Relatório MC404 - Trabalho 2 - Prof. Rodolfo

Gustavo Sverzut Barbieri, Ivens Prates Telles Alves

Grupo:
Gustavo Sverzut Barbieri RA: 008849
Ivens Prates Telles Alves RA: 008908

1 O Projeto

O segundo projeto de mc404 consta em implementar um boot loader, ou seja, um programa que carregue um outro programa de menos de 512 bytes para o primeiro setor de um disquete com uma assinatura que permite à BIOS reconhecer que esse disquete trata-se de um disquete de boot. Fizemos esse programa como um programa que carregasse qualquer outro programa em assembly, com menos de 8,5kb, no boot. Esse segundo programa será copiado juntamente com o primeiro pelo nosso boot loader, mas para o setor 2 até o setor 18 (por isso cabendo 8,5Kb (17*512/1024). Assim, pudemos rodar o Escalonador dentro dele, o que provavelmente nos garantirá mais 20% em cada trabalho!

2 Funções/Procedimentos:

Nosso programa está basicamente dividido em 3 partes:

2.1 Boot Loader

O arquivo G27-Boot.asm contém o nosso programa que copia os outros 2 programas para o local correto no disquete para serem bootados. Ele se divide em:

Carregar Prog1.com: Carrega um arquivo chamado Prog1.com, que deve estar no mesmo diretório do programa compilado a partir do arquivo G27-Boot.asm, que deverá ter menos de 512 bytes. Esse parte ainda escreve a assinatura nos 2 ultimos bytes desse setor.

Carregar Prog2.com: Carrega um arquivo chamado Prog2.com, que deve estar no mesmo diretório do programa compilado a partir do arquivo G27-Boot.asm, que deverá conter menos de 512 bytes. Esse programa só será rodado se o programa Prog1.com for a compilação do arquivo G27-Carregador.asm. que será explicado adiante. Se o arquivo Prog2.com naum existir, o Boot Loader considera que o Prog1.com é um programa que será rodado sozinho, e assim deixa de copiar o Prog2.com sem maiores problemas.

2.2 Carregador

O arquivo G27-Carregador.asm é um pequeno programa que se compilado como Prog1.com e colocado no disquete rodando-se o Boot Loader, nos permite carregar na memória os próximos 17 setores do disquete, que seria o Prog2.com. Ele faz isso inicializando todos os registradores e segmentos, e depois copia para um lugar na memória (1234h) esses setores, no offset 0100h, para que o Prog2.com possa ser qualquer arquivo que rode no DOS, sem precisar de mudar o ORG para set bootado.

2.3 Move Objeto

É um programa independente que desenha uma cruz na tela e a movimenta pelas setas do teclado. Esse programa deverá ser compilador como Prog2.com, para assim ser carregado pelo Boot Loader no disquete. Como compilado tem menos que 512 bytes, pode ser carregado diretamente como Prog1.com, contanto que mude-se o [ORG 0x0100] para [ORG 0x7C00]. Ele se divide em algumas partes:

Movimenta: trata de capturar as setas do teclado e assim chamar as funções corretas para cuidar disso.
Teste: testa se a cruz passou dos limites da tela.
Pinta/Apaga: rotinas que pintam ou apagam a cruz da tela nas posições passadas por [POSX] e [POSY].
Move: Move para esquerda, direita, cima ou abaixo a cruz, apagando a cruz na posição atual e pintando a cruz na nova posição.

3 Decisões e Dificuldades encontradas:

3.1 Decisões

Decidimos fazer o boot loader focado no carregador, pois queríamos os 20%. Assim, desenhamos algo bem simples no Move Objeto (uma Cruz, ou um alvo, como desejar) e gastamos um maior tempo no boot loader e no carregador. Esses dois são padrão, então não tivemos nenhuma decisão muito importante quanto o que usar neles.

3.2 Dificuldades

Entender como o boot funciona, descobrir o RESB, e a maior de todas, entender como o carregador funcionaria, pois não sabíamos como dar um JUMP para a posiçào 0100h. Com o IRET, isso foi possível. O Boot Loader em si é simples (depois de ler muito), assim como o Move Figura.

```
;; Esse é o arquivo G27-Boot.asm

[BITS 16]
[ORG 0x0100]

[SEGMENT .text]

; Aqui temos o carregamento de um arquivo chamado prog1.com no
; primeiro setor do disquete.

      mov ah,3Dh            ; serviço da 21h para abrir arquivo
      mov al,00h            ; o arquivo será aberto com permissão
                            ; de ler apenas
      mov dx,arkivo1        ; nome do arquivo a ser lido (prog1.com)
      int 21h               ; chama a int 21h

      mov dx,msg1           ; coloca mensagem de erro Abrir Arquivo
      jc  Pulo              ; se o carry for 1, deu pau. OBS: Pulo
                            ; muito longo, usei uma ponte!

      mov [pontero],ax      ; coloca o file handle em pontero
      xor ax,ax             ; limpa ax

      mov ah,3Fh            ; serviço da 21h para abrir arquivo
      mov bx,[pontero]      ; copia o file handle contido em
                            ; pontero para bx
      mov cx,512            ; teremos até 512 bytes lidos
      mov dx,bloko          ; e será alocado na memória reservada
                            ; pela variável bloko
      int 21h               ; chama a int 21h

      mov dx,msg2           ; coloca mensagem de erro Ler Arquivo
      jc  Pulo              ; se o carry for 1, deu pau. OBS: uso
                            ; pulo pq senaum naum ia

      xor ax,ax             ; limpa ax

      ;; Faz a assinatura
      mov al,0x55           ; ajusta os 2 bytes do bloko para ser
                            ; identificado como boot
      mov [ds:bloko+510], al; Primeiro Byte
      mov al,0xAA
      mov [ds:bloko+511], al; Segundo Byte
```

```
      xor ax,ax             ; limpa ax

      mov ah,03h            ; coloca os dados da memória no setor
                            ; desejado
      mov al,1              ; numero de setores a serem lidos
      mov ch,0              ; track
      mov cl,1              ; sector
      mov dh,0              ; head number
      mov dl,0              ; 0 = drive de diskete
      mov bx,bloko          ; mov ES:BX
      int 13h

      mov dx,msg4           ; coloca mensagem de erro Copiar para
                            ; Setor
      jc msg_erro           ; se o carry for 1, deu pau

      xor ax,ax             ; limpa ax

; -------------
      jmp Pula_o_pulo       ; PONTE!!!
Pulo:                       ; Ignore
      jmp msg_erro          ; essa
Pula_o_pulo:                ; parte!!!
; -------------

      mov ah,3Eh            ; serviço da 21h para fechar arquivo
      mov bx,[pontero]      ; seleciona o file handle
      int 21h               ; chama a int 21h

      mov dx,msg3           ; coloca mensagem de erro Fechar Arquivo
      jc  msg_erro          ; se o carry for 1, deu pau

; Aqui começa o carregamento do segundo arquivo, prog2.com, que pode
; ter no máximo 8704 bytes (8,5Kb), q eh o que cabe em 17
; setores. Caso o arquivo prog2.com naum exista, essa parte será
; pulada para o fim do programa.

      mov ah,3Dh            ; serviço da 21h para abrir arquivo
      mov al,00h            ; o arquivo será aberto com permissão
                            ; de ler apenas
      mov dx,arkivo2        ; nome do arquivo a ser lido (prog2.com)
      int 21h               ; chama a int 21h
```

```
      mov dx,msg5           ; coloca mensagem de erro Abrir Arquivo
      jc  msg_erro          ; se o carry for 1, soh carrega
                            ; prog1.com, pois
                            ; prog2.com naum existe.

      mov [pontero],ax      ; coloca o file handle em pontero
      xor ax,ax             ; limpa ax

      mov ah,3Fh            ; serviço da 21h para abrir arquivo
      mov bx,[pontero]      ; copia o file handle contido em
                            ; pontero para bx
      mov cx,8704           ; teremos até 8704 bytes lidos
      mov dx,blokao         ; e será alocado na memória reservada
                            ; pela variável blokao
      int 21h               ; chama a int 21h

      xor ax,ax             ; limpa ax

        mov ah,03h          ; coloca os dados da memória no setor
                            ; desejado
        mov al,17           ; numero de setores a serem lidos
        mov ch,0            ; track
        mov cl,2            ; sector
        mov dh,0            ; head number
        mov dl,0            ; 0 = drive de diskete
      mov bx,blokao         ; mov ES:BX
      int 13h               ; chama a int 13h para copiar

      mov dx,msg4           ; coloca mensagem de erro Copiar para
                            ; Setor
      jc msg_erro           ; se o carry for 1,deu pau

      xor ax,ax             ; limpa ax

      mov ah,3Eh            ; serviço da 21h para fechar arquivo
      mov bx,[pontero]      ; seleciona o file handle
      int 21h               ; chama a int 21h

      mov dx,msg3           ; coloca mensagem de erro Fechar Arquivo
      jc  msg_erro          ; se o carry for 1, deu pau

Fim:
      int 20h               ; sai do programa
```

```
msg_erro:                   ; imprime string em dx e sai do programa
      mov ah,09h
      int 21h               ; imprime string
      int 20h               ; sai do programa


[SEGMENT .data]

      pontero dw 0
      arkivo1 db "prog1.com",0
      arkivo2 db "prog2.com",0
      bloko resb 512
      blokao resb 8704
      msg1 db "Naum consegui abrir o arkivo prog1.com!$"
      msg2 db "Naum consegui ler o arkivo!$"
      msg3 db "Naum consegui fechar o arkivo!$"
      msg4 db "Naum consegui copiar para o setor desejado!$"
      msg5 db "O programa soh carregou o arquivo prog1.com!$"
```

```
;; Esse é o arquivo G27-Carregador.asm

[BITS 16]
[ORG 0x0100]

[SEGMENT .text]

; Instruções básicas para o bom andamento do programa do primeiro setor

mov ax,0 ;    Inicia
mov bx,0 ;      os
mov cx,0 ;    quatro
mov dx,0 ; registradores
mov ax,cs ; Copia cs para ax. cs e ip jah veem iniciados.
mov ds,ax ; copia o cs para ds
mov ss,ax ; copia o cs para ss
mov es,ax ; copia o cs para es
mov sp,0F000h ; inicia a pilha com um valor razoavelmente alto
xor ax,ax ; limpa ax!

; Fim das instruções básicas. Iniciando o programa em si:

mov ax,1234h ; Escolhe um lugar na memória
mov es,ax ; para o programa rodar.
        mov ah,02h ; Coloca os dados do setor no lugar desejado de
                        ; memória
        mov al,17 ; Numero de setores a serem lidos
        mov ch,0 ; Track
        mov cl,2                ; Sector
        mov dh,0                ; Head Number
        mov dl,0 ; 0 = drive de diskete
mov bx,0100h ; Escolhe o local onde será rodado o programa.
; No padrão DOS, [ORG 0x0100]
int 13h ; chama a int 13 para ler na memória

   pushf ; coloca as flags na pilha
   mov ax,1234h ; coloca o endereço de memória escolhido
   push ax ; acima na pilha, onde o programa está.
   mov ax,0100h ; coloca o endereço de offset escolhido
   push ax ; acima na pilha, para o [ORG XXXXXX].
   iret ; Joga tudo para ser rodado!
```

```
;; Esse é o arquivo G27-MoveObjeto.asm

[BITS 16]
[ORG 0x0100]

[SEGMENT .text]

    mov ah,00h       ; selecionar modo de vídeo
    mov al,12h       ; modo gráfico
    int 10h          ; chama o serviço de vídeo da BIOS

    mov cx,[POSX]    ; Se assegura de que a primeira cruz
    mov dx,[POSY]    ; será pintada no centro
    call Pinta_Cruz  ; Pinta uma Cruz (alvo)

Movimenta:
    mov ah,10h       ; Espera até vir uma entrada pelo
                     ; teclado
    int 16h          ; e coloca em ax

    cmp ah,4Bh       ; se foi pressionado seta para esquerda,
    je TesteEsquerda ; vai para TesteEsquerda
    cmp ah,4Dh       ; se foi pressionado seta para direita,
    je TesteDireita  ; vai para TesteDireita
    cmp ah,50h       ; se foi pressionado seta para cima,
    je TesteCima     ; vai para TesteCima
    cmp ah,48h       ; se foi pressionado seta para baixo,
    je TesteBaixo    ; vai para TesteBaixo

    jmp Movimenta    ; Loop

; ------------------------------------------------------------- ;

    ;; Rotinas idênticas para testar se o objeto não está
    ;; ultrapassando
    ;; os limites da tela (640x480)

TesteEsquerda:
    mov ax,[POSX]
    cmp ax,10
    je Movimenta
    jmp Esquerda
```

```
TesteDireita:
    mov ax,[POSX]
    cmp ax,630
    je Movimenta
    jmp Direita

TesteCima:
    mov ax,[POSY]
    cmp ax,470
    je Movimenta
    jmp Cima

TesteBaixo:
    mov ax,[POSY]
    cmp ax,10
    je Movimenta
    jmp Baixo

    ;; Fim das rotinas de teste de ultrapassagem da tela

; ------------------------------------------------------------- ;

    ;; Rotinas para pintar/apagar o objeto Cruz da tela

Pinta_Cruz:
    mov al,4         ; color 4 - red
    call Cruz        ; chama a rotina Cruz
    ret

Apaga_Cruz:
    mov al,0         ; color 0 - black
    call Cruz        ; chama a rotina Cruz
    ret

Pinta:
    push ax          ; salva ax pois ele será mudado
    mov ah,0Ch       ; function 0Ch - Write
    int 10h          ; call BIOS Video Service
    pop ax           ; retorna o ax salvo
    ret

Cruz:
    mov cx,[POSX]    ; Assegura que cx e dx estaum
    mov dx,[POSY]    ; nas posições atuais.
```

```
    mov bx,21         ; Desenha
    sub cx,10         ;   a
    call Reta1        ;   reta
    mov cx,[POSX]     ;   horizontal
    mov bx,21         ; Desenha
    sub dx,10         ;   a
    call Reta2        ;   reta
    mov dx,[POSY]     ;   vertical

    ret

Return:
    ret

Reta1:
    call Pinta        ; pinta o ponto atual
    inc cx            ; anda 1 pixel para a direita
    dec bx            ; decrementa o contador
    cmp bx,0          ; se o contador eh zero,
    je Return         ; retorna para a funcao q a chamou
    jmp Reta1         ; loop

Reta2:
    call Pinta        ; pinta o ponto atual
    inc dx            ; anda um pixel para cima
    dec bx            ; decrementa o contador
    cmp bx,0          ; se o contador eh zero,
    je Return         ; retorna para a funcao q a chamou
    jmp Reta2         ; loop

    ;; Fim das rotinas para desenhar/apagar a cruz na tela

; ------------------------------------------------------------ ;

    ;; Rotinas para escrever a Cruz quando elas saum movimentadas

Esquerda:
    call Apaga_Cruz   ; Apaga a cruz nas coordenadas atuais
    mov cx,[POSX]     ; x position
    sub cx,5          ; anda para a esquerda
    mov [POSX],cx     ; salva a nova posição
    mov dx,[POSY]     ; y position
    call Pinta_Cruz   ; chama a rotina para pintar nas coordenadas
                      ; escolhidas
```

```
    jmp Movimenta    ; volta para a rotina de Movimentacao

Direita:
    call Apaga_Cruz  ; Apaga a cruz nas coordenadas atuais
    mov cx,[POSX]    ; x position
    add cx,5         ; anda para a direita
    mov [POSX],cx    ; Salva a posicao
    mov dx,[POSY]    ; y position
    call Pinta_Cruz  ; chama a rotina para pintar nas coordenadas
                     ; escolhidas
    jmp Movimenta    ; volta para a rotina de Movimentacao

Cima:
    call Apaga_Cruz  ; Apaga a cruz nas coordenadas atuais
    mov cx,[POSX]    ; x position
    mov dx,[POSY]    ; y position
    add dx,5         ; Anda para cima
    mov [POSY],dx    ; Salva a posição
    call Pinta_Cruz  ; chama a rotina para pintar nas coordenadas
                     ; escolhidas
    jmp Movimenta    ; volta para a rotina de Movimentacao

Baixo:
    call Apaga_Cruz  ; Apaga a cruz nas coordenadas atuais
    mov cx,[POSX]    ; x position
    mov dx,[POSY]    ; y position
    sub dx,5         ; anda para baixo
    mov [POSY],dx    ; salva posição
    call Pinta_Cruz  ; chama a rotina para pintar nas coordenadas
                     ; escolhidas
    jmp Movimenta    ; volta para a rotina de Movimentacao

    ;; Fim
; ------------------------------------------------------------- ;

[SEGMENT .data]
    POSX dw 320
    POSY dw 240

; ------------------------------------------------------------- ;
```