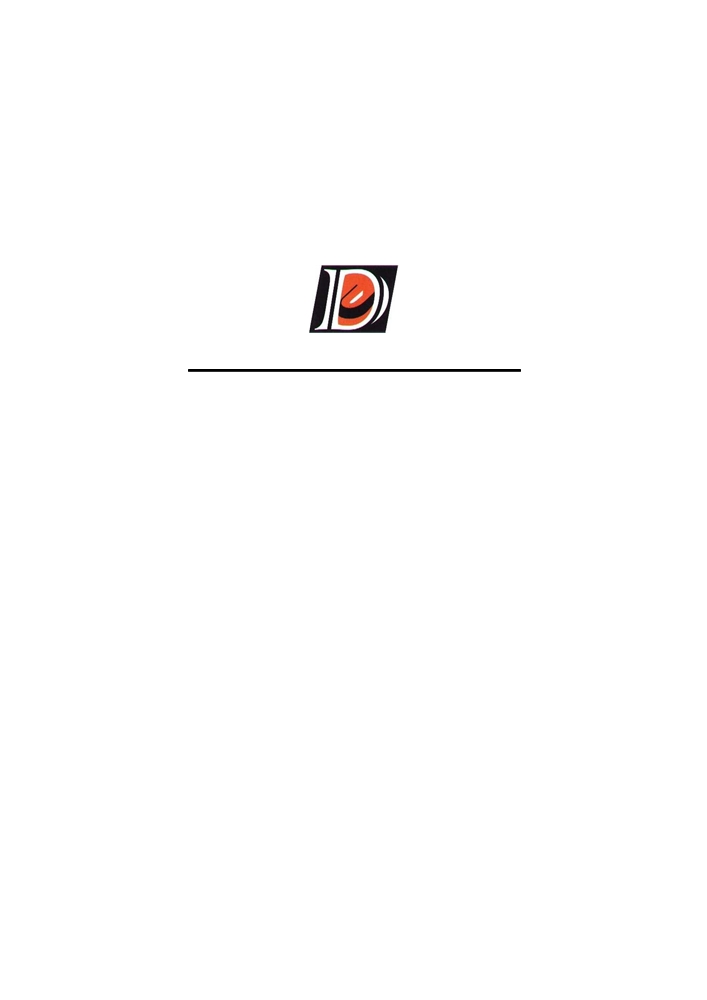
**A PROJECT REPORT ON**

**“Analog Clock”**

**Prepared by**

**Sagar R. Vasoya(IT-108)**

**Guided by**

**Prof.R.S.Chhajed**

**DHARAMSINH DESAI UNIVERSITY**

Department of

**INFORMATION TECHNOLOGY**

**CERTIFICATE**

This is to certify that **Mr.Sagar R. Vasoya** of Department of Information Technology from Dharamsinh Desai University have done their project work on ‘**Analog Clock**’ as a part of their Bechelor of Technology program in Vth  semester.



**Prof.R.S.Chhajjed**

(Head of the Department)

**INTRODUCTION**

The subject of my project is Analog clock.In this project my program will display an analog clock according to system’s time.For implement this, i have used INT 21H and INT 10H interrupts.The functions which are used in program are as follows:BACKGROUND ,IDENTITY,CIRCLE,OTHER,HOUR,MINUTE,SECOND,NUMBER.

In this program, first of all we will get system time using INT 21 interrupt and then as per the time program,we will draw three hands & clock.Then after program will continuously check for time change and then according to that change hands will re-draw.

**FLOW CHART**

END

Get system time using int 21H & Draw three hands according to that

Draw clock without hands

Colour background & display identities

Switch on the video mode of type 13 using INT 10H

Initialize DS,ES & SP

Define one macro for memory to memory transfer

START

**INT 10H**

INT 10h / AH = 0 - set video mode.

input:

AL = desired video mode.

these video modes are supported:

00h - text mode. 40x25. 16 colors. 8 pages.

03h - text mode. 80x25. 16 colors. 8 pages.

13h - graphical mode. 40x25. 256 colors. 320x200 pixels. 1 page.

INT 10h / AH = 01h - set text-mode cursor shape.

input:

CH = cursor start line (bits 0-4) and options (bits 5-7).

CL = bottom cursor line (bits 0-4).

when bit 5 of CH is set to 0, the cursor is visible. when bit 5 is 1, the

cursor is not visible.

; hide blinking text cursor:

mov ch, 32

mov ah, 1

int 10h

; show standard blinking text cursor:

mov ch, 6

mov cl, 7

mov ah, 1

int 10h

; show box-shaped blinking text cursor:

mov ch, 0

mov cl, 7

mov ah, 1

int 10h

; note: some bioses required CL to be >=7,

; otherwise wrong cursor shapes are displayed.

INT 10h / AH = 2 - set cursor position.

input:

DH = row.

DL = column.

BH = page number (0..7).

int 10h

INT 10h / AH = 03h - get cursor position and size.

input:

BH = page number.

return:

DH = row.

DL = column.

CH = cursor start line.

CL = cursor bottom line.

INT 10h / AH = 05h - select active video page.

input:

AL = new page number (0..7).

the activated page is displayed.

INT 10h / AH = 06h - scroll up window.

INT 10h / AH = 07h - scroll down window.

input:

AL = number of lines by which to scroll (00h = clear entire window).

BH = attribute used to write blank lines at bottom of window.

CH, CL = row, column of window's upper left corner.

DH, DL = row, column of window's lower right corner.

INT 10h / AH = 08h - read character and attribute at cursor position.

input:

BH = page number.

return:

AH = attribute.

AL = character.

INT 10h / AH = 09h - write character and attribute at cursor position.

input:

AL = character to display.

BH = page number.

BL = attribute.

CX = number of times to write character.

INT 10h / AH = 0Ah - write character only at cursor position.

input:

AL = character to display.

BH = page number.

CX = number of times to write character.

INT 10h / AH = 0Ch - change color for a single pixel.

input:

AL = pixel color

CX = column.

DX = row.

INT 10h / AH = 0Dh - get color of a single pixel.

input:

CX = column.

DX = row.

output:

AL = pixel color

INT 10h / AH = 0Eh - teletype output.

input:

AL = character to write.

this functions displays a character on the screen, advancing the cursor and

scrolling the screen as necessary. the printing is always done to current active

page.

example:

mov al, 'a'

mov ah, 0eh

int 10h

; note: on specific systems this

; function may not be supported in graphics mode.

INT 10h / AH = 13h - write string.

input:

AL = write mode:

bit 0: update cursor after writing;

bit 1: string contains attributes.

BH = page number.

BL = attribute if string contains only characters (bit 1 of AL is zero).

CX = number of characters in string (attributes are not counted).

DL,DH = column, row at which to start writing.

ES:BP points to string to be printed.

INT 10h / AX = 1003h - toggle intensity/blinking.

input:

BL = write mode:

0: enable intensive colors.

1: enable blinking (not supported by the emulator and windows

command prompt). BH = 0 (to avoid problems on some adapters).

bit color table:

character attribute is 8 bit value, low 4 bits set fore color, high 4 bits set

background color.

note: the emulator and windows command line prompt do not support

background blinking, however to make colors look the same in dos and in full

screen mode it is required to turn off the background blinking.

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

note:

; use this code for compatibility with dos/cmd prompt full screen mode:

mov ax, 1003h

mov bx, 0 ; disable blinking.

int 10h

**INT 21H**

INT 21h / AH=1 - read character from standard input, with echo, result is

stored in AL.

if there is no character in the keyboard buffer, the function waits until any key is

pressed.

INT 21h / AH=2 - write character to standard output.

entry: DL = character to write, after execution AL = DL.

INT 21h / AH=5 - output character to printer.

entry: DL = character to print, after execution AL = DL.

INT 21h / AH=6 - direct console input or output.

parameters for output: DL = 0..254 (ascii code)

parameters for input: DL = 255

for output returns: AL = DL

for input returns: ZF set if no character available and AL = 00h, ZF clear if

character available.

AL = character read; buffer is cleared.

INT 21h / AH=7 - character input without echo to AL.

if there is no character in the keyboard buffer, the function waits until any key is

pressed.

INT 21h / AH=9 - output of a string at DS:DX. String must be terminated by '$'.

INT 21h / AH=0Ah - input of a string to DS:DX, fist byte is buffer size, second

byte is number of chars actually read. this function does not add '$' in the end of

string. to print using INT 21h / AH=9 you must set dollar character at the end

of it and start printing from address DS:DX + 2.

INT 21h / AH=0Bh - get input status;

returns: AL = 00h if no character available, AL = 0FFh if character is available.

INT 21h / AH=0Ch - flush keyboard buffer and read standard input.

entry: AL = number of input function to execute after flushing buffer (can be

01h,06h,07h,08h, or 0Ah - for other values the buffer is flushed but no input is

attempted); other registers as appropriate for the selected input function.

INT 21h / AH= 0Eh - select default drive.

Entry: DL = new default drive (0=A:, 1=B:, etc)

Return: AL = number of potentially valid drive letters

Notes: the return value is the highest drive present.

INT 21h / AH= 19h - get current default drive.

Return: AL = drive (0=A:, 1=B:, etc)

INT 21h / AH=25h - set interrupt vector;

input: AL = interrupt number. DS:DX -> new interrupt handler.

INT 21h / AH=2Ah - get system date;

return: CX = year (1980-2099). DH = month. DL = day. AL = day of week

(00h=Sunday)

INT 21h / AH=2Ch - get system time;

return: CH = hour. CL = minute. DH = second. DL = 1/100 seconds.

INT 21h / AH=35h - get interrupt vector;

entry: AL = interrupt number;

return: ES:BX -> current interrupt handler.

INT 21h / AH= 39h - make directory.

entry: DS:DX -> ASCIZ pathname; zero terminated string,

INT 21h / AH= 3Ah - remove directory.

Entry: DS:DX -> ASCIZ pathname of directory to be removed.

Return:

CF is clear if successful, AX destroyed CF is set on error AX = error code.

Notes: directory must be empty (there should be no files inside of it).

INT 21h / AH= 3Bh - set current directory.

Entry: DS:DX -> ASCIZ pathname to become current directory (max 64 bytes).

Return:

Carry Flag is clear if successful, AX destroyed.

Carry Flag is set on error AX = error code.

Notes: even if new directory name includes a drive letter, the default drive is not

changed,

only the current directory on that drive.

INT 21h / AH= 3Ch - create or truncate file.

xentry:

CX = file attributes:

mov cx, 0 ; normal - no attributes.

mov cx, 1 ; read-only.

mov cx, 2 ; hidden.

mov cx, 4 ; system

mov cx, 7 ; hidden, system and read-only!

mov cx, 16 ; archive

DS:DX -> ASCIZ filename.

returns:

CF clear if successful, AX = file handle.

CF set on error AX = error code.

note: if specified file exists it is deleted without a warning.

INT 21h / AH= 3Dh - open existing file.

Entry:

AL = access and sharing modes:

mov al, 0 ; read

mov al, 1 ; write

mov al, 2 ; read/write

DS:DX -> ASCIZ filename.

Return:

CF clear if successful, AX = file handle.

CF set on error AX = error code.

note 1: file pointer is set to start of file.

note 2: file must exist.

INT 21h / AH= 3Eh - close file.

Entry: BX = file handle

Return:

CF clear if successful, AX destroyed.

CF set on error, AX = error code (06h).

INT 21h / AH= 3Fh - read from file.

Entry:

BX = file handle.

CX = number of bytes to read.

DS:DX -> buffer for data.

Return:

CF is clear if successful - AX = number of bytes actually read; 0 if at EOF (end of

file) before call.

CF is set on error AX = error code.

Note: data is read beginning at current file position, and the file position is

updated after a successful read the returned AX may be smaller than the request

in CX if a partial read occurred.

INT 21h / AH= 40h - write to file.

entry:

BX = file handle.

CX = number of bytes to write.

DS:DX -> data to write.

return:

CF clear if successful; AX = number of bytes actually written.

CF set on error; AX = error code.

note: if CX is zero, no data is written, and the file is truncated or extended to the

current position data is written beginning at the current file position, and the file

position is updated after a successful write the usual cause for AX < CX on

return is a full disk.

INT 21h / AH= 41h - delete file (unlink).

Entry:

DS:DX -> ASCIZ filename (no wildcards, but see notes).

return:

CF clear if successful, AX destroyed. AL is the drive of deleted file

(undocumented).

CF set on error AX = error code.

Note: DOS does not erase the file's data; it merely becomes inaccessible because

the FAT chain for the file is cleared deleting a file which is currently open may

lead to filesystem corruption.

INT 21h / AH= 42h - SEEK - set current file position.

Entry:

AL = origin of move: 0 - start of file. 1 - current file position. 2 - end of file.

BX = file handle.

CX:DX = offset from origin of new file position.

Return:

CF clear if successful, DX:AX = new file position in bytes from start of file.

CF set on error, AX = error code.

Notes:

for origins 1 and 2, the pointer may be positioned before the start of the file; no

error is returned in that case, but subsequent attempts to read or write the file

will produce errors. If the new position is beyond the current end of file, the file

will be extended by the next write (see AH=40h).

INT 21h / AH= 47h - get current directory.

Entry:

DL = drive number (00h = default, 01h = A:, etc)

DS:SI -> 64-byte buffer for ASCIZ pathname.

Return:

Carry is clear if successful

Carry is set on error, AX = error code (0Fh)

Notes:

the returned path does not include a drive and the initial backslash.

INT 21h / AH=4Ch - return control to the operating system (stop program).

INT 21h / AH= 56h - rename file / move file.

Entry:

DS:DX -> ASCIZ filename of existing file.

ES:DI -> ASCIZ new filename.

Return:

CF clear if successful.

CF set on error, AX = error code.

Note: allows move between directories on same logical drive only; open files

should not be renamed!

**ASM FILES**

**(1)Without background continuous clock**

**DATA SEGMENT**

**XC DB 169,177,185,192,200,207,213,219,225,230,234,237,239,240,240,240,239,237,234,230,225,219,213,207,200,192,185,177,169,160,152,144,136,129,121,114,108,102,96,91,87,84,82,81,80,81,82,84,87,91,96,102,108,114,121,129,136,144,152,160**

**YC DB 21,22,24,27,31,35,40,46,52,60,67,75,83,91,100,110,118,126,134,141,149,155,161,166,170,174,177,179,180,180,180,179,177,174,170,166,161,155,149,141,134,126,118,110,100,91,83,75,67,60,52,46,40,35,31,27,24,22,21,20**

**XO DB 240,241,241,79,80,79,160,161,161,161,160,161,160,160,161,161,200,201,201,230,231,231,230,231,231,200,201,201,120,120,121,90,90,91,90,90,91,121,120,120**

**YO DB 101,100,101,100,101,101,181,180,181,20,21,21,100,101,100,101,30,31,30,61,60,61,140,140,141,171,170,171,170,171,171,141,140,140,60,61,61,30,31,30**

**XN DB 25,28,30,28,25,20,15,11,9,11,15,20**

**YN DB 3,7,12,17,21,23,21,17,12,7,3,2**

**VI DB '01','02','03','04','05','06','07','08','09','10','11','12'**

**H1 DB 'HOUR'**

**M1 DB 'MINUTE'**

**SE DB 'SECOND'**

**TITLE1 DB 'ANALOG CLOCK'**

**X1 DW 160**

**X2 DW 160**

**Y1 DW 20**

**Y2 dw 100**

**COLOR DB 02h**

**COUNT DW 0**

**X DW ?**

**Y DW ?**

**ERROR DW ?**

**DIFFX DW ?**

**FLAG DB 0**

**DIFFY DW ?**

**SWAP DW ?**

**S1 DW ?**

**S2 DW ?**

**TEMP DW ?**

**TEMP1 DW 0**

**TEMP2 DW 1**

**TEMP3 DW 1**

**TEMP4 DW 5**

**DATA ENDS**

**STACK\_SEG SEGMENT STACK**

**DW 100 DUP(0)**

**TOP\_STACK LABEL WORD**

**STACK\_SEG ENDS**

**CODE SEGMENT**

**ASSUME CS:CODE,SS:STACK\_SEG,DS:DATA,ES:DATA**

**START: MOV AX,DATA**

**MOV DS,AX**

**MOV ES,AX**

**MOV AX,STACK\_SEG**

**MOV ES,AX**

**MOV SP,TOP\_STACK**

**MEM2MEM MACRO A,B**

**MOV AX,B**

**MOV A,AX**

**ENDM**

**MOV CX,60**

**AGAIN: MOV AX,0013H**

**INT 10H**

**;CALL BACKG**

**CALL IDEN**

**CALL CIRCLE**

**CALL OTHER**

**CALL NUMBER**

**CALL HOUR**

**CALL MINUTE**

**CALL SECOND**

**JMP AGAIN**

**BACKBONE:MOV AH,07**

**INT 21H**

**CMP AL,27**

**JNE BACKBONE**

**INT 3**

**NUMBER PROC**

**PUSH AX**

**PUSH BX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**PUSH ES**

**PUSH DS**

**POP ES**

**LEA SI,XN**

**LEA DI,YN**

**MOV BX,0**

**REPEAT: MOV CX,02**

**MOV DL,DS:BYTE PTR[SI]**

**MOV DH,DS:BYTE PTR[DI]**

**LEA BP,VI[BX]**

**PUSH BX**

**MOV BX,000DH**

**MOV AX,1300H**

**INT 10H**

**INC SI**

**INC DI**

**POP BX**

**INC BX**

**INC BX**

**CMP BX,24**

**JNZ REPEAT**

**POP ES**

**POP BP**

**POP DX**

**POP CX**

**POP BX**

**POP AX**

**RET**

**NUMBER ENDP**

**IDEN PROC**

**MOV AL,color**

**MOV BX,X2**

**MOV CX,Y1**

**MOV DX,Y2**

**MOV BP,x1**

**PUSH AX**

**PUSH BX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**MOV COLOR,02H**

**MOV X1,0**

**MOV Y1,196**

**MOV X2,8**

**MOV Y2,196**

**CALL bresenham**

**mov color,01**

**MOV X1,0**

**MOV Y1,188**

**MOV X2,8**

**MOV Y2,188**

**call bresenham**

**MOV color,0eh**

**mov X1,0**

**mov Y1,180**

**MOV X2,8**

**MOV Y2,180**

**call bresenham**

**MOV AX,1300H**

**MOV BX,0004H**

**MOV CX,4**

**MOV DH,24**

**MOV DL,2**

**LEA BP,H1**

**MOV SI,DS**

**MOV ES,SI**

**INT 10H**

**MOV AX,1300H**

**MOV BX,0004H**

**MOV CX,6**

**MOV DH,23**

**MOV DL,2**

**LEA BP,M1**

**MOV SI,DS**

**MOV ES,SI**

**INT 10H**

**MOV AX,1300H**

**MOV BX,0004H**

**MOV CX,6**

**MOV DH,22**

**MOV DL,2**

**LEA BP,SE**

**MOV SI,DS**

**MOV ES,SI**

**INT 10H**

**MOV AX,1300H**

**MOV BX,0004H**

**MOV CX,12**

**MOV DH,1**

**MOV DL,15**

**LEA BP,TITLE1**

**MOV SI,DS**

**MOV ES,SI**

**INT 10H**

**POP BP**

**POP DX**

**POP CX**

**POP BX**

**POP AX**

**MOV X1,BP**

**MOV X2,BX**

**MOV Y1,CX**

**MOV Y2,DX**

**MOV color,AL**

**RET**

**IDEN ENDP**

**BACKG PROC**

**PUSH AX**

**PUSH BX**

**MOV BX,000AH**

**MOV AX,0C09H**

**MOV CX,01H**

**REPEAT2:MOV DX,01H**

**REPEAT1:INT 10H**

**INC DX**

**CMP DX,200**

**JNZ REPEAT1**

**INC CX**

**CMP CX,320**

**JNZ REPEAT2**

**POP BX**

**POP AX**

**RET**

**BACKG ENDP**

**HOUR PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**MOV COLOR,02**

**MOV AH,2CH**

**INT 21H**

**MOV CL,CH**

**XOR CH,CH**

**MOV AX,CX**

**MUL TEMP4**

**MOV CX,AX**

**SUB CX,1**

**MOV BP,CX**

**MOV AH,00H**

**MOV AL,DS:XC[BP]**

**MOV X1,AX**

**MOV AL,DS:YC[BP]**

**MOV Y1,AX**

**CALL bresenham**

**POP BP**

**POP DX**

**POP CX**

**POP AX**

**RET**

**HOUR ENDP**

**MINUTE PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**X**

**MOV COLOR,01**

**MOV AH,2CH**

**INT 21H**

**MOV CH,00H**

**SUB CX,1**

**MOV BP,CX**

**MOV AH,00H**

**MOV AL,DS:XC[BP]**

**MOV X1,AX**

**MOV AL,DS:YC[BP]**

**MOV Y1,AX**

**CALL bresenham**

**POP BP**

**POP DX**

**POP CX**

**POP AX**

**RET**

**MINUTE ENDP**

**SECOND PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**MOV COLOR,0Eh**

**MOV AH,2CH**

**INT 21H**

**MOV DL,DH**

**XOR DH,DH**

**SUB DX,1**

**MOV BP,DX**

**MOV AH,00H**

**MOV AL,DS:XC[BP]**

**MOV X1,AX**

**MOV AL,DS:YC[BP]**

**MOV Y1,AX**

**CALL bresenham**

**POP BP**

**POP DX**

**POP CX**

**POP AX**

**RET**

**SECOND ENDP**

**CIRCLE PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH SI**

**PUSH DI**

**LEA SI,XC**

**LEA DI,YC**

**MOV CX,60**

**SAGAR: MOV AX,0C04H**

**PUSH CX**

**MOV DH,00H**

**MOV CH,00H**

**MOV CL,DS:BYTE PTR[SI]**

**MOV DL,DS:BYTE PTR[DI]**

**INT 10H**

**INC SI**

**INC DI**

**POP CX**

**DEC CX**

**CMP CX,00H**

**JNZ SAGAR**

**POP DI**

**POP SI**

**POP DX**

**POP CX**

**POP AX**

**RET**

**CIRCLE ENDP**

**OTHER PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH SI**

**PUSH DI**

**LEA SI,XO**

**LEA DI,YO**

**MOV CX,40**

**SAGAR1: MOV AX,0C04H**

**PUSH CX**

**MOV DH,00H**

**MOV CH,00H**

**MOV CL,DS:BYTE PTR[SI]**

**MOV DL,DS:BYTE PTR[DI]**

**INT 10H**

**INC SI**

**INC DI**

**POP CX**

**DEC CX**

**CMP CX,00H**

**JNZ SAGAR1**

**POP DI**

**POP SI**

**POP DX**

**POP CX**

**POP AX**

**RET**

**OTHER ENDP**

**BRESENHAM PROC ; Bresenham line algorithm**

**mov ax, y2**

**cmp ax, y1**

**jne cont3**

**mov ax, x2**

**cmp ax, x1**

**jne cont3**

**ret**

**cont3 : mov ax, y2**

**sub ax, y1**

**jnc next1**

**neg ax**

**mov s2, 0**

**sub s2, 1**

**mov diffy, ax**

**jmp x3**

**next1: mov diffy, ax**

**cmp diffy, 0**

**je eq1**

**mov s2, 1**

**jmp x3**

**eq1: mov s2, 0**

**x3: mov ax, x2**

**sub ax, x1**

**jnc next2**

**neg ax**

**mov s1, 0**

**sub s1, 1**

**mov diffx, ax**

**jmp calc**

**next2: mov diffx, ax**

**cmp diffx, 0**

**je eq2**

**mov s1, 1**

**jmp calc**

**eq2: mov s1, 0**

**calc: mem2mem x, x1**

**mem2mem y, y1**

**mov ax, diffx**

**cmp diffy, ax**

**jnc more**

**mov swap, 0**

**jmp cont**

**more: mem2mem temp, diffx**

**mem2mem diffx, diffy**

**mem2mem diffy, temp**

**mov swap, 1**

**cont: mov ax, diffy**

**shl ax, 1**

**sub ax, diffx**

**mov error, ax**

**mov count, 1**

**for: call putpixel**

**while1: cmp error, 0**

**jl fin**

**cmp swap, 1**

**je forx**

**mov ax, y**

**add ax, s2**

**mov y, ax**

**jmp fory**

**forx: mov ax, x**

**add ax, s1**

**mov x, ax**

**fory:mov ax, diffx**

**shl ax, 1**

**sub error, ax**

**jmp while1**

**fin: cmp swap, 1**

**je fry**

**mov ax, x**

**add ax, s1**

**mov x, ax**

**jmp frx**

**fry: mov ax, y**

**add ax, s2**

**mov y, ax**

**frx: mov ax, diffy**

**shl ax, 1**

**add error, ax**

**inc count**

**mov ax, diffx**

**cmp ax, count**

**jnc for**

**RET**

**BRESENHAM ENDP**

**X**

**PUTPIXEL PROC ; Put a pixel on screen**

**cmp flag, 0**

**je first**

**cmp y, 41**

**jc dontput**

**cmp y, 478**

**ja dontput**

**cmp x, 0**

**je dontput**

**cmp x, 638**

**ja dontput**

**first : mov cx, x ; Getting coordinates of**

**mov dx, y ; target pixel.**

**mov ah, 0ch**

**mov al, color ; Color of pixel.**

**int 10h**

**dontput : RET**

**ENDP PUTPIXEL**

**CODE ENDS**

**END START**

**(2)With background steady clock**

**DATA SEGMENT**

**XC DB 169,177,185,192,200,207,213,219,225,230,234,237,239,240,240,240,239,237,234,230,225,219,213,207,200,192,185,177,169,160,152,144,136,129,121,114,108,102,96,91,87,84,82,81,80,81,82,84,87,91,96,102,108,114,121,129,136,144,152,160**

**YC DB 21,22,24,27,31,35,40,46,52,60,67,75,83,91,100,110,118,126,134,141,149,155,161,166,170,174,177,179,180,180,180,179,177,174,170,166,161,155,149,141,134,126,118,110,100,91,83,75,67,60,52,46,40,35,31,27,24,22,21,20**

**XO DB 240,241,241,79,80,79,160,161,161,161,160,161,160,160,161,161,200,201,201,230,231,231,230,231,231,200,201,201,120,120,121,90,90,91,90,90,91,121,120,120**

**YO DB 101,100,101,100,101,101,181,180,181,20,21,21,100,101,100,101,30,31,30,61,60,61,140,140,141,171,170,171,170,171,171,141,140,140,60,61,61,30,31,30**

**XN DB 25,28,30,28,25,20,15,11,9,11,15,20**

**YN DB 3,7,12,17,21,23,21,17,12,7,3,2**

**VI DB '01','02','03','04','05','06','07','08','09','10','11','12'**

**H1 DB 'HOUR'**

**M1 DB 'MINUTE'**

**SE DB 'SECOND'**

**TITLE1 DB 'ANALOG CLOCK'**

**X1 DW 160**

**X2 DW 160**

**Y1 DW 20**

**Y2 dw 100**

**COLOR DB 02h**

**COUNT DW 0**

**X DW ?**

**Y DW ?**

**ERROR DW ?**

**DIFFX DW ?**

**FLAG DB 0**

**DIFFY DW ?**

**SWAP DW ?**

**S1 DW ?**

**S2 DW ?**

**TEMP DW ?**

**TEMP1 DW 0**

**TEMP2 DW 1**

**TEMP3 DW 1**

**TEMP4 DW 5**

**DATA ENDS**

**STACK\_SEG SEGMENT STACK**

**DW 100 DUP(0)**

**TOP\_STACK LABEL WORD**

**STACK\_SEG ENDS**

**CODE SEGMENT**

**ASSUME CS:CODE,SS:STACK\_SEG,DS:DATA,ES:DATA**

**START: MOV AX,DATA**

**MOV DS,AX**

**MOV ES,AX**

**MOV AX,STACK\_SEG**

**MOV ES,AX**

**MOV SP,TOP\_STACK**

**MEM2MEM MACRO A,B**

**MOV AX,B**

**MOV A,AX**

**ENDM**

**MOV CX,60**

**AGAIN: MOV AX,0013H**

**INT 10H**

**CALL BACKG**

**CALL IDEN**

**CALL CIRCLE**

**CALL OTHER**

**CALL NUMBER**

**CALL HOUR**

**CALL MINUTE**

**CALL SECOND**

**BACKBONE:MOV AH,07**

**INT 21H**

**CMP AL,27**

**JNE BACKBONE**

**INT 3**

**NUMBER PROC**

**PUSH AX**

**PUSH BX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**PUSH ES**

**PUSH DS**

**POP ES**

**LEA SI,XN**

**LEA DI,YN**

**MOV BX,0**

**REPEAT: MOV CX,02**

**MOV DL,DS:BYTE PTR[SI]**

**MOV DH,DS:BYTE PTR[DI]**

**LEA BP,VI[BX]**

**PUSH BX**

**MOV BX,000DH**

**MOV AX,1300H**

**INT 10H**

**INC SI**

**INC DI**

**POP BX**

**INC BX**

**INC BX**

**CMP BX,24**

**JNZ REPEAT**

**POP ES**

**POP BP**

**POP DX**

**POP CX**

**POP BX**

**POP AX**

**RET**

**NUMBER ENDP**

**IDEN PROC**

**MOV AL,color**

**MOV BX,X2**

**MOV CX,Y1**

**MOV DX,Y2**

**MOV BP,x1**

**PUSH AX**

**PUSH BX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**MOV COLOR,02H**

**MOV X1,0**

**MOV Y1,196**

**MOV X2,8**

**MOV Y2,196**

**CALL bresenham**

**mov color,01**

**MOV X1,0**

**MOV Y1,188**

**MOV X2,8**

**MOV Y2,188**

**call bresenham**

**MOV color,0eh**

**mov X1,0**

**mov Y1,180**

**MOV X2,8**

**MOV Y2,180**

**call bresenham**

**MOV AX,1300H**

**MOV BX,0004H**

**MOV CX,4**

**MOV DH,24**

**MOV DL,2**

**LEA BP,H1**

**MOV SI,DS**

**MOV ES,SI**

**INT 10H**

**MOV AX,1300H**

**MOV BX,0004H**

**MOV CX,6**

**MOV DH,23**

**MOV DL,2**

**LEA BP,M1**

**MOV SI,DS**

**MOV ES,SI**

**INT 10H**

**MOV AX,1300H**

**MOV BX,0004H**

**MOV CX,6**

**MOV DH,22**

**MOV DL,2**

**LEA BP,SE**

**MOV SI,DS**

**MOV ES,SI**

**INT 10H**

**MOV AX,1300H**

**MOV BX,0004H**

**MOV CX,12**

**MOV DH,1**

**MOV DL,15**

**LEA BP,TITLE1**

**MOV SI,DS**

**MOV ES,SI**

**INT 10H**

**POP BP**

**POP DX**

**POP CX**

**POP BX**

**POP AX**

**MOV X1,BP**

**MOV X2,BX**

**MOV Y1,CX**

**MOV Y2,DX**

**MOV color,AL**

**RET**

**IDEN ENDP**

**BACKG PROC**

**PUSH AX**

**PUSH BX**

**MOV BX,000AH**

**MOV AX,0C09H**

**MOV CX,01H**

**REPEAT2:MOV DX,01H**

**REPEAT1:INT 10H**

**INC DX**

**CMP DX,200**

**JNZ REPEAT1**

**INC CX**

**CMP CX,320**

**JNZ REPEAT2**

**POP BX**

**POP AX**

**RET**

**BACKG ENDP**

**HOUR PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**MOV COLOR,02**

**MOV AH,2CH**

**INT 21H**

**MOV CL,CH**

**XOR CH,CH**

**MOV AX,CX**

**MUL TEMP4**

**MOV CX,AX**

**SUB CX,1**

**MOV BP,CX**

**MOV AH,00H**

**MOV AL,DS:XC[BP]**

**MOV X1,AX**

**MOV AL,DS:YC[BP]**

**MOV Y1,AX**

**CALL bresenham**

**POP BP**

**POP DX**

**POP CX**

**POP AX**

**RET**

**HOUR ENDP**

**MINUTE PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**MOV COLOR,01**

**MOV AH,2CH**

**INT 21H**

**MOV CH,00H**

**SUB CX,1**

**MOV BP,CX**

**MOV AH,00H**

**MOV AL,DS:XC[BP]**

**MOV X1,AX**

**MOV AL,DS:YC[BP]**

**MOV Y1,AX**

**CALL bresenham**

**POP BP**

**POP DX**

**POP CX**

**POP AX**

**RET**

**MINUTE ENDP**

**SECOND PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH BP**

**MOV COLOR,0Eh**

**MOV AH,2CH**

**INT 21H**

**MOV DL,DH**

**XOR DH,DH**

**SUB DX,1**

**MOV BP,DX**

**MOV AH,00H**

**MOV AL,DS:XC[BP]**

**MOV X1,AX**

**MOV AL,DS:YC[BP]**

**MOV Y1,AX**

**CALL bresenham**

**POP BP**

**POP DX**

**POP CX**

**POP AX**

**RET**

**SECOND ENDP**

**CIRCLE PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH SI**

**PUSH DI**

**LEA SI,XC**

**LEA DI,YC**

**MOV CX,60**

**SAGAR: MOV AX,0C04H**

**PUSH CX**

**MOV DH,00H**

**MOV CH,00H**

**MOV CL,DS:BYTE PTR[SI]**

**MOV DL,DS:BYTE PTR[DI]**

**INT 10H**

**INC SI**

**INC DI**

**POP CX**

**DEC CX**

**CMP CX,00H**

**JNZ SAGAR**

**POP DI**

**POP SI**

**POP DX**

**POP CX**

**POP AX**

**RET**

**CIRCLE ENDP**

**OTHER PROC**

**PUSH AX**

**PUSH CX**

**PUSH DX**

**PUSH SI**

**PUSH DI**

**LEA SI,XO**

**LEA DI,YO**

**MOV CX,40**

**SAGAR1: MOV AX,0C04H**

**PUSH CX**

**MOV DH,00H**

**MOV CH,00H**

**MOV CL,DS:BYTE PTR[SI]**

**MOV DL,DS:BYTE PTR[DI]**

**INT 10H**

**INC SI**

**INC DI**

**POP CX**

**DEC CX**

**CMP CX,00H**

**JNZ SAGAR1**

**POP DI**

**POP SI**

**POP DX**

**POP CX**

**POP AX**

**RET**

**OTHER ENDP**

**BRESENHAM PROC ; Bresenham line algorithm**

**mov ax, y2**

**cmp ax, y1**

**jne cont3**

**mov ax, x2**

**cmp ax, x1**

**jne cont3**

**ret**

**cont3 : mov ax, y2**

**sub ax, y1**

**jnc next1**

**neg ax**

**mov s2, 0**

**sub s2, 1**

**mov diffy, ax**

**jmp x3**

**next1: mov diffy, ax**

**cmp diffy, 0**

**je eq1**

**mov s2, 1**

**jmp x3**

**eq1: mov s2, 0**

**x3: mov ax, x2**

**sub ax, x1**

**jnc next2**

**neg ax**

**mov s1, 0**

**sub s1, 1**

**mov diffx, ax**

**jmp calc**

**next2: mov diffx, ax**

**cmp diffx, 0**

**je eq2**

**mov s1, 1**

**jmp calc**

**eq2: mov s1, 0**

**calc: mem2mem x, x1**

**mem2mem y, y1**

**mov ax, diffx**

**cmp diffy, ax**

**jnc more**

**mov swap, 0**

**jmp cont**

**more: mem2mem temp, diffx**

**mem2mem diffx, diffy**

**mem2mem diffy, temp**

**mov swap, 1**

**cont: mov ax, diffy**

**shl ax, 1**

**sub ax, diffx**

**mov error, ax**

**mov count, 1**

**for: call putpixel**

**while1: cmp error, 0**

**jl fin**

**cmp swap, 1**

**je forx**

**mov ax, y**

**add ax, s2**

**mov y, ax**

**jmp fory**

**forx: mov ax, x**

**add ax, s1**

**mov x, ax**

**fory:mov ax, diffx**

**shl ax, 1**

**sub error, ax**

**jmp while1**

**fin: cmp swap, 1**

**je fry**

**mov ax, x**

**add ax, s1**

**mov x, ax**

**jmp frx**

**fry: mov ax, y**

**add ax, s2**

**mov y, ax**

**frx: mov ax, diffy**

**shl ax, 1**

**add error, ax**

**inc count**

**mov ax, diffx**

**cmp ax, count**

**jnc for**

**RET**

**BRESENHAM ENDP**

**PUTPIXEL PROC ; Put a pixel on screen**

**cmp flag, 0**

**je first**

**cmp y, 41**

**jc dontput**

**cmp y, 478**

**ja dontput**

**cmp x, 0**

**je dontput**

**cmp x, 638**

**ja dontput**

**first : mov cx, x ; Getting coordinates of**

**mov dx, y ; target pixel.**

**mov ah, 0ch**

**mov al, color ; Color of pixel.**

**int 10h**

**dontput : RET**

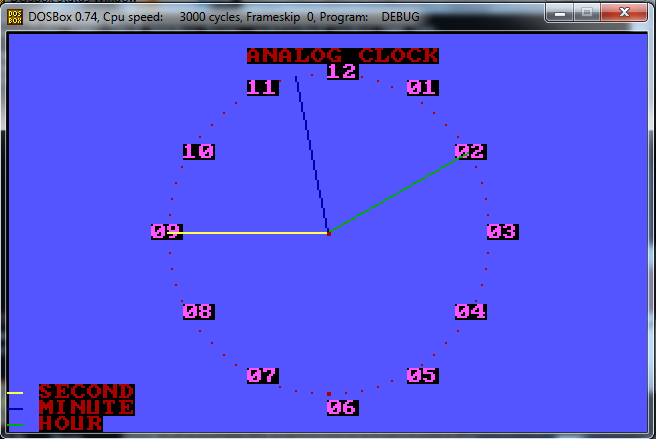
**ENDP PUTPIXEL**

**CODE ENDS**

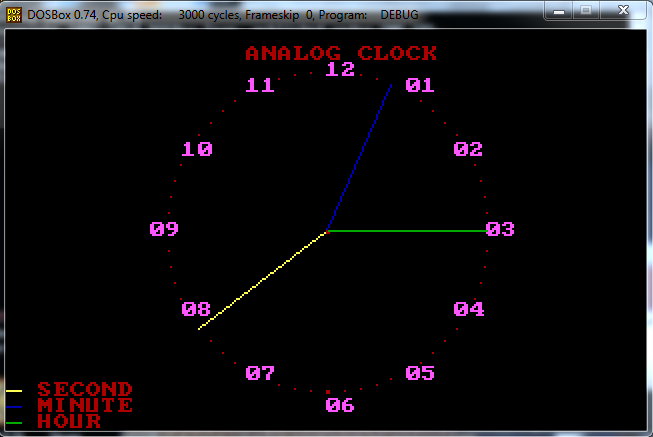
**END START**

**SNAPS**

**(1)With background steady clock**

****

(2**)Without background continuous clock**

**xxx**