Lista 2 – Programação Assembly (Slide17)

Nome Integrante 1: **Bernardo Menin**

Nome Integrante 2: **Jackson Felipe Magnabosco**

Observações:

- NÃO IMPRIMIR ESTA FOLHA.
- Edite este arquivo e adicione as respostas.
- Códigos–fonte (se necessários) devem ser anexados no final deste arquivo, devidamente identificados com a questão correspondente.
- Novas questões serão adicionadas neste documento. Apenas enviar após todas as 14 questões terem sido respondidas.
- Enviar para danielmenintortelli@gmail.com com o título de "Lista 2"

1) Indique se as instruções abaixo são válidas (**V** – Válido | **I** – Inválido):

```
( V ) mov ax,bx
( I ) mov dx,bl
( V ) mov ecx,edx
( V ) mov si,di
( V ) mov ds,ax
( I ) mov ds,es
( V ) mov al,dh
( I ) mov ax,dh
( V ) mov ip,ax
( I ) mov si,cl
( V ) mov edx,ax
( V ) mov ax,es
```

2) Indique se as instruções abaixo são válidas (**V** – Válido | **I** – Inválido):

```
( V ) mov ax,16
( V ) mov dx,7F65h
( V ) mov ecx,6F23458h
( V ) mov si,-1
( I ) mov ds,1000h
( I ) mov al,100h
( I ) mov 123,dh
( I ) mov ss,ds
( I ) mov 0FABh,ax
( I ) mov si,cl
( V ) mov edx,esi
( V ) mov edx,-2
```

3) Use os seguintes dados para resolver as questões abaixo:

3.a) Escreva uma instrução que incrementa **val2**.

```
.386
.model flat, stdcall
.stack 4096
ExitProcess proto, dwExictCode:dword

.data
val2 DWORD 2d

.code
main proc
    incrementa:
    inc val2
    JMP incrementa
    invoke ExitProcess, 0
    main endp
end main
```

3.b) Escreva uma instrução que subtraia **val3** a partir do registrador **EAX**.

```
.386
.model flat, stdcall
.stack 4096
ExitProcess proto, dwExictCode:dword

.data
val3 DWORD 2d

.code
main proc
    mov eax, 10
    sub eax, val3

    invoke ExitProcess,0
    main endp
end main
```

3.c) Escreva uma instrução que subtraia **val4** de **val2**.

```
.386
.model flat, stdcall
.stack 4096
ExitProcess proto, dwExictCode:dword

.data
val2 DWORD 10d
val4 DWORD 2d

.code
main proc
    mov eax, val2
    sub eax, val4
    mov val2, eax

    invoke ExitProcess,0
    main endp
end main
```

3.d) Se **val2** é incrementado de 1 usando a instrução ADD, quais serão os valores para Carry Flag e Sign Flag?

CF: 0; SF: 0;

3.e) Se **val4** é incrementado de 1 usando a instrução ADD, quais serão os valores para Overflow Flag e Sign Flag?

OF: 1; SF:

4) Indique se as instruções abaixo são válidas (**V** – Válido | **I** – Inválido). Caso a instrução for inválida, anote o erro exibido no Visual Studio ao lado da instrução:

```
a.( V ) add ax, bx
b.( I ) add dx, bl
c.( I ) add ecx, dx
d.( V ) sub si, di
e.( V ) add bx, 90000
f.( I ) sub ds, 1
g.( I ) dec ip
h.( V ) dec edx
i.( V ) add edx, 1000h
j.( V ) sub ah, 126h
k.( V ) sub al, 256
l.( I ) inc ax, 1
```

5) Indique qual será o valor da Carry Flag depois da execução de cada instrução abaixo (**1** – Carry Flag ON | **0** – Carry Flag OFF)

```
a.( 1 ) mov ax, 0FFFFh
        add ax, 1
b.( 0 ) mov bh, 2
        sub bh, 2
c.( 1 ) mov dx, 0
        dec dx
d.( 1 ) mov al, 0DFh
        add al, 32h
e.( 0 ) mov si, 0B9F6h
        sub si, 9874h
f.( 1 ) mov cx, 695Fh
        sub cx, A218h
```

6) Indique qual será o valor da Sign Flag depois da execução de cada instrução abaixo (**1** – Negativo| **0** – Positivo):

```
a.( 1 ) mov ax, 0FFFFh
        sub ax, 1
b.( 1 ) mov bh, 2
        sub bh, 3
c.( 1 ) mov dx, 0
        dec dx
d.( 1 ) mov ax, 7FFEh
        add ax, 22h
e.( 0 ) mov si, 0B9F6h
        sub si, 9874h
f.( 1 ) mov cx, 8000h
        add cx, A69Fh
```

7.1 - Indique se as instruções abaixo são válidas (V – Válido | I – Inválido). Caso a instrução for inválida, anote o erro exibido no Visual Studio ao lado da instrução

a.(**I**) mov ax, byteVal

b.(**V**) mov dx, wordVal

c.(**V**) mov ecx, dwordVal

d.(**I**) mov si, aString

e.(**V**) mov esi, OFFSET aString

f.(**V**) mov al, byteVal

7.2 - Indique se as instruções abaixo são válidas (V – Válido | I – Inválido). Caso a instrução for inválida, anote o erro exibido no Visual Studio ao lado da instrução

a.(**V**) mov eax, OFFSET byteVal

b.(**V**) mov dx, wordVal + 2

c.(**V**) mov ecx, OFFSET dwordVal

d.(**I**) mov si, dwordVal

e.(**V**) mov esi, OFFSET aString + 2

f.(**I**) mov al, OFFSET byteVal + 1

7.3 - Indique o valor hexadecimal movido para o operando de destino após cada instrução MOV

a. mov ax, OFFSET byteVal

ax = 00000000h

b. mov dx, wordVal

dx = 1000h

c. mov ecx, dwordVal

ecx = 12345678h

d. mov esi, OFFSET wordVal

esi = 00000004h

e. mov esi, OFFSET aString

esi = 00000014h

f. mov al, aString + 2

al = 43h

g. Mov edi, OFFSET dwordVal

edi = 000000013h

7.4 - Indique o valor hexadecimal movido para o operando de destino após cada instrução MOV

a. mov eax, OFFSET byteVal + 2

eax = 00000002h

b. mov dx, wordVal + 4

dx = 3000h

c. mov ecx, dwordVal + 4

ecx = 34567890h

d. mov esi, OFFSET wordVal + 4

esi = 00000008h

e. mov esi, OFFSET aString - 1

esi = 00000013h

8 - Indique o valor hexadecimal movido para o operando de destino após cada instrução MOV

a. mov eax, OFFSET wordVal

b. mov dx, wordVal + 4

c. mov ecx, dwordVal + 4

d. mov si, dwordValSiz

e. mov al, byteVal + 4

f. mov ax, dwordVal + 2

g. mov dx, wordVal – 2

h. mov eax, ptrByte

i. mov esi, ptrWord

j. mov edi, OFFSET dwordVal + 2

Eax: 00000006h

Dx: 0002h

Ecx: 00000007h

Si: 0010h

Al: 58h

Ax: Erro

Dx: lixo

Eax: 00FFh

Esi: 00000006h

Edi: 00000006h

9) Preencha os valores dos registradores de acordo com a instrução:

```
mov esi,OFFSET myBytes
mov al,[esi] ; a. AL =
mov al,[esi+3] ; b. AL =
mov esi,OFFSET myWords + 2
mov ax,[esi] ; c. AX =
mov edi,8
mov edx,[myDoubles + edi] ; d. EDX =
mov edx,myDoubles[edi] ; e. EDX =
mov ebx,myPointer
mov eax,[ebx+4] ; f. EAX =
```

AL = 10h

AL = 03bh

AX = 072h

EDX = 0003h

EDX = 0003h

EAX = 0002h

10) Preencha os valores dos registradores de acordo com a instrução:

```
mov esi,OFFSET myBytes
mov ax,[esi] ; a. AX =
mov eax,DWORD PTR myWords ; b. EAX =
```

```
mov esi,myPointer
 mov ax,[esi+2]                    ;                    c. AX =
 mov ax,[esi+6]                    ;                    d. AX =
 mov ax,[esi-4]                    ;                    e. AX =
```

AX = 2010h

EAX = d1372010h

AX = 0000h

AX = 0000h

AX = 0044h

11) Escreva um programa em Assembly que usa um loop para calcular os primeiros sete valores da sequência de Fibonacci, descritos pela fórmula:

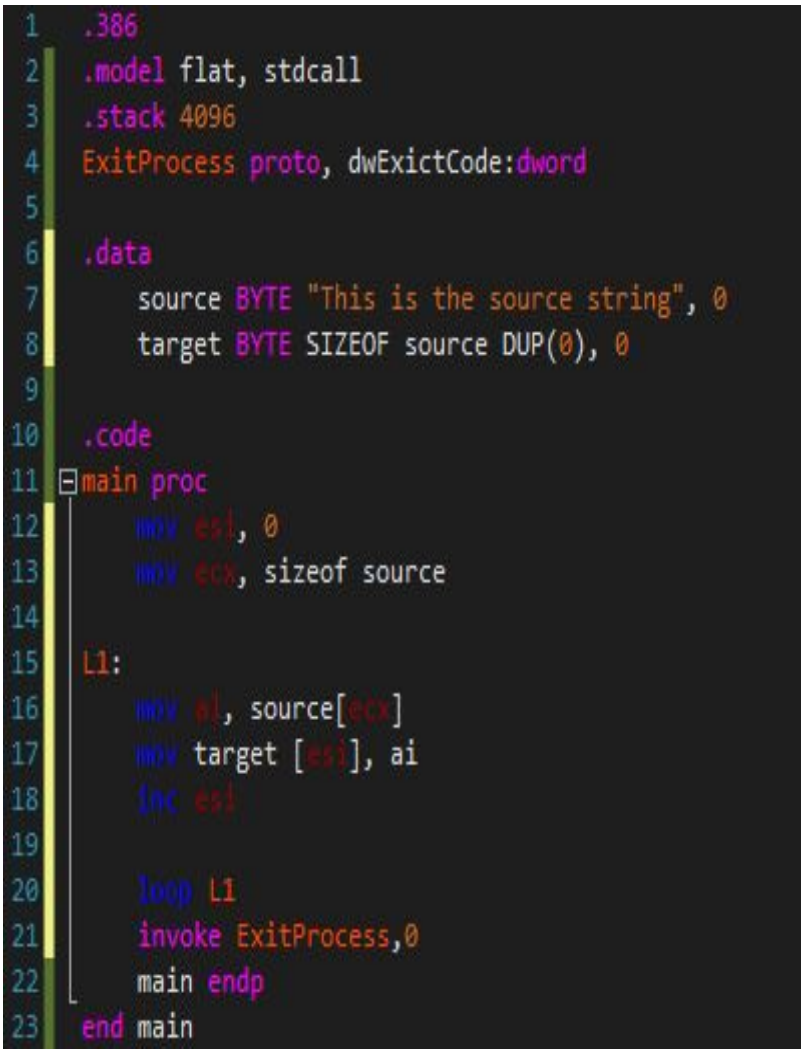
Fib(1) = 1, Fib(2) = 1, Fib(n) = Fib(n – 1) + Fib(n – 2).

```
.386
.model flat, stdcall
.stack 4096
ExitProcess proto, dwExictCode:dword

.data
    fibArray DWORD ?
    sum DWORD ?
    aux1 DWORD ?
    aux2 DWORD ?

.code
main proc
    MOV esi, OFFSET fibArray
    mov eax, 1
    mov [esi], eax
    add esi, 4
    ;add esi, 4

    mov ecx, 5
L1:
    ;add esi, 4
    mov eax, [esi]
    mov aux1, eax
    sub esi, 4
    mov ebx, [esi]
    mov aux2, ebx
    add esi, 8

    mov eax, aux2
    add eax, aux1
    mov [esi], eax
    ;add esi, 4

    loop L1
    mov eax, [esi]
    invoke ExitProcess,0
    main endp
end main
```

12) Escreva um programa em Assembly que usa um loop e endereçamento indireto para copiar uma string de source para target, porém na ordem inversa.

source BYTE "This is the source string",0

target BYTE SIZEOF source DUP('#')



```
.386
.model flat, stdcall
.stack 4096
ExitProcess proto, dwExictCode:dword

.data
    source BYTE "This is the source string", 0
    target BYTE SIZEOF source DUP(0), 0

.code
main proc
    mov esi, 0
    mov ecx, sizeof source

L1:
    mov al, source[ecx]
    mov target [esi], ai
    inc esi

    loop L1
    invoke ExitProcess,0
    main endp
end main
```

13) Escreva um programa em Assembly que usa um loop com endereçamento indireto ou indexado para inverter a ordem dos elementos de um array.

Observação: o programa deve usar apenas um array.

```
.386
.model flat, stdcall
.stack 4096
ExitProcess proto, dwExictCode:dword

.data
    arrayA BYTE 10 dup(?)

.code
main proc

    mov ecx, lengthof arrayA / 2
    mov edx, lengthof  arrayA
    mov esi,  arrayA

l1:
    mov al,  arrayA[esi]
    mov bl,  arrayA[edx]
    mov  arrayA[esi], bl
    mov  arrayA[edx], al
    inc esi
    inc edx

    loop l1

    invoke ExitProcess,0
    main endp
end main
```

14) Escreva um programa em Assembly que usa um loop para copiar todos os elementos de um array do tipo WORD (16-bit) para um array DWORD (32-bit).

```
.386
.model flat, stdcall
.stack 4096
ExitProcess proto, dwExictCode:dword

.data
    txtWord word 10 dup(?)
    txtDWord dword 10 dup(?)

.code
main proc

    mov ecx, lengthof txtWord
    mov esi,  txtDWord
    mov ebx, 0

l1:
    mov ax, txtWord[ebx]
    movzx txtDWord[esi], eax
    inc esi
    inc ebx

    loop l1

    invoke ExitProcess,0
    main endp
end main
```