Unidade: Modos de Endereçamento e Linguagem de Montagem

Unidade: Modos de Endereçamento e Linguagem de Montagem

1) Modos de endereçamento.

Grande parte das instruções deve buscar informações (operando) na memória ou em algum dispositivo de entrada e saída, portanto, é preciso definir algum modo para que essas informações sejam levadas para processamento; a estes modo de operação chamamos de endereçamento. Há várias instruções que trabalham com endereçamento, vamos avaliar abaixo alguns destes modos.

1.1) Endereçamento imediato

É considerado o modo mais simples de endereçamento, pois em vez de indicar uma posição de memória para pegar instrução de onde está o operando, ela aponta direto para o operando. Neste caso, a parte da instrução que deveria ter o endereço de onde está posicionado o operando, tem armazenada a informação do próprio operando. Esse tipo de instrução não faz referência extra à memória, e sim, a um valor numérico considerado o próprio operando.

Neste tipo de endereçamento, conforme apresentado na figura 1, o contador de programas recebe o conteúdo indicado na memória. Veja que antes o contador de programas (PC) tinha o valor 2007, e depois foi alterado para o valor contido na memória (30 A1).

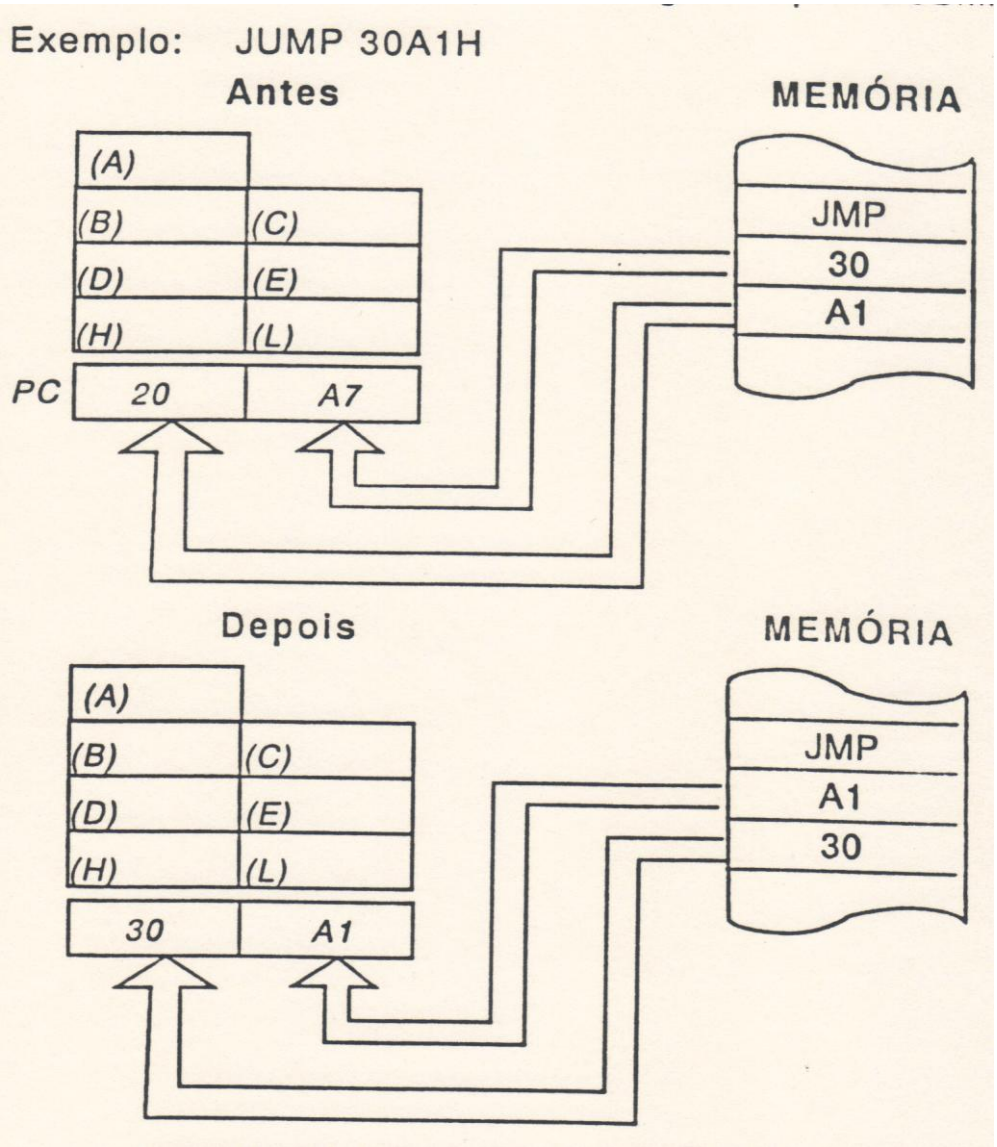

Figura 1: Endereçamento imediato.

1.2) Endereçamento direto

Neste caso, a instrução tem o endereço completo de onde está posicionado o operando. Este tipo de endereçamento é usado ao se definirem a um programa uma variável global, pois o conteúdo pode ser alterado, mas a posição de memória será sempre a mesma.

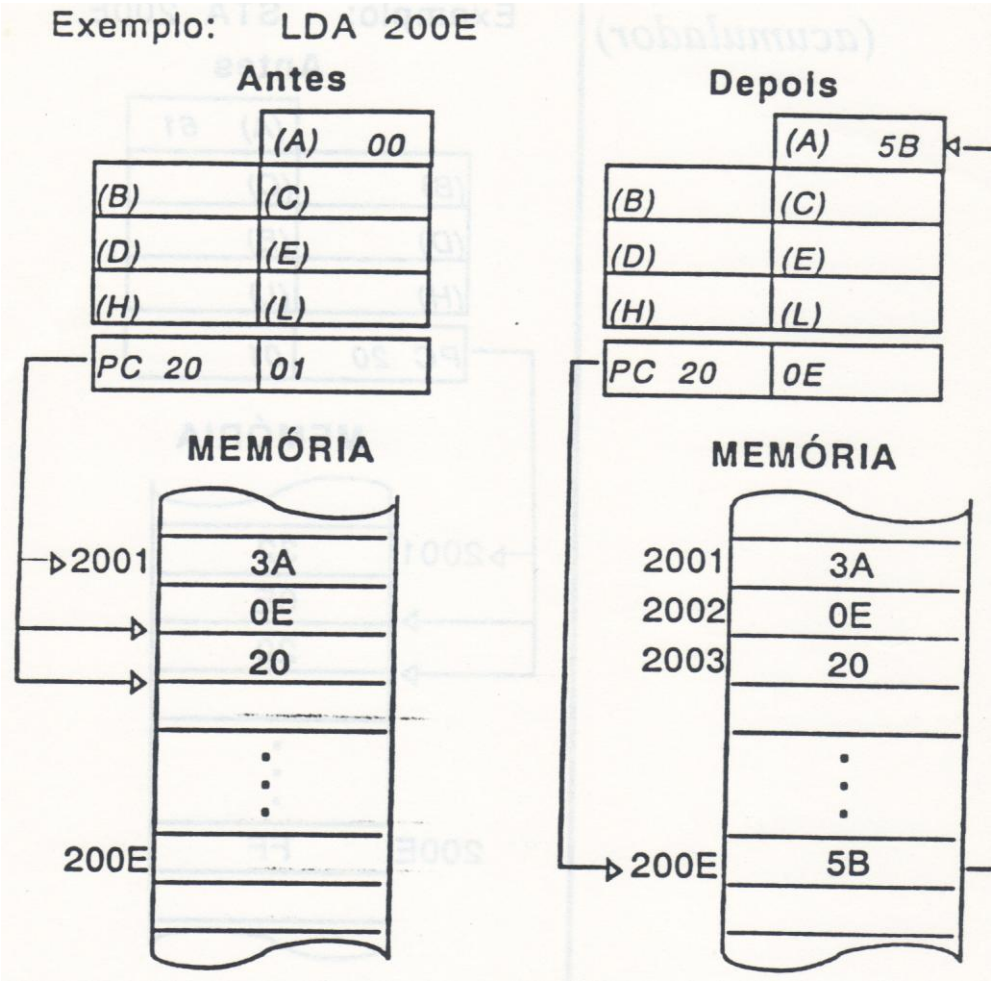

Figura 2: Endereçamento direto.

Veja na figura acima que o contador de programas aponta para o endereço de memória 200E. Neste endereço, está o conteúdo que deve ser enviado ao acumulador - esta ação pode ser verificada do lado direito da figura – portanto, o conteúdo do endereço de memória 200E é carregado para o registrador “A”. Neste caso, o conteúdo 5B é carregado para o acumulador (registrador A).

1.3) Endereçamento de registrador

Este tipo de endereçamento é equivalente ao endereçamento direto, no entanto, em vez da instrução carregar o endereço de memória, ela especifica um registrador. É considerado o endereçamento mais comum na maioria dos computadores, pois se beneficiam da velocidade e endereços curtos inseridos nos registradores.

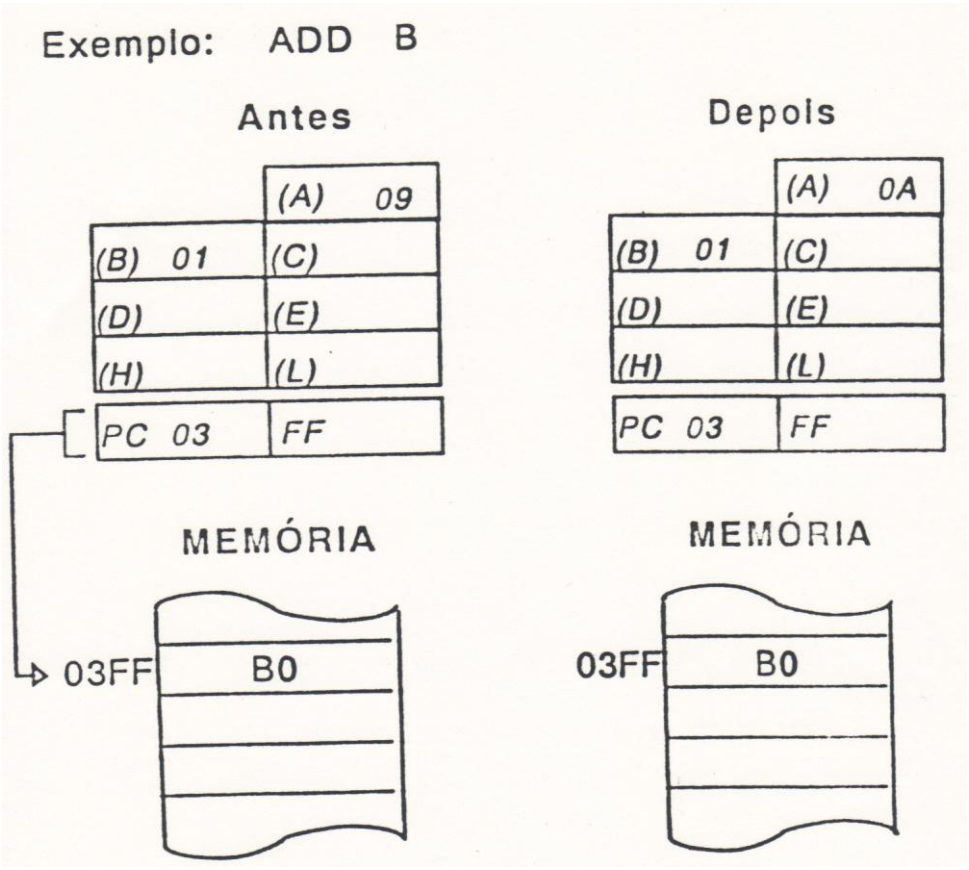

Figura 3: Endereçamento de registrador.

A figura acima demonstra um endereçamento de registrador: o exemplo mostra como ficam os registradores e a memória antes de uma operação de soma (ADD B) e depois da soma. Neste tipo de operação, o valor contido no registrador “B” será somado ao conteúdo do registrador “A”. Perceba, pela figura 3, que antes da soma temos o valor 09 no acumulador (registrador A) e o valor 01 no registrador “B”; o contador de programa está apontando para o endereço 03FF cujo conteúdo é “B0”. Depois da adição, você pode notar que o acumulador trocou seu valor para “0A”, ou seja, foi adicionado o valor 01 ao valor que tinha no acumulador; portanto, seu valor agora é 0A, e os outros acumuladores e a memória não foram alterados.

1.4) Endereçamento indireto de registrador

Neste modo de endereçamento, o operando que está sendo especificado vem da memória ou então vai para memória; mas seu endereço não está ligado à instrução, como vimos no caso do endereçamento direto. A instrução aponta para um registrador que contém o endereço de onde o

operando está posicionado. Uma vantagem desse tipo de endereçamento é que os registradores armazenam parte do endereço para acessar o operando.

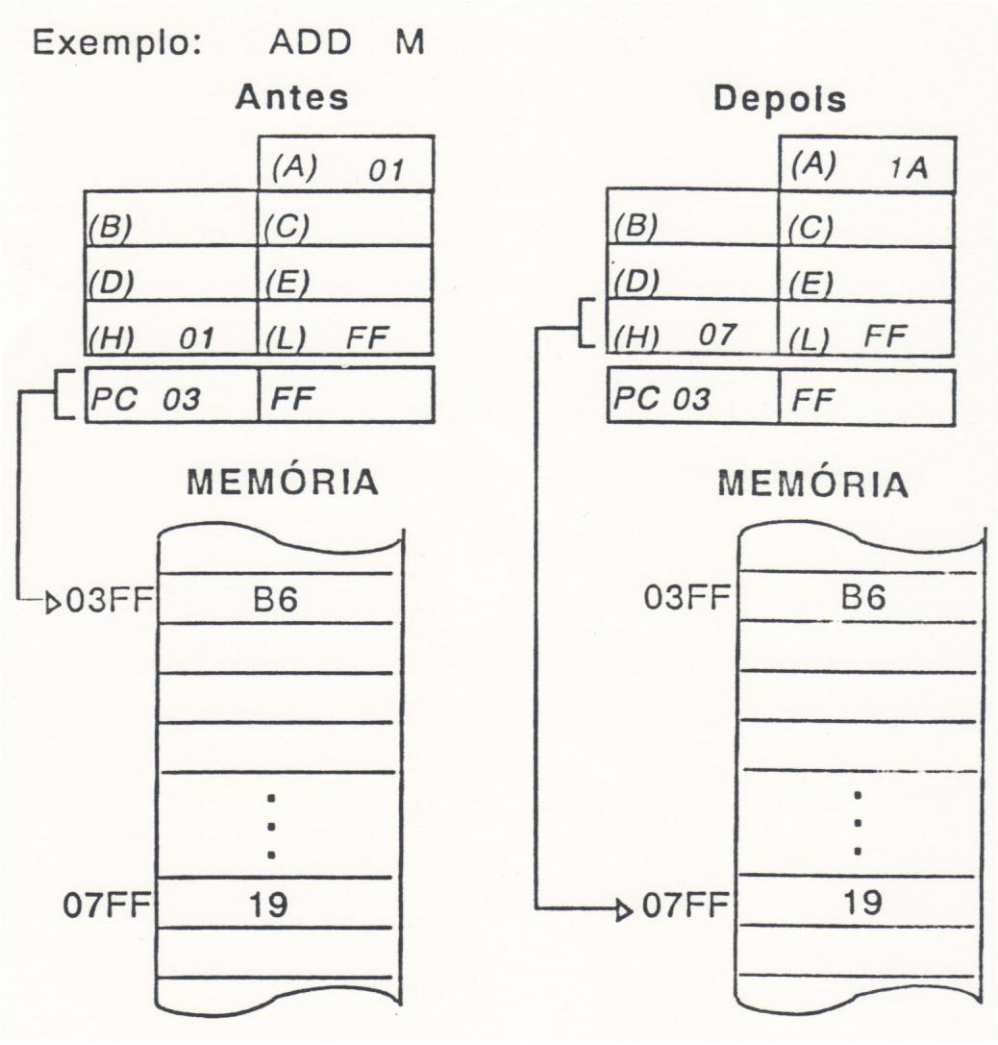

Figura 4: Endereçamento indireto de registrador.

No exemplo da figura acima, os registradores “H” e “L” formam o endereço onde está o conteúdo a ser somado ao valor do conteúdo do registrador “A”. Note que a instrução solicita o endereço que está no par “H” e “L”, e então soma o conteúdo deste endereço (07FF) ao conteúdo do acumulador. O resultado fica no próprio acumulador (19 + 1 =1A em hexadecimal).

1.5) Endereçamento indexado

Algumas vezes, é interessante referenciar um conteúdo da memória cujo deslocamento em relação a uma informação no registrador seja conhecido. O endereçamento indexado fornece um registrador e mais um deslocamento constante. Tendo essas duas informações, o acesso à posição da memória pode ser alcançada.

1.6) Endereçamento de base indexado

Neste tipo de endereçamento, as informações de dois registradores são somadas mais um deslocamento que pode ser opcional. Um dos registradores é a base, e outro é um índice.

2) Linguagem de montagem

Bem, chegamos a uma parte bastante interessante do curso, os estudos feitos até aqui serviram de base para o bom entendimento sobre linguagem de montagem. Você deverá estar familiarizado com números binários, números hexadecimais, estrutura da CPU, funções de cada registrador e modo de endereçamento.

Linguagem de montagem é uma linguagem de programação considerada de baixo nível, ou seja, os comandos atuam bem próximos do *hardware* sem interpretadores intermediários como acontece em linguagens de alto nível.

Todas as instruções são baseadas na arquitetura interna do processador. Como já vimos em aulas anteriores, um processador é composto da unidade lógica e aritmética e da unidade de controle; dentro destas unidades, há os registradores, cada um com uma função específica.

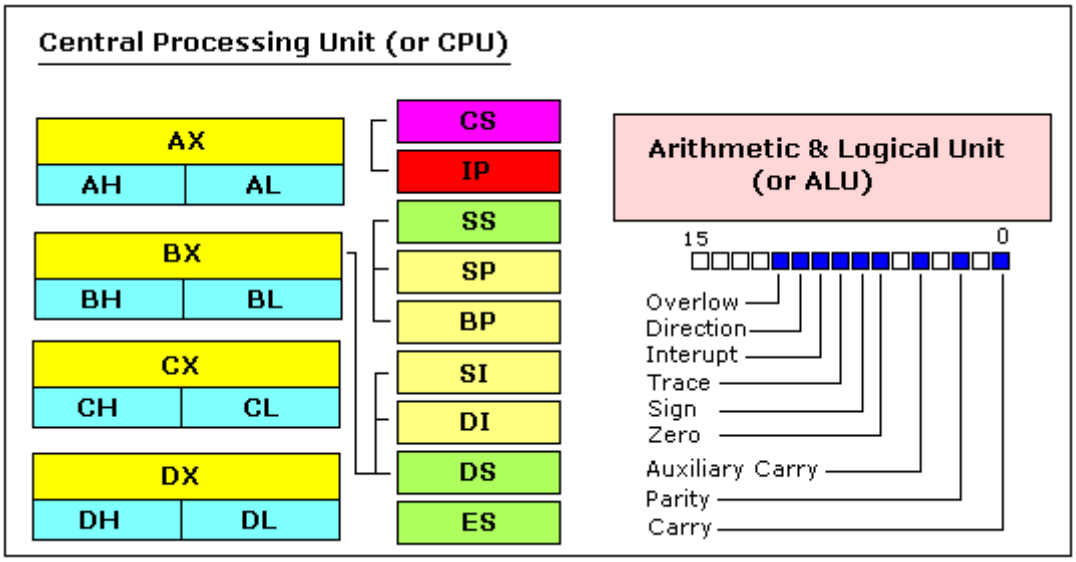

Figura 5: unidade Central de Processamento. Fonte: *http://www.emu8086.com*. Acesso em: 20 de set. 2010.

O processador 8086 tem 8 registradores de propósito geral, abaixo se encontra uma breve descrição destes registradores:

AX – è chamado de acumulador, dividido em duas partes AH / AL;

BX – registrador de endereço base, dividido em BH / BL;

CX – registrador contador, dividido em CH / CL;

DX – registrador de dados, dividido em CH / DL;

SI – registrador de índice de origem;

DI – registrador de índice de destino;

BP – ponteiro base;

SP – ponteiro da pilha.

É o programador que determina o uso de cada registrador de uso geral. A função destes registradores é armazenar temporariamente uma instrução ou dado; neste caso, cada instrução ou dado poderá ser armazenado, no máximo, em 16 *bits*, por exemplo: o número binário (0011001100001111b) que equivale 13071 em decimal.

Os registradores (AX, BX, CX, DX) são divididos em dois registros separados, cada qual com 8 *bits*, por exemplo: se o valor inserido no acumulador AX for 0011001100001111b, então, AH será igual a 00110011b e AL ficará com 00001111b. A letra “H” (*Hight*) associada aos registradores refere-se aos *bits* mais significativos, ou seja, os 8 *bits* mais a esquerda, e a letra “L”(Low) indica os *bits* menos significativos, ou seja, os 8 *bits* mais a

direita. Portanto, dos 16, os 8 *bits* mais a direita são considerados os *bits* mais significativos, e os 8 *bits* mais a esquerda são considerados os *bits* menos significativos.

Por estarem dentro de um mesmo *chip*, os registradores são mais rápidos que a memória principal; pois para acessar um dado na memória RAM, há um barramento externo que, por motivos técnicos, diminui a velocidade de transmissão destes.

Registradores de segmento

CS – Este registrador aponta para o segmento que contém o programa atual;
DS – Usado para apontar para o segmento de dados;
ES – Este registrador de segmento extra pode ser definido pelo programador;
SS – Aponta para o segmento que contém a pilha.

Embora os registradores de segmentos possam ser usados para armazenar dados, não é aconselhável fazer isto; eles podem ser usados para apontar blocos de memória que são acessíveis.

Os registradores de segmentos trabalham junto com o registrador de uso geral para acessar um determinado conteúdo na memória. Por exemplo, o acesso a memória física 12345h (hexadecimal), deve-se definir o registrador DS= 1230h e SI=0045h. Ao usar dois registradores, podemos endereçar mais memória do que se estivéssemos usando apenas um registrador. A CPU de posse destas informações faz o cálculo do endereço físico, multiplicando-se o valor contido no registrador de segmento por 10h e adiciona-se o valor do registrador SI (1230h * 10h + 45h=12345h).

Por padronização, o registrador BX, SI e DI trabalham em conjunto com o registrador de segmento de dados DS. Os registradores BP e SP trabalham com o registrador de segmento SS.

2.1) Registradores de propósito especiais

O IP é considerado o registrador do ponteiro de instrução, ele trabalha em conjunto com o registrador de segmento CS que tem a função de apontar para a instrução que está em execução. O registrador FLAGS, conforme já vimos em aulas anteriores, é alterado automaticamente pela unidade lógica e aritmética após cálculos matemáticos e comparações; isto permite determinar o

resultado e condições para transferir o controle para outras partes do programa.

2.2) Instruções Aritméticas e lógicas

Existem três grupos de instruções, as quais estudaremos logo abaixo:

O primeiro grupo de operandos que estudaremos são os seguintes:

- **REG, memória** -> Registrador para a memória;
- **Memória, REG** -> Memória para registrador;
- **REG, REG** -> registrador para registrador;
- **Memória, imediato** -> Memória para endereçamento imediato;
- **REG, imediata** -> Registrador para endereçamento imediato.

Os registradores (REG), descritos acima poderão ser os seguintes: AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL, SI, BP, SP.

A memória (Memória) pode ser representada por: [BX], {BX +SI 7], variável, etc.

O endereçamento imediato é representado por números inteiros positivos, negativos, hexadecimais e binário. Exemplo: 5, -24, 3Fh, 10001101b, etc.

Após a operação entre operandos, o resultado sempre ficará armazenado no primeiro operando. As instruções de comparações, por definição, não armazenam resultados; estas instruções irão alterar o valor de um dos *bits* do registrador FLAG.

Abaixo são apresentadas algumas instruções e uma breve descrição deste primeiro grupo:

- **ADD** – soma o valor do segundo operando com o valor do primeiro operando;
- **SUB** - subtrai o valor do segundo operando com o valor do primeiro operando;
- **CMP** – compara o valor do segundo operando com o valor do primeiro operando;
- **E** – aplica o operador lógico “and” entre os dois operandos;
- **OR** - aplica o operador lógico “or” entre os dois operandos;
- **XOR** - aplica o operador lógico “xor” entre os dois operandos.

O segundo grupo é composto pelas instruções: MUL, IMUL, DIV, IDIV e os operandos suportados são:

- **REG;**
- **Memória.**

Os registradores (REG) poderão ser os seguintes: AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL, SI, BP, SP.

A memória (Memória) pode ser representada por: [BX], {BX +SI 7], variável, etc.

As instruções MUL e IMUL afetam os *bits* "C" e "O" do registrador FLAG. Abaixo é apresentada uma breve descrição destas instruções:

- **MUL – Multiplicação sem sinal**

  Quando o operando é um *byte*, então:

  AX = AL * operando (O sinal * significa multiplicação)

  Quando o operando for uma palavra (16 *bits*):

  (DX AX) = AX * operando (O sinal * significa multiplicação)

- **IMUL – Multiplicação com sinal**

  Quando o operando é um *byte*, então:

  AX = AL * operando (O sinal * significa multiplicação)

  Quando o operando for uma palavra (16 *bits*)

  (DX AX) = AX * operando (O sinal * significa multiplicação)

  DIV – divisão sem sinal

  Quando o operando é um *byte*:

  AL = AX / operando

  AH = contém o resto da divisão (módulo)

  Quando o operando for uma palavra (16 *bits*)

  AX = (DX AX) / operando

  DX = contém o resto da divisão (Módulo)

- **IDIV – divisão com sinal**

  Quando o operando é um *byte*:

  AL = AX / operando

  AH = contém o resto da divisão (módulo)

  Quando o operando for uma palavra (16 *bits*)

  AX = (DX AX) / operando

  DX = contém o resto da divisão (Módulo)

O terceiro grupo de instruções refere-se às instruções INC, DEC, NOT e NEG e os seguintes operandos são suportados.

- **REG;**
- **Memória.**

Os registradores (REG) poderão ser os seguintes: AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL, SI, BP, SP.

A memória (memória) pode ser representada por: [BX], {BX +SI 7], variável, etc.

As instruções INC e DEC podem alterar os seguintes os *bits* “C”, “S”, “O”, “P” e “A” do registrador FLAG. A instrução NEG altera os *bits* “C”, “Z”, “S”, “O”, “P” e “A” e a instrução NOT não altera bits do registrador FLAG.

2.3) Controle de fluxo do programa

Os processadores disponibilizam algumas instruções para controlar o fluxo do programa, isto é necessário em alguns casos onde se deve tomar uma decisão de acordo com determinado valor assumido em uma comparação. Abaixo você irá encontrar quais as instruções disponíveis para controlar o fluxo.

2.3.1) Saltos incondicionais

A instrução JMP transfere o controle para outro ponto do programa, a sintaxe básica para este comando é:

JMP Etiqueta

Para declarar (criar) uma etiqueta no programa, basta inserir o nome da etiqueta e acrescentar o sinal de dois pontos (:). O nome das etiquetas podem ter qualquer combinação de caracteres, mas não pode começar com números, veja abaixo três nomes de etiquetas possíveis.

etiqueta1:

um:

soma:

Abaixo é apresentado um programa que calcula a soma de dois números, neste caso o número 2 com o número 5, usando etiquetas para desviar o fluxo do programa:

```
org 100h

 mov ax , 5 ; atribui 5 ao registrador ax.
 mov bx , 2 ;  bx é definido como 2.

 jmp     calc ;  vá para a etiqueta 'calc'.

 back: jmp  stop ; vá para a etiqueta 'stop'.

 calc:
 add ax , bx ; soma o valor de bx com valor de ax.
 jmp    back ; volta para a etiqueta 'back'.

 stop:
 ret ; retornar ao sistema operacional.
```

Perceba que quando é executada a linha onde se encontra a instrução JMP, incondicionalmente, o salto é feito para outro local do programa; este local é determinado pelo nome da etiqueta.

2.3.2) Salto condicional.

Há casos em que um programa tem que desviar a sua execução sobre determinadas condições, a esta característica chamamos de saltos condicionais. Este tipo de instrução é dividido em três grupos: o primeiro grupo testa apenas uma FLAG; o segundo grupo compara números com sinal; e o terceiro grupo compara números sem sinal. Abaixo serão apresentadas as instruções pertencentes a cada grupo.

Instrução de salto que testa apenas uma FLAG:

Instrução	Descrição	Condição (FLAGS)
JZ, JE	Salta se for zero	Z =1
JC, JB, JNAE	Desvia se *Carry* =1	C=1
JS	Salta se houver sinal	S=1
JO	Salta se houve *overflow*	O=1
JPE, JP	Salta se a paridade for par	P=1
JNZ, JNE	Desvia se não for zero	Z=0
JNC, JNB,JAE	Salta se *Carry* for igual a zero	C=0
JNS	Salta se não tiver sinal	S=0
JNO	Salta se não houve *overflow*	O=0
JPO, JNP	Salta se paridade for ímpar	P=0

Os três grupos apresentados acima descrevem as características de alguns comandos que podem ser usados em linguagem de baixo nível. Você terá oportunidade, nas próximas aulas, de fazer *download* de um simulador e desenvolver pequenos programas, e avaliar o estado dos registradores e memória.

Referências Bibliográficas

REED D. **"A Balanced Introduction to Computer Science and Programming".**

Tradução do capítulo 14. Creighton University**,** Prentice Hall, 2004. Disponível em *http://professores.faccat.br/assis/davereed/14-DentroDoComputador.html*. Acesso em: 20 de set. 2010.

STALLINGS, W. **Arquitetura e Organização de Computadores:** Projeto Para o Desempenho. 5. ed. São Paulo: Prentice Hall, 2004.

TANENBAUM, A. S. **Organização e Arquitetura de Computadores**. %. Ed. São Paulo: Pearson, Pretice Hall, 2007.

Responsável pelo Conteúdo:
Prof. Ms. Vagner da Silva

.

www.cruzeirodosul.edu.br

Campus Liberdade

Rua Galvão Bueno, 868

01506-000

São Paulo SP Brasil

Tel: (55 11) 3385-3000