

Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №7

«Защищенный и реальный режим процессора.

Переход из одного режима в другой и обработка прерываний»

Выполнил:

Студент группы 950501

Деркач А.В.

Проверил:

Преподаватель

Одинец Д.Н.

Минск, 2021

1. Постановка задачи

Написать программу, которая выполняет следующие действия:

- Переход из реального режима в защищенный.
- Перехватывает аппаратное прерывание от клавиатуры, в обработчике которого считываются скан-коды клавиш и выводятся на экран. По нажатию клавиши «esc» осуществляется обратный переход в реальный режим.
- Перехватывает аппаратное прерывание от таймера, в обработчике которого отсчитывает секунды и выводит их на экран. По истечении времени, введенного при старте программы осуществляется обратный переход в реальный режим.

2. Алгоритм

- 1) Вводим время нахождения в защищенном режиме в секундах.
- 2) Настраиваем таймер на частоту 20 Гц.
- 3) Загружаем линейные адреса сегментов в дескрипторы.
- 4) Загружаем таблицу глобальных дескрипторов в регистр gdtr.
- 5) Запрещаем прерываний.
- 6) Сохраняем маски прерываний.
- 7) Инициализируем контроллеры.
- 8) Загружаем таблицу дескрипторов исключений в регистр idtr.
- 9) Переходим в защищенный режим.
- 10) Загружаем селекторы в регистры сегментов.
- 11) Разрешаем прерывания.
- 12) Производится обработка прерываний от клавиатуры и таймера.
- 13) Запрещаем прерываний.
- 14) Настраиваем теневые регистры сегментов для работы в реальном режиме.
- 15) Переходим в реальный режим.
- 16) Восстанавливаем значения сегментов.
- 17) Восстанавливаем таблицу векторов прерываний.
- 18) Инициализируем контроллеры.
- 19) Восстанавливаем маски прерываний.
- 20) Завершаем.

3. Листинг программы

Далее приведен листинг программы, реализующей все поставленные задачи.

```
.386P
.MODEL    LARGE

CODE_RM segment para use16
CODE_RM_BEGIN    = $
    assume cs:CODE_RM, ds:DATA, es:DATA
START:
    mov ax, DATA
    mov ds, ax
```

```

    mov es,ax
    lea dx,MSG_ENTER
    mov ah,9h
    int 21h
    call INPUT
    mov ds:[TIME], al
    call FILL_CR_0_BUFFER_RM
    lea dx, BUFFER_CR_0_RM
    mov ah, 9h
    int 21h
    lea dx,MSG_HELLO
    mov ah,9h
    int 21h
    mov ah,7h
    int 21h
PREPARE_RTC:
    mov al,0Bh
    out 70h,al
    in  al,71h
    or  al,00000100b
    out 71h,al
ENABLE_A20:
    in  al,92h
    or  al,2
    out 92h,al
SAVE_MASK:
    in      al,21h
    mov     INT_MASK_M,al
    in      al,0A1h
    mov     INT_MASK_S,al
DISABLE_INTERRUPTS:
    cli
    in  al,70h
    or  al,10000000b
    out 70h,al
    nop
LOAD_GDT:
    mov ax,DATA
    mov dl,ah
    xor dh,dh
    shl ax,4
    shr dx,4
    mov si,ax
    mov di,dx
WRITE_GDT:
    lea bx,GDT_GDT
    mov ax,si
    mov dx,di
    add ax,offset GDT
    adc dx,0
    mov [bx][S_DESC.BASE_L],ax
    mov [bx][S_DESC.BASE_M],dl

```

```

        mov [bx][S_DESC.BASE_H],dh
WRITE_CODE_RM:
        lea bx,GDT_CODE_RM
        mov ax,cs
        xor dh,dh
        mov dl,ah
        shl ax,4
        shr dx,4
        mov [bx][S_DESC.BASE_L],ax
        mov [bx][S_DESC.BASE_M],dl
        mov [bx][S_DESC.BASE_H],dh
WRITE_DATA:
        lea bx,GDT_DATA
        mov ax,si
        mov dx,di
        mov [bx][S_DESC.BASE_L],ax
        mov [bx][S_DESC.BASE_M],dl
        mov [bx][S_DESC.BASE_H],dh
WRITE_STACK:
        lea bx, GDT_STACK
        mov ax,ss
        xor dh,dh
        mov dl,ah
        shl ax,4
        shr dx,4
        mov [bx][S_DESC.BASE_L],ax
        mov [bx][S_DESC.BASE_M],dl
        mov [bx][S_DESC.BASE_H],dh
WRITE_CODE_PM:
        lea bx,GDT_CODE_PM
        mov ax,CODE_PM
        xor dh,dh
        mov dl,ah
        shl ax,4
        shr dx,4
        mov [bx][S_DESC.BASE_L],ax
        mov [bx][S_DESC.BASE_M],dl
        mov [bx][S_DESC.BASE_H],dh
WRITE_IDT:
        lea bx,GDT_IDT
        mov ax,si
        mov dx,di
        add ax,OFFSET IDT
        adc dx,0
        mov [bx][S_DESC.BASE_L],ax
        mov [bx][S_DESC.BASE_M],dl
        mov [bx][S_DESC.BASE_H],dh
        mov IDTR.IDT_L,ax
        mov IDTR.IDT_H,dx
FILL_IDT:
        irpc    N, 0123456789ABCDEF
        lea eax, EXC_0&N

```

```

        mov IDT_0&N.OFFS_L,ax
        shr eax, 16
        mov IDT_0&N.OFFS_H,ax
    endm

    irpc    N, 0123456789ABCDEF
        lea eax, EXC_1&N
        mov IDT_1&N.OFFS_L,ax
        shr eax, 16
        mov IDT_1&N.OFFS_H,ax
    endm

    lea eax, TIMER_HANDLER
    mov IDT_TIMER.OFFS_L,ax
    shr eax, 16
    mov IDT_TIMER.OFFS_H,ax

    lea eax, KEYBOARD_HANDLER
    mov IDT_KEYBOARD.OFFS_L,ax
    shr eax, 16
    mov IDT_KEYBOARD.OFFS_H,ax

    irpc    N, 234567
        lea eax,IDLE_IRQ_MASTER
        mov IDT_2&N.OFFS_L, AX
        shr eax,16
        mov IDT_2&N.OFFS_H, AX
    endm

    irpc    N, 89ABCDEF
        lea eax,IDLE_IRQ_SLAVE
        mov IDT_2&N.OFFS_L,ax
        shr eax,16
        mov IDT_2&N.OFFS_H,ax
    endm
    lgdt fword ptr GDT_GDT
    lidt fword ptr IDTR
    mov eax,cr0
    or  al,00000001b
    mov cr0,eax
OVERLOAD_CS:
    db  0eah
    dw  offset OVERLOAD_SEGMENT_REGISTERS
    dw  CODE_RM_DESC
OVERLOAD_SEGMENT_REGISTERS:
    mov ax,DATA_DESC
    mov ds,ax
    mov es,ax
    mov ax,STACK_DESC
    mov ss,ax
    xor ax,ax
    mov fs,ax
    mov gs,ax

```

```

        lldt ax
PREPARE_TO_RETURN:
        push cs
        push offset BACK_TO_RM
        lea edi,ENTER_PM
        mov eax,CODE_PM_DESC
        push eax
        push edi
REINITIALIAZE_CONTROLLER_FOR_PM:
        mov al,00010001b
        out 20h,al
        out 0A0h,al
        mov al,20h
        out 21h,al
        mov al,28h
        out 0A1h,al
        mov al,04h
        out 21h,al
        mov al,02h
        out 0A1h,al
        mov al,11h
        out 21h,al
        mov al,01h
        out 0A1h,al
        mov al, 0
        out 21h,al
        out 0A1h,al
ENABLE_INTERRUPTS_0:
        in  al,70h
        and  al,01111111b
        out 70h,al
        nop
        sti
GO_TO_CODE_PM:
        db 66h
        retf

BACK_TO_RM:
        cli
        in  al,70h
        or   AL,10000000b
        out 70h,AL
        nop
REINITIALISE_CONTROLLER:
        mov al,00010001b
        out 20h,al
        out 0A0h,al
        mov al,8h
        out 21h,al
        mov al,70h
        out 0A1h,al
        mov al,04h

```

```

    out 21h,al
    mov al,02h
    out 0A1h,al
    mov al,11h
    out 21h,al
    mov al,01h
    out 0A1h,al
PREPARE_SEGMENTS:
    mov GDT_CODE_RM.LIMIT,0FFFFh
    mov GDT_DATA.LIMIT,0FFFFh
    mov GDT_STACK.LIMIT,0FFFFh
    db 0EAH
    dw offset CONTINUE
    dw CODE_RM_DESC
CONTINUE:
    mov ax,DATA_DESC
    mov ds,ax
    mov es,ax
    mov fs,ax
    mov gs,ax
    mov ax,STACK_DESC
    mov ss,ax
ENABLE_REAL_MODE:
    mov eax,cr0
    and al,11111110b
    mov cr0,eax
    db 0EAH
    dw offset CONTINUE2
    dw CODE_RM
CONTINUE2:
    mov ax,STACK_A
    mov ss,ax
    mov ax,DATA
    mov ds,ax
    mov es,ax
    xor ax,ax
    mov fs,ax
    mov gs,ax
    mov IDTR.LIMIT, 3FFH
    mov dword ptr IDTR+2, 0
    lidt fword ptr IDTR
REPAIR_MASK:
    mov al,INT_MASK_M
    out 21h,al
    mov al,INT_MASK_S
    out 0A1h,al
ENABLE_INTERRUPTS:
    in al,70h
    and al,01111111b
    out 70h,al
    nop
    sti

```

```

DISABLE_A20:
    in  al,92h
    and al,11111101b
    out 92h, al
EXIT:
    mov ax,3h
    int 10H
    lea dx,MSG_EXIT
    mov ah,9h
    int 21h

    call FILL_CR_0_BUFFER_RM
    lea dx, BUFFER_CR_0_RM
    mov ah, 9h
    int 21h

    mov ax,4C00h
    int 21H

INPUT proc near
    mov ah,0ah
    xor di,di
    mov dx,offset ds:[INPUT_TIME]
    int 21h
    mov dl,0ah
    mov ah,02
    int 21h

    mov si,offset INPUT_TIME+2
    cmp byte ptr [si],"-"
    jnz ii1
    mov di,1
    inc si
II1:
    xor ax,ax
    mov bx,10
II2:
    mov cl,[si]
    cmp cl,0dh
    jz ii3
    cmp cl,'0'
    jl er
    cmp cl,'9'
    ja er

    sub cl,'0'
    mul bx
    add ax,cx
    inc si
    jmp ii2
ER:

```



```

        mov dx, offset MSG_ERROR
        mov ah,09
        int 21h
        int 20h
II3:
        ret
INPUT endp

FILL_CR_0_BUFFER_RM proc near
        push eax
        push esi
        push dx

        mov eax, cr0
        xor dx, dx
        mov cx, 32
        lea esi, BUFFER_CR_0_RM
        fill_cr_0_loop_rm:
        mov dl, al
        shl dl, 7
        shr dl, 7
        shr eax, 1
        add dl, 48
        mov [esi], dl
        inc esi
        xor dl, dl
        loop fill_cr_0_loop_rm

        pop dx
        pop esi
        pop eax
        ret
FILL_CR_0_BUFFER_RM endp

SIZE_CODE_RM      = ($ - CODE_RM_BEGIN)

CODE_RM ends

CODE_PM segment para use32
CODE_PM_BEGIN     = $
        assume cs:CODE_PM,ds:DATA,es:DATA
ENTER_PM:
call CLRSCR
        xor edi,edi
        lea esi,MSG_HELLO_PM
        call BUFFER_OUTPUT
        add edi,160
        lea esi,MSG_KEYBOARD
        call BUFFER_OUTPUT
        mov edi,320
        lea esi,MSG_TIME

```

```

        call BUFFER_OUTPUT
        mov edi,480
        lea esi,MSG_COUNT
        call BUFFER_OUTPUT

        call FILL_CR_0_BUFFER
        mov edi, 640
        lea esi, BUFFER_CR_0
        call BUFFER_OUTPUT

        mov DS:[COUNT],0
WAITING_ESC:
        jmp  WAITING_ESC
EXIT_PM:
        db 66H
        retf
EXIT_FROM_INTERRUPT:
        popad
        pop es
        pop ds
        pop eax
        pop eax
        pop eax
        sti
        db 66H
        retf

WORD_TO_DEC proc near
        pushad
        movzx eax,ax
        xor cx,cx
        mov bx,10
LOOP1:
        xor dx,dx
        div bx
        add dl,'0'
        push dx
        inc cx
        test ax,ax
        jnz LOOP1
LOOP2:
        pop dx
        mov [di],dl
        inc di
        loop LOOP2
        popad
        ret
WORD_TO_DEC endp

FILL_CR_0_BUFFER proc near

```

```

    push eax
    push esi
    push dx

    mov eax, cr0
    xor dx, dx
    mov cx, 32
    lea esi, BUFFER_CR_0
    fill_cr_0_loop:
    mov dl, al
    shl dl, 7
    shr dl, 7
    shr eax, 1
    add dl, 48
    mov [esi], dl
    inc esi
    xor dl, dl
    loop fill_cr_0_loop

    pop dx
    pop esi
    pop eax
    ret
FILL_CR_0_BUFFER endp

```

```

DIGIT_TO_HEX proc near
    add al, '0'
    cmp al, '9'
    jle DTH_END
    add al, 7
DTH_END:
    ret
DIGIT_TO_HEX endp

```

```

BYTE_TO_HEX proc near
    push ax
    mov ah, al
    shr al, 4
    call DIGIT_TO_HEX
    mov [di], al
    inc di
    mov al, ah
    and al, 0Fh
    call DIGIT_TO_HEX
    mov [di], al
    inc di
    pop ax
    ret
BYTE_TO_HEX endp

```

```

M = 0
IRPC N, 0123456789ABCDEF
EXC_0&N label word
    cli
    jmp EXC_HANDLER
endm
M = 010H
IRPC N, 0123456789ABCDEF
EXC_1&N label word
    cli
    jmp EXC_HANDLER
endm

EXC_HANDLER proc near
    call CLRSCR
    lea esi, MSG_EXC
    mov edi, 40*2
    call BUFFER_OUTPUT
    pop eax
    pop eax
    pop eax
    sti
    db 66H
    retf
EXC_HANDLER      ENDP

IDLE_IRQ_MASTER proc near
    push eax
    mov al,20h
    out 20h,al
    pop eax
    iretd
IDLE_IRQ_MASTER endp

IDLE_IRQ_SLAVE  proc near
    push eax
    mov al,20h
    out 20h,al
    out 0A0h,al
    pop eax
    iretd
IDLE_IRQ_SLAVE  endp

TIMER_HANDLER proc near
    push ds
    push es

```

```

        pushad
        mov  ax,DATA_DESC
        mov  ds,ax
        inc  ds:[COUNT]
        lea  edi,ds:[BUFFER_COUNT]
        mov  ax,ds:[COUNT]
        call WORD_TO_DEC
        mov  edi,538
        lea  esi,BUFFER_COUNT
        call BUFFER_OUTPUT
SHOW_TIMER:
        mov  al,0h
        out  70h,al
        in   al,71h
        cmp  al,ds:[SECOND]
        je   SKIP_SECOND
        mov  ds:[SECOND],al
        mov  al,ds:[TIME]
        cmp  ds:[TIME],0
        je   DISABLE_PM
        xor  ah,ah
        lea  edi,ds:[BUFFER_TIME]
        call WORD_TO_DEC
        mov  edi,356
        lea  esi,BUFFER_TIME
        call BUFFER_OUTPUT
        dec  ds:[TIME]
        lea  esi,BUFFER_TIME
        call BUFFER_CLEAR
        jmp  SKIP_SECOND
DISABLE_PM:
        mov  al,20h
        out  20h,al
        db  0eah
        dd  OFFSET EXIT_FROM_INTERRUPT
        dw  CODE_PM_DESC
SKIP_SECOND:
        mov  al,20h
        out  20h,al
        popad
        pop  es
        pop  ds
        iretd
TIMER_HANDLER endp

KEYBOARD_HANDLER proc near
        push ds
        push es
        pushad
        in   al,60h
        cmp  al,1

```

```

        je     KEYBOARD_EXIT
        mov     ds:[KEY_SCAN_CODE],al
        lea     edi,ds:[BUFFER_SCAN_CODE]
        mov     al,ds:[KEY_SCAN_CODE]
        xor     ah,ah
        call    BYTE_TO_HEX
        mov     edi,200
        lea     esi,BUFFER_SCAN_CODE
        call    BUFFER_OUTPUT
        jmp     KEYBOARD_RETURN
KEYBOARD_EXIT:
        mov     al,20h
        out     20h,al
        db      0eah
        dd      OFFSET EXIT_FROM_INTERRUPT
        dw      CODE_PM_DESC
KEYBOARD_RETURN:
        mov     al,20h
        out     20h,al
        popad
        pop     es
        pop     ds
        iretd
KEYBOARD_HANDLER endp

```

```

CLRSCR  proc near
        push    es
        pushad
        mov     ax,TEXT_DESC
        mov     es,ax
        xor     edi,edi
        mov     ecx,80*25
        mov     ax,700h
        rep     stosw
        popad
        pop     es
        ret
CLRSCR  endp

```

```

BUFFER_CLEAR  proc near
        mov     al,' '
        mov     [esi+0],al
        mov     [esi+1],al
        mov     [esi+2],al
        mov     [esi+3],al
        mov     [esi+4],al
        mov     [esi+5],al
        mov     [esi+6],al
        mov     [esi+7],al
        ret

```

```
BUFFER_CLEAR    endp
```

```
BUFFER_OUTPUT proc near
    push es
    PUSHAD
    mov  ax,TEXT_DESC
    mov  es,ax
OUTPUT_LOOP:
    lodsb
    or   al,al
    jz   OUTPUT_EXIT
    stosb
    inc  edi
    jmp  OUTPUT_LOOP
OUTPUT_EXIT:
    popad
    pop  es
    ret
BUFFER_OUTPUT endp
```

```
SIZE_CODE_PM      =      ($ - CODE_PM_BEGIN)
CODE_PM    ENDS
```

```
DATA    segment para use16
DATA_BEGIN    = $
```

```
    S_DESC  struc
        LIMIT      dw 0
        BASE_L     dw 0
        BASE_M     db 0
        ACCESS     db 0
        ATTRIBS    db 0
        BASE_H     db 0
```

```
    S_DESC  ends
```

```
    I_DESC  struc
        OFFS_L     dw 0
        SEL        dw 0
        PARAM_CNT  db 0
        ACCESS     db 0
        OFFS_H     dw 0
```

```
    I_DESC  ends
```

```
    R_IDTR  struc
        LIMIT      dw 0
        IDT_L      dw 0
        IDT_H      dw 0
```

```
    R_IDTR  ends
```

```
GDT_BEGIN    = $
```

```

GDT_label    word
GDT_0        S_DESC <0,0,0,0,0,0>
GDT_GDT      S_DESC <GDT_SIZE-1,,,10010010b,0,>
GDT_CODE_RM  S_DESC <SIZE_CODE_RM-1,,,10011010b,0,>
GDT_DATA     S_DESC <SIZE_DATA-1,,,11110010b,0,>
GDT_STACK    S_DESC <1000h-1,,,10010010b,0,>
GDT_TEXT     S_DESC <2000h-1,8000h,0Bh,11110010b,0,0>
GDT_CODE_PM  S_DESC <SIZE_CODE_PM-1,,,10011010b,01000000b,>
GDT_IDT      S_DESC <SIZE_IDT-1,,,10010010b,0,>
GDT_SIZE     = ($ - GDT_BEGIN)

CODE_RM_DESC = (GDT_CODE_RM - GDT_0)
DATA_DESC    = (GDT_DATA - GDT_0)
STACK_DESC   = (GDT_STACK - GDT_0)
TEXT_DESC    = (GDT_TEXT - GDT_0)
CODE_PM_DESC = (GDT_CODE_PM - GDT_0)
IDT_DESC     = (GDT_IDT - GDT_0)

;IDT
IDTR    R_IDTR <SIZE_IDT,0,0>
IDT_label word
IDT_BEGIN = $
IRPC     N, 0123456789ABCDEF
        IDT_0&N I_DESC <0, CODE_PM_DESC,0,10001111b,0>
ENDM
IRPC     N, 0123456789ABCDEF
        IDT_1&N I_DESC <0, CODE_PM_DESC, 0, 10001111b, 0>
ENDM
IDT_TIMER I_DESC <0, CODE_PM_DESC,0,10001110b,0>
IDT_KEYBOARD I_DESC <0, CODE_PM_DESC,0,10001110b,0>
IRPC     N, 23456789ABCDEF
        IDT_2&N I_DESC <0, CODE_PM_DESC, 0, 10001110b, 0>
ENDM
SIZE_IDT = ($ - IDT_BEGIN)

MSG_HELLO      db "Press key to change mode to PM",13,10,"$"
MSG_HELLO_PM   db "We are in PM. Press ESC or wait till timer
ends to exit PM",0
MSG_EXIT       db "We are in RM",13,10,"$"
MSG_KEYBOARD   db "Scan code:",0
MSG_TIME       db "Go back to RM in XXXXXXXX seconds",0
MSG_COUNT      db "Amount of interrupt calls:",0
MSG_EXC        db "Exception: XX",0
MSG_ENTER      db "Enter time in protected mode: $"
MSG_ERROR      db "incorrect error$"
HEX_TAB        db "0123456789ABCDEF"
ESP32          dd 1 dup(?)
INT_MASK_M     db 1 dup(?)
INT_MASK_S     db 1 dup(?)
KEY_SCAN_CODE  db 1 dup(?)
SECOND         db 1 dup(?)
TIME           db 1 dup(10)

```



```

COUNT          dw  1 dup(0)
BUFFER_COUNT     db  8 dup(' ')
                 db  1 dup(0)
BUFFER_SCAN_CODE db  8 dup(' ')
                 db  1 dup(0)
BUFFER_TIME      db  8 dup(' ')
                 db  1 dup(0)
INPUT_TIME       db  6,7 dup(?)
BUFFER_CR_0      db 32 dup('?')
                 db  1 dup(0)
BUFFER_CR_0_RM   db 32 dup('?'), 13, 10, "$"

SIZE_DATA = ($ - DATA_BEGIN)
DATA      ends
STACK_A segment para stack
          db 1000h dup(?)
STACK_A   ends
end START

```

4. Тестирование программы

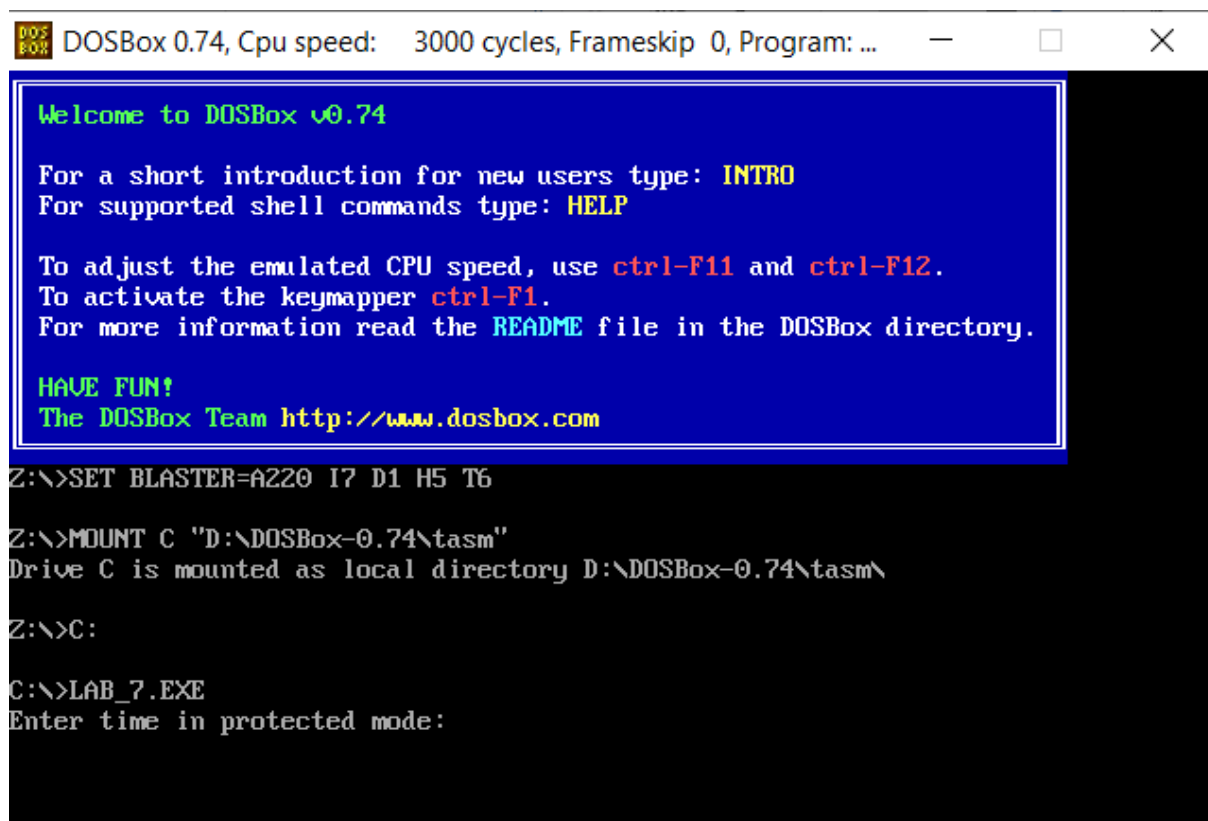


Рисунок 4.1. — Старт программы.

5. Заключение

В данной лабораторной работе были выполнены все поставленные задачи: был выполнен успешный переход в защищенный режим и возврат из него. Были написаны обработчики прерываний клавиатуры и таймера, выполняющие свою работу в защищенном режиме.

Программа запускалась в DOS, который эмулировался с помощью VirtualBox.