```
0D82:0100 B402          MOV     AH,02
0D82:0102 B241          MOV     DL,41
0D82:0104 CD21          INT     21
0D82:0106 CD20          INT     20
0D82:0108 69            DB      69
```

Capítulo 4

CÁLCULOS MATEMÁTICOS BÁSICOS

*Antes mesmo de estudar a linguagem de programação Assembly 8086/8088 (mesmo em nível básico, como é a proposta desta obra), é importante ter uma noção do funcionamento de um computador controlado pelo seu código nativo, ou seja, em linguagem de máquina, pois esse conhecimento dá uma ideia do quão mais fácil é usar a linguagem de programação Assembly.Este capítulo dá noção das operações computacionais de máquina (cálculos matemáticos, baseados nas quatro operações aritméticas) realizadas com o programa **Enhanced DEBUG**. Ensina a executar algumas instruções internas do programa, além de utilizar a aritmética com valores hexadecimais, negativos e a operação de cálculos matemáticos simples (adição, subtração, multiplicação e divisão) com códigos de máquina.*

4.1 - O Programa Enhanced DEBUG

Os sistemas operacionais MS-DOS de 16 *bits* e Microsoft Windows em todas as suas versões de 32 *bits* até a versão 10 que necessita da instalação do programa **NTVDM**, possuem no diretório básico de operação o programa **DEBUG**. No sistema operacional MS-DOS o programa fica instalado no diretório C:\DOS, e no Microsoft Windows o programa fica instalado na pasta C:\WINDOWS\COMMAND ou na pasta C:\WINDOWS\SYSTEM32. O programa **DEBUG** (foi originalmente escrito por Tim Paterson em 1980) não sendo encontrado nas edições de 64 *bits* do sistema operacional da Microsoft Windows. Neste caso, pode-se fazer uso de uma versão alternativa do programa desenvolvida pelo professor Paul Vojta para o projeto *FreeDOS*, denominado **Enhanced DEBUG** que opera apenas sob 32 *bits*.

O programa **DEBUG** (versão original) pode ser executado no sistema operacional Microsoft Windows 7 de 64 *bits*, por meio do recurso de virtualização **XP Mode**, desde que este recurso esteja instalado. No caso dos sistemas operacionais Microsoft Windows Vista, 7, 8, 8.1 e 10 de 64 *bits* uma solução é o uso do programa **Enhanced DEBUG** executado preferencialmente com auxilio do emulador de sistema operacional MS-DOS denominado **vDos**.

O nome *debug* está associado à retirada de defeitos e erros existentes em um programa. O termo *bug* (bâgui), que em inglês significa inseto, foi associado a falhas computacionais quando ocorreu uma parada inesperada do computador ENIAC por ser encontrada uma mariposa presa em um dos circuitos eletrônicos (MANZANO & MANZANO, 1998). Há autores que afirmam ter sido uma parada no computador MARK I (NORTON & SOCHA, 1988). Independentemente da versão histórica, o fato é sempre associado ao evento de ter sido encontrado um inseto. Assim sendo, *debug* significa algo como dedetizar, ou seja, retirar insetos, remover problemas.

O programa **Enhanced DEBUG** é uma ferramenta básica que permite estabelecer a manipulação de dados e de registradores de memória em linguagem de máquina, dando suporte à ação de comandos da linguagem de programação *Assembly* para o padrão de microprocessadores 8086/8088 (IBM, 1995). É uma ferramenta simples, mas poderosa, que possui a capacidade de criar pequenos programas em linguagem de máquina. Sua capacidade de trabalho está limitada à criação de programas com extensão **.COM**. Por esta razão, esses programas podem ter no máximo 64 Kb de memória e devem sempre ser iniciados no endereço de deslocamento de memória **0100h**.

É oportuno esclarecer onde se encontra contextualmente as ações de trabalho das linguagens de programação relacionadas a códigos de máquina, ao código *Assemlby* e as linguagens de alto nível. A Figura 4.1, adaptada de Yadav (2008, p. 25) apresenta o esquema de distribuição de acesso entre os *hardwares* e as linguagens de programação de máquina, *assembly* e alto nível.

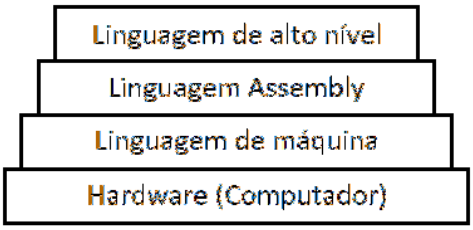

Figura 4.1 - Esquema de distribuição de acesso entre o hardware e as linguagens de programação.

Para iniciar a execução do programa **Enhanced DEBUG** selecione na área de trabalho do sistema operacional Microsoft Windows o ícone do programa **vDos** e na linha de comando apresentada informe os comandos:

```
C:\>E:      <Enter>
E:\>DEBUGX <Enter>
```

A indicação **<Enter>** sugere que a tecla **<Enter>** seja acionada a cada comando fornecido, isto é, não se deve escrever após a indicação de cada comando o indicativo **<Enter>**. Assim que o programa é carregado, ele apresenta seu *prompt* de trabalho com um pequeno traço (um hífen) à esquerda da tela. Observe na Figura 4.2 a tela após a chamada do programa **Enhanced DEBUG**.

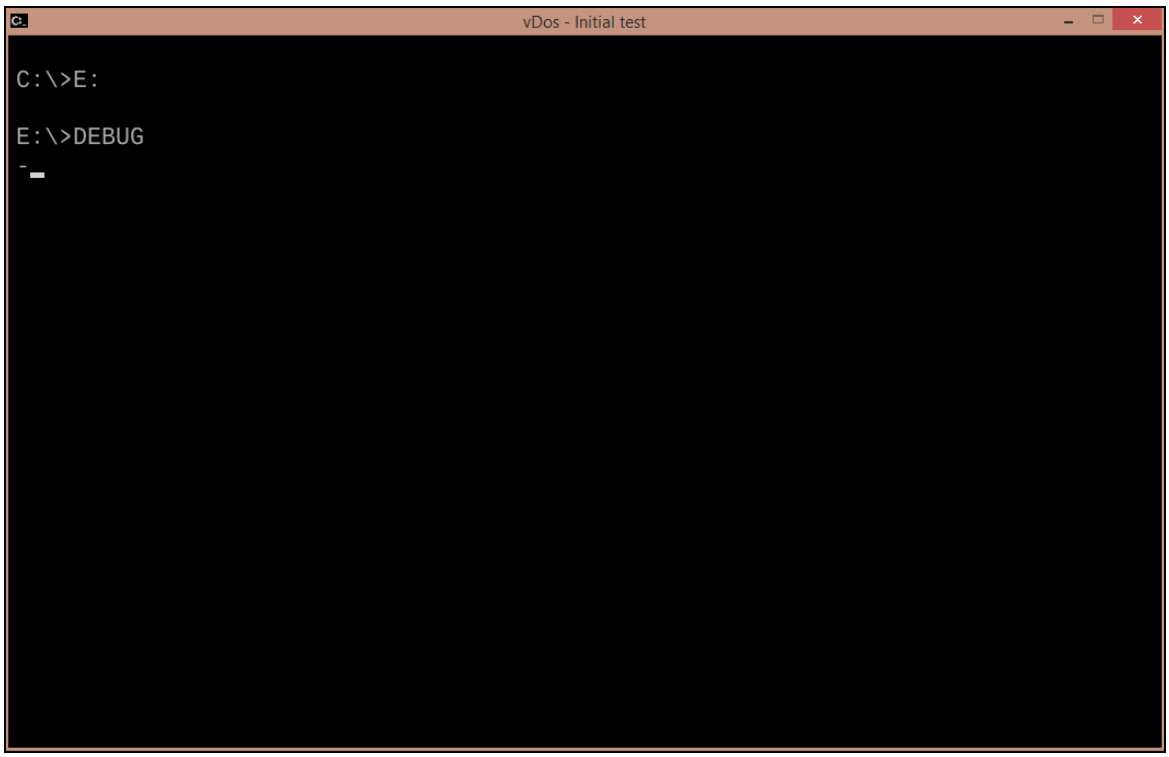

Figura 4.2 - Aparência da tela com o programa DEBUG em operação (Microsoft).

Para sair do programa **Enhanced DEBUG,** basta informar no seu *prompt* de trabalho o código **Q** (*quit*).

É pertinente indicar que todos os comandos internos de referência ao programa **Enhanced DEBUG** podem ser escritos em caracteres maiúsculos ou minúsculos. No entanto, neste capítulo as referências aos comandos são apresentadas sempre em caracteres maiúsculos.

O programa **Enhanced DEBUG** tem uma série de comandos que facilitam as operações em memória. Para ter uma ideia dos comandos disponíveis, acione no *prompt* do programa o comando **?** (*help*), o qual apresenta a lista completa de comandos, conforme exibe as Figuras 4.3 e 4.4. No decorrer do livro, alguns desses comandos internos serão utilizados em várias oportunidades.

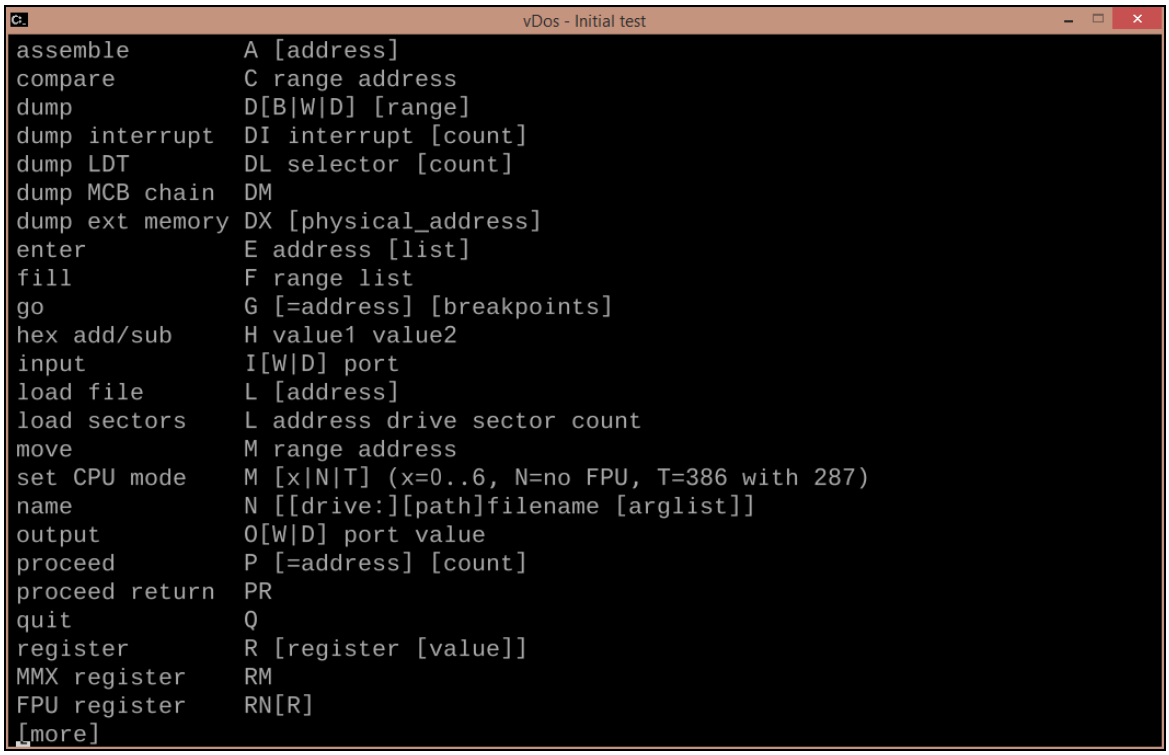

Figura 4.3 - Lista de comandos do programa Enhanced DEBUG – Parte 1.

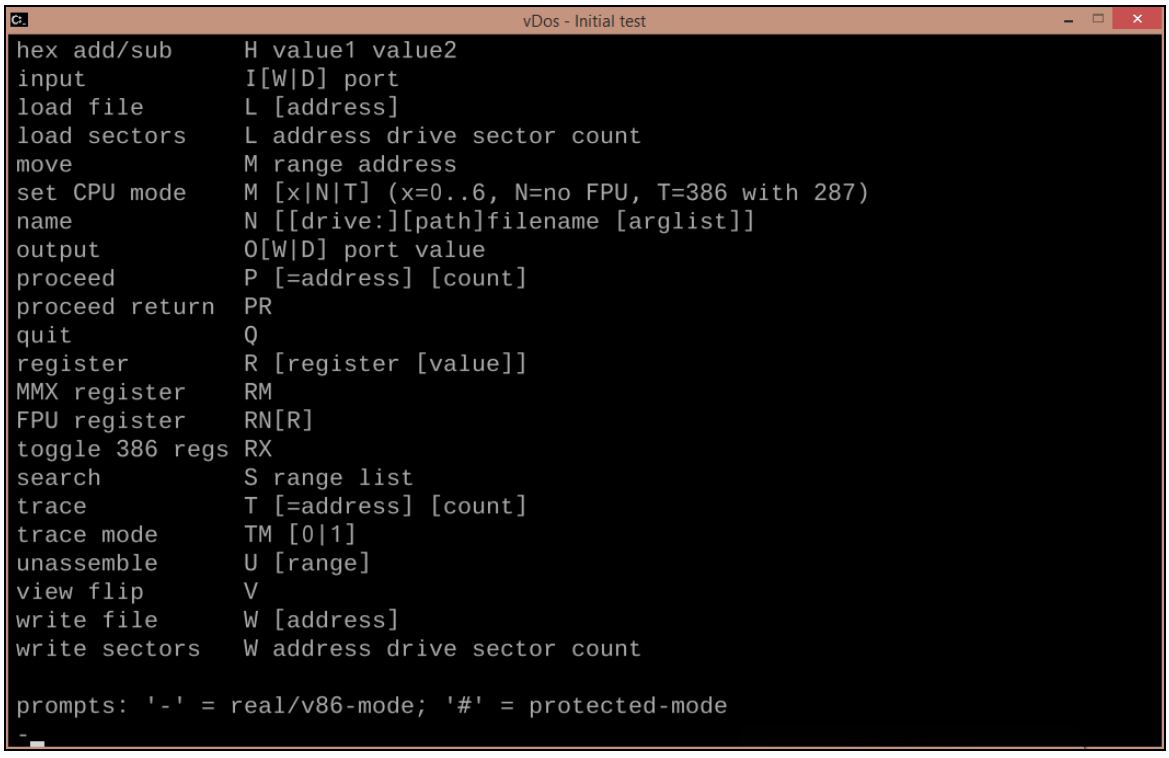

Figura 4.4 - Lista de comandos do programa Enhanced DEBUG – Parte 2.

É pertinente salientar que não serão vistos todos os comandos do utilitário **Enhanced DEBUG**, apenas os comandos que são encontrados igualmente em todos os programas *DEBUG*. Maiores detalhes sobre o programa *DEBUG* podem ser obtidos junto ao sítio **www.debug.manzano.pro**.

4.2 - Aritmética em modo hexadecimal

O recurso de cálculo numérico oferecido no programa **Enhanced DEBUG** permite obter os resultados de adição e subtração de dois valores hexadecimais (não é possível trabalhar com números decimais) fornecidos no *prompt* do programa.

Para adição e subtração de valores hexadecimais é necessário utilizar o comando **H** (*hex*) e informar sempre dois valores hexadecimais. Escreva no *prompt* do programa o comando **H** seguido dos valores hexadecimais **0005** e **0001** (que possuem o mesmo significado dos seus equivalentes em decimal) e acione a tecla **<Enter>**. Veja a indicação:

```
H 0005 0001 <Enter>
```

Após executar a operação anterior, o programa apresenta como resultado a informação:

```
0006 0004
```

O valor hexadecimal **0005** e o valor hexadecimal **0001** resultaram na apresentação dos valores **0006** e **0004**, sendo **0006** o valor da adição de **0005** com **0001**, e **0004** o valor da subtração entre **0005** e **0001**. Observe que o programa **DEBUG** faz os dois cálculos aritméticos simultaneamente e fornece as duas respostas.

No entanto, um valor hexadecimal é diferente de um valor decimal. Por exemplo, execute o seguinte comando de cálculo aritmético para calcular os resultados de adição e subtração entre os valores hexadecimais **0009** e **0001**.

```
H 0009 0001 <Enter>
```

Após a execução da ação anterior, o programa apresenta como resultado a informação:

```
000A 0008
```

O valor **000A** (hexadecimal) corresponde ao valor **10** (decimal). Apesar de ser uma operação de fácil execução, é necessário ter muito cuidado com a leitura do valor numérico, uma vez que não está sendo utilizada base decimal de representação numérica.

Outro cuidado a ser tomado é em relação à utilização de valores numéricos maiores que a capacidade de cálculo do programa **Enhanced DEBUG**. Você deve ter notado que a resposta dada à solicitação de um determinado cálculo aritmético (adição e subtração) é sempre apresentada em quatro posições (dígitos). Números hexadecimais com quatro posições usam a estrutura de dados *word* para serem armazenados e processados. Números com dígitos acima dessa capacidade resultam em um estouro de cálculo, e isso é considerado um erro de operação.

Se estiver em uso o programa **DEBUG** (original) este tem a limitação de trabalhar com valores até o tamanho de um *word*, compatíveis com microprocessador de 16 bits. Por exemplo, se houver a tentativa de realizar o cálculo aritmético entre os valores hexadecimais **B000** (45.056 em decimal) e **A000** (40.960 em decimal), ocorre um erro de estouro:

```
H B000 A000 <Enter>
```

Após a ação anterior o programa apresenta como resultado a informação:

```
5000 1000
```

O valor hexadecimal **5000**, resultado da adição dos valores **B000** e **A000** no programa **DEBUG** original que deveria ser apresentado como sendo o valor hexadecimal **15000** (86.016 em decimal) como ocorre no uso do programa **Enhanced DEBUG** ao ser apresentado o valor **00015000**.Já o valor hexadecimal **1000** (4.096 decimal) está sendo apresentado de forma correta tanto no **DEBUG** original como no **Enhanced DEBUG**. O programa **DEBUG** original mostra apenas os quatro valores à direita do resultado da adição. Essa restrição do programa limita o seu uso em algumas operações. Já o programa **Enhanced DEBUG** apresenta o resultado de forma completa pelo fato de operar com microprocessadores de 32 *bits* e não 16 *bits* como ocorre com o programa **DEBUG** oroginal.

Em particular o programa **Enhanced DEBUG** possui a capacidade de efetuar suas operações também em modo 32 *bits*. Para fazer uso deste recurso basta executar no *prompt* do programa o comando **RX** que apresentará a mensagem **386 regs on**. A partir deste instante o modo 32 *bits* de operação do programa torna-se ativo. Para voltar o programa **Enhanced DEBUG** ao modo de operação 16 *bits* basta executar novamente a instrução **RX**. A Figura 4.5 mostra a ativação do modo 32 e 16 *bits* e o uso do comando **R** que apresenta o estado dos registradores manipulador pelo programa.

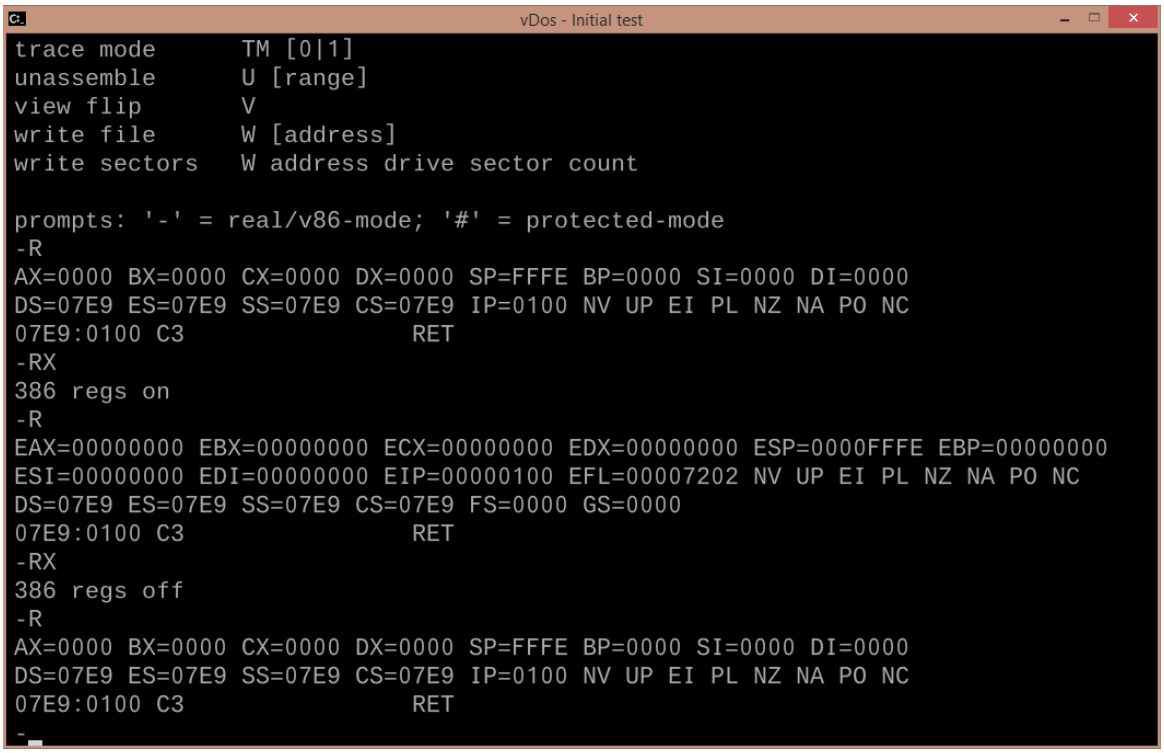

Figura 4.5 - Modelos de de uso de registradores em 16 ou 32 bits.

Os exemplos de operação apresentados neste trabalho consideram apenas o uso de registradores em modo 16 *bits* atendendo o uso do microprocessador 8086.

4.3 - Representação de valores negativos

Como diz Hyde (2003) os valores numéricos representados em um computador digital não podem ser infinitos, como ocorre no universo matemático, pelo simples fato de que a quantidade de *bits* disponível para representá-los em um computador é restrita (8, 16, 32, 64, 128 ou qualquer outra quantidade que nunca será muito grande, ou seja, sempre haverá um limite máximo). Com um número fixo de *bits*, o valor máximo também é fixo. Por exemplo, com 8 *bits* (1 *byte*) pode-se obter no máximo 256 combinações de valores diferentes. Se houver a necessidade de expressar números negativos (o que é comum e necessário), será preciso dividir as 256 possibilidades existentes em dois conjuntos numéricos.

Assim sendo, ter-se-á metade de um conjunto para representar os valores positivos e a outra metade para representar os valores negativos. Essa atitude diminui o valor máximo de combinações, porém aumenta o valor mínimo de combinações. Se a divisão for benfeita, será possível obter valores na faixa numérica de –128 até 127. O mesmo raciocínio pode ser aplicado para valores numéricos de 16 *bits*, 32 *bits* etc.

Para exemplificar uma operação aritmética que represente a obtenção de valores negativos, considere o cálculo entre os valores hexadecimais **0005** e **0006** (nessa ordem). Informe na linha de *prompt* do programa:

```
H 0005 0006 <Enter>
```

Após a ação anterior o programa apresenta como resultado a informação:

```
000B FFFF
```

Certamente o leitor estava esperando a apresentação do valor hexadecimal de adição **000B** (o que ocorreu) e o valor hexadecimal de subtração **–0001** (que está sendo representado pelo valor hexadecimal **FFFF**). No contexto exposto o valor hexadecimal **FFFF** está representando realmente o valor decimal **–1**. Assim sendo, a operação está correta. Tanto que, se for feito o cálculo da soma do valor hexadecimal **0005** com o valor hexadecimal **FFFF**, a resposta será os valores **0004** e **0006**:

```
H 0005 FFFF <Enter>
```

O programa apresenta como resultado a informação:

```
00010004 0006
```

O valor hexadecimal **0005** com o valor hexadecimal **FFFF** (equivalente a **-1** em decimal) resultam no valor de adição hexadecimal **00010004** (valor decimal **5**, mais o valor decimal **-1**) e no valor de subtração hexadecimal **0006** (valor decimal **5**, menos valor decimal **–1**). Se estiver em uso o programa **DEBUG** (original) o resultado apresentado será diferente como **0004 0006**. O valor hexadecimal **FFFF,** dependendo do contexto em que é aplicado, pode ser interpretado como o valor decimal **65.535**. O valor **65.535** é o maior valor positivo que pode ser representado em um *word*.

A consideração em relação ao valor hexadecimal **FFFF** ser interpretado como o valor decimal **–1** ou como o valor decimal **65.535** depende de levar em conta se o valor hexadecimal **FFFF** está ou não sinalizado. Se o valor hexadecimal **FFFF** for positivo (equivalente ao valor decimal **65.535**), será considerado um valor não sinalizado. Agora, se o valor hexadecimal **FFFF** for negativo (equivalente ao valor decimal **-1**), será considerado um valor sinalizado, ou seja, um valor que possui o sinal de subtração à sua frente para determinar sua condição. Desta forma, todos os valores numéricos que gerarem um estouro e estiverem situados na faixa numérica hexadecimal de **8000** até **FFFF** poderão ser representados como valores sinalizados (valores negativos) ou como valores positivos (valores não sinalizados).

A informação que determina se um valor hexadecimal é negativo toma por base o estouro do valor quando o resultado de uma operação aritmética é apresentado (ou seja, um valor com cinco posições numéricas), desde que o estouro esteja situado na faixa numérica hexadecimal de **8000** até **FFFF;** caso contrário, ter-se-á um erro como o exemplificado no final do tópico anterior.

Por exemplo, a subtração do valor hexadecimal **0005** do hexadecimal **FFFF** (considerado **-1**) resulta na resposta **0004**. No entanto, internamente ocorre, como já mencionado, a soma dos valores e não a subtração, ou seja, o valor hexadecimal **0005** é somado ao valor hexadecimal **FFFF**, que resulta em um valor hexadecimal equivalente a **00010004**.

No caso do programa **DEBUG** (original) ocorre a desconsideração dos quatro valores à esquerda, ou seja, despreza-se o valor **0001** mantendo-se apenas as quatro posições à direita do valor **0004**. O valor **1** é, na verdade, a indicação de um estouro (ocorrência de *overflow*) da capacidade máxima de armazenamento dos valores numéricos num dado *word*, ficando armazenado num registrador de *flag* (que será apresentado um pouco mais adiante).

É conveniente levar em consideração o fato de que os computadores digitais somente processam informações em formato binário. O programa **Enhanced DEBUG** (e mesmo o programa **DEBUG**) aceita a entrada de valor numérico em formato hexadecimal e o apresenta em formato hexadecimal, mas o converte internamente em seu respectivo valor binário. A real determinação de um valor ser expresso em formato positivo ou negativo ocorre na esfera binária do processamento da máquina, e não na esfera hexadecimal.

Para entender melhor a representação de valores negativos, considere como exemplos os valores hexadecimais e binários exibidos na Tabela 4.1, a qual mostra os limites mínimos e máximos dos valores positivos e negativos que podem ser representados em um *word*.

Note que os valores positivos estão situados na faixa de **0000h** (valor **0** em decimal) até **7FFFh** (valor **32.767** em decimal) e os valores negativos estão situados na faixa de **8000h** (valor **-32.768** em decimal) até **FFFFh** (valor **-1** decimal).

Observe na tabela a indicação da segunda coluna com a apresentação dos valores binários equivalentes aos valores hexadecimais indicados na primeira coluna.

Tabela 4.1 - Faixa de Valores Positivos e Negativos

Faixa de Valores Positivos e Negativos	
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

Os valores binários negativos possuem o seu décimo sexto *bit* (primeiro valor à esquerda) mais significativo iniciado com o valor binário **1** (consideram-se então valores com sinalização), enquanto os valores binários positivos possuem o décimo sexto *bit* iniciado com o valor binário **0** (consideram-se valores sem sinalização). Por meio do décimo sexto *bit* (considerando-se o uso de 16 bits) o processador reconhece o uso ou não de um número negativo.

Um *word* pode representar valores positivos na faixa numérica decimal de **0** até **65.535**, e pode também representar valores numéricos positivos ou negativos na faixa decimal de **–32.768** até **32.767**. A forma de representação numérica depende do tipo de instrução em uso no processador para que o décimo sexto *bit* seja ou não considerado.

A título de ilustração, apresenta-se a seguir a regra de complementação por dois que estabelece o critério para determinar se um valor é ou não positivo. Considere o fato de converter o valor positivo decimal **10** no valor negativo decimal **–10**. Na esfera decimal, basta multiplicar o valor positivo por **–1**, e pronto. Mas na esfera binária, o processo é diferente.

O valor decimal **10** equivale ao valor hexadecimal **000A** que corresponde ao valor binário **1010**. Para efetuar a conversão do valor decimal **10** no respectivo valor binário negativo, é necessário:

- Converter o valor decimal **10** (valor hexadecimal A) em sua forma binária, ou seja, **1010**.
- Acrescentar à esquerda do valor binário valores **0** para formar um *word* completo, ou seja, no caso do valor **1010**, é necessário acrescentar zeros para formar o tamanho do *word* **0000 0000 0000 1010**.
- Converter o valor a ser definido em binário negativo com a inversão de todos os valores para seus opostos (a esse processo dá-se o nome de complementação – *complementação de base* ou *complemento de 1*), ou seja, o valor binário **0000 0000 0000 1010** deve ser escrito como **1111 1111 1111 0101**. Note que os valores **0** tornaram-se **1** e os valores **1** tornaram-se **0**.
- Somar o valor binário **1** ao valor binário **1111 1111 1111 0101** resulta no valor **1111 1111 1111 0110** que será considerado negativo decimal **–10** ou seu equivalente em hexadecimal **FFF6**. O valor binário **1111 1111 1111 0101** é considerado negativo, pois seu bit mais significativo (décimo sexto bit) está sinalizado com valor 1. Se o valor binário **1111 1111 1111 0101** não for sinalizado (isso ocorre se estiver em uso mais de 16 bits e nesta condição possuir no novo tamanho o bit mais significativo igual a zero) este valor será considerado positivo equivalente a 65525.

De acordo com a mesma estrutura de raciocínio veja, por exemplo, como fica o valor negativo decimal **–10** apresentado em formato positivo. Acompanhe os passos:

- A partir do valor negativo binário **1111 1111 1111 0110** faça a inversão dos valores pela técnica de complementação para obter o valor binário **0000 0000 0000 1001**.
- Some o valor binário **1** ao valor binário **0000 0000 0000 1001** e será obtido o valor **0000 0000 0000 1010**, que será considerado o valor positivo decimal **10** ou seu equivalente em hexadecimal **000A**.

A regra de complementação por dois é realizada em duas etapas (daí a denominação complemento por dois ou complementação por dois). Na primeira etapa inverte-se o valor binário e na segunda soma-se o valor 1 ao número invertido.

4.4 - Cálculos em códigos de máquina

No tópico anterior foram apresentadas instruções para fazer cálculos simples com dois valores numéricos hexadecimais utilizando o comando **H** dos programas **Enhanced DEBUG**. Neste tópico as operações matemáticas serão realizadas

com o armazenamento de valores hexadecimais (códigos escritos em linguagem de máquina) nos registradores gerais **AX** e **BX**, e em linguagem de máquina e não em linguagem *Assembly*, pelo menos por ora.

A ideia geral do que são registradores foi explanada no capítulo anterior. No entanto, é bom lembrar que um registrador tem comportamento semelhante a uma variável em uma linguagem de alto nível.

Para visualizar a estrutura de registradores do processador do computador em uso, é necessário informar no *prompt* dos programas **Enhanced DEBUG** o comando **R** (*register*):

```
R <Enter>
```

Ao acionar o comando **R**, aparece uma listagem como a indicada a seguir:

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PO NC
07E9:0100 C3        RET
```

Em seu computador a primeira linha é idêntica à primeira linha da imagem anterior, mas é bem provável que na segunda e terceira linhas sejam apresentados dados com valores diferentes. A diferença encontrada nos valores se refere à quantidade de memória do computador em uso.

Considere apenas neste momento as quatro informações da primeira linha (lado esquerdo) que fazem menção ao uso dos registradores gerais **AX**, **BX**, **CX** e **DX**, os quais estão definidos com valor hexadecimal **0000**. Os demais itens serão vistos mais adiante.

Todos os registradores apresentados têm um valor numérico hexadecimal de quatro posições (dígitos), os quais representam o armazenamento de um *word*, cada um com o tamanho de 16 *bits*.

Os registradores gerais podem armazenar valores numéricos inteiros sem sinalização de 0 até 65.535 (em decimal) ou valores numéricos inteiros com sinalização de -32.768 até 32.768.

4.4.1 - Adição de valores hexadecimais

O comando **R**, além de apresentar informações sobre o estado dos registradores, possibilita fazer a atribuição de valores a um determinado registrador. Imagine que você queira armazenar o valor hexadecimal **000A** (equivalente ao valor **10** em decimal) no registrador geral **AX**. Neste caso, informe na linha de *prompt* do programa a instrução:

```
R AX 000A <Enter>
```

Em seguida execute o comando:

```
R <Enter>
```

Observe o resultado apresentado na tela de acordo com o trecho sinalizado em negrito a seguir:

```
AX=000A BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PO NC
07E9:0100 C3                RET
```

É possível realizar a entrada de qualquer valor hexadecimal em qualquer um dos registradores existentes pelo comando **R**. Sendo assim, informe para o registrador geral **BX** o valor hexadecimal **0001** (equivalente ao valor **1** em decimal), como demonstrado a seguir:

```
R BX 0001 <Enter>
```

Em seguida execute o comando:

```
R <Enter>
```

Observe o resultado apresentado na tela de acordo com o trecho sinalizado em negrito a seguir:

```
AX=000A BX=0001 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PO NC
07E9:0100 C3                RET
```

A partir do momento em que há dois valores armazenados nos registradores gerais **AX** e **BX**, torna-se possível fazer os cálculos matemáticos de adição, subtração, multiplicação e divisão.

Dentre as quatro operações será demonstrado primeiro como somar os valores armazenados nos registradores. Para que essa ação seja efetivada, é necessário colocar na memória do computador dois códigos numéricos (código de máquina) que correspondem à operação de adição.

Observação

O programa **Enhanced DEBUG** tem uma característica que o programa **DEBUG** não tem. Ele permite a entrada direta de um valor em um registrador, usando no *prompt* do programa uma entrada como **R BX 0001** para então acionar a tecla **<Enter>**. No caso do programa **DEBUG** usa-se primeiro o comando **R BX** e após acionar a tecla **<Enter>** informa-se o valor **0001** e aciona-se **<Enter>** novamente.

O problema agora é saber em que lugar da memória serão armazenados os dois códigos (código numérico em linguagem de máquina da operação matemática desejada) que permitirão fazer o cálculo. Por questões técnicas do próprio computador, a memória é segmentada em lotes de 64 KB, denominados *segmentos de dados* (este assunto será tratado mais adiante).

Numa programação em grande escala é necessário saber em que posição de qual segmento de memória uma determinada operação será definida. Pelo fato de estar sendo utilizado o programa **Enhanced DEBUG**, é preciso apenas informar em qual posição (deslocamento) de um determinado segmento (escolhido pelos programas **Enhanced DEBUG**) as instruções de operação de cálculo devem ser definidas.

Fica estabelecido, pela própria engenharia operacional do programa **Enhanced DEBUG**, o uso para a entrada dos códigos em linguagem de máquina do endereço de memória a partir de **0100** (valor em hexadecimal que representa o *byte* de deslocamento a partir do endereço inicial do segmento de memória em uso, neste caso **07E9**, podendo muitas vezes ser definido pelo programa outro segmento de memória). Como é necessário informar dois códigos de máquina, o primeiro será guardado no endereço hexadecimal **0100** e o segundo no endereço hexadecimal **0101**.

Para a entrada de valores em um determinado endereço de memória, deve-se utilizar o comando **E** (*enter*). Neste caso, informe na linha de *prompt* do programa a instrução:

```
E 0100 <Enter>
```

O programa apresenta algo semelhante as linhas seguintes:

```
07E9:0100  C3.
```

O valor **07E9** (que em seu computador pode a vir a ser outro) apresentado antes do valor de deslocamento **0100** mostra o segmento de memória eleito pelos programas **Enhanced DEBUG**, o qual será usado para a ação aritmética ser implementada. Observe que, ao lado direito do endereço, aparece um valor antes de um ponto. Esse valor pode também ser diferente em seu computador. Ao lado do símbolo de ponto informe o código hexadecimal:

```
03 <Enter>
<Enter>
```

Em seguida, na linha de *prompt* do programa, informe a instrução:

```
E 0101 <Enter>
```

O programa apresenta então algo semelhante a:

```
07E9:0101  00.
```

Ao lado do símbolo de ponto, informe o código hexadecimal:

```
C3 <Enter>
<Enter>
```

Os valores em hexadecimal **03** e **C3** representam os códigos em linguagem de máquina responsáveis pela operação de adição dos valores armazenados nos registradores gerais **AX** e **BX**, mais precisamente da adição do valor do registrador geral **BX** com o valor definido no registrador geral **AX**. Esses códigos em hexadecimal são os *opcodes* (códigos operacionais) que permitem o controle operacional de um microprocessador em linguagem de máquina. Em linguagem de programação *Assembly* o código de operação em linguagem de máquina (*opcode*) **03C3** faz a soma (**ADD**) entre os conteúdos dos registradores **AX** e **BX**.

Em seguida execute o comando **R** para que seja apresentado o estado atual dos registradores:

```
R <Enter>
```

Observe atentamente os trechos em negrito como é mostrado em seguida:

```
AX=000A BX=0001 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PO NC
07E9:0100 03C3              ADD     AX,BX
```

Atente para os pontos marcados em negrito. Os registradores gerais **AX** e **BX** apresentam, respectivamente, os valores hexadecimais **000A** e **0001**. Observe a mudança ocorrida na terceira linha, a qual mostra o segmento em uso, no caso **07E9**, e o endereço de deslocamento **0100** (**07E9:0100**) definido com o par de códigos de máquina **03C3** (que foram informados separadamente), os quais representam a operação de adição identificada pela instrução **ADD** (*addition*) que será executada sobre os registradores gerais **AX** e **BX**. Note que a operação de um computador em linguagem de máquina via processador ocorre com instruções armazenadas em forma de números, neste caso, hexadecimais.

Em seguida é necessário informar ao processador onde encontrar na memória os códigos de ação da operação de adição. Todo o trabalho anterior foi apenas para entrar os valores e determinar a operação que será ainda realizada. Para a ação desejada o processador precisa saber o segmento e o deslocamento que devem ser utilizados, ou seja, onde está a instrução de processamento. Essas duas informações foram armazenadas nos registradores de apontamentos **CS** (*code segment* - registrador de segmento) e **IP** (*instruction pointer* - registrador de deslocamento), como pode ser notado nos trechos em negrito. Note também a terceira linha com as posições de segmento **07E9** (que pode ser diferente em seu computador) e de deslocamento **0100** marcadas em negrito.

Observação

Caso o registrador de apontamento **CS** e o registrador de deslocamento **IP** não apontem para os endereços definidos, respectivamente, para o segmento e para o deslocamento, eles necessitam ser informados manualmente por intermédio do comando **R**, de forma semelhante à entrada de valores nos registradores gerais **AX** e **BX**.

Assim que todas as verificações são feitas, basta pedir para o programa **Enhanced DEBUG** executar com o comando **T** (*trace*). Observe que o programa tem apenas uma instrução (**ADD AX,BX**):

```
T <Enter>
```

Após executar o comando **T** será efetuado o cálculo da adição e apresentado o estado dos registrados, semelhantemente à forma seguinte:

```
AX=000B BX=0001 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0102 NV UP EI PL NZ NA PO NC
07E9:0102 0000              ADD     [BX+SI],AL                      DS:0001=20
```

Após a execução do comando **T,** observe os pontos marcados em negrito. O valor da adição foi armazenado no registrador geral **AX** (como sendo o valor hexadecimal **000B**, equivalente ao valor **11** em decimal), ou seja, ocorreu uma operação aritmética semelhante a **AX ← AX + BX**.

O valor do deslocamento (*offset*) do registrador de apontamento **IP** está indicando para o endereço de deslocamento **0102**, porque o comando **T** executa as instruções do programa passo a passo, a partir do primeiro até o último deslocamento sem nenhuma instrução. Isso pode ser percebido pela indicação da terceira linha.

Por exemplo, aponte o registrador de deslocamento **IP** para o endereço de deslocamento **0100** novamente. Execute no *prompt* do programa a instrução:

```
R IP 0100 <Enter>
```

Em seguida acione o comando **R** e observe o valor existente no registrador **IP**, como é indicado a seguir em negrito. Note também a alteração da informação da terceira linha.

```
AX=000B BX=0001 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PO NC
07E9:0100 03C3              ADD     AX,BX
```

Execute o comando **T** novamente:

```
T <Enter>
```

O valor hexadecimal **0001** do registrador geral **BX** é somado ao valor hexadecimal **000B** do registrador geral **AX**, colocando no registrador geral **AX** o valor hexadecimal **000C** (equivalente ao valor **12** em decimal), como é indicado a seguir:

```
AX=000C BX=0001 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0102 NV UP EI PL NZ NA PO NC
07E9:0102 0000              ADD     [BX+SI],AL                      DS:0001=20
```

A ocorrência de deslocamento registrada no registrador de deslocamento **IP** indica novamente o posicionamento em **0102**. Se forem repetidas as ações descritas, ter-se-á um acumulador de valores em memória.

Anteriormente foi usado o par de valores de *opcodes* **03** e **C3** para somar o valor do registrador geral **BX** com o valor do registrador geral **AX**. É possível também fazer a operação inversa, ou mesmo efetuar operações com a porção menos significativa dos registradores gerais **AX** e **BX**. A Tabela 4.2 apresenta as operações de adição com o *opcode* usual e também um *opcode* alternativo quando houver. O *opcode* alternativo é uma segunda opção para executar uma determinada ação computacional.

Teste os *opcodes* apresentados na tabela para os valores atuais dos registradores gerais **AX** e **BX**, considerando inclusive as partes menos significativas dos respectivos registrados gerais mencionados, como foram os valores apresentados (valores de um *byte*). Caso queira entrar um valor que ocupe um *word*, faça-o utilizando todas as posições numéricas. Por exemplo, experimente os valores **1111** e **2222** para os Registradores gerais **AX** e **BX**.

Tabela 4.2 - Opcode usual e alternativo para adição

Operação Opcode		Adição	
Usual	Alternativo	Registradores Gerais	Operação
02 C3	00 D8	AL, BL	AL ← AL + BL
03 C3	01 D8	AX, BX	AX ← AX + BX
02 D8	-	BL, AL	BL ← BL + AL
03 D8	-	BX, AX	BX ← BX + AX

A instrução **ADD** quando executada altera o estado dos registradores de estados (*flags*) **AF**, **CF**, **OF**, **PF**, **SF** e **ZF**, possuindo como possibilidade de trabalho as operações sobre:

- registrador, registrador (para este tipo de ação usa 2 *bytes* de memória);
- memória, registrador (para este tipo de ação usa de 2 a 4 *bytes* de memória);
- registrador, memória (para este tipo de ação usa de 2 a 4 *bytes* de memória);
- registrador, constante (para este tipo de ação usa de 3 a 4 *bytes* de memória);
- memória, constante (para este tipo de ação usa de 3 a 6 *bytes* de memória).

Encerre a execução do programa **Enhanced DEBUG**.

4.4.2 - Subtração de valores hexadecimais

Carregue para a memória o programa **Enhanced DEBUG** e para a subtração de valores considere o fato de fornecer para os registradores gerais **AX** e **BX**, respectivamente, os valores hexadecimais **000A** e **0002**. Informe na linha de *prompt* do programa a instrução:

```
R AX 000A <Enter>
R BX 0002 <Enter>
R IP 0100 <Enter>
```

Ao lado do símbolo de dois-pontos informe o valor hexadecimal **000A** e acione a tecla **<Enter>**. Acione o comando **R** e observe o valor existente no registrador geral **AX**, como é indicado a seguir em negrito.

Acione o comando **R** e observe os valores existentes nos registradores gerais **AX** e **BX**, como é indicado a seguir em negrito.

```
AX=000A BX=0002 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PE NC
07E9:0100 C3                RET
```

O processo de subtração é conseguido com o par de códigos *opcodes* (valores hexadecimais) **2B** e **C3**, os quais devem ser armazenados, respectivamente, nos endereços de deslocamento **0100** e **0101**. Para efetuar a entrada de valores de endereço, utilize novamente o comando **E**:

```
E 0100 2B C3 <Enter>
```

Observe que os códigos hexadecimais **2B** e **C3** são responsáveis pela operação de subtração dos valores armazenados nos registradores gerais **AX** e **BX**. Veja a seguir o detalhamento completo da operação anterior.

Na sequência acione o comando **R** para que seja apresentado o estado atual dos registradores, como é mostrado em seguida:

```
AX=000A BX=0002 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PE NC
07E9:0100 2BC3              SUB     AX,BX
```

As informações dos registradores **AX**, **BX** e **IP** já são conhecidas. Atente especialmente para a terceira linha, na qual se encontra a instrução **SUB** (*subtract*), indicando que será executada a subtração dos valores armazenados nos registradores gerais **AX** e **BX** (**SUB AX,BX**), ou seja, ocorrerá a operação aritmética semelhante a **AX ← AX – BX**. Na sequência execute o comando **T** e observe o cálculo da subtração, como indicado a seguir:

```
AX=0008 BX=0002 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0102 NV UP EI PL NZ NA PO NC
07E9:0102 0000              ADD     [BX+SI],AL                      DS:0002=FF
```

Observe que o *flag* PE é mostrado como PO a partir da ação de uma subtração. Mais adiante maiores detalhes serão dados em relação aos registradores de estado (*flags*).

Seria conveniente agora fazer um cálculo de subtração para obter um resultado negativo. Entre para o registrador geral **BX** o valor hexadecimal **000A** e ajuste o endereço do registrador de deslocamento **IP** para **0100** com as instruções:

```
R BX 000A <Enter>
R IP 0100 <Enter>
```

Em seguida execute o comando **T** e será efetuada a subtração com resultado negativo. Observe atentamente o resultado **FFFE** apresentado no registrador **BX** e a mudança dos registradores de *flags* PL (anteriormente sinalizado como NG), NA (anteriormente sinalizado como AC) e CY (anteriormente sinalizado como NC) marcados em negrito e indicados a seguir:

```
AX=FFFE BX=000A CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI NG NZ AC PO CY
07E9:0102 0000              ADD     [BX+SI],AL                      DS:000A=39
```

As alterações sobre os registradores de *flag* **PL** para NG, **NA** para AC e **NC** para CY ocorreram porque a subtração calculada gerou um resultado negativo. Neste momento, não se preocupe com esses detalhes.

A apresentação do valor hexadecimal **FFFE** no registrador geral **BX** equivale ao valor decimal **-2**. Lembre-se do estouro que ocorre para a representação de valores negativos.

Anteriormente foi usado o par de valores de *opcodes* **2B C3** para subtrair o valor do registrador geral **BX** do valor do registrador geral **AX**. É possível também realizar a operação inversa, ou mesmo operações com a porção menos significativa dos registradores gerais **AX** e **BX**. A Tabela 4.3 mostra as operações de subtração possíveis.

Tabela 4.3 - Opcode usual e alternativo para subtração

Operação Opcode		Subtração	
Usual	**Alternativo**	**Registradores Gerais**	**Operação**
2A C3	28 D8	AL, BL	AL ← AL - BL
2B C3	29 D8	AX, BX	AX ← AX - BX
2A D8	-	BL, AL	BL ← BL - AL
2B D8	-	BX, AX	BX ← BX - AX

Teste os *opcodes* apresentados na tabela para os valores atuais dos registradores gerais **AX** e **BX**, considerando inclusive as partes menos significativas dos respectivos registradores gerais mencionados, como foram os valores apresentados (valores de um *byte*).

A instrução **SUB** quando executada altera o estado dos registradores de estados (*flags*) **AF**, **CF**, **OF**, **PF**, **SF** e **ZF**, possuindo como possibilidade de trabalho as operações sobre:

- registrador, registrador (para este tipo de ação usa 2 *bytes* de memória);
- memória, registrador (para este tipo de ação usa de 2 a 4 *bytes* de memória);
- registrador, memória (para este tipo de ação usa de 2 a 4 *bytes* de memória);
- registrador, constante (para este tipo de ação usa de 3 a 4 *bytes* de memória);
- memória, constante (para este tipo de ação usa de 3 a 6 *bytes* de memória).

Encerre a execução do programa **Enhanced DEBUG**.

4.4.3 - Multiplicação de valores hexadecimais

Carregue para a memória o programa **Enhanced DEBUG** e para a multiplicação de valores considere o fato de fornecer para os registradores gerais **AX** e **BX**, respectivamente, os valores hexadecimais **0005** e **0003**. Informe na linha de *prompt* do programa as instruções:

```
R AX 0005 <Enter>
R BX 0003 <Enter>
R IP 0100 <Enter>
```

Acione o comando **R** e observe os valores existentes nos registradores gerais **AX** e **BX** e no registrador **IP** como é indicado a seguir em negrito.

```
AX=0005 BX=0003 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PO NC
07E9:0100 C3                RET
```

O processo de multiplicação é conseguido com o par de códigos *opcodes* (valores hexadecimais) **F7** e **E3**, os quais devem ser armazenados, respectivamente, nos endereços de deslocamento **0100** e **0101**. Assim sendo, execute a instrução:

```
E 0100 F7 E3 <Enter>
```

Execute o comando **R** para que seja apresentado o estado atual dos registradores, como é mostrado em seguida:

```
AX=0005 BX=0003 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PO NC
07E9:0100 F7E3              MUL     BX
```

As informações dos registradores **AX**, **BX** e **IP** já são conhecidas. Atente especialmente para a terceira linha, na qual se encontra a definição da instrução **MUL** (*multiply*) apontando para o registrador geral **BX** (**MUL BX**).

Execute o comando **T** para que o resultado da multiplicação seja armazenado no registrador geral **AX** (neste caso será armazenado o valor hexadecimal **000F**) e para que as informações dos registradores sejam apresentadas.

```
AX=000F BX=0003 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0102 NV UP EI PL NZ NA PO NC
07E9:0102 0000              ADD     [BX+SI],AL                      DS:0003=9F
```

No exemplo anterior foi realizada a multiplicação de dois valores numéricos hexadecimais muito pequenos. Mas imagine se a multiplicação ocorresse entre os valores bem maiores. Neste caso, deve-se considerar que a multiplicação de dois valores numéricos de 16 *bits* (*word*) pode resultar num valor numérico de resposta na casa de 32 *bits* (*double word*).

Por exemplo, informe para o registrador geral **AX** o valor hexadecimal **7D3C** (equivalente ao valor **32.060** em decimal) e para o registrador geral **BX** o valor hexadecimal **0100** (equivalente ao valor **256** em decimal). Depois execute o comando **R IP** e forneça como endereço de deslocamento inicial o valor hexadecimal **0100** de acordo com as instruções:

```
R AX 7D3C <Enter>
R BX 0100 <Enter>
R IP 0100 <Enter>
```

Na sequência acione o comando **R** para que seja apresentado o estado atual dos registradores, depois acione o comando **T** para que a multiplicação seja processada. Veja a seguir a sequência de resultados:

```
AX=7D3C BX=0100 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI PL NZ NA PO NC
07E9:0100 F7E3              MUL     BX

AX=3C00 BX=0100 CX=0000 DX=007D SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0102 OV UP EI NG NZ AC PO CY
07E9:0102 0000              ADD     [BX+SI],AL                  DS:0100=F7
```

Observe também a mudança ocorrida nos registradores de *flags* **OV** (anteriormente sinalizado como NV) e **CY** (anteriormente sinalizado como NC). Mais adiante serão dados mais detalhes sobre estas ocorrências.

O resultado da multiplicação é muito grande para ser armazenado apenas no registrador geral **AX**. A leitura do valor calculado deve ser feita na sequência de registradores gerais **DX** e **AX**, ou seja, **DX:AX**. O registrador geral **DX** (que armazena os 16 *bits* mais significativos - mais altos - do resultado de uma multiplicação) possui o valor hexadecimal **007D** e o registrador geral **AX** (que armazena os 16 *bits* menos significativos - mais baixos - do resultado de uma multiplicação) possui o valor hexadecimal **3C00**, o que resulta um valor hexadecimal de 32 *bits* representado como **007D3C00** (equivalente ao valor **8.207.360** decimal), ou seja, ocorre a operação aritmética semelhante a **DX:AX ← AX * BX**.

Anteriormente foi usado o par de valores de *opcodes* **F7 E3** para multiplicar o valor do registrador geral **BX** pelo valor do registrador geral **AX**. A Tabela 4.4 apresentada as operações de multiplicação possíveis.

Tabela 4.4 - Opcode usual para subtração

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

Teste os *opcodes* apresentados na tabela para os valores atuais dos registradores gerais **AX** e **BX**, considerando inclusive as partes menos significativas dos respectivos registrados gerais mencionados, como foram os valores apresentados (valores de um *byte*). Caso queira entrar um valor que ocupe um *byte*, faça-o utilizando as posições numéricas da direita. Por exemplo, experimente multiplicar os valores **007F** e **0002** para os registradores gerais **AX** e **BX** e forneça o par de *opcodes* **F6 E3**, que resulta o valor **00FE** no registrador geral **AX**.

A instrução **MUL** quando executada altera o estado dos registradores de estados (*flags*) **CF** e **OF**, possuindo como possibilidade de trabalho as operações sobre:

- registrador (para este tipo de ação usa 2 *bytes* de memória);
- memória (para este tipo de ação usa de 2 a 4 *bytes* de memória).

Encerre a execução do programa **Enhanced DEBUG**.

4.4.4 - Divisão de valores hexadecimais

Carregue para a memória o programa **Enhanced DEBUG** e para a divisão de dois valores considere o fato de fornecer para os registradores gerais **AX** e **BX**, respectivamente, os valores hexadecimais **0009** e **0002**. Assim sendo, informe as instruções:

```
R AX 0009 <Enter>
R BX 0002 <Enter>
R IP 0100 <Enter>
```

Acione o comando **R** e observe os valores existentes nos registradores gerais **AX** e **BX**, como é indicado a seguir em negrito.

```
AX=0009 BX=0002 CX=0000 DX=007D SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI NG NZ NA PO NC
07E9:0100 C3                RET
```

O processo de divisão é conseguido com o par de códigos em linguagem de máquina (valores hexadecimais) **F7** e **F3**, os quais devem ser armazenados, respectivamente, nos endereços de deslocamento **0100** e **0101**. Desta forma, execute a instrução:

```
E 0100 F7 F3 <Enter>
```

Acione o comando **R** para que seja apresentado o estado atual dos registradores, como indicado em seguida:

```
AX=0009 BX=0002 CX=0000 DX=007D SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0100 NV UP EI NG NZ NA PO NC
07E9:0100 F7F3              DIV     BX
```

As informações dos registradores **AX**, **BX** e **IP** já são conhecidas. Atente especialmente para a terceira linha, na qual se encontra a definição da instrução **DIV** (*divide*) apontando para o registrador geral **BX** (**DIV BX**).

Execute o comando **T** para que o resultado da divisão seja armazenado nos registradores gerais **AX** e **DX**, como segue:

```
AX=0004 BX=0002 CX=0000 DX=0001 SP=FFFE BP=0000 SI=0000 DI=0000
DS=07E9 ES=07E9 SS=07E9 CS=07E9 IP=0102 NV UP EI NG NZ NA PO NC
07E9:0102 0000              ADD     [BX+SI],AL                      DS:0002=FF
```

O resultado da divisão (quociente) é armazenado no registrador geral **AX,** enquanto o resultado do resto é armazenado no registrador geral **DX**. Assim como a multiplicação, a operação de divisão utiliza o par de registradores gerais **AX** e **DX**.

Anteriormente foi usado o par de valores de *opcodes* **F7 F3** para efetuar a multiplicação do valor do registrador geral **BX** pelo valor do registrador geral **AX**. A Tabela 4.5 apresentada as operações de divisão possíveis.

Teste os *opcodes* apresentados na tabela para os valores atuais dos registradores gerais **AX** e **BX**, considerando inclusive as partes menos significativas dos respectivos registrados gerais mencionados, como foram os valores apresentados (valores de um byte). Caso queira entrar um valor que ocupe um byte, faça-o utilizando as posições numéricas da direita. Por exemplo, experimente multiplicar os valores hexadecimais **0009** e **0002** para os registradores gerais **AX** e **BX** e forneça o par de *opcodes* **F6 F3**, que resulta o valor hexadecimal 0104 no registrador geral **AX**, em que o valor **04** (lado menos significativo) é o quociente da operação e o valor **01** (lado mais significativo) é o resto da divisão.

Tabela 4.5 - Opcode usual para divisão

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

A instrução **DIV** quando executada não altera o estado dos registradores de estados (*flags*), possuindo como possibilidade de trabalho as operações sobre:

- registrador (para este tipo de ação usa 2 *bytes* de memória);
- memória (para este tipo de ação usa de 2 a 4 *bytes* de memória).

Encerre a execução do programa **Enhanced DEBUG**.

Anotações