

| | |
|--|-----------|
| 1. INTRODUÇÃO | 3 |
| 2. CONCEITOS BÁSICOS..... | 4 |
| 2.1. SISTEMA DECIMAL | 4 |
| 2.2. O SISTEMA BINÁRIO | 4 |
| 2.3. FORMATO BINÁRIO..... | 5 |
| 2.4. ORGANIZAÇÃO DOS DADOS | 5 |
| 2.5. O SISTEMA DE NUMERAÇÃO HEXADECIMAL..... | 6 |
| 3. PROCESSADOR 8086/88..... | 9 |
| 3.1. A FAMÍLIA INTEL..... | 9 |
| 3.2. ESTRUTURA BÁSICA..... | 11 |
| <i>Estrutura de memória</i> | <i>11</i> |
| <i>Segmentação de Memória.....</i> | <i>12</i> |
| <i>Estrutura de entrada/saída</i> | <i>13</i> |
| 3.3. MODELO DO PROGRAMADOR | 14 |
| <i>Registros Gerais.....</i> | <i>14</i> |
| <i>Registros de ponteiros e índices.....</i> | <i>15</i> |
| <i>Registros de segmento.....</i> | <i>15</i> |
| <i>FLAGS.....</i> | <i>16</i> |
| 3.4. MODOS DE ENDEREÇAMENTO DE DADOS | 17 |
| <i>Endereçamento por registro</i> | <i>17</i> |
| <i>Endereçamento absoluto/direto</i> | <i>18</i> |
| <i>Endereçamento imediato.....</i> | <i>18</i> |
| <i>Endereçamento por registro indireto.....</i> | <i>19</i> |
| <i>Endereçamento baseado.....</i> | <i>19</i> |
| <i>Endereçamento indexado.....</i> | <i>20</i> |
| <i>Baseado indexado.....</i> | <i>20</i> |
| <i>Endereçamento relativo.....</i> | <i>20</i> |
| 4. LINGUAGEM ASSEMBLY | 21 |
| 4.1. INTRODUÇÃO:..... | 21 |
| 4.2. LINGUAGEM ASSEMBLY..... | 22 |
| <i>Processo de desenvolvimento de um programa usando a linguagem assembly:</i> | <i>22</i> |
| <i>CARACTERÍSTICAS DA LINGUAGEM ASSEMBLY.....</i> | <i>22</i> |
| <i>FORMATOS DA LINGUAGEM ASSEMBLY.....</i> | <i>23</i> |
| 4.3. OPERAÇÕES EM TEMPO DE ASSEMBLY, TEMPO DE CARGA E TEMPO DE EXECUÇÃO:..... | 24 |
| 5. LINGUAGEM ASSEMBLY DO 8086/88..... | 25 |
| 5.1. SINTAXE DO ASSEMBLY DO 8086/88 | 25 |
| 5.2. DECLARAÇÕES (STATEMENTS):..... | 25 |
| 5.3. FORMATO DE DADOS, VARIÁVEIS E CONSTANTES | 26 |
| 6. CONJUNTO DE INSTRUÇÕES | 29 |
| 6.1. INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS | 29 |
| 6.2. INSTRUÇÕES ARITMÉTICAS | 29 |
| 6.3..... | 30 |
| 6.4. INSTRUÇÕES LÓGICAS..... | 30 |
| 6.5. INSTRUÇÕES DE DESVIO DE FLUXO | 30 |
| 6.6. INSTRUÇÕES DE ENTRADA/SAÍDA | 30 |
| 6.7. INSTRUÇÕES DE CONTROLE..... | 31 |
| 6.8. INSTRUÇÕES DE ROTAÇÃO E DESLOCAMENTO | 31 |
| 7. ALGUMAS INSTRUÇÕES BÁSICAS | 32 |
| 7.1. DESCRIÇÃO DOS FORMATOS DAS INSTRUÇÕES | 32 |
| 7.2. DETALHES DE ALGUMAS TABELAS | 33 |
| 7.3. TABELA DE SÍMBOLOS UTILIZADOS:..... | 34 |
| 7.4. MOV..... | 36 |
| 7.5. ADD | 37 |
| 7.6. SUB | 38 |
| 7.7. INC..... | 39 |
| 7.8. DEC | 40 |

| | | |
|------------|---|-----------|
| 8. | ESTRUTURA DE UM PROGRAMA EM LINGUAGEM ASSEMBLY | 41 |
| 8.1. | MODELO DE MEMÓRIA | 41 |
| 8.2. | SEGMENTO DE PILHA | 41 |
| 8.3. | SEGMENTO DE DADOS | 42 |
| 8.4. | SEGMENTO DE CÓDIGO | 42 |
| 8.5. | CONCLUSÃO | 42 |
| 9. | INSTRUÇÕES DE CONTROLE DE FLUXO | 43 |
| 9.1. | LAÇOS DE REPETIÇÃO | 43 |
| 9.2. | SALTO INCONDICIONAL | 43 |
| 9.3. | SALTOS CONDICIONAIS | 44 |
| | <i>Salto de flags simples</i> | <i>44</i> |
| | <i>Salto sinalizados e não sinalizados</i> | <i>45</i> |
| 9.4. | INSTRUÇÕES VISTAS NESTE CAPÍTULO | 46 |
| 10. | INSTRUÇÕES LÓGICAS, DE DESLOCAMENTO E DE ROTAÇÃO | 50 |
| 10.1. | INSTRUÇÕES LÓGICAS | 50 |
| 10.2. | INSTRUÇÕES DE DESLOCAMENTO | 51 |
| | <i>SHL/SAL – deslocamento para a esquerda.....</i> | <i>52</i> |
| | <i>SAR – deslocamento aritmético para a direita</i> | <i>52</i> |
| | <i>SHR – deslocamento lógico para a direita</i> | <i>52</i> |
| 10.3. | INSTRUÇÕES DE ROTAÇÃO..... | 53 |
| | <i>RCL – rotação para a esquerda com carry flag</i> | <i>53</i> |
| | <i>RCR – rotação para a direita com carry flag.....</i> | <i>53</i> |
| | <i>ROL – rotação para a esquerda sem carry.....</i> | <i>53</i> |
| | <i>ROR – rotação para a direita sem carry</i> | <i>54</i> |
| 11. | SUBROTINAS..... | 55 |
| 11.1. | INSTRUÇÕES DE MANIPULAÇÃO DE PILHA | 55 |
| 11.2. | INSTRUÇÕES DE CHAMADA/RETORNO SUBROTINA | 57 |
| 11.3. | PASSAGEM DE PARÂMETROS | 60 |
| 12. | INTERRUPÇÕES..... | 62 |
| 13. | MACROS | 63 |
| 14. | INTEGRAÇÃO ASSEMBLY+PASCAL..... | 64 |
| 15. | APÊNDICE A – TABELA CARACTERES ASCII..... | 65 |

1. INTRODUÇÃO

O conteúdo desta apostila é uma compilação de diversas fontes tais como livros, apostilas, páginas de internet, etc. e tem o objetivo de servir como material básico para as aulas da disciplina de Linguagem de Máquina – Univali – São José. Trata-se de uma primeira versão e portanto a existência de erros é inevitável.

O capítulo 2 faz uma revisão dos sistemas de numeração e a conversão entre os mesmos. Também introduz algumas nomenclaturas e convenções que serão usadas ao longo do curso. O estudo da linguagem assembly está ligado a um determinado processador, que no nosso caso será o 8086/88 da Intel e que é apresentado no capítulo 3. Os conceitos genéricos da linguagem assembly estão no capítulo 4, enquanto que o capítulo 5 traz as particularidades da linguagem assembly da Borland para o processador Intel 8086. Uma visão do conjunto de instruções do 8086/88 é apresentada no capítulo 6 e os capítulos 7 a 12 vão detalhar as instruções que serão estudadas na disciplina. Os capítulos de 11 a 14 apresentam algumas diretivas do montador assembler e também alguns recursos para acesso ao hardware básico que estão disponíveis nos computadores IBM-PC compatíveis. Por fim, o capítulo 15 mostra a integração da linguagem assembly com outras linguagens, tal como Turbo Pascal e Turbo C.

2. CONCEITOS BÁSICOS

Os computadores atuais representam seus dados (variáveis) no sistema binário ao invés do sistema decimal. Este capítulo tem o objetivo de rever os conceitos referentes aos sistemas de numeração, definições de termos e as formas de conversões entre estes sistemas.

2.1. SISTEMA DECIMAL

Quando rotineiramente representamos um valor, como por exemplo, o número '123', não estamos pensando sobre o valor 123; ao invés, formamos uma imagem mental de quantos itens este valor representa. Na realidade, entretanto, o número 123 representa:

$$1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$$

ou

$$100 + 20 + 3$$

Cada dígito aparecendo à esquerda do ponto decimal representa um valor entre zero e nove vezes uma potência de dez crescente.

2.2. O SISTEMA BINÁRIO

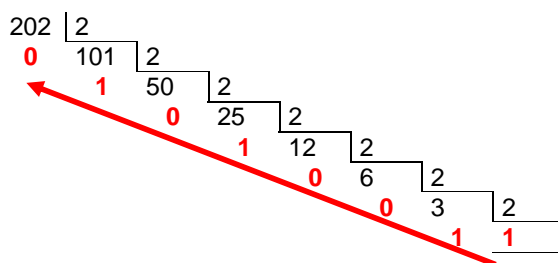
Os computadores normalmente operam usando lógica binária. Os computadores representam seus valores internamente usando dois níveis de voltagem (por exemplo: 0v e 5v). Devido a estas características, os computadores usam o sistema binário de numeração. O sistema binário trabalha de maneira análoga ao sistema decimal, com duas exceções, temos somente dois dígitos: 0 e 1 e se usa potências de dois ao invés de potências de dez. Assim, para converter um número binário para decimal deve-se: para cada '1' na representação do número em binário, adiciona-se 2^n onde 'n' é a posição do dígito binário. Por exemplo, o valor binário 11001010_2 representa:

$$1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 202_{10}$$

A conversão de decimal para binário é feita pelo método das divisões sucessivas, conforme mostrado abaixo:

$$202_{10} = (?)_2$$

procedimento: efetuar divisões por dois até que não seja mais possível dividir. O valor em binário é o obtido juntando-se os restos das divisões no sentido indicado pela seta.



Neste caso, o resultado é: $(11001010)_2$

2.3. **FORMATO BINÁRIO**

Normalmente ao representar um número em binário usa-se escrever múltiplos de 4 dígitos. Por exemplo, para escrever o número cinco usa-se:

0101 ou 0000101

Temos também o hábito dar um espaço a cada 3 dígitos ao escrever um número em decimal. Usaremos esta mesma técnica para representar números binários. Por exemplo o número 0100101001111001 será escrito como 0100 1010 0111 1001.

Para cada dígito de um número escrito em binário, atribui-se um valor numérico que identifica sua posição. Este valor é atribuído da seguinte forma:

- a) o dígito mais a direita é dito ser o de posição zero;
- b) para cada bit a esquerda é dado o número de posição sucessivo;

Um valor binário de 8 bits usa os números de 0 a 7 para identificar cada um dos 8 bits:

$$X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0$$

Um valor binário de 16 bits usa os números de 0 a 15 para identificar os dígitos:

$$X_{15} X_{14} X_{13} X_{12} X_{11} X_{10} X_9 X_8 X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0$$

O bit zero é usualmente designado com bit menos significativo (**low order bit**), enquanto o bit mais a esquerda é denominado bit mais significativo (**high order bit**).

2.4. **ORGANIZAÇÃO DOS DADOS**

Na matemática pura, um valor pode ter um número arbitrário de bits. Computadores, por outro lado, geralmente trabalham com um número específico de bits. Coleções comuns são: bits simples, grupos de quatro bits (chamados *nibbles*), grupos de 8 bits (chamados *bytes*), grupos de 16 bits (chamados *words*). Os tamanhos não são arbitrários e são descritos a seguir.

bit – é a menor unidade de dado em um computador binário. São capazes de assumir somente dois valores diferentes (tipicamente zero e um), mas podem ser usados para representar uma infinidade de coisas, como por exemplo, macho/fêmea, aberto/fechado, acesa/apagada, ligado/desligado, etc. Podem-se ter ainda diferentes bits representando coisas diferentes, por exemplo: um bit poderia ser usado para representar os valores zero e um, enquanto o bit adjacente poderia ser usado para representar o estado de uma porta (aberta/fechada). Daí surge uma questão: olhando para um bit, como descobrir o que ele representa? A resposta é que não é possível. Isto ilustra a idéia por traz das estruturas de dados no computador: dados são o que você define que eles sejam. Se você usa um bit para representar o estado de uma porta (aberta/fechada), então aquele bit (pela sua definição) representa aberta ou fechada. Desde que a maior parte dos itens que se vai modelar requer mais que dois valores diferentes, bits simples não são tipos de dados usados com freqüência.

nibble – é uma coleção de quatro bits. Esta não é uma estrutura de dado particularmente interessante exceto para dois itens: números BCD (binary coded decimal) e números hexadecimal. São precisos quatro bits para representar um dígito simples em BCD ou Hexadecimal. Com um nibble, é possível representar até 16 valores distintos. No caso de números hexadecimal, os valores 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, e F são representados com quatro bits (veja sistema de numeração hexadecimal no item XX a seguir). A codificação BCD usa dez dígitos diferentes (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) e também requerem quatro bits e podem ser representados por um nibble.

byte – um byte consiste de oito bits e é o menor item de dado endereçável pelos processadores 80x86. Tanto a memória quanto os endereços de E/S do 80X86 são acessíveis byte a byte. Isto significa que o menor item de dado que pode ser individualmente acessado e manipulado pelos programas no 80x86 é um valor de 8 bits. O acesso a qualquer coisa menor vai requerer que se leia o byte contendo os dados para então efetuar a manipulação dos bits desejados. Os bits em um byte são normalmente numerados de zero a sete usando a convenção já citada anteriormente e mostrada nas figuras abaixo:



Fig. 2.1 - numeração dos bits em um byte

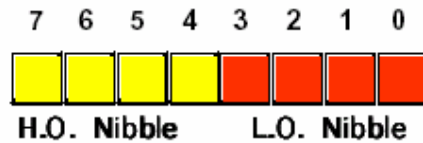


Fig. 2.2 - disposição dos Nibbles em um byte

O bit 0 corresponde ao bit menos significativo e o bit 7 ao bit mais significativo. Notar também que um byte contém exatamente dois nibbles (veja figura 1.2). Os bits 0..3 correspondem ao nibble menos significativo e os bits 4..7 ao nibble mais significativo. Desde que um byte contém oito bits, ele pode representar 2^8 , ou 256, valores diferentes. Geralmente usa-se um byte para representar valores numéricos na faixa de 0..255, números com sinal na faixa de -128..+127, códigos ASCII de caracteres (ver **apêndice A**)

word – é um grupo de 16 bits. Os bits em uma Word são numerados de zero a quinze (ver figura 1.3). Como o byte, o bit 0 é o bit menos significativo e o bit 15 o mais significativo. Notar que uma Word contém exatamente dois bytes. Os bits 0..7 correspondem ao byte menos significativo, enquanto os bits 8..15 correspondem ao byte mais significativo. Com 16 bits, pode-se representar 2^{16} (65536) valores diferentes. Estes valores podem corresponder a valores numéricos na faixa de 0..65535, valores numéricos com sinal na faixa de -32768..+32767 ou qualquer outro tipo de dado com até 65536 valores.

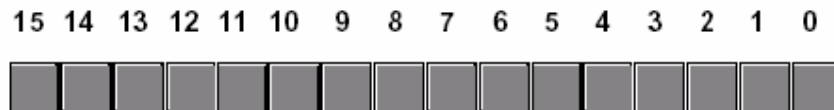


Fig. 2.3 – numeração dos bits em uma word

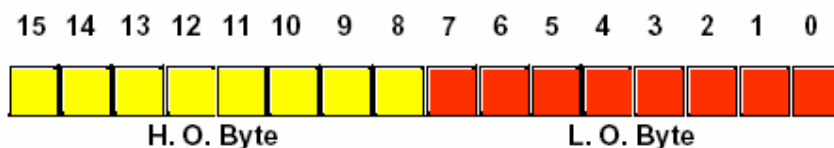


Fig. 2.4 – disposição dos bytes em uma word

2.5. O SISTEMA DE NUMERAÇÃO HEXADECIMAL

Um dos problemas em se representar números binários é a quantidade de dígitos que são necessários. Por exemplo, para o valor 202_{10} seriam necessários 8 dígitos binários. No sistema decimal precisamos de apenas três dígitos e portanto, pode-se representar o número de uma forma mais compacta. Porém os computadores “pensam” em binário e portanto é freqüente a utilização deste sistema para a representação de valores. Embora a conversão entre o sistema decimal e binário, esta conversão não é trivial. O sistema de numeração hexadecimal resolve este problema pois possui duas características interessantes: permitem escrever valores numéricos em uma forma compacta e pode ser facilmente convertido para o sistema binário. Em função destas características o sistema hexadecimal é bastante utilizado para a representação de valores numéricos quando se está programando em assembly. Como a base do sistema de numeração hexadecimal é 16, cada dígito hexadecimal a esquerda do ponto hexadecimal representa algum valor vezes uma potência de 16

sucessiva, analogamente ao que foi visto para o sistema decimal e binário. Por exemplo o número 123416 é igual a:

$$1 \cdot 16^3 + 2 \cdot 16^2 + 3 \cdot 16^1 + 4 \cdot 16^0$$

ou

$$4096 + 512 + 48 + 4 = 4660_{10}.$$

Cada dígito hexadecimal pode representar um de dezesseis valores entre 0 e 15₁₀. Como existem somente dez dígitos, é necessário seis dígitos adicionais para representar os valores entre 10₁₀ e 15₁₀. Para isto são usadas as 6 primeiras letras do alfabeto: A..F. Os exemplos a seguir são valores hexadecimais válidos:

1234₁₆

DEAD₁₆

BEEF₁₆

FEED₁₆

DEAF₁₆

Quando se trata de programação, é preciso utilizar uma representação que mostre em qual base um número está escrito. As seguintes convenções são normalmente adotadas em assembly:

- Todo valor numérico (independente da raiz) começa com um dígito decimal;
- Todo valor hexadecimal termina com a letra 'h' (ou 'H');
- Todo valor binário termina com a letra 'b' (ou 'B');
- Valores decimais terminam com a letra 'd' (ou 'D'), porém esta é opcional;

Exemplos de números hexadecimais são:

1234h

DEADh

BEEFh

FEEDh


DEAFh

Antes de mostrar a conversão hexadecimal <-> binário considere a seguinte tabela:

| Binary | Hexadecimal |
|--------|-------------|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | A |
| 1011 | B |
| 1100 | C |
| 1101 | D |
| 1110 | E |
| 1111 | F |


Esta tabela mostra a representação de dígitos hexadecimais e seus correspondentes em binário. Para converter um número hexadecimal para binário, simplesmente substitua os quatro bits correspondentes a cada dígito hexadecimal. Por exemplo:

| | | | | | |
|------|------|------|------|------|---|
| 0 | A | B | C | D | h |
| 0000 | 1010 | 1011 | 1100 | 1101 | b |



Para converter um número binário para hexadecimal também é bastante simples: o primeiro passo é preencher o número binário com zeros à esquerda de forma que o total de dígitos do número seja múltiplo de 4. Por exemplo, o número binário 1011001010b (10 dígitos) deve ser reescrito como 001011001010b (12 dígitos). O próximo passo é separar o valor binário em grupos de 4 bits. Finalmente converta estes grupos de 4 bits para o dígito correspondente em hexadecimal. Exemplo:

| | | | |
|------|------|------|-------------|
| 0010 | 1100 | 1010 | binário |
| 2 | C | A | hexadecimal |



Comparadas com as conversões para decimal, as conversões entre hexadecimal <-> binário são mais simples e diretas. Com a prática a tabela mostrada acaba sendo memorizada, o que acelera ainda mais a conversão.

3. PROCESSADOR 8086/88

3.1. A FAMÍLIA INTEL

Em 1968, a Intel Corporation foi criada para fabricar pastilhas de memória. Logo depois, ela foi contactada por um fabricante de calculadoras que queria uma CPU em uma única pastilha para sua calculadora, e por um fabricante de terminais que queria um controlador em uma única pastilha para o seu terminal. A Intel produziu ambas as pastilhas, o 4004, uma CPU de 4 bits, e o 8008, uma CPU de 8 bits. Estas foram as primeiras CPUs numa única pastilha do mundo.

A Intel não esperava outros interessados além dos clientes originais, de maneira que estabeleceu uma linha de produção de baixo volume. Estavam errados. Houve um interesse tremendo, por isto começaram a projetar uma pastilha de CPU de uso geral, que resolvesse o problema do limite de 16K de memória do 8008 (imposto pelo número de pinos da pastilha). Este projeto resultou no 8080, uma pequena CPU de uso geral. Semelhante ao ocorrido com o PDP-8, este produto tomou a indústria de assalto, e instantaneamente tornou-se um item de venda em massa. Porém, ao invés de vender milhares, como a DEC, a Intel vendeu milhões.

Dois anos mais tarde, em 1976, a Intel lançou o 8085, um 8080 encapsulado com alguns detalhes extras de entrada/saída. Depois surgiu o 8086, uma verdadeira CPU de 16 bits numa única pastilha. O 8086 foi seguido pelo 8088, baseado na mesma arquitetura do 8086 e executava os mesmos programas, mas possuía um barramento de 8 bits, ao invés de um barramento de 16 bits, o que o tornava mais lento, porém mais barato que o 8086. Quando a IBM escolheu o 8088 para CPU do IBM PC original, esta pastilha tornou-se rapidamente o padrão da indústria de computadores pessoais.

Nos anos seguintes, a Intel lançou o 80186 e o 80188, essencialmente novas versões dos 8086 e 8088, respectivamente, mas contendo também uma grande quantidade de circuitaria de entrada/saída. Nunca foram amplamente utilizados.

Nem o 8088 nem o 8086 podiam endereçar mais de 1 megabyte de memória. No início dos anos 80, isto se tornou um problema cada vez mais sério, por isso a Intel projetou o 80286, uma versão superior, compatível com o 8086. O conjunto básico de instruções era basicamente o mesmo do 8086 e 8088, mas a organização de memória era muito diferente, e um tanto desajeitada, devido à necessidade de compatibilidade com as pastilhas anteriores. O 80286 foi utilizado no IBM PC/AT e nos modelos intermediários do PS/2. Do mesmo jeito que o 8088, obteve um grande sucesso.

O próximo passo lógico foi uma CPU verdadeiramente de 32 bits numa pastilha, o 80386. Como o 80286, este microprocessador era mais ou menos compatível com todos os anteriores, até o 8008, o que era de grande ajuda para aqueles que executarem software antigo era importante, mas incômodo para aqueles que preferiam uma arquitetura moderna, limpa e simples, sem os erros e a tecnologia do passado. Como o 80286, esta pastilha foi amplamente utilizada. O 80386SX é uma versão especial do 80386, projetado para ser compatível com o soquete do 80286 de modo a prover um melhoramento parcial às máquinas já existentes.

O 80486 é uma versão superior, compatível com o 80386. Todos os programas para o 80386 serão executados no 80486 sem modificações. A diferença básica entre o 80486 e o 80386 é a presença de um co-processador de ponto-flutuante, controlador de acesso direto à memória e 8K bytes de memória cache na pastilha. Além disso, o 80486 é tipicamente duas a quatro vezes mais rápido que o 80386, e também mais adequado a sistemas multiprocessadores.

A evolução da linha de CPU da Intel reflete a evolução da indústria de computadores como um todo. Em cerca de uma década e meia, houve uma evolução de uma CPU de 4 bits para uma CPU de 32 bits. Um quadro resumo é mostrado abaixo:

| Nome | Ano | Largura dos registradores | Largura do barramento de dados | Espaço de endereçamento | Comentários |
|---------|------|---------------------------|--------------------------------|-------------------------|---|
| 4004 | 1971 | 4 | 4 | 1K | primeiro processador em uma pastilha |
| 8008 | 1972 | 8 | 8 | 16K | primeiro microprocessador de 8 bits |
| 8080 | 1974 | 8 | 8 | 64K | primeira CPU de uso geral em uma pastilha |
| 8085 | 1974 | 8 | 8 | 64K | 8080 reencapsulado |
| 8086 | 1978 | 16 | 16 | 1M | primeira CPU de 16 bits em uma pastilha |
| 8088 | 1980 | 16 | 8 | 1M | processador utilizado no IBM PC |
| 80186 | 1982 | 16 | 16 | 1M | 8086 + suporte de entrada/saída em uma pastilha |
| 80188 | 1982 | 16 | 16 | 1M | 8088 + suporte de entrada/saída em uma pastilha |
| 80286 | 1982 | 16 | 16 | 16M | espaço de endereçamento aumentado para 16 megabytes |
| 80386 | 1985 | 32 | 32 | 70T | verdadeira CPU de 32 bits em uma pastilha |
| 80386SX | 1988 | 32 | 16 | 70T | 80386 com barramento de 80286 |
| 80486 | 1989 | 32 | 32 | 70T | versão mais rápida do 80386 |

Ao longo do curso estudaremos a CPU 8086 (8088), cujas características serão apresentadas nos itens a seguir.

3.2. ESTRUTURA BÁSICA

A estrutura básica de um computador baseado na CPU 8086 é mostrada na figura 1.1. A arquitetura do processador 8086 suporta entrada/saída isolada, sendo que pelo menos logicamente poderia existir barramentos separados de memória e entrada/saída. Entretanto, nos circuitos integrados do 8086 e 8088, o barramento de entrada/saída e o barramento de memória compartilham os mesmos pinos. O processador gera sinais no **barramento de controle** para indicar se uma transação usando estes pinos compartilhados são para a interface de entrada/saída ou para a memória.

Como indicado na figura 1.1, a parte de dados do barramento de memória do 8086 tem largura de 16 bits. Portanto, uma informação de 16 bits pode ser transferida em um único ciclo de leitura. No 8088, o barramento de dados tem somente 8 bits de largura, e portanto toda transferência de dados são de um byte de cada vez. Os circuitos do 8086 e do 8088 são idênticos em quase todos outros aspectos.

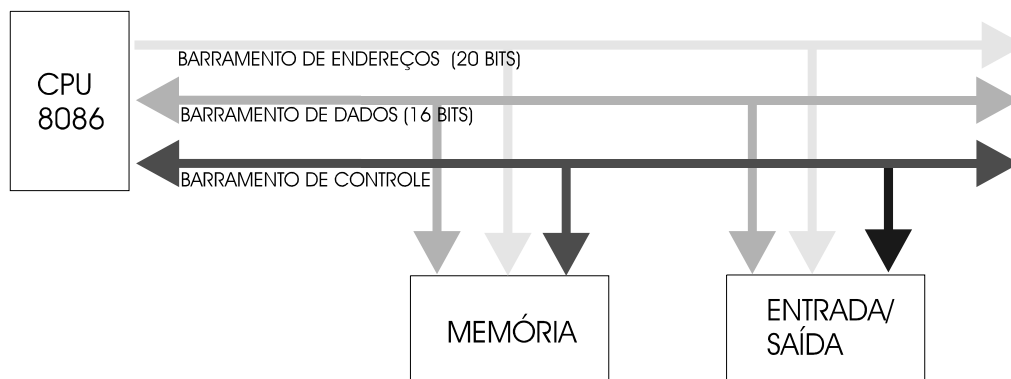


FIG. 3.1 - Diagrama de blocos de um computador baseado no 8086

Estrutura de memória

A memória em um sistema 8086 é uma seqüência de no máximo 2^{20} (aproximadamente 1.000.000) unidades de 8 bits (bytes). A cada byte é associado um endereço único (número inteiro positivo) variando de 0 até $2^{20} - 1$ (0000 0000 0000 0000 0000 até 1111 1111 1111 1111 1111 em binário, 00000 até FFFFF em hexadecimal). A figura 3.2 ilustra esta estrutura de memória.

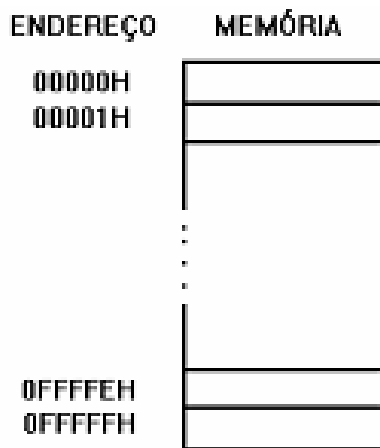


Fig. 3.2 – Estrutura de memória

Quaisquer 2 bytes consecutivos na memória são definidos como uma **word**. Cada byte em uma word tem um endereço distinto, e o menor deles é usado para endereçar uma word. Exemplos de word são mostrados na figura 3.3.

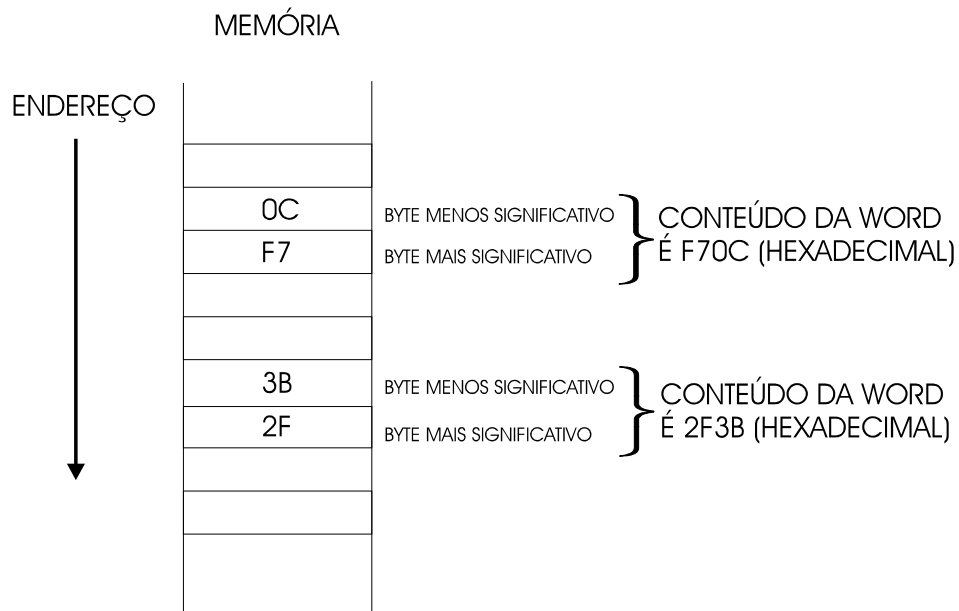


FIG. 3.3 - Formas de armazenamento de words na memória

Uma word contém 16 bits. O byte com o endereço de memória mais alto contém os 8 bits mais significativos da word, e o byte com o endereço de memória mais baixo contém os 8 bits menos significativos.

O 8086 possui algumas instruções que acessam (leitura ou escrita) bytes e instruções que acessam words. O total de informação transferida da ou para a memória em uma operação é sempre 16 bits. No caso de instruções que operam com bytes, somente oito destes bits são usados, sendo os oito restantes desprezados. Os 16 bits são sempre o conteúdo de dois bytes consecutivos na memória começando em um endereço par. O que significa que uma instrução de manipulação de word que lê ou escreve uma word em um endereço par pode executar a operação de uma única vez. Entretanto, se a word iniciar em um endereço ímpar, mais trabalho deve ser realizado: dois acessos a memória deve ser feito para dois endereços de word pares, ignorar a metade que não interessa de cada word, e efetuar uma manipulação entre os bytes que interessam. O programa no 8086 desconhece estes detalhes de acesso a memória; uma instrução simplesmente solicita o acesso (leitura ou escrita) de um byte ou word, e o processador faz tudo que é necessário para executar tal acesso. Vale salientar que as características de acesso citadas acima não tem sentido para o 8088, já que seus acessos à memória sempre lê ou escreve 1 byte de cada vez.

Segmentação de Memória

Considerando que o 8086 pode acessar 2^{20} bytes de memória, é de se esperar que no seu interior os endereços sejam representados como quantidades de 20 bits. Porém, o 8086 foi projetado para executar aritmética de 16 bits, e portanto os objetos de endereços que manipula podem ter somente 16 bits de comprimento. Um mecanismo adicional faz-se necessário para montar os endereços.

Podemos imaginar o segmento de 1 megabyte de memória como um número arbitrário de **segmentos**, cada um contendo no máximo 2^{16} (aproximadamente 65.000) bytes. Cada segmento começa em um byte cujo endereço é divisível por 16 (ou seja, os quatro bits menos significativos do endereço são "0"). A qualquer momento, o programa pode acessar imediatamente o conteúdo de quatro segmentos distintos. Estes quatro segmentos são: *segmento de código*, *segmento de dado*, *segmento de pilha (stack)*, e o *segmento extra*. (O segmento extra é uma área de propósito geral frequentemente tratada como um segmento adicional de dados). Cada segmento corrente é identificado pela colocação dos 16 bits mais significativos do endereço do seu primeiro byte em um dos quatro registros dedicados. Estes registros são chamados *registros de segmento*. Os segmentos não necessariamente são únicos e podem se sobrepor. Exemplos de segmentos são mostrados na figura 1.4.

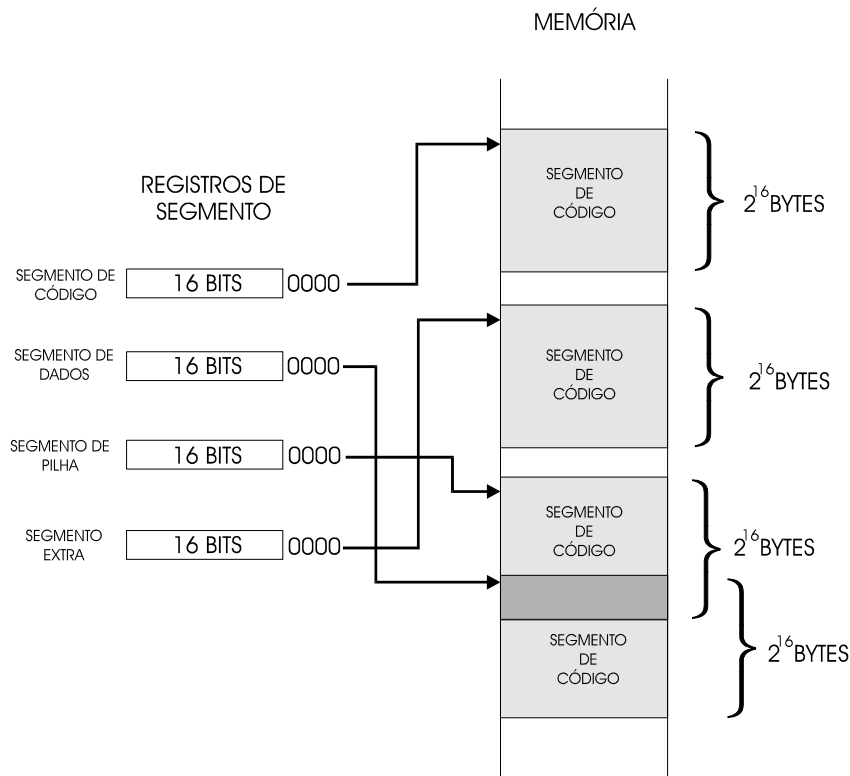
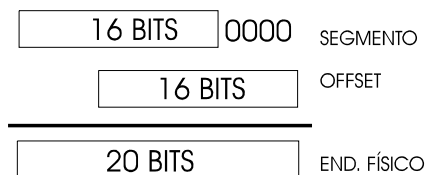


FIG. 3.4 - Esquema de segmentação de memória utilizado pelo 8086

Como um exemplo, assuma que o registro de código de 16 bits contém o valor hexadecimal C018. Isto significa que o segmento de código inicia no endereço C0180 e se estende por um total de 2^{16} (10000 hexadecimal) bytes. O último byte no segmento de código é portanto o byte no endereço D017F.

Os bytes e words no interior de um segmento são referenciados pelo uso de um *endereço de offset* de 16 bits. O processador monta o endereço de 20-bits pela adição dos 16 bits do endereço de offset ao conteúdo de 16 bits do registro de segmento com quatro zeros adicionados à direita, como mostrado a seguir:



Portanto, no exemplo anterior, o byte no endereço CFFFF encontra-se no interior do segmento de código. Especificamente, ele possui um endereço de offset de FE7F (CFFFF-C0180) no interior do segmento.

Estrutura de entrada/saída

Um sistema 8086 se comunica com o resto do mundo através de portas ("ports"). É através destas portas que o 8086 pode receber informações sobre eventos externos e pode enviar sinais que controlam outros eventos.

O 8086 pode acessar até 2^{16} (aproximadamente 65.000) portas de 8 bits de forma análoga aos bytes de memória. Cada porta de 8 bits é associada a um endereço único que varia de 0 até $2^{16} - 1$. Duas portas consecutivas qualquer podem ser tratadas como uma porta de 16 bits, análogo a uma word de memória. De fato, portas são endereçadas da mesma maneira que bytes ou words de memória, exceto que não existem segmentos para acesso a portas. Em outras palavras, todas as portas são consideradas pertencentes a um segmento.

O 8086 tem instruções para ler informações das portas de entrada e para escrever informações nas portas de saída.

3.3. MODELO DO PROGRAMADOR

O processador 8086 contém um total de 13 registros e nove flags de 1 bit. Para propósitos descritivos, os registros são subdivididos em quatro conjuntos. Três destes conjuntos contém quatro registros cada. O 13º registro, chamado apontador de instruções (= contador de programas), não é diretamente acessível pelo programador. Os registros do 8086 são mostrados na figura 3.5.

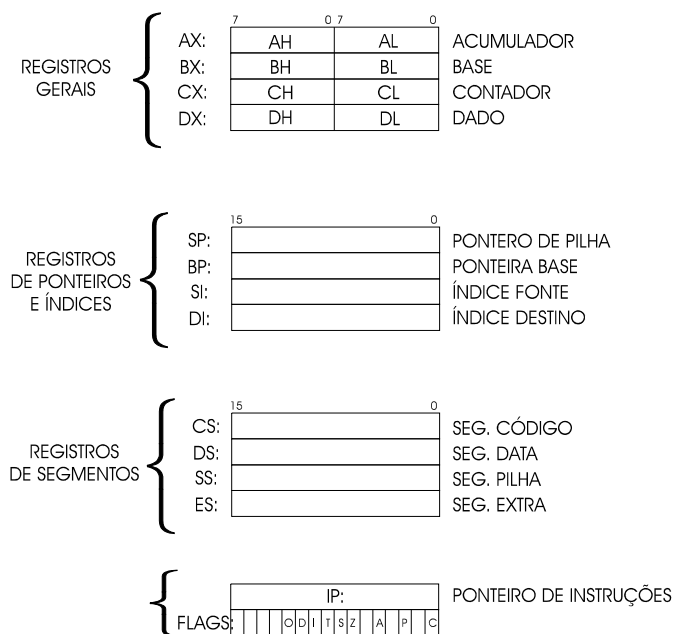


FIG. 3.5 - Registros internos do 8086

Os três conjuntos de registros acessíveis são os registros gerais, os registros de ponteiros e índices, e os registros de segmento. Os registros gerais são usados primeiramente para armazenar operandos para operações aritméticas e lógicas. Os registros de ponteiros e índices são usados para armazenar o offset de endereços no interior dos segmentos. Os registros de segmentos são usados especificamente para iniciar o segmento de endereços.

Registros Gerais

Em um processador sem registros gerais, cada instrução deverá buscar seu operando na memória e retornar o resultado para a memória. Mas o acesso a memória toma tempo. Este tempo poderia ser reduzido através da armazenagem temporária dos operandos e resultados em um lugar de rápido acesso. O conjunto de registros gerais do 8086 correspondem a estes locais.

Os registros gerais do 8086 são os registros de 16-bit AX, BX, CX e DX. A metade superior e inferior de cada registro podem ser usadas separadamente como dois registros de 8-bit ou juntos como um registro de 16-bit. Portanto, cada metade de um registro geral possui seu próprio nome. As metades inferiores (menos significativas) são chamadas AL, BL, CL e DL, e as metades superiores (mais significativas) são chamadas de AH, BH, CH, DH. A natureza dual destes registros permitem manusear tanto byte ou word com igual facilidade.

Em muitos pontos, o conteúdo dos registros gerais podem participar em uma operação aritmética e lógica no 8086. Por exemplo, a instrução ADD pode adicionar o conteúdo de qualquer registro geral de 8 ou 16 bits para qualquer outro registro geral do mesmo tamanho e armazenar o resultado em um outro registro. Entretanto, existem algumas poucas instruções que dedicam certos registros para uso específico. Por exemplo, as instruções de manipulação de string necessitam que o registro CX contenha o contador do número de elementos na string. Nenhum dos outros registros AX, BX ou DX podem ser usados para este propósito. Este uso especializado do registro CX sugere o seu nome descritivo CONTADOR. O uso especializado para os

registros AX, BX e DX, conforme será visto posteriormente sugerem, respectivamente, os nomes descritivos ACUMULADOR, BASE e DADO.

Registros de ponteiros e índices

Uma instrução que acessa uma posição na memória poderia especificar o endereço daquela posição diretamente. Este endereço ocupa espaço na instrução, portanto aumentando o tamanho do código. Se endereços das posições freqüentemente usadas fossem armazenados em registros especiais, instruções que acessam estas posições não mais necessitariam conter o endereço, mas poderiam ao invés especificar o registro que contenha o endereço. Tais registros são algumas vezes chamados ponteiros ou índices.

O registro de ponteiro e índice do 8086 consiste dos ponteiros de 16 bits SP, BP, SI e DI. Estes registros geralmente contém endereços de offset para endereçar algo no interior de um segmento. Por exemplo, uma instrução ADD poderia especificar que um dos seus operandos está localizado no segmento de dados da memória em um offset contido em um ponteiro ou índice particular (ex: SI).

Registros de ponteiros e índices servem para outra função além de reduzir o tamanho das instruções; eles permitem as instruções acessar posições cujos endereços de offset são resultados de algum cálculo anterior efetuado durante a execução do programa. É freqüentemente necessário executar tais cálculos de modo a estabelecer o offset de endereço das variáveis, especialmente nos programas em linguagem de alto nível. Estes cálculos poderiam ser executados em um registro geral e o seu resultado movido para um registro de ponteiro ou índice para ser usado como um offset. A eliminação destas movimentações poderiam resultar em programas menores. Por esta razão, os valores contidos em um registro de ponteiro ou índice são permitidos participarem de operações lógicas e aritméticas entre registros gerais de 16 bits. Portanto, a instrução ADD mencionada anteriormente poderia especificar que seu outro operando é o conteúdo do registrador DI.

Existem algumas diferenças entre os registros que resultam na divisão deste conjunto de registros nos registros de ponteiros SP e BP, e os registros de índices SI e DI. Os registros de ponteiros são voltados para fornecer um acesso conveniente aos dados no segmento de pilha na forma de um segmento de dados. Este uso do segmento de pilha como área de dados tem certas vantagens para a implementação de linguagens de alto nível. Portanto, a não ser que um segmento seja especificamente designado, o conteúdo do offset contido nos registros de ponteiros são referentes ao segmento de pilha, enquanto offset contidos nos registros de índices são geralmente referentes ao segmento de dados. Por exemplo, se um instrução ADD especifica que SI contém o offset de um dos operandos, aquele operando irá assumir o segmento de dados a menos que a instrução ADD explicitamente determine algum outro segmento.

Existem algumas instruções que fazem distinção entre os dois registros ponteiros SP e BP. As instruções PUSH e POP obtêm o offset para a localização do topo da pilha no registro SP, assim sugestionando o nome de ponteiro de pilha para este registro. O registro BP pode não ser usado para este propósito. Isto deixa o registro BP livre para conter o offset da base da área de dados do segmento de pilha, sugerindo assim o nome descritivo de ponteiro base.

Além disso, as instruções associadas a string fazem uma distinção entre os dois registros de índices SI e DI. Estas instruções de string necessitam de um operando origem, obtendo o offset para este operando de SI; da mesma forma, DI contém o offset do operando destino. Isto sugere os respectivos nomes descritivos ÍNDICE ORIGEM e ÍNDICE DESTINO. Devido à estas instruções de manipulação de strings, os papéis dos registros SI e DI não podem ser trocados. Como um exemplo, a instrução de movimentação de string irá mover a string localizada no segmento de dados iniciando no offset contido no movimento SI e relocando ele para o segmento extra com um offset contido em DI; os registros SI e DI não são explicitamente mencionados pela instrução de movimentação de string. (Incidentalmente, a string destino está no segmento extra ao invés do segmento de dados, sendo que cada string deverá ter um segmento próprio e poderá ter no máximo 216 bytes de comprimento)

Registros de segmento

Recordando que o 8086 tem um megabyte de memória, mas os endereços contidos nas instruções e nos registros de ponteiros e índices são de somente 16 bits de tamanho. Estes endereços não podem ser endereços em um megabyte de memória, mas devem ser offset de endereços em algum byte particular de um segmento de 65000 byte. Mas qual?

O registro de segmento do 8086 são os registros de 16 bits CS, DS, SS e ES. Estes registros são usados para identificar os quatro segmentos que são correntemente endereçáveis. Cada registro identifica um segmento em particular, e eles não podem ser trocados: CS identifica o segmento de código, DS o segmento de dados, SS o segmento de pilha e ES o segmento extra.

Uma instrução especifica um offset em um segmento, e os registros de segmentos especificam os quatro segmentos que podem ser usados. Qual deles deve ser selecionado? A resposta depende de como o offset

será usado. Um offset poderia especificar a próxima instrução a ser executada, ou poderia especificar um operando para uma instrução.

Todas instruções são buscadas com base no segmento de código. Portanto necessitamos um registro que contenha o offset no segmento de código corrente da próxima instrução a ser executada. Este é o propósito do IP, o ponteiro de instruções. Por exemplo, se o conteúdo de CS fosse igual ao hexadecimal 1FF7 e o conteúdo do IP igual ao hexadecimal 003A, então a próxima instrução a ser buscada na memória viria do endereço de memória 1FFAA porque:

| | |
|--------|--|
| 1FF70 | endereço inicial do segmento de código |
| + 003A | offset contido em IP |
| 1FFAA | endereço de memória da próxima instrução |

(Recordar na figura 2.3 que o dígito hexadecimal "0" é adicionado ao valor do registro de segmento na montagem do endereço de memória)

O segmento para busca de operandos pode geralmente ser especificado precedendo a instrução com um prefixo especial de 1-byte. Este prefixo especifica de qual dos quatro segmentos o operando irá ser buscado. Na ausência de tal prefixo (caso mais usual), o operando é buscado no segmento de dados a não ser que:

- (1) o endereço de offset foi calculado a partir do conteúdo do registro de ponteiro, neste caso o segmento de pilha é usado;
- (2) o operando é o operando de destino de uma instrução de manipulação de string, neste caso o segmento extra é utilizado.

Como um exemplo, considere uma instrução ADD que tem um de seus operandos no segmento de dados e com um offset contido em SI. A instrução deverá especificar SI no seu campo de operando, mas não poderá fazer menção de DS. Quando executando a instrução, o processador saberá que deve usar o conteúdo de DS juntamente com o conteúdo de SI de forma a localizar o operando. A seguir, considere uma instrução ADD na qual o operando está no segmento de código (o que poderia ser o caso de constantes em ROM) e com o offset contido em SI. Esta instrução ADD deverá, como antes, especificar SI no seu campo de operando; mas, em adição, a instrução deverá ser precedida por byte de prefixo especificando CS.

FLAGS

O 8086 contém nove flags que são usados para armazenar informações de status do processador (flags de status) ou para controlar operações do processador (flags de controle). Os flags de status são geralmente atualizados após a execução de uma instrução lógica ou aritmética para refletir certas propriedades dos resultados de tais operações. Estes flags são:

- . CF - carry flag - se a instrução gerou um vai um no bit mais significativo;
- . AF - auxiliary carry flag - indica se a instrução gerou um vai-um dos quatro bits menos significativos;
- . OF - overflow flag - indicando se a execução da instrução gerou um overflow;
- . ZF - zero flag - indicando se a instrução gerou um resultado zero;
- . SF - sign flag - indicando se a instrução gerou um resultado negativo;
- . PF - parity flag - indicando se a instrução gerou um resultado tendo uma quantidade ímpar de bits 1s;

Os flags de controle são:

- . DF - direction flag - controla a direção das instruções de manipulação de string;
- . IF - interrupt flag - habilita ou desabilita interrupções externas;
- . TF - trap flag - coloca o processador no modo single-step para depuração de programas;

3.4. MODOS DE ENDEREÇAMENTO DE DADOS

Muitas instruções do 8086 especificam seus operandos nos registros gerais ou na memória. Dependendo do tipo de instrução, um operando pode ser um byte ou uma word. Instruções que manipulam words podem acessar qualquer um dos registros AX a DI. Instruções que manipulam bytes tratam os registros como um conjunto de registros de 8 bits cada AH-DI como mostrado na fig 2.X. Em ambos os casos, um campo de 3 bits na instrução é usado para codificar o número do registro.

Instruções básicas do 8086 e modos de endereçamento especificam segmentos de offset de 16 bits. Após um segmento de 16 bits ter sido definido, ele é combinado com um registro de segmento para produzir um endereço físico de 20 bits, conforme veremos posteriormente.

Chamaremos este endereço de offset de 16 bits calculado por um determinado modo de endereçamento de **endereço efetivo** em compatibilidade com o vocabulário usado na descrição de modos de endereçamento.

O 8086 oferece uma grande variedade de endereçamento, sendo as principais descritas a seguir.

Endereçamento por registro

Neste modo de endereçamento, o operando a ser manipulado está armazenado num dos registros internos do processador. A figura 2.1 mostra a codificação de uma instrução com modo de endereçamento por registro típica. Nesta e em outras figuras que virão, a palavra "OPCODE" denota o campo da instrução que contém o código da instrução e outras informações tal como modo de endereçamento e seleção de word/byte.

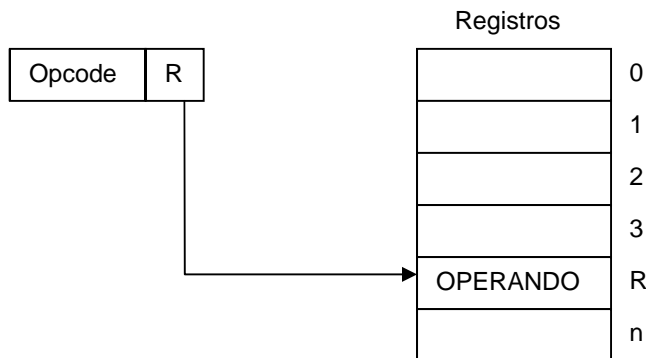


FIG. 3.6 – Endereçamento por Registro

Um exemplo de instrução que utiliza este modo de endereçamento é a instrução INC, cujo uso mais comum é o de incrementar o conteúdo de um registro de ponteiro ou índice ou um registro geral de 16 bits. Para estes operandos a instrução toma a seguinte forma:

Esta instrução contém um campo de 3 bits denominado reg que especifica um dos 8 registros de 16 bits (geral, ponteiro e índice). Os cinco bits restantes da instrução identifica a operação e são coletivamente referenciados como opcode. No caso da instrução INC, o opcode é 01000.

Ex.: INC BP -> 01000 101

Endereçamento absoluto/direto

Neste modo, que representa a maneira mais simples de especificar um endereço de memória, o endereço do operando é incorporado à instrução, o que corresponde a um endereço efetivo de 16 bits.

Existe também o modo de endereçamento absoluto longo, no qual a instrução contém um endereço base de 16 bits e também o offset de 16 bits, permitindo ao programa acessar diretamente qualquer endereço lógico. Entretanto, este modo está disponível somente para as instruções jump e call, e não com instruções de manipulação de dados em geral.

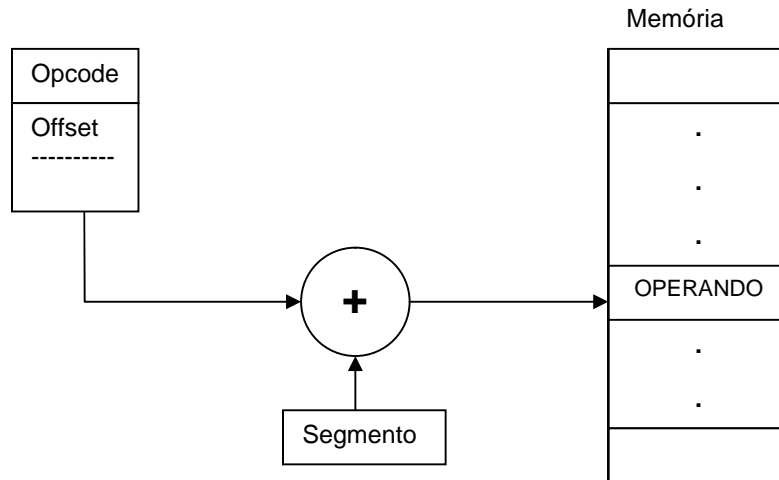


FIG. 3.7 – Endereçamento Absoluto/Direto

Endereçamento imediato

O modo de endereçamento imediato é aquele no qual o valor a ser manipulado é incorporado diretamente à instrução. O uso típico deste modo de endereçamento são:

- . inicialização de variáveis através de constantes;
- . operação com variáveis e constantes
- . carregamento de endereços constantes em registros que serão utilizados posteriormente por outros modos de endereçamento.

Nesta forma de endereçamento, um dos operandos está presente no byte(s) imediatamente seguinte ao código objeto da instrução. Ex. :

ADD AX,3064H

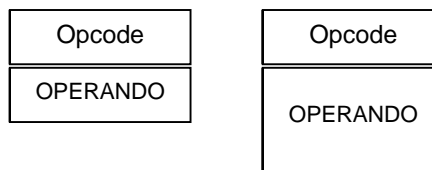


FIG. 3.8 – Endereçamento Imediato

Endereçamento por registro indireto

No modo de endereçamento por registro indireto, um registro ou um par de registros contém o endereço efetivo do operando. O 8086 contém 4 registros de base: BX, BP, SI, DI. Entretanto, o uso deste modo de endereçamento está disponível somente com 3 destes registros base: BX, SI e DI.

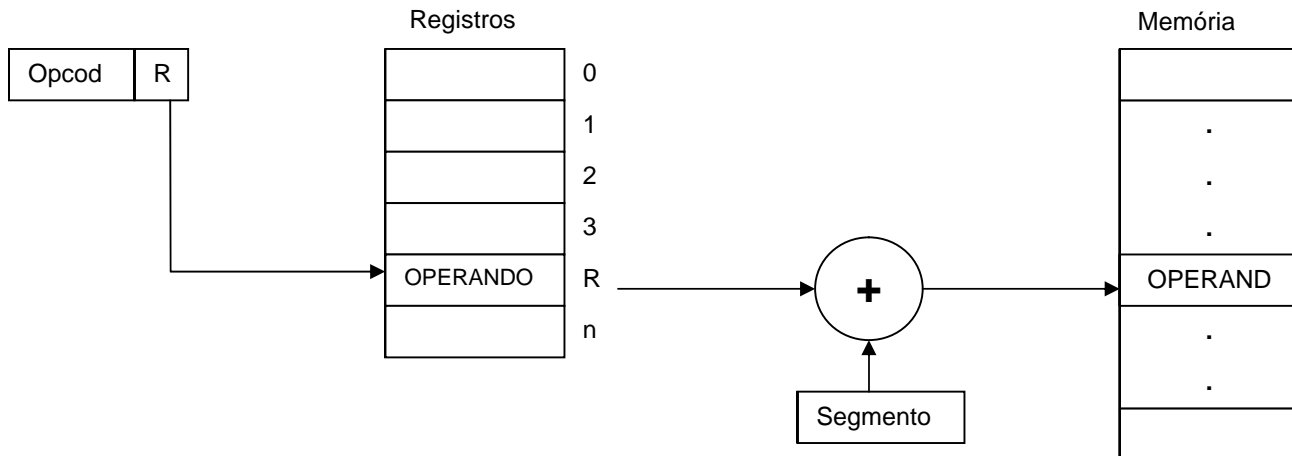


FIG. 3.9 – Endereçamento por Registro Indireto

Endereçamento baseado

Neste tipo de endereçamento, a instrução contém o offset e um registro de base vai conter o endereço de base. O endereçamento baseado é usado tipicamente nos casos em que a posição relativa de um componente é conhecida, mas não o seu endereço de base (este só será conhecido em tempo de execução). Qualquer um dos 4 registros BX, BP, SI ou DI pode ser usado como um registro de base. O endereço efetivo do operando é formado pela adição dos 16 bits de endereço especificado no registro de base mais os 8 ou 16 bits de deslocamento contido na instrução.

Aplicações típicas do endereçamento baseado são:

- . acesso a dados numa área de parâmetros cujo endereço de base é passado a uma rotina;
- . acesso a dados e salto para endereços em programas independentes de posição, quando o registro de base é carregado com o endereço inicial do programa.

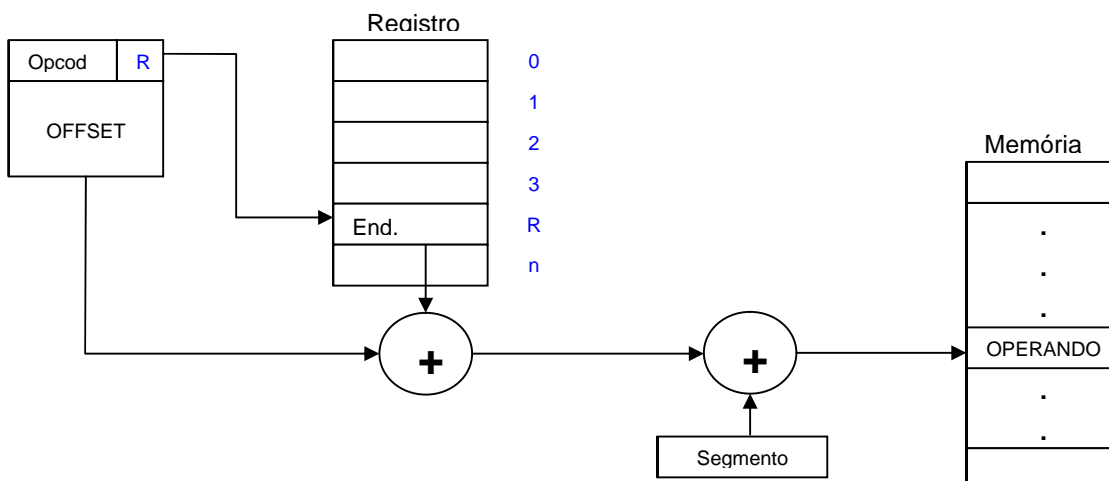


FIG. 3.10 – End. Baseado

Endereçamento indexado

O 8086 não tem um modo de endereçamento indexado separado. Entretanto, o modo baseado com um deslocamento de 16 bits é logicamente equivalente ao modo indexado. Neste caso, a instrução contém os 16 bits do endereço base, e o registrador base contém um valor de índice.

Embora qualquer um dos 4 registradores base do 8086 possam ser usados nos modos baseado e indexado, os projetistas do 8086 escolheram os registros BX e BP para serem usados como registros de base e SI e DI como registros de índice.

Enquanto os modos de endereçamento baseado e indexado possuam a mesma codificação na linguagem de máquina do 8086, a linguagem assembly do 8086 fornece uma notação separada para elas. A notação para o modo indexado é `addr16[bireg]`, e é similar à notação usada nas linguagens de alto nível e em outras linguagem assembly.

Baseado indexado

O modo de endereçamento baseado indexado forma um endereço efetivo pela adição de um endereço base em um deslocamento, ambos contidos em registradores. Isto permite que tanto o endereço base da estrutura de dados como um deslocamento para um de seus componentes sejam calculados em tempo de execução. Tanto BX ou BP podem ser usados como registro de endereço base, e tanto SI ou DI podem ser usados como registros de índice. Portanto, existem 4 combinações diferentes de registros que podem ser usados no endereçamento baseado indexado.

Endereçamento relativo

No 8086, o endereçamento relativo é usado somente nas instruções de jump, call, desvio condicional e controle de loop. O endereço efetivo é calculado como a soma do IP e um deslocamento contido na instrução. O valor do IP usado é o endereço do byte seguinte a instrução corrente.

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.