Task 3: ball.asm

Description of the task

The main goal of this lesson to demonstrate the scheduling options in assembler routines. The scheduling happen with the AH00, INT1Ah function. The program simulates a "ball" falling off. An „O” character symbolize the ball, that falls off gradually accelerating, and than rebound. The rebound is lossless, so the ball reach the starting positions, while it slow down gradually.

Introducing the program:

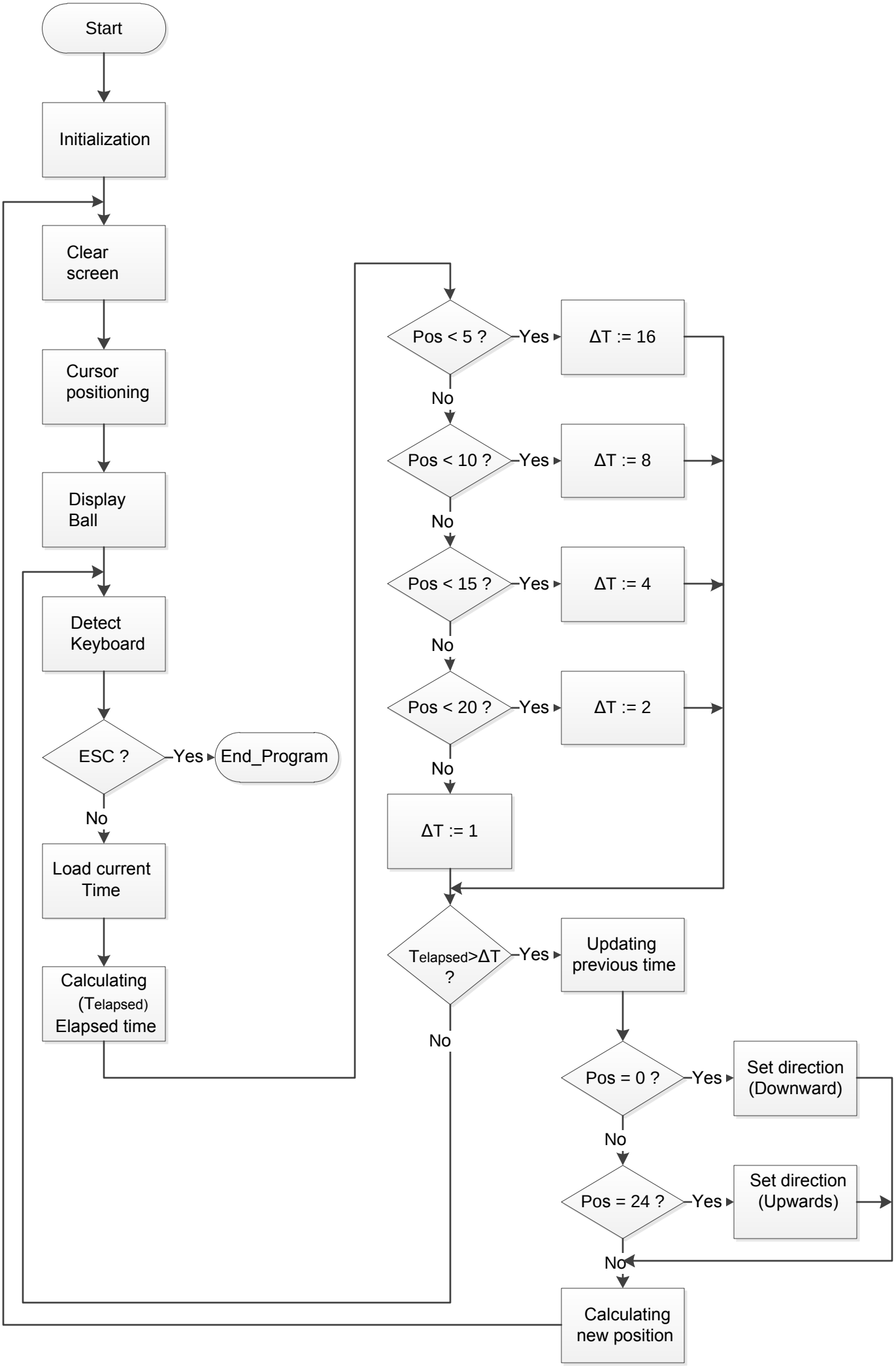
- First the program has to be prepared with the exception of delaying routines. Now the program waiting for a keytsroke, that realizes through the AH00, INT16h function, instead of keyboard monitoring. Thus, the program will only step to the next iteration, if a key is pressed. Must be checked that the "ball" do not move out of the screen. The lowest position is reached, the ball moves upwards, and passing down after achieving highest position.
- After testing the program should be corrected the part of waiting for a keytsroke to the AH01, INT16h function. This function doesn't interrupt the program, it's only changes the states of FLAGs, and modifies the value of AL register, if a keystroke was detected. Should be completed the delaying part of the program. The basic element of the delay is the AH00, INT1Ah function, that loads the system time to CX:DX registers. One counter change is about 1/18 sec.

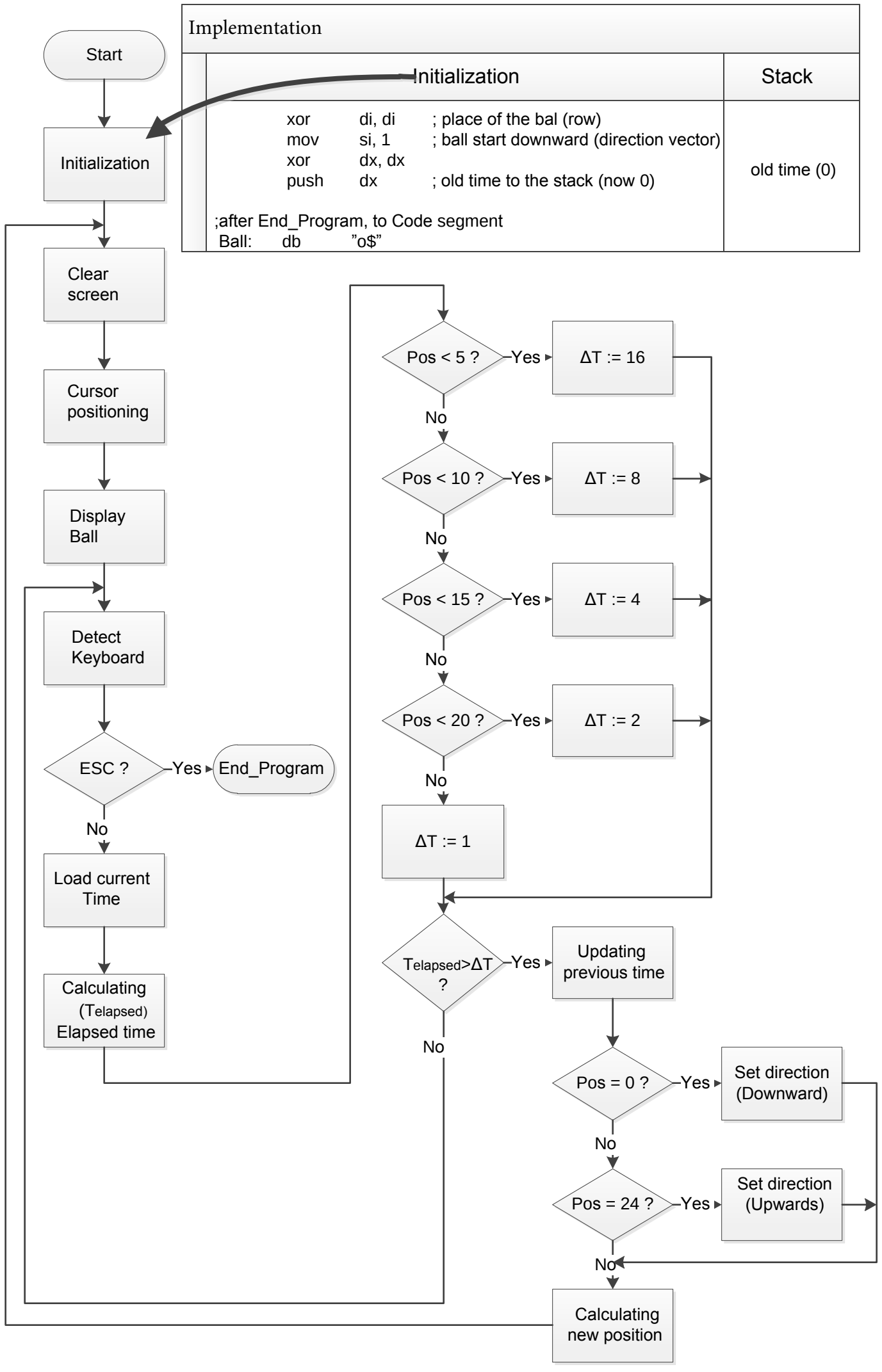
Implementation

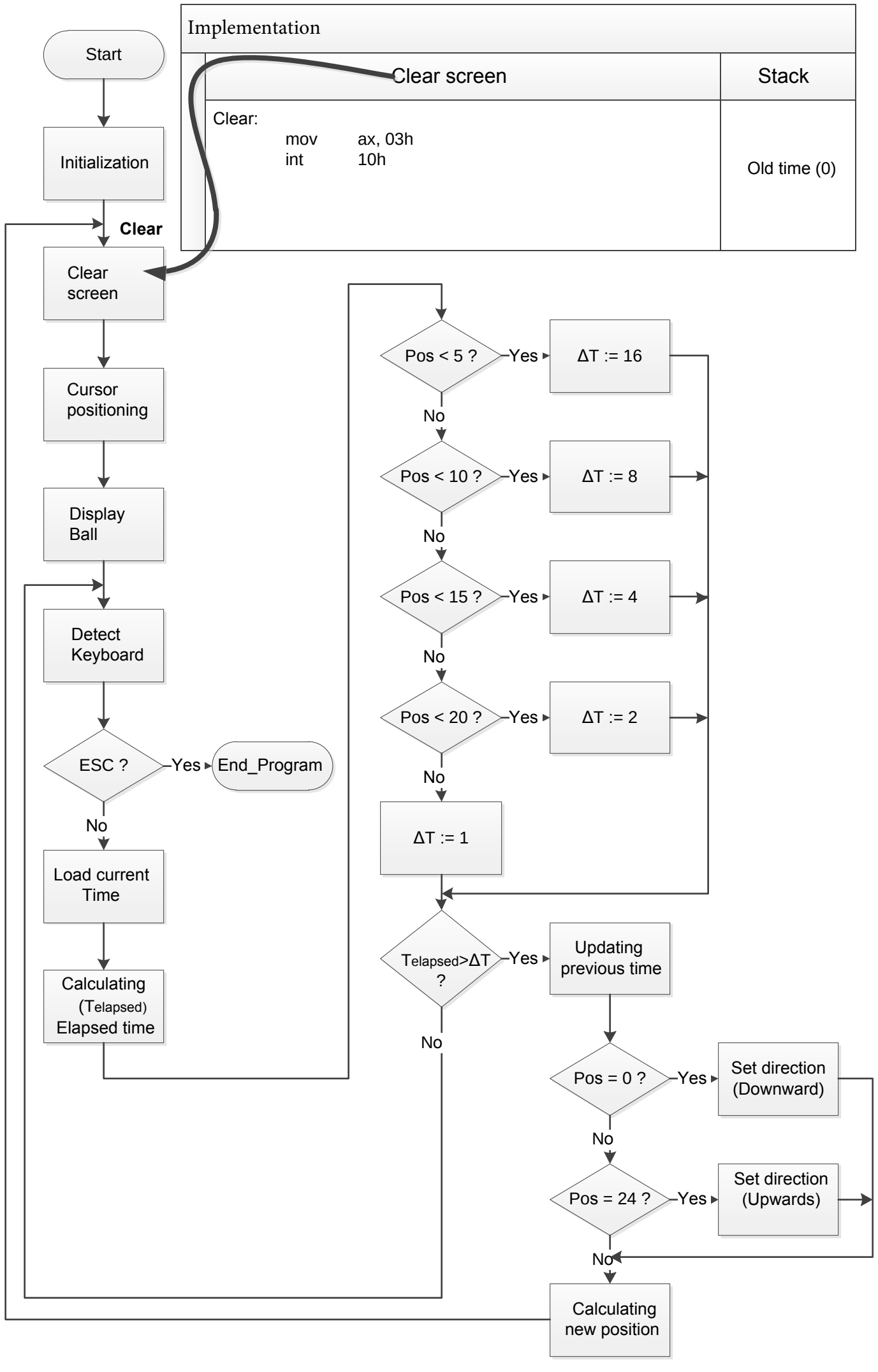
```
Code    Segment
        assume CS:Code,
        DS:Data, SS:Stack
Start:
        mov     ax, Code
        mov     DS, ax

        xor     di, di
        mov     si, 1

        xor     dx, dx
        push    dx
Delete:
        mov     ax, 03h
        int     10h

        mov     dx, di
        mov     dh, dl

        mov     dl, 40
        xor     bh, bh
        mov     ah, 02h
        int     10h

        mov     dx, offset Ball
        mov     ah, 09h
        int     21h
Delay:
        mov     ah, 01h
        int     16h
        ;jnz    End_Program

        jz      Nokey
        mov     ah, 00h
        int     16h
        cmp     al, 27
        jz      End_Program
Nokey:
        xor     ah, ah
        int     1ah

        pop     cx
        push    cx
        mov     ax, dx
        sub     dx, cx
        push    ax

        cmp     di, 5
        jnc     Time1
        mov     al, 16
        jmp     Set
Time1:
        cmp     di, 10
        jnc     Time2
        mov     al, 8
        jmp     Set
Time2:
        cmp     di, 15
        jnc     Time3:
        mov     al, 4
        jmp     Set
Time3:
```

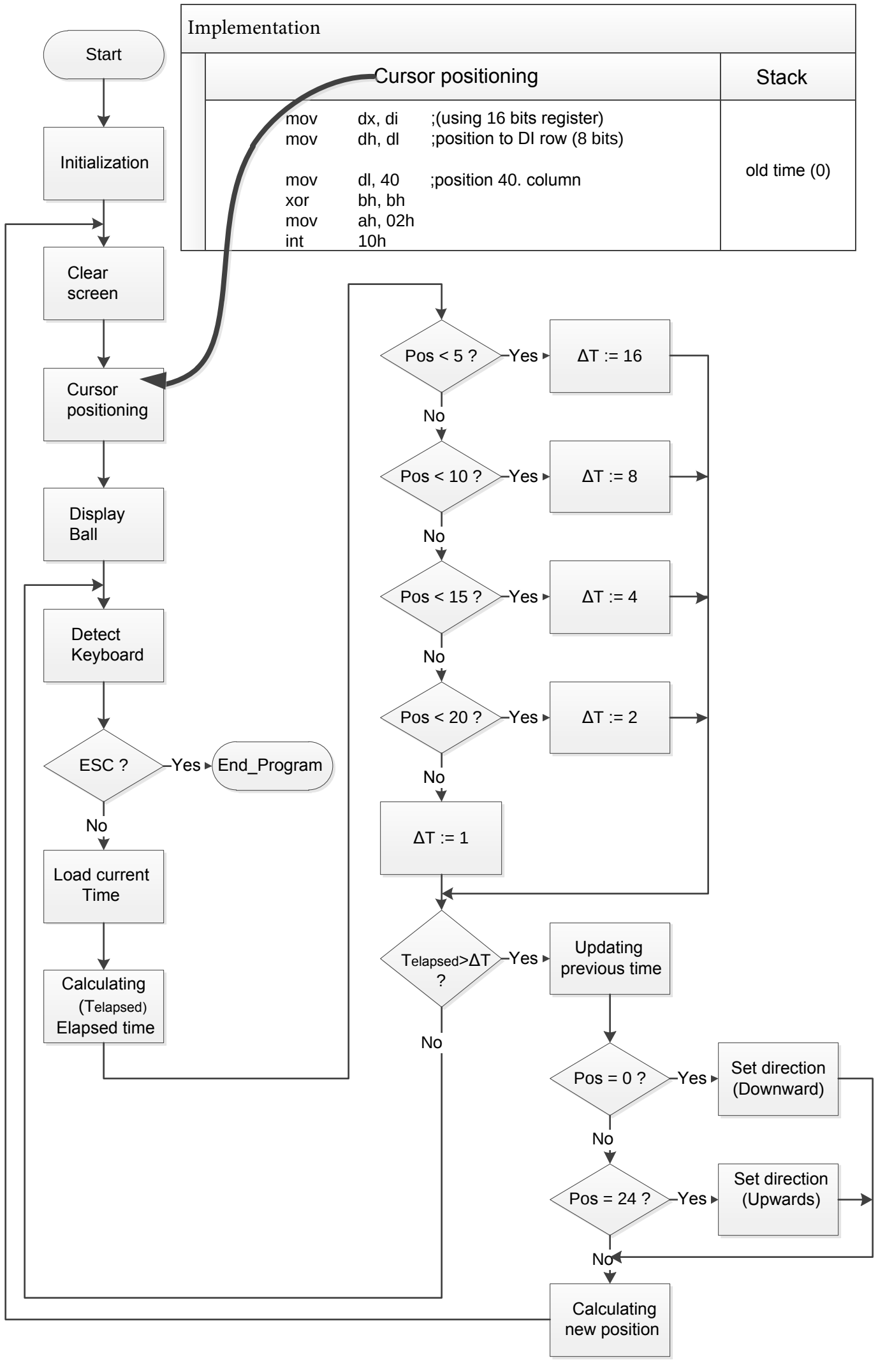
Implementation

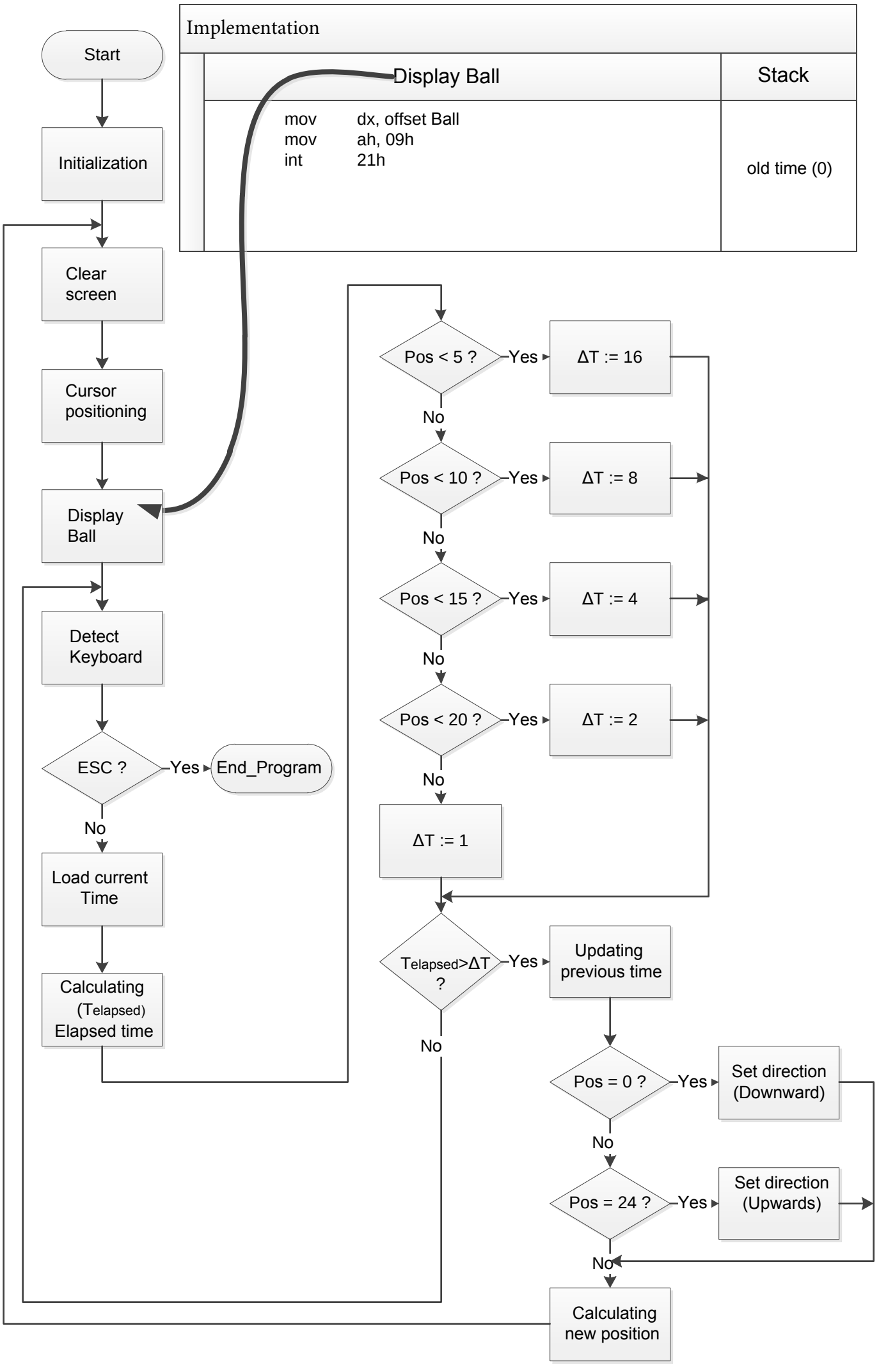
```
        cmp     di, 20
        jnc     Time4
        mov     al, 2
        jmp     Set
Time4:
        mov     al, 1
Set:
        xor     ah, ah
        cmp     dx, ax

        pop     ax

        jc      Delay

        pop     cx
        push    ax

        cmp     di, 0
        jz      Downward

        cmp     di, 24
        jz      Upward
Movement:
        add     di, si
        jmp     Delete
Downward:
        mov     si, 1
        jmp     Movement
Upwards:
        mov     si, -1
        jmp     Movement

End_Program:
        pop     cx
        mov     ax, 4c00h
        int     21h

Ball:
        db      "O$"

Code    Ends

Data    Segment
Data    Ends

Stack   Segment
Stack   Ends

        End Start
```

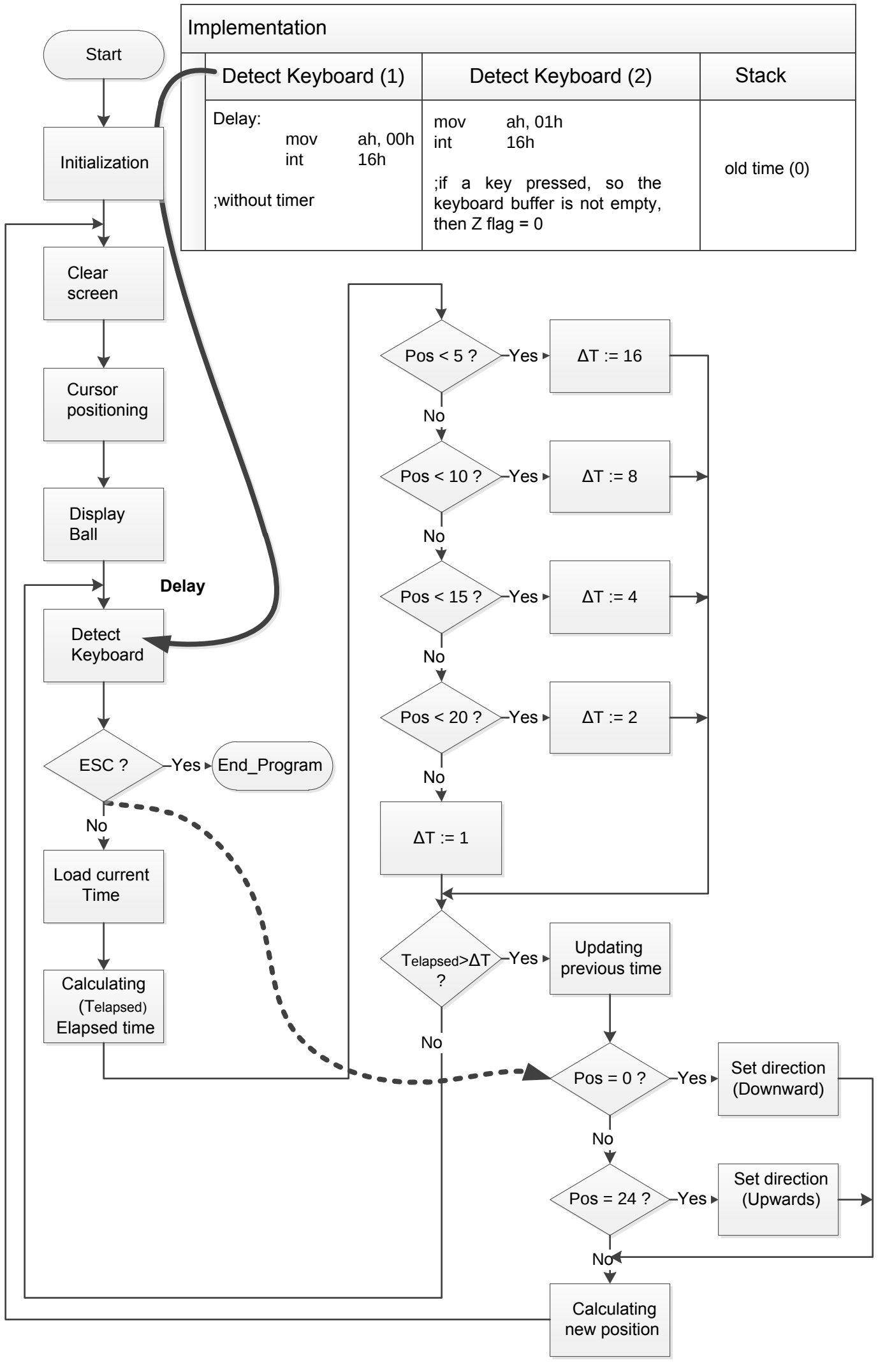

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

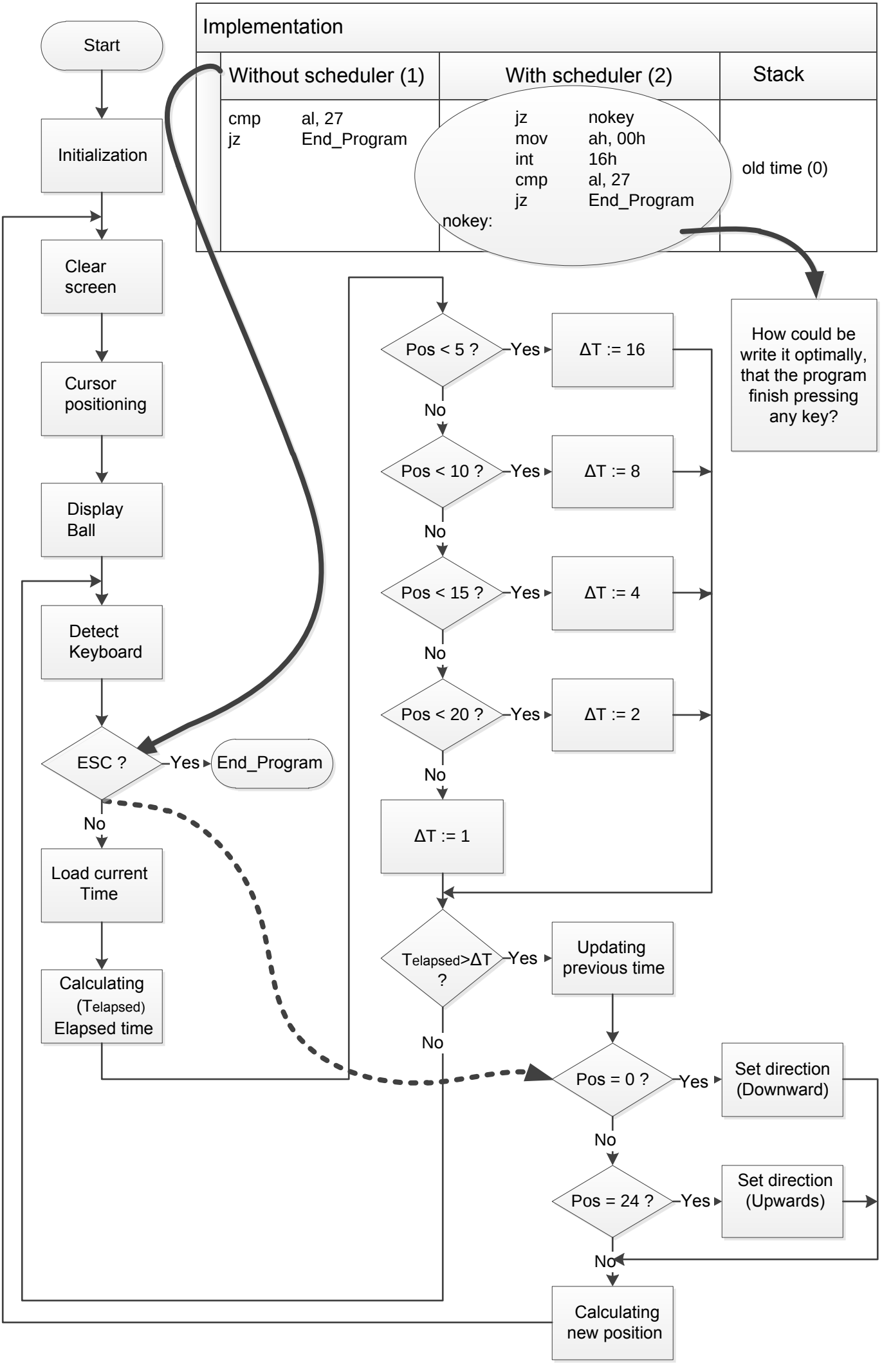

Implementation
Initialization
Stack
xor di, di ; place of the bal (row)
mov si, 1 ; ball start downward (direction vector)
xor dx, dx
push dx ; old time to the stack (now 0)
old time (0)
;after End_Program, to Code segment
Ball: db "o$"
Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

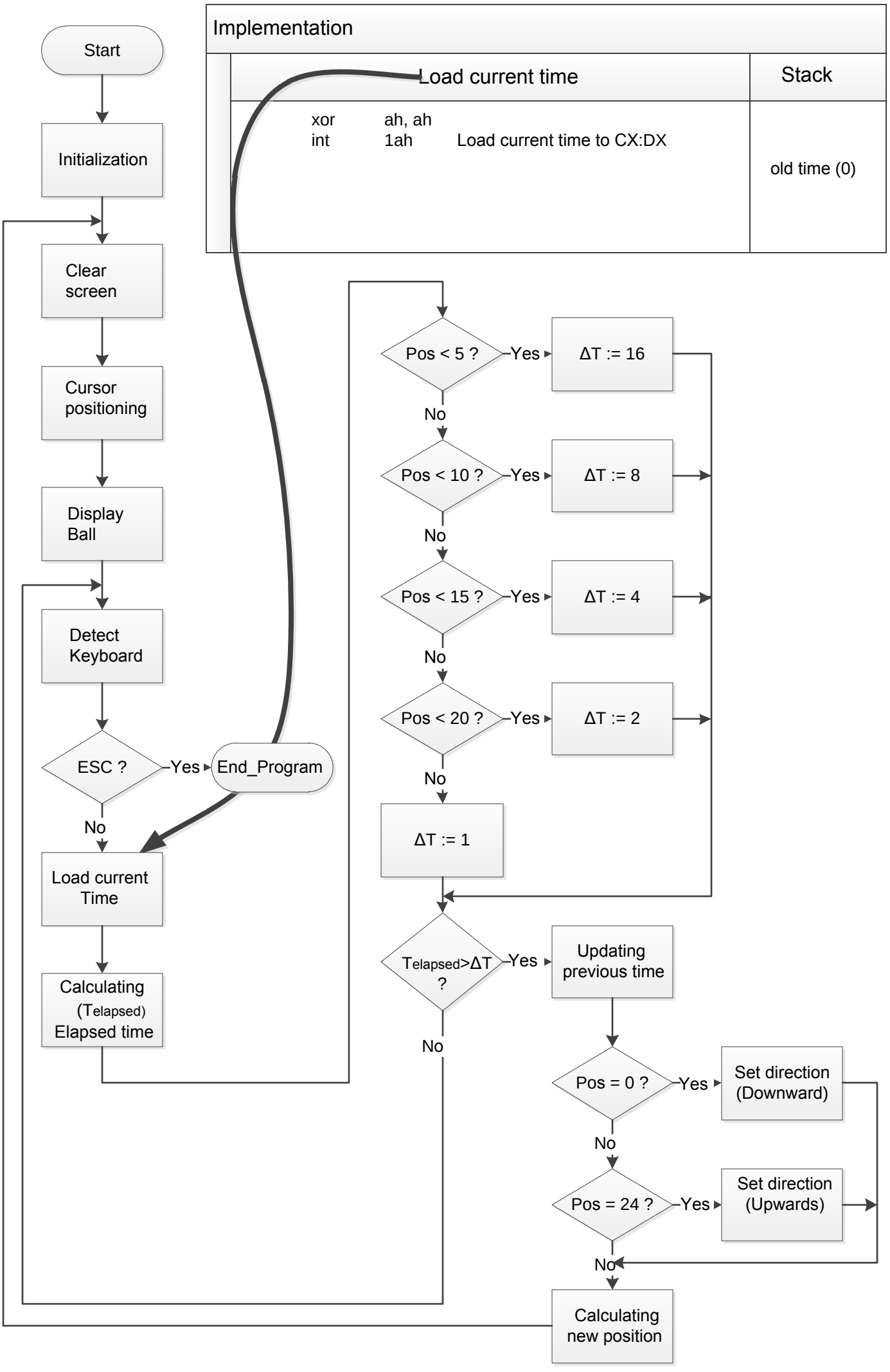

Implementation
Clear screen
Stack
Clear:
mov ax, 03h
int 10h
Old time (0)
Start
Initialization
Clear
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

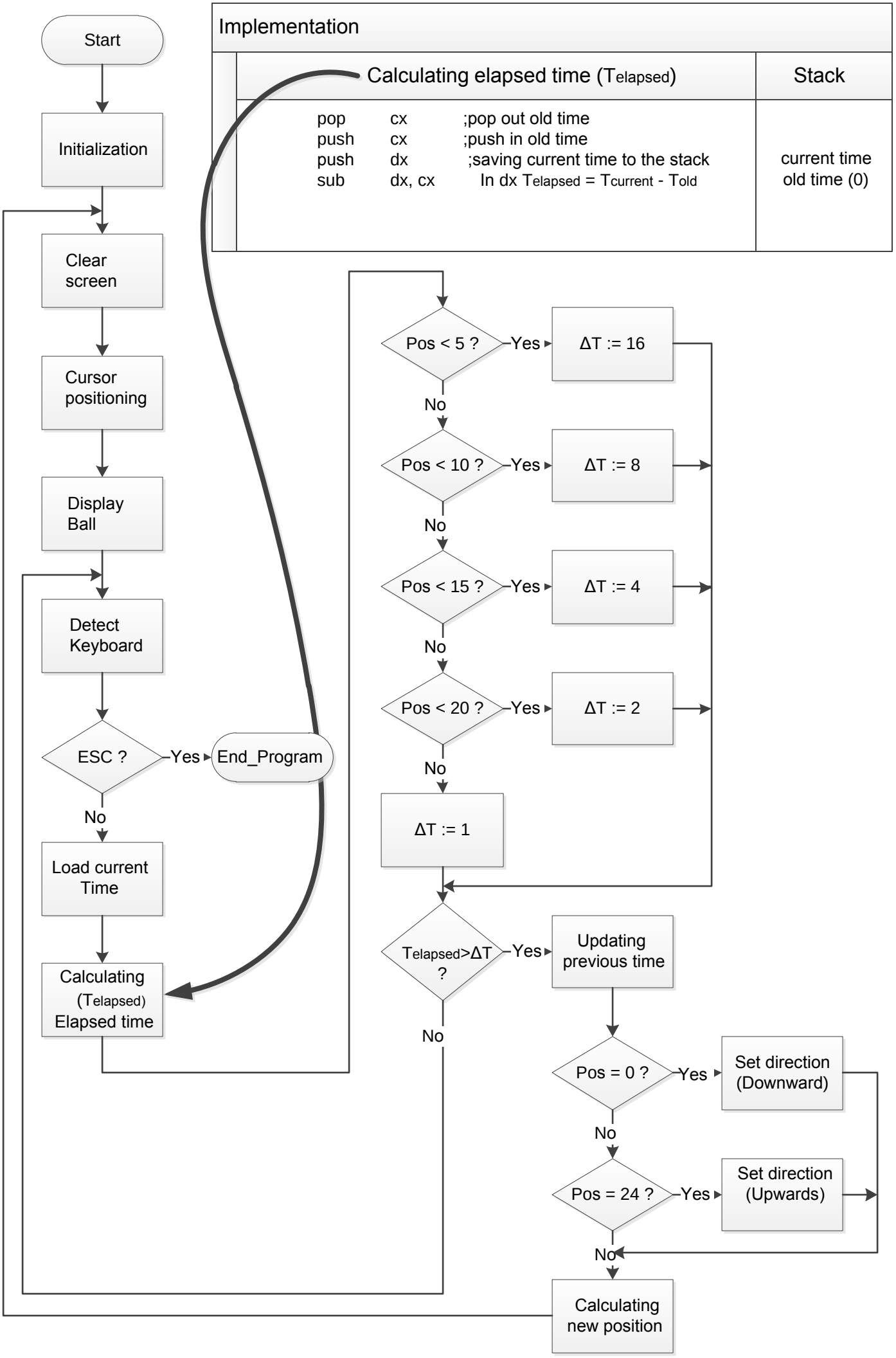

Implementation
Cursor positioning
Stack
mov dx, di ;(using 16 bits register)
mov dh, dl ;position to DI row (8 bits)
mov dl, 40 ;position 40. column
xor bh, bh
mov ah, 02h
int 10h
old time (0)
Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

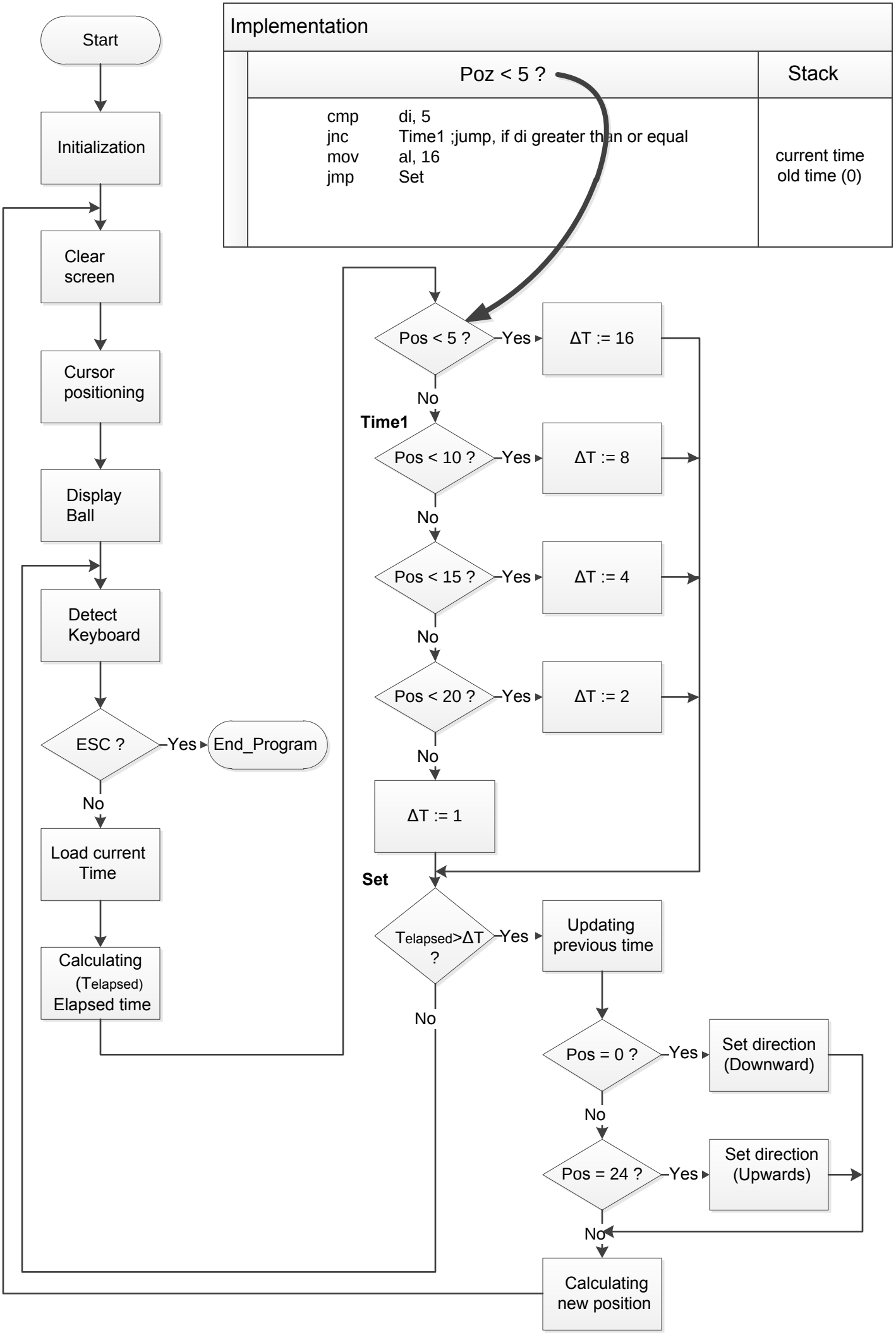

Implementation
Display Ball
Stack
mov dx, offset Ball
mov ah, 09h
int 21h
old time (0)
Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

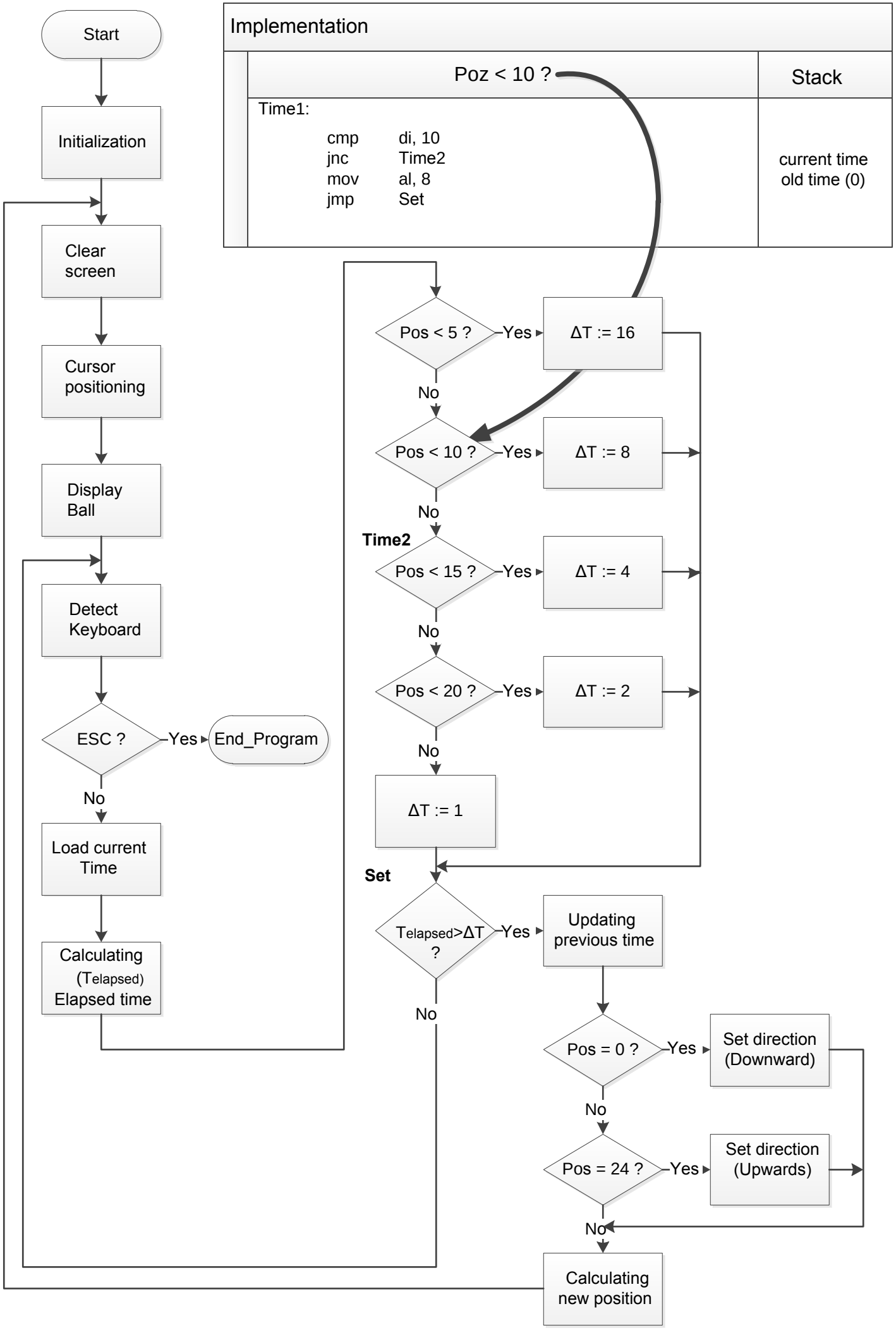

Implementation
Detect Keyboard (1)
Detect Keyboard (2)
Stack
Delay:
mov ah, 00h
int 16h
;without timer
mov ah, 01h
int 16h
;if a key pressed, so the keyboard buffer is not empty, then Z flag = 0
old time (0)
Start
Initialization
Clear screen
Cursor positioning
Display Ball
Delay
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

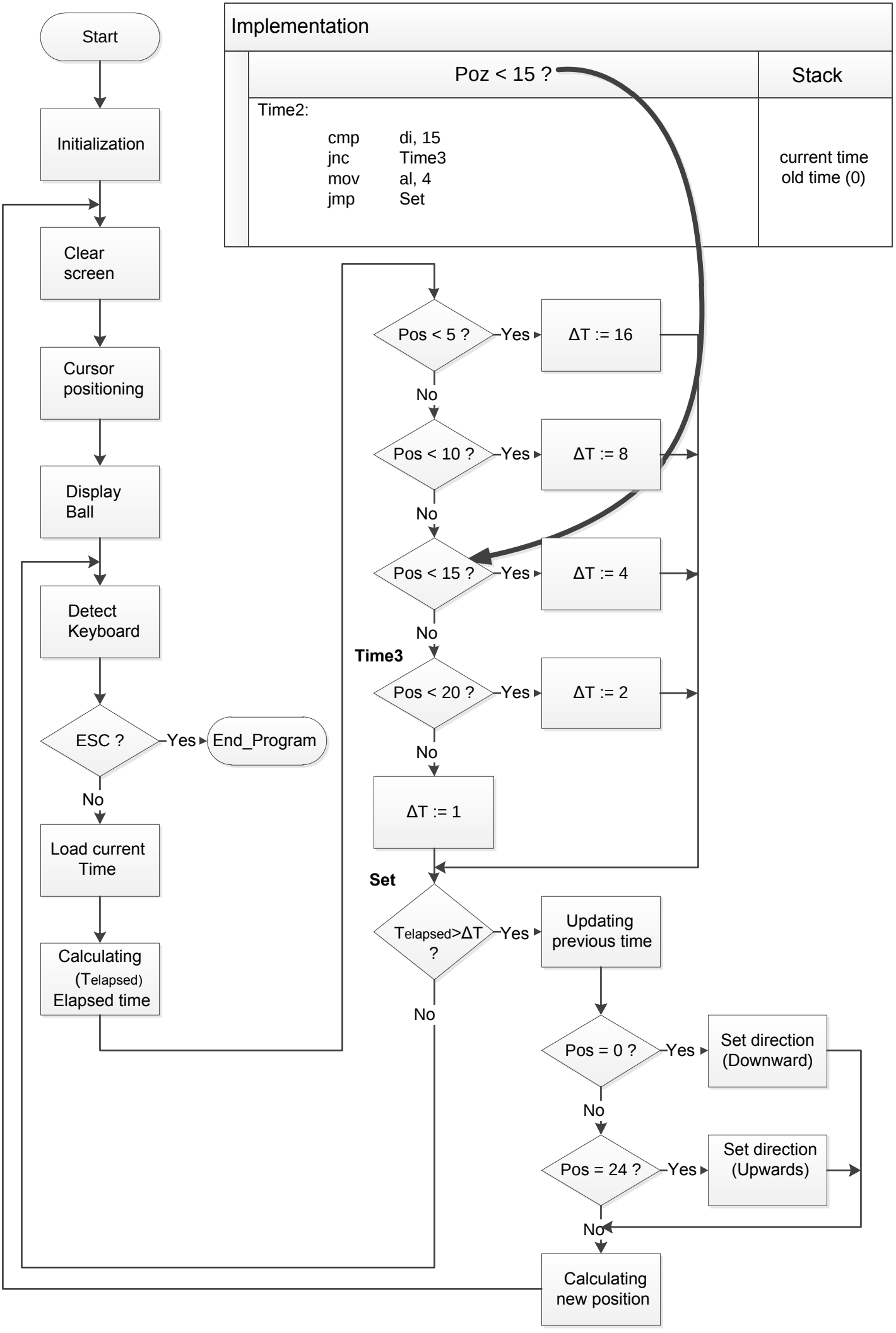

Implementation
Without scheduler (1)
With scheduler (2)
Stack
cmp al, 27
jz End_Program
jz nokey
mov ah, 00h
int 16h
cmp al, 27
jz End_Program
nokey:
old time (0)
How could be write it optimally, that the program finish pressing any key?
Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

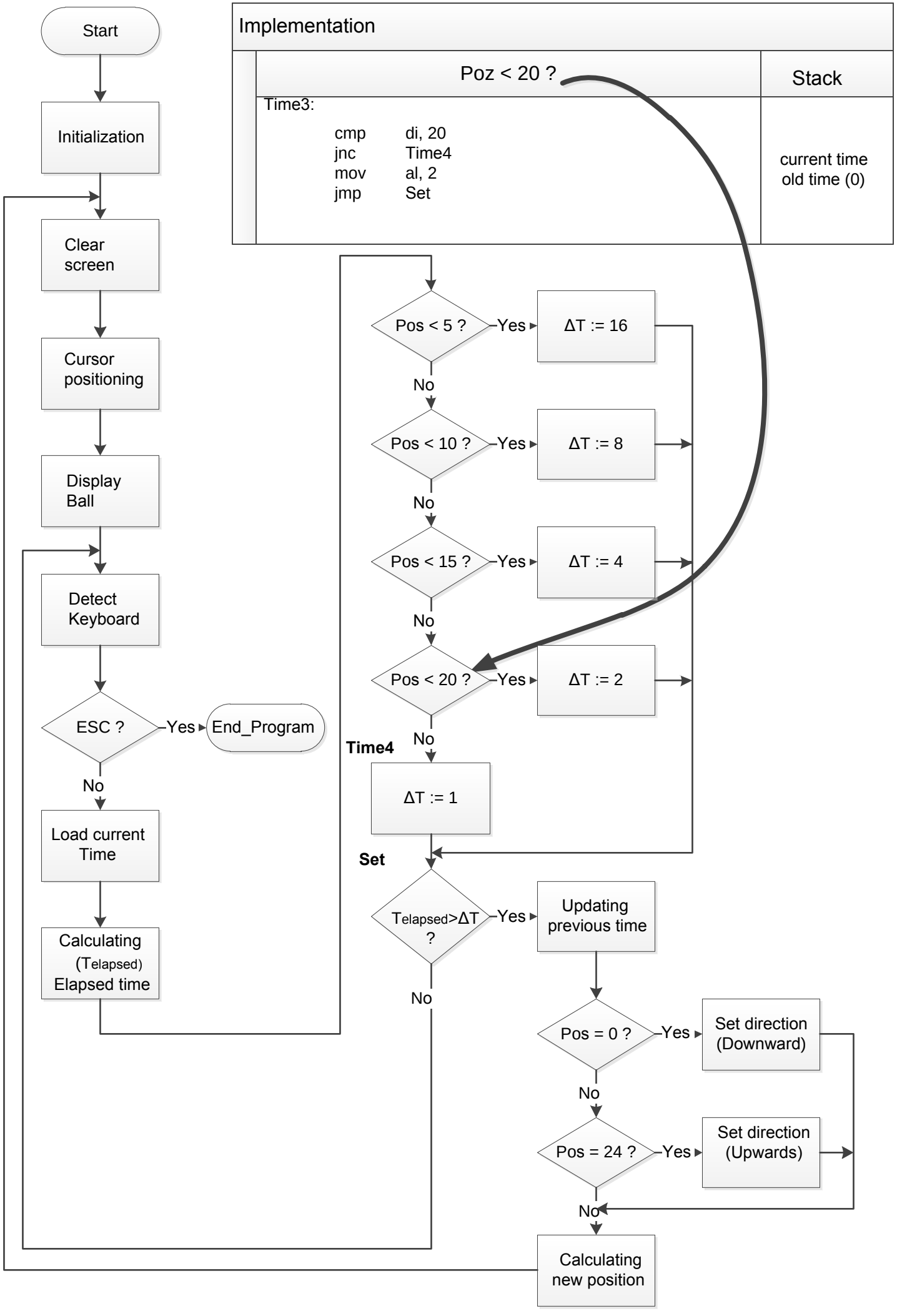

Implementation
Load current time
Stack
xor ah, ah
int 1ah Load current time to CX:DX
old time (0)
Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

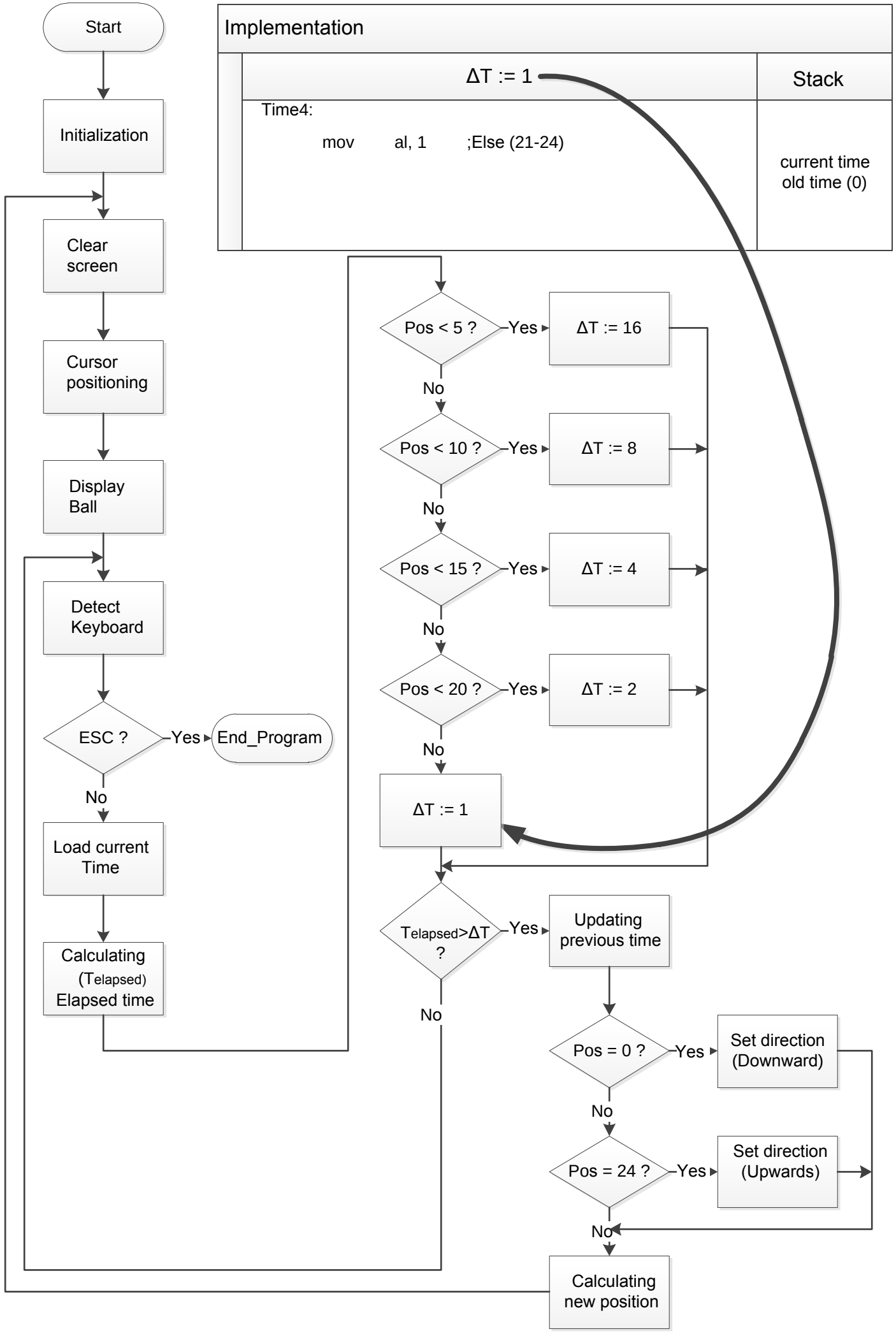

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Calculating elapsed time (Telapsed)
Stack
pop cx ;pop out old time
push cx ;push in old time
push dx ;saving current time to the stack
sub dx, cx In dx Telapsed = Tcurrent - Told
current time
old time (0)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

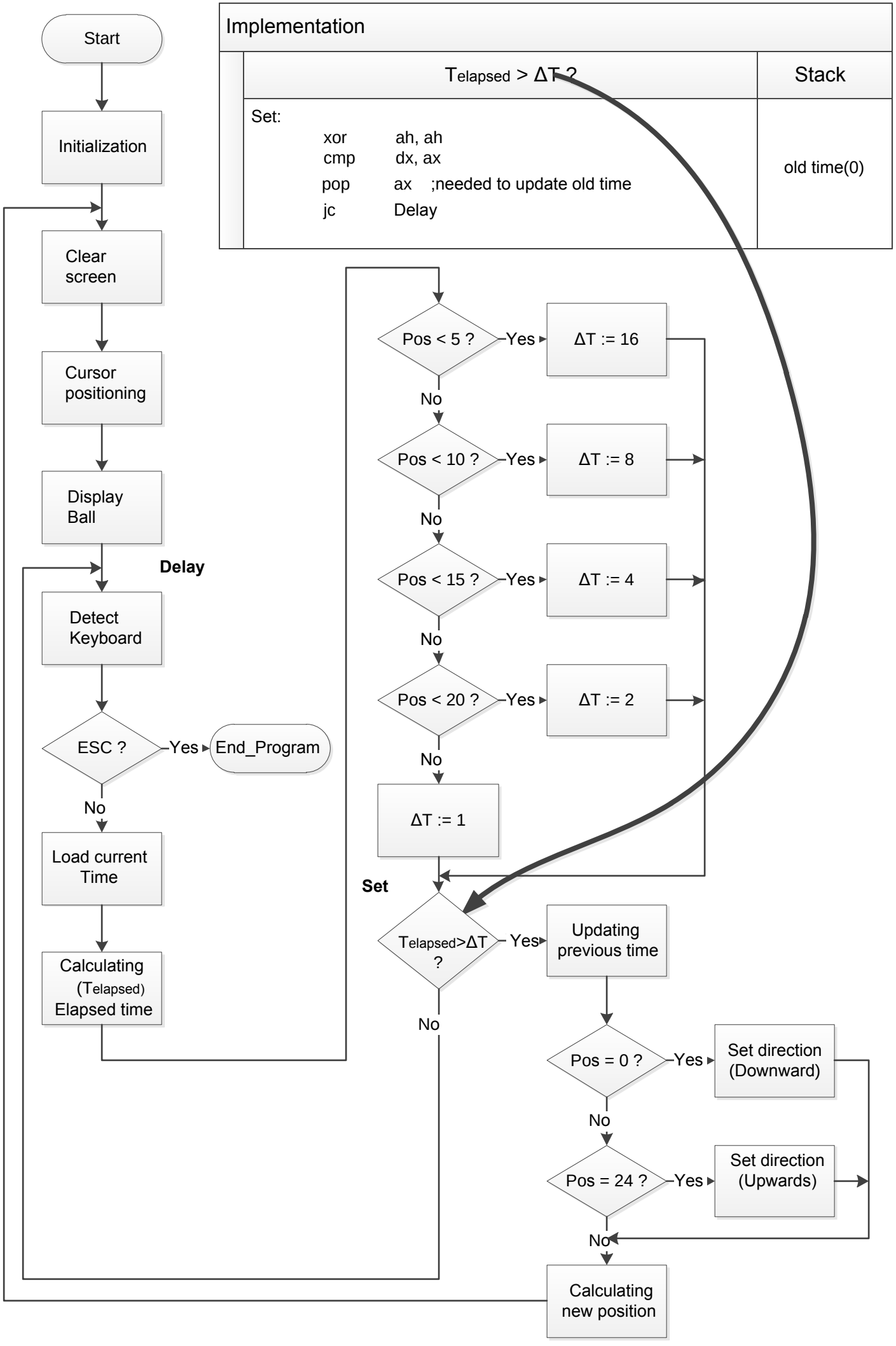

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Poz < 5 ?
Stack
cmp di, 5
jnc Time1 ;jump, if di greater than or equal
mov al, 16
jmp Set
current time
old time (0)
Pos < 5 ?
Yes
ΔT := 16
No
Time1
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Set
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

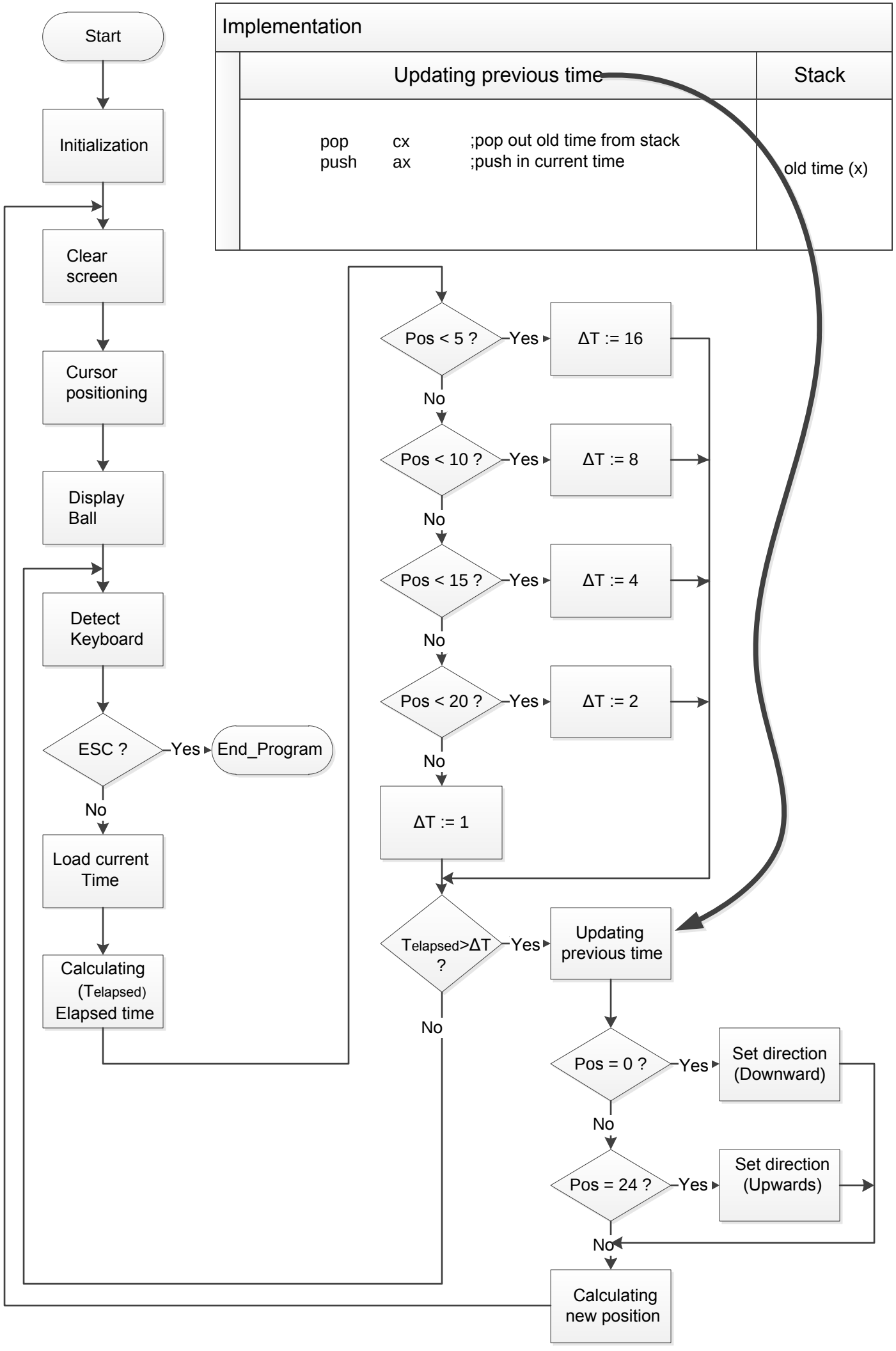

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Poz < 10 ?
Stack
Time1:
cmp di, 10
jnc Time2
mov al, 8
jmp Set
current time
old time (0)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Time2
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Set
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

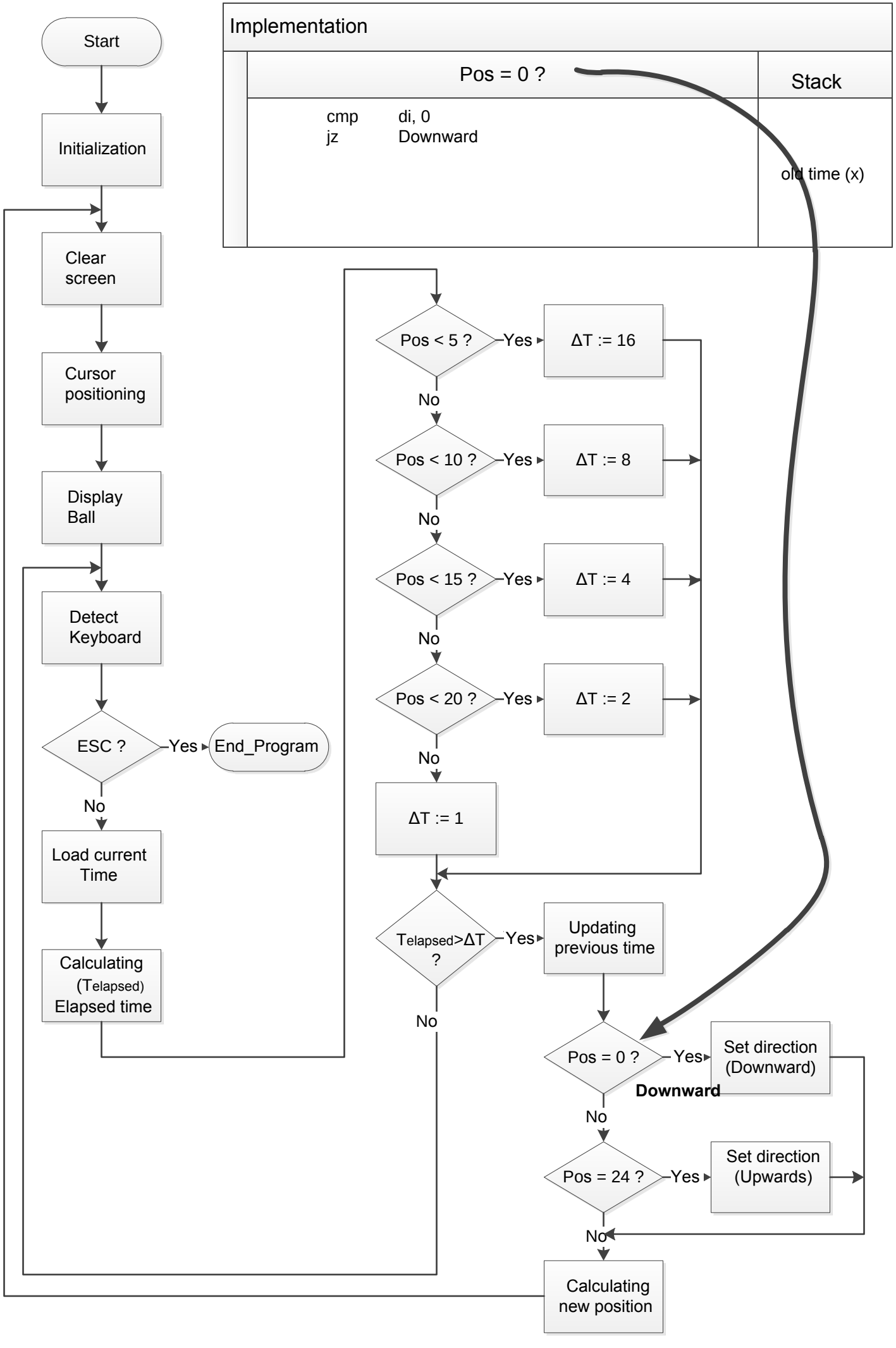

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Poz < 15 ?
Stack
Time2:
cmp di, 15
jnc Time3
mov al, 4
jmp Set
current time
old time (0)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Time3
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Set
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

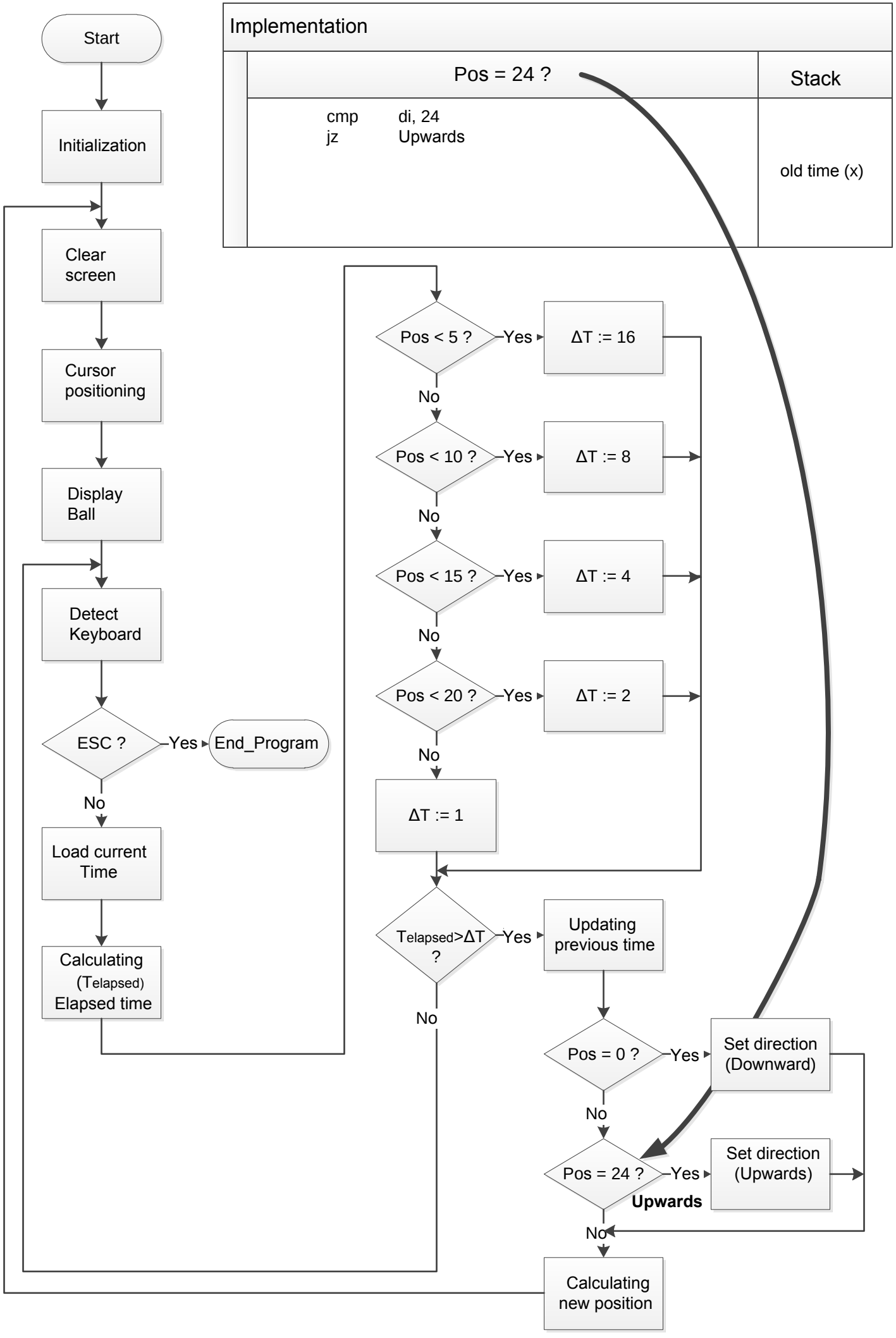

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Poz < 20 ?
Stack
Time3:
cmp di, 20
jnc Time4
mov al, 2
jmp Set
current time
old time (0)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
Time4
ΔT := 1
Set
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

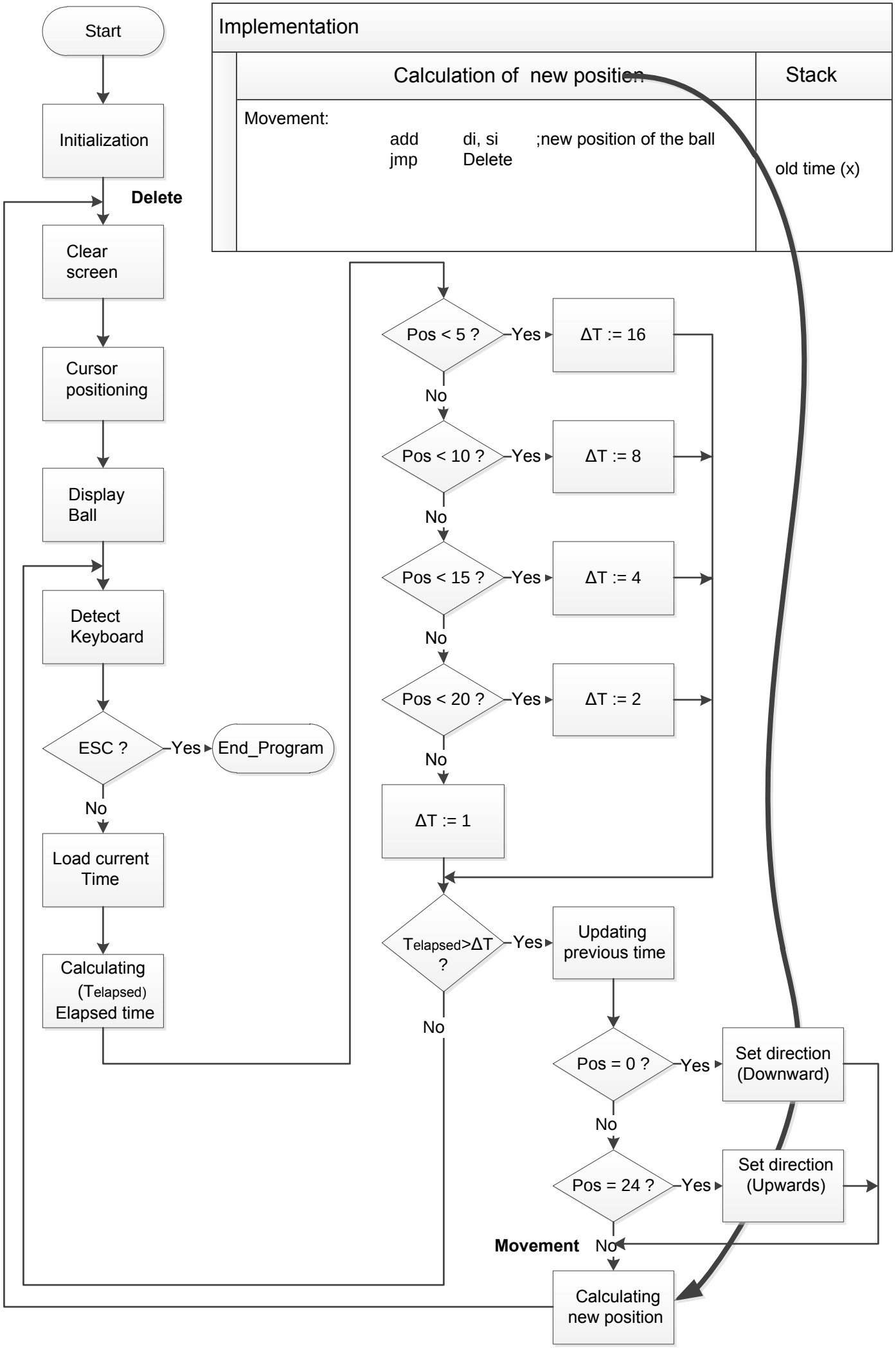

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
ΔT := 1
Stack
Time4:
mov al, 1 ;Else (21-24)
current time
old time (0)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

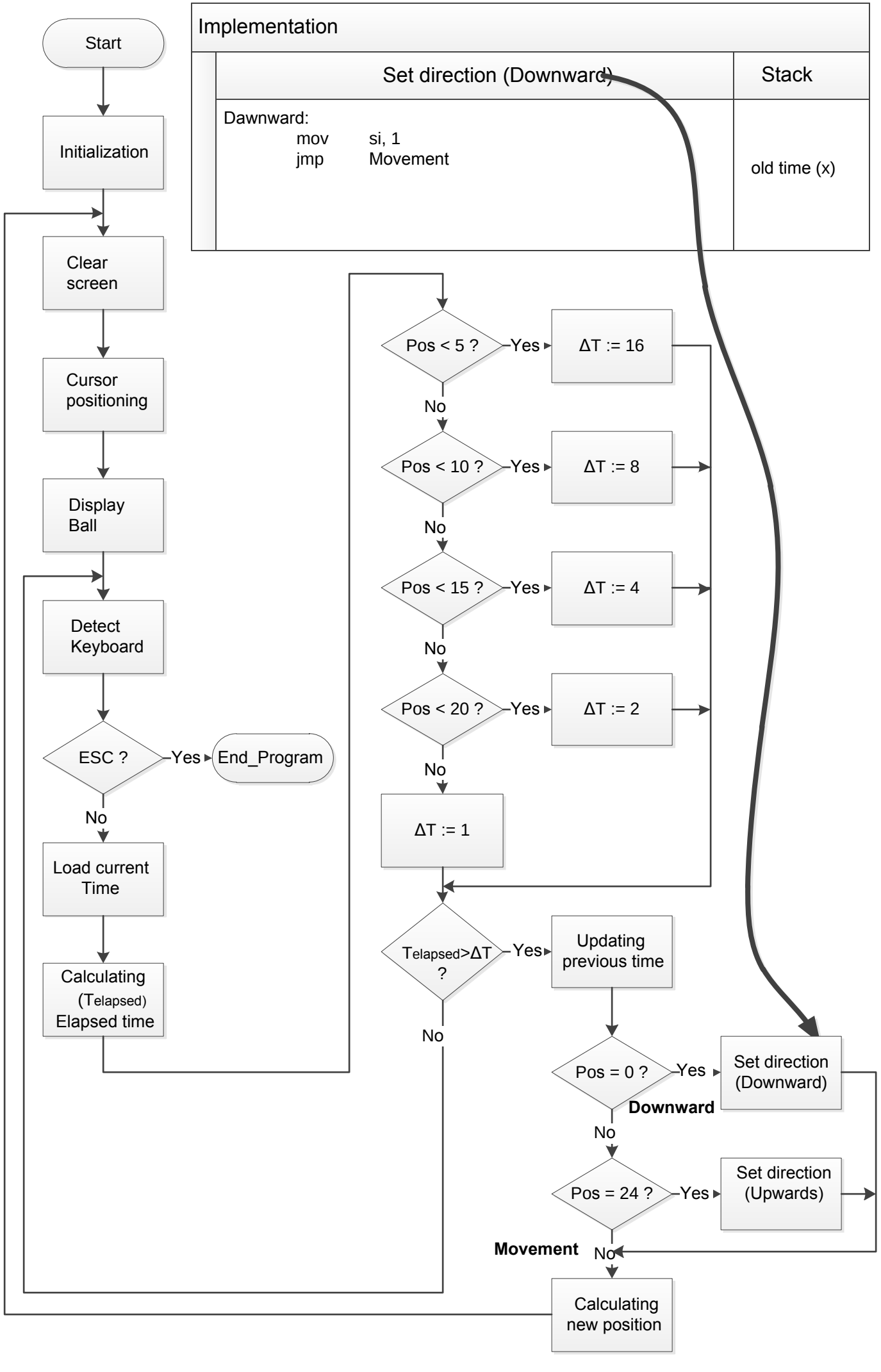

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Delay
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Set
Telapsed>ΔT ?
Yes
No
Updating previous time
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Implementation
Telapsed > ΔT ?
Stack
Set:
xor ah, ah
cmp dx, ax
pop ax ;needed to update old time
jc Delay
old time(0)

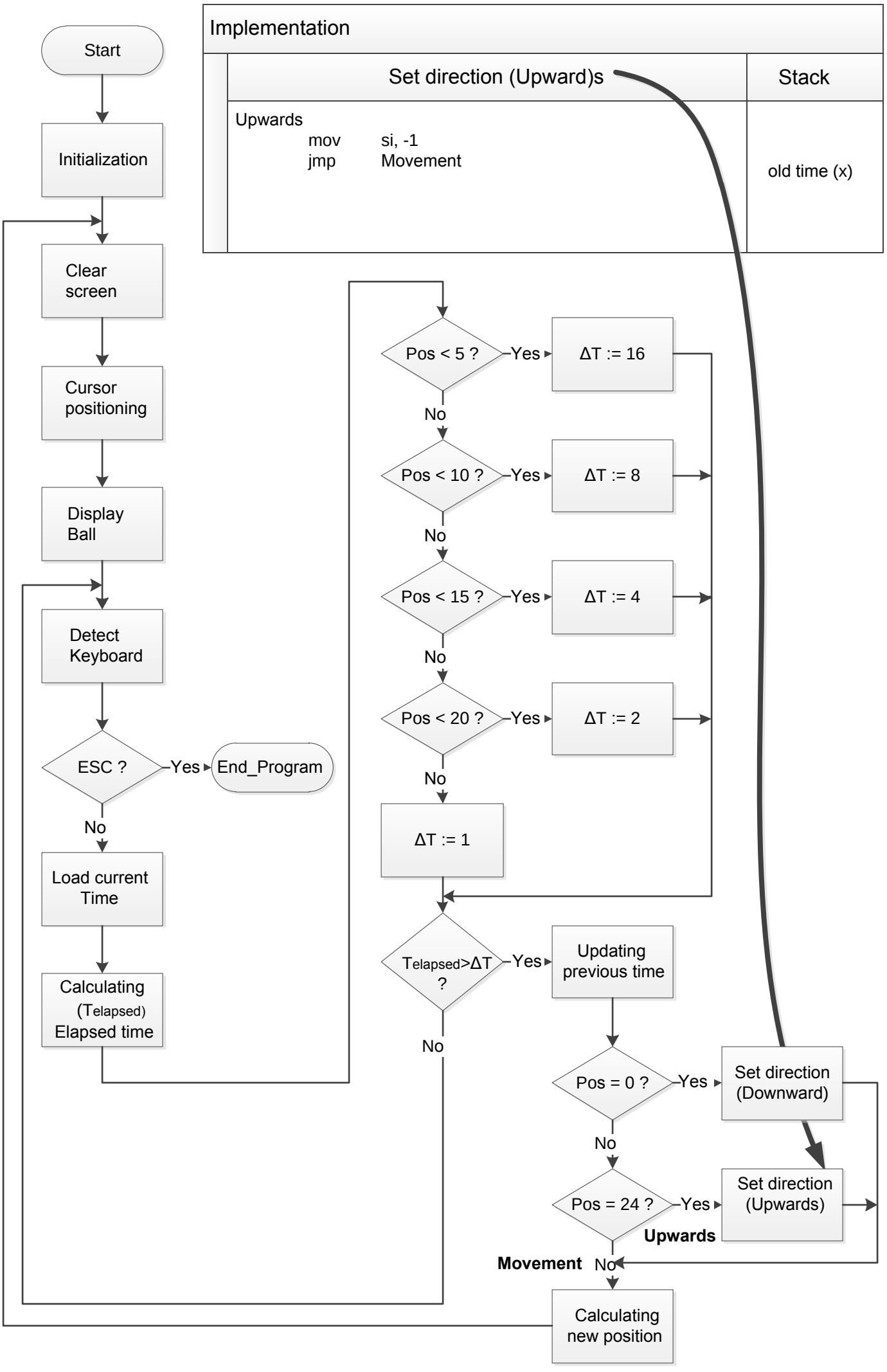

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Updating previous time
Stack
pop cx ;pop out old time from stack
push ax ;push in current time
old time (x)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position

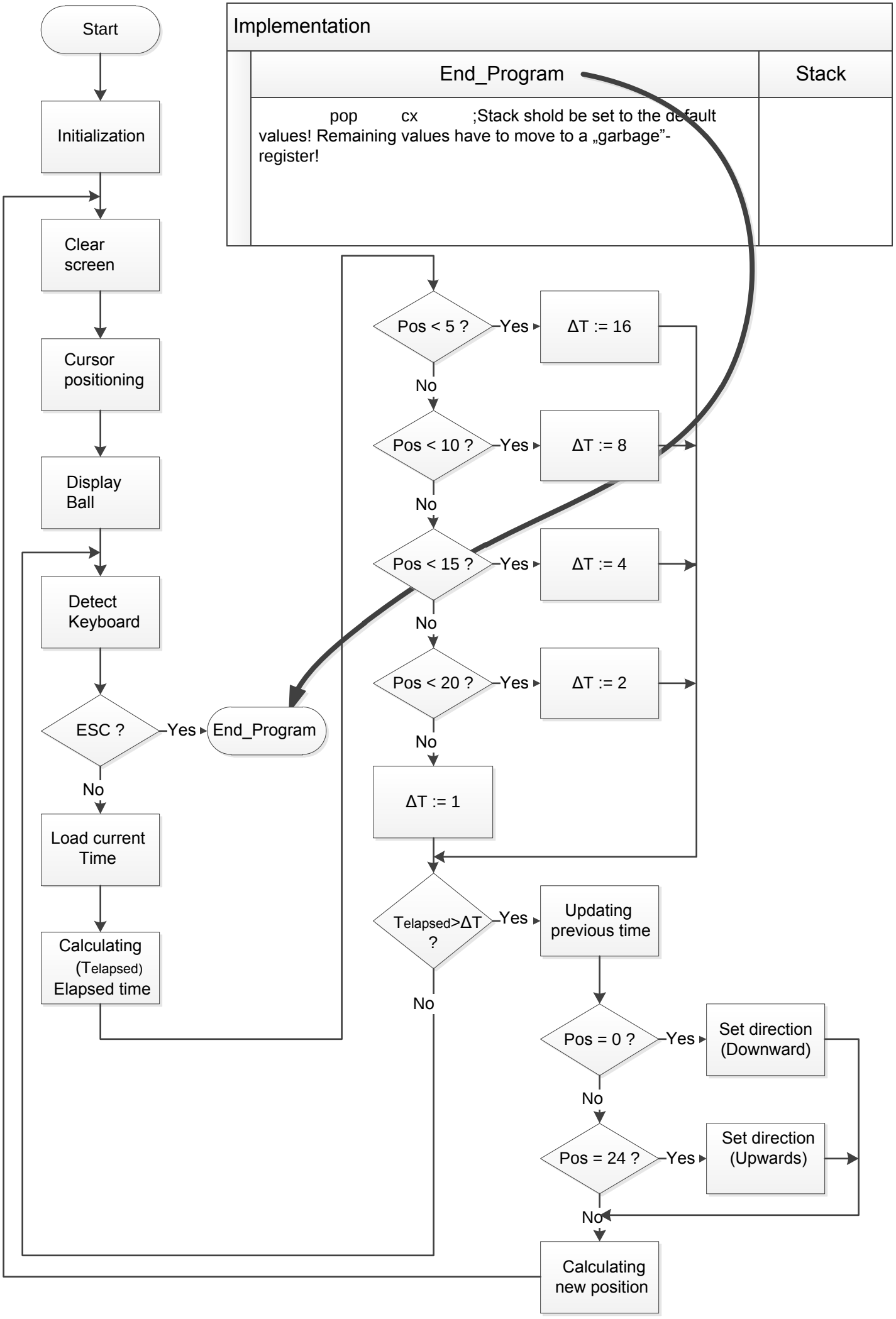

Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Pos = 0 ?
Stack
cmp di, 0
jz Downward
old time (x)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
Downward
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position


Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Pos = 24 ?
Stack
cmp di, 24
jz Upwards
old time (x)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Upwards
Set direction (Upwards)
No
Calculating new position


Start
Initialization
Delete
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Calculation of new position
Stack
Movement:
add di, si ;new position of the ball
jmp Delete
old time (x)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
Movement
No
Calculating new position


Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
Set direction (Downward)
Stack
Dawnward:
mov si, 1
jmp Movement
old time (x)
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
Downward
No
Pos = 24 ?
Yes
Set direction (Upwards)
Movement
No
Calculating new position


Implementation
Set direction (Upward)s
Stack
Upwards
mov si, -1
jmp Movement
old time (x)
Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
Upwards
Movement
No
Calculating new position


Start
Initialization
Clear screen
Cursor positioning
Display Ball
Detect Keyboard
ESC ?
Yes
End_Program
No
Load current Time
Calculating (Telapsed) Elapsed time
Implementation
End_Program
Stack
pop cx ;Stack shold be set to the default values! Remaining values have to move to a „garbage”-register!
Pos < 5 ?
Yes
ΔT := 16
No
Pos < 10 ?
Yes
ΔT := 8
No
Pos < 15 ?
Yes
ΔT := 4
No
Pos < 20 ?
Yes
ΔT := 2
No
ΔT := 1
Telapsed>ΔT ?
Yes
Updating previous time
No
Pos = 0 ?
Yes
Set direction (Downward)
No
Pos = 24 ?
Yes
Set direction (Upwards)
No
Calculating new position