Aula 6

Conceitos Fundamentais – 8086

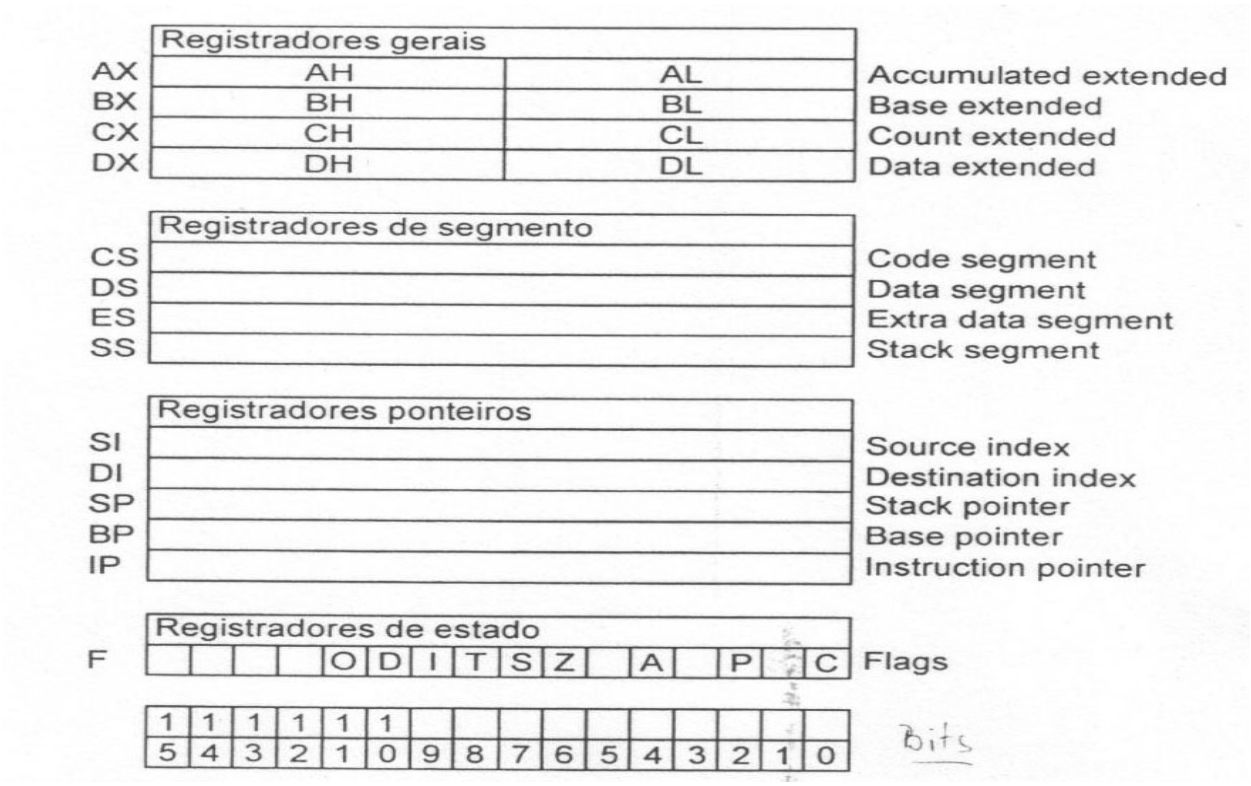
1) Byte / Nible / Word / Double Word / Quad Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte mais significativo								*Byte* menos significativo							
Nibble 3				*Nibble* 2				*Nibble* 1				*Nibble* 0			
Nibble mais sig.												*Nibble* menos sig.			

Word – Valores inteiros e deslocamentos de segmento
Double Word – Valores inteiros de 32 bits, valores ponto flutuante de 32 bits e endereços segmentados
Quad Word – 64 bits.

2) Registradores

Assembly → armazenar valores para programas
Registradores → variáveis (locais de armaz.)



Registradores de Estado:

Registradores de estado (Flags)															
				O	D	I	T	S	Z		A		P		C

1	1	1	1	1	1										
5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0

C = Carry
P = Parity
A = Auxiliary carry
Z = Zero
S = Sign
T = Trap
I = Interrupt enable
D = Direction
O = Overflow

O *flag* **CF** (bit 0) indica a sinalização de *carry* (vai um) para a próxima posição, após a efetivação de uma operação aritmética.

O *flag* **PF** (bit 2) indica se a paridade de um resultado é par ou ímpar após uma operação aritmética ou lógica.

O *flag* **AF** (bit 4) indica a sinalização de *carry* dos quatro primeiros bits quando da simulação de operações aritméticas de adição e subtração com valores decimais, representados por seqüências numéricas escritas em formato BCD (*Binary Coded Decimal*).

O *flag* **ZF** (bit 6) indica se o resultado após uma operação aritmética é zero, ou se o resultado de comparação após uma ação lógica é igual.

O *flag* **SF** (bit 7) mostra a obtenção de um resultado negativo ou positivo após uma operação aritmética.

O *flag* **TF** (bit 8) possibilita uma interrupção especial após executar uma única instrução, com a finalidade de acompanhar passo a passo a execução individual das instruções de um determinado programa.

O *flag* **IF** (bit 9) habilita ou desabilita o reconhecimento à chamada de interrupções.

O *flag* **DF** (bit 10) aponta a direção das operações de manipulação de blocos da direita para a esquerda ou vice-versa em seqüências de caracteres.

O *flag* **OF** (bit 11) indica a obtenção de um valor muito grande após uma operação aritmética ou lógica, estouro de capacidade.

3) Segmentos e Deslocamentos

Assembly 8086 → utiliza características do proc.

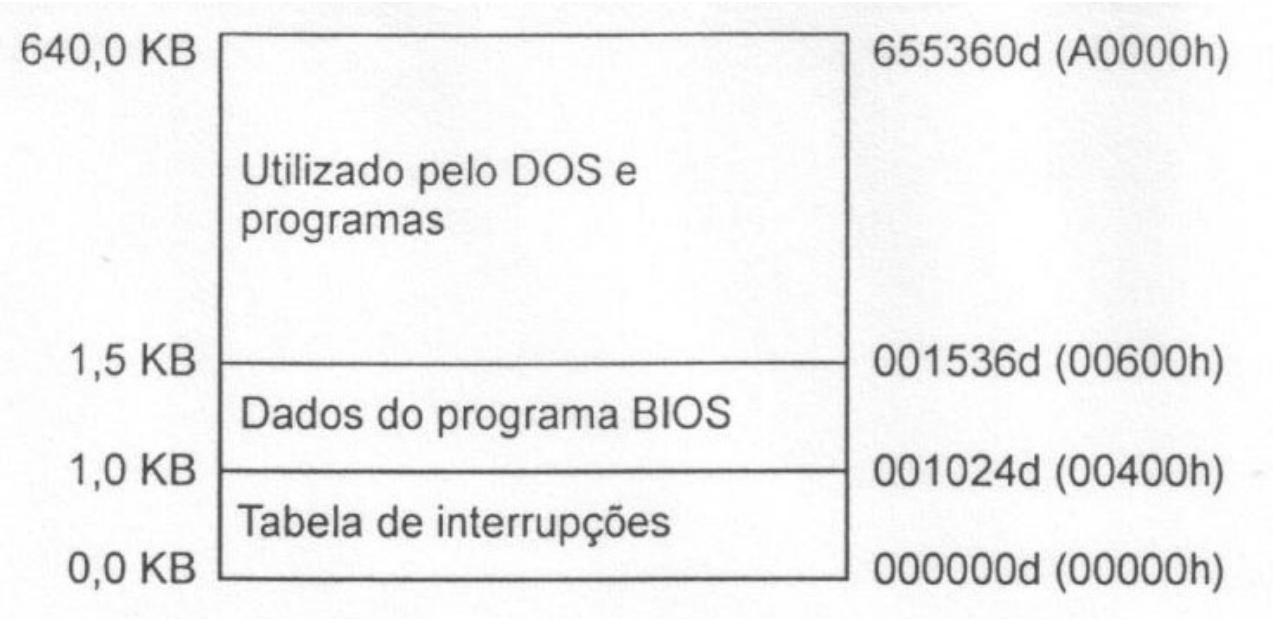
Gerenciamento de Memória:
Capacidade de endereçar uma MP de até 1 MB
Divida em 16 blocos (0 a F) com 64Kbytes cada.

Blocos → Segmentos → Ender. 0xxxxh até Fxxxxh
"0" é o primeiro segmento e "F" o último.

Cada segmento é divido em deslocamentos de 0000h até FFFFh (portanto 16bits de enderec.)

4) Endereçamento de Memória

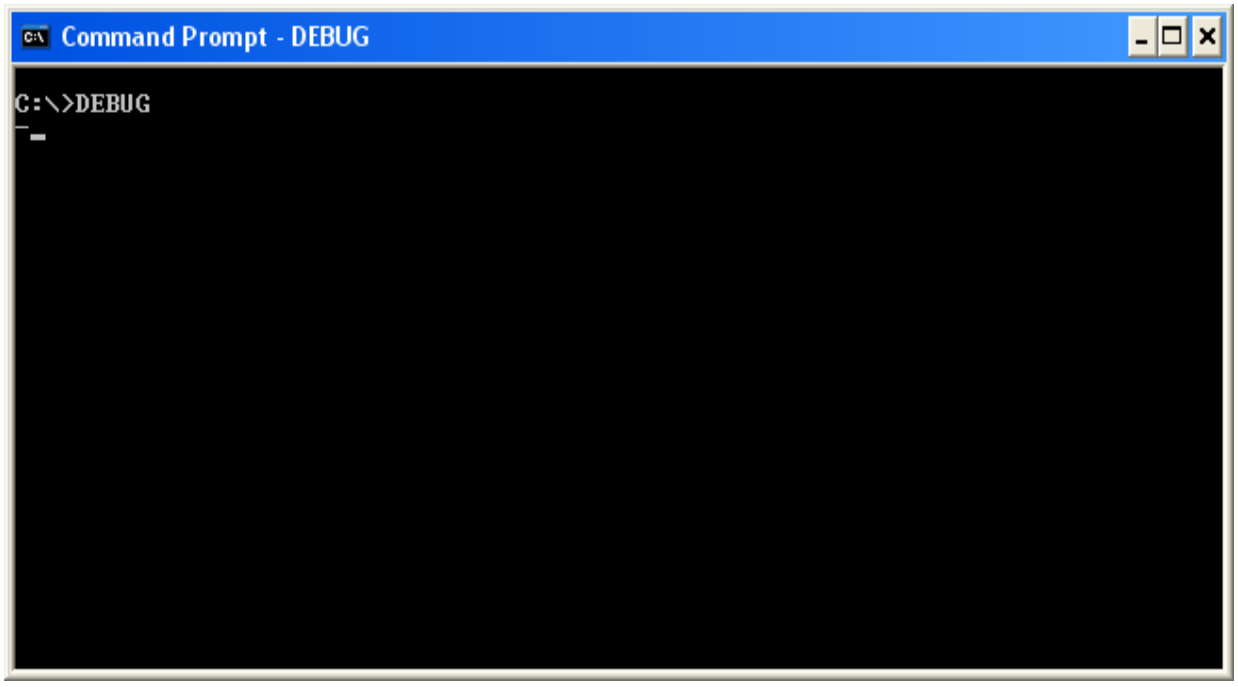
Células → 8 bits



Programação com DEBUG

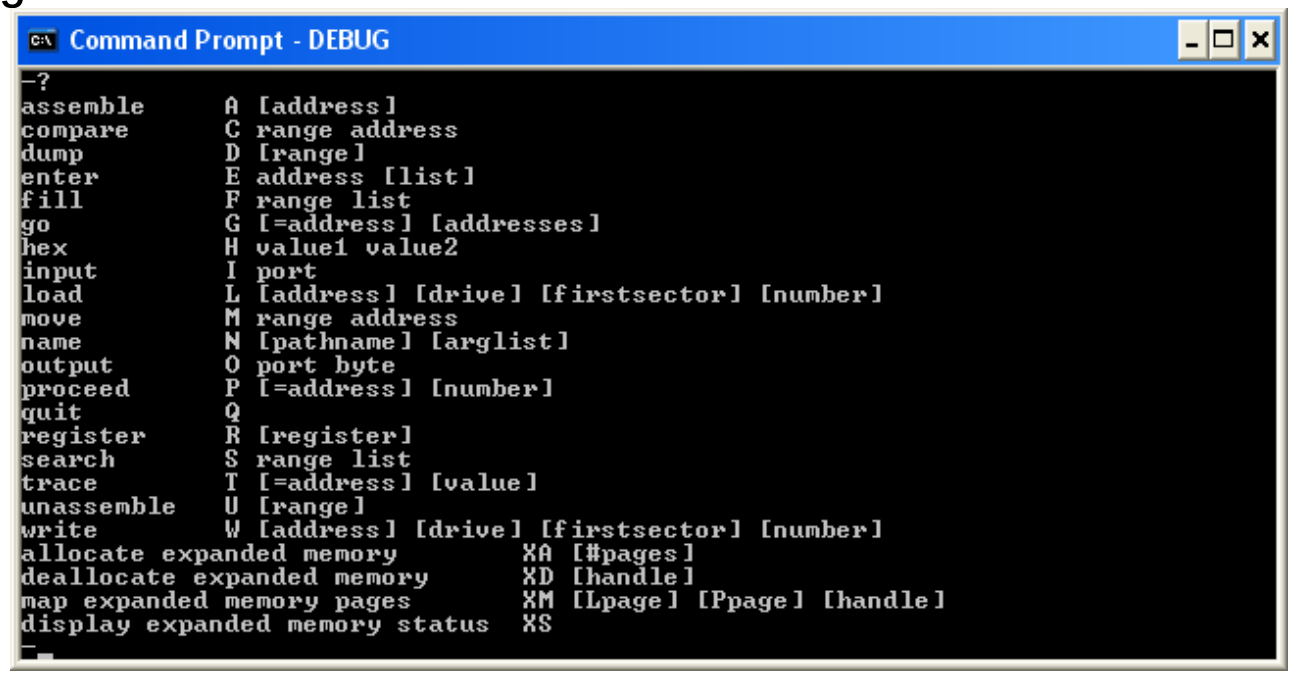
1) Cálculos Matemáticos Básicos

C:\> DEBUG <Enter>



Para sair do DEBUG, informar no prompt Q (quit).

Se digitar ? será mostrada Lista de Comandos do Programa DEBUG.



Aritmética em Modo Hexadecimal:

Para Adição e Subtração: Comando H
Supondo dois número em hex:0005 e 0001 Escreva:

H 0005 0001 <Enter>

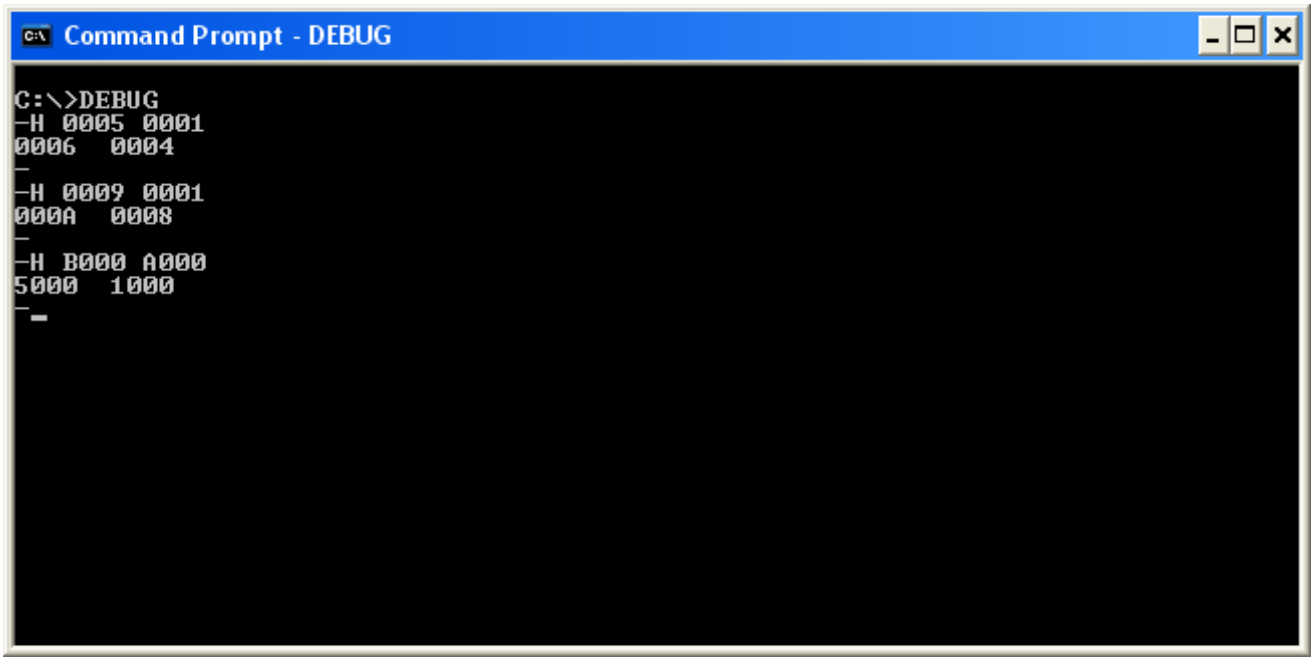
Após a execução, o resultado será:

-H 0005 0001 <Enter>
0006 0004

Sendo o primeiro o valor da Adição e o segundo da Subtração.

Vamos tentar os valores: 0009 e 0001
Resultado:
000A 0008

Outros valores: B000 e A000
Resultado:
5000 1000
(o primeiro valor deveria ser 15000. O DEBUG só apresenta os quatro valores à esquerda)



Representação de Valore Negativos:

Tente: H 0005 0006 <Enter>
Resultado:
000B FFFF

O segundo valor deveria ser: -0001 ???
FFFF corresponde a -0001

Tente: H 0005 FFFF
Resultado:
0004 0006

O que ocorre na verdade é a soma dos dois valores: 0005 + FFFF = 10004 (overflow)

Observe a tabela a seguir:

Valor hexadecimal	Valor binário
0000 (positivo)	0000 0000 0000 0000
7FFF (positivo)	0111 1111 1111 1111
8000 (negativo)	1000 0000 0000 0000
FFFF (negativo)	1111 1111 1111 1111

Valores Sinalizados: o bit 15 representa o sinal

2) Cálculos em Código de Máquina

Trabalhando com os registradores AX e BX

Vamos ver o que existe nos registradores?

No prompt do DEBUG digite: R

```
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 51            PUSH    CX
-
```

O Comando R, além de apresentar as informações dos registradores permite fazer atribuição de valores a um determinado registrador:

Vamos armazenar 000A no registrador AX:

R AX <Enter>
Será mostrado:

```
-R AX
AX 0000
:_
```

Basta digitar o valor e teclar <Enter>

```
-R AX
AX 0000
:000A <Enter>
```

Se verificarmos o estado dos registradores agora:

```
-R
AX=000A  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 51            PUSH    CX
-
```

Vamos armazer o valor 0001 em BX:
-R BX <Enter>
:0001 <Enter>
_

```
-R
AX=000A  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 51            PUSH    CX
-
```

Vamos ver uma primeira operação: Adição

Temos que armazenar o código (programa!!)

Por estarmos trabalhando com o programa DEBUG e ser este programa de pequena escala, o segmento de memória já está definido, basta estabelecermos o deslocamento:

Digamos: **0100** (valor em hex.)
Como é necessário informar dois códigos: o primeiro fica em 0100 e o segundo em 0101.

Para entrada de valores em um endereço de memória, utilize o comando E

E 0100 <Enter>

O programa apresenta algo semelhante a:
-E 0100
0B24:0100 51. (onde 0B24 → segmento mem.)

Ao lado, informe o valor 01 e tecle <Enter>

Depois faça: **E 0101 <Enter>**

Entre com o valor **D8 <Enter>**

A seqüência fica da seguinte forma:

```
-E 0100
0B24:0100 51.01
-E 0101
0B24:0101 80.D8
```

Os códigos hex. 01 e D8 representam os códigos em L.M. da operação (OPCODES) de adição dos valores armazenados nos registradores AX e BX

Se verificarmos o estado atual dos registradores, teremos:

```
-R
AX=000A  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 01D8          ADD     AX,BX
-
```

O próximo passo é informar ao processador onde encontrar na memória os códigos de ação da operação de adição.

Para tanto, o processador precisa saber qual o segmento e deslocamento devem ser utilizados, ou seja, onde está a instrução de processamento.

Essas duas informações são armazenadas em CS (code segment) e IP (Instruction Pointer).

Note as informações de CS e IP:

```
-R
AX=000A  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 01D8          ADD     AX,BX
-
```

Agora, basta pedir para o DEBUG executar a próxima instrução.

Vamos utilizar o comando T (trace). Ela executa uma instrução por vez.

```
-R
AX=000A  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 01D8          ADD     AX,BX

-T
AX=000B  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0102   NV UP EI PL NZ NA PO NC
0B24:0102 CF            IRET
-
```

O comando foi executado!!

AX ← AX + BX

Outros códigos de operação (OPCODES):

Operação	*Adição*	
Opcode	Registradores gerais	Operação
00 D8	AL, BL	AL ← AL + BL
01 D8	AX, BX	AX ← AX + BX
02 D8	BL, AL	BL ← BL + AL
03 D8	BX, AX	BX ← BX + AX

Subtração de Valore Hexadecimais:

Vamos sair e entrar no DEBUB.
Vamos carregar o valor 000A em AX e 0002 em BX

R AX <Enter>

... teríamos:

```
C:\>DEBUG
-R AX
AX 0000
:000a
-R BX
BX 0000
:0002
-R
AX=000A  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 01D8          ADD     AX,BX
-
```

Vamos informar a operação: **29 D8**

E 0100 <Enter>
... teríamos:

```
-E 0100
0B24:0100  01.29
-E 0101
0B24:0101  D8.D8
-R
AX=000A  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 29D8          SUB     AX,BX
-
```

Com IP = 0100, vamos executar o comando: T

Teríamos:

```
-R
AX=000A  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL
NZ NA PO NC
0B24:0100 29D8          SUB     AX,BX
```

```
-T
AX=0008  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0102   NV UP EI PL NZ NA PO NC
0B24:0102 CF            IRET
-
```

Os demais Códigos de Operação (OPCODES) são:

Operação	*Subtração*	
Opcode	Registradores gerais	Operação
28 D8	AL, BL	AL ← AL - BL
29 D8	AX, BX	AX ← AX - BX
2A D8	BL, AL	BL ← BL - AL
2B D8	BX, AX	BX ← BX - AX

Multiplicação de Valores Hexadecimais:

Carreguem os registradores AX e BX com os valores 0005 e 0003.

Carreguem os códigos de operação F7 e E3 nas posições 0100 e 0101.

Se tudo der certo, teremos:

```
C:\>DEBUG
-R AX
AX 0000
:0005
-R BX
BX 0000
:0003
-E 0100
0B24:0100  29.F7
-E 0101
0B24:0101  D8.E3
-
-R
AX=0005  BX=0003  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 F7E3           MUL     BX
-
```

Depois de executar (Comando T), teremos:

```
-T
AX=000F  BX=0003  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0102   NV UP EI PL NZ NA PE NC
0B24:0102 CF            IRET
-
```

Entretanto, uma multiplicação pode resultar em um número de 32 bits!!!

O que acontece?

Experimente carregar AX = 7D3C e BX = 0100

Vamos voltar o IP para o valor 0100 (onde está armazenada a instrução de multiplicação).

E vamos executar – Comando T

```
-R AX
AX 000F
:7D3C
-R BX
BX 0003
:0100
-R IP
IP 0102
:0100
-R
AX=7D3C  BX=0100  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PE NC
0B24:0100 F7E3          MUL     BX

-T
AX=3C00  BX=0100  CX=0000  DX=007D  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0102   OV UP EI PL NZ NA PE CY
0B24:0102 CF            IRET
-
```

Observe os registradores **AX e DX**
DX armazena os 16 bits mais significativos (altos) do resultado
AX os 16 bits menos significativos (baixos)

Os OPCODES são:

Operação	*Multiplicação*	
Opcode	**Registradores gerais**	**Operação**
F6 E3	BL	AX ← AL * BL
F7 E3	BX	DX:AX ← AX * BX

Divisão de Valores Hexadecimais

Vamos carregar os valores 0009 e 0002 para AX e BX, respectivamente.

Carregar os códigos F7 e F3 para as posições 0100 e 0101, respectivamente.

Teremos:

```
C:\>DEBUG
-R AX
AX 0000
:0009
-R BX
BX 0000
:0002
-E 0100 F7 F3
-R
AX=0009  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 F7F3          DIV     BX
-
```

Ao executarmos, teremos:

```
-T
AX=0004  BX=0002  CX=0000  DX=0001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0102   NV UP EI PL NZ NA PO NC
0B24:0102 CF            IRET
-
```

O Resultado da divisão (quociente) é armazenado em AX e o resto é armazenado em DX.

Os demais OPCODES são:

Operação	*Divisão*	
Opcode	**Registradores gerais**	**Operação**
F6 F3	BL	AX ← AL / BL AL = quociente AH = resto
F7 F3	BX	DX:AX ← AX / BX AX = quociente DX = resto

2) Apresentação de Dados

Apresentação de um Caracter

Vamos supor que queramos apresentar o caracter arroba (@) na tela.

A Interrupção responsável por controlar a apresentação de caracteres no vídeo é a **21**.

Carregue AX com 0200
Carregue DX com 0040 (valor do caracter @)

O valor 0200 em AX será utilizado para informar ao SO que ele deverá apresentar algo na Tela. O que vale aqui é o valor em AH = 02.

Vamos carregar o comando: CD 21 em 0100

E 0100 CD 21 <Enter>

Teríamos o seguinte:

```
C:\>DEBUG
-R AX
AX 0000
:0200
-R DX
DX 0000
:0040
-E 0100 CD 21
-R
AX=0200  BX=0000  CX=0000  DX=0040  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0100   NV UP EI PL NZ NA PO NC
0B24:0100 CD21           INT     21
-
```

Vamos executar agora com o comando G (go).

Para tanto, devemos informar até onde ele deve executar: G 0102 <Enter>

Teríamos o seguinte:

```
-G 0102
@
AX=0240  BX=0000  CX=0000  DX=0040  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=0102   NV UP EI PL NZ NA PO NC
0B24:0102 CF            IRET
-
```

Vamos acrescentar uma outra linha neste programa:

E 0102 CD 20 <Enter>

INT 20 (interrupção 20) → devolve o controle ao sistema operacional. Dessa forma, podemos executar o programa com a opção G, sem o endereço de termino.

Vamos mover o IP para 0100

E Executarmos o programa com G:

Assim teríamos:

```
-E 0102 CD 20

-R IP
IP 0102
:0100

-G
@
Program terminated normally
-
```

Para ver uma listagem do código, utilize o comando U (unassemble), informando o ponto inicial e o ponto final.

U 0100 0103 <Enter>

Teríamos:

```
-U 0100 0103
0B24:0100 CD21          INT     21
0B24:0102 CD20          INT     20
-
```

Para o próximo “programa”, vamos armazenar os valores 0200 em AX e 0046 em DX

Agora vamos entrar com os códigos utilizando o comando A (assemble):

A 0100 <Enter>

Automaticamente será apresentada a linha (segmento:deslocamento) em que o código deve ser definido. Vamos digitar o seguinte:

INT 21 <Enter>
INT 20 <Enter>
<Enter>

Teriamos o seguinte:

```
-A 0100
0B24:0100 INT 21
0B24:0102 INT 20
0B24:0104
-
```

Ao executarmos o programa, Teriamos:

```
-G
F
Program terminated normally
-
```

Movimentação de Dados:

Comando **MOV destino, origem**

Se desejarmos colocar o valor AABB no registrador AX, poderiamos utilizar o comando MOV da seguinte forma:

```
MOV AX,AABB
```

Podemos mover dados entre registradores:

```
MOV AX,BX   (que significa: AX ← BX)
```

Com esses conceitos, execute o comando:

```
A 0100 <Enter>
MOV AH,02 <Enter>
MOV DL,41 <Enter>
INT 21 <Enter>
INT 20 <Enter>
<Enter>
```

Ao executarmos:

```
-G
A
O programa terminou de forma normal
```

Este último programa, com 4 linhas, já pode ser gravado.

Entretanto, o processo de gravação deve seguir alguns passos:

1) É necessário saber o número de bytes que o programa ocupará em disco. Pegue o endereço de deslocamento da primeira instrução (0100) e o endereço de deslocamento da última instrução (0108) e subtraia. Utilize o comando H:
H 0108 0100 <Enter>
0208 0008 (sendo a segunda resposta o número de bytes).

2) Com o valor do número de bytes em mãos é necessário informá-lo aos registradores gerais BX e CX. O registrador BX somente será utilizado no caso do tamanho em bytes não cabe no registrador CX. Com o comando R CX informe o valor 0008.

3) Após executar os procedimento acima, use o comando N (name) para atribuir um nome ao programa. Para esse teste, forneça o nome MOSTRA.COM. Os programas gravados com a ferramenta DEBUG devem ter extensão .COM. Utilize a linha de código N MOSTRA.COM

4) Em seguida é necessário gravar o programa com o comando W (write). Basta apenas acionar W e aparece a mensagem de que o arquivo foi gravado. Essa mensagem informa também o número de bytes usados na gravação.

Agora que o arquivo foi gravado, saia do programa DEBUG e no prompt do DOS, digite:
MOSTRA e observe a apresentação da letra A.

Vamos agora apresentar uma mensagem (uma seqüência de caracteres).

Entre com o seguinte código:

```
-A 0100   <enter>
MOV AH,09 <enter>
MOV DX,0200  <enter>
INT 21  <enter>
INT 20  <enter>
<enter>
```

E logo em seguida, armazene no endereço 0200 a seguinte mensagem (você pode ser mais criativo do que eu e armazenar uma outra frase!!)

```
E 0200 “OBCLM eh otima!” 24
```

O Significado do número 24 no final da mensagem, é que este número corresponde ao valor em hexadecimal do caracter $, que indica o final de uma String.
A mensagem também poderia ser armazenada da seguinte forma:

```
E 0200 “OBCLM eh otima!$”
```

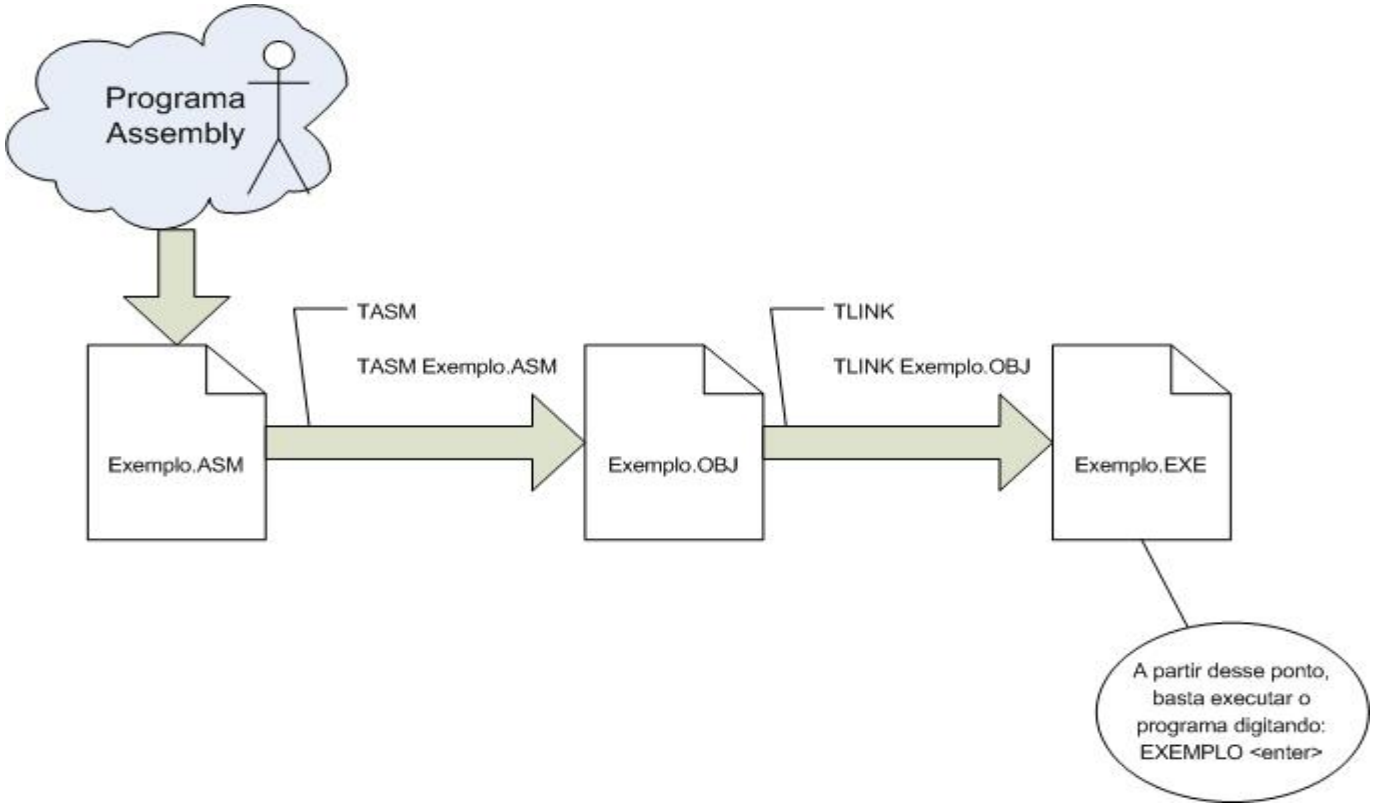
Mova o ponteiro IP para 0100 e execute esta seqüência de comandos com a opção G.

Se tudo deu certo o resultado deverá ser:

```
-G <enter>
OBCLM eh otima!
-
```

Fazer programas em Assembly já bem complicado. Utilizar o DEBUG para isso, fica mais difícil ainda.

A maneira com que faremos a programação em Assembly será da seguinte forma:



Editaremos a seqüência de instruções em um editor de texto puro (sem códigos de controle, como o bloco de notas). Salvaremos o arquivo com um determinado nome, com extensão .ASM.
Depois faremos a compilação, utilizando o TASM (Turbo Assembler – Borland). Se algum erro for evidenciado, editaremos novamente o arquivo e faremos as devidas correções.
Ao chegarmos a um processo de compilação sem erros, o TASM gerará um arquivo objeto (com extensão .OBJ). Depois disso, faremos a Ligação com o TLINK (Turbo Linker – Borland).
Se tudo der certo, teremos o arquivo executável (.EXE ou .COM – conforme configuração).

Para exemplificar tal processo, vamos escrever um programa simples, para fazer a mesma operação de escrita de uma mensagem na tela, realizada com o programa DEBUG.

Abra um editor de texto puro (como o BLOCO DE NOTAS) e digite o seguinte programa:

```
TITLE PGM1: MENSAGEM
.MODEL SMALL
.STACK 100H
.DATA
mensagem DB ‘OBCLM eh otima!$’
.CODE
MAIN PROC
  MOV AX,@DATA
  MOV DS, AX
  LEA DX,mensagem
  MOV AH,9
  INT 21H
  MOV AH,4CH
  INT 21H
MAIN ENDP
END MAIN
```

Grave o arquivo como MENSAGEM.ASM

Na linha de comando do DOS (no mesmo diretório onde você salvou o arquivo) digite:

TASM MENSAGEM.ASM <enter>

Ele apresentará a seguinte mensagem:

```
C:\Tasm>edit mensagem.asm

C:\Tasm>TASM MENSAGEM.ASM
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996
Borland International

Assembling file:   MENSAGEM.ASM
**Error** MENSAGEM.ASM(5) Extra characters on line
Error messages:    1
Warning messages:  None
Passes:            1
Remaining memory:  452k


C:\Tasm>
```

Observe que no exemplo acima ocorreu um erro no processo de compilação, indicando que o erro está na linha 5.
Basta editar o arquivo .ASM, corrigir o erro e compilar novamente o código.

Se tudo der certo, você terá obterá as seguintes mensagens:

```
C:\Tasm>TASM MENSAGEM.ASM~
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996
Borland International

Assembling file:   MENSAGEM.ASM~
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  452k

C:\Tasm>
```

Agora basta fazer a ligação com o TLINK.

TLINK MENSAGEM.OBJ <enter>

Se tudo der certo, você terá a seguinte mensagem:

```
C:\Tasm>TLINK MENSAGEM.OBJ
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996
Borland International

C:\TASM>
```

Pronto! Agora é só executar o programa!!
Se tudo der certo, você terá:

```
C:\TASM>MENSAGEM
OBCLM eh otima!
C:\TASM>
```