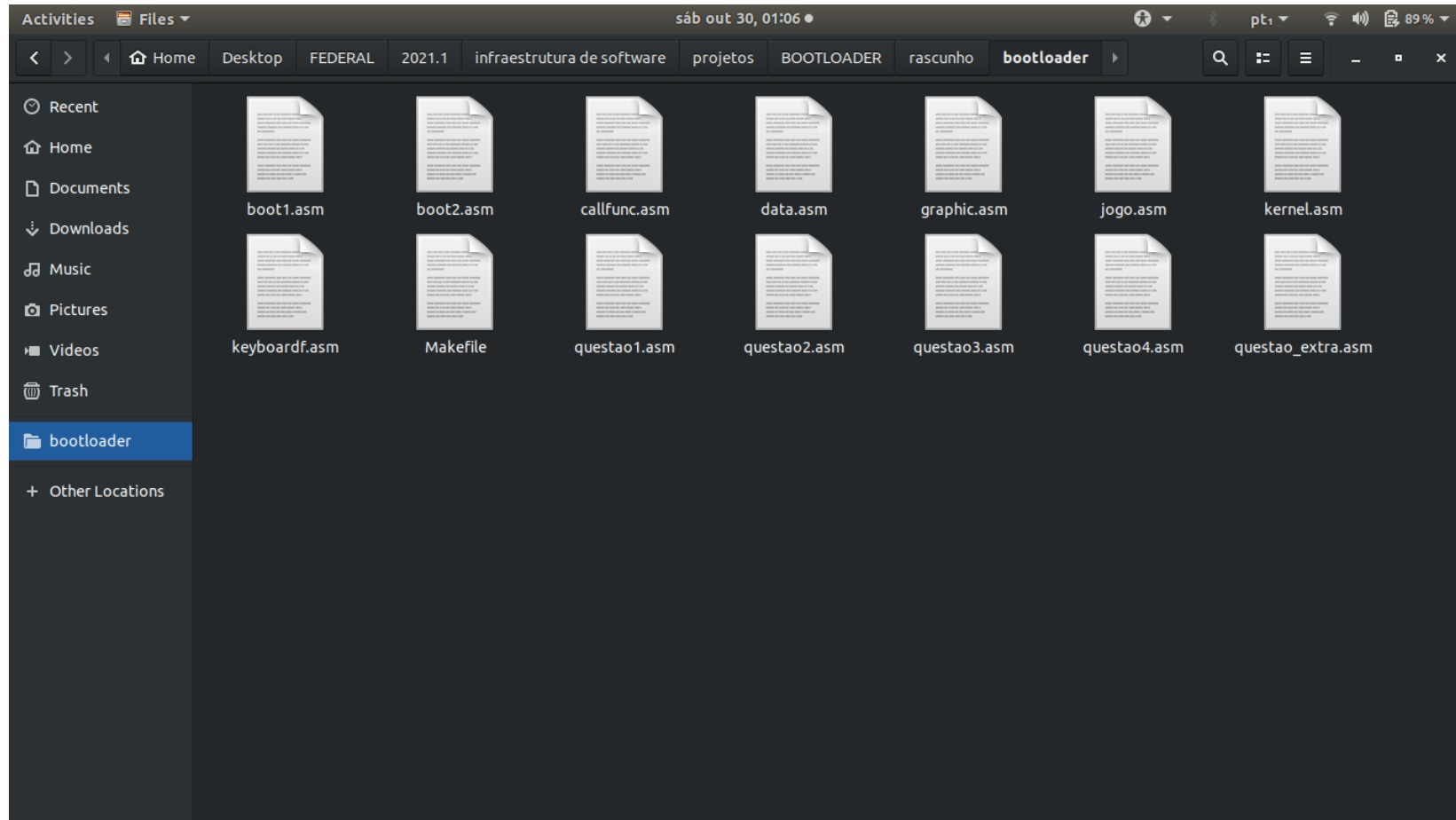


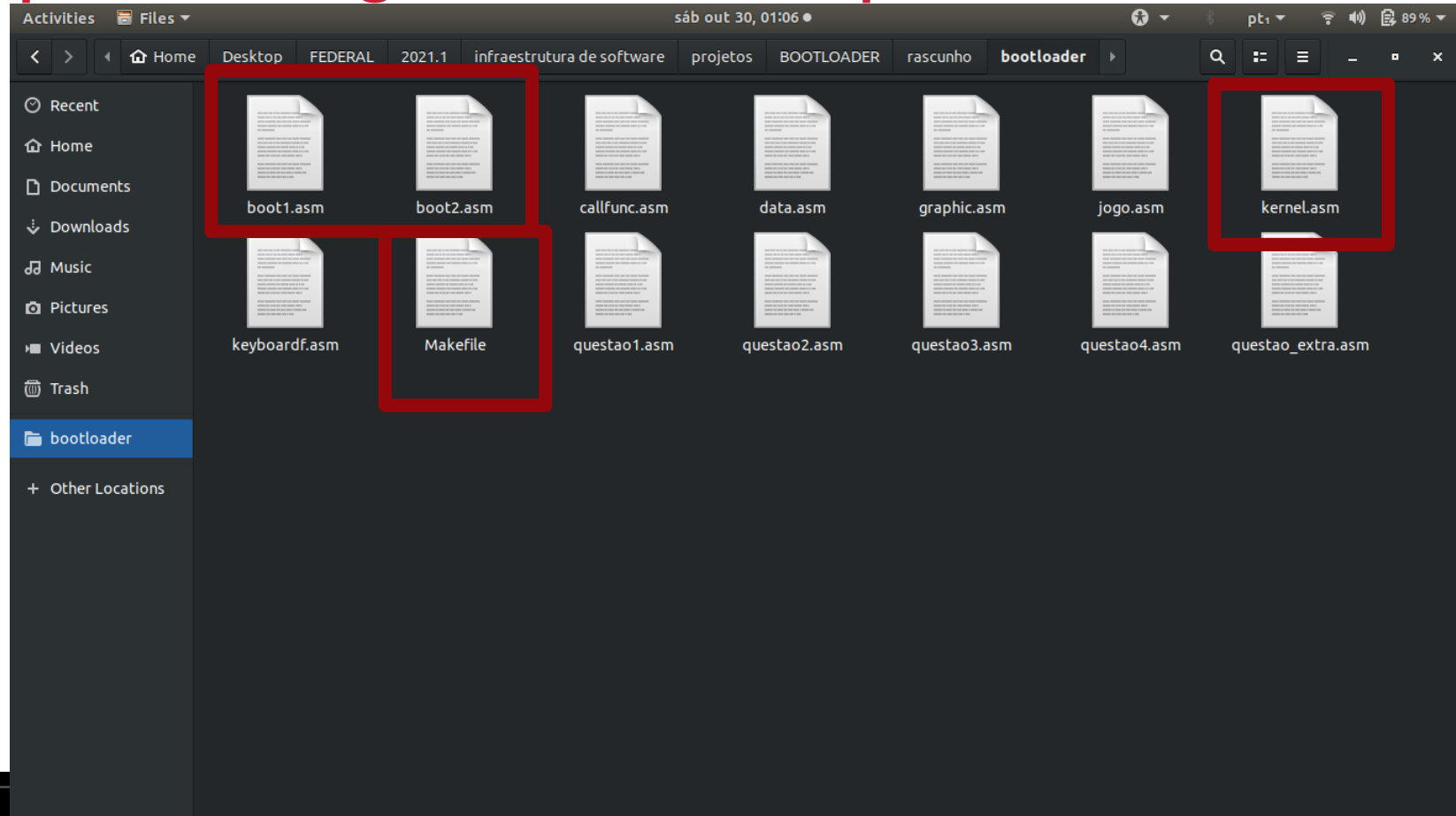
# Guia para o projeto bootloader

Aluno: gabriel ferreira da silva - gfs4

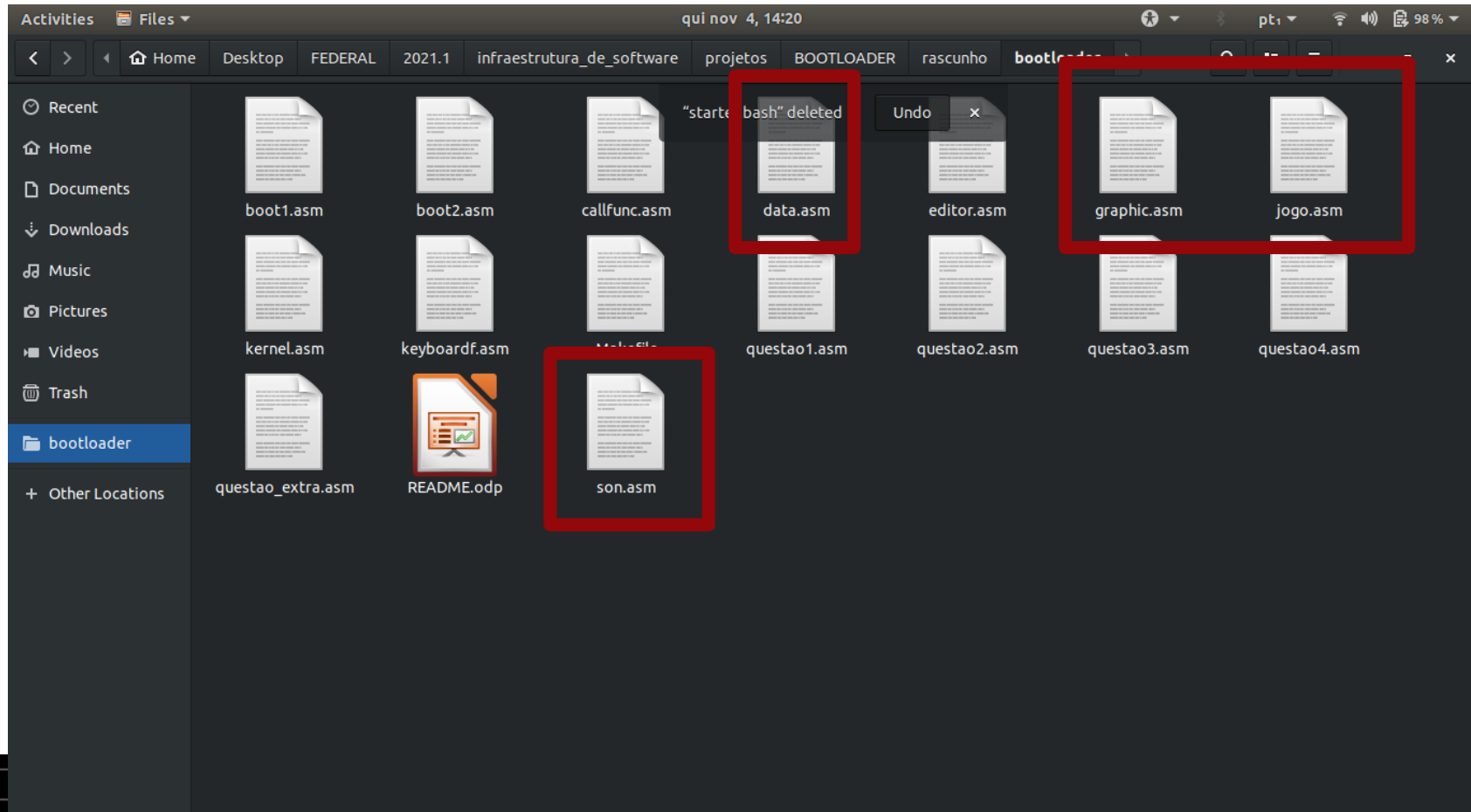
- Esses slides servem para ajudar o professor e os monitores a entenderem o meu bootloader
- O bootloader esta dividido em vários arquivos. Fiz isso para facilitar a organização na hora de programar. Espero que seja permitido.
- os arquivos são os seguintes:



# Arquivos originais do template



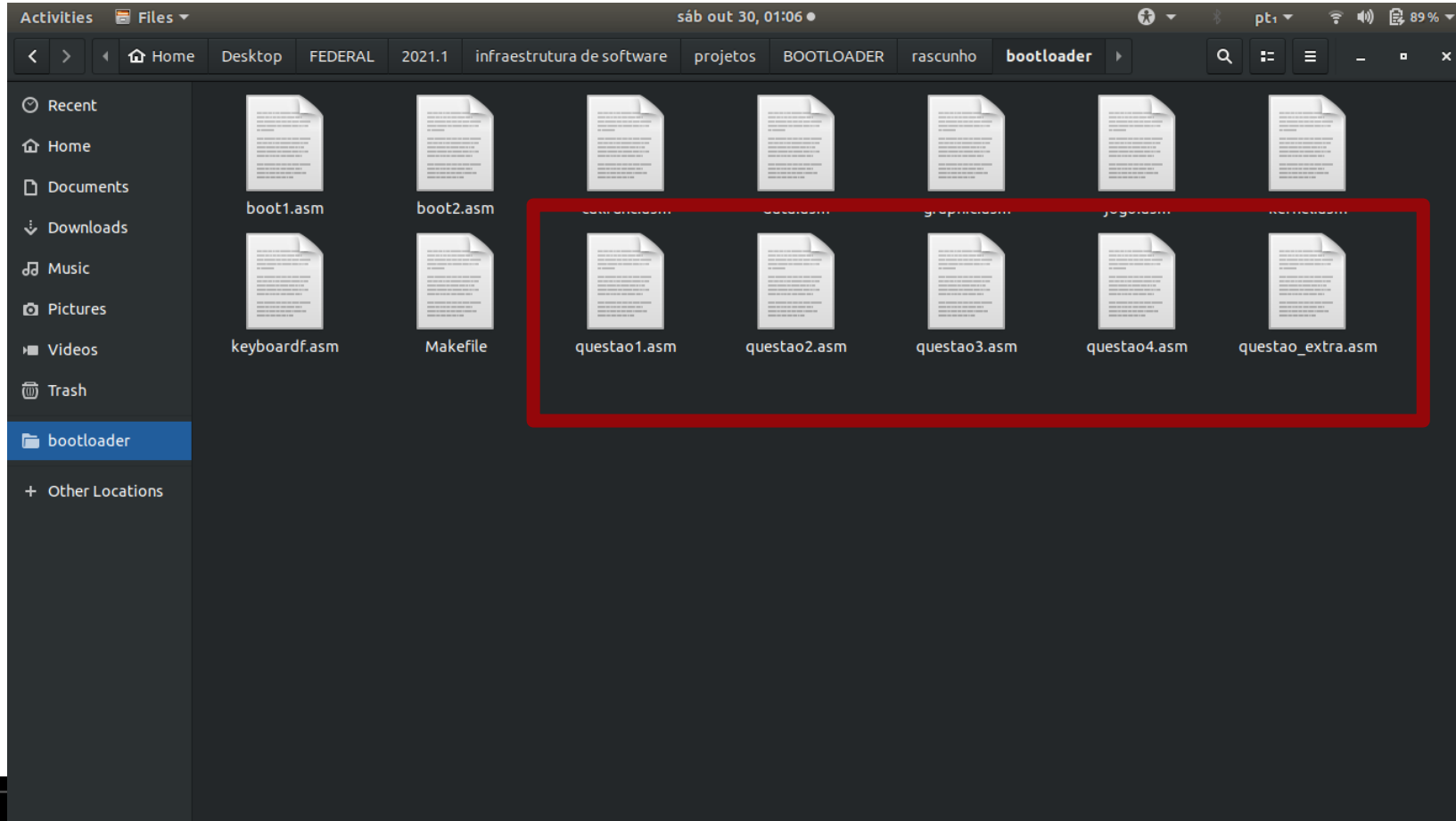
# Programas do jogo



# Programas do jogo

- Data.asm → armazena os vetores posição dos objetos do jogo e as sprites (nave, laser, cometa)
- Graphic.asm → funções responsáveis por desenhar as sprites, e outras como corrigir posições dos objetos, rotina de tempo etc...
- Jogo.asm → o jogo basicamente lê o teclado e atualiza o comportamento dos objetos

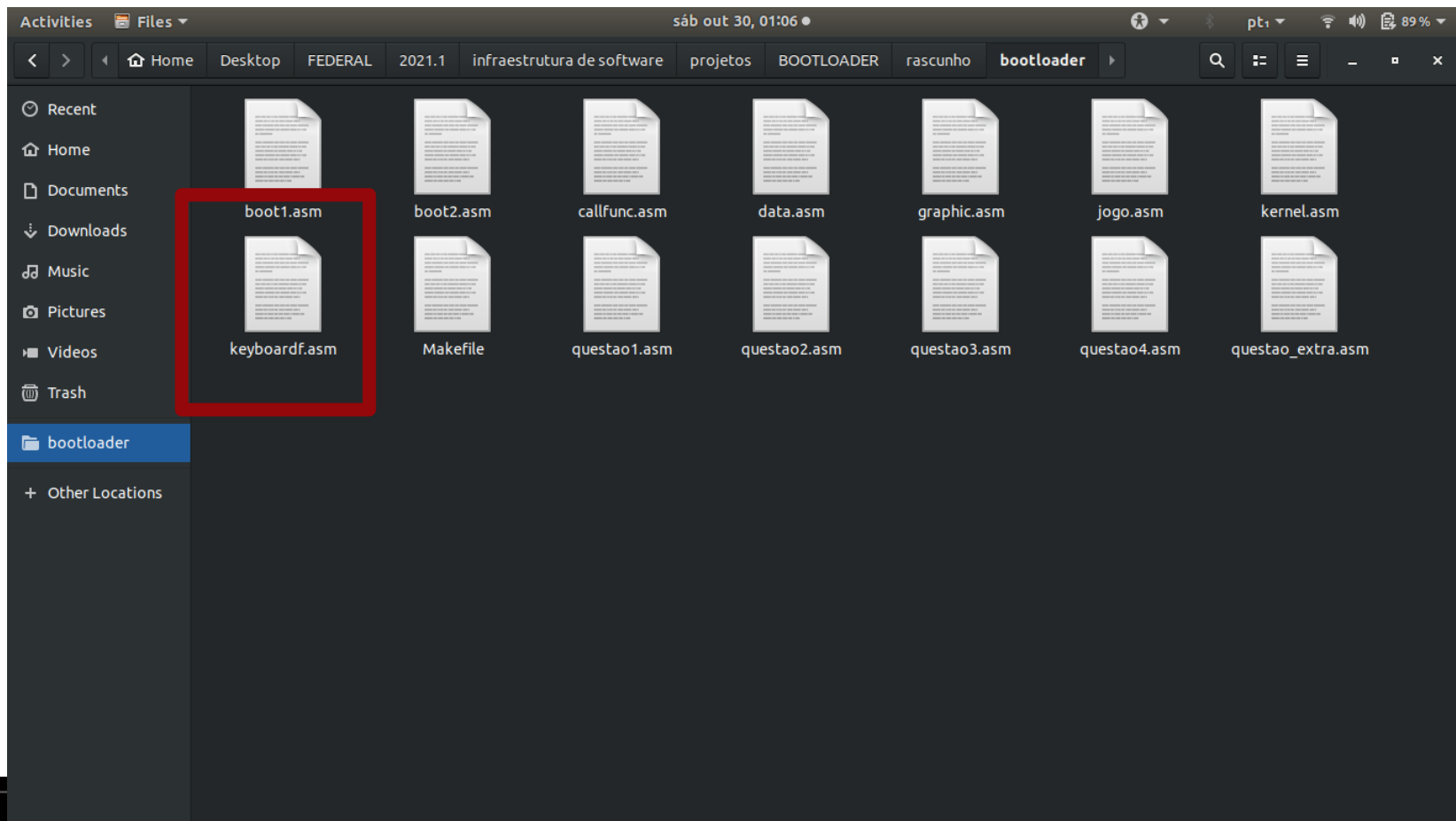
# Programas da lista ASM



- Os programas da lista ASM foram implementados aqui como funções
- Eles podem ser chamados com os comandos:
- Asm1 → chama a questão 1
- Asm2 → chama a questão 2
- Asm3 → chama a questão 3
- Asm4 → chama a questão 4
- Asmx → chama a questão extra
- Para sair desses programas depois que executa-los basta apertar ENTER duas vezes

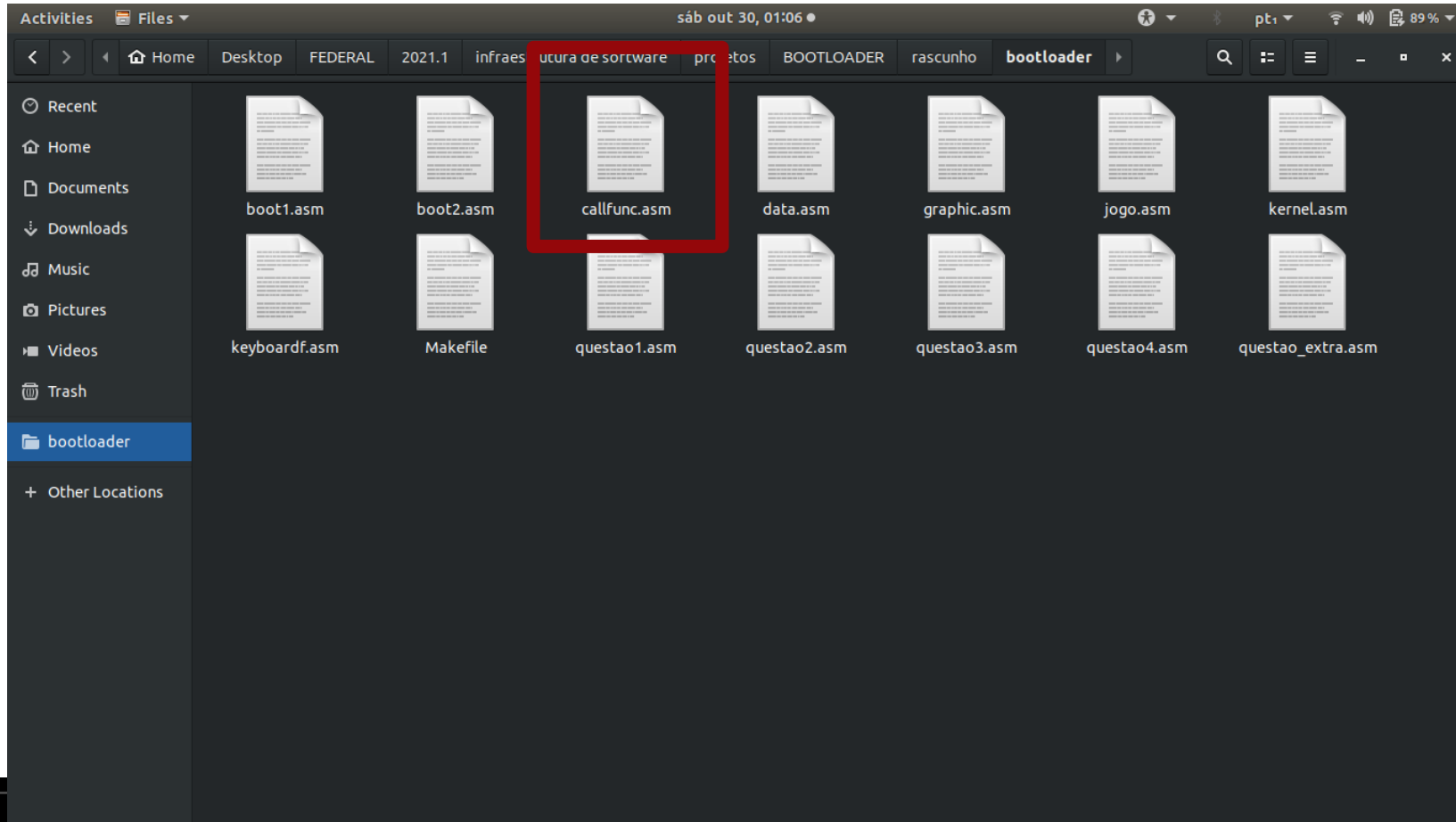


# Funções do teclado



- Aqui coloquei funções relacionadas ao teclado como putchar, getchar
- E também a função \_clear que limpa a tela e \_fim que finaliza as funções e “retorna” para kernel

# Arquivo que chama as funções



esses arquivos que compoẽ o kernel sãõ chamados por ele pela diretiva %include

Activities Text Editor

~/Desktop/FEDERAL/2021.1/infraestrutura de s...

kernel.asm x jogo.asm x graph...

```
1 org 0x7e00
2 jmp 0x0000:_start
3
4 %include "data.asm"
5
6 %include "callfunc.asm" ; responsável por testar as entradas da linha
7 %include "keyboardf.asm" ; função de leitura e entrada do teclado
8
9 %include "questao1.asm" ; programas das questões da lista ASM
10 %include "questao2.asm"
11 %include "questao3.asm"
12 %include "questao4.asm"
13 %include "questao_extra.asm"
14
15 %include "editor.asm" ; programa do editor de texto
16
17 %include "graphic.asm" ; função de gráfico para o jogo
18 %include "jogo.asm" ; programa do jogos
19
20
21
22 data:
23     line times 100 db 0
24
25     texto times 512 db 0
26     texto2 times 512 db 0
27
28
29 _start: ; limpa a tela e seta o vi-deo mode
30     xor ax, ax
31     xor bx, bx
32     xor cx, cx
33     xor dx, dx
34     mov bx, 15
35     mov ds, ax
```

O kernel

- O **kernel.asm** faz três coisas
- 1 – cria um ponteiro line
- 2 – a função `_wait` lê o keyboard e armazena em “line”
- 3 – a o pressionar ENTER a função `_funcs` é chamada. Nela, line é comparada com o nome de outras funções

```
50
51     mov al, '#'
52     call _putchar
53     mov al, ' '
54     call _putchar
55
56     mov di, line
57
58 _wait:                                ; nesse loop o lê teclado
59     call _getchar
60     call _putchar
61     cmp al, 13
62     je _funcs                        ; le o teclado ao ser pressionado
63     stosb
64     jmp _wait
65
66 _funcs:                              ; aqui as funções são testadas
67     mov al, 0
68     stosb
69
70     call _cmp_asm1 ; essas cinco funções são os pr
71     call _cmp_asm2
72     call _cmp_asm3
73     call _cmp_asm4
74     call _cmp_asmx
75
76     call _cmp_editor ; programa do editor
77
78     call _cmp_jogo ; programa do jogo
79
80     call _quebralinha
81
82     jmp _set
83
```

- Essas funções estão em **callfunc.asm**. Elas comparam line com um ponteiro que armazena o nome de uma função específica. Se line for igual a alguma delas essa função é chamada, se não ela retorna para o `_wait`, do kernel

```
questao1 db "asm1", 0  
questao2 db "asm2", 0  
questao3 db "asm3", 0  
questao4 db "asm4", 0  
questaox db "asmx", 0
```

```
editor db "editor", 0
```

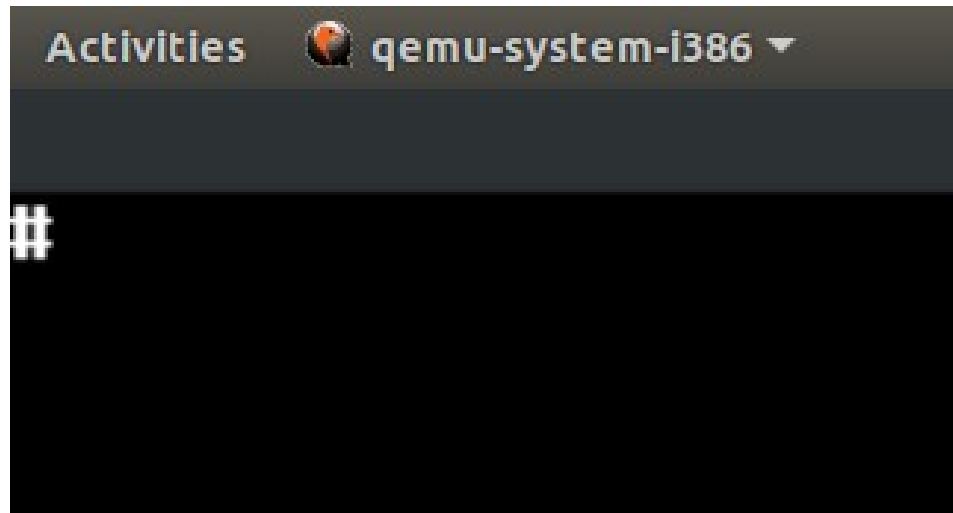
```
jogo db "jogo", 0
```

```
_cmp_asm1:  
    mov si, line  
    mov di, questao1  
    call _strcmp  
    jne .fim  
    call _questao1  
    .fim  
    ret
```

```
_cmp_asm2:  
    mov si, line  
    mov di, questao2  
    call _strcmp  
    jne .fim  
    call _questao2  
    .fim  
    ret
```

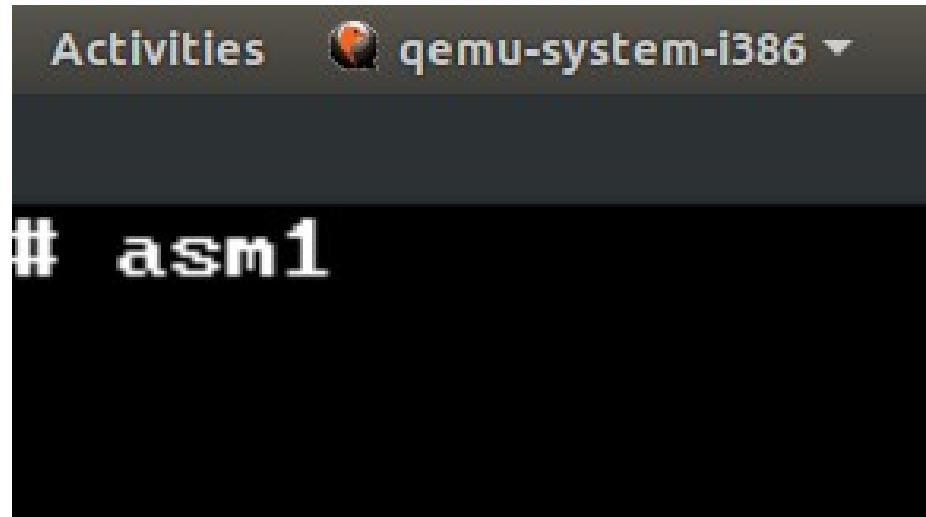
```
_cmp_asm3:  
    mov si, line  
    mov di, questao3  
    call _strcmp  
    jne .fim
```

# O bootloader é carregado

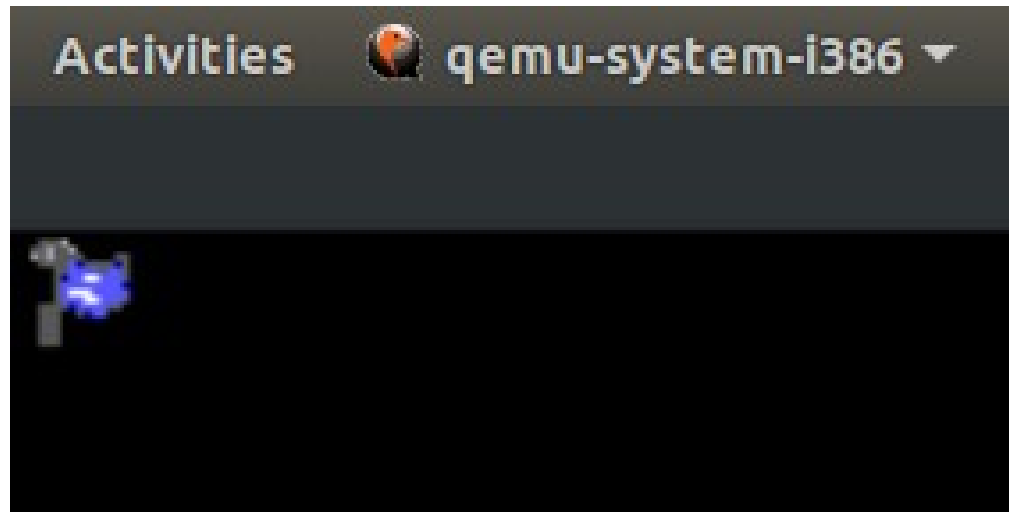




O nome da função é digitado e pressiona  
ENTER

A screenshot of a terminal window. The title bar at the top shows 'Activities' on the left, a circular icon in the center, and 'qemu-system-i386' on the right with a dropdown arrow. The terminal area has a dark background and displays the text '# asm1' in a white, monospaced font.

# Função é chamada



- Nome das funções no kernel:
- # asm1
- # asm2
- # asm3
- # asm4
- # asmX
- # jogo

# O JOGO

## O jogo

- Três arquivos compoẽ o jogo:
- jogo.asm, data.asm, graphic.asm

# O jogo - jogo.asm

- No inicio são definidos as teclas de controle e algumas propriedades dos objetos como velocidade, posição inicial etc



```
Activities Text Editor ~/Desktop/FEDE
jogo.asm
%define VEL_SHOOT 8 ; algumas variaveis do jogo
%define VEL_NAVE 7

%define DOWN_KEY 's'
%define UP_KEY 'w'
%define RIGHT_KEY 'd'
%define LEFT_KEY 'a'

%define CX_INICIAL 20
%define DX_INICIAL 50

%define SHOOT_KEY 'k'

; tamanho da tela 100 x 100

_jogo:

    call _clear

    mov si, vida
    call _print_line

    mov si, texto_apoio
    call _print_line

    mov ah, 00h
    mov al, 08h
    int 10h

    call _print_barra_vida

    mov di, vida_dx
```

# O jogo - jogo.asm

- A função `_jogo` faz a inicialização do jogo.
- No bloco ao lado ele limpa a tela, coloca os textos iniciais, coloca a resolução 260x100 na tela e printa a barra de vida



```
Activities Text Editor
~/Desktop/FEDE

jogo.asm x

%define VEL_SHOOT 8      ; algumas variaveis do jogo
%define VEL_NAVE 7

%define DOWN_KEY 's'
%define UP_KEY 'w'
%define RIGHT_KEY 'd'
%define LEFT_KEY 'a'

%define CX_INICIAL 20
%define DX_INICIAL 50

%define SHOOT_KEY 'k'

; tamanho da tela 100 x 100

_jogo:

    call _clear

    mov si, vida
    call _print_line

    mov si, texto_apoio
    call _print_line

    mov ah, 00h
    mov al, 08h
    int 10h

    call _print_barra_vida

    mov di, vida_dx
```

# O jogo - jogo.asm

- Carrega os objetos com suas posições.
- Os ponteiros com suas posições( **shoot\_life**, **shoot\_pos**, **nave\_pos** e etc..) são carregados
- A função `_print_sprite` é responsável por printar um vetor imagem armazenado em SI

```
Activities Text Editor
Open ~/Desl
jogo.asm x
mov di, vida_dx
mov ax, 150
stosw

xor ax, ax
mov ds, ax
mov es, ax
mov dx, DX_INICIAL ;linha
mov cx, CX_INICIAL ;coluna
mov si, 0
mov di, 0

mov di, life
mov ax, 3
stosw

xor ax, ax
mov di, ax

mov di, shoot_life
stosw
mov di, nave_pos
mov ax, dx
stosw
mov ax, cx
stosw
mov di, shoot_pos
mov ax, dx
stosw
mov ax, cx
stosw

mov dx, 80
mov cx, 80

mov di, cometa_pos
mov ax, dx
stosw
mov ax, cx
stosw

mov si, cometa
call _print_sprite

mov ax, 0h

mov dx, DX_INICIAL
mov cx, CX_INICIAL

mov si, nave
call _print_sprite

mov dx, 80
```

```
Activities Text Editor
Open ~/Desl
jogo.asm x
mov di, shoot_life
stosw
mov di, nave_pos
mov ax, dx
stosw
mov ax, cx
stosw
mov di, shoot_pos
mov ax, dx
stosw
mov ax, cx
stosw

mov dx, 80
mov cx, 80

mov di, cometa_pos
mov ax, dx
stosw
mov ax, cx
stosw

mov si, cometa
call _print_sprite

mov ax, 0h

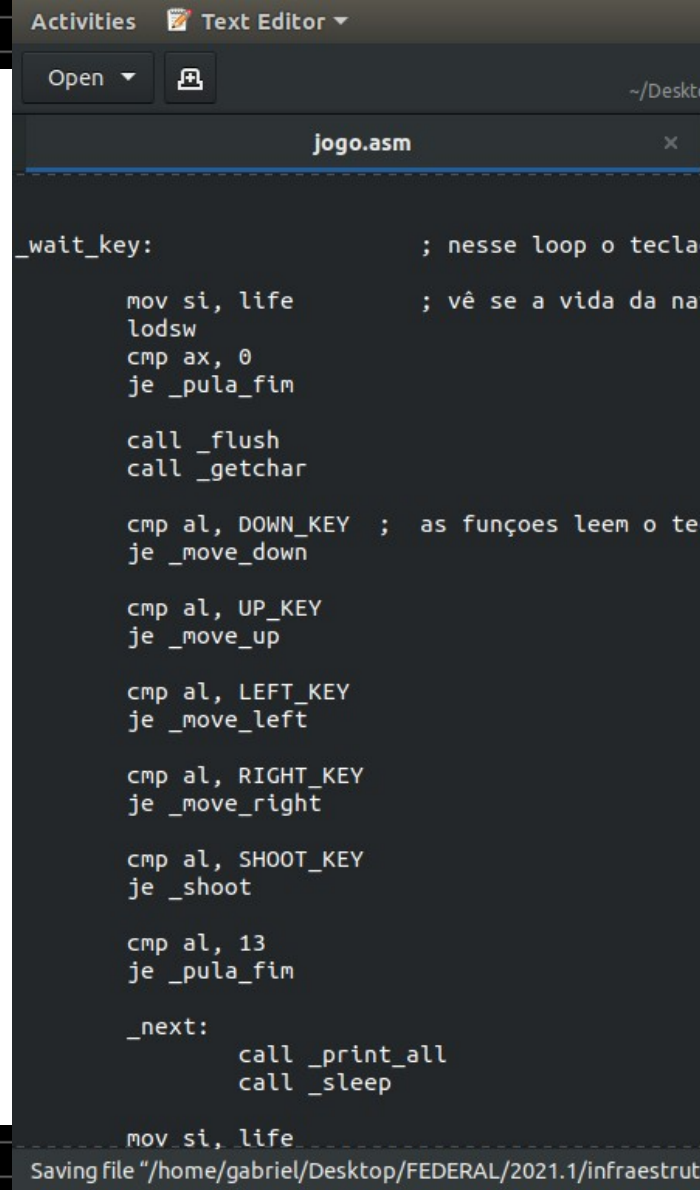
mov dx, DX_INICIAL
mov cx, CX_INICIAL

mov si, nave
call _print_sprite
```



# O jogo - jogo.asm

- A próxima função é `_wait_key`, que é um loop que lê o input do teclado e atualiza o estado dos objetos



The screenshot shows a text editor window titled "Text Editor" with a file named "jogo.asm" open. The code is written in assembly language and implements a game loop. The loop starts with a label "\_wait\_key:" and a comment "; nesse loop o teclado é lido". It then moves the "life" variable to the "si" register and loads a word from memory. A comparison is made between "ax" and 0, and a jump is made to "\_pula\_fim" if the value is less than or equal to 0. The code then calls "\_flush" and "\_getchar" to read a key. It then checks for specific keys: "DOWN\_KEY", "UP\_KEY", "LEFT\_KEY", "RIGHT\_KEY", and "SHOOT\_KEY", each with a corresponding jump to a function like "\_move\_down", "\_move\_up", "\_move\_left", "\_move\_right", and "\_shoot". After checking for these keys, it compares "al" with 13 and jumps to "\_pula\_fim" if less than or equal to 0. The loop then jumps to "\_next:", which calls "\_print\_all" and "\_sleep" before jumping back to "\_wait\_key:". The code ends with "mov si, life".

```
_wait_key:                                ; nesse loop o teclado é lido
    mov si, life                          ; vê se a vida da na
    lodsw
    cmp ax, 0
    je _pula_fim

    call _flush
    call _getchar

    cmp al, DOWN_KEY ; as funções leem o te
    je _move_down

    cmp al, UP_KEY
    je _move_up

    cmp al, LEFT_KEY
    je _move_left

    cmp al, RIGHT_KEY
    je _move_right

    cmp al, SHOOT_KEY
    je _shoot

    cmp al, 13
    je _pula_fim

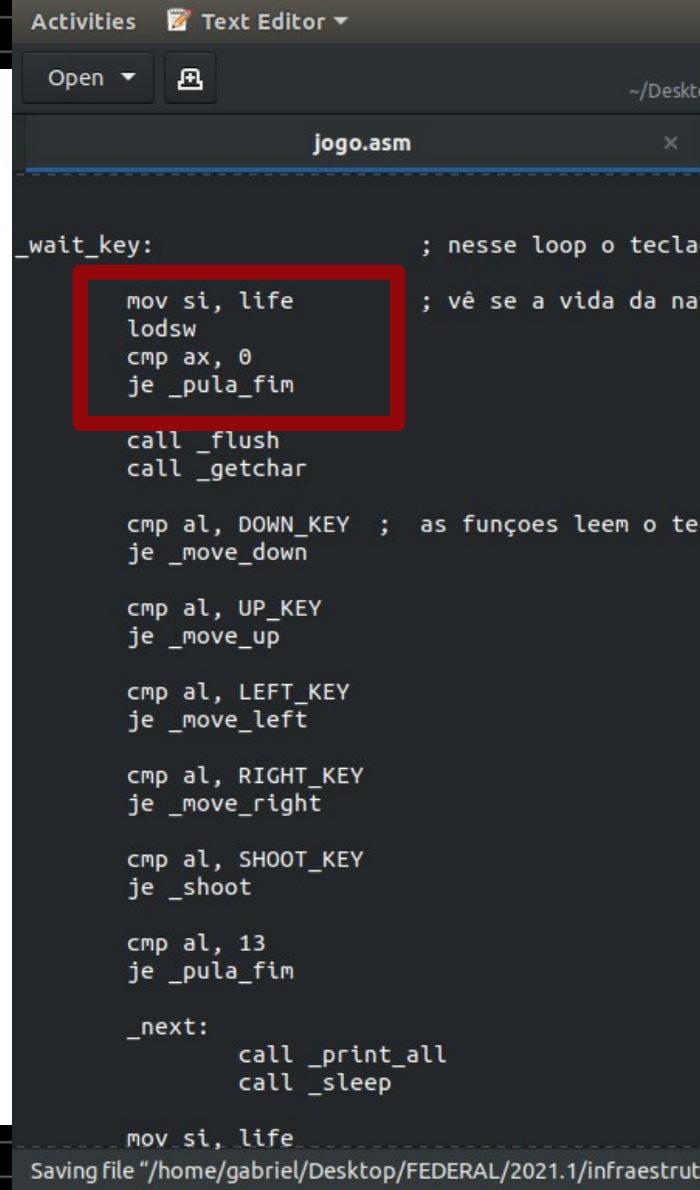
    _next:
        call _print_all
        call _sleep

    mov si, life
```

Saving file "/home/gabriel/Desktop/FEDERAL/2021.1/infraestrut

# O jogo - jogo.asm

- Primeiramente ele checa se a “vida” acabou.
- Se sim ele pula para `_pula_fim`, a função que finaliza o jogo



```
Activities Text Editor
Open
~/Desktop
jogo.asm
_wait_key: ; nesse loop o tecla
    mov si, life ; vê se a vida da na
    lodsw
    cmp ax, 0
    je _pula_fim
    call _flush
    call _getchar

    cmp al, DOWN_KEY ; as funções leem o te
    je _move_down

    cmp al, UP_KEY
    je _move_up

    cmp al, LEFT_KEY
    je _move_left

    cmp al, RIGHT_KEY
    je _move_right

    cmp al, SHOOT_KEY
    je _shoot

    cmp al, 13
    je _pula_fim

_next:
    call _print_all
    call _sleep

    mov si, life
Saving file "/home/gabriel/Desktop/FEDERAL/2021.1/infraestrut
```

# O jogo - jogo.asm

- Depois ele reseta o buffer do teclado e lê um caractere
- Depois ele checa a tecla e pula para a função a que ela se refere
- As teclas w,s,d,a movimentam a nave
- A tecla k atira
- E ENTER (13) pula para o fim também

```
Activities Text Editor
Open
~/Desktop
jogo.asm
_wait_key: ; nesse loop o teclado é lido
    mov si, life ; vê se a vida da nave não acabou
    lodsw
    cmp ax, 0
    je _pula_fim

    call _flush
    call _getchar

    cmp al, DOWN_KEY ; as funções leem o teclado
    je _move_down

    cmp al, UP_KEY
    je _move_up

    cmp al, LEFT_KEY
    je _move_left

    cmp al, RIGHT_KEY
    je _move_right

    cmp al, SHOOT_KEY
    je _shoot

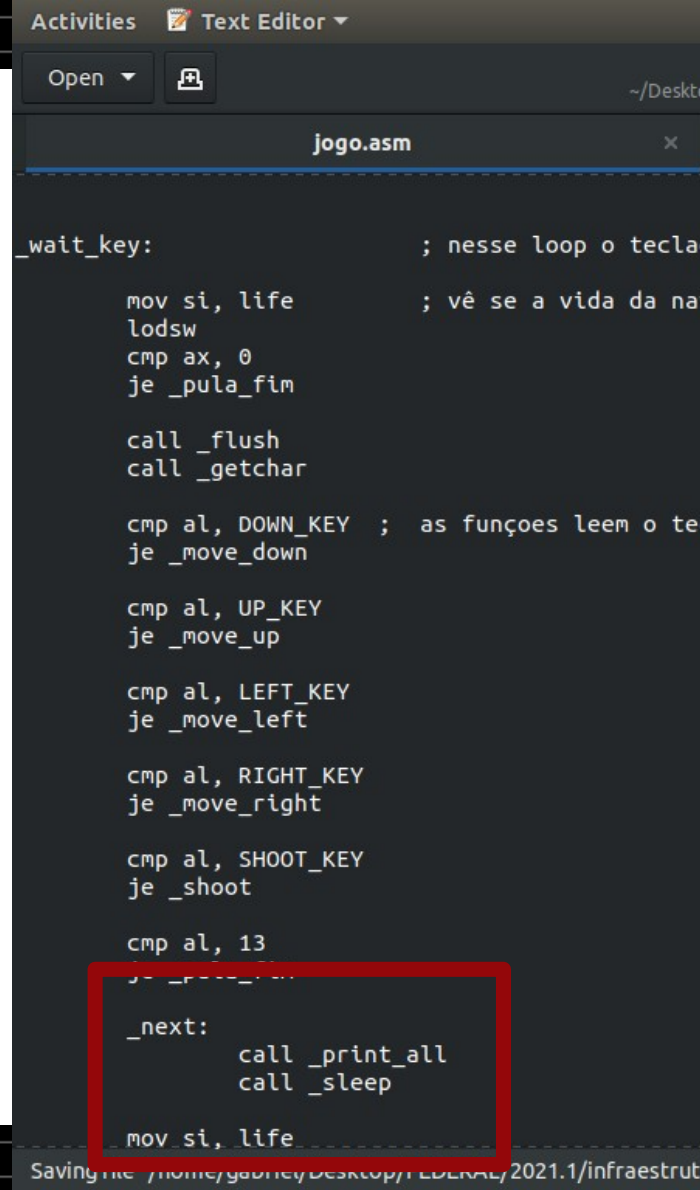
    cmp al, 13
    je _pula_fim

_next:
    call _print_all
    call _sleep

    mov si, life
Saving file "/home/gabriel/Desktop/FEDERAL/2021.1/infraestrutura/
```

# O jogo - jogo.asm

- A final ela printa e retorna para ela mesma em um ciclo



```
Activities Text Editor
Open
~/Desktop
jogo.asm

_wait_key: ; nesse loop o teclado é lido
    mov si, life ; vê se a vida da na
    lodsw
    cmp ax, 0
    je _pula_fim

    call _flush
    call _getchar

    cmp al, DOWN_KEY ; as funções leem o te
    je _move_down

    cmp al, UP_KEY
    je _move_up

    cmp al, LEFT_KEY
    je _move_left

    cmp al, RIGHT_KEY
    je _move_right

    cmp al, SHOOT_KEY
    je _shoot

    cmp al, 13
    je _pula_fim

_next:
    call _print_all
    call _sleep

    mov si, life

Saving file ~/home/yadney/Desktop/INFRA/2021.1/infraestrut
```

# O jogo - jogo.asm

- 4 funções movimentam a nave
- `_mov_down`, `_mov_up`, `_mov_left`, `_mov_right`
- Elas funcionam de maneira similar por isso só explicarei `_mov_down`

```
Activities Text Editor
Open
jogo.asm

mov si, life
lodsw
cmp ax, 0
je _pula_fim

jmp _wait_key

_move_down:

mov si, nave_pos           ; ca
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null               ; apag
call _print_sprite

mov ax, VEL_NAVE           ; mov
add dx, ax

call _ajustar_tela         ; che

mov di, nave_pos           ; co
mov ax, dx
stosw

mov ax, cx
stosw

jmp _next

_move_up:
```

# O jogo - jogo.asm

- Primeiro ele carrega a atual posição da nave
- Acessando o ponteiro nave\_pos e armazenando seus dados em dx e cx

```
Activities Text Editor
Open
jogo.asm

mov si, life
lodsw
cmp ax, 0
je _pula_fim

jmp _wait_key

_move_down:
    mov si, nave_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax

    mov si, null
    call _print_sprite

    mov ax, VEL_NAVE
    add dx, ax

    call _ajustar_tela

    mov di, nave_pos
    mov ax, dx
    stosw

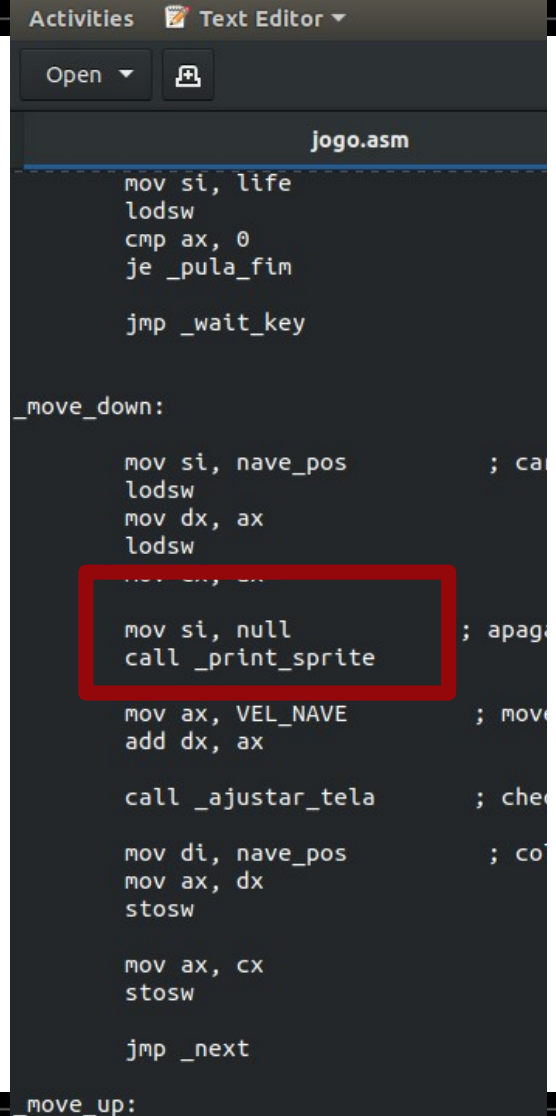
    mov ax, cx
    stosw

    jmp _next

_move_up:
```

# O jogo - jogo.asm

- Essa atual posição é apagada
- (Null é uma sprite preta usada justamente para apagar as outras sprites)



```
Activities Text Editor
Open
jogo.asm

mov si, life
lodsw
cmp ax, 0
je _pula_fim

jmp _wait_key

_move_down:

mov si, nave_pos ; ca
lodsw
mov dx, ax
lodsw

mov si, null ; apag
call _print_sprite

mov ax, VEL_NAVE ; mov
add dx, ax

call _ajustar_tela ; che

mov di, nave_pos ; co
mov ax, dx
stosw

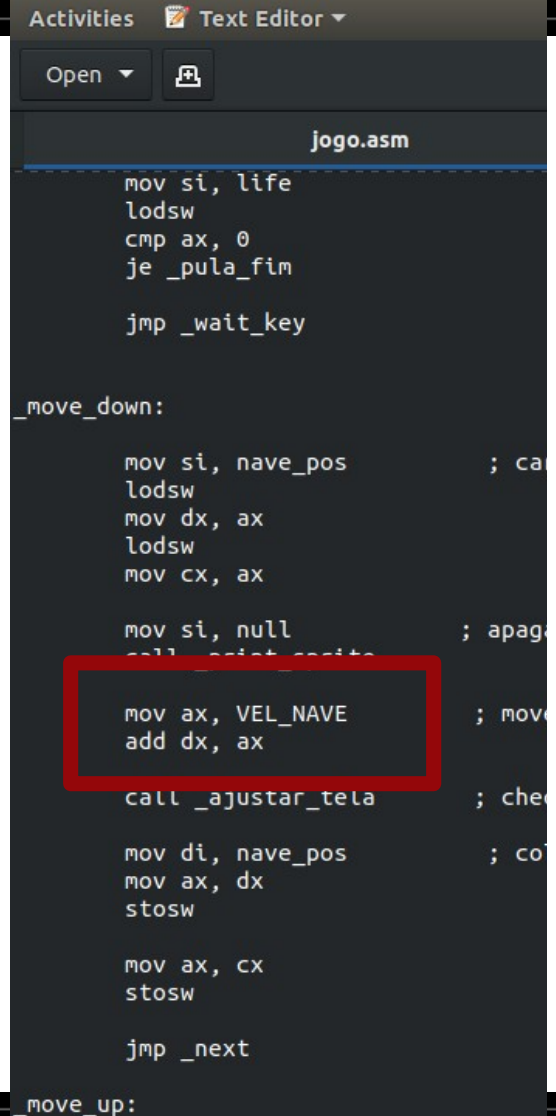
mov ax, cx
stosw

jmp _next

_move_up:
```

# O jogo - jogo.asm

- Depois a posição atual é atualizada
- No caso de mov\_down (que move a nave para baixo) é o registrador dx ( o “eixo y” ) é incrementado



```
Activities Text Editor
Open
jogo.asm

mov si, life
lodsw
cmp ax, 0
je _pula_fim

jmp _wait_key

_move_down:

mov si, nave_pos ; ca
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null ; apag
call _print_posita

mov ax, VEL_NAVE ; mov
add dx, ax

call _ajustar_tela ; che

mov di, nave_pos ; co
mov ax, dx
stosw

mov ax, cx
stosw

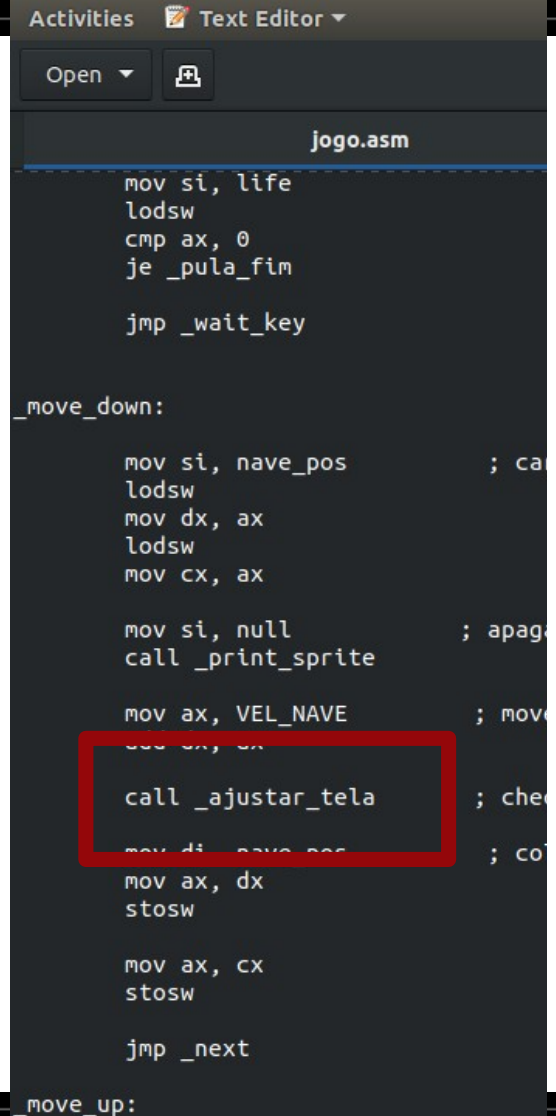
jmp _next

_move_up:
```



# O jogo - jogo.asm

- A função ajusta a tela confere se a nave esta dentro da resolução da tela



```
Activities Text Editor
Open
jogo.asm

mov si, life
lodsw
cmp ax, 0
je _pula_fim

jmp _wait_key

_move_down:

mov si, nave_pos ; ca
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null ; apag
call _print_sprite

mov ax, VEL_NAVES ; mov
xor cx, cx

call _ajustar_tela ; che

mov di, nave_pos ; co
mov ax, dx
stosw

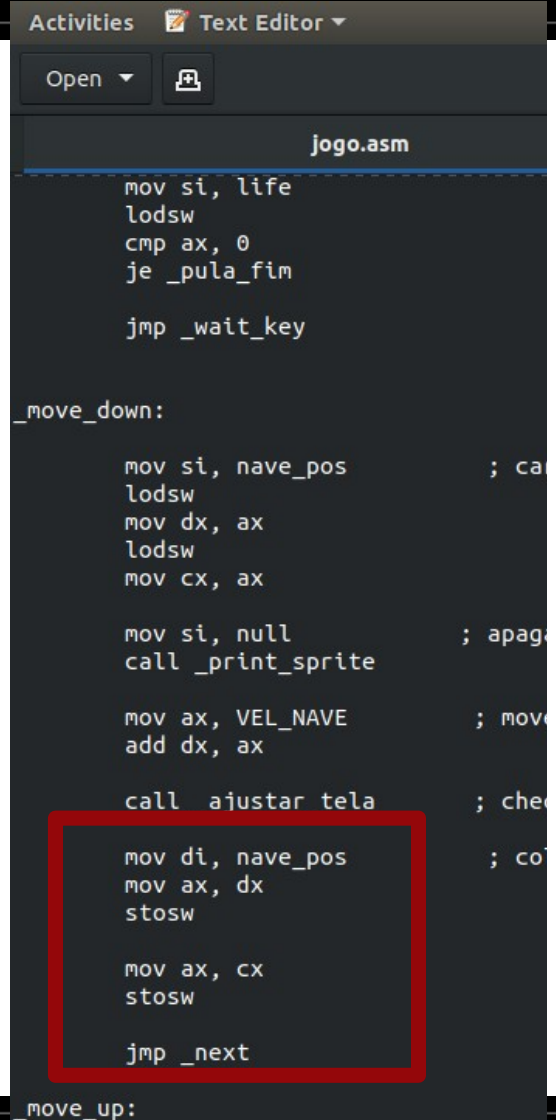
mov ax, cx
stosw

jmp _next

_move_up:
```

# O jogo - jogo.asm

- Por fim a nova posição é guardada de novo em nave\_pos e a função acaba



```
Activities Text Editor
Open
jogo.asm

mov si, life
lodsw
cmp ax, 0
je _pula_fim

jmp _wait_key

_move_down:

mov si, nave_pos ; ca
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null ; apag
call _print_sprite

mov ax, VEL_NAVE ; mov
add dx, ax

call ajustar_tela ; che

mov di, nave_pos ; co
mov ax, dx
stosw

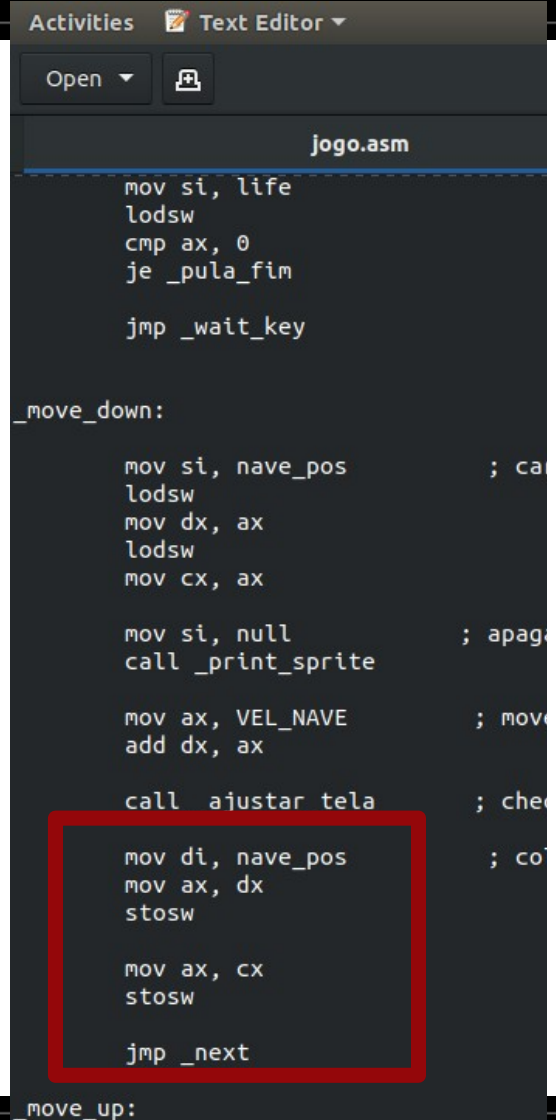
mov ax, cx
stosw

jmp _next

_move_up:
```

# O jogo - jogo.asm

- Como dito antes, o que diferencia os `_mov`'s são qual registrador (cx, dx) ele incrementa ou decrementa



```
Activities Text Editor
Open
jogo.asm

mov si, life
lodsw
cmp ax, 0
je _pula_fim

jmp _wait_key

_move_down:

mov si, nave_pos           ; ca
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null               ; apag
call _print_sprite

mov ax, VEL_NAVE           ; mov
add dx, ax

call ajustar_tela         ; che


mov di, nave_pos           ; co
mov ax, dx
stosw

mov ax, cx
stosw

jmp _next

_move_up:
```

Activities Text Editor ▾

Open ▾ 

jogo.asm

```

mov si, life
lodsw
cmp ax, 0
je _pula_fim

jmp _wait_key

_move_down:
mov si, nave_pos      ; ca
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null          ; apaga
call _print_sprite

mov ax, VEL_NAVE      ; move
add dx, ax

call _ajustar_tela    ; che

mov di, nave_pos      ; co
mov ax, dx
stosw


mov ax, cx
stosw

jmp _next

_move_up:

```

Activities Text Editor ▾

Open ▾ 

jogo.asm

```

mov ax, cx
stosw

jmp _next

_move_up:
mov si, nave_pos
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null
call _print_sprite

mov ax, VEL_NAVE
sub dx, ax

call _ajustar_tela

mov di, nave_pos
mov ax, dx
stosw


mov ax, cx
stosw

jmp _next

_move_right:
mov si, nave_pos
lodsw
mov dx, ax
lodsw

```

Activities Text Editor ▾

Open ▾ 

jogo.asm

```

jmp _next

_move_left:
mov si, nave_pos
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null
call _print_sprite

mov ax, 5h
sub cx, ax

call _ajustar_tela

cmp cx, 16
jge .continua

mov cx, 16

.continua:
mov di, nave_pos
mov ax, dx
stosw


mov ax, cx
stosw

jmp _next

shooting:

```

Activities Text Editor ▾

Open ▾ 

jogo.asm

```

mov ax, cx
stosw

jmp _next

_move_right:
mov si, nave_pos
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null
call _print_sprite

mov ax, 5h
add cx, ax

call _ajustar_tela

mov di, nave_pos
mov ax, dx
stosw

mov ax, cx
stosw

jmp _next

_move_left:
mov si, nave_pos
lodsw
mov dx, ax

```

## O jogo - jogo.asm

- A função shoot é um pouco diferente. Ela atualiza o tiro da nave
- A nave só dá um tiro por vez e só pode dar um novo tiro quando outro tiro acaba

```
_shoot:

    mov bx, 0
    mov si, shoot_life
    lodsw
    cmp ax, 0
    jg .fim

    mov di, shoot_life
    mov ax, 100
    stosw

    mov si, nave_pos
    lodsw
    mov dx, ax

    lodsw
    mov cx, ax
    add cx, 16

    call _ajustar_tela

    mov di, shoot_pos
    mov ax, dx
    stosw
    mov ax, cx
    stosw

.fim:
    jmp _next
```

## O jogo - jogo.asm

- Primeiramente ela carrega o ponteiro shoot\_life e compara com zero para ver se há ou não um tiro na tela
- Se houver ela sai de \_shoot e volta para a wait\_key

\_shoot:

```
mov bx, 0
mov si, shoot_life
lodsw
cmp ax, 0
jg .fim
```

```
mov di, shoot_life
mov ax, 100
stosw
```

```
mov si, nave_pos
lodsw
mov dx, ax
```

```
lodsw
mov cx, ax
add cx, 16
```

```
call _ajustar_tela
```

```
mov di, shoot_pos
mov ax, dx
stosw
mov ax, cx
stosw
```

```
.fim:
```

```
jmp _next
```

# O jogo - jogo.asm

- Se não houver tiro ele carrega a vida de um novo

```
_shoot:

    mov bx, 0
    mov si, shoot_life
    lodsw
    cmp ax, 0
    jg .fim

    mov di, shoot_life
    mov ax, 100
    stosw

    mov si, nave_pos
    lodsw
    mov dx, ax

    lodsw
    mov cx, ax
    add cx, 16

    call _ajustar_tela

    mov di, shoot_pos
    mov ax, dx
    stosw
    mov ax, cx
    stosw

.fim:
    jmp _next
```



# O jogo - jogo.asm

- Ele pega a atual posição da nave para saber de onde o tiro esta começando

```
_shoot:

    mov bx, 0
    mov si, shoot_life
    lodsw
    cmp ax, 0
    jg .fim

    mov di, shoot_life
    mov ax, 100
    stosw

    mov si, nave_pos
    lodsw
    mov dx, ax

    lodsw
    mov cx, ax
    add cx, 16

    call _ajustar_tela

    mov di, shoot_pos
    mov ax, dx
    stosw
    mov ax, cx
    stosw

.fim:
    jmp _next
```



## O jogo - jogo.asm

- Depois ele finaliza ajustando a posição do tiro e armazenando a posição em shoot\_pos

```
_shoot:

    mov bx, 0
    mov si, shoot_life
    lodsw
    cmp ax, 0
    jg .fim

    mov di, shoot_life
    mov ax, 100
    stosw

    mov si, nave_pos
    lodsw
    mov dx, ax

    lodsw
    mov cx, ax
    add cx, 16

    call _ajustar_tela

    mov di, shoot_pos
    mov ax, dx
    stosw
    mov ax, cx
    stosw

.fim:
    jmp _next
```

## O jogo - jogo.asm

- A ultima função pula\_fim  
printa game over varias  
vezes e chama a função  
\_fim, a função que termina  
o atual programa e volta  
para a tela do kernel

```
graphic.asm

mov ax, dx
stosw
mov ax, cx
stosw

.fim:
                                jmp _next

_pula_fim:
                                ;

                                call _flush

                                call _clear

                                mov si, gameover
                                call _print_line
                                mov si, gameover
                                call _print_line
                                mov si, gameover
                                call _print_line
                                mov si, gameover
                                call _print_line
                                mov si, gameover
                                call _print_line

                                mov si, press_enter
                                call _print_line
                                jmp _fim
```

# O jogo - graphic.asm

- O arquivo graphic.asm tem uma função gigante print\_all que printa, atualiza, checa vida do tiro do meteoro e tudo mais
- Porém não falarei dela agora

```
graphic.asm
#define VEL_SHOOT 8 ; algumas v
#define VEL_NAVES 7
```

```
_print_all:
```

```
mov si, nave_pos
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, nave
call _print_sprite
```

```
.print_cometa:
```

```
mov si, cometa_pos
lodsw
mov dx, ax
lodsw
mov cx, ax

mov si, null
call _print_sprite

mov di, cometa_pos
mov ax, dx
stosw
dec cx
mov ax, cx
```

# O jogo - graphic.asm

- Primeiro falarei sobre print\_sprite
- Ela é a função que imprime as imagens que cujo endereço esta em SI
- Eu criei um formato de imagem próprio para esse jogo

```
_print_sprite:                                ; imprime um sprite

    push dx
    push cx

    .loop:
        lodsb
        cmp al,'0'
        je .fim

        cmp al,'.'
        je .next_line

    .next_pixel:
        call _print_pixel
        inc cx
        jmp .loop

    .next_line:

        pop cx
        push cx

        inc dx
        lodsb
        jmp .next_pixel

    .fim:
        pop cx
        pop dx
        ;call _sleep
        ret

_print_pixel:
    mov ah, 0ch
    mov bh, 0
    int 10h
    ret
```

# O jogo - graphic.asm

[illegible]

# O jogo - graphic.asm

- As imagens dos sprites são um array onde cada ponto representa a cor do pixel naquele local
- Cada linha é terminada por ' . '
- E o sprite termina com ' 0 '
- Esse formato ajudar muito a ser interado

```

nave db
4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 4, 4, 7, 7, 7, 7, 4, 4, 4, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 1, 1, 1, 7, 7, 7, 4, 4, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 4, 4, 7, 1, 1, 1, 7, 4, 4, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 4, 7, 7, 7, 7, 1, 7, 4, 4, 4, 4, 0, 0,
0, 0, 0, 0, 0, 0, 4, 7, 7, 7, 7, 1, 7, 4, 4, 4, 4, 0, 0,
0, 0, 0, 0, 0, 4, 7, 7, 1, 1, 1, 7, 4, 4, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 1, 1, 1, 7, 7, 7, 4, 4, 0, 0, 0, 0, 0, 0,
0, 0, 0, 4, 7, 7, 7, 7, 7, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

```

# O jogo - graphic.asm

- Voltando a função print\_sprite...
- Primeiramente ela compara se o sprite chegou ao final checando se si esta com '0'

```
_print_sprite:                                ; imprime um sprite

    push dx
    push cx

    .loop
        lodsb
        cmp al, '0'
        je .fim

        cmp al, '.'
        je .next_line

    .next_pixel:
        call _print_pixel
        inc cx
        jmp .loop

    .next_line:

        pop cx
        push cx

        inc dx
        lodsb
        jmp .next_pixel

    .fim:

        pop cx
        pop dx
        ;call _sleep
        ret

_print_pixel:
    mov ah, 0ch
    mov bh, 0
    int 10h
    ret
```

# O jogo - graphic.asm

- Se não ele compara se chegou no final da linha

checando se si esta com

' '

```
_print_sprite:                                ; imprime um sprite

    push dx
    push cx

    .loop:
        lodsb
        cmp al, '0'
        je .fim

        cmp al, '.'
        je .next_line

        .next_pixel:
            call _print_pixel
            inc cx
            jmp .loop

        .next_line:
            pop cx
            push cx

            inc dx
            lodsb
            jmp .next_pixel

        .fim:
            pop cx
            pop dx
            ; call _sleep
            ret

_print_pixel:
    mov ah, 0ch
    mov bh, 0
    int 10h
    ret
```



# O jogo - graphic.asm

- A sub função next\_pixel printa o pixel na posição cx, dx

```
_print_sprite:                                ; imprime um sprite

    push dx
    push cx

    .loop:
        lodsb
        cmp al, '0'
        je .fim

        cmp al, '.'
        je .next_line

    .next_pixel:
        call _print_pixel
        inc cx
        jmp .loop

    .next_line:

        pop cx
        push cx

        inc dx
        lodsb
        jmp .next_pixel

    .fim:
        pop cx
        pop dx
        ;call _sleep
        ret

_print_pixel:
    mov ah, 0ch
    mov bh, 0
    int 10h
    ret
```

# O jogo - graphic.asm

- A next\_line pula para a próxima linha

```
_print_sprite:                                ; imprime um sprite

    push dx
    push cx

    .loop:
        lodsb
        cmp al,'0'
        je .fim

        cmp al,'.'
        je .next_line

    .next_pixel:
        call _print_pixel
        inc cx
        jmp .loop

    .next_line:
        pop cx
        push cx

        inc dx
        lodsb
        jmp .next_pixel

    .fim:
        pop cx
        pop dx
        ;call _sleep
        ret

_print_pixel:
    mov ah, 0ch
    mov bh, 0
    int 10h
    ret
```

# O jogo - graphic.asm

- Agora voltando a print\_all
- A função ficou monstruosamente gigantesca por fazer varias coisas

```
graphic.asm
#define VEL_SHOOT 8 ; algumas
#define VEL_NAVE 7
```

```
_print_all:
```

```
    mov si, nave_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax
```

```
    mov si, nave
    call _print_sprite
```

```
.print_cometa:
```

```
    mov si, cometa_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax
```

```
    mov si, null
    call _print_sprite
```

```
    mov di, cometa_pos
    mov ax, dx
    stosw
    dec cx
    mov ax, cx
```

# O jogo - graphic.asm

- Primeiro ela printa a nave

```
graphic.asm
#define VEL_SHOOT 8           ; algumas v
#define VEL_NAVES 7
```

```
_print_all:
```

```
    mov si, nave_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax

    mov si, nave
    call _print_sprite
```

```
.print_cometa:
```

```
    mov si, cometa_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax
```

```
    mov si, null
    call _print_sprite
```

```
    mov di, cometa_pos
    mov ax, dx
    stosw
    dec cx
    mov ax, cx
```

# O jogo - graphic.asm

- Depois ele printa o cometa e o tiro
- Aqui o bicho pega

```
graphic.asm
#define VEL_SHOOT 8           ; algumas v
#define VEL_NAVES 7
```

```
_print_all:
```

```
    mov si, nave_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax
```

```
    mov si, nave
    call print_sprite
```

```
.print_cometa:
```

```
    mov si, cometa_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax
```

```
    mov si, null
    call _print_sprite
```

```
    mov di, cometa_pos
    mov ax, dx
    stosw
    dec cx
    mov ax, cx
```

## O jogo - graphic.asm

- print\_cometa inicia limpando a sprite atual do cometa e carregando sua nova posição em dx, cx

```
call _print_sprite
```

```
.print_cometa:
```

```
mov si, cometa_pos  
lodsw  
mov dx, ax  
lodsw  
mov cx, ax
```

```
mov si, null  
call _print_sprite
```

```
mov di, cometa_pos  
mov ax, dx  
stosw  
dec cx  
mov ax, cx  
stosw
```

```
.cmp_shoot_cometa:
```

```
mov si, shoot_pos  
lodsw
```

```
sub ax, dx  
call _modulo_ax
```

```
cmp ax, 10  
jg .continua_
```

```
lodsw  
cmp ax, cx  
jg .reseta_cometa
```

```
.continua_:
```

## O jogo - graphic.asm

- Depois ele compara se o tiro e o cometa estão em posições próximas, isto é, se eles colidiram
- Se sim, a função .reseta\_cometa é chamada

```
call _print_sprite

.print_cometa:

    mov si, cometa_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax

    mov si, null
    call _print_sprite

    mov di, cometa_pos
    mov ax, dx
    stosw
    dec cx
    mov ax, cx
    stosw
```

```
.cmp_shoot_cometa:

    mov si, shoot_pos
    lodsw

    sub ax, dx
    call _modulo_ax

    cmp ax, 10
    jg .continua_

    lodsw
    cmp ax, cx
    jg .reseta_cometa

    .continua_:
```

.reseta\_cometa:

; reseta cometa para posicao inicial

```
mov si, shoot_pos
lodsw
mov dx, ax
lodsw
mov cx, ax
mov di, shoot_pos
;mov ax, 0
stosw
stosw
mov si, null
call _print_sprite
```

;apaga tiro:

I

```
mov si, null
call _print_sprite
mov cx, 200
mov di, cometa_pos
```

; apaga cometa

; recoloca posicao inicial em cx do cometa

```
push dx
mov dx, 0
pop ax
add ax, 41
push cx
mov cx, 100
div cx
pop cx
mov ax, dx
stosw
mov ax, cx
stosw
```

; da uma nova posição "aleatoria" para o cometa

III

```
mov di, shoot_life
mov ax, 0
stosw
```

; zera a vida do tiro pra que ele desapareca

IV



## O jogo - graphic.asm

- A função .reseta\_cometa faz o seguinte
- I – ela destroi a posição do tiro e sua sprite
- II – ela destroi a posição do cometa e sua sprite
- III – ela gera um posição pseudo aleatoria para o eixo y do cometa. Isso é feito atraves do modulo 100 **mod** 41
- IV – zera a vida do tiro

## O jogo - graphic.asm

- Outra comparação é se o meteoro esta numa posição cx menor que 10, ou seja, ele checa se o meteoro bateu na base

```
cmp cx,10  
jg .continue
```

```
.add_dano: ; se o meteoro bateu na base
```

```
mov si, life  
lodsw  
dec ax  
mov di, life  
stosw
```

```
mov si, vida_dx ; deleta o meteoro  
lodsw  
push ax  
mov dx, ax  
mov cx, 4  
mov si, null  
call _print_sprite
```

```
mov di, vida_dx  
pop ax  
add ax, 16  
stosw
```

## O jogo - graphic.asm

- Se for maior que cx (não bateu) ele pula e continua com a função
- Se não, a subfunção add\_dano adiciona dano a vida e apaga um pedaço da barra de vida

```
cmp cx,10  
jg .continue
```

```
.add_dano: ; se o meter
```

```
    mov si, life  
    lodsw  
    dec ax  
    mov di, life  
    stosw
```

```
    mov si, vida_dx ; deleta p  
    lodsw  
    push ax  
    mov dx, ax  
    mov cx, 4  
    mov si, null  
    call _print_sprite
```

```
    mov di, vida_dx  
    pop ax  
    add ax, 16  
    stosw
```

## O jogo - graphic.asm

- A ultima função a ser chamada é .continue
- Que finalmente printa o meteoro
- Ele é printado três vezes por motivos de... assembly...

```
.continue:
```

```
    mov si, cometa      ; pr  
    call _print_sprite  
    mov si, cometa  
    call _print_sprite  
    mov si, cometa  
    call _print_sprite
```

# O jogo - graphic.asm

- A função print\_tiro tem o mesmo modus operandi das demais ate aqui
- Ela checa se existe tiro pra ser printado
- Caso não ela sai do loop
- Caso sim ela :
  - → carrega de shoot\_pos a atual posição do sprite do tiro
  - → apaga a atual sprite do tiro
  - → atualiza a posição
  - → printa o sprite na nova posição
  - → guarda essa nova posição em shoot\_pos de novo
  - → confere se a vida do tiro acabou, caso ela apaga a atual posição e o destroi. Caso não ela sai da função

.print\_shoot:

```
    mov si, shoot_life
    lodsw
    cmp ax, 0
    jng .apaga_tiro
```

```
    sub ax, VEL_SHOOT
    mov di, shoot_life
    stosw
```

```
    mov si, shoot_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax
```

```
    push cx
    push dx
```

```
    mov si, null
    call _print_sprite
```

```
    add cx, VEL_SHOOT
```

```
    mov si, shoot
    call _print_sprite
```

```
    mov di, shoot_pos
    pop ax
    stosw
    pop ax
    add ax, VEL_SHOOT
    stosw
```

```
    mov si, shoot_life
    lodsw
    cmp ax, 0
    jg .fim
```

```
    push cx
    push dx
```

```
    mov si, null           ; l
    call _print_sprite
```

```
    add cx, VEL_SHOOT      ; f
```

```
    mov si, shoot          ; p
    call _print_sprite
```

```
    mov di, shoot_pos      ;
    pop ax
    stosw
    pop ax
    add ax, VEL_SHOOT
    stosw
```

```
    mov si, shoot_life
    lodsw
    cmp ax, 0              ; v
    jg .fim
```

.apaga\_tiro:

```
    mov si, shoot_pos
    lodsw
    mov dx, ax
    lodsw
    mov cx, ax
```

```
    mov si, shoot_num
    lodsw
    dec ax
    mov di, shoot_num
    stosw
```

```
    mov si, null
    call _print_sprite
```

## O jogo - graphic.asm

- Algumas funções auxiliares estão aqui como:
- `modulo_ax` – retorna o modulo do valor presente em `ax`
- `Sleep` – faz o programa dormir por alguns milisegundos
- `Ajusta_tela` – ajusta aposição das sprites

## O jogo - son.asm

- As funções do arquivo son.asm (escrevi som errado em tudo sem querer) são 3
- play\_note toca a nota musical cuja frequencia esta armazenada em ax

```
_play_note:      ; toca nota armazenada em ax

    push ax
    push dx

    mov dx, ax
    mov al, 0b6h
    out 43h, al
    mov ax, dx
    out 42h, al
    mov al, ah
    out 42h, al

    ; start the sound
    in al, 61h
    or al, 3h
    out 61h, al

    pop dx
    pop ax

    ret
```



## O jogo - son.asm

- stop\_note para de tocar qualquer nota

```
_stop_note:                                ; para o  
  
    push ax  
  
    in al, 61h  
    and al, 0fch  
    out 61h, al  
  
    pop ax  
    ret
```

## O jogo - son.asm

- atualiza\_som é responsável por administrar as notas
- Antes de falar nela é preciso apresentar os ponteiros note\_time e nota

## O jogo - son.asm

- note\_time guarda o tempo que uma nota sera tocada
- Nota guarda a nota a ser tocada

```
son_time times 10 db 0  
nota times 10 db 0
```

## O jogo - son.asm

- Em atualiza\_som, ela primeira checa em som\_time se existe som para ser tocado
- Se não houver ela finaliza a função

```
son_time times 10 db 0  
nota times 10 db 0
```

```
_atualiza_som:
```

```
    mov si, son_time  
    lodsw  
    cmp ax, 0  
    jbe .fim
```

```
    mov di, son_time  
    dec ax  
    stosw
```

```
    mov si, nota  
    lodsw  
    call _play_note
```

```
    pop ax
```

```
    ret
```

```
.fim:  
    call _stop_note  
    pop ax  
    ret
```

## O jogo - son.asm

- Caso existe ela decrementa o tempo da nota e depois a toca

```
son_time times 10 db 0
nota times 10 db 0

_atualiza_son:

    push ax

    mov si, son_time
    lodsw
    cmp ax, 0                ; existe
    jbe .fim                ; se n

    mov di, son_time
    dec ax
    stosw

    mov si, nota
    lodsw
    call _play_note

    pop ax

    ret

.fim:
    call _stop_note
    pop ax
    ret
```

## O jogo - son.asm

- A função `atualiza_son` é chamada em `jogo.asm` no loop `wait_key`

```
_wait_key:                                ;  
  
    call _atualiza_son  
  
    mov si, life                          ;  
    lodsw  
    cmp ax, 0  
    je _pula_fim  
  
    call _flush  
    call _getchar  
  
    cmp al, DOWN_KEY                      ; as  
    je _move_down  
  
    cmp al, UP_KEY  
    je _move_up
```

## O jogo - son.asm

- Já a modificação dos ponteiros note\_time e nota para que as notas sejam tocadas e o tempo delas estão localizadas em eventos específicos
- Eu as coloquei 3 momentos

```
_wait_key: ;  
  
    call _atualiza_son  
  
    mov si, life ;  
    lodsw  
    cmp ax, 0  
    je _pula_fim  
  
    call _flush  
    call _getchar  
  
    cmp al, DOWN_KEY ; as  
    je _move_down  
  
    cmp al, UP_KEY  
    je _move_up
```

## O jogo - son.asm

- Quando o cometa é destruído ,  
em .reseta\_cometa

```
.reseta_cometa:                                ; reset  
  
        push ax                                ; som  
        mov si, nota  
        lodsw  
        cmp ax, NOTA_DANO  
        je .ignore  
        mov di, nota  
        mov ax, NOTA_COMETA  
        stosw  
        mov di, son_time  
        mov ax, 5  
        stosw  
  
        .ignore:  
        pop ax  
  
        mov si, shoot_pos                      ; apaga t  
        lodsw
```



## O jogo - son.asm

- Quando o cometa bate na base, em .add\_dano

```
.add_dano:                                ; se  
    push ax  
    mov di, nota  
    mov ax, NOTA_DANO  
    stosw  
    mov di, son_time  
    mov ax, 10  
    stosw  
    pop ax  
    mov si, life  
    lodsw
```

## O jogo - son.asm

- E quando o tiro laser é criado em .print\_shoot

```
push ax
mov di, nota
mov ax, NOTA_SHOOT
stosw
mov di, son_time
mov ax, 5
stosw
pop ax
```

## O jogo - son.asm

- As contantes com o valor da nota a ser tocada estão no começo de graphic.asm

```
%define VEL_SHOOT 8 ; a
%define VEL_NAVE 7

%define NOTA_DANO 0a0h
%define NOTA_SHOOT 0feh
%define NOTA_COMETA 0ffh
```

## O jogo - son.asm

- Por fim, por algum motivo o qemu inicia com o som desligado.
- Pra habilitar o som do qemu é necessario sempre habilitar nas configurações o volume dele para que se possa ouvi-lo
- (Obrigado mateus soares por me avisar isso)

# O jogo - son.asm

The screenshot displays a Linux desktop environment with three main windows:

- File Manager:** Shows a directory structure with files like `boot1.asm`, `boot1.bin`, and `gabriel@ubuntugab: ~/Desktop/FEDERAL/2021.1/`. A file named `vida` is highlighted.
- Terminal:** Contains assembly code and game instructions:

```
%define VEL_SHOOT 8
%define VEL_NAVES 7

; algumas variaveis do jogo

boot1.asm
boot1.bin
gabriel@ubuntugab: ~/Desktop/FEDERAL/2021.1/
File
WARNI
perat
vida
*****
cuidado! nao deixe o cometa bater na bas
w = cima
s = baixo
a = esquerda
d = direita
k = atirar
para sair pressio ENTER x2
```
- System Settings:** The **Sound** settings window is open, showing the **Applications** tab. A red box highlights the volume controls for `qemu-system-i386`, which are currently set to **ON**.

## O jogo – data.asm

- Em data.asm estão armazenadas as sprites do jogo no formato que eu criei, os textos de apoio do jogo, os ponteiros usados nos outros dois arquivos



```
nave_pos times 10 db 0
life times 10 db 0

shoot_num times 10 db 0
shoot_life times 10 db 0
shoot_pos times 20 db 0

cometa_life times 10 db 0
cometa_pos times 10 db 0

vida db 10,10,10,10,10,10,10,10,10,10,"vida", 0
vida_dx times 10 db 0

gameover db "GAME OVER !!!",10,13, 0

press_enter db 10,13,10,13, "pressione ENTER", 0

texto_apoio db 10, 10,10,10,10,13,"*****"
cima",10,13,"s = baixo",10,13,"a = esquerda",10,13,"
0,10,13,"*****",
```

( Extra ): Coisas que eu não consegui fazer



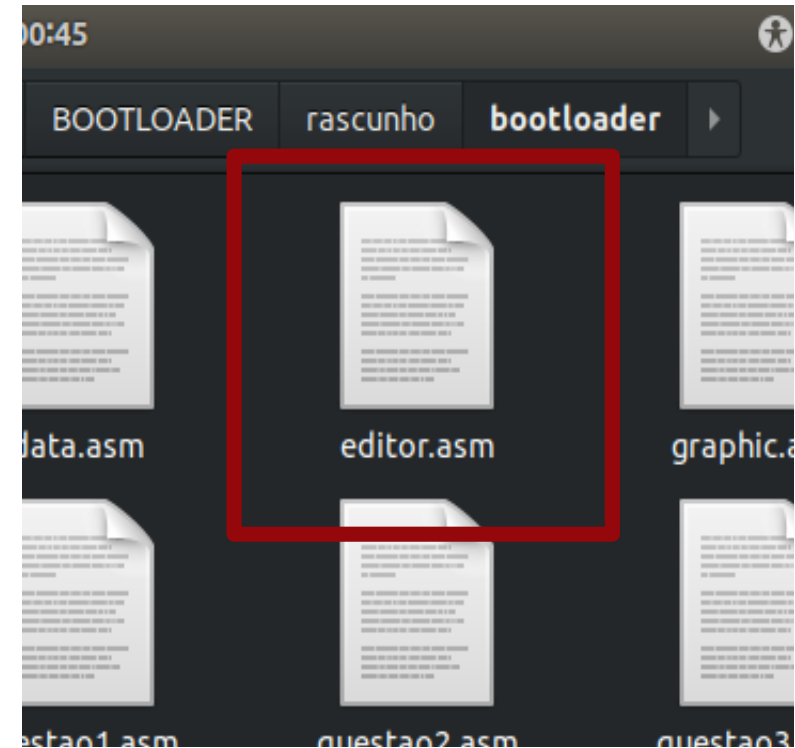


## ( Extra ): Coisas que eu não consegui fazer

- Tiveram algumas coisas que eu não consegui implementar
- Vou deixar a sobra delas , ou que eu tentei implementar delas no projeto

# O editor

- Queria implementar um editor de texto simples que escrevesse e lesse um texto diretamente do disco rígido
- Um monitor me falou que eu poderia usar a pilha ou ponteiros para guarda o texto. Mas eu queria muito operar um disco rígido
- Mas não consegue implementa-lo por que não soube como escrever e ler nenhuma interrupção de hard disk, como int 13, int 25,26



# O editor

- Tudo o que eu tentava dava errado
- Ao lado esta uma das várias funções que tentei ler o hard disk. Ela esta no arquivo editor.asm

```
_read_log:

    mov ah, 24h
    mov al, 2h
    mov dl, 80h
    ;int 13h

    mov ax, texto      ;0x7E
    mov es, ax
    xor bx, bx

    mov ah, 21h
    mov al, 1h         ;porção d
    mov ch, 0h         ;track 0
    mov cl, 3h         ;setor 3
    mov dh, 1h         ;head 0
    mov dl, 0h         ;drive 0
    int 13h

    mov si, marreco
    cmp ah, 0
    je .fim
    mov si, tigre

.fim
ret

_write_log:

    mov ah, 24h
    mov al, 1h
    mov dl, 80h
```

## editor

- O que eu queria mesmo fazer era um gerenciador de pastas e arquivos similar ao linux. Onde se poderia criar diretórios e escrever arquivos neles
- Mas seria muito complexo e eu não conseguia nada com o hard disk e desisti

fim