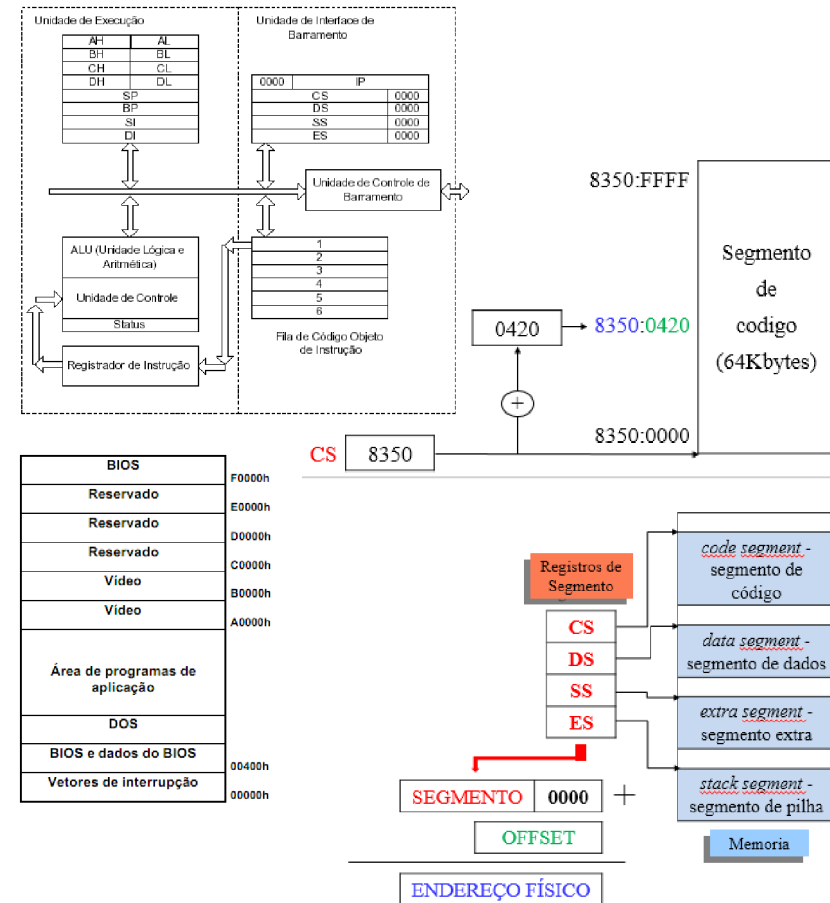


# Microprocessadores 8086

*Professor*

Dr. Jorge Leonid Aching Samatelo

[jlasm001@gmail.com](mailto:jlasm001@gmail.com)



# Índice

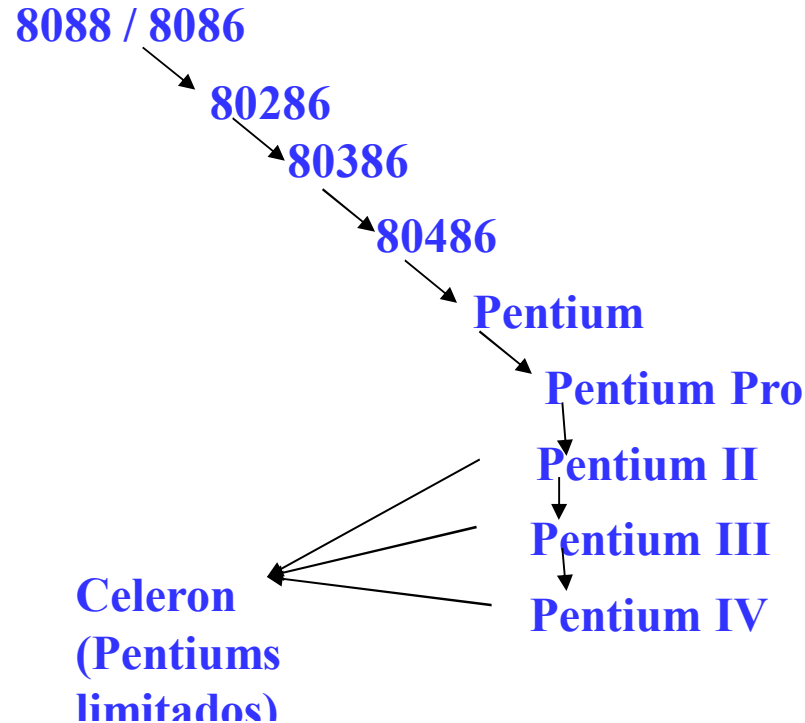
- ☐ O microprocessador 8086
- ☐ Debug
- ☐ Laboratório

# **O microprocessador 8086**

# O microprocessador 8086

## Porquê o estudo do 8086?

- ❑ O estudo da arquitetura do 8086 (ou 8088) **permite entender a arquitetura dos processadores mais modernos.**
- ❑ O modelo de programação básico é muito similar aos processadores mais modernos e **todos os recursos em nível de aplicativos, como *registradores*, *tipos de dados* e *modo de endereçamento*, são extensões do conjunto de recursos do 8086 original.**



# O microprocessador 8086

## O microprocessador 8086

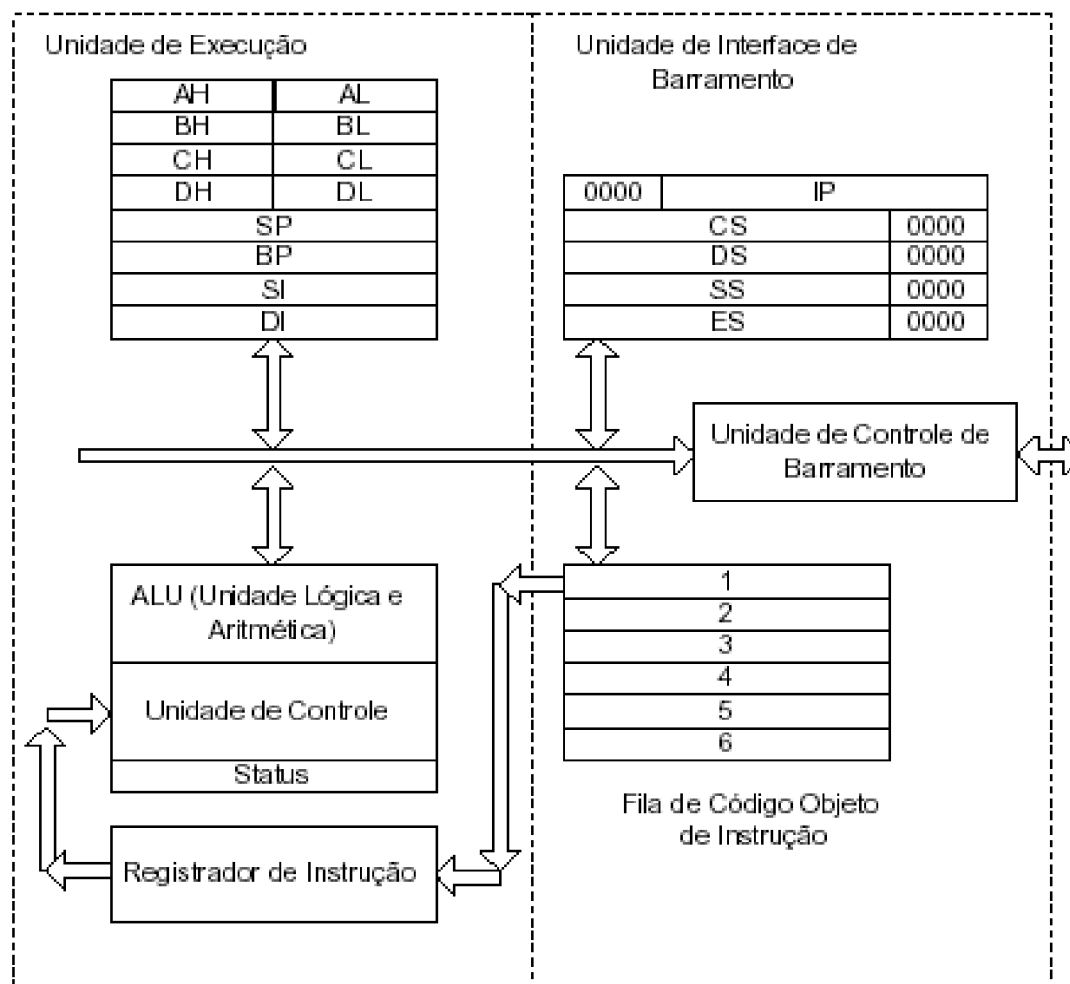
- ❑ O microprocessador 8086 da Intel é um **microprocessador de 16 bits**, de forma que:
  - sua **unidade lógica e aritmética**,
  - seus **registradores internos**, e
  - a maior parte das suas **instruções**
- ❑ foram projetados para **trabalhar com palavras de 16 bits**.
  
- ❑ Além disso o 8086 tem:
  - **UM BARRAMENTO DE DADOS** de largura 16 bits,
    - ❖ ou seja, pode ler e escrever na memória ou nos portos de E/S **utilizando 16 bits de uma vez só**.
  - **UM BARRAMENTO DE ENDEREÇOS** de 20 bits,
    - ❖ de forma que o 8086 pode endereçar 1 MB ( =  $2^{20}$  bits) posições de memória.
    - ❖ Cada uma destas posições de memória é ocupada por um **BYTE**.
  - Tem um conjunto de cerca de **123 instruções**.

# O microprocessador 8086

## Diagrama de blocos do processador 8088

❑ No diagrama de blocos estão ilustrados os componentes principais do microprocessador 8086:

- registradores,
- unidade lógica e aritmética (ALU),
- barramentos.



# O microprocessador 8086

## Registradores

❑ Elementos de memória muito rápida dentro da CPU.

➤ de dados, ou de propósito geral.

➤ de endereços (segmentos, apontadores e índices)

➤ sinalizadores de estado e controle (FLAGS)

	15	8	7	0	
AX	AH		AL		Acumulador
BX	BH		BL		Base
CX	CH		CL		Contador
DX	DH		DL		Dado
SP					Ponteiro para pilha
BP					Ponteiro base
SI					Índice fonte
DI					Índice destino
IP					Apontador a instruções
FLAG					FLAGS
CS					Segmento de código
DS					Segmento de dados
SS					Segmento de pilha
ES					Segmento extra

# O microprocessador 8086

## Registradores

### Registradores de dados

#### AX, BX, CX e DX

- ❑ São todos registradores de 16 bits.
- ❑ Utilizados nas operações aritméticas e lógicas.
- ❑ Podem ser usados como registradores de 16 ou 8 bits.

➤ 8 registradores de 8 bits cada

❖ AX é dividido em AH e AL.

❖ BX é dividido em BH e BL.

❖ CX é dividido em CH e CL.

❖ DX é dividido em DH e DL.

▪ "H" ⇒ byte alto ou superior

▪ "L" ⇒ byte baixo ou inferior

	15	8	7	0	
AX	AH		AL		Acumulador
BX	BH		BL		Base
CX	CH		CL		Contador
DX	DH		DL		Dado



# O microprocessador 8086

## Registradores

### Registradores de segmento

CS, DS, SS e ES

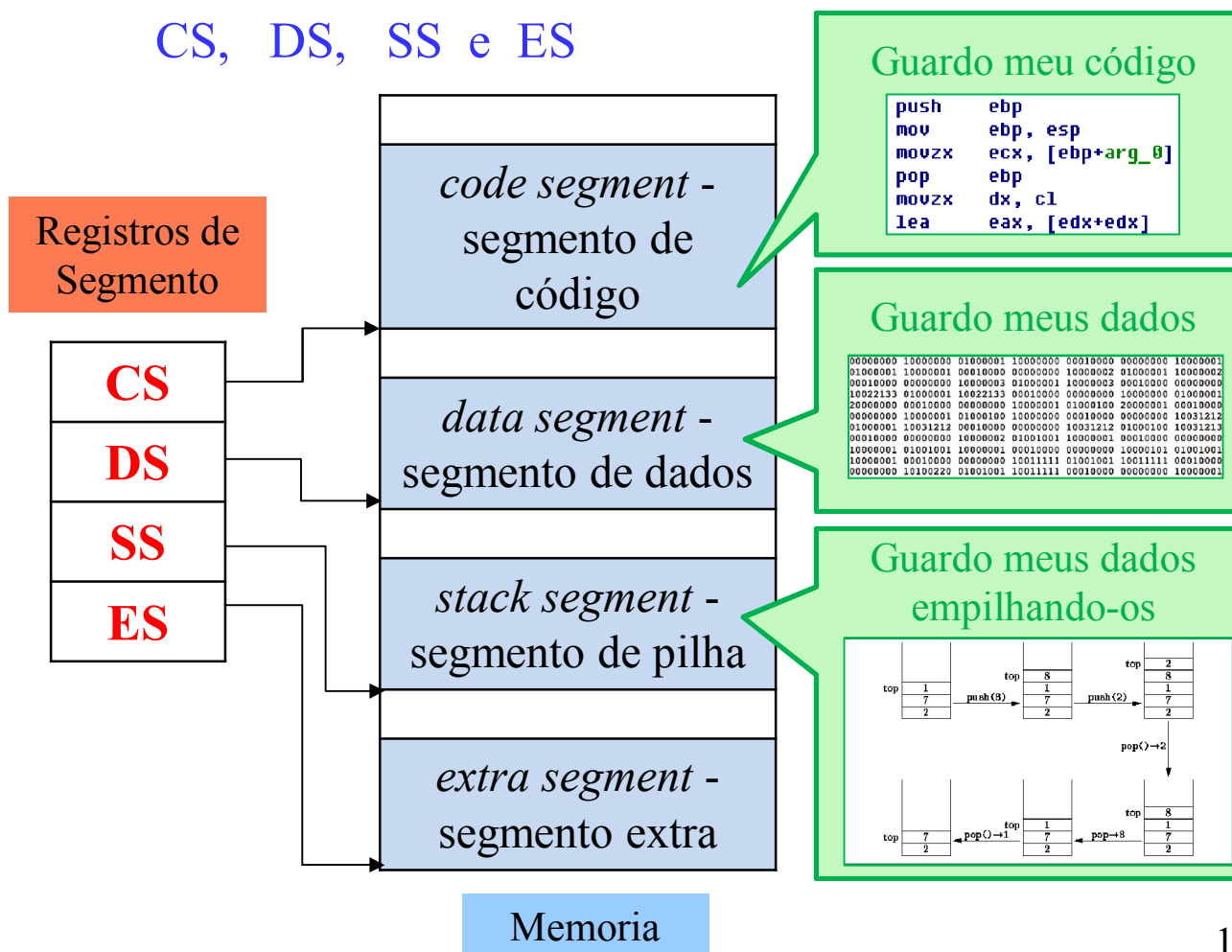
- ❑ São todos registradores de 16 bits.
- ❑ O endereçamento no 8086 é diferenciado para:
  - código de programa (instruções)
  - dados
  - Pilhas
- ❑ **Segmento**: é um bloco de memória de 64 KBytes, endereçável.
- ❑ durante a execução de um programa no 8086, há 4 segmentos ativos:
  - segmento de código ⇒ endereçado por CS
  - segmento de dados ⇒ endereçado por DS
  - segmento de pilha ⇒ endereçado por SS (*stack segment*)
  - segmento extra ⇒ endereçado por ES

# O microprocessador 8086

## Registradores

### Registradores de segmento

CS, DS, SS e ES



# O microprocessador 8086

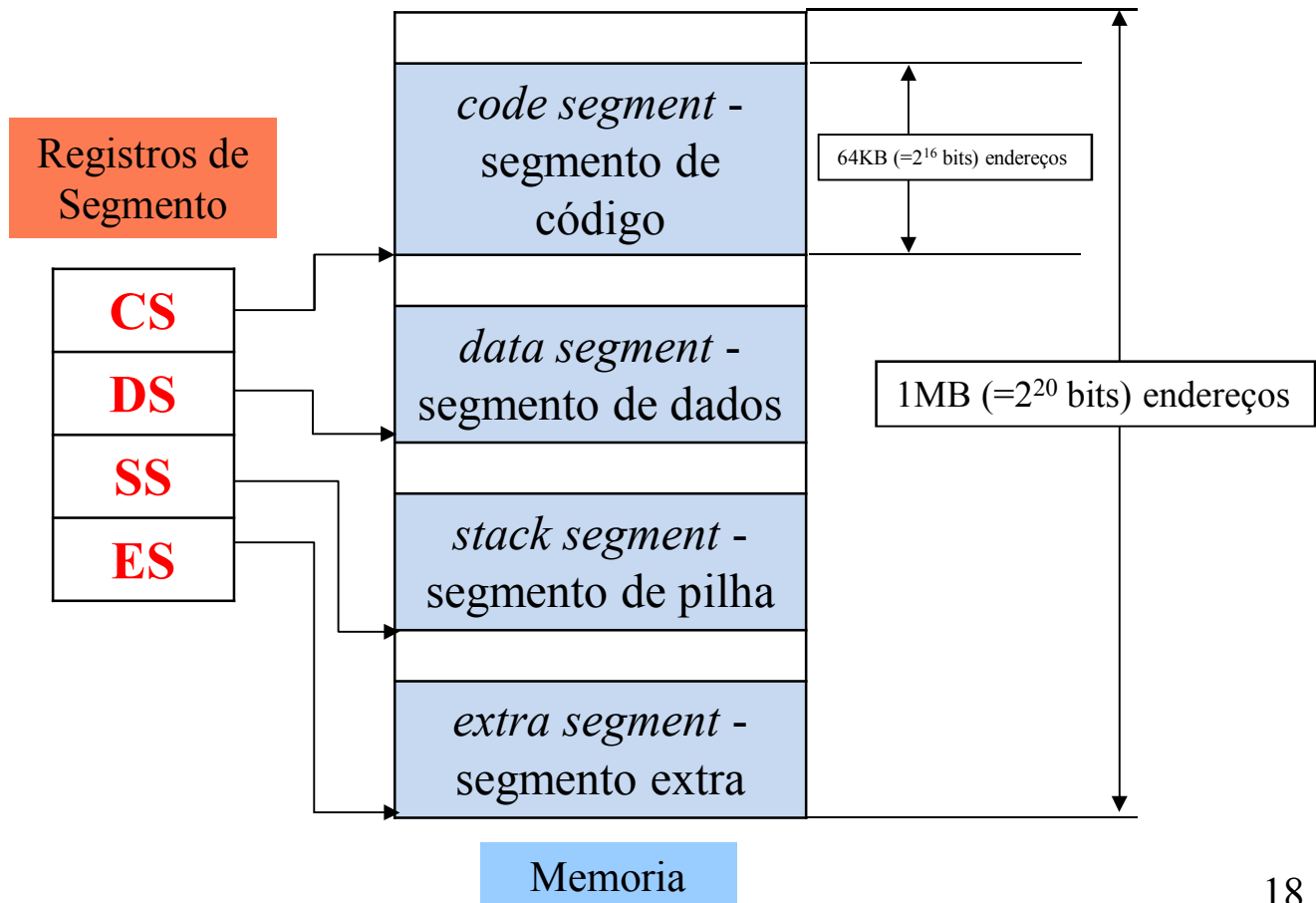
## Registradores

### Registradores de segmento



Em total temos 16 ( $=2^{20}/2^{16}$ ) segmentos, mas ativos para um processo unicamente 4 segmentos.

CS, DS, SS e ES



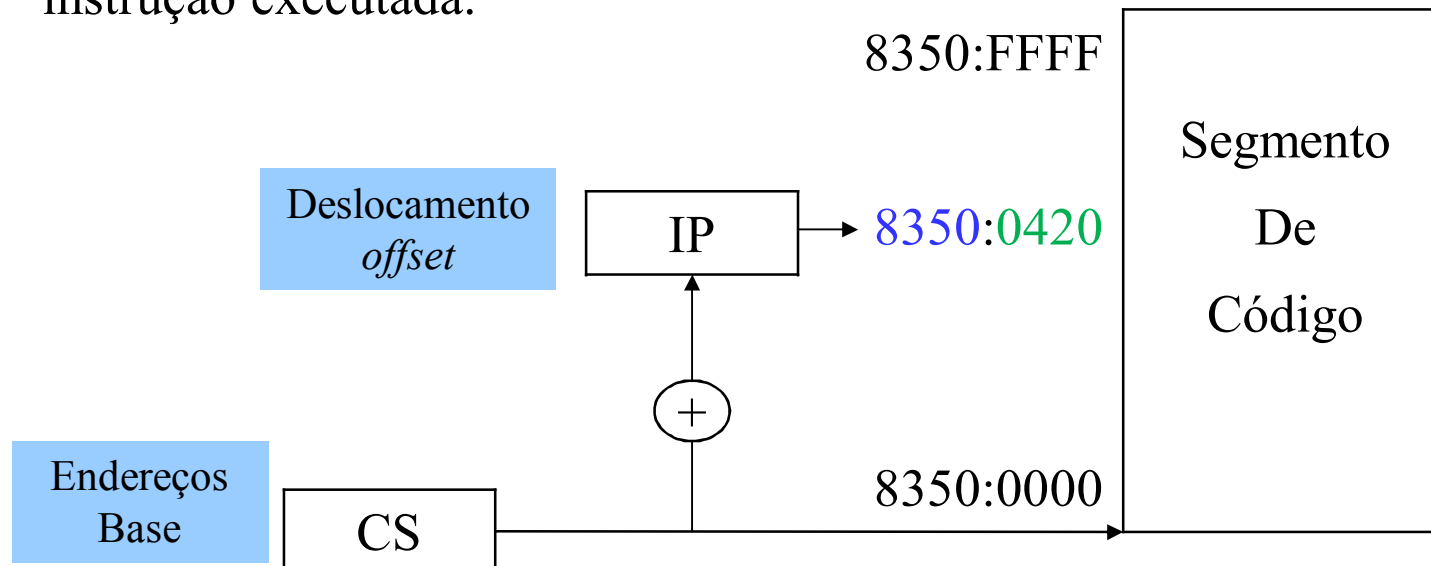
# O microprocessador 8086

## Registradores

### Registrador apontador de instrução

#### *IP (instruction pointer)*

- ❑ Utilizado em conjunto com **CS** para localizar a posição, dentro do **segmento de código** corrente, da próxima instrução a ser executada.
- ❑ **IP** é automaticamente incrementado em função do número de bytes da instrução executada.



# O microprocessador 8086

## Registradores

### Registradores apontador de pilha e de índice

SP, BP, SI, DI

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

# O microprocessador 8086

## Registradores

### Registrador de sinalizadores (FLAGS)

#### FLAGS

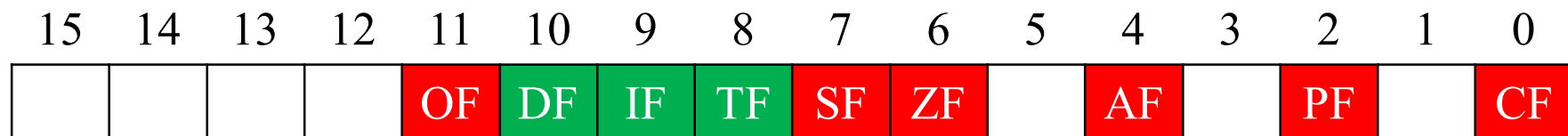
- ❑ Indica o estado do microprocessador durante a execução de cada instrução.
- ❑ Conjunto de bits, cada qual indicando alguma propriedade.
- ❑ Subdividem-se em:
  - **FLAGS da estado** (*status*) e
  - **FLAGS de controle**.

# O microprocessador 8086

## Registradores

### Registrador de sinalizadores (FLAGS)

#### FLAGS



#### ❑ Organização:

- 1 registrador de 16 bits
  - ❖ 6 FLAGS de estado
  - ❖ 3 FLAGS de controle
  - ❖ 7 bits não utilizados (sem função)

# O microprocessador 8086

## Organização de memória

- ❑ Cada byte na memória possui um endereço de 20 bits iniciando em 0 até  $2^{20}-1$  ou seja, 1M de memória endereçável;
- ❑ **Endereços físicos** são representados por 5 dígitos hexadecimais; de 00000h – FFFFFh. **Exemplo:**

0000 0000 0000 0000 0000b  $\Rightarrow$  0000h

0000 0000 0000 0000 0001b  $\Rightarrow$  0001h

0000 0000 0000 0000 0010b  $\Rightarrow$  0002h  $\Rightarrow$  5 dígitos hexadecimais

0000 0000 0000 0000 0011b  $\Rightarrow$  0003h

....

1111 1111 1111 1111 1111b  $\Rightarrow$  FFFFh



# O microprocessador 8086

## Segmentação da Memória

❑ O 8086 opera internamente com 16 bits

### ➤ Problema

❖ 20 bits de endereços é grande demais para ser colocado em registradores de 16 bits;

### ➤ Solução

❖ Utilizar a ideia da segmentação de memória!

▪ A memória usada por um programa é dividida em blocos de memória.

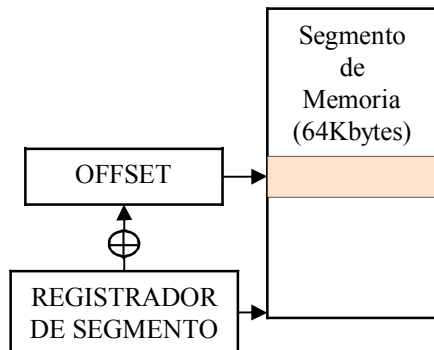
• Cada bloco de memória é de 64Kbytes ( $=2^{16}\text{bits}=65.536\text{bytes}$ ) consecutivos;

▪ Então, para determinar o endereço de um dado específico, são usados dois registradores:

• Um registrador de 16 bits para identificar o **segmento de memória**;

• Um registrador de 16 bits para identificar a posição de memória específica dentro do segmento, ou seja o **deslocamento (offset)** em relação ao endereço do segmento.

• Faixa de endereços em um segmento vai de 0000h a FFFFh



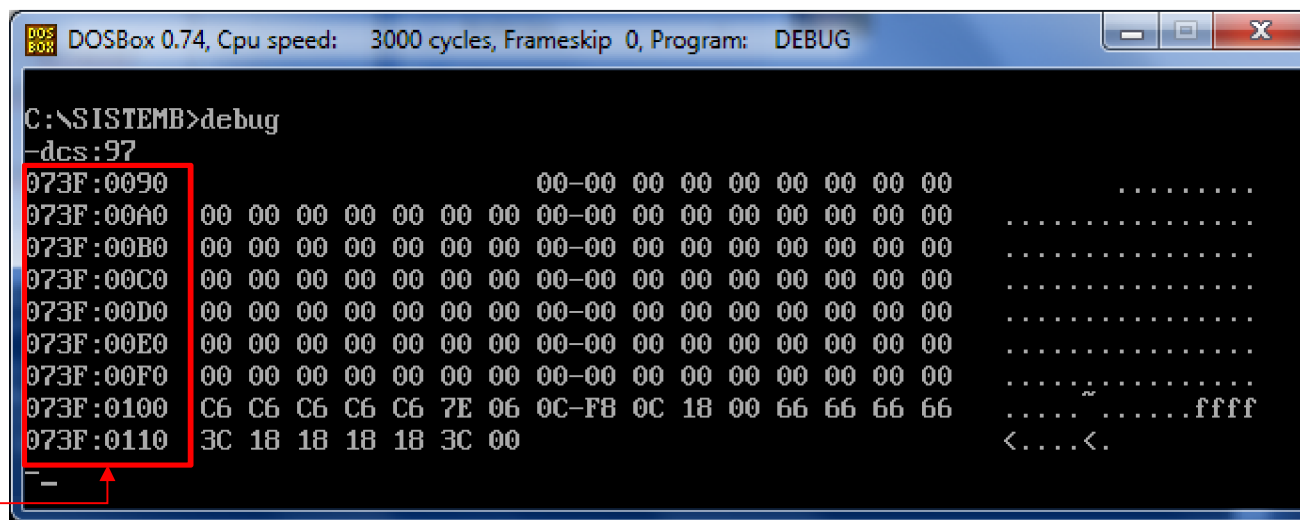
# O microprocessador 8086

## Segmentação da Memória

- ❑ Uma **posição de memória** é especificada
  - pelo **número de segmento** e
  - por um **deslocamento (offset)** em relação ao início do segmento.

### ❑ Sintaxes do formato de endereço lógico

**SEGMENTO:OFFSET**



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-dcs:97
073F:0090 00-00 00 00 00 00 00 00 00 00 .....
073F:00A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00B0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00C0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:0100 C6 C6 C6 C6 C6 7E 06 0C-F8 0C 18 00 66 66 66 .....ffff
073F:0110 3C 18 18 18 18 3C 00 <....<
```

Endereço lógico  
**SEGMENTO:OFFSET**

# O microprocessador 8086

## Segmentação da Memória

- ❑ Uma **posição de memória** é especificada
  - pelo **número de segmento** e
  - por um **deslocamento (*offset*)** em relação ao início do segmento.

### ❑ Sintaxes do formato de endereço lógico

**SEGMENTO:OFFSET**

- ❑ O endereço físico é calculado como:

**ENDEREÇO FÍSICO** = **SEGMENTO**\*16 + **OFFSET**

**ENDEREÇO FÍSICO** = **SEGMENTO**<<4 + **OFFSET**



# O microprocessador 8086

## Segmentação da Memória

### **Exemplo:**

- ❑ Dado o endereço lógico de uma instrução: 8350:0420h, reconhecer:
    1. Número de segmento
    2. Deslocamento
    3. Endereço físico
-

# O microprocessador 8086

## Segmentação da Memória

### Exemplo:

❑ Dado o endereço lógico de uma instrução: 8350:0420h, reconhecer:

1. Número de segmento
2. Deslocamento
3. Endereço físico

---

❑ reconhece-se:

- Número de segmento, 8350h
- Deslocamento, 0420h

# O microprocessador 8086

## Segmentação da Memória

### Exemplo:

❑ Dado o endereço lógico de uma instrução: 8350:0420h, reconhecer:

1. Número de segmento
2. Deslocamento
3. **Endereço físico**

---

❑ Reconhece-se:

- Número de segmento, 8350h
- Deslocamento, 0420h

❑ O endereço físico vale:

$$\begin{array}{rcl} 83500h & \Rightarrow & \text{desloca-se 1 casa hexa (4 casas binárias)} \\ + \quad 0420h & \Rightarrow & \text{soma-se o deslocamento} \\ \hline 83920h & \Rightarrow & \text{endereço físico resultante (20 bits)} \end{array}$$

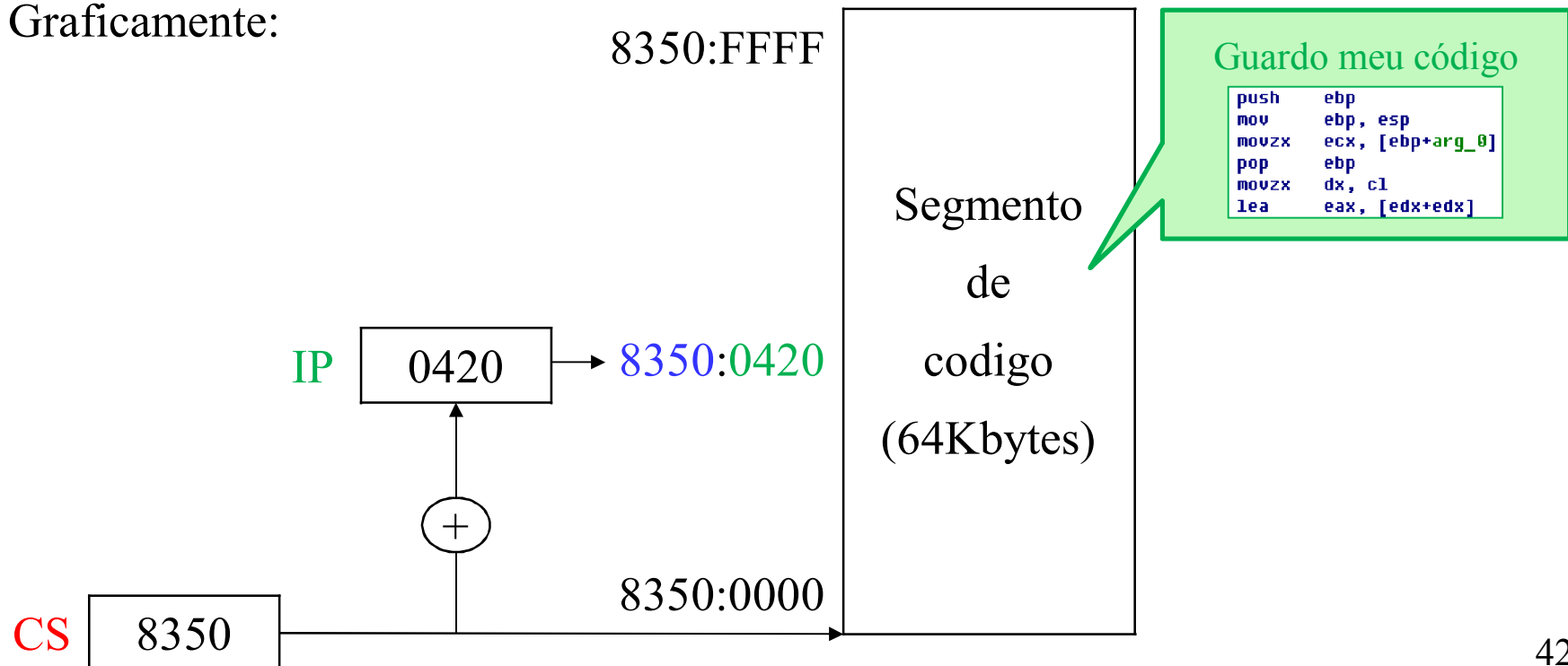
# O microprocessador 8086

## Segmentação da Memória

### Exemplo:

- ❑ Dado o endereço lógico de uma instrução: 8350:0420h, reconhecer:
  1. Número de segmento
  2. Deslocamento
  3. **Endereço físico**

- ❑ Graficamente:



**Debug**



# Debug

## Introdução

- ❑ O DEBUG é um programa que tem sua origem no sistema operacional **MS-DOS**, e serve para a criação e depuração de programas.
- ❑ Através do DEBUG, pode-se:
  - verificar os registradores do sistema,
  - efetuar consultas na memória do sistema e
  - desenvolver e alterar programas simples.
- ❑ O programa esta disponível para **Windows 3.1, Windows 95, Windows 98, Windows NT, Windows XP, Windows Vista 32 bits, Windows 7 32 bits**, poderia ser acessado pelo shell (command.com ou cmd), porém foi removido do **Microsoft Windows Vista 64 bits, windows 7 64 bits e windows 8 de 32 e 64 bits**.
- ❑ O **DOSBox** é um **emulador** que emula (vagamente "simula") um computador IBM PC compatível rodando em cima deste um antigo sistema operacional, o MS-DOS.



# Debug

## Comandos

❑ Os principais comandos do programa DEBUG estão ilustrados na sgte. Tabela

A	( <i>Assemble</i> ) Montar instruções simbólicas em código de máquina
D	( <i>Dumpping</i> ) Mostrar o conteúdo de uma região da memória
E	( <i>Enter</i> ) Entrar dados na memória, iniciando num endereço específico
G	( <i>Go</i> ) Executar um programa executável na memória
N	( <i>Name</i> ) Dar nome a um programa
P	( <i>Proceed</i> ) Proceder, ou executar um conjunto de instruções relacionadas
Q	( <i>Quit</i> ) Sair do programa DEBUG
R	( <i>Register</i> ) Mostrar o conteúdo de um ou mais registradores
T	( <i>Trace</i> ) Executar passo a passo as instruções
U	( <i>Unassemble</i> ) Desmontar o código de máquina em instruções simbólicas
W	( <i>Write</i> ) Gravar um programa em disco

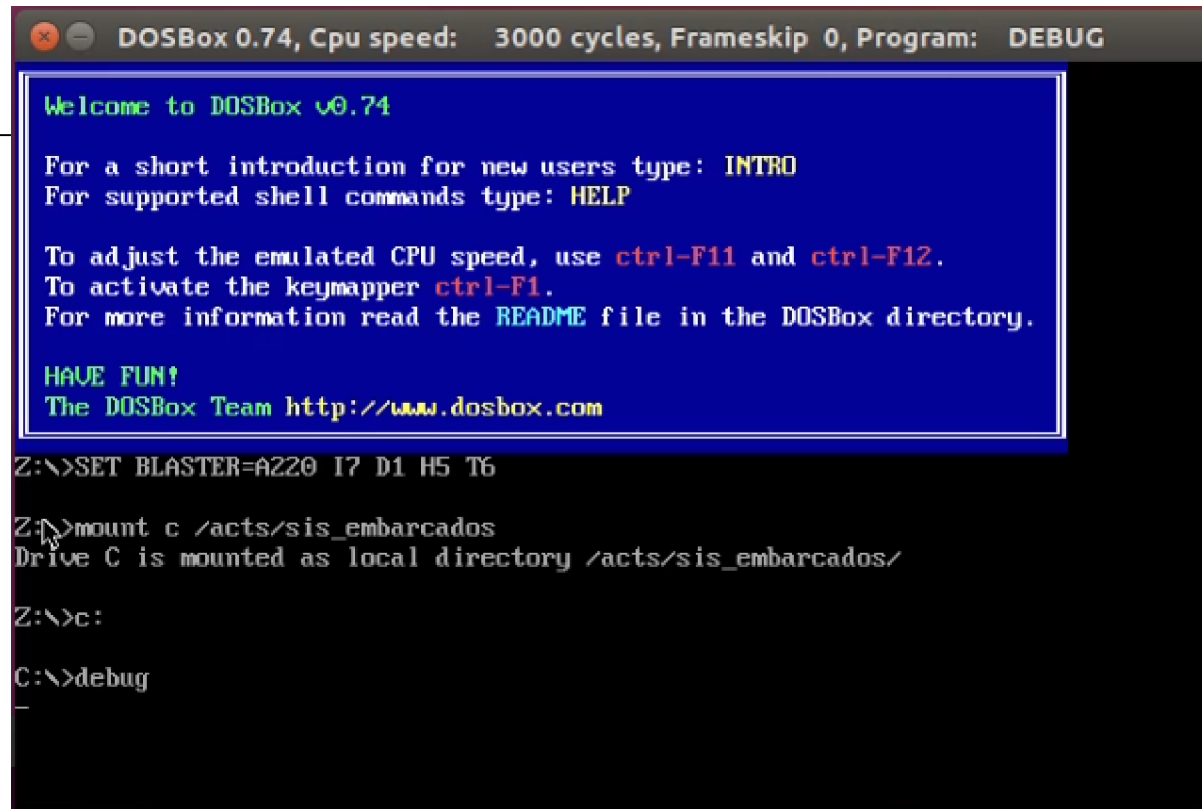
# Laboratório

# Laboratório

## 1- Entrar/sair do programa DEBUG:

- ❑ (a) Digite DEBUG na linha de comando do **DOSBox** para chamar o programa DEBUG. Um caracter – vai aparecer como *prompt* do DEBUG;

```
Z:\>mount c /acts/sis_embarcados↵  
Z:\>c:↵  
C:\>debug↵  
-
```



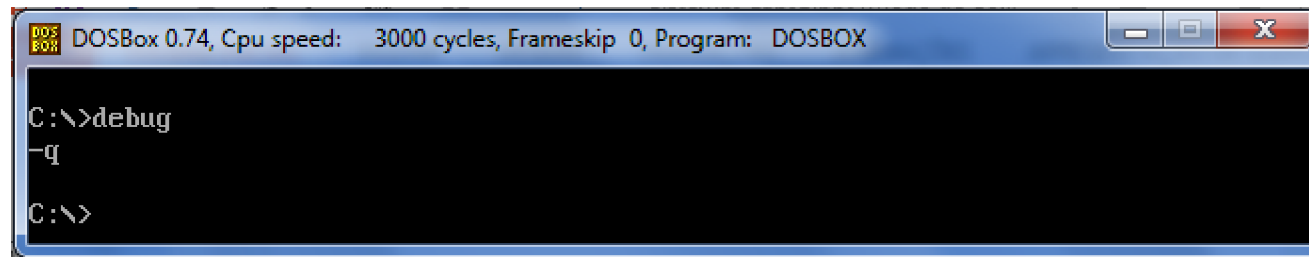
```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG  
Welcome to DOSBox v0.74  
For a short introduction for new users type: INTRO  
For supported shell commands type: HELP  
  
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.  
To activate the keymapper ctrl-F1.  
For more information read the README file in the DOSBox directory.  
  
HAVE FUN!  
The DOSBox Team http://www.dosbox.com  
  
Z:\>SET BLASTER=A220 I7 D1 H5 T6  
Z:\>mount c /acts/sis_embarcados  
Drive C is mounted as local directory /acts/sis_embarcados/  
Z:\>c:  
C:\>debug  
-
```

# Laboratório

1- Entrar/sair do programa DEBUG:

- ❑ (b) Digite a **tecla Q** para sair do programa DEBUG e voltar para o **DOSBox**.

-q ↵



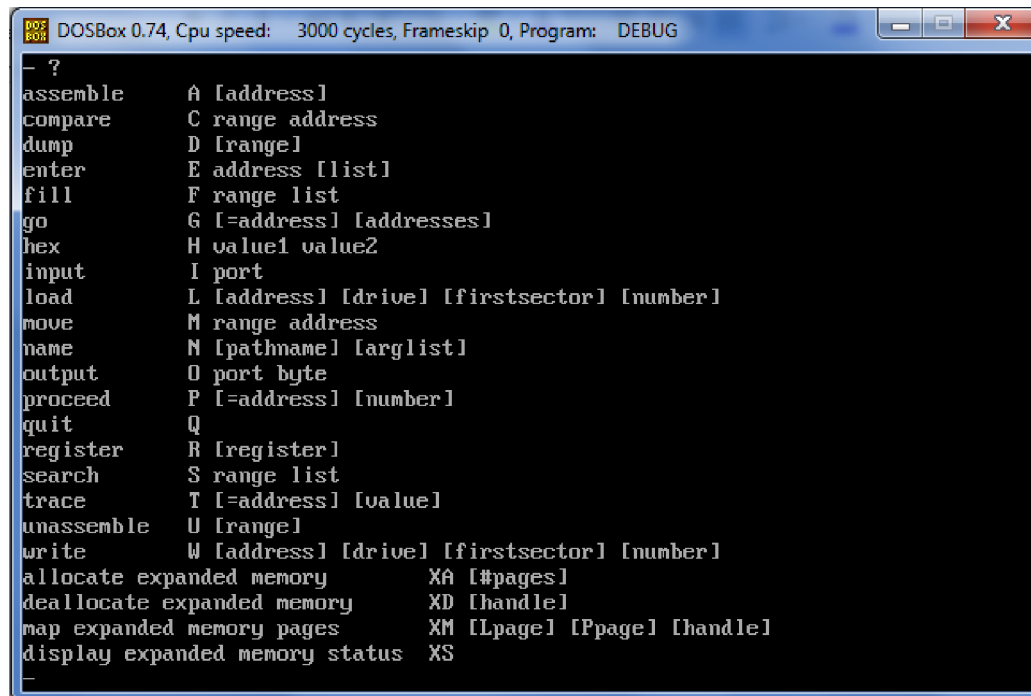
# Laboratório

## 2 – Listar comandos do DEBUG:

- ❑ (a) Com o programa DEBUG ativo, digite a tecla **?** seguida da tecla **<enter>** para listar um resumo dos principais comandos.

– ? ↵

ALTGR+W



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
- ?
assemble      A [address]
compare       C range address
dump          D [range]
enter         E address [list]
fill          F range list
go            G [=address] [addresses]
hex           H value1 value2
input         I port
load          L [address] [drive] [firstsector] [number]
move          M range address
name          N [pathname] [arglist]
output        O port byte
proceed       P [=address] [number]
quit          Q
register       R [register]
search        S range list
trace         T [=address] [value]
unassemble    U [range]
write         W [address] [drive] [firstsector] [number]
allocate expanded memory  XA [#pages]
deallocate expanded memory XD [handle]
map expanded memory pages XM [Lpage] [Ppage] [handle]
display expanded memory status XS
-
```

- ❑ **NOTA:** Não utilize o comando W, pois este comando pode escrever diretamente em trilha e setor do disco rígido podendo danificar os dados armazenados no seu computador.

# Laboratório

## 3 – Verificar e modificar conteúdo de registros:

- ❑ A verificação e modificação dos conteúdos dos registros é utilizado o comando R nas formas:

- (a) Digite o comando **-R**

- ❖ Lista o conteúdo de todos os registros.
- ❖ Observe que, além disso, também é apresentada a próxima instrução a ser executada (apontada por **CS:IP**).
- ❖ Observe ainda que, se um dos operandos da instrução estiver na memória, então o valor deste operando é também apresentado;

-R

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0100  NV UP EI PL NZ NA PO NC
073F:0100 C6C6C6      MOV     DH,C6
```

Conteúdo dos registradores

Próxima instrução

# Laboratório

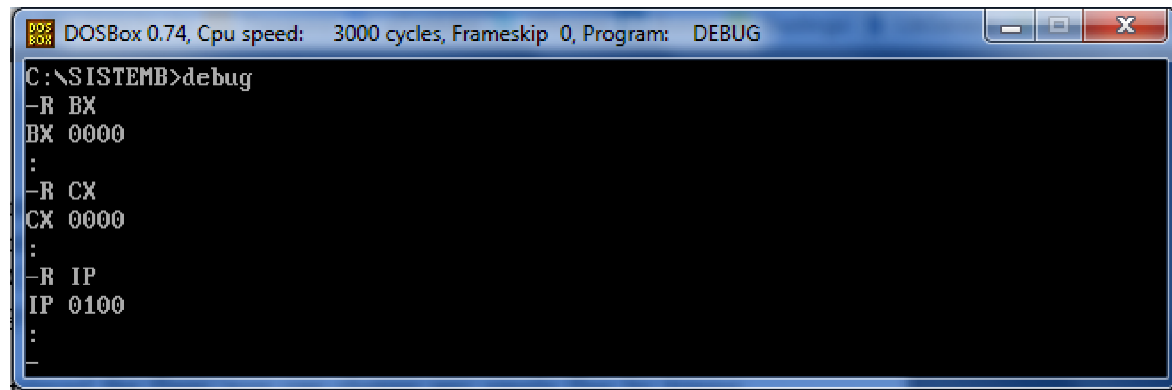
## 3 – Verificar e modificar conteúdo de registros:

- ❑ A verificação e modificação dos conteúdos dos registros é utilizado o comando R nas formas:

- **(b) Experimente para outros registros.**

- ❖ Digite **-R BX** ↵ Mostra o conteúdo atual do registro **BX**.
- ❖ Digite **-R CX** ↵ Mostra o conteúdo atual do registro **CX**.
- ❖ Digite **-R IP** ↵ Mostra o conteúdo atual do registro **IP**.

```
-R BX↵  
BX 0000  
:↵  
-R CX↵  
CX 0000  
:↵  
-R IP↵  
IP 0100  
:↵  
-
```



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG  
C:\SISTEMB>debug  
-R BX  
BX 0000  
:  
-R CX  
CX 0000  
:  
-R IP  
IP 0100  
:  
-
```



# Laboratório

## 3 – Verificar e modificar conteúdo de registros:

❑ A verificação e modificação dos conteúdos dos registros é utilizado o comando R nas formas:

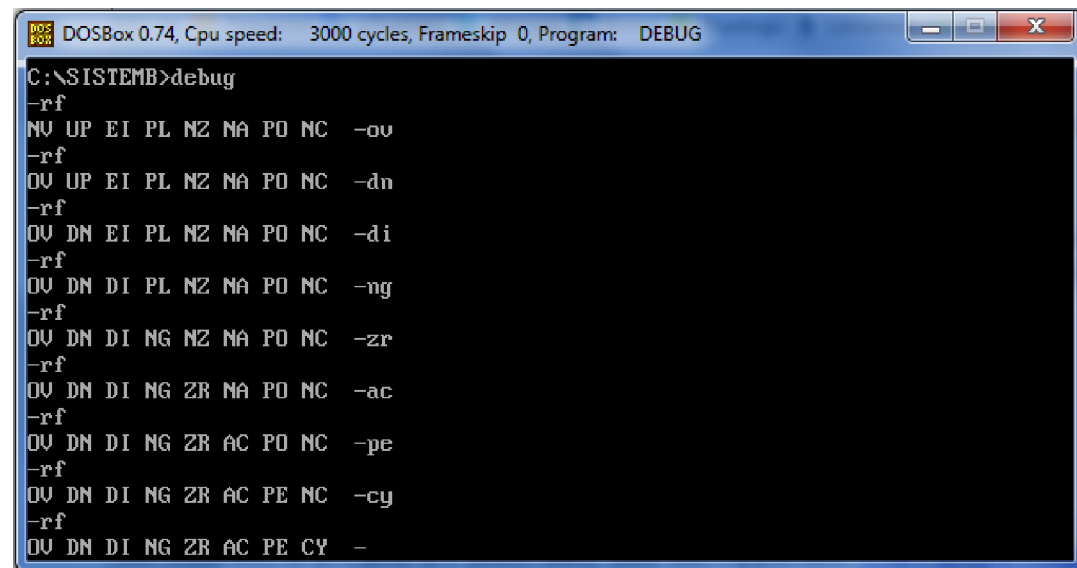
➤ (c) Digite o comando **-R F**

❖ Lista o conteúdo armazenado no registro de flags.

❖ Observe que logo após a listagem com o conteúdo de cada flag há um caracter **-**.

❖ Entre com o literal do(s) flag(s) que se deseja alterar, separados por espaços e terminado por ↵. Exemplo, se o conteúdo de **OF** é **NV**, então mude digitando **-OV**↵.

Flag	Conteúdo	Descrição
OF	OV	Ocorrência de overflow
(Overflow)	NV	Não ocorrência de overflow
DF	UP	Aumentou
(Direção)	DN	Diminuiu
IF	EI	Interrupções mascaráveis habilitadas
(Interrupção)	DI	Interrupções mascaráveis não habilitadas
SF (Sinal)	PL	Resultado positivo
	NG	Resultado Negativo
ZF (Zero)	ZR	Resultado nulo
	NZ	Resultado não nulo
AF (Transp. auxiliar)	AC	Ocorreu transporte (byte menos signif.)
	NA	Não ocorreu transporte (byte menos signif.)
PF (Paridade)	PE	Resultado par
	PO	Resultado ímpar
CF (Transporte)	CY	Ocorreu transporte (operação de word)
	NC	Não ocorreu transporte (operação de Word)



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-rf
NV UP EI PL NZ NA PO NC -ov
-rf
OV UP EI PL NZ NA PO NC -dn
-rf
OV DN EI PL NZ NA PO NC -di
-rf
OV DN DI PL NZ NA PO NC -ng
-rf
OV DN DI NG NZ NA PO NC -zr
-rf
OV DN DI NG ZR NA PO NC -ac
-rf
OV DN DI NG ZR AC PO NC -pe
-rf
OV DN DI NG ZR AC PE NC -cy
-rf
OV DN DI NG ZR AC PE CY -
```

# Laboratório

4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>,<offset fim>**↵

➤ onde os parâmetros:

❖ *segmento*, *offset início* e *offset fim* são opcionais.

❖ o parâmetro *segmento* pode ser tanto o nome de um registro de segmento (**DS**,**CS**,**ES** e **SS**), quanto um valor em hexadecimal.

---

# Laboratório

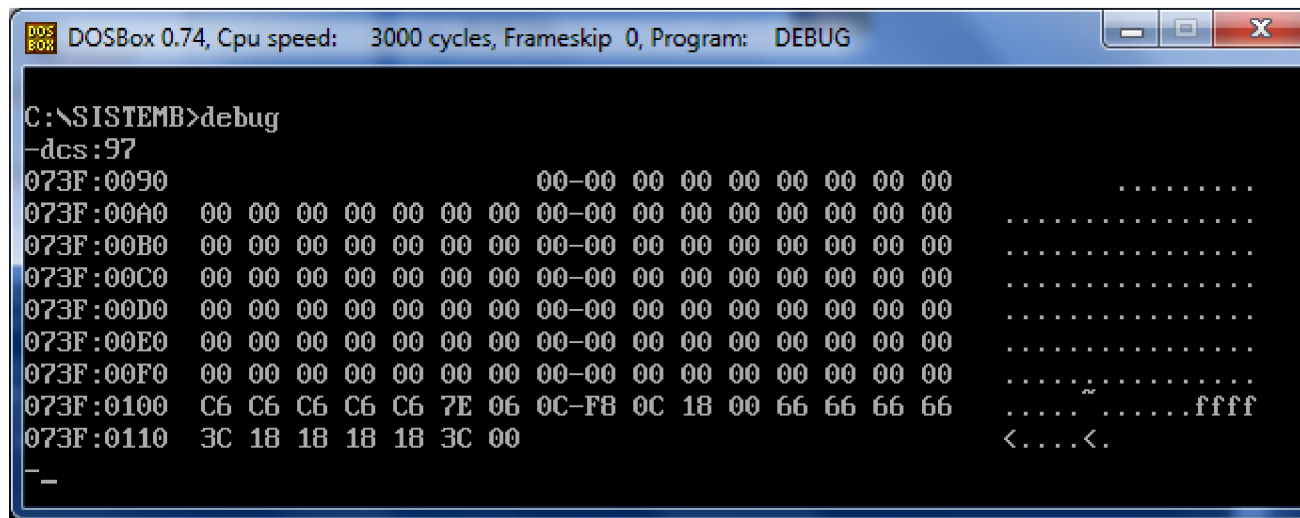
## 4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>,<offset fim>**↵

➤ **Exemplo 1:** Digite **-dcs:97**↵

❖ Isso vai exibir o conteúdo dos 128 bytes apontados pelo CS, começando da posição 97, ou seja, do nonagésimo oitavo byte do segmento;

-dcs:97↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-dcs:97
073F:0090 00-00 00 00 00 00 00 00 00 00 .....
073F:00A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00B0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00C0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:00F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:0100 C6 C6 C6 C6 C6 7E 06 0C-F8 0C 18 00 66 66 66 .....ffff
073F:0110 3C 18 18 18 18 3C 00 <....<
```

# Laboratório

## 4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>, <offset fim>**↵

➤ **Exemplo 1:** Digite **-dcs:97**↵

❖ Isso vai exibir o conteúdo dos 128 bytes apontados pelo **CS**, começando da posição 97, ou seja, do nonagésimo oitavo byte do segmento;

-dcs:97↵

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-dcs:97
073F:0090 00-00 00 00 00 00 00 00 00 00
073F:00A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
073F:00B0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
073F:00C0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
073F:00D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
073F:00E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
073F:00F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
073F:0100 C6 C6 C6 C6 C6 7E 06 0C-F8 0C 18 00 66 66 66 66
073F:0110 3C 18 18 18 18 3C 00
.....
.....
.....
.....
.....
.....
.....
.....ffff
<....<.
```

Endereço logico  
**SEGMENTO:OFFSET**

bytes em hexadecimal do  
parágrafo da memória solicitado

os caracteres ASCII correspondente  
aos bytes , se existir, senão exibe-se  
um ponto.

# Laboratório

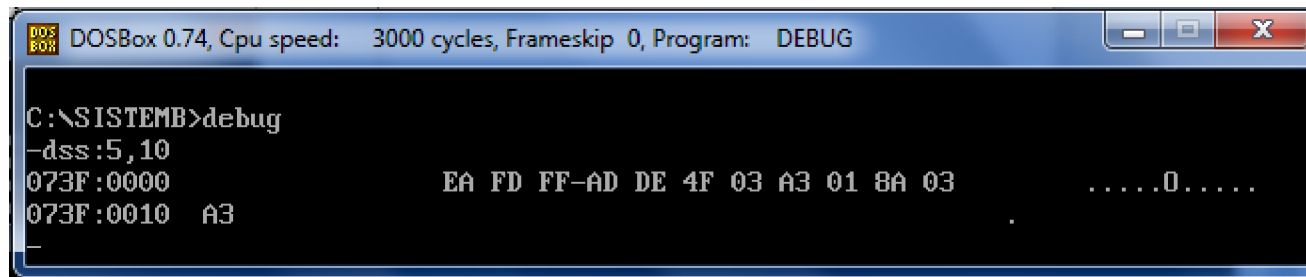
4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>,<offset fim>**↵

➤ **Exemplo 2:** Digite **-dss:5,10**↵

❖ Isso vai exibir o conteúdo dos doze bytes (5,6,7,8,9,A,B,C,D,E,F,10) bytes apontados pelo **SS**, ou seja, do sexto ao décimo sétimo byte;

-dss:5,10↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-dss:5,10
073F:0000      EA FD FF AD DE 4F 03 A3 01 8A 03      .....0.....
073F:0010  A3
```

# Laboratório

4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>,<offset fim>**↵

➤ **Exemplo 2:** Digite **-dss:5,10**↵

❖ Isso vai exibir o conteúdo dos doze bytes (5,6,7,8,9,A,B,C,D,E,F,10) bytes apontados pelo **SS**, ou seja, do sexto ao décimo sétimo byte;

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\>debug
-dss:5,5
073F:0000      EA      .
-dss:5,6
073F:0000      EA FD    ..
-dss:5,7
073F:0000      EA FD FF   ...
-dss:5,8
073F:0000      EA FD FF-AD  ....
-dss:5,9
073F:0000      EA FD FF-AD DE  ....
-dss:5,A
073F:0000      EA FD FF-AD DE 4F  ....0
-dss:5,B
073F:0000      EA FD FF-AD DE 4F 03  ....0.
-dss:5,C
073F:0000      EA FD FF-AD DE 4F 03 A3  ....0..
-dss:5,D
073F:0000      EA FD FF-AD DE 4F 03 A3 01  ....0...
-dss:5,E
073F:0000      EA FD FF-AD DE 4F 03 A3 01 8A  ....0....
-dss:5,F
073F:0000      EA FD FF-AD DE 4F 03 A3 01 8A 03  ....0.....
-dss:5,10
073F:0000      EA FD FF-AD DE 4F 03 A3 01 8A 03  ....0.....
073F:0010  A3      .
```

# Laboratório

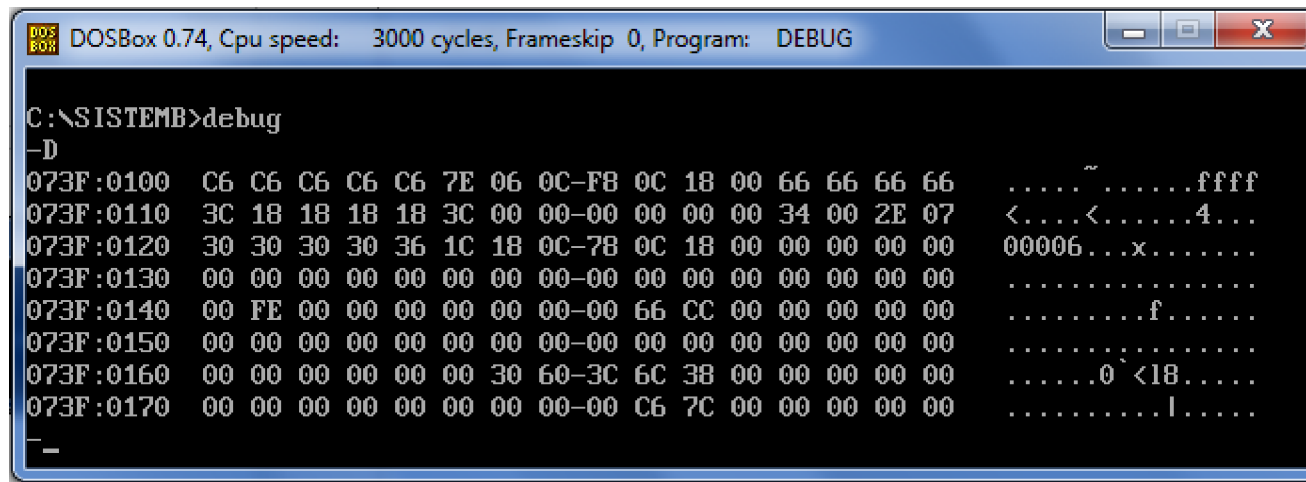
4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>,<offset fim>**↵

➤ **Exemplo 3:** Digite **-d**↵

❖ Isso vai exibir o conteúdo dos 128 bytes apontados pelo ultimo segmento verificado (no caso **SS**).

-d↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG

C:\SISTEMB>debug
-d
073F:0100  C6 C6 C6 C6 C6 7E 06 0C-F8 0C 18 00 66 66 66 66  .....~.....ffff
073F:0110  3C 18 18 18 18 3C 00 00-00 00 00 00 34 00 2E 07  <....<.....4...
073F:0120  30 30 30 30 36 1C 18 0C-78 0C 18 00 00 00 00 00  00006...x.....
073F:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
073F:0140  00 FE 00 00 00 00 00 00-00 66 CC 00 00 00 00  .....f.....
073F:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
073F:0160  00 00 00 00 00 00 30 60-3C 6C 38 00 00 00 00  .....0`<18....
073F:0170  00 00 00 00 00 00 00 00-00 C6 7C 00 00 00 00  .....!.....
_
```

# Laboratório

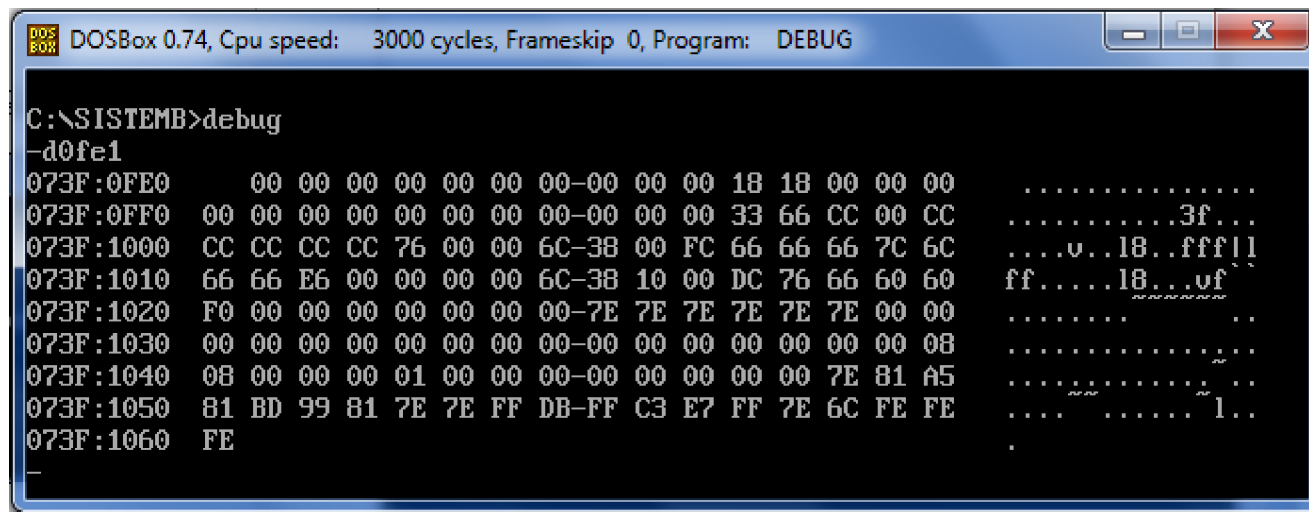
4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>,<offset fim>**↵

➤ **Exemplo 4:** Digite **-d0fe1**↵

❖ Isso vai exibir o conteúdo dos 128 bytes apontados pelo **DS**, começando pelo byte da posição **0fe1**.

-d0fe1↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-d0fe1
073F:0FE0  00 00 00 00 00 00 00 00-00 00 00 18 18 00 00 00  .....
073F:0FF0  00 00 00 00 00 00 00 00-00 00 00 33 66 CC 00 CC  .....3f...
073F:1000  CC CC CC CC 76 00 00 6C-38 00 FC 66 66 66 7C 6C  ....v..18..fff!l
073F:1010  66 66 E6 00 00 00 00 6C-38 10 00 DC 76 66 60 60  ff.....18...vf
073F:1020  F0 00 00 00 00 00 00 00-7E 7E 7E 7E 7E 7E 00 00  .....
073F:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 08  .....
073F:1040  08 00 00 00 01 00 00 00-00 00 00 00 00 7E 81 A5  .....
073F:1050  81 BD 99 81 7E 7E FF DB-FF C3 E7 FF 7E 6C FE FE  .... 1..
073F:1060  FE
```



# Laboratório

## 4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>,<offset fim>**↵

➤ **Exemplo 4:** Digite **-d0fe1**↵

❖ Isso vai exibir o conteúdo dos 128 bytes apontados pelo **DS**, começando pelo byte da posição **0fe1**.

`-d0fe1↵`

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-d0fe1
073F:0FE0  00 00 00 00 00 00 00 00-00 00 00 18 18 00 00 00  .....
073F:0FF0  00 00 00 00 00 00 00 00-00 00 00 33 66 CC 00 CC  .....3f...
073F:1000  CC CC CC CC 76 00 00 6C-38 00 FC 66 66 66 7C 6C  ....v..18..fff!l
073F:1010  66 66 E6 00 00 00 00 6C-38 10 00 DC 76 66 60 60  ff.....18...vf
073F:1020  F0 00 00 00 00 00 00 00-7E 7E 7E 7E 7E 7E 00 00  .....
073F:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 08  .....
073F:1040  08 00 00 00 01 00 00 00-00 00 00 00 7E 81 A5  .....
073F:1050  81 BD 99 81 7E 7E FF DB-FF C3 E7 FF 7E 6C FE FE  ....1..
073F:1060  FE
```

# Laboratório

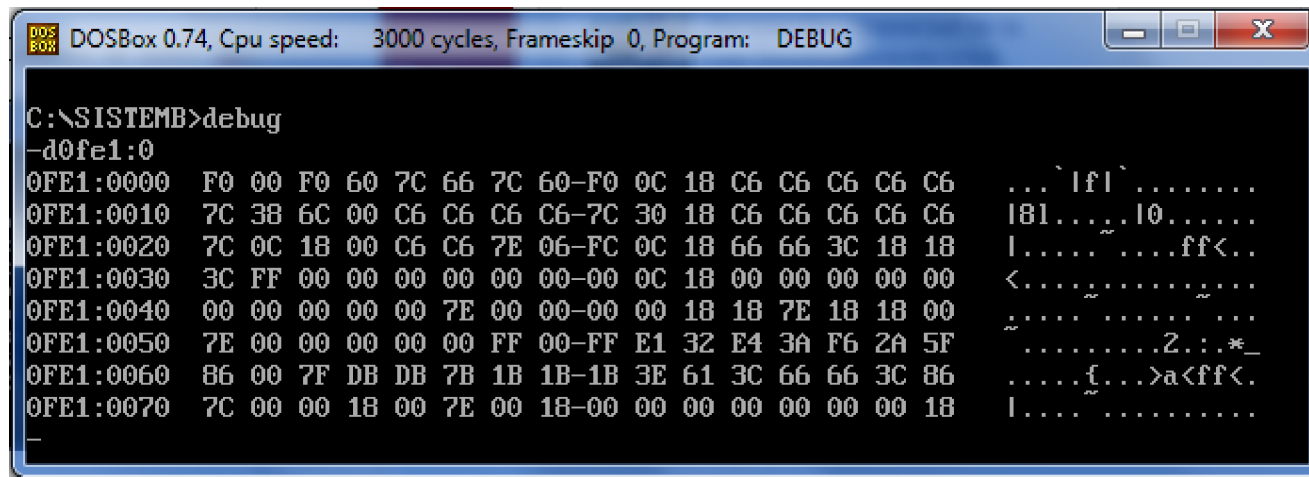
4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>,<offset fim>**↵

➤ **Exemplo 5:** Digite **-d0fe1:0**↵

❖ Isso vai exibir o conteúdo dos 128 bytes armazenados no segmento **0fe1h**, começando pelo byte da posição **0h**.

-d0fe1:0↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-d0fe1:0
0FE1:0000  F0 00 F0 60 7C 66 7C 60-F0 0C 18 C6 C6 C6 C6 C6  ...`lfI`.....
0FE1:0010  7C 38 6C 00 C6 C6 C6 C6-7C 30 18 C6 C6 C6 C6 C6  181.....l0.....
0FE1:0020  7C 0C 18 00 C6 C6 7E 06-FC 0C 18 66 66 3C 18 18  l.....~....ff<..
0FE1:0030  3C FF 00 00 00 00 00 00-00 0C 18 00 00 00 00 00  <.....~.....
0FE1:0040  00 00 00 00 00 7E 00 00-00 00 18 18 7E 18 18 00  ~.....~.....
0FE1:0050  7E 00 00 00 00 00 FF 00-FF E1 32 E4 3A F6 2A 5F  .....2...*_
0FE1:0060  86 00 7F DB DB 7B 1B 1B-1B 3E 61 3C 66 66 3C 86  .....{...>a<ff<.
0FE1:0070  7C 00 00 18 00 7E 00 18-00 00 00 00 00 00 00 18  l.....~.....
```

# Laboratório

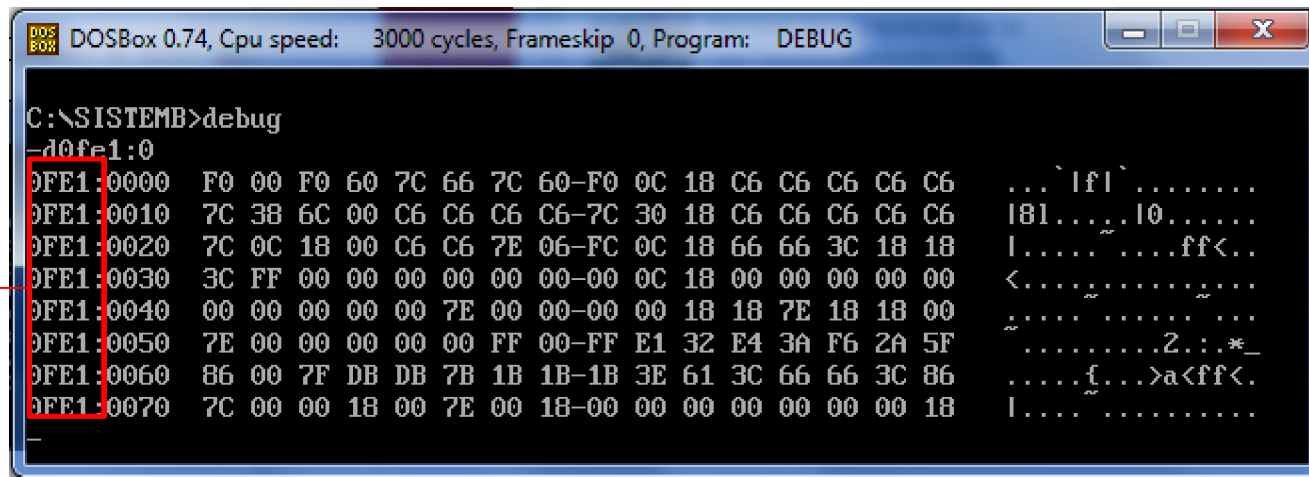
4 – Verificar o conteúdo de um segmento da memória:

❑ Digite **D<segmento>:<offset início>,<offset fim>**↵

➤ **Exemplo 5:** Digite **-d0fe1:0**↵

❖ Isso vai exibir o conteúdo dos 128 bytes armazenados no segmento **0fe1h**, começando pelo byte da posição **0h**.

-d0fe1:0↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-d0fe1:0
0FE1:0000  F0 00 F0 60 7C 66 7C 60-F0 0C 18 C6 C6 C6 C6 ...`lfl`.....
0FE1:0010  7C 38 6C 00 C6 C6 C6 C6-7C 30 18 C6 C6 C6 C6 181.....l0.....
0FE1:0020  7C 0C 18 00 C6 C6 7E 06-FC 0C 18 66 66 3C 18 18 l.....~....ff<..
0FE1:0030  3C FF 00 00 00 00 00 00-00 0C 18 00 00 00 00 00 <.....~.....
0FE1:0040  00 00 00 00 00 7E 00 00-00 00 18 18 7E 18 18 00 ~.....~.....
0FE1:0050  7E 00 00 00 00 00 FF 00-FF E1 32 E4 3A F6 2A 5F .....2...*_
0FE1:0060  86 00 7F DB DB 7B 1B 1B-1B 3E 61 3C 66 66 3C 86 .....{...>a<ff<..
0FE1:0070  7C 00 00 18 00 7E 00 18-00 00 00 00 00 00 00 18 l.....~.....
```

# Laboratório

## 5 – Verificar e alterar o conteúdo de uma posição de memória

❑ Digite **E**<segmento>:<offset início>↵

➤ onde o parâmetro:

❖ **segmento** é opcional (**DS** é o segmento *default*).

➤ Observe que o comando **E** é importante para iniciar ou alterar valores de variáveis na memória.

---

# Laboratório

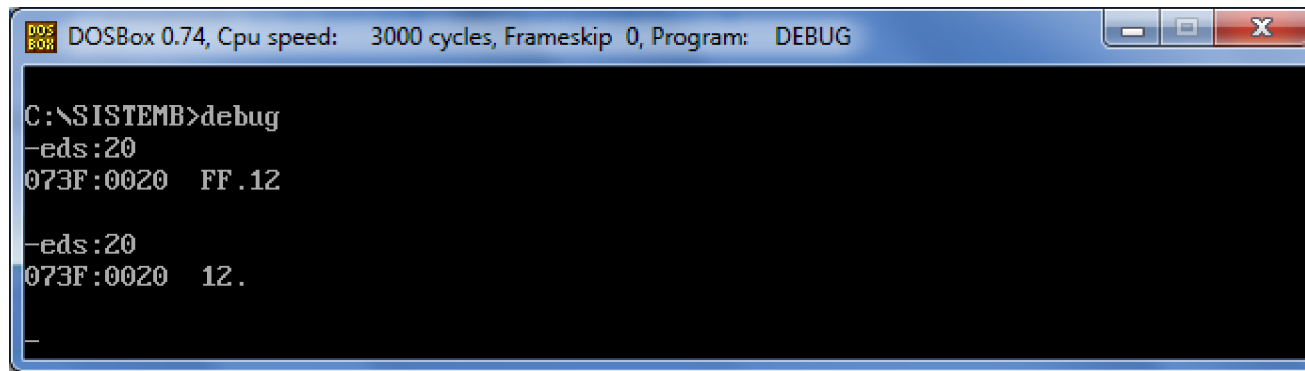
## 5 – Verificar e alterar o conteúdo de uma posição de memória

❑ Digite **E<segmento>:<offset início>**↵

➤ **Exemplo 1:** Digite **—eds:20**↵

❖ Isso vai exibir o conteúdo do byte contido na posição 20 do segmento de memória apontado por **DS**. Logo após, é exibido um *prompt* para entrada de um novo valor para o byte exibido. Entre com o valor em hexadecimal e tecle ↵ caso deseje alterá-lo, senão tecle ↵ simplesmente;

—eds:20↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG

C:\SISTEMB>debug
-eds:20
073F:0020  FF.12

-eds:20
073F:0020  12.
```

# Laboratório

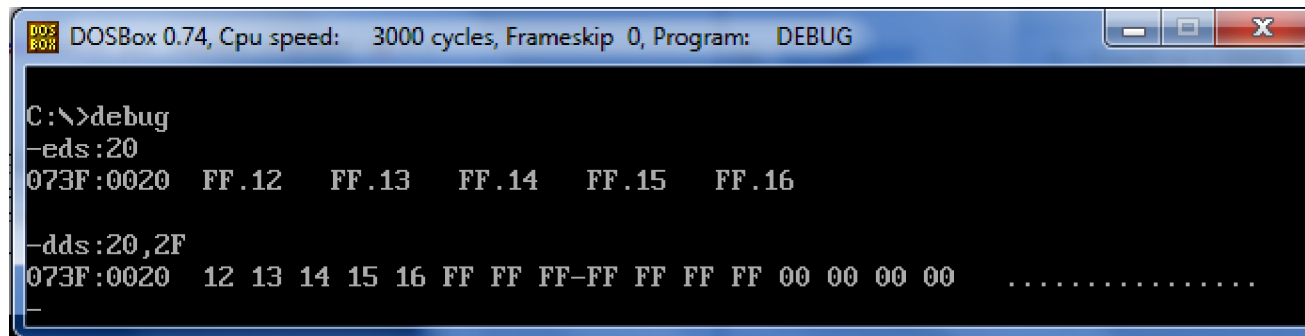
## 5 – Verificar e alterar o conteúdo de uma posição de memória

❑ Digite **E<segmento>:<offset início>**↵

➤ **Exemplo 1:** Digite **–eds:20**↵

❖ Em vez de alterar o valor do byte, tecle **SPACE**. Observe que o conteúdo do byte da posição 21 do **DS** passa a ser exibido, podendo também ser alterado. Verifique a correspondência do valor do byte exibido com o comando **D**, caso não esteja convencido.

–eds:20↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\>debug
-eds:20
073F:0020  FF.12  FF.13  FF.14  FF.15  FF.16
-dds:20,2F
073F:0020  12 13 14 15 16 FF FF FF FF FF FF FF 00 00 00 00 .....
```

# Laboratório

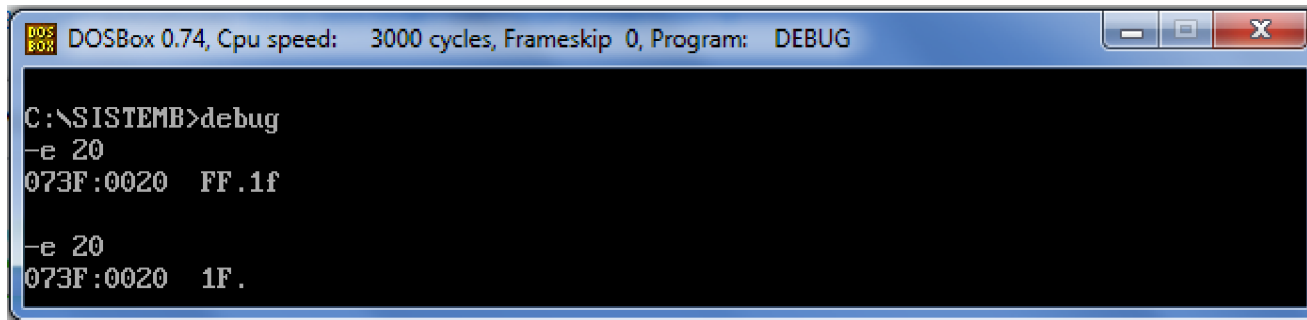
## 5 – Verificar e alterar o conteúdo de uma posição de memória

❑ Digite **E<segmento>:<offset início>**↵

➤ **Exemplo 2:** Digite **–e 20**↵

❖ Observe o mesmo efeito provocado pelo comando anterior (**DS** é o segmento por *default*);

–e 20↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
–e 20
073F:0020 FF.1f
–e 20
073F:0020 1F.
```

# Laboratório

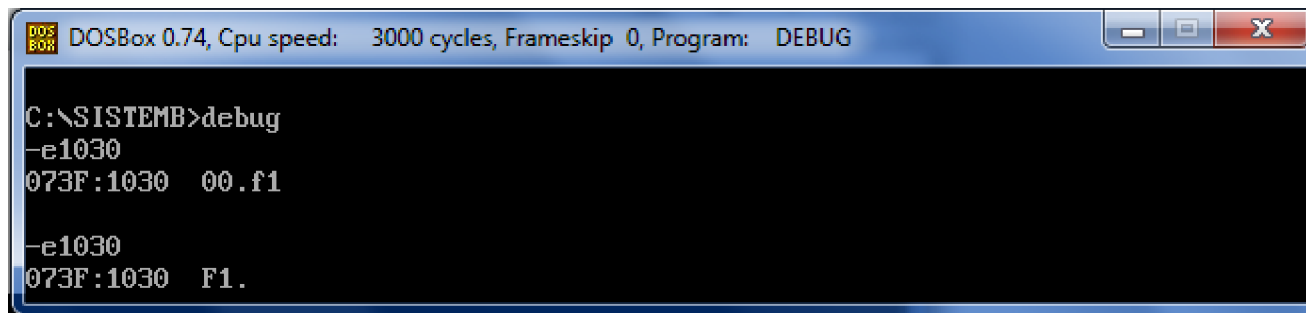
## 5 – Verificar e alterar o conteúdo de uma posição de memória

❑ Digite **E<segmento>:<offset início>**↵

➤ **Exemplo 3:** Digite **-e1030**↵

❖ Isso exibe o conteúdo do primeiro byte apontado pela posição 1030 do segmento **DS**, permitindo sua posterior alteração;

-e1030↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG

C:\SISTEMB>debug
-e1030
073F:1030  00.f1

-e1030
073F:1030  F1.
```



# Laboratório

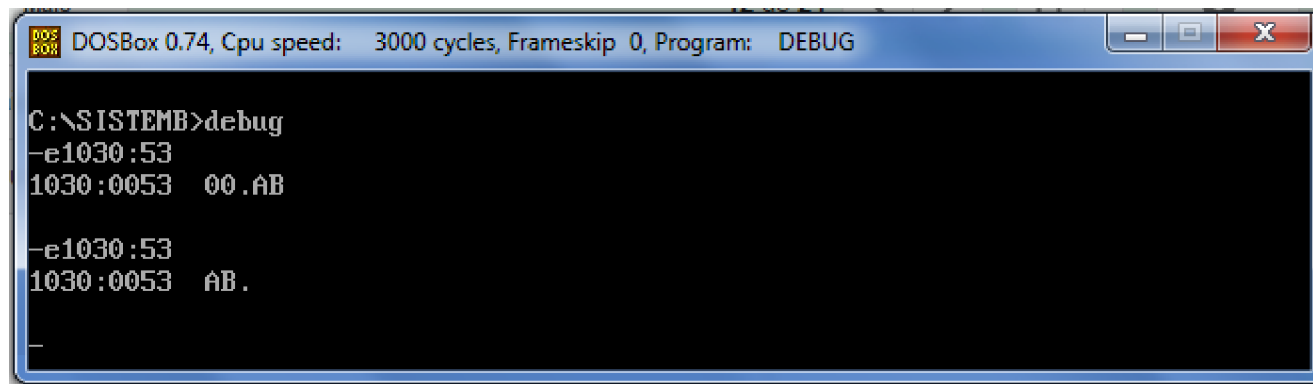
## 5 – Verificar e alterar o conteúdo de uma posição de memória

❑ Digite **E<segmento>:<offset início>**↵

➤ **Exemplo 4:** Digite **-e1030:53**↵

❖ Isso exibe o conteúdo do byte da posição 53 do segmento 1030, permitindo sua posterior alteração.

-e1030:53↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-e1030:53
1030:0053  00.AB

-e1030:53
1030:0053  AB.
```

# Laboratório

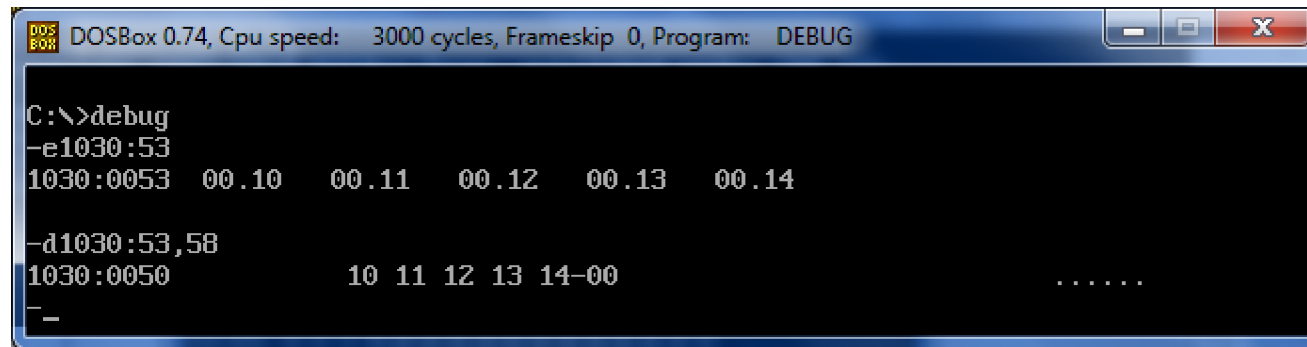
## 5 – Verificar e alterar o conteúdo de uma posição de memória

❑ Digite **E<segmento>:<offset início>**

➤ **Exemplo 4:** Digite **-e1030:53**

❖ Em vez de alterar o valor do byte, tecle **SPACE**. Observe que o conteúdo do byte da posição 53 passa a ser exibido, podendo também ser alterado. Verifique a correspondência do valor do byte exibido com o comando **D**, caso não esteja convencido.

-e1030:53



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\>debug
-e1030:53
1030:0053 00.10 00.11 00.12 00.13 00.14
-d1030:53,58
1030:0050 10 11 12 13 14-00
.....
```

# Laboratório

## 5- Assemblar/Desassemblar um programa:

❑ (a) Para **ASSEMBLAR** um programa use o comando

**A<segmento>:<offset início>**↵

➤ onde o parâmetro:

❖ **segmento** é opcional (**CS** é o default). Isso faz com que o programa DEBUG entre com a posição onde a instrução será inserida.

➤ **Procedimento:**

❖ Entre com a instrução e tecle ↵.

❖ Isso faz com que o programa DEBUG entre com a posição seguinte, onde uma nova instrução poderá ser inserida.

❖ Caso nenhuma nova instrução deva ser inserida, tecle ↵ simplesmente, finalizando o comando **A**.

---

# Laboratório

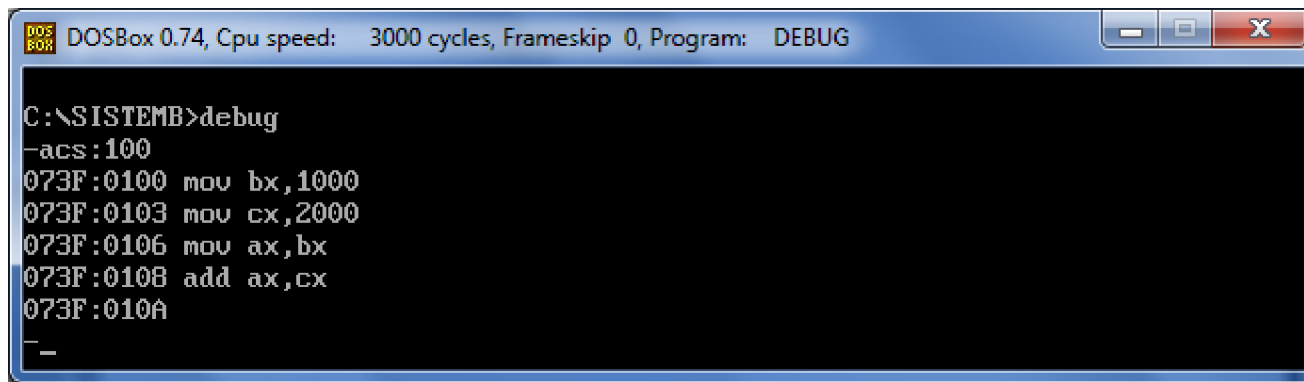
## 5- Assemblar/Desassemblar um programa:

- ❑ (a) Para **ASSEMBLAR** um programa use o comando

**A<segmento>:<offset início>**

- **Exemplo:** Entre com o seguinte programa na posição **CS:100**:

```
MOV  BX,1000
MOV  CX,2000
MOV  AX,BX
ADD  AX,CX
```



The screenshot shows a DOSBox window titled "DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG". The command prompt shows the following sequence of commands and output:

```
C:\SISTEMB>debug
-acs:100
073F:0100 mov bx,1000
073F:0103 mov cx,2000
073F:0106 mov ax,bx
073F:0108 add ax,cx
073F:010A
_
```

# Laboratório

## 5- Assemblar/Desassemblar um programa:

- ❑ (b) Para **DESASSEMBLAR** um programa use o comando  
**U<segmento>:<offset início>, <offset fim>**↵
    - onde os parâmetros:
      - ❖ **segmento**, *offset início* e *offset fim* são opcionais.
    - O comando **U** lista em cada linha,
      - ❖ a posição de memória onde cada instrução é colocada,
      - ❖ os bytes em hexadecimal que compõe a instrução e
      - ❖ mnemônico da instrução,
    - permitindo a verificação de um programa recém assemblado.
-

# Laboratório

## 5- Assemblar/Desassemblar um programa:

❑ (b) Para **DESASSEMBLAR** um programa use o comando

**U<segmento>:<offset início>, <offset fim>**

- **Exemplo:** Experimente entrar com o comando **-ucs:100** ou simplesmente **-u100** para verificar se o programa do exemplo anterior foi entrado corretamente.
- As outras instruções que aparecem é a codificação dos bytes existentes na memória.

`-ucs:100`

Endereço lógico de cada instrução  
**SEGMENTO:OFFSET**

Bytes em hexadecimal da instrução

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG

```
C:\SISTEMB>debug
-ucs:100
073F:0100 BB0010
073F:0103 B90020
073F:0106 89D8
073F:0108 01C8
073F:010A 1800
073F:010C 66
073F:010D 66
073F:010E AE
073F:010F FE00
073F:0111 F0
073F:0112 46
073F:0113 7400
073F:0115 00B200B2
073F:0119 1099002E
073F:011D 07
073F:011E 2E
073F:011F 07
-
```

MOV	BX,1000
MOV	CX,2000
MOV	AX,BX
ADD	AX,CX
SBB	[BX+SI],AL
DB	66
DB	66
SCASB	
INC	BYTE PTR [BX+SI]
LOCK	
INC	SI
JZ	0115
ADD	[BP+SI+B200],DH
ADC	[BX+DI+2E00],BL
POP	ES
CS:	
POP	ES

Instruções

# Laboratório

## 6 – Execução passo a passo de um programa:

- ❑ (a) Para executar um programa no DEBUG é preciso, primeiramente, garantir que o par **CS:IP** esteja apontando para a primeira instrução do programa a ser executado.
- ❑ Use o comando **R** para verificar se isto acontece e se não, use o mesmo comando **R** para alterar o **IP** para apontar para a primeira instrução(ou o parâmetro **<=endereço>**);

IP na posição certa  
0100

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG

C:\SISTEMB>debug
-u100
073F:0100 BB0010      MOV     BX,1000
073F:0103 B90020      MOV     CX,2000
073F:0106 89D8        MOV     AX,BX
073F:0108 01C8        ADD     AX,CX
073F:010A 1800        SBB     [BX+SI],AL
073F:010C 66             DB      66
073F:010D 66             DB      66
073F:010E AE             SCASB
073F:010F FE00        INC     BYTE PTR [BX+SI]
073F:0111 F0             LOCK
073F:0112 46             INC     SI
073F:0113 7400        JZ      0115
073F:0115 00B200B2      ADD     [BP+SI+B200],DH
073F:0119 10990034      ADC     [BX+DI+3400],BL
073F:011D 002E0730      ADD     [3007],CH
-R IP
IP 0100
:
```

# Laboratório

## 6 – Execução passo a passo de um programa:

- ❑ (a) Para executar um programa no DEBUG é preciso, primeiramente, garantir que o par **CS:IP** esteja apontando para a primeira instrução do programa a ser executado.
- ❑ Use o comando **R** para verificar se isto acontece e se não, use o mesmo comando **R** para alterar o **IP** para apontar para a primeira instrução(ou o parâmetro **<=endereço>**);

IP na posição errada  
0200 e corrigida com o  
comando **R**.

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
073F:0106 89D8      MOV     AX,BX
073F:0108 01C8      ADD     AX,CX
073F:010A 1800      SBB     [BX+SI],AL
073F:010C 66        DB     66
073F:010D 66        DB     66
073F:010E AE        SCASB
073F:010F FE00      INC     BYTE PTR [BX+SI]
073F:0111 F0        LOCK
073F:0112 46        INC     SI
073F:0113 7400      JZ      0115
073F:0115 00B200B2  ADD     [BP+SI+B200],DH
073F:0119 10990034  ADC     [BX+DI+3400],BL
073F:011D 002E0730  ADD     [3007],CH
-R IP
IP 0200
:0100
-R IP
IP 0100
:
```



# Laboratório

## 6 – Execução passo a passo de um programa:

❑ (b) A seguir, digite

**T <=endereço>,<número de instruções>↵**

➤ onde os parâmetros:

❖ endereço e número de instruções são opcionais.

❖ Observe que

- o parâmetro endereço deve ser usado quando o IP não estiver apontando para a posição da primeira instrução a ser executada.
- o parâmetro número de instruções faz o processador executar esta quantidade de instruções.

➤ Entre cada instrução executada, o comando mostra o conteúdo de todos os registros (comando R implícito).

---

# Laboratório

## 6 – Execução passo a passo de um programa:

❑ (b) A seguir, digite

**T <=endereço>,<número de instruções>**

➤ **Exemplo 1:** Experimente executar o programa entrado no item 5 com o comando **-T=100,4**;

**-T=100,4**

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-T=100,4

AX=0000 BX=1000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0103  NU UP EI PL NZ NA PO NC
073F:0103 B90020          MOV     CX,2000

AX=0000 BX=1000 CX=2000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0106  NU UP EI PL NZ NA PO NC
073F:0106 89DB          MOV     AX,BX

AX=1000 BX=1000 CX=2000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0108  NU UP EI PL NZ NA PO NC
073F:0108 01CB          ADD     AX,CX

AX=3000 BX=1000 CX=2000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=010A  NU UP EI PL NZ NA PE NC
073F:010A 1800          SBB     [BX+SI],AL      DS:1000=CC
-
```

# Laboratório

## 6 – Execução passo a passo de um programa:

❑ (b) A seguir, digite

**T <=endereço>,<número de instruções>↵**

➤ **Exemplo 2:** Mude o programa para carregar outros valores nos registros e teste novamente.;

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG

```
C:\SISTEMB>debug
-acs:100
073F:0100 mov bx,1ff1
073F:0103 mov cx,200e
073F:0106 mov ax,bx
073F:0108 add ax,cx
073F:010A
-T=100.4
```

AX=0000	BX=1FF1	CX=0000	DX=0000	SP=00FD	BP=0000	SI=0000	DI=0000
DS=073F	ES=073F	SS=073F	CS=073F	IP=0103	NV UP EI PL NZ NA PO NC		

```
073F:0103 B90E20      MOV     CX,200E
```

AX=0000	BX=1FF1	CX=200E	DX=0000	SP=00FD	BP=0000	SI=0000	DI=0000
DS=073F	ES=073F	SS=073F	CS=073F	IP=0106	NV UP EI PL NZ NA PO NC		

```
073F:0106 89D8      MOV     AX,BX
```

AX=1FF1	BX=1FF1	CX=200E	DX=0000	SP=00FD	BP=0000	SI=0000	DI=0000
DS=073F	ES=073F	SS=073F	CS=073F	IP=0108	NV UP EI PL NZ NA PO NC		

```
073F:0108 01CB      ADD     AX,CX
```

AX=3FFF	BX=1FF1	CX=200E	DX=0000	SP=00FD	BP=0000	SI=0000	DI=0000
DS=073F	ES=073F	SS=073F	CS=073F	IP=010A	NV UP EI PL NZ NA PE NC		

```
073F:010A 1800      SBB     [BX+SI],AL      DS:1FF1=00
```

Estado dos  
registros  
depois da  
execução do  
instrução  
atual

Seguinte  
instrução a  
executar

# Laboratório

## 7 – Execução direta de um programa:

- ❑ Outra maneira de executar um programa é com o comando **G**.
  - ❑ Este comando executa o programa direto até encontrar um terminador do programa ou executa o programa **até encontrar um *breakpoint* definido no comando**.
    - Para o DEBUG, o terminador do programa é a instrução **INT3**.
  - ❑ O formato geral do comando G é  
**–G<=endereço>,<endereços>↵**
    - onde
      - ❖ **endereço** é o IP inicial e
      - ❖ **endereços** é um *breakpoint*.
-

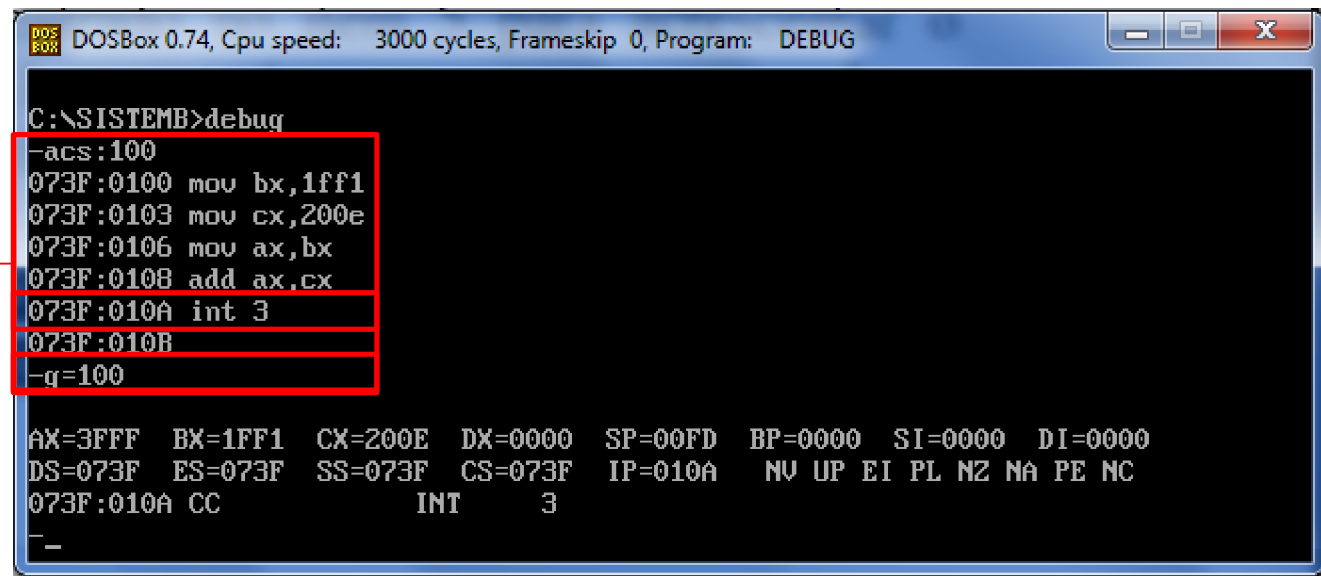
# Laboratório

## 7 – Execução direta de um programa:

❑ Outra maneira de executar um programa é com o comando **G**.

➤ **Exemplo 1:** Modifique o programa entrado no item 5 para acrescentar o terminador de programa **INT3**. Logo após, execute o comando **-G=100** e verifique o resultado final com o comando **R**.

Ingressar o programa e  
executar o comando  
**G=100**



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SISTEMB>debug
-acs:100
073F:0100 mov bx,1ff1
073F:0103 mov cx,200e
073F:0106 mov ax,bx
073F:0108 add ax,cx
073F:010A int 3
073F:010B
-g=100

AX=3FFF BX=1FF1 CX=200E DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=010A NV UP EI PL NZ NA PE NC
073F:010A CC INT 3
-
```

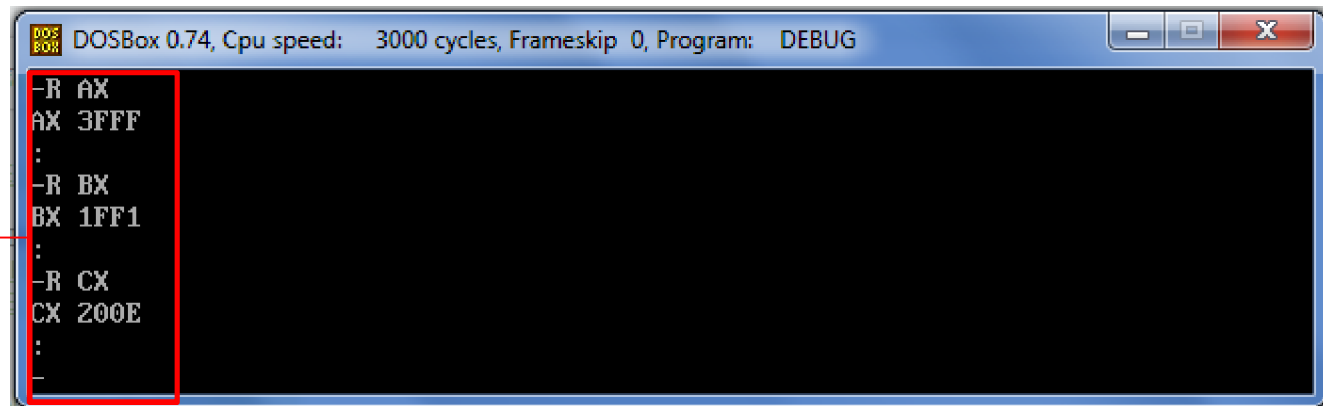
# Laboratório

## 7 – Execução direta de um programa:

❑ Outra maneira de executar um programa é com o comando **G**.

➤ **Exemplo 1:** Modifique o programa entrado no item 5 para acrescentar o terminador de programa **INT3**. Logo após, execute o comando **-G=100** e verifique o resultado final com o comando **R**.

Verificar o resultado  
com o comando **R**



The screenshot shows a DOSBox 0.74 window with the title bar 'DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG'. The command prompt shows the following output:

```
-R AX
AX 3FFF
:
-R BX
BX 1FF1
:
-R CX
CX 200E
:
-
```

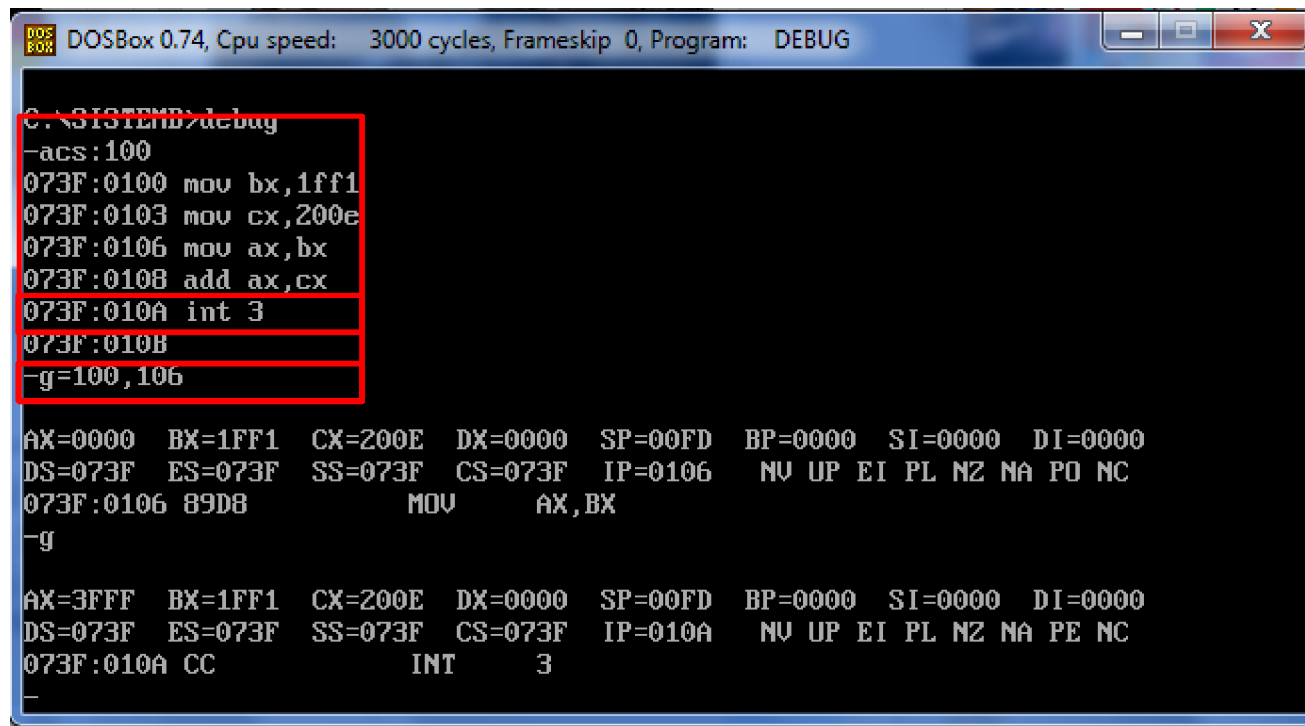
A red box highlights the register values, and a red arrow points from the text 'Verificar o resultado com o comando R' to the first line of the output.

# Laboratório

## 7 – Execução direta de um programa:

❑ Outra maneira de executar um programa é com o comando **G**.

➤ **Exemplo 2:** Entre com o comando **-G=100,106** e verifique o resultado.  
Termine o programa com o comando **-G** e verifique o resultado.



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\SYSTEM32>debug
-acs:100
073F:0100 mov bx,1ff1
073F:0103 mov cx,200e
073F:0106 mov ax,bx
073F:0108 add ax,cx
073F:010A int 3
073F:010B
-g=100,106

AX=0000 BX=1FF1 CX=200E DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0106  NV UP EI PL NZ NA PO NC
073F:0106 89D8      MOV     AX,BX
-g

AX=3FFF BX=1FF1 CX=200E DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=010A  NV UP EI PL NZ NA PE NC
073F:010A CC      INT     3
-
```

# Laboratório

## 9 – Exercício

- ❑ Se tiver tempo, teste outros comandos do DEBUG tais como
    - -C: Comparação
    - -F: Iniciar bloco de memória (FILL)
    - -H: Somar e subtrair números
    - -M: Mover blocos de memória.
  - ❑ Não utilize o comando **W** pois este comando pode escrever diretamente em trilha e setor do disco rígido podendo danificar os dados armazenados no seu computador.
-



# Laboratório

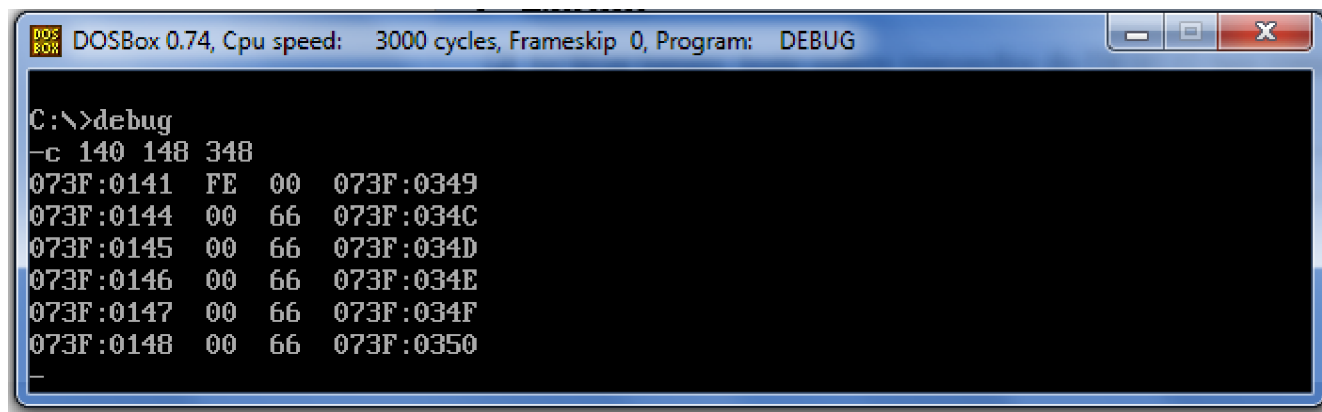
## 9 – Exercício

❑ Se tiver tempo, teste outros comandos do DEBUG tais como

➤ -C: Comparação

❖ Compara dois blocos de memória. Se não há diferenças, então o DEBUG simplesmente exibe -.

-c 140 148 348↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\>debug
-c 140 148 348
073F:0141 FE 00 073F:0349
073F:0144 00 66 073F:034C
073F:0145 00 66 073F:034D
073F:0146 00 66 073F:034E
073F:0147 00 66 073F:034F
073F:0148 00 66 073F:0350
-
```

❑ Os bytes entre os *offset* 140 a 148 estão sendo comparados com aqueles do *offset* 340 (até 348, implícita); os bytes são apresentados lado a lado para aqueles que são diferentes (com os seus endereços exactos, incluindo o segmento, em ambos lados destas).

# Laboratório

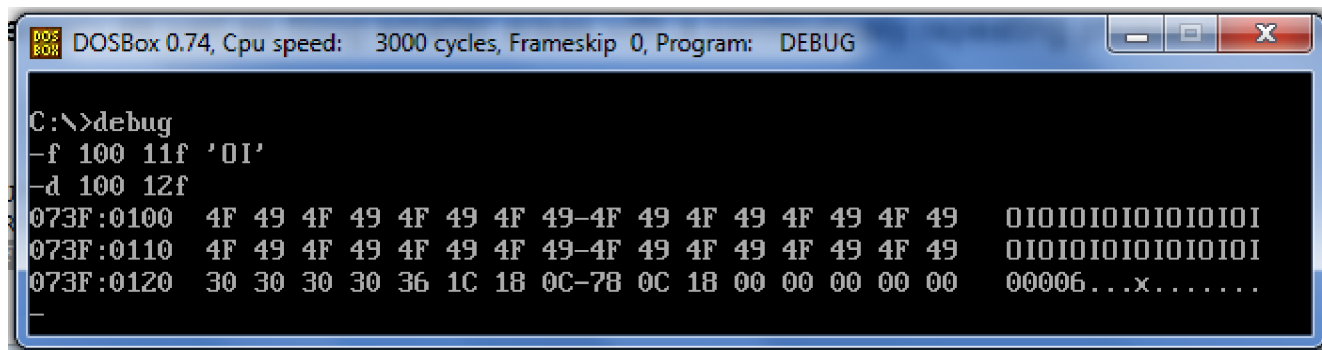
## 9 – Exercício

❑ Se tiver tempo, teste outros comandos do DEBUG tais como

➤ -F: Iniciar bloco de memória (FILL)

❖ Este comando também pode ser usado para limpar todo um segmento de memória, assim como para preencher áreas menores com uma frase continuamente repetida ou um único byte.

```
-f 100 11f 'OI' ↵  
-d 100 12f
```



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG  
C:\>debug  
-f 100 11f 'OI'  
-d 100 12f  
073F:0100  4F 49 4F 49 4F 49 4F 49-4F 49 4F 49 4F 49 4F 49  0101010101010101  
073F:0110  4F 49 4F 49 4F 49 4F 49-4F 49 4F 49 4F 49 4F 49  0101010101010101  
073F:0120  30 30 30 30 36 1C 18 0C-78 0C 18 00 00 00 00 00  00006...x.....  
-
```

# Laboratório

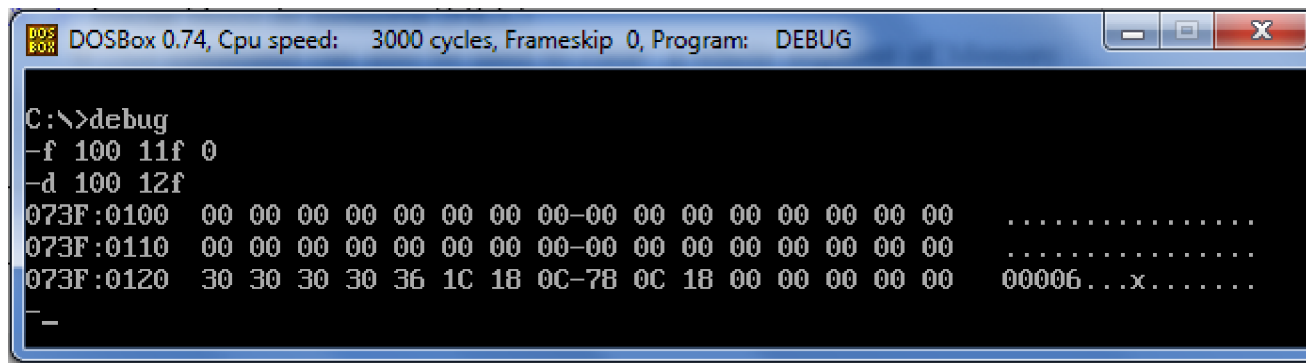
## 8 – Exercício

❑ Se tiver tempo, teste outros comandos do DEBUG tais como

➤ -F: Iniciar bloco de memória (FILL)

❖ Este comando também pode ser usado para limpar todo um segmento de memória, assim como para preencher áreas menores com uma frase continuamente repetida ou um único byte.

```
-f 100 11f 0  
-d 100 12f
```



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG  
C:\>debug  
-f 100 11f 0  
-d 100 12f  
073F:0100  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....  
073F:0110  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....  
073F:0120  30 30 30 30 36 1C 18 0C-78 0C 18 00 00 00 00 00  00006...x.....  
_
```

❑ Este último exemplo preenche com zeros os bytes entre os *offset* 100h a 11Fh.

# Laboratório

## 9 – Exercício

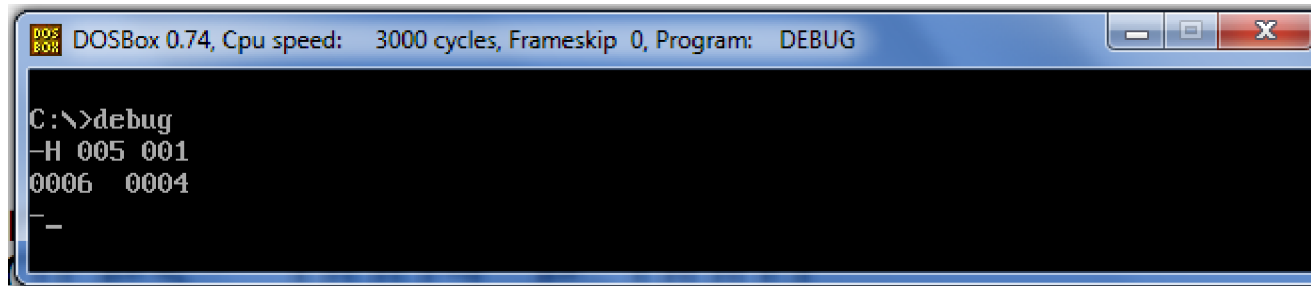
❑ Se tiver tempo, teste outros comandos do DEBUG tais como

➤ -H: Somar e subtrair números

❖ Uma muito simples (apenas somar e subtrair) calculadora Hexadecimal.

▪ O primeiro o valor da Adição e o segundo da Subtração.

-H 005 001 ↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG

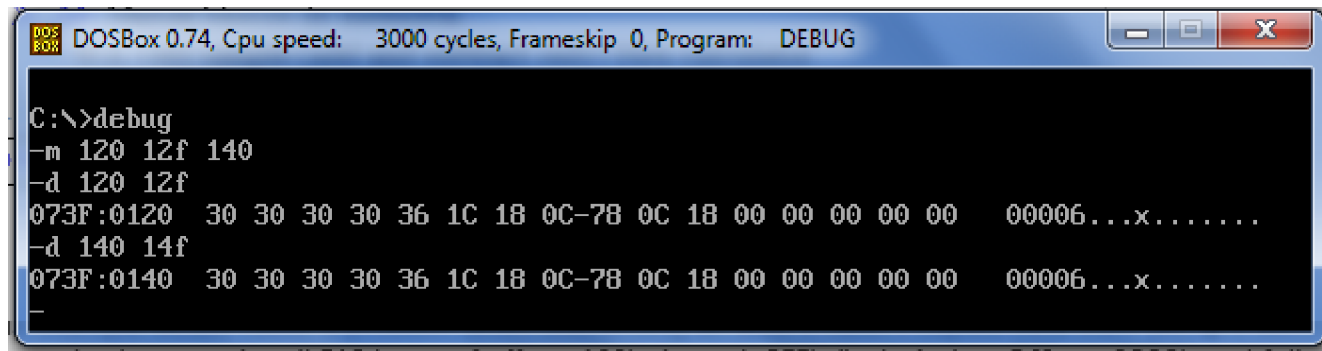
C:\>debug
-H 005 001
0006 0004
-
```

# Laboratório

## 9 – Exercício

- ❑ Se tiver tempo, teste outros comandos do DEBUG tais como
  - -M: Mover blocos de memória.

-m 120 12f 140↵



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\>debug
-m 120 12f 140
-d 120 12f
073F:0120 30 30 30 30 36 1C 18 0C-78 0C 18 00 00 00 00 00 00 00006...x.....
-d 140 14f
073F:0140 30 30 30 30 36 1C 18 0C-78 0C 18 00 00 00 00 00 00 00006...x.....
```

- ❑ Copia todos os 16 bytes entre os *offset* 120h a 12Fh para o *offset* 140h e seguintes.

# Anexos

# Anexos

## Instalação do DOSBox em Windows

1. Instalar DOSBOX.exe

**Link de descarga:**

<http://sourceforge.net/projects/dosbox/files/dosbox/0.74/DOSBox0.74-win32-installer.exe/download>

**Nota:** ver o vídeo: *How to install Debug.exe in Windows 7/8/8.1 64bit*

<https://www.youtube.com/watch?v=uxOi86OnoGw>

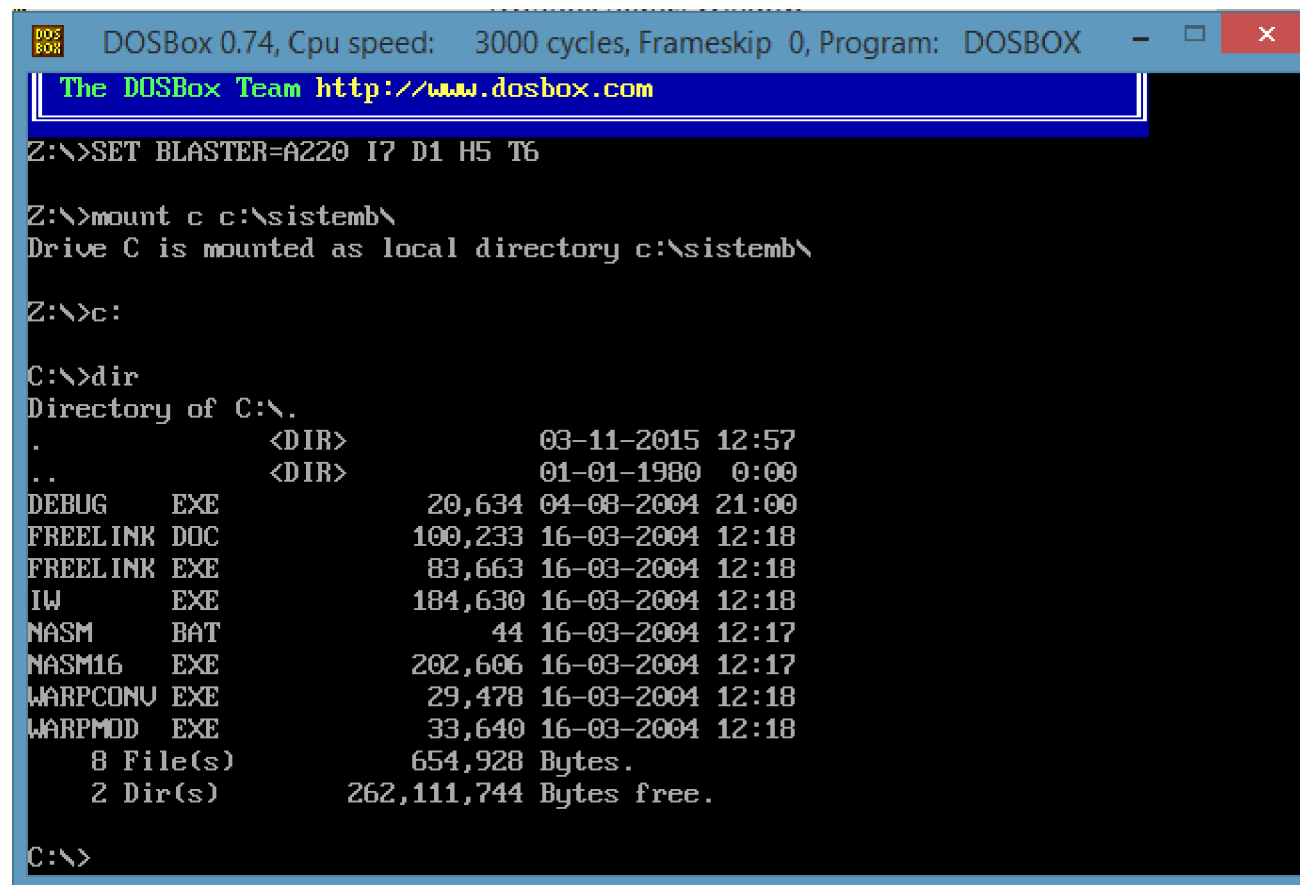
2. Criar uma pasta em C, (por exemplo C:\SistEmb) e copiar em ela os executáveis *debug.exe*, *freelink.exe*, *nasm.exe*, *nas16.exe*, *iw.exe*, *warpconv.exe*, *warpmo.exe* e *nasm.bat*.

# Anexos

## Instalação do DOSBox em Windows

3. Abrir DOSBOX.exe e montar a pasta onde estão os executáveis

`z:\>mount c c:\sistemb\`



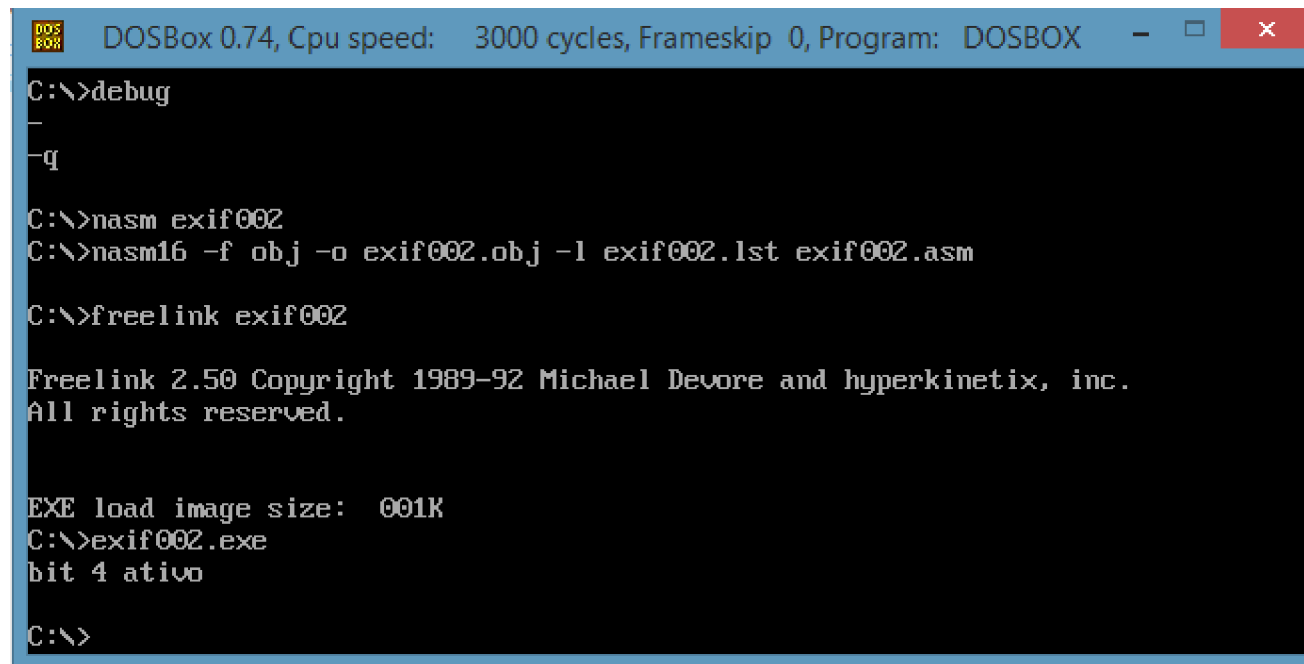
```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
The DOSBox Team http://www.dosbox.com
Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount c c:\sistemb\
Drive C is mounted as local directory c:\sistemb\
Z:\>c:
C:\>dir
Directory of C:\.
.                <DIR>                03-11-2015 12:57
..               <DIR>                01-01-1980  0:00
DEBUG    EXE                20,634 04-08-2004 21:00
FREELINK DOC            100,233 16-03-2004 12:18
FREELINK EXE            83,663 16-03-2004 12:18
IW       EXE            184,630 16-03-2004 12:18
NASM     BAT              44 16-03-2004 12:17
NASM16   EXE            202,606 16-03-2004 12:17
WARPCONV EXE            29,478 16-03-2004 12:18
WARPMOD  EXE            33,640 16-03-2004 12:18
      8 File(s)            654,928 Bytes.
      2 Dir(s)            262,111,744 Bytes free.
C:\>
```



# Anexos

## Instalação do DOSBox em Windows

4. testar o debug, nasm, freelink (no caso do exemplo esta sendo usado o programa exif002.asm)



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>debug
-
-q

C:\>nasm exif002
C:\>nasm16 -f obj -o exif002.obj -l exif002.lst exif002.asm

C:\>freelink exif002

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

EXE load image size: 001K
C:\>exif002.exe
bit 4 ativo

C:\>
```

# Anexos

## Instalação do DOSBox em Windows

### □ Vídeos

- DEBUG : Tutorial, Construindo programa, Acessando Memória
  - ❖ [https://www.youtube.com/watch?v=QFwo14\\_O8bg](https://www.youtube.com/watch?v=QFwo14_O8bg)