Universidade Federal do Espirito Santo

Departamento de Engenharia

Engenharia de Computação

Laboratório 03
Montagem de programas

Aluno(a): Arthur Coelho Estevão

Professora: Camilo Arturo Rodriguez Diaz

Vitória

2022

1 Introdução

Este Laboratório tem como objetivo principal Verificar a codificação básica das instruções dos microprocessadores 8086/8088 e os seus modos de endereçamento da memória.

2 Teoria

2.1 NASM

Netwide Assembler (NASM), é um montador e desmontador que suporta as arquiteturas IA-32 e x86-64. O NASM permite o desenvolvimento de linguagem de baixo nível(Assembly) em diversas arquiteturas de sistemas operacionais, sendo mais popular para o Linux.

NASM foi originalmente escrito por Simon Tatham com a ajuda de Julian Hall. A partir de 2016, ela é mantida por uma pequena equipe liderada por H. Peter Anvin. É um software de código aberto lançado sob os termos de uma licença BSD simplificada (2 cláusulas).

2.2 Freelink

Freelink tmabém pode ser chamado de linkador, para a computação, um linkador, vinculador ou editor de ligação é um programa utilitário que recebe um ou mais arquivos objeto gerados por um compilador e combina-os em um único arquivo executável, arquivo de biblioteca ou outro arquivo 'objeto'.

2.3 Instruções de montagem

Na linha de comando:

- nasm (ou nasm16) $<nome\_do\_arquivo>$.asm -f obj -o $<nome\_do\_arquivo>$.obj -l $<nome\_do\_arquivo>$.lst $<nome\_do\_arquivo>$.asm (assemblando o código .asm)
- freelink $<nome\_do\_arquivo>$ (não tem .lst e nem .obj mesmo) (linkando o programa)
- $<nome\_do\_arquivo>$.exe (rodando o programa)

Com a configuração dos caminhos e possível executar utilizando (nasm $<nome\_do\_arquivo>$) que o DOSBox já vai rodar a primeira linha (nasm -f obj ...) sem precisar digitar todo o comando. Outro atalho é utilizando o nasm2 escrevendo (nasm2 $<nome\_do\_arquivo>$) no terminal, que cuida de executar o nasm e o freelink juntamente.

Com o nasm3 (nasm3 $<nome\_do\_arquivo>$) no terminal, executa o nasm o freelink e o .exe gerado. Por fim temos o nasm4(nasm4 $<nome\_do\_arquivo>$) que além de executar tudo roda o debug do executaveu.

3 Resultados

O código a baixo foi comentado de acordo com o entendimento de sua execução, com tudo é possível concluir que o código pega uma string de tamanho 3 lida do teclado e de acordo com a tabela ASCII soma 1 em cada posição da string de forma a gerar uma nova string deslocada, caso sejam digitados números em ASCII a saída sera algo similar a soma de cada digito do numero, exemplo, para uma entrada ”123”a saída gerada sera ”234”.

```
segment code
..start:
; iniciar os registros de segmento DS e SS e o ponteiro de pilha SP
  mov ax,data
  mov ds,ax
  mov ax,stack
  mov ss,ax
  mov sp,stacktop

; iniciar o codigo do programa
  mov bx,three_chars  ; passa o endereco de trhree_chars para o registrador BX

  mov ah,1
  int 21h       ; funcao do dos de entrada de caracter. Retorna em AL

  dec al        ; decrementa o valor em AL
  mov [bx],al   ; Coloca no endere o de bx (referente a primeira possi  o de
                 three_chars) o valor em AL

  inc bx        ; anda para a segunda possicao de three_chars
  int 21h       ;

  dec al        ; decrementa novamente o valor em AL
  mov [bx],al   ; Coloca no endere o de bx (referente a segunda possicao de
                 three_chars) o valor em AL


  inc bx        ; anda para a terceira possicao de three_chars
  int 21h       ;

  dec al        ; decrementa novamente o valor em AL
  mov [bx],al   ; Coloca no endere o de bx (referente a terceira possicao de
                 three_chars) o valor em AL

  mov dx,display_string
  mov ah,9
  int 21h       ; funcao do dos de Impress o de carcateres.

; Terminar o programa e voltar para o sistema operacional
```

```
37    mov ah,4ch
      int 21h
39  segment data
      CR equ 0dh ; 10
41    LF equ 0ah ; 13
      display_string db CR,LF    ; Serve para demonstrar a mensagem
43    three_chars resb 3         ; reserva 3 bits para a mensagem
          db '$'                 ; acresenta o caracter expecial
45  segment stack stack
      resb 256
47  stacktop:
```

O programa abaixo foi criado para multiplicar um vetor do tipo palavra , por outro vetor do tipo palavra, elemento por elemento, colocando o resultado em outro vetor do tipo palavra dupla. O código é feito em 3 etapas em um loop de multiplicação que a cada conjunto de execução multiplica os valores de cada vetor, armazena o resultado no vetor duplo para por fim andar com os indices dos vetores.

```
1   segment code
    ..start:
3   ; iniciar os registros de segmento DS e SS e o ponteiro de pilha SP
        mov     ax,data
5       mov     ds,ax
        mov     ax,stack
7       mov     ss,ax
        mov     sp,stacktop
9       mov     cx,[size]
    ; iniciar o codigo do programa
11
        MULTIPLICACAO:
13          mov bx, [index_vetores]         ; Posi  o inicial dos dois vetores
            mov ax, [Vet1+bx]               ; Multiplicando
15          mov bx, [Vet2+bx]               ; Multiplicador
            mul bx                          ; Executa multiplica  o colocando em Ax e Dx
                                            o resultado
17
            mov bx, [inde_result]           ; Posi  o inicial do resultado
19          mov word [resultado+bx], ax     ; Coloca o resuldado da multiplica  o no
                                            endere o de resultado
            mov word [resultado+bx+2], dx   ;
21
            ; Incrementa os contadores
23          mov bx, [index_vetores]
            add bx, 2                       ; anda para a pr xima possi  o dos vetores
25          mov [index_vetores], bx
```

```
27          mov bx, [inde_result]
            add bx, 4               ; anda para a pr xima possi  o do resultado
                                    incrementa duas vezes pois o word    duplo
29          mov [inde_result], bx

31      loop MULTIPLICACAO
        int 3
33
; Terminar o programa e voltar para o sistema operacional
35  mov         ah,4ch
    int         21h
37
    segment data
39      Vet1                dw    2, 4, 6
        Vet2                dw    5, 7, 25
41      resultado           resw 6

43      size                dw  3
        index_vetores       dw  0
45      inde_result         dw  0

47  segment stack stack
        resb 256
49  stacktop:
```

Na Figura 1 Temos os valores armazenados dos vetores 1 e 2, respectivamente circulados de vermelho. Já na Figura 2 e 3 temos os valores armazenados antes e depois da execução do código, onde inicialmente a memoria esta com valores aleatórios e posteriormente execução a memoria possui os valores dos resultados das multiplicações de cada posição dos vetores 1 e 2.

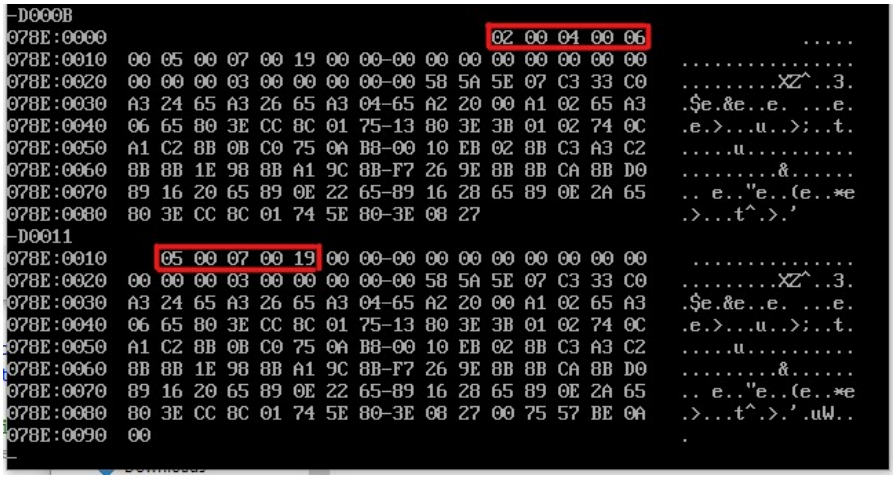

Figura 1: Vetores 1 e 2

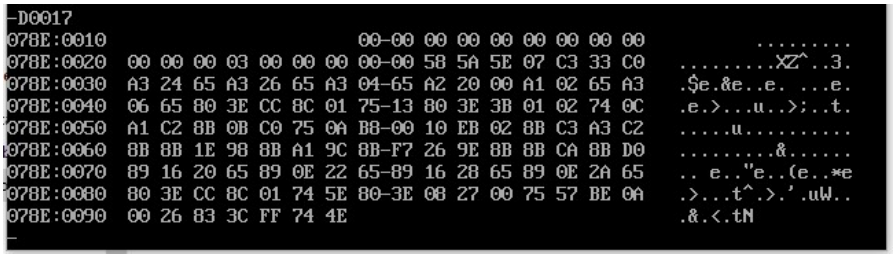

Figura 2: Vetor de Resultado no inicio

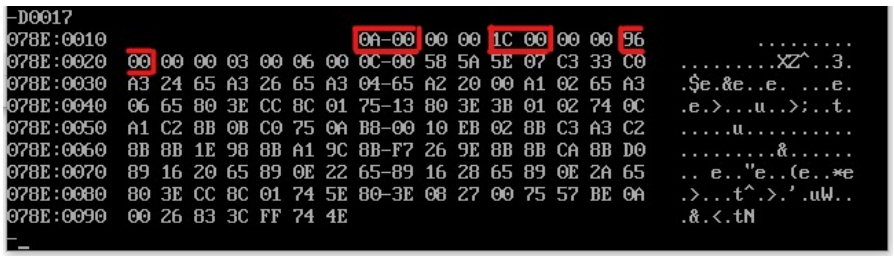

Figura 3: Vetor de Resultado no final

4 Referências

https://ava.ufes.br/pluginfile.php/539919/mod,$esource/content/2/Modelo$

http://marco.uminho.pt/ joao/Computacao2/node35.html