Sistemas Embarcados I – Laboratório 05

Objetivo: Implementar programas que fazem chamadas de funções passando parâmetros pela pilha.

Para este laboratório, o aluno utilizará o conjunto de rotinas gráficas que se encontram no arquivo "linec.asm". O arquivo é composto pelas seguintes funções:

Chamada	Parâmetros a serem passados	Modo	Função
cursor	dh = linha (0-39) e dl=coluna (0-79)	Gráfico (VGA)	Posiciona cursor
caracter	al= caracter a ser escrito; cor definida na variavel cor	Gráfico (VGA)	Escreve 1 caracter
plot_xy	push x; push y; call plot_xy; ($x \leq 639$, $y \leq 479$); cor definida na variavel cor	Gráfico (VGA)	Coloca um pixel na na posição (x,y)
circle	push xc; push yc; push r; call circle; ($xc+r \leq 639, yc+r \leq 479$)e($xc-r \geq 0, yc-r \geq 0$) ; cor definida na variavel cor	Gráfico (VGA)	Desenha uma circunferência
full_circle	push xc; push yc; push r; call full_circle; ($xc+r \leq 639, yc+r \leq 479$) e ($xc-r \geq 0, yc-r \geq 0$); cor definida na variavel "cor"	Gráfico (VGA)	Desenha um círculo e o colore
line	push x1; push y1; push x2; push y2; call line; ($x \leq 639$, $y \leq 479$)	Gráfico (VGA)	Desenha um segmento de reta

Tais funções fazem uso da interrupção de software "int 10h" definida pela BIOS (*Basic Input Output System*). O modo VGA (*Video Graphics Array*) permite uma resolução de 640×480 pontos em modo gráfico, cada ponto com até 16 cores. Permite igualmente 256 cores com uma definição de 320×200 pontos.

Observe que nas chamadas a estas funções, os parâmetros desejados (posição, cor, etc.) são passados pela pilha. Por exemplo:

Chamada da função	Função chamada

.
.
.
;A variável " cor" recebe um valor na faixa [0,15]. A
;declaração das cores segue abaixo (declarado
;em *segment dados*)
 	
mov byte [cor], azul
;Empilha as coordenadas (x,y)
push ax
push dx
 	
;Chama a função.
call plot_xy
.
.
.
segment dados
preto equ 0
azul equ 1
verde equ 2
cyan equ 3
vermelho equ 4
magenta equ 5
marrom equ 6
branco equ 7
cinza equ 8
azul_claro equ 9
verde_claro equ 10
cyan_claro equ 11
rosa equ 12
magenta_claro equ 13
amarelo equ 14
branco_intenso equ 15
	plot_xy:
	;Faz BP apontar para o topo da pilha, antes de salvar o
	;contexto
	push bp
	mov bp,sp
	
	;Salvando o contexto, empilhando registradores
	pushf
	push ax
	push bx
	push cx
	push dx
	push si
	push di
	;Preparando para chamar a int 10h
	; cor é uma variável global
	mov ah,0ch
	mov al,[cor]
	mov bh,0
	mov dx,479
	;Aqui, desempilha-se os parâmetros passados pela pilha,
	;no caso 2 *words* (16 bits cada)
	sub dx,[bp+4]
	mov cx,[bp+6]
	int 10h
	; recupera-se o contexto
	pop di
	pop si
	pop dx
	pop cx
	pop bx
	pop ax
	popf
	pop bp
	
	;O "4" como parâmetro de "ret" faz um *flush* na pilha,
	;desempilhando os parâmetros ;passados pela pilha na
	;chamada.
	ret 4

1) Estude a passagem de parâmetros, conforme mostrado acima. Faça um diagrama de como a passagem se processa na pilha, mostrando SP e BP. Quando ocorre um *call* (do tipo *near*), observe que é empilhado o registrador IP. Para o exemplo da tabela acima, tente explicar as instruções "sub dx,[bp+4]" e " mov cx,[bp+6]" e o motivo de "ret 4". Veja outros exemplos para as rotinas "line" e "full_circle".

2) Gere o código executável, usando NASM/FREELINK, e veja o comportamento do programa (são gerados vários segmentos de reta e círculos, formando um desenho). Modifique os parâmetros e veja o comportamento.

3) Faça agora um programa que desenhe um quadrado de 640×480 pixels, borda branca e fundo preto. Em processamento de imagens/vídeo, o ponto (0,0) é o ponto superior esquerdo da tela. Depois, acrescente um círculo, de cor vermelha e raio = 10, no meio da tela.

4) Agora, faça uma animação com o círculo vermelho (bola vermelha), de modo que, logo no início da animação, a bola se desloque a 45°, para cima, pela tela e ao se chocar com as laterais, esta deve desviar de trajetória da mesma forma que um raio de luz o faria ao ser refletido por uma superfície reflexiva especular. Para fazer o tempo de animação, use a função "delay", ajustando seus parâmetros para o seu programa.

```
delay:  ; Esteja atento pois talvez seja importante salvar  contexto (no caso, CX, o que NÃO foi feito aqui).
      mov       cx, word [velocidade]     ; Carrega "velocidade" em cx (contador para loop)
del2:
      push      cx                ; Coloca cx na pilha para usa-lo em outro loop
      mov       cx, 0800h         ; Teste modificando este valor
del1:
      loop      del1              ; No loop del1, cx  é decrementado até que volte a ser zero
      pop       cx                ; Recupera cx da pilha
      loop      del2              ; No loop del2, cx é decrementado até que seja zero
      ret
```

Observe que, no início, deve-se escolher o modo gráfico do vídeo. Portanto, é necessário armazenar o modo inicial. Isto é feito usando AH=0Fh e chamando-se INT 10H. O valor de retorno em AL deve ser guardado em uma variável, por exemplo "modo_anterior", para ser restaurado ao sair do programa. Depois, para por no modo VGA, usa-se AX= 12H e chama-se INT 10h. Ao sair, com AL=[modo anterior], ao se fazer AH=0Fh e chamando-se INT 10H, restaura-se o modo de vídeo original. Então, as partes inicial e final de seu código devem ser, como se segue:

```
segment code
..start:
        mov     ax,data
        mov     ds,ax
        mov     ax,stack
        mov     ss,ax
        mov     sp,stacktop
; salvar modo corrente de vídeo
        mov     ah,0Fh
        int     10h
        mov     [modo_anterior],al
; alterar modo de video para gráfico 640x480 16 cores
        mov     al,12h
        mov     ah,0
        int     10h
                ; Aqui entra seu código (loop infinito)  para fazer a animação.
; Para sair, faça
sai:
      mov     ah,0                    ; set video mode
      mov     al,[modo_anterior]      ; recupera o modo anterior
      int     10h
      mov     ax,4c00h
      int     21h
```

Como seu programa será um *loop* infinito, é necessário colocar uma forma de sair do programa. Assim, dentro de seu *loop* infinito, o seguinte trecho de código, baseado na int 21h, deve aparecer:

```
    mov  ah,0bh
     int  21h          ; Le buffer de teclado
     cmp  al,0       ; Se AL =0 nada foi digitado. Se AL =255 então há algum caracter na STDIN
    jne  adelante
    jmp segue       ; se AL = 0 então nada foi digitado e a animação do jogo deve continuar
adelante:
     mov ah, 08H   ;Ler caracter da STDIN
     int 21H
     cmp al, 's'      ;Verifica se foi 's'. Se foi, finaliza o programa
     jne adianta
     jmp sai
segue:    a animação deve seguir a partir daqui
```

Observe que para AH=0bh, a INT 21H apenas averigua se o *buffer* do teclado foi carregado com algum valor de tecla (e não fica esperando pela digitação da tecla). Se AL = 0 então nada foi digitado; se AL =255 então há algum caracter no *buffer* que precisa ser lido. Caso haja caracter, fazendo-se ah=8 e chamando-se a int 21h o resultado da tecla digitada aparece em al. Observe que AH=8 não mostra (ecoa) o caracter na tela.

Lembre-se que santos condicionais (jne, jc, je, jz, jnz, etc) são deslocam o IP na faxa máxima [-128, 127]. Para saltos maiores que a faixa permite, na versão do NASM16 que usamos no laboratório, ocorre erro de compilação. Porém, em programas, é comum acontecerem saltos extrapolarem esta faixa. Portanto, faça conforme o exemplo:

O que gostaria de fazer (o rótulo “ igual” indica uma posição de memória que extrapola a faixa [-128, 127]. Situação onde “igual” está muito longe.	Mas deu erro de estouro de salto. Então, deve-se fazer (observe que usa-se a instrução jmp, que não sofre deste problema):
. . . cmp ax, FFFFh je igual jmp diferente . . . igual:	. . . cmp ax, FFFFh jne diferente jmp igual diferente: ; aqui vêm as instruções que tratam quando for diferente jmp vai_continuar_em_algum_outro_ponto . . . igual: