

## Sistemas Embarcados I – Laboratório 03

**Objetivo:** Estudar as operações básicas de montagem de programas utilizando o montador **NASM** e criando um programa executável com o ligador **FREELINK**.

Os componentes e a estrutura básica de um programa para o montador NASM estão mostrados abaixo:

---

### **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

; codigo do programa
; aqui entram as instruções do programa
:
:
:
; Terminar o programa e voltar para o sistema operacional
    mov     ah,4ch
    int     21h

; definicao das variaveis
segment data
; Aqui entram as definições das variáveis do programa
:
:
:
; definição da pilha com total de 256 bytes
segment stack stack
    resb 256
stacktop:
```

---

**segment code :** Define o início do segmento de código. Aqui entram as instruções do programa.

**..start:** : Este rótulo indica para o NASM onde o programa começa.

Estas instruções são obrigatórias para iniciar o registro DS para apontar para o segmento de dados e o registro SS e o SP para apontarem para a pilha.

```
    mov     ax,data
    mov     ds,ax
    mov     ax,stack
    mov     ss,ax
    mov     sp,stacktop
```

**segment data :** Define o início do segmento de dados. Aqui serão definidas as variáveis do programa.

**segment stack stack :** Define o início do segmento de pilha e associa um nome a este segmento.

**resb 256** : Reserva um determinado número de bytes (256 no caso) para a pilha.  
**stacktop:** : Rótulo que será usado para indicar onde começa a pilha. Pilha vazia quando SP aponta para stacktop.

O arquivo contendo o programa deve ser editado por um editor (EDIT do FREEDOS) no modo não documento e possuir a extensão **.ASM**.

### Definições de variáveis:

O formato geral de uma linha de definição de variáveis é o seguinte:

**<nome da variável> <definição do tipo> <valor inicial> ;<comentário>**

**<nome da variável>**

Qualquer nome de identificador começado por letra e com caracteres de separação.

**<definição do tipo>** Os tipos possíveis são:

DB	define byte	ocupa 1 byte
DW	define Word	ocupa 2 bytes
DD	define double Word	ocupa 4 bytes (real ou inteiro)
DQ	define quad word	ocupa 8 bytes (real ou inteiro)
DT	real precisão estendida	ocupa 10 bytes

Alguns exemplos:

```
b1          db      0x55          ; byte 0x55
tres_bytes  db      0x55,0x56,0x57 ; três bytes sucessivos
            db      'a',0x55      ; caracteres são constantes OK
mensagem    db      'hello',13,10,'$' ; strings constantes
word1       dw      0x1234        ; 0x34 0x12
word2       dw      'a'          ; 0x41 0x00 (é um número)
word3       dw      'ab'         ; 0x41 0x42
word4       dw      'abc'        ; 0x41 0x42 0x43 0x00 (string)
dword1      dd      0x12345678    ; 0x78 0x56 0x34 0x12
real1       dd      1.234567e20   ; constante floating-point
dreal1      dq      1.234567e20   ; double-precision float
ereal1      dt      1.234567e20   ; extended-precision float
vetor1      dw      0,1,2,3,4,5   ; 5 elementos tipo word
vetor_de_quadrados
            dw      0,1,4,9,16
            dw      25,36,49,64,81,100
```

**RESB, RESW, RESD, RESQ e REST** declaram espaço de armazenamento não iniciado. Eles têm um único operando que define o número de bytes, palavras ou o tipo de dado que se quer armazenar.

Exemplo:

```
buffer:     resb     64           ; reserva 64 bytes
wordvar:    resw     1           ; reserve uma word
realarray   resq     10          ; reserve um vetor de 10 reais
```

O prefixo **TIMES** faz com que a instrução seja montada múltiplas vezes.

Exemplo:

```
zerobuf:          times 64 db 0           ; aloca 64 bytes iniciados com 0
```

Por default a base é 10 (decimal). Podemos definir números em outras bases (binária, octal e hexadecimal) como abaixo:

```
mov     ax,100           ; decimal
mov     ax,0a2h          ; hexadecimal
mov     ax,$0a2          ; hexadecimal. O zero é necessário
mov     ax,0xa2          ; hexadecimal
mov     ax,777q          ; octal
mov     ax,10010011b     ; binário
```

**<comentário>** : Em qualquer linha, depois do ; o NASM considera como comentários até o final da linha.

**Atenção:**

- 1- Quando o nome de uma variável é utilizado em uma instrução, o NASM entende que é o offset da variável que vai ser utilizado e não o conteúdo da variável. Para obtermos o conteúdo devemos colocar o nome da variável entre colchetes ([ ]).
- 2- O NASM não guarda o tipo da variável associada ao seu nome. Logo devemos usar as palavras BYTE, WORD, DWORD para informa o número de bytes correspondente à variável.

segment data

```
b1      db      10           ; offset 0
w1      dw      0x1020       ; offset 1
mov     ax,w1              ; coloca 1 em ax
mov     ax,[w1]            ; coloca 0x1020 em AX
mov     word [w1],10       ; coloca 0x0010 em w1
```

E mais:

**EQU:** Define uma constante associando um nome a ela, como nos exemplos:

```
Nove      EQU      9
Baud_rate EQU      (9600/10) + 5
```

**\$** Define o valor do contador de localização (offset) do segmento na posição em que o \$ for encontrado.

```
Array_ex      db      10,20,'ERRO DE TRANSMISAO', 0dh,0ah
Tamanho_array_ex equ   ($-Array_ex)
```

### **Definição das linhas de instruções:**

O formato geral da linha de instruções é o seguinte:

**<rótulo>:**     **<instrução do 8086>** **<operandos se houver>**     **<comentários>**

**<rótulo>:**     um identificador qualquer que vai se referir ao endereço daquela instrução.

**<instrução do 8086>:** Um mnemônico de qualquer instrução válida do 8086, por exemplo MOV, ADD, DIV etc.

## Parte Prática:

1- Montagem de um programa usando o NASM.

a) Usando o editor EDIT do FREDOS, edite as linhas abaixo, com cuidado, em um arquivo OI.ASM.

```
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
    mov     ah,9
    mov     dx,mensagem
    int     21h
; Terminar o programa e voltar para o sistema operacional
    mov     ah,4ch
    int     21h
segment data
CR        equ     0dh
LF        equ     0ah
mensagem  db      'Oi, olha eu aqui',CR,LF,'$'
segment stack stack
    resb 256
stacktop:
```

b) Salve o arquivo e saia do EDIT.

c) A linha de comando do NASM para criar um arquivo .obj( para ser ligado pelo FREELINK e gerar um arquivo .EXE) e um arquivo de listagem é:

**nasm16 -f obj -o %1.obj -l %1.lst %1.asm**

Existe um arquivo **nasm.bat** que monta um programa com extensão **.ASM** e cria um arquivo de listagem com extensão **.LST** e um arquivo objeto com extensão **.OBJ**.

Assim, entre com o comando: **nasm oi**

Verifique se os arquivos **oi.obj** e **oi.lst** foram criados. Corrija os erros que forem indicados se houver algum.

d) Uma vez montado o programa sem erros, ele deverá ser ligado usando o programa FREELINK.

Entre com o comando : **freelink oi**

Verifique a criação do arquivo **oi.exe**.

e) Chame diretamente o programa oi na linha de comandos do FREEDOS e veja o resultado. Usando a apostila de interrupções do DOS, tente entender o programa digitado, verificando o que cada linha faz.

f) Chame agora o programa do debug digitando: **debug oi.exe**

1. Verifique os registros do 8086
2. Com o comando U do debug, desassemble o programa e compare com o conteúdo do arquivo .lst.

g) Digite o programa abaixo tentando entender o seu funcionamento. Acrescente comentários ao texto para melhorar o seu entendimento. Repita todos os passos de montagem, ligação e testes.

```
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
    mov     bx,three_chars
    mov     ah,1
    int     21h           ; função do dos de entrada de caractere. Retorna em AL
    dec     al
    mov     [bx],al
    inc     bx
    int     21h
    dec     al
    mov     [bx],al
    inc     bx
    int     21h
    dec     al
    mov     [bx],al
    mov     dx,display_string
    mov     ah,9
    int     21h
; Terminar o programa e voltar para o sistema operacional
    mov     ah,4ch
    int     21h

segment data
CR      equ     0dh
LF      equ     0ah
display_string  db     CR,LF
three_chars    resb    3
                db     '$'

segment stack stack
    resb 256
stacktop:
```

h) Escreva um programa 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. Edite o programa, definindo os valores dos vetores no arquivo (comandos DW) e reservando um espaço para o vetor resultado. Coloque a terminação de programa do debug (INT 3) no lugar da terminação do DOS. Monte e ligue gerando um .EXE. Teste o programa no debug.