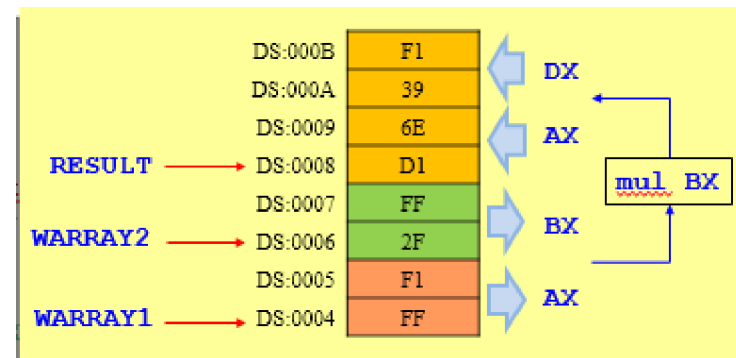
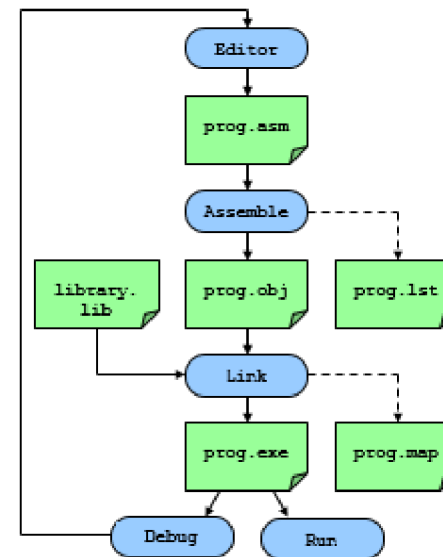


NASM

Dr. Jorge Leonid Aching Samatelo
jlasam001@gmail.com



Índice

- ☐ NASM
- ☐ Definição de dados
- ☐ Laboratório

NASM

NASM

Assembly e Assembler

□ Assembly

- ou **LINGUAGEM DE MONTAGEM** é uma linguagem de programação de baixo nível com a qual escrevemos programas usando uma notação humanamente legível para o código de máquina.

Linguagem de máquina	Linguagem de montagem
BA0B01	mov dx,msg
B409	mov ah,9
48656C6C6F2C20576F	msg db 'Hello,World! ',0Dh,0Ah,'\$'

O tamanho das instruções são variáveis

□ Assembler

- é o **PROGRAMA QUE TRANSFORMA** o código de linguagem de montagem para linguagem de máquina compreensível para um determinado processador.
 - ❖ **NASM (Netwide Assembler)** é um *assembler* livre, para arquitetura x86 que permite programar em linguagem de máquina usando **mnemônicos**.

NASM

Ciclo de montagem (*assemble*), linkagem (*link*) e depuração (*debug*)

❑ **Edição:** programa *notepad++*

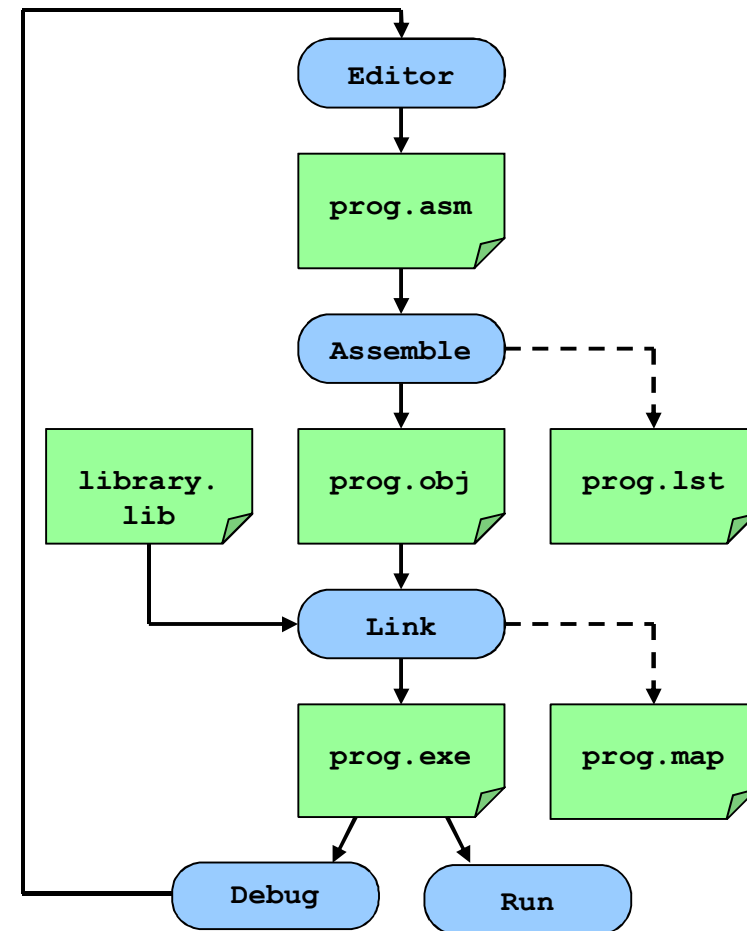
- O arquivo contendo o programa deve ser editado por um editor (*notepad++*) e possuir a extensão *.ASM*.

❑ **Montagem:** programa *NASM16.exe*

- A partir do arquivo *.ASM* cria um arquivo *.OBJ* (representação em linguagem de máquina) e um arquivo de listagem *.LST* (mostra o trabalho do montador).

❑ **Linkagem:** programa *freelink.exe*

- Copia as sub-rotinas necessárias da *library.lib* para o arquivo *.OBJ*. e gera um arquivo executável (*.EXE*) o qual pode ser executado no *prompt* do *DOSbox*.



NASM

Estrutura de um programa para NASM

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

```
segment code
    ..start:
    ; -----INICIAR DS, SS e SP-----
    ; ds<-data
    mov ax,data
    mov ds,ax
    ; ss<-stack
    mov ax,stack
    mov ss,ax
    ; sp<-stacktop
    mov sp,stacktop
    ; -----CODIGO DO PROGRAMA-----

    ; -----SAIDA DO PROGRAMA-----
    mov ah,0x4c
    int 0x21
segment data
    ; -----DEF. VAR,CONST E ALOCACAO-----

segment stack stack
    ; -----DEF. PILHA-----
    resb 256 ; definição da pilha com total de 256 bytes
stacktop:
```

NASM

Estrutura de um programa para NASM

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

```
segment code
    ..start:
    ; -----
    ; ds<-data
    mov ax,data
    mov ds,ax
    ; ss<-stack
    mov ax,stack
    mov ss,ax
    ; sp<-stacktop
    mov sp,stacktop
    ; -----CODIGO DO PROGRAMA-----

    ; -----SAIDA DO PROGRAMA-----
    mov ah,0x4c
    int 0x21
segment data
    ; -----DEF. VAR,CONST E ALOCACAO-----

segment stack stack
    ; -----DEF. PILHA-----
    resb 256 ; definição da pilha com total de 256 bytes
stacktop:
```

Define o **INÍCIO** do **segmento de código**. Aqui entram as instruções do programa.

NASM

Estrutura de um programa para NASM

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

```
segment code
    ..start:
    ; -----
    ; ds<-data
    mov ax,data
    mov ds,ax
    ; ss<-stack
    mov ax,stack
    mov ss,ax
    ; sp<-stacktop
    mov sp,stacktop
    ; -----CODIGO DO PROGRAMA-----

    ; -----SAIDA DO PROGRAMA-----
    mov ah,0X4c
    int 0x21
segment data
    ; -----DEF. VAR,CONST E ALOCACAO-----

segment stack stack
    ; -----DEF. PILHA-----
    resb 256 ; definição da pilha com total de 256 bytes
stacktop:
```

Este rótulo indica para o NASM onde o programa começa.

NASM

Estrutura de um programa para NASM

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

```
segment code
    ..start:
    ; -----INICIAR DS, SS e SP-----
    ; ds<-data
    mov ax,data
    mov ds,ax
    ; ss<-stack
    mov ax,stack
    mov ss,ax
    ; sp<-stacktop
    mov sp,stacktop
    ; -----CODIGO DO PROGRAMA-----

    ; -----SAIDA DO PROGRAMA-----
    mov ah,0X4c
    int 0x21
segment data
    ; -----DEF. VAR,CONST E ALOCACAO-----

segment stack stack
    ; -----DEF. PILHA-----
    resb 256 ; definição da pilha com total de 256 bytes
stacktop:
```

Instruções obrigatórias para INICIAR o registrador DS para apontar para o segmento de dados e o registradores SS e SP para apontarem para a pilha.

NASM

Estrutura de um programa para NASM

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

```
segment code
    ..start:
    ; -----INICIAR DS, SS e SP-----
    ; ds<-data
    mov ax,data
    mov ds,ax
    ; ss<-stack
    mov ax,stack
    mov ss,ax
    ; sp<-stacktop
    mov sp,stacktop
    ; -----CODIGO DO PROGRAMA-----

    ; -----SAIDA DO PROGRAMA-----
    mov ah,0x4c
    int 0x21
segment data
    ; -----DEF. VA
segment stack stack
    ; -----DEF. PILHA-----
    resb 256 ; definição da pilha com total de 256 bytes
stacktop:
```

Define o **INÍCIO** do **segmento de dados**. Aqui serão definidas as variáveis do programa.

NASM

Estrutura de um programa para NASM

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

```
segment code
    ..start:
    ; -----INICIAR DS, SS e SP-----
    ; ds<-data
    mov ax,data
    mov ds,ax
    ; ss<-stack
    mov ax,stack
    mov ss,ax
    ; sp<-stacktop
    mov sp,stacktop
    ; -----CODIGO DO PROGRAMA-----

    ; -----SAIDA DO PROGRAMA-----
    mov ah,0x4c
    int 0x21
segment data
    ; -----DEF. VAR,CONST E ALOCACAO-----

segment stack stack
    ; -----
    resb 256 ; definição da pilha com total de 256 bytes
stacktop:
```

Define o **INÍCIO** do **segmento de pilha** e associa um nome a este segmento.

NASM

Estrutura de um programa para NASM

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

```
segment code
    ..start:
    ; -----INICIAR DS, SS e SP-----
    ; ds<-data
    mov ax,data
    mov ds,ax
    ; ss<-stack
    mov ax,stack
    mov ss,ax
    ; sp<-stacktop
    mov sp,stacktop
    ; -----CODIGO DO PROGRAMA-----

    ; -----SAIDA DO PROGRAMA-----
    mov ah,0X4c
    int 0x21
segment data
    ; -----DEF. VAR,CONST E ALOCACAO-----

segment stack stack
    ; -----
    resb 256 ; definição da pilha
stacktop:
```

Reserva um determinado número de bytes (256 no caso) para a pilha.

NASM

Estrutura de um programa para NASM

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

```
segment code
    ..start:
    ; -----INICIAR DS, SS e SP-----
    ; ds<-data
    mov ax,data
    mov ds,ax
    ; ss<-stack
    mov ax,stack
    mov ss,ax
    ; sp<-stacktop
    mov sp,stacktop
    ; -----CODIGO DO PROGRAMA-----

    ; -----SAIDA DO PROGRAMA-----
    mov ah,0X4c
    int 0x21
segment data
    ; -----DEF. VAR.CONST E ALOCACAO-----

segment stack stack
    ; -----
    resb 256 ; definição da pilha
stacktop:
```

Rótulo que será usado para indicar onde **começa a pilha**. Pilha vazia quando o registrador SP aponta para *stacktop*.

Definição de dados

Definição de dados

Declaração de Números

❑ Binários:

1110101b ou 1110101B

❑ Decimais:

64223

- 1110101 é considerado decimal (ausência do B)
- -2184 (número negativo)

❑ Octal:

777q

❑ Hexadecimais:

64223h, 64223H, 0x64223, \$0a2

- 0FFFFh começa com um decimal e termina com h
- \$0a2 o zero é necessário.

No programa `NASM16.exe` a base por *default* é decimal ao contrario do programa `debug.exe` cuja base pro *default* é hexadecimal.

Definição de dados

Declaração de Números

❑ Exemplos de números ilegais

➤ **Errado:** 1,234

▪ caractere estranho (vírgula)

➤ **Correto:** 1.234

➤ **Errado:** FFFFh

❖ não começa por número de 0 a 9 difícil distinguir do nome de uma variável.

➤ **Correto:** 0xFFFF, 0xFFFFh

➤ **Errado:** 1B4D

❖ não tem um indicativo que é um hexadecimal.

➤ **Correto:** 0x1B4D

Definição de dados

Caracteres ASCII

- ❑ Definidos por aspas simples ou aspas duplas.

- ❑ Exemplo:

"Bla" ou 'Bla'

Definição de dados

Definição de variáveis

❑ Características:

- Possui um tipo de dado.
- Recebe um endereço de memória

❑ Usa-se **PSEUDO-INSTRUÇÕES** para definir o tipo da dado.

- Variam de acordo com o tamanho da memória alocada

Pseudo-Instruções	Descrição
DB	Define um byte (8 bits)
DW	Define um word (16 bits, 2 BYTES consecutivos)
DD	Define um doubleword (2 WORDS, 4 BYTES consecutivos)
DQ	Define um quadword (4 WORDS, 8 BYTES consecutivos)
DT	define ten bytes (10 BYTES consecutivos)

Definição de dados

Definição de variáveis

❑ Estrutura

<Nome> <Pseudo-instrução> <valor>

❑ Exemplo:

Teste	DB	0x0	;equivale a 00h
Bli	DB	0x10	
Foo	DB	?	;não inicializa
Ex	DB	0x0150	;Erro!

Foo → DS:0002

Bli → DS:0001

Teste → DS:0000

10
00

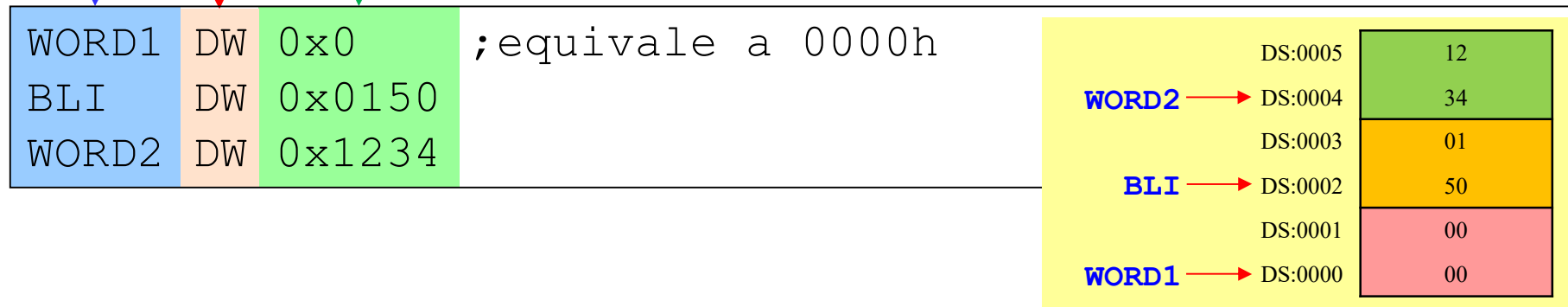
Definição de dados

Definição de variáveis

❑ Estrutura

<Nome> <Pseudo-instrução> <valor>

❑ Exemplo:

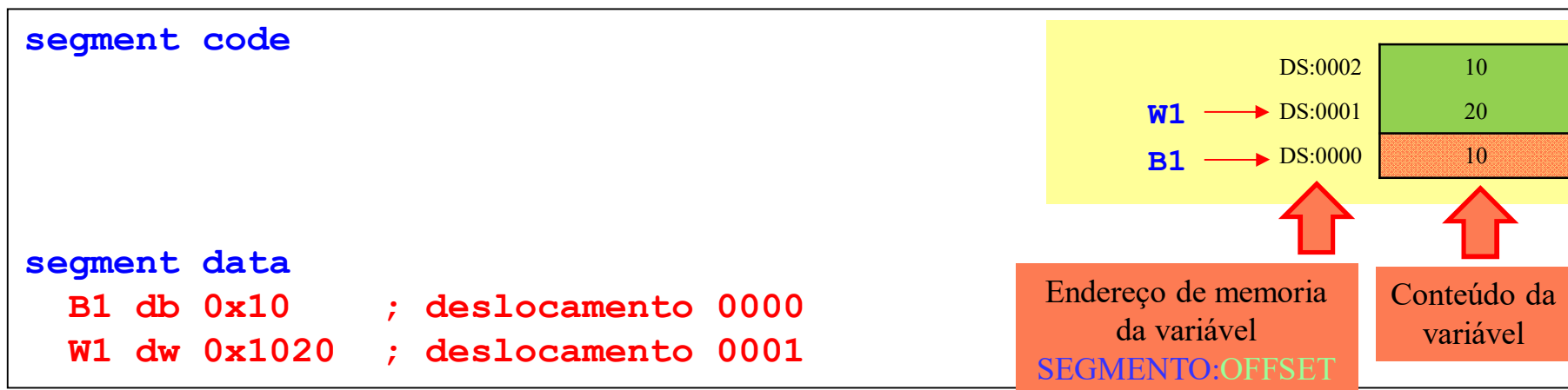


Definição de dados

Definição de Constantes

❑ Atenção

- Quando o nome de uma variável é utilizado em uma instrução, o NASM entende que **é o endereço de memória da variável (deslocamento) que vai ser utilizado e não o conteúdo da variável.**
- Para obtermos o conteúdo devemos colocar o nome da variável entre colchetes (**[]**).



Definição de dados

Definição de Constantes

❑ Atenção

- Quando o nome de uma variável é utilizado em uma instrução, o NASM entende que **é o endereço de memória da variável (deslocamento) que vai ser utilizado e não o conteúdo da variável.**
- Para obtermos o conteúdo devemos colocar o nome da variável entre colchetes ([]).

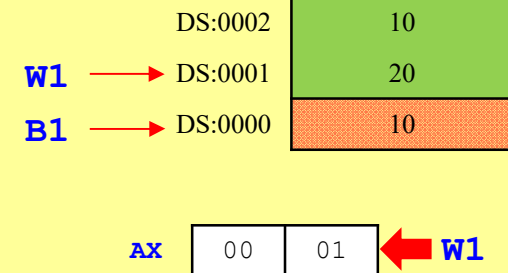
```
segment code
```

```
▶ mov ax,w1      ; coloca 0001 em ax  
  mov ax,[w1]    ; coloca 0x1020 em AX  
  mov word [w1],10 ; coloca 0x0010 em w1
```

```
...
```

```
segment data
```

```
  B1 db 0x10      ; deslocamento 0000  
  W1 dw 0x1020    ; deslocamento 0001
```



Guardo em AX o endereço de memória (deslocamento) da variável W1.

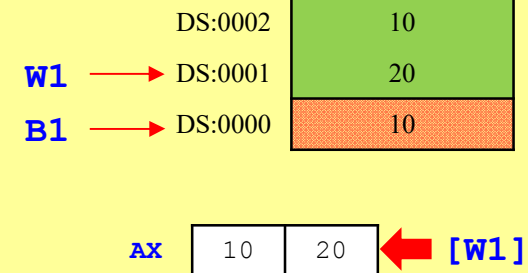
Definição de dados

Definição de Constantes

❑ Atenção

- Quando o nome de uma variável é utilizado em uma instrução, o NASM entende que **é o endereço de memória da variável (deslocamento) que vai ser utilizado e não o conteúdo da variável.**
- Para obtermos o conteúdo devemos colocar o nome da variável entre colchetes ([]).

```
segment code
    mov ax,w1      ; coloca 0001 em ax
    ➤mov ax,[w1]    ; coloca 0x1020 em AX
    mov word [w1],10 ; coloca 0x0010 em w1
    ...
segment data
    B1 db 0x10      ; deslocamento 0000
    W1 dw 0x1020    ; deslocamento 0001
```



Guardo em AX o conteúdo da variável W1.

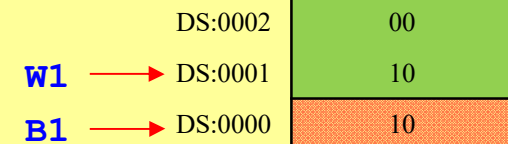
Definição de dados

Definição de Constantes

❑ Atenção

- Quando o nome de uma variável é utilizado em uma instrução, o NASM entende que **é o endereço de memória da variável (deslocamento) que vai ser utilizado e não o conteúdo da variável.**
- Para obtermos o conteúdo devemos colocar o nome da variável entre colchetes ([]).

```
segment code
    mov ax,w1      ; coloca 0001 em ax
    mov ax,[w1]    ; coloca 0x1020 em AX
    mov word [w1],10 ; coloca 0x0010 em w1
    ...
segment data
    B1 db 0x10      ; deslocamento 0000
    W1 dw 0x1020    ; deslocamento 0001
```



Guardo no endereço de memória (deslocamento) da variável W1 o valor 0x1020

➤ 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.

Definição de dados

Definição de variáveis

□ Arrays

- Conjunto de BYTES ou WORDS em sequência na memória.

□ Estrutura

<Nome> <Pseudo-instrução> <valor>

□ Exemplo: *Arrays* numéricos

BARRAY	DB	0x10, 0x20, 0x30, 0x01
WARRAY	DW	0x1000, 0x123, 0x0, 0xFFFF

	DS:000B	FF
	DS:000A	FF
	DS:0009	00
	DS:0008	00
	DS:0007	01
	DS:0006	23
	DS:0005	10
WARRAY →	DS:0004	00
	DS:0003	01
	DS:0002	30
	DS:0001	20
BARRAY →	DS:0000	10

Definição de dados

Definição de variáveis

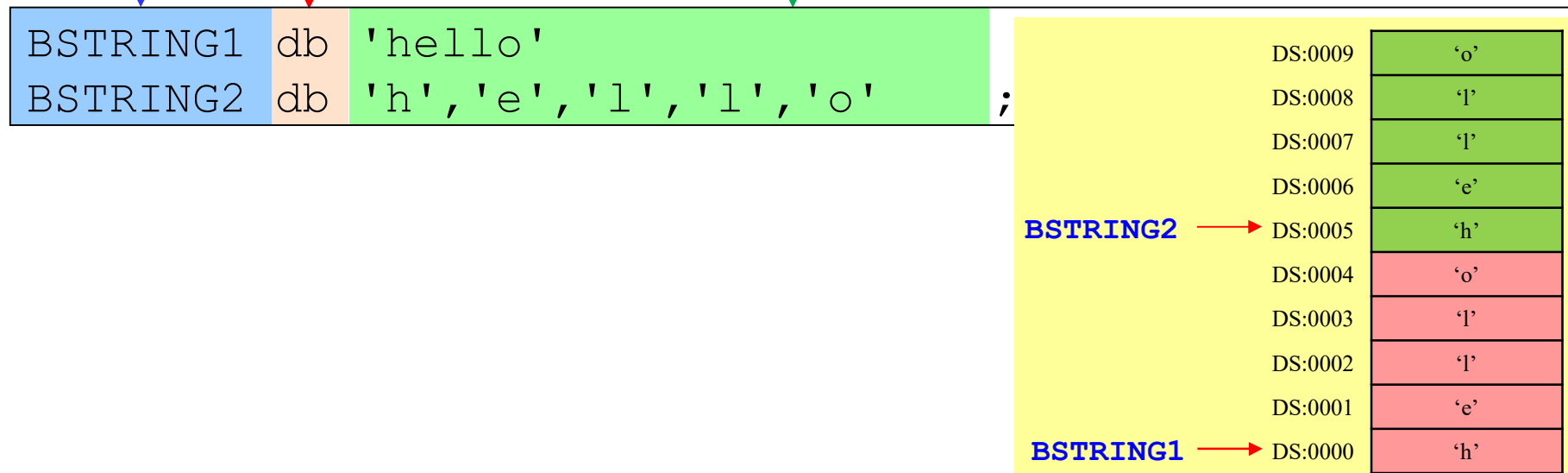
□ Arrays

- Conjunto de BYTES ou WORDS em sequência na memória.

□ Estrutura

<Nome> <Pseudo-instrução> <valor>

□ Exemplo: *Arrays* de caracteres (*string*)



Definição de dados

Definição de variáveis

□ Arrays

- Conjunto de BYTES ou WORDS em sequência na memória.

□ Estrutura

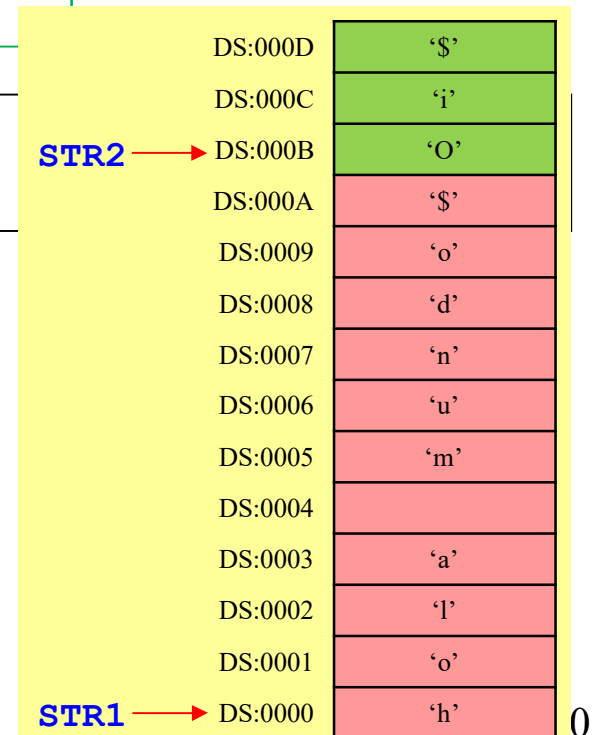
<Nome> <Pseudo-instrução> <valor>

□ Exemplo: *Arrays* de caracteres (*string*)

```
STR1 DB "Hola mundo", "$"  
STR2 DB 'Oi', '$'
```

- Quando uma *string* é impressa usando a interrupção `int 0x21`, o `$` indica o final de linha.

```
segment code  
mov dx, STR1  
mov ah, 09  
int 21h
```



Definição de dados

Exemplo: Hola mundo

```
segment code
```

HolaMd.asm

```
...
```

```
; -----CODIGO DO PROGRAMA-----
```

```
mov dx, STR1
```

```
mov ah, 09
```

```
int 21h
```

```
mov dx, STR2
```

```
mov ah, 09
```

```
int 21h
```

```
; -----SAIDA DO PROGRAMA-----
```

```
mov ah, 0X4c
```

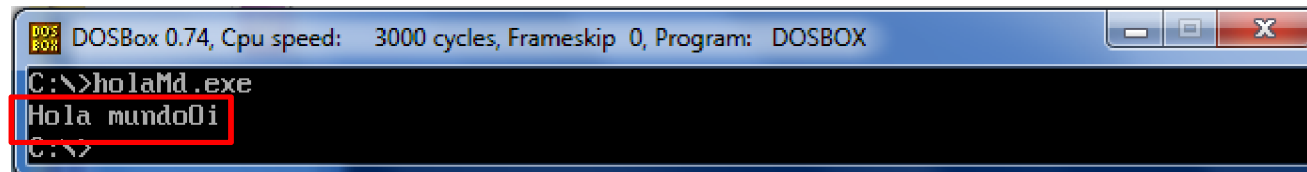
```
int 0X21
```

```
segment data
```

```
STR1 DB "Hola mundo", "$"
```

```
STR2 DB 'Oi', '$'
```

```
...
```



Definição de dados

Definição de Constantes

❑ Para definir uma constante utiliza-se da **pseudo-instrução EQU**.

❑ Estrutura

<nome> EQU <valor_constante>

❑ Exemplo:

LF	EQU	0x0A ;character <i>Line Feed</i> como LF
CR	EQU	0x0D ;character <i>Carriage return</i> como CR
STR1	DB	"Hola mundo", LF, CR, "\$"
STR2	DB	'Oi', LF, CR, '\$'



LF + CR desloca o cursor à primeira posição de uma linha.

Enter

DOS é um sistema baseado em ASCII, portanto, uma **nova linha** é codificada como a combinação de LF (*salto de linha*) e CR (*retorno de carro*)



Definição de dados

Exemplo: Hola mundo

segment code

HolaMd.asm

...

; -----CODIGO DO PROGRAMA-----

mov dx, STR1

mov ah, 09

int 21h

mov dx, STR2

mov ah, 09

int 21h

; -----SAIDA DO PROGRAMA-----

mov ah, 0X4c

int 0X21

segment data

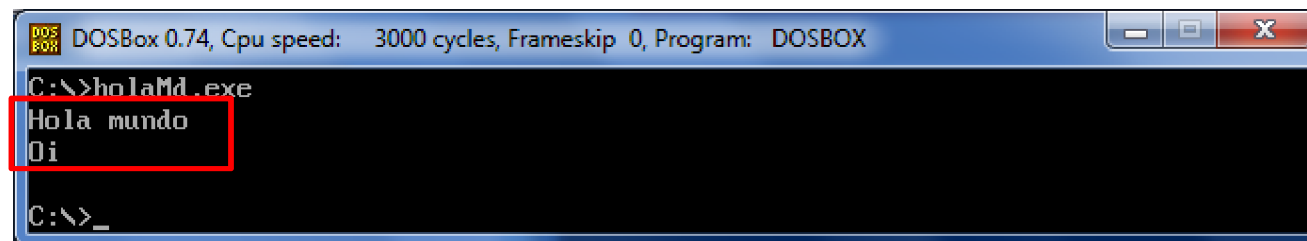
LF EQU 0x0A ;caracter *Line Feed* como LF

CR EQU 0x0D ;caracter *Carriage return* como CR

STR1 DB "Hola mundo", LF, CR, "\$"

STR2 DB 'Oi', LF, CR, '\$'

...



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>holaMd.exe
Hola mundo
Oi
C:\>_
```

Definição de dados

Definição de espaço

❑ Características:

- 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.

❑ Usa-se **PSEUDO-INSTRUÇÕES** para definir o **espaço de armazenamento não iniciado**.

- Variam de acordo com o tamanho da memória alocada

Pseudo-Instruções	Descrição
RESB	Declara um byte (8 bits)
RESW	Declara um word (16 bits, 2 BYTES consecutivos)
RESQ	Declara um quadword (4 WORDS, 8 BYTES consecutivos)
REST	Declara um ten bytes (10 BYTES consecutivos)

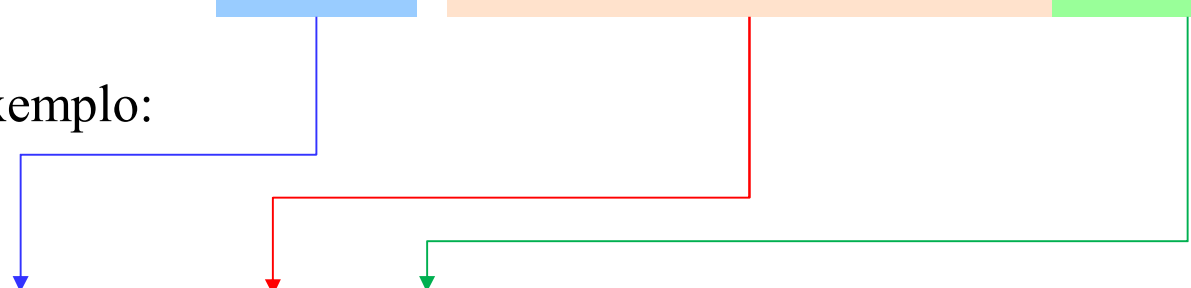
Definição de dados

Definição de espaço

❑ Estrutura

<Nome>: **<Pseudo-instrução>** **<valor>**

❑ Exemplo:



buffer:	resb	64	; reserva 64 bytes
wordvar:	resw	1	; reserve um word
Realarray:	resw	10	; reserve um vetor de 10 words

Laboratório

Laboratório

1. Montagem de um programa usando o NASM.

❑ Usando o editor *notepad++*, escreva as seguintes linhas de código.

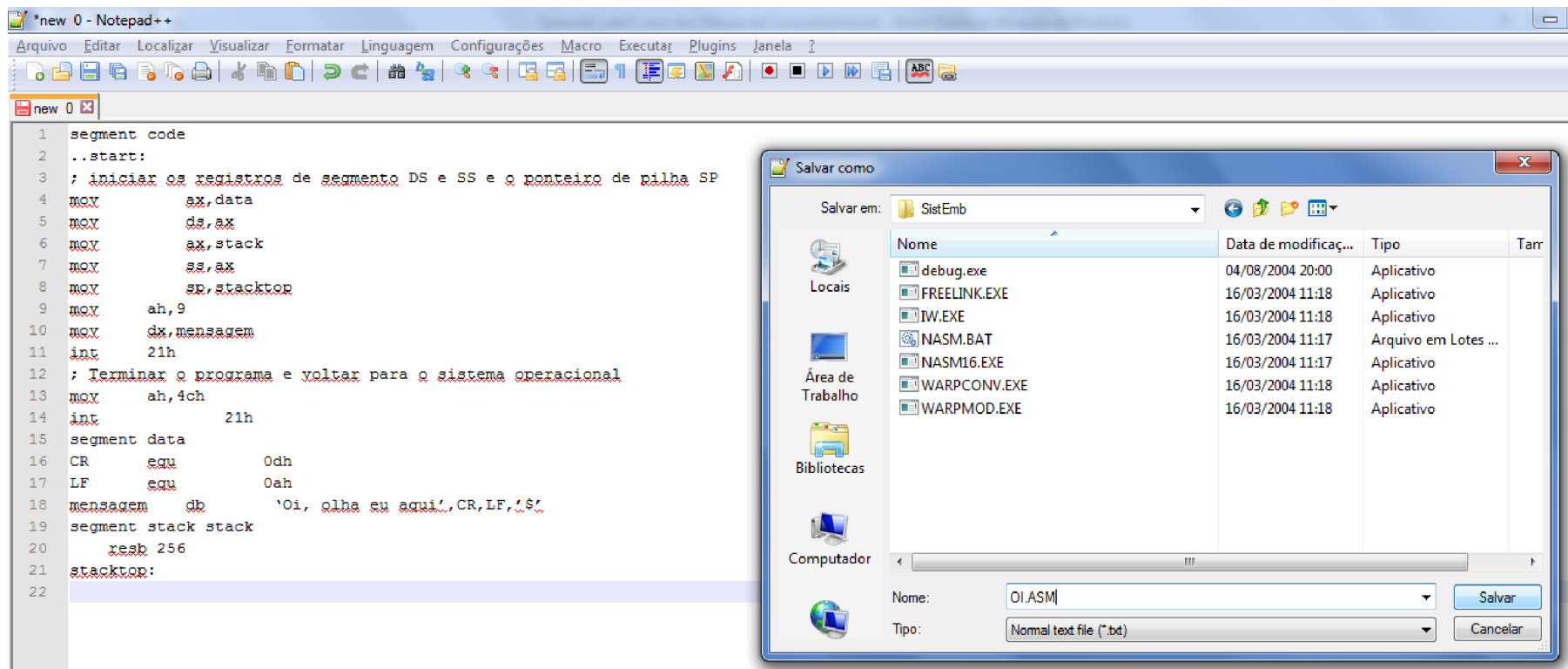
```
segment code
..start:
; iniciar os registros de segmento DS e SS e o ponteiro de pilha SP
    mov ax,data
    mov ds,ax
    mov ax,stack
    mov ss,ax
    mov sp,stacktop
    mov ah,9
    mov dx,mensagem
    int 21h
; Terminar o programa e voltar para o sistema operacional
    mov ah,4ch
    int 21h
segment data
    CR equ 0x0d
    LF equ 0x0a
    mensagem db 'Oi, olha eu aqui',CR,LF,'$'
segment stack stack
    resb 256
stacktop:
```

Laboratório

1. Montagem de um programa usando o NASM.

- ❑ Salve o arquivo com o nome **OI.ASM** na pasta da disciplina (.../acts/sis_embarcados) e saia do editor.

Os nomes dos arquivos devem ter no máximo oito caracteres (sem considerar a extensão ASM)



Laboratório

1. Montagem de um programa usando o NASM.

- ❑ 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
```

- ❑ Mas, para facilitar o processo de montagem, 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`.
- ❑ Para executar o arquivo `nasm.bat`, só é necessário escrever o seguinte comando no *DOSbox*:

```
nasm nome_arquivo ↵
```

- ❑ Para o caso do exemplo:

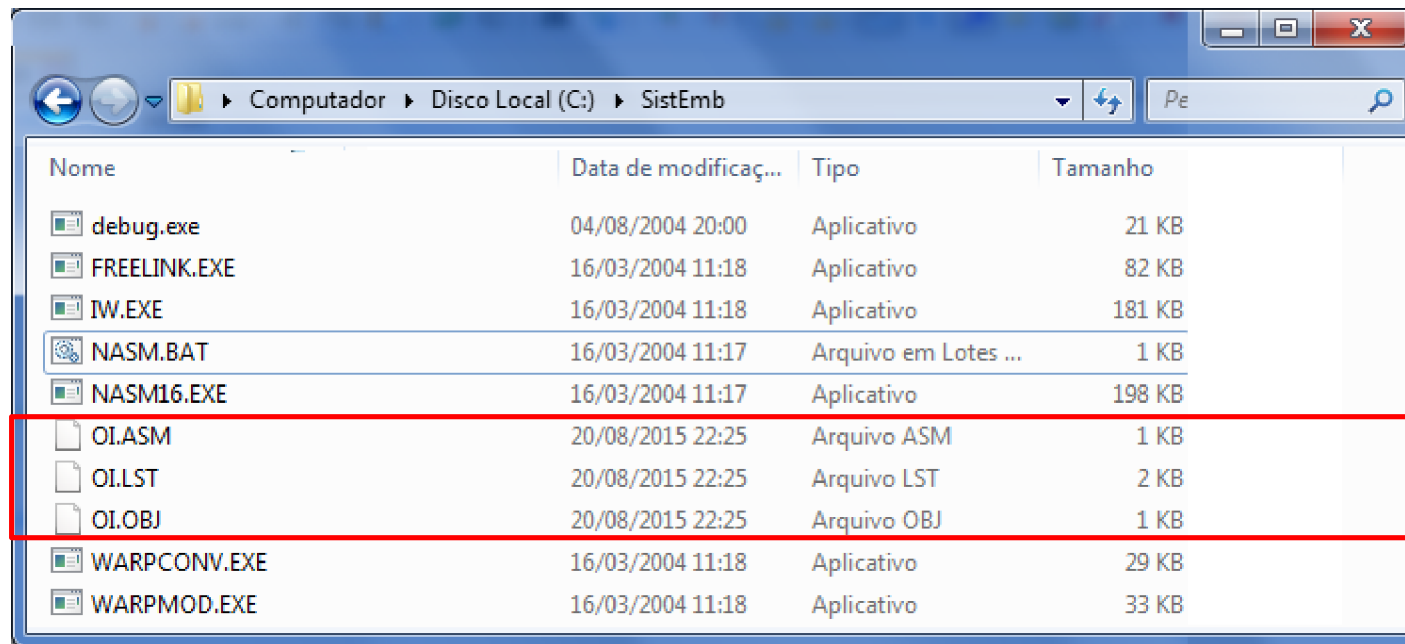
```
nasm oi ↵
```

❑ *Nota*

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

Laboratório

1. Montagem de um programa usando o NASM.



Laboratório

1. Montagem de um programa usando o NASM.

- ❑ Uma vez montado o programa sem erros, ele deverá ser ligado usando o programa FREELINK.
- ❑ Para chamar ao programa `freelink`, só é necessário escrever o seguinte comando no *DOSbox*:

`freelink nome_arquivo` ↵

- ❑ Para o caso do exemplo:

`freelink oi`

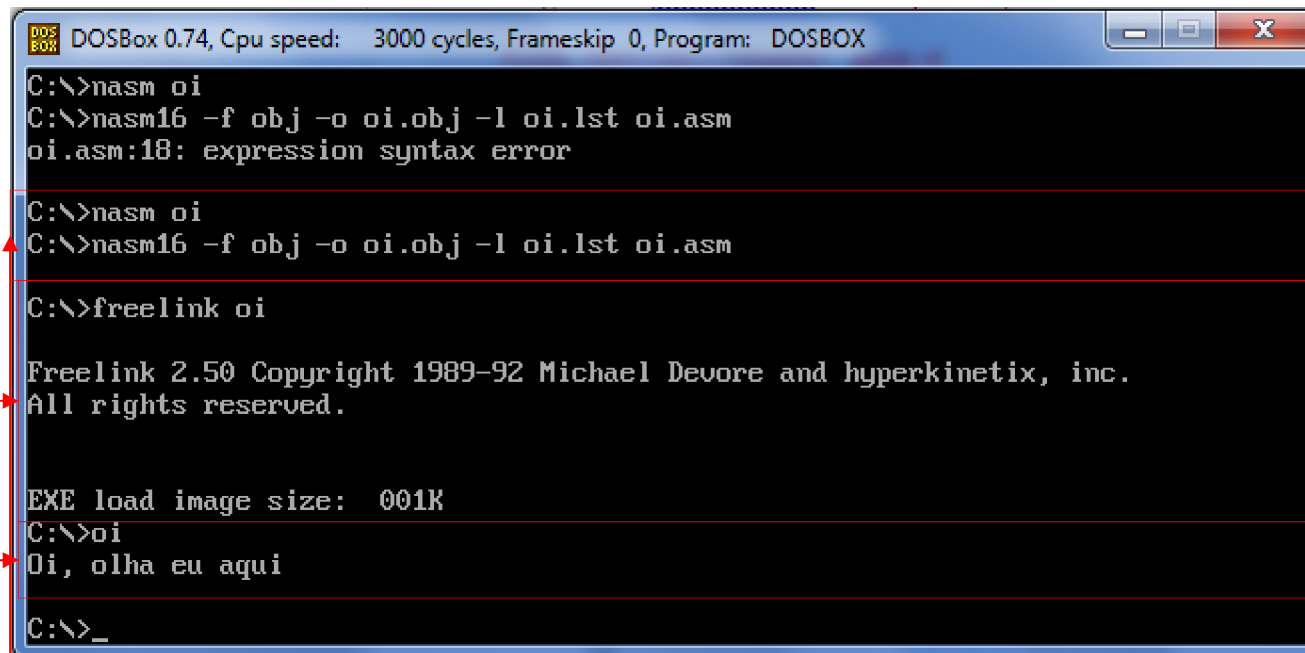
- ❑ *Nota*

- Verifique a criação do arquivo `oi.exe`.
-

Laboratório

1. Montagem de um programa usando o NASM.

- ❑ Chame diretamente o programa `oi` na linha de comandos do *DosBox* e veja o resultado. Tente entender o programa digitado, verificando o que cada linha faz.



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>nasm oi
C:\>nasm16 -f obj -o oi.obj -l oi.lst oi.asm
oi.asm:18: expression syntax error

C:\>nasm oi
C:\>nasm16 -f obj -o oi.obj -l oi.lst oi.asm

C:\>freelink oi

Freelink 2.50 Copyright 1989-92 Michael Devore and hyperkinetix, inc.
All rights reserved.

EXE load image size: 001K
C:\>oi
Oi, olha eu aqui

C:\>_
```

Chamando o arquivo bat `nasm`

Chamando o programa `freelink`

Chamando o executável do programa desenvolvido `oi.exe`

Laboratório

1. Montagem de um programa usando o NASM.

- ❑ Chame agora o programa do debug digitando: `debug oi.exe`
 - Com o comando **R** do debug, verifique os registros do 8086
 - Com o comando **U** do debug, desassemble o programa e compare com o conteúdo do arquivo `.lst`.

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\>debug oi.exe
-r
AX=FFFF BX=0000 CX=002B DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=076D CS=076A IP=0000 NV UP EI PL NZ NA PO NC
076A:0000 B86B07      MOV     AX,076B
-
-ucs:000
076A:0000 B86B07      MOV     AX,076B
076A:0003 8ED8          MOV     DS,AX
076A:0005 B86D07      MOV     AX,076D
076A:0008 8ED0          MOV     SS,AX
076A:000A BC0001      MOV     SP,0100
076A:000D B409          MOV     AH,09
076A:000F BA0800      MOV     DX,0008
076A:0012 CD21          INT     21
076A:0014 B44C          MOV     AH,4C
076A:0016 CD21          INT     21
076A:0018 4F            DEC     DI
076A:0019 69            DB      69
076A:001A 2C20          SUB     AL,20
076A:001C 6F            DB      6F
076A:001D 6C            DB      6C
076A:001E 68            DB      68
076A:001F 61            DB      61
-
```


Laboratório

1. Montagem de um programa usando o NASM.

Resumo (*Linux*)



❑ *Passos:*

- Comando para montar a pasta de trabalho no DOSBox (.../acts/sis_embarcados)

```
mount c /acts/sis_embarcados ↵  
c: ↵
```

- Comando para gerar o arquivo .OBJ: `nasm nome_arquivo` ↵

- Comando para gerar o arquivo .EXE: `freelink nome_arquivo` ↵

- Depurar o arquivo .EXE: `debug nome_arquivo.exe` ↵

❖ Para depurar linha por linha é o comando `t`.

- Executar o .EXE: `nome_arquivo.exe` ↵

Laboratório

1. Montagem de um programa usando o NASM.

Resumo (*Windows*)



❑ *Passos:*

- Comando para montar a pasta de trabalho no DOSBox (c:\sistemb1\frsm)

```
mount c c:\sistemb1\frsm\ ↵  
c: ↵
```

- Comando para gerar o arquivo .OBJ: `nasm nome_arquivo ↵`

- Comando para gerar o arquivo .EXE: `freelink nome_arquivo ↵`

- Depurar o arquivo .EXE: `debug nome_arquivo.exe ↵`

❖ Para depurar linha por linha é o comando `t`.

- Executar o .EXE: `nome_arquivo.exe ↵`

Laboratório

2. Estudo de um programa em assembly.

- ☐ Estude o programa `ex00201.asm`. Acrescente comentários ao texto para melhorar o seu entendimento. Repita todos os passos de montagem, ligação e testes.
-

Laboratório

2. Estudo de um programa em assembly. Dica 1,

segment code

ex00201.asm

...

; -----CODIGO DO PROGRAMA-----

mov bx,strIn

mov ah,1

int 21h

mov byte[bx+2],al

int 21h

mov byte[bx+3],al

int 21h

mov byte[bx+4],al

mov dx, strIn

mov ah, 0x09

int 21h

; -----SAIDA DO PROGRAMA-----

mov ah,0X4c

int 0X21

segment data

; -----DEF. VAR, CONST E ALOCACAO-----

LF EQU 0x0A ;caracter Line Feed como LF

CR EQU 0x0D ;caracter Carriage return como CR

strIn DB CR,LF

RESB 3

DB '\$'

Laboratório

2. Estudo de um programa em assembly. Dica 1,

segment code

...

; -----CODIGO DO PROGRAMA-----

mov bx,strIn

mov ah,1

int 21h

mov byte[bx+2],al

int 21h

mov byte[bx+3],al

int 21h

mov byte[bx+4],al

mov dx, strIn

mov ah, 0x09

int 21h

; -----SAIDA DO PROGRAMA-----

mov ah,0X4c

int 0X21

segment data

; -----DEF. VAR,CONST E ALOCACAO-----

LF EQU 0x0A ;caracter Line Feed como LF

CR EQU 0x0D ;caracter Carriage return como CR

strIn DB CR,LF

RESB 3

DB '\$'



Laboratório

2. Estudo de um programa em assembly. Dica 1,

segment code

...

; -----CODIGO DO PROGRAMA-----

mov bx, strIn

mov ah, 1

int 21h

mov byte[bx+2], al

int 21h

mov byte[bx+3], al

int 21h

mov byte[bx+4], al

mov dx, strIn

mov ah, 0x09

int 21h

; -----SAIDA DO PROGRAMA-----

mov ah, 0x4c

int 0x21

segment data

; -----DEF. VAR, CONST E ALOCACAO-----

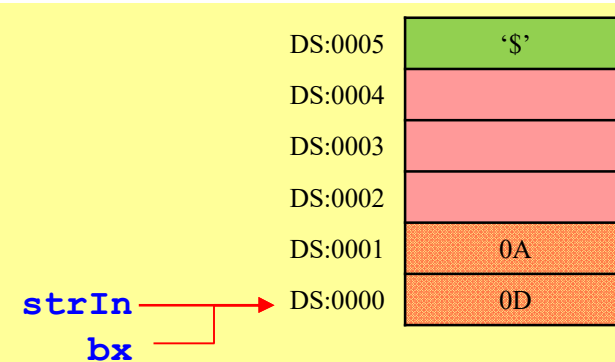
LF EQU 0x0A ;character Line Feed como LF

CR EQU 0x0D ;character Carriage return como CR

strIn DB CR, LF

RESB 3

DB '\$'



Laboratório

2. Estudo de um programa em assembly. Dica 1,

segment code

...

; -----CODIGO DO PROGRAMA-----

mov bx,strIn

mov ah,1

int 21h

mov byte[bx+2],al

int 21h

mov byte[bx+3],al

int 21h

mov byte[bx+4],al

mov dx, strIn

mov ah, 0x09

int 21h

; -----SAIDA DO PROGRAMA-----

mov ah,0X4c

int 0X21

segment data

; -----DEF. VAR,CONST E ALOCACAO-----

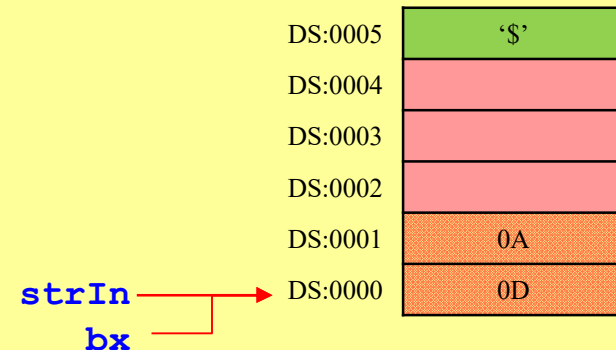
LF EQU 0x0A ;caracter Line Feed como LF

CR EQU 0x0D ;caracter Carriage return como CR

strIn DB CR,LF

RESB 3

DB '\$'



❑ Quando $ah=0x01$ a interrupção `int 0x21`, permite ingressar um byte por linha de comandos e guarda o byte ingressado em `AL`.

Laboratório

2. Estudo de um programa em assembly. Dica 1,

segment code

...

; -----CODIGO DO PROGRAMA-----

mov bx,strIn

mov ah,1

int 21h

mov byte[bx+2],al

int 21h

mov byte[bx+3],al

int 21h

mov byte[bx+4],al

mov dx, strIn

mov ah, 0x09

int 21h

; -----SAIDA DO PROGRAMA-----

mov ah,0X4c

int 0X21

segment data

; -----DEF. VAR, CONST E ALOCACAO-----

LF EQU 0x0A ;caracter Line Feed como LF

CR EQU 0x0D ;caracter Carriage return como CR

strIn DB CR,LF

RESB 3

DB '\$'



❑ Quando `ah=0x01` a interrupção `int 0x21`, permite ingressar um byte por linha de comandos e guarda o byte ingressado em `AL`.

Laboratório

2. Estudo de um programa em assembly. Dica 1,

segment code

...

; -----CODIGO DO PROGRAMA-----

mov bx,strIn

mov ah,1

int 21h

mov byte[bx+2],al

int 21h

mov byte[bx+3],al

int 21h

mov byte[bx+4],al

mov dx, strIn

mov ah, 0x09

int 21h

; -----SAIDA DO PROGRAMA-----

mov ah,0X4c

int 0X21

segment data

; -----DEF. VAR,CONST E ALOCACAO-----

LF EQU 0x0A ;character Line Feed como LF

CR EQU 0x0D ;character Carriage return como CR

strIn DB CR,LF

RESB 3

DB '\$'



❑ Quando `ah=0x01` a interrupção `int 0x21`, permite ingressar um byte por linha de comandos e guarda o byte ingressado em `AL`.

Laboratório

2. Estudo de um programa em assembly. Dica 1,

segment code

...

; -----CODIGO DO PROGRAMA-----

mov bx,strIn

mov ah,1

int 21h

mov byte[bx+2],al

int 21h

mov byte[bx+3],al

int 21h

mov byte[bx+4],al

mov dx, strIn

mov ah, 0x09

int 21h

; -----SAIDA DO PROGRAMA-----

mov ah,0X4c

int 0X21

segment data

; -----DEF. VAR, CONST E ALOCACAO-----

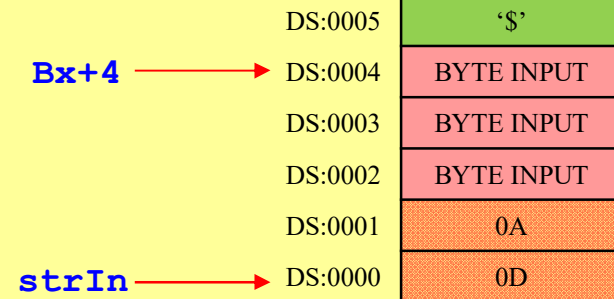
LF EQU 0x0A ;character Line Feed como LF

CR EQU 0x0D ;character Carriage return como CR

strIn DB CR,LF

RESB 3

DB '\$'



❑ Quando `ah=0x01` a interrupção `int 0x21`, permite ingressar um byte por linha de comandos e guarda o byte ingressado em `AL`.

Laboratório

3. Exercício

- ❑ 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 (`INT3`) no lugar da terminação do DOS. Monte e ligue gerando um .EXE. Teste o programa no debug.
-

Laboratório

3. Exercício: Dica 1, instrução `mul`

```
segment code
```

```
..start:
```

```
; -----INICIAR DS, SS e SP-----
```

```
; ds<-data
```

```
mov ax,data
```

```
mov ds,ax
```

```
; ss<-stack
```

```
mov ax,stack
```

```
mov ss,ax
```

```
; sp<-stacktop
```

```
mov sp,stacktop
```

```
; -----CODIGO DO PROGRAMA-----
```

```
mov ax,0xf1ff
```

```
mov bx,0xff2f
```

```
mov dx,0x0000
```

```
mul bx
```

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

```
mov ah,0x4c
```

```
int 0x21
```

```
segment data
```

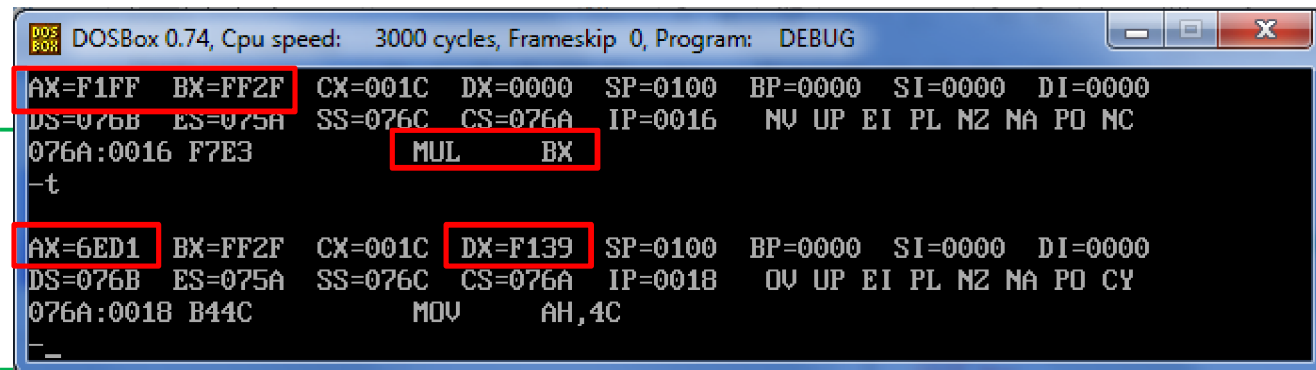
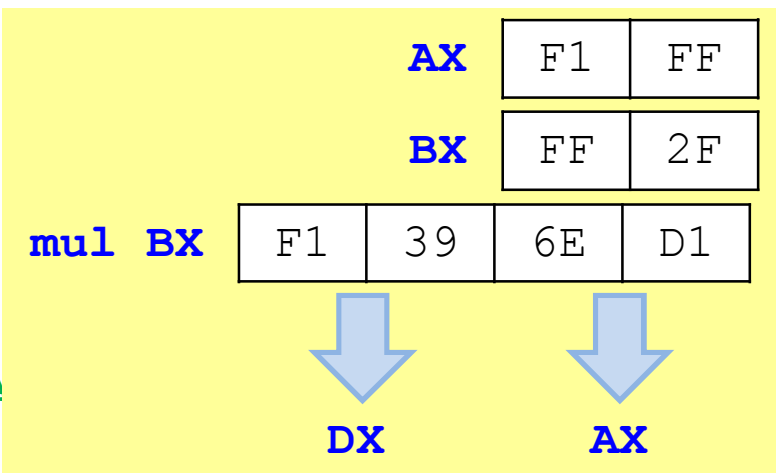
```
segment stack
```

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

```
resb 256 ; definição da pilha com total de 256 bytes
```

```
stacktop:
```

mul00101.asm



Laboratório

3. Exercício: Dica 2, multiplicação de variáveis

segment code

mul00102.asm

...

; -----CODIGO DO PROGRAMA-----

mov ax,[WARRAY1]

mov bx,[WARRAY2]

mov dx,0x0000

mul bx

mov WORD[RESULT],ax

mov WORD[RESULT+2],dx

; -----SAIDA DO PROGRAMA-----

mov ah,0x4c

int 0x21

segment data

; -----DEF. VAR, CONST E ALOCACAO-----

WARRAY1 DW 0xf1ff ; 1 word

WARRAY2 DW 0xff2f ; 1 word

RESULT: RESW 2 ; reserva 2 words

segment stack stack

...

Laboratório

3. Exercício: Dica 2, multiplicação de variáveis

segment code

...

```
; -----  
mov ax,[WARRAY1]  
mov bx,[WARRAY2]  
mov dx,0x0000  
mul bx  
mov WORD[RESULT],ax  
mov WORD[RESULT+2],dx  
; -----
```

mov ah,0x4c

int 0x21

segment data

; -----

WARRAY1 DW 0xf1ff ; 1 word

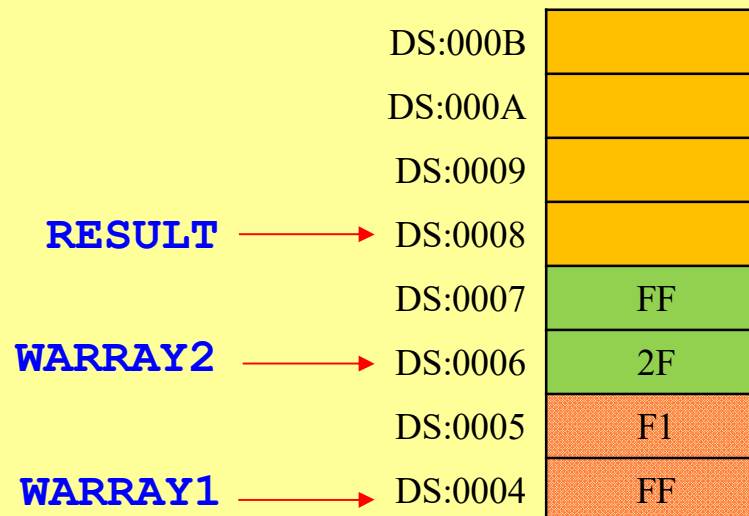
WARRAY2 DW 0xff2f ; 1 word

RESULT: RESW 2 ; reserva 2 words

segment stack stack

...

mul00102.asm



Laboratório

3. Exercício: Dica 2, multiplicação de variáveis

segment code

...

```
; -----  
mov ax, [WARRAY1]  
mov bx, [WARRAY2]  
mov dx, 0x0000  
mul bx  
mov WORD[RESULT], ax  
mov WORD[RESULT+2], dx  
; -----
```

segment data

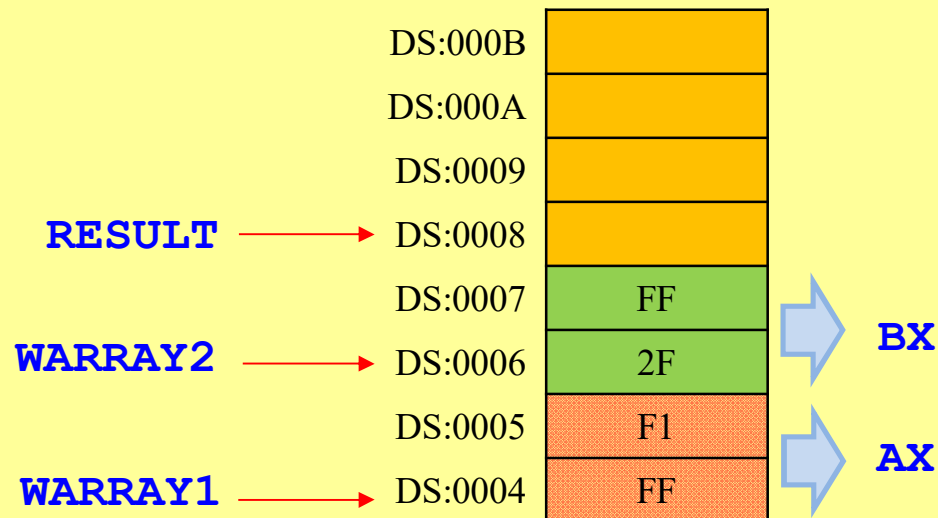
; -----

```
WARRAY1 DW 0xf1ff ; 1 word  
WARRAY2 DW 0xff2f ; 1 word  
RESULT: RESW 2 ; reserva 2 words
```

segment stack stack

...

mul00102.asm



Laboratório

3. Exercício: Dica 2, multiplicação de variáveis

segment code

...

```
; -----  
mov ax,[WARRAY1]  
mov bx,[WARRAY2]  
mov dx,0x0000  
mul bx  
mov WORD[RESULT],ax  
mov WORD[RESULT+2],dx  
; -----
```

mov ah,0x4c

int 0x21

segment data

; -----

WARRAY1 DW 0xf1ff ; 1 word

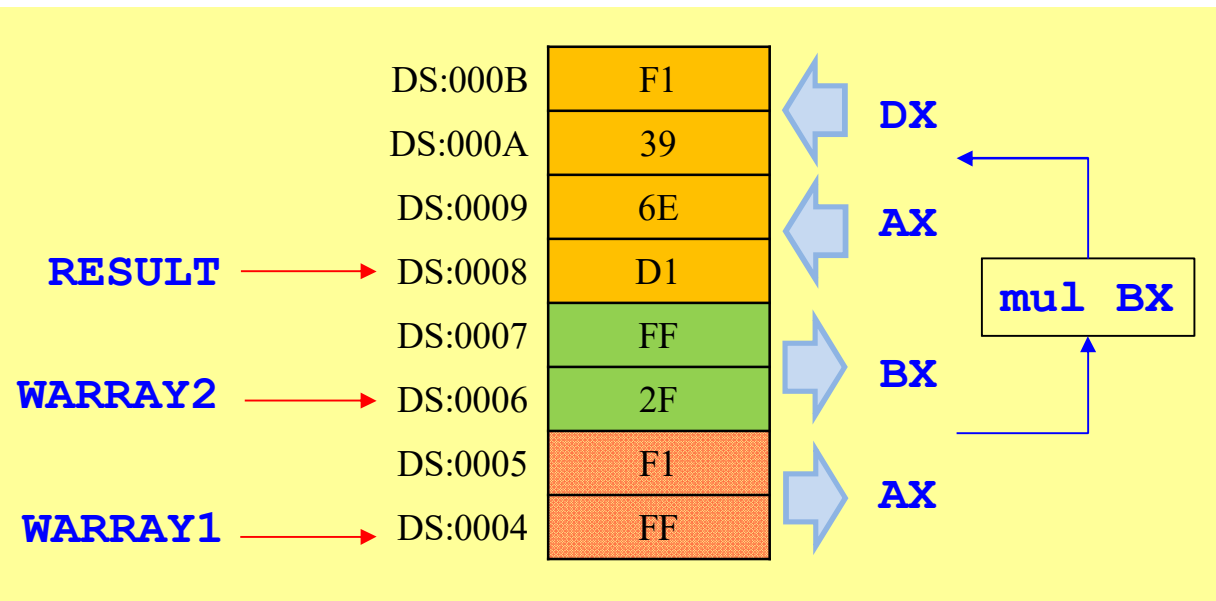
WARRAY2 DW 0xff2f ; 1 word

RESULT: RESW 2 ; reserva 2 words

segment stack stack

...

mul00102.asm



Laboratório

3. Exercício: Dica 2, multiplicação de variáveis

segment code

...

```
; -----  
mov ax,[WARRAY1]  
mov bx,[WARRAY2]  
mov dx,0x0000  
mul bx  
mov WORD[RESULT],ax  
mov WORD[RESULT+2],dx  
; -----
```

segment data

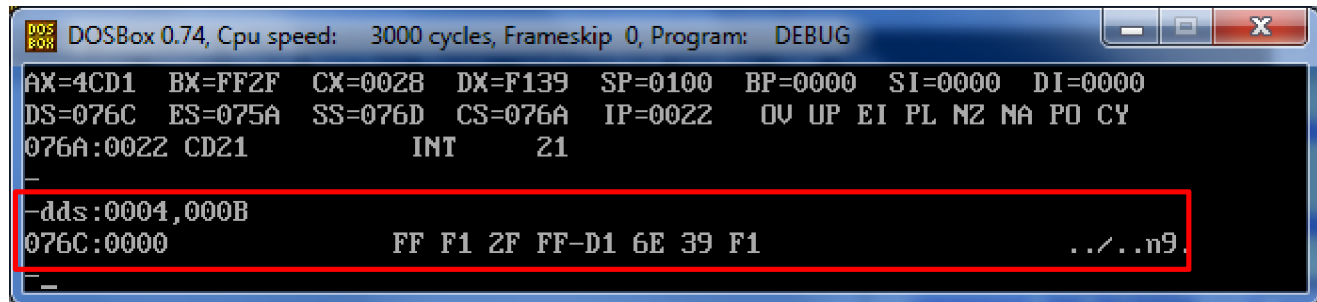
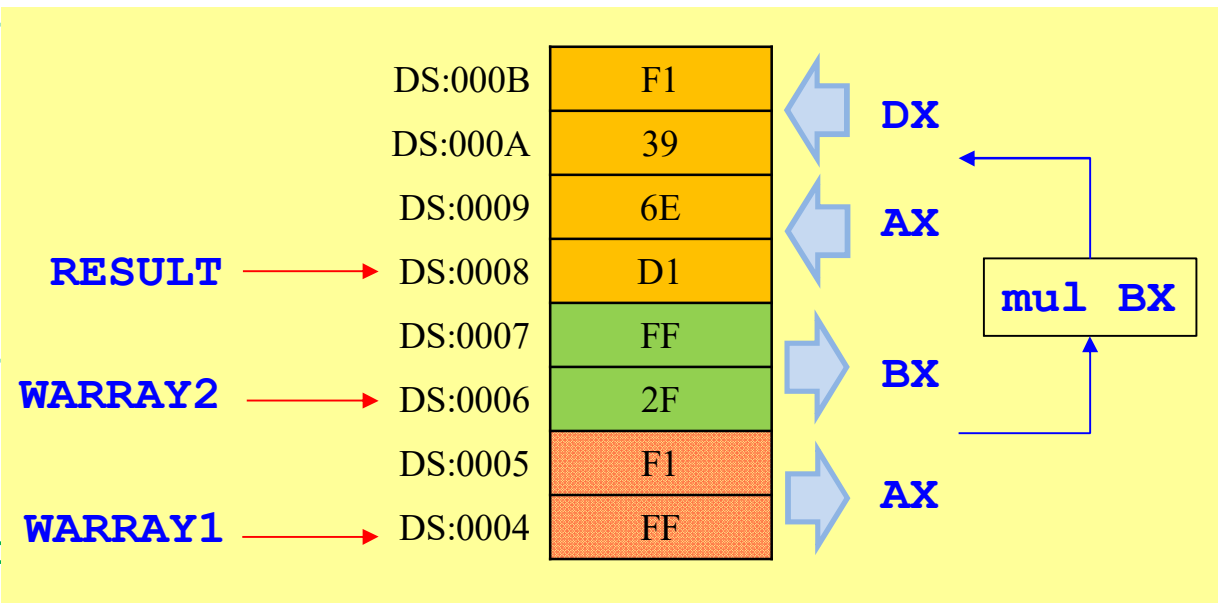
; -----

```
WARRAY1 DW 0xf1ff ; 1 word  
WARRAY2 DW 0xff2f ; 1 word  
RESULT: RESW 2 ; reserva 2 words
```

segment stack stack

...

mul00102.asm



Laboratório

3. Exercício: SOL1⇒multiplicação de vetores (endereçamentos base e indexado)

```
segment data
```

```
; -----DEF. VAR,CONST E ALOCACAO-----
```

```
WARRAY1 DW 0x1000, 0x0123 ; vetor de 2 words
```

```
WARRAY2 DW 0x1000, 0x0123 ; vetor de 2 words
```

```
RESULT: RESW 4 ; reserva 4 words
```

```
segment stack stack
```

```
; -----DEF. PILHA-----
```

```
resb 256 ; definição da pilha com total c
```

```
stacktop:
```

mul00103.asm

	DS:0015	
	DS:0014	
	DS:0013	
	DS:0012	
	DS:0011	
	DS:0010	
	DS:000F	
RESULT	→ DS:000E	
	DS:000D	01
	DS:000C	23
	DS:000B	10
WARRAY2	→ DS:000A	00
	DS:0009	01
	DS:0008	23
	DS:0007	10
WARRAY1	→ DS:0006	00

Laboratório

3. Exercício: SOL1 ⇒ multiplicação de vetores (endereçamentos base e indexado)

```
segment code
```

```
...
```

```
; -----CODIGO DO PROGRAMA-----
```

```
;guardando os indices
```

```
mov si, WARRAY1 ; source index<-WARRAY1
```

```
mov di, WARRAY2 ; destination index<-WARRAY2
```

```
mov bx, RESULT ; bx <- RESULT
```

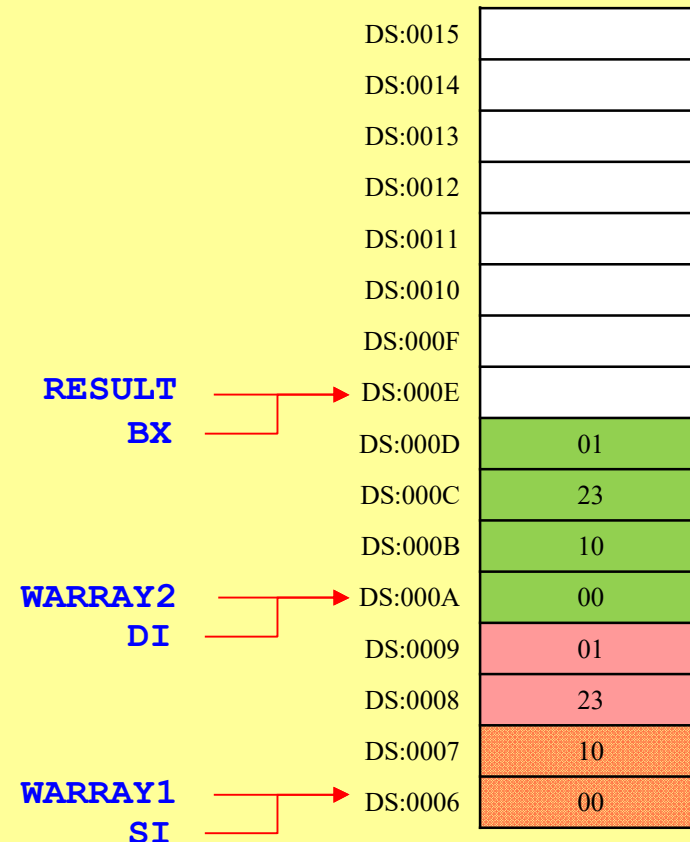
```
...
```

mul00103.asm

Registradores apontadores de índice

□ SI, DI

- Armazenam valores de **deslocamento de endereços** (*offset*), a fim de acessar regiões da memória como: blocos de dados e *arrays* e *strings*.
- Podem ser utilizados em operações aritméticas e lógicas, possibilitando que os valores de deslocamento sejam resultados de computações anteriores.



Laboratório

3. Exercício: SOL1⇒multiplicação de vetores (endereçamentos base e indexado)

```
segment code
```

```
...
```

```
; -----CODIGO DO PROGRAMA-----
```

```
;guardando os indices
```

```
mov si, WARRAY1 ; source index<-WARRAY1
```

```
mov di, WARRAY2 ; destination index<-WARRAY2
```

```
mov bx, RESULT ; bx <- RESULT
```

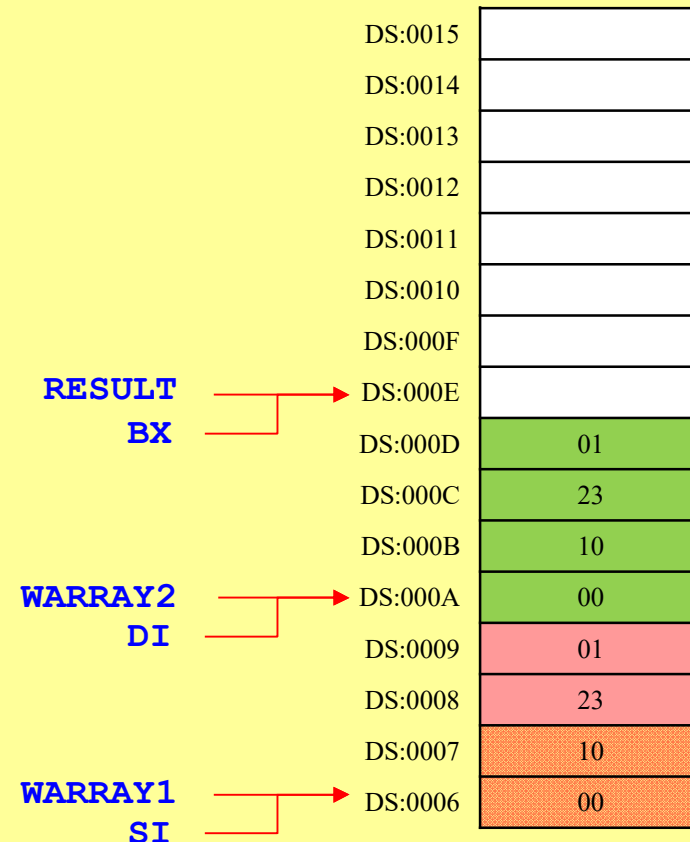
```
...
```

mul00103.asm

Registrador de Base

❑ BX

- Usado como registrador de BASE para referenciar posições de memória;
- BX armazena o endereço BASE de uma tabela ou vetor de dados, a partir do qual outras posições são obtidas adicionando-se um valor de deslocamento (**offset**).



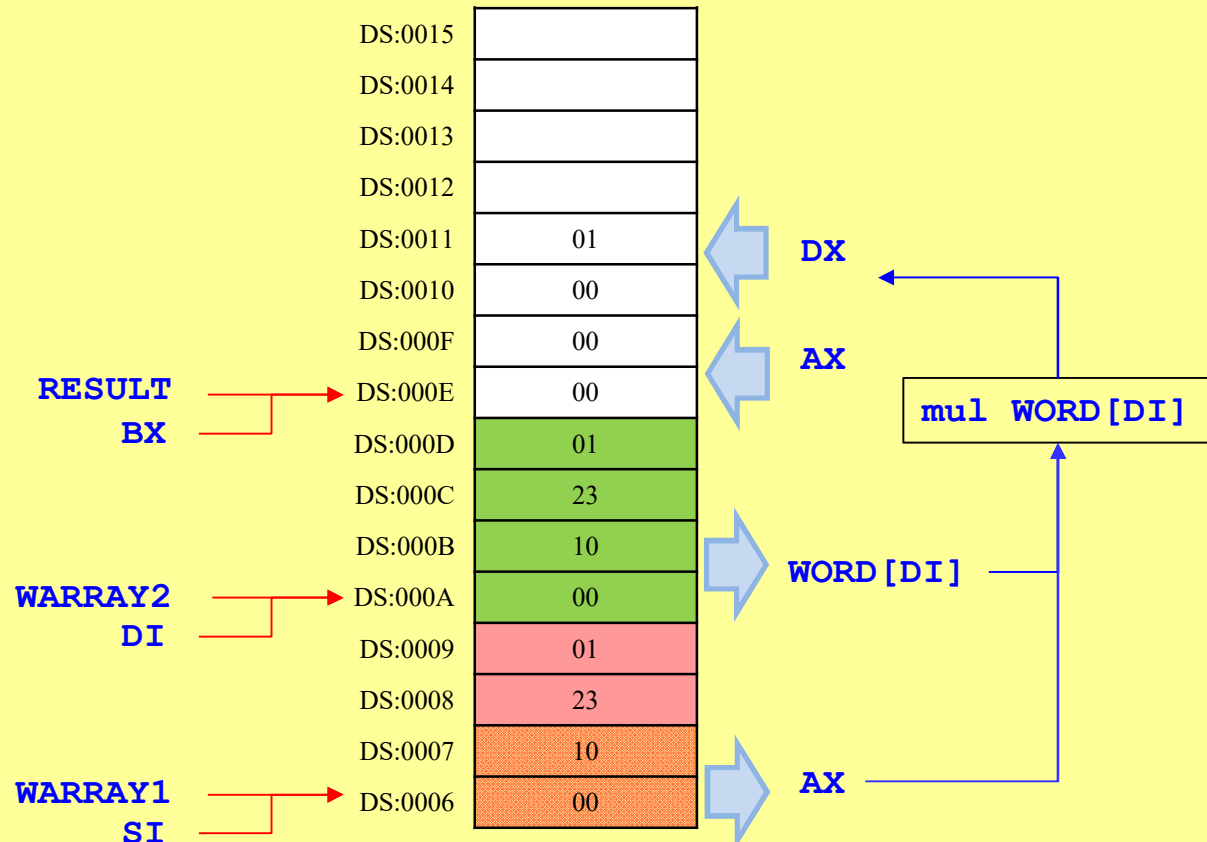
Laboratório

3. Exercício: SOL1⇒multiplicação de vetores (endereçamentos base e indexado)

segment code

mul00103.asm

```
...  
; -----CODIGO DO PROGRAMA-----  
;guardando os indices  
mov si, WARRAY1 ; source index<-WARRAY1  
mov di, WARRAY2 ; destination index<-WARRAY2  
mov bx, RESULT ; base  
;primeiro elemento  
mov ax,WORD[si]  
mov dx,0x0000  
mul WORD[di]  
mov WORD[bx],ax  
mov WORD[bx+2],dx
```



Laboratório

3. Exercício: SOL1⇒multiplicação de vetores (endereçamentos base e indexado)

segment code

mul00103.asm

```
...  
; -----CODIGO DO PROGRAMA-----  
;guardando os indices  
mov si, WARRAY1 ; source index<-WARRAY1  
mov di, WARRAY2 ; destination index<-WARRAY2  
mov bx, RESULT ; base address of result  
;primeiro elemento  
mov ax,WORD[si]  
mov dx,0x0000  
mul WORD[di]  
mov WORD[bx],ax  
mov WORD[bx+2],dx  
add si,2  
add di,2  
add bx,4
```

	DS:0015	
	DS:0014	
	DS:0013	
BX →	DS:0012	
	DS:0011	01
	DS:0010	00
	DS:000F	00
RESULT →	DS:000E	00
	DS:000D	01
DI →	DS:000C	23
	DS:000B	10
WARRAY2 →	DS:000A	00
	DS:0009	01
SI →	DS:0008	23
	DS:0007	10
WARRAY1 →	DS:0006	00

Laboratório

3. Exercício: SOL1⇒multiplicação de vetores (endereçamentos base e indexado)

segment code

mul00103.asm

...

; -----CODIGO DO PROGRAMA-----

;guardando os indices

mov si, WARRAY1 ; source index<-WARRAY1

mov di, WARRAY2 ; destination index<-WARRAY2

mov bx, RESULT ; base

;primeiro elemento

mov ax,WORD[si]

mov dx,0x0000

mul WORD[di]

mov WORD[bx],ax

mov WORD[bx+2],dx

add si,2

add di,2

add bx,4

;segundo elemento

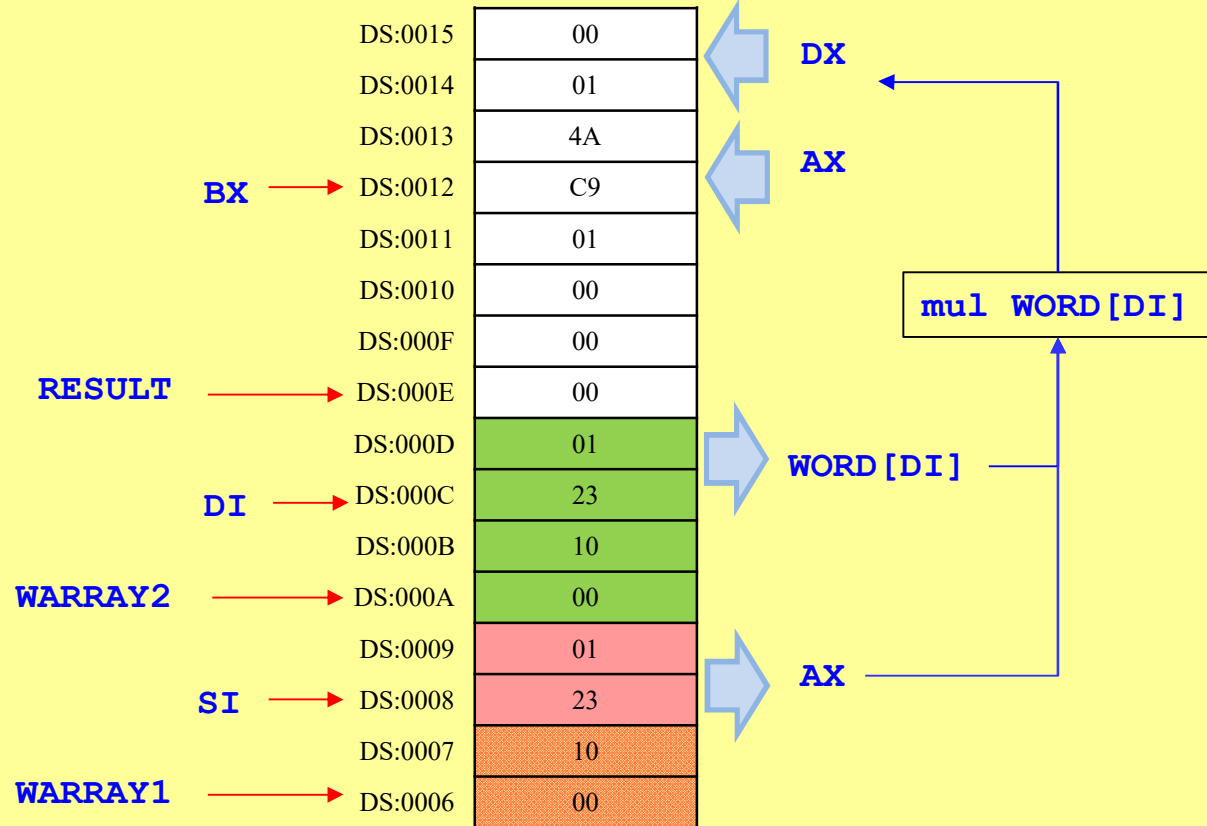
mov ax,WORD[si]

mov dx,0x0000

mul WORD[di]

mov WORD[bx],ax

mov WORD[bx+2],dx



Laboratório

3. Exercício: SOL1⇒multiplicação de vetores (endereçamentos base e indexado)

segment code

mul00103.asm

```
...  
; -----CODIGO DO PROGRAMA-----  
;guardando os indices  
mov si, WARRAY1 ; source index<-WARRAY1  
mov di, WARRAY2 ; destination index<-WARRAY2  
mov bx, RESULT ; bx <- RESULT
```

;primeiro elemento

```
mov ax,WORD[si]
```

```
mov dx,0x0000
```

```
mul WORD[di]
```

```
mov WORD[bx],ax
```

```
mov WORD[bx+2],dx
```

```
add si,2
```

```
add di,2
```

```
add bx,4
```

;segundo elemento

```
mov ax,WORD[si]
```

```
mov dx,0x0000
```

```
mul WORD[di]
```

```
mov WORD[bx],ax
```

```
mov WORD[bx+2],dx
```

; -----SAIDA DO PROGRAMA-----

```
int3
```

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG  
C:\>debug mul001.exe  
-R  
AX=FFFF BX=0000 CX=005F DX=0000 SP=0100 BP=0000 SI=0000 DI=0000  
DS=075A ES=075A SS=0771 CS=076A IP=0000 NV UP EI PL NZ NA PO NC  
076A:0000 B86F07 MOV AX,076F  
-a=0000  
AX=A6D4 BX=0017 CX=005F DX=03B4 SP=0100 BP=0000 SI=0007 DI=0000  
DS=076F ES=075A SS=0771 CS=076A IP=0052 OV UP EI PL NZ NA PE CY  
076A:0052 CC  
-  
INT 3
```

No NASM as seguintes sintaxes não são validas
int 0x03 ou int 3

Laboratório

3. Exercício: SOL2⇒multiplicação de vetores (endereçamento indexado+desl)

segment data

; -----DEF. VAR,CONST E ALOCACAO-----

WARRAY1 DW 0x1000, 0x0123 ; vetor de 2 words

WARRAY2 DW 0x1000, 0x0123 ; vetor de 2 words

RESULT: RESW 4 ; reserva 4 words

segment stack stack

; -----DEF. PILHA-----

resb 256 ; definição da pilha com total c

stacktop:

mul00104.asm

	DS:0015	
	DS:0014	
	DS:0013	
	DS:0012	
	DS:0011	
	DS:0010	
	DS:000F	
RESULT →	DS:000E	
	DS:000D	01
	DS:000C	23
	DS:000B	10
WARRAY2 →	DS:000A	00
	DS:0009	01
	DS:0008	23
	DS:0007	10
WARRAY1 →	DS:0006	00

Laboratório

3. Exercício: SOL2⇒multiplicação de vetores (endereçamento indexado+desl)

```
segment code
```

```
...
```

```
; -----CODIGO DO PROGRAMA-----
```

```
;inicializando os indices fonte e destino
```

```
mov si, 0x00
```

```
mov di, 0x00
```

```
...
```

mul00103.asm

Registradores apontadores de índice

□ SI, DI

- Armazenam valores de **deslocamento de endereços** (*offset*), a fim de acessar regiões da memória como: blocos de dados e *arrays* e *strings*.
- Podem ser utilizados em operações aritméticas e lógicas, possibilitando que os valores de deslocamento sejam resultados de computações anteriores.

	DS:0015	
	DS:0014	
	DS:0013	
	DS:0012	
	DS:0011	
	DS:0010	
	DS:000F	
	DS:000E	
	DS:000D	01
	DS:000C	23
	DS:000B	10
	DS:000A	00
	DS:0009	01
	DS:0008	23
	DS:0007	10
	DS:0006	00

RESULT → DS:000E

WARRAY2 → DS:000A

WARRAY1 → DS:0006

Laboratório

3. Exercício: SOL2⇒multiplicação de vetores (endereçamento indexado+desl)

segment code

...

; -----CODIGO DO PROGRAMA-----

;guardando os indices

mov si, 0x00

mov di, 0x00

;primeiro elemento

mov ax, [WARRAY1+si

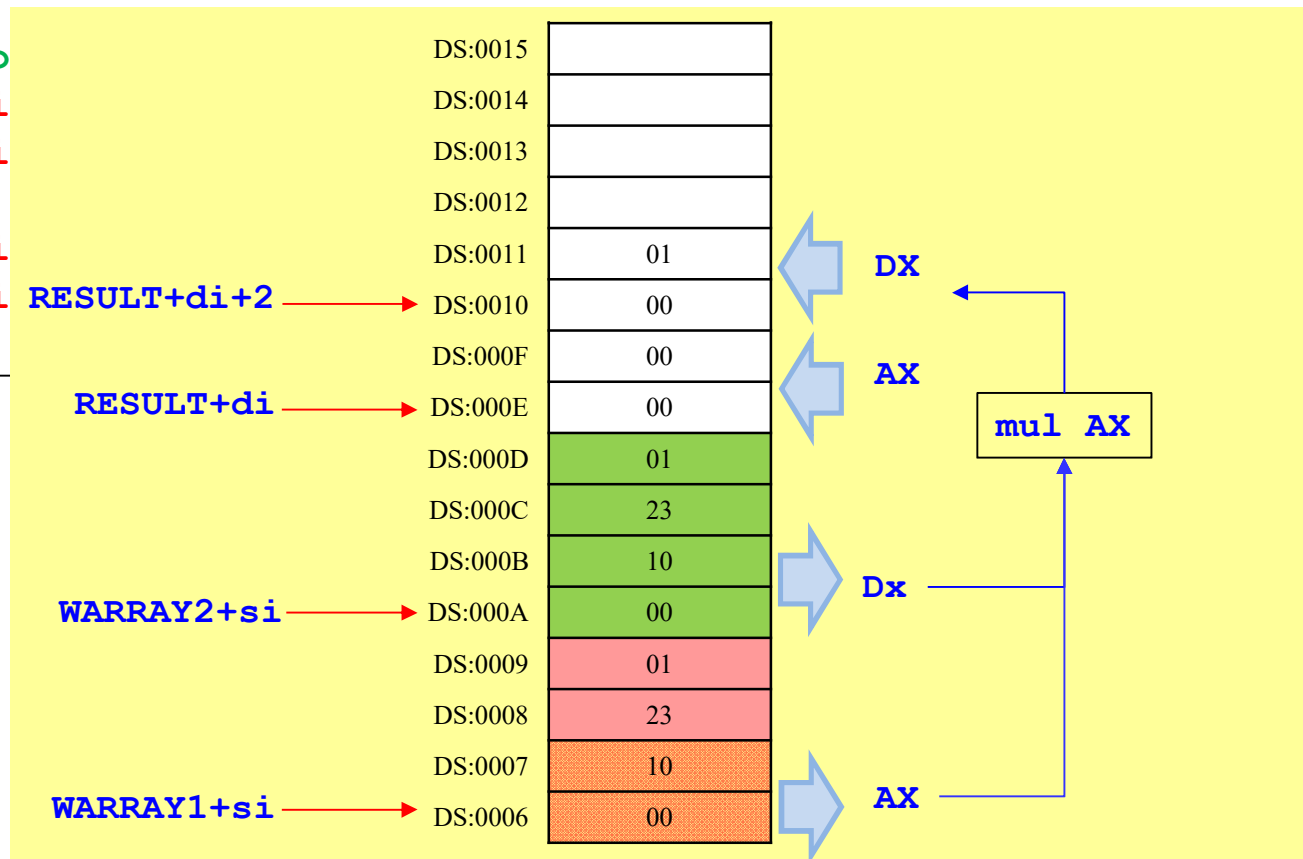
mov dx, [WARRAY2+si

mul ax

mov word[RESULT+di

mov word[RESULT+di

mul00103.asm



Laboratório

3. Exercício: SOL2⇒multiplicação de vetores (endereçamento indexado+desl)

segment code

mul00103.asm

...

; -----CODIGO DO PROGRAMA-----

;guardando os indices

mov si, 0x00

mov di, 0x00

;primeiro elemento

mov ax,[WARRAY1+si]

mov dx,[WARRAY2+si]

mul ax

mov word[RESULT+di],ax

mov word[RESULT+di+2],dx

add si,2

add di,4

	DS:0015	
RESULT+di+2	DS:0014	
	DS:0013	
RESULT+di	DS:0012	
	DS:0011	01
	DS:0010	00
	DS:000F	00
RESULT	DS:000E	00
	DS:000D	01
WARRAY2+si	DS:000C	23
	DS:000B	10
WARRAY2	DS:000A	00
	DS:0009	01
WARRAY1+si	DS:0008	23
	DS:0007	10
WARRAY1	DS:0006	00

Laboratório

3. Exercício: SOL2⇒multiplicação de vetores (endereço+desl)

segment code

mul00103.asm

...

; -----CODIGO DO PROGRAMA-----

;guardando os indices

mov si, 0x00

mov di, 0x00

;primeiro elemento

mov ax,[WARRAY1+si]

mov dx,[WARRAY2+si]

mul ax

mov word[RESULT+di],ax

mov word[RESULT+di+2],dx

add si,2

add di,4

;segundo elemento

mov ax,[WARRAY1+si]

mov dx,[WARRAY2+si]

mul ax

mov word[RESULT+di],ax

mov word[RESULT+di+2],dx

