|  |  |
| --- | --- |
|  |  |

PONTIFÍCIA UNIVERSIDADE CATÓLICA

DO RIO GRANDE DO SUL – PUCRS

FACULDADE DE INFORMÁTICA

**Tutorial Arquitetura Intel 8086**

**Arquitetura de Computadores I**

**PROF.EDSON IFARRAGUIRRE MORENO**

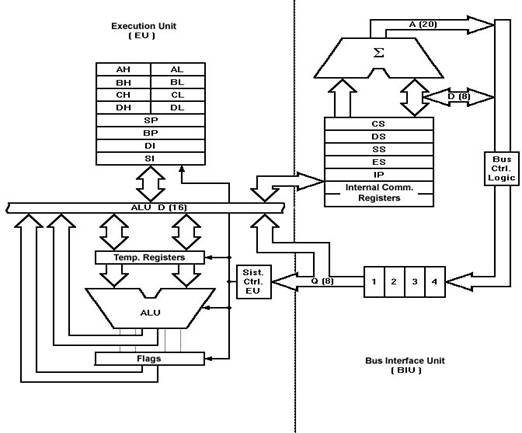
Daniel Centeno Einloft

Porto Alegre, 05 de maio de 2016.

# **Introdução**

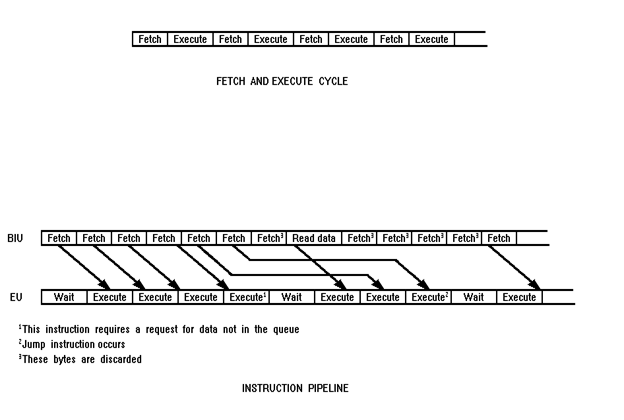
O 8086 é um microprocessador de 16 bits criado pela Intel em 1978. Muito famoso pela compatibilidade reversa, a arquitetura do 8086 é organizada em duas partes: a Unidade de Execução ( *Execution Unit*, EU) e Unidade de Interface de Barramento ( *BUS Interface Unit*, BUI).

A BIU é responsável pelas funções de I/O do processador, possuindo uma variedade de registradores e barramentos dedicados à operações de acesso à memória e troca de dados com periféricos. Esta unidade é encarregada de enviar informações de dados para a outra unidade, a EU, que possui a finalidade de executar as instruções e armazenar os resultados nos registradores de propósito geral. A Figura 1 mostra como é feita esta organização.



**Figura 1**: Arqutietura do 8086, dividida na EU e BIU.

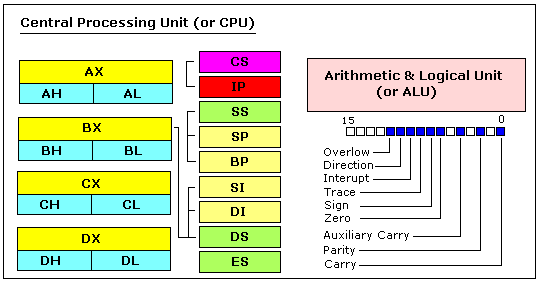
Esta organização garante que a computação não seja influenciada pelo *delay* do acesso à memória. A BIU vai acessando as insturções e colocando no barramento de instruções, em quanto que a EU vai acessado as intruções direto do barramento, com um tempo de acesso muito menor. O fluxo de “*Fetch*” e “*Execute*” só é interrompido quando a) EU precisa acessar uma posição de memória não localizada na pilha de endereços provida pela BIU, b) em casos de saltos e c) na execução de instruções que demoram mais que o previsto para executar, como divisões. A Figura 2 mostra uma sequência de operações das duas unidades.



**Figura 2**: Sequência de Fetch e Execute da BIU e EU.

# **Registradores do 8086**

O 8086 possui uma série de registradores que podem ou não ser acessados pelo programador. A Figura 3 mostra todos os registradores disponíveis nesta arquitetura.



**Figura 3**: Conjunto de registradores do 8086.

Os registradores de próposito geral são:

* **AX**: acumulador base, dividido em AH e AL, ambos de 8 bits.
* **BX**: endereço base, dividido em BH e BL, ambos de 8 bits.
* **CX**: contador base, dividido em CH e CL, ambos de 8 bits.
* **DX**: registrador de dados, dividido em DH e DL, ambos de 8 bits.
* **SI**: Source Index.
* **DI**: Destination Index.
* **BP**: Base Pointer.
* **SP**: Stack Pointer.

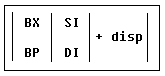
Embora estes registradores possuirem nomes para operações específicas, eles podem ser manipulados pelo programador de acordo com as necessidades. Existem também quatro registradores de Registradores de Segmento:

* **CS**: Aponta para o segmento que possui o programa atual na memória.
* **DS**: Aponta para segmento onde as variáveis estão armazenadas na memória.
* **ES**: Registrador de Segmento extra, que pode ser utilizado como o programador quiser.
* **SS**: Aponta para o segmento que contém a pilha.

Finalmente, exisite mais um tipo de registradores na arquitetura, chamados Registradores de Propósito Especial. Estes registradores possuem propósitos especificos e não são manipulados pelo programador. O registrador IP aponta para a próxima instrução a ser executada, em quanto o Flags Register possui diversas funcionalidades, cada bit para uma função específica. Como a Figura 3 mostra, ele armazena valores como Overflow, Direção, Interrupção, Carry, etc. A forma de manipulação desses registradores vão ser explicadas nos capítulos a seguir.

# **Acesso à Memória**

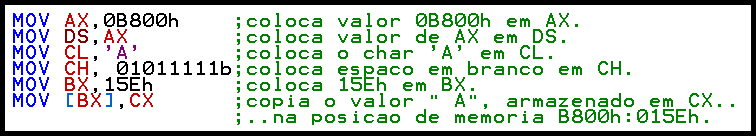
Para acessar a memória, normalmente é usado os registradores BX, SI, DI e BP. É possível utilizar também valores imediatos para acessar a memória, fazendo deslocamento de 8 ou 16 bits (CUIDADO: VALORES IMEDIATOS POSSUEM SINAL). Um fator importante que deve ser notado é a utilização do registrador DS (*Data Segment*), que é utilizado sempre que for acessar uma posição de memória. Outra característica é que os registradores BX (*base address*) e BP (*base pointer*) não podem ser utilizados juntos, da mesma forma que os registradores SI (*source index*) e DI (*destination index*) também não podem. A Figura 4 mostra as possibilidades de combinação de registradores, sendo que não é possível utilizar dois registradores da mesma coluna para acessar memória.



**Figura 4:** Combinações possíveis para acesso à memória.

Por exemplo, se os registradores possuírem os seguintes valores: DS = 100, BX =30 e SI = 70. Se deseja-se acessar a memória na posição [BX + SI] + 25 (deslocamento), a posição de memória física acessada será: 100\*16 + 30 + 70 + 25 =1725. Logo, o acesso à memória é feito pela adição do valor no registrador de segmento, multiplicado por 16 (10h), com os valores de offset. É comum encontrar em tutoriais a representação do acesso à memória da seguinte forma: **SEGMENTO:OFFSET**.

Para acessar a memória, de fato, normalmente é utilizado a instrução **MOV**. Esta instrução serve para mover um valor de um registrador/memória para registrador/memória. Logo, os operandos suportados são: MOV REG,MEM; MOV MEM,REG; MOV REG,REG; MOV MEM,IMEDIATO; MOV REG,IMEDIATO. A única restrição é que os operandos devem possuir o mesmo tamanho. A Figura 5 mostra um exemplo de programa que acessa à memória.



**Figura 5**: Exemplo de código utilizando a instrução MOV.

# **Criação e Manipulação de Variáveis**

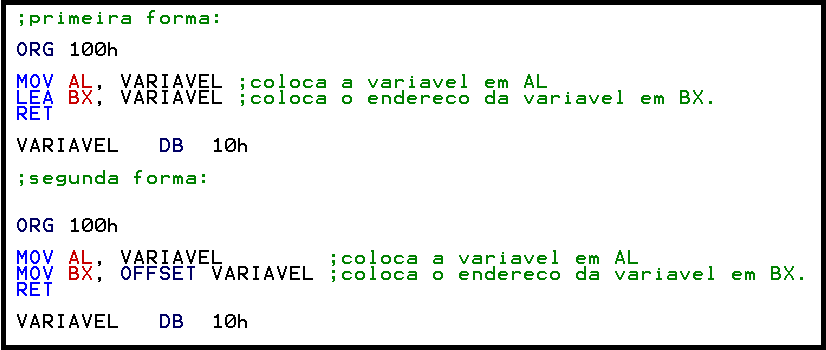
Uma variável é, na prática, uma posição de memória com um *label* (no processo de compilação, o label é substituído pela posição de memória). No assembly do 8086, variáveis podem ser de dois tipos: BYTE ou WORD. A sintáxe de declaração é: nome\_da\_variável **DB/DW** valor\_da\_variável, sendo que DB e DW representam Define Byte e Define Word, respectivamente. O nome da variável pode ser uma sequência de números ou letras (mas deve iniciar com letra) e o valor pode ser qualquer valor decimal, hexadecimal, binário ou não definido (sendo representado por ‘?’). Por exemplo: **cinco DB 5h**.

Da mesma forma que se declara variáveis de um byte, é possível declarar vetores (*arrays*). Para a declaração de um vetor, basta colocar vários valores em sequência ( como “**a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h**” ou “ **b DB 'Hello', 0**”). O compilador entende isso como uma sequência de bytes, que podem ser acessados pelo programa da mesma forma que são acessados na linguagem C ( a[2], variável a, posição 2). Por último, é possível também declarar constantes, com a diferetiva **EQU** ao invés de DB ou DW (exemplo: **constante EQU 3**). Vale ressaltar que os valores de constantes não podem ser modificados.

No processo de compilação, o compilador troca os *labels* das variáveis pelos seus respectivos *offsets*, que indicarão aonde os dados estarão localizados na memória. A posição de memória onde os dados começam a ser guardados é indicado pelo registrador DS (*Data Segment Register*).

Mas aonde os dados começam a ser armazenados, de fato? Como o compilador sabe aonde colocar as informações? Para isso, é utilizado a diretiva **ORG 100h** (para saídas de arquivos do tipo .COM). É a partir desta diretiva que o compilador vai calcular a posição correta de cada variável em memória. É importante ressaltar que isto só é valido para arquivos de saída do tipo .COM, onde os dados do programa são armazenados a partir do byte 256 (100h). Para saídas do tipo .EXE, o *offset* inicial é 000h.

Para acessar o endereço de memória desejado, pode ser utilizado a instrução **LEA**, que retorna o endereço da variável. Outra alternativa é o uso da difetiva **OFFSET**, que possui o mesmo funcionamento (mas pode ser utilizado “dentro” de uma instrução). A Figura 6 mostra as duas formas de manipulação de variáveis.



**Figura 6:** Duas formas de acesso às variáveis.

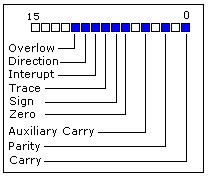
# **Operações Lógicas e Aritméticas**

Existem três tipos de operações lógicas e aritméticas na arquitetura do 8086. O primeiro tipo consiste em operações mais básicas entre dois valores, como soma e subtração (**ADD** e **SUB**, operações aritméticas simples) e e,or,xor (**AND**,**OR**,**XOR**, operações lógicas simples). Os operandos destas instruções podem ser: registrador, memória; memória, registrador; registrador, registrador; memória,imediato;registrador, imediato. A instrução **CMP** (compare), que vai ser muito utilizada para operações de saltos condicionais, também segue este modelo.

O segundo grupo consiste de operações de multiplicação e divisão, definidas pelas instruções **MUL**, **IMUL** (multiplicação com sinal), **DIV**, **IDIV** (divisão com sinal). Estas instruções aceitam apenas um operando (o multiplicador/divisor), podendo ser ou um registrador, ou uma posição de memória. Estas operações multiplicam os valores armazenados em AL (se operando for um byte) ou AX (se operando for uma palavra).

O terceiro grupo consiste em operações lógicas e aritméticas sobre um valor só. Estas operações consistem em incrementar, decrementar e negativar (**INC**, **DEC** e **NEG**, respectivamente, operações aritméticas) e negação lógica (**NOT**). Como as operações do segundo grupo, estas instruções aceitam apenas um operando, que pode ser um registrador ou posição de memória.

Um fator importante destas operações é que elas podem apresentar problemas como *overflow* e *carry*. O 8086 possui um registrador de status (ou *flags*), onde cada bit possui um significado diferente e podem ser modificados com a utilizações destas instruções. A Figura 7 mostra todos as *flags* presentes na presente arquitetura.



**Figura 7:** *Flags* do 8086

Logo, as instruções do primeiro grupo e a instrução **NEG** podem afetar as flags de *Overflow*, *Carry*, Zero, Sinal, Paridade e Auxiliar (para *overflow* sem sinal). As instruções DEC e INC podem afetar as flags de Zero, Sinal, *Overflow*, Paridade e Auxiliar. Por último, as multiplicações afetam os registradores *Carry* e *Overflow*. A Figura 8 mostra um exemplo de código que utiliza algumas destas instruções.

\*FIGURA 8\*