```
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 2

CONCEITOS FUNDAMENTAIS

Este capítulo trata dos conceitos essenciais e preliminares da estrutura computacional, que fornecem a base para o estudo inicial da linguagem de programação Assembly para os microprocessadores da família x86, mais precisamente o 8086 ou seu irmão 8088. Assim sendo, destacam-se detalhes do sistema computacional (unidade central de processamento, unidade de memória e unidade de entrada e saída), a organização interna dos dados (nibble, byte, word, double word e quad word), o uso dos registradores (gerais, segmento, apontamento e estado), as interrupções (hardware, exceção e software), os segmentos, deslocamentos e endereçamento de memória.

2.1 Sistema computacional

Um sistema computacional é o conjunto de componentes que formam a configuração de um computador, incluindo seu sistema operacional e todos os periféricos conectados a esse computador. Dentre os vários componentes, os mais importantes são a unidade central de processamento, as unidades de memória (memória RAM, memória ROM e memória de massa ou secundária), as unidades de entrada e saída, a unidade aritmética, unidade de controle lógico, unidade de saída e registradores as quais são descritas a seguir. A Figura 2.1 mostra um modelo esquemático da organização interna de um computador em linha geral.

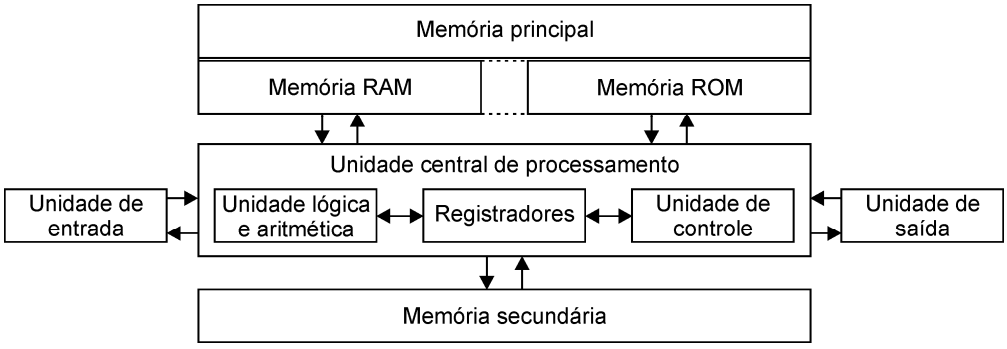

Figura 2.1 - Organização geral de um computador (TOKHEIN, 1985, p.2).

A memória principal se divide em memória RAM (*Random Access Memory* - memória de acesso randômico) e memória ROM (*Read Only Memory* - memória somente de leitura).

A memória RAM é o local onde os programas e dados são armazenados para a operação do sistema. Sendo volátil, ela não armazena os dados e programas para uso futuro. Quando o sistema é desligado, todo e qualquer dado ou programa armazenado nessa memória é perdido. Para sanar essa deficiência aparente, usa-se a memória secundária formada pelas unidades de armazenamento de dados conectadas ao computador, sendo a principal delas o HD (*Hard disk* - disco rígido) conectado dentro do gabinete.

Na memória ROM se encontra armazenado o programa básico de entrada e saída do sistema, e os dados armazenados nessa memória somente são lidos, não sendo nenhum dado escrito.

O *sistema computacional* de um microcomputador padrão IBM-PC (PC de *Personal Computer*) que faz uso da tecnologia Intel/AMD possui em seu microprocessador as várias unidades ligadas por um conjunto de componentes eletrônicos denominados *bus* (também conhecido como *barramento* ou *via*). Por meio do *bus* os sinais binários são transportados entre as unidades de barramento e de execução para a comunicação com a memória. Para que as operações sejam internamente executadas, existem o *bus de controle lógico*, o *bus de endereçamento de dados*, o *bus de operação externa* e o *bus de dados*. Mais detalhes sobre esta parte encontram-se no tópico sobre a arquitetura 8086, neste capítulo.

2.1.1 - Unidade central de processamento

A *unidade central de processamento*, também denominada CPU (*Central Processing Unit*), é erroneamente confundida com o gabinete do equipamento. O gabinete é apenas o compartimento (invólucro) que protege a placa mãe (*mainboard*) e seus componentes eletrônicos, entre eles o mais importante componente denominado CPU.

Dos componentes eletrônicos existentes em um microcomputador a CPU (microprocessador) é o mais importante, na qual se encontra a unidade de controle lógica e aritmética do computador, conhecida pelo nome ALU (*Aritmetic and Logic Unit*, ou ULA, unidade lógica e aritmética). Ela é responsável por executar as operações matemáticas e o controle lógico das ações que a máquina executa.

De forma mais ampla, a CPU tem a função de controlar a leitura e escrita de dados e informações nos registradores gerais, temporários, de endereçamentos e de estado (registradores são células de memória usadas para armazenamento temporário de dados, localizadas na memória RAM que se assemelham ao conceito de variáveis), além de controlar o tráfego de dados que passa pelo barramento de dados e executar as instruções de um programa em uso.

2.1.2 - Unidade de memória

É a parte responsável por armazenar dados para que eles possam ser controlados posteriormente. A unidade de memória opera em conjunto com a CPU, mas não faz parte direta da CPU. A unidade de memória se divide em duas partes, a saber:

- A memória principal conhecida pelo nome de memória RAM (Random Access Memory), ou seja, memória de acesso aleatório. Nessa memória a CPU executa as ações de controle, na qual se encontram os registradores. As informações armazenadas nessa memória são voláteis, ou seja, temporárias. Após o desligamento do computador os dados armazenados nos registradores são automaticamente perdidos. A memória principal é responsável por manter um programa em execução, o qual é controlado pela CPU. Pelo fato de essa memória ser volátil, torna-se necessária a utilização de memórias secundárias, quando se deseja manter uma informação ou dados gravados.
- A memória secundária também referenciada pelo termo memória auxiliar ou memória de massa. Nesse tipo de memória os dados de um programa permanecem armazenados por um determinado espaço de tempo. São memórias secundárias as fitas magnéticas, os discos magnéticos (disquetes e discos rígidos), os discos ópticos (CDs, CD-Rs, CD-RWs), cartões de memória, entre outros mecanismos que possam ser utilizados como elementos de armazenamento.

2.1.3 - Unidades de entrada e de saída

As unidades de entrada e de saída são os componentes responsáveis pela comunicação do mundo exterior com um computador e vice-versa. Efetivamente essa unidade faz uso do *bus* de operação externa.

Dos componentes para a efetivação de entrada, os mais conhecidos e utilizados são o teclado, o mouse, os scanners de leitura óptica (semelhantes aos usados nos caixas de supermercados). Com esse tipo de componente é possível obter dados do mundo externo para que sejam armazenados na memória principal e manipulados pela CPU, e posteriormente podem ser gravados em uma memória secundária.

Dos componentes para a efetivação de saída, os mais conhecidos são o vídeo e a impressora. Com esses componentes é possível enviar para o mundo externo os dados e informações existentes em uma memória secundária, ou mesmo fornecidos anteriormente por um componente de entrada.

2.2 - Organização de dados

Os valores binários utilizados internamente pelos vários circuitos de um computador digital para a formação dos caracteres alfanuméricos e também para a definição de suas operações internas são agrupados em conjuntos de *bits* (**bi**nary digi**t** - dígito binário). *Bit* é a menor informação que pode ser manipulada em um computador.

O conjunto de *bits* mais conhecido é o *byte* (conjunto de oito *bits*, em português octeto), no entanto há outras formas de representar conjuntos numéricos de *bits* em um computador, que são menos discutidas e, por conseguinte, menos conhecidas, tais como *nibble*, *word*, *double word* e *quad word*. Esses conjuntos de agrupamento de *bits* são responsáveis pela forma de organização e representação interna dos dados na memória de um computador digital.

Partindo da premissa de que um computador digital manipula *bits* de dados em memória, e esses *bits* são representados apenas pelos valores binários 1 (um - circuito ligado) e 0 (zero - circuito desligado), torna-se óbvia a necessidade de combinar esses valores para que se consigam sinais diferentes.

Se um computador digital possuísse a capacidade de manipular apenas dois *bits*, ele conseguiria representar no máximo quatro valores diferentes, sendo eles: 00, 01, 11 e 10, ou seja, o valor 2 (quantidade de valores binários - 0 e 1) elevado a 2 (capacidade máxima de combinação entre os valores binários) num total de quatro combinações.

A partir desse detalhe técnico passou-se a considerar a necessidade de uma capacidade de manipulação de dados maior em um computador digital, e à medida que o tempo passa, os computadores digitais passam também a manipular quantidades maiores de dados. Esse cálculo (capacidade máxima de manipulação da quantidade de dados) é sempre feito com múltiplos de 2.

Considerando a quantidade máxima anterior de combinação entre os valores binários e multiplicando esse valor por 2, ter-se-á o valor 4, ou então, basta efetuar o cálculo $2^2 = 4$. A partir desse raciocínio fica fácil compor a capacidade das próximas combinações a serem manipuladas em um computador, que serão 8, 16, 32, 64, 128 e assim por diante.

2.2.1 - Nibble

É um conjunto numérico de quatro *bits* (quarteto), sendo a menor estrutura numérica manipulada internamente em um computador. A capacidade de armazenamento baseada em um *nibble* é obtida a partir do cálculo de 2^4, o que possibilita representar até 16 valores diferentes (em hexadecimal de 0 a F, em decimal de 0 a 15 e em binário de 0000 a 1111). Um *nibble* é usado para facilitar a representação de valores binários em formato hexadecimal. Os valores em formato hexadecimal são representados pelos símbolos ("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E" e "F"), perfazendo um total de dezesseis dígitos diferentes.

A Tabela 2.2 apresenta os dezesseis valores que podem ser armazenados em um *nibble* de memória no formato decimal, hexadecimal e binário. Observe a coluna identificada como **Binário** com o agrupamento de valores que formam o conjunto de quatro *bits* (*nibble*).

Tabela 2.1 - Valores que podem ser armazenados em um nibble de memória

Decimal	Hexadecimal	Binário
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

Decimal	Hexadecimal	Binário
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Perceba que na coluna **Binário** encontram-se as dezesseis combinações possíveis de serem conseguidas com quatro valores binários.

2.2.2 - Byte

É um conjunto de oito *bits* (octeto). Com esta estrutura é possível representar numericamente até 256 (que equivale a 2^8) valores diferentes que podem ser utilizados por um computador digital, principalmente nos equipamentos da família IBM-PC.

A estrutura interna de um único *byte* é numerada de 7 (sete) a 0 (zero) da esquerda para a direita, sendo os *bits* de posição 7, 6, 5 e 4 os mais significativos e os *bits* de posição 3, 2, 1 e 0 os menos significativos. Desta forma a estrutura interna de um *byte* é representada pelo formato indicado junto a Tabela 2.2.

Tabela 2.2 - Estrutura de um byte de memória

7	6	5	4	3	2	1	0
Bits mais significativos				*Bits* menos significativos			
Nibble 1 (mais significativo)				*Nibble* 0 (menos significativo)			

É pertinente salientar que o *bit* 7 (primeiro *bit* na posição esquerda, ou *bit* mais significativo) é reservado como *bit de sinal* usado na representação de valores positivos ou negativos. O valor de um *byte* é positivo quando o *bit* 7 é representado pelo valor binário 0 (zero) e será negativo caso este valor seja representado pelo valor binário 1 (um). Por exemplo, o valor binário **00001010** representa em decimal **10** positivo, enquanto que **10001010** representa em decimal **10** negativo. Assim sendo, observe na Tabela 2.3 o formato de um *byte* a partir da ótica de uso do *bit de sinal*.

Tabela 2.3 - Formato de um byte de memória a partir do bit de sinal

7	6	5	4	3	2	1	0
Sinal	*Magnitude*						

Observe que a indicação *Sinal* corresponde ao uso do *bit de sinal* que determina se o valor nas demais posições é positivo ou negativo. Os *bits* que representam a *Magnitude* são usados na representação do valor absoluto em si, independentemente deste valor ser positivo ou negativo. Um valor **+10** ou **-10** é **10** independentemente deste ser positivo ou negativo.

Perceba que um *byte* é formado por um conjunto de dois *nibbles*. Assim sendo, há um *nibble* (zero) menos significativo e outro *nibble* (um) mais significativo. O *nibble* mais significativo está representado pelos *bits* situados nas posições de 7 a 4 e o *nibble* menos significativo está representado pelos *bits* situados nas posições de 3 a 0.

2.2.3 - Word

Representa um conjunto de dezesseis *bits*. A estrutura de um tipo de dado *word* é numerada numa sequência de 15 (quinze) a 0 (zero) *bits*, sendo os *bits* 15, 14, 13, 12, 11, 10, 9 e 8 considerados os mais significativos e os *bits* 7, 6, 5, 4, 3, 2, 1 e 0 os menos significativos. Com a estrutura de dados *word* é possível representar numericamente até 65.536 (2 elevado a 16) valores diferentes. O tipo de dado *word* possui seu formato como indicado na Tabela 2.4.

Tabela 2.4 - Estrutura de um word de memória

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

Um dado *word* é formado por dois *bytes*. Assim sendo, há um *byte* menos significativo e outro mais significativo. O *byte* mais significativo está representado pelos *bits* situados nas posições de 15 a 8 e o menos significativo está representado pelos *bits* situados nas posições de 7 a 0. O tipo de dado *word* pode ser dividido em quatro *nibbles*. Neste caso o *nibble 3* é o mais significativo e o *0* é o menos significativo. Os *nibbles 2* e *1* são simples, sem nenhuma classificação estrutural.

Como apresentado o tipo de dado *word* pode representar 65.536 valores diferentes entre 0 (zero) e 65.535. Também é possível representar valores numéricos entre -32.766 até 32.767. Normalmente se utiliza o tipo de dado *word* para representar valores inteiros e deslocamentos de segmento.

2.2.4 - Double word

É um conjunto de dois *words*, ou seja, um conjunto de 32 *bits*. A estrutura de um dado do tipo *double word* é numerada de 31 (trinta e um) a 0 (zero). Assim sendo, um *double word* é formado por dois *words*, ou quatro *bytes*, ou ainda oito *nibbles*.

Com o tipo *double word* é possível representar 4.294.967.296 valores numéricos diferentes (ou seja, 2 elevado a 32) entre 0 (zero) e 4.294.967.295. Também é possível representar com o dado do tipo *double word* valores numéricos entre -2.147.483.648 e 2.147.483.647. Normalmente se utiliza *double word* para representar valores inteiros de 32 *bits*, valores de ponto flutuante de 32 *bits* e endereços segmentados.

2.2.5 - Quad word

É um conjunto de dois *double words*, ou seja, um conjunto de 64 *bits*. A estrutura de um dado do tipo *quad word* é numerada de 63 (sessenta e três) a 0 (zero). Assim sendo, um *quad word* é formado por dois *double words*, ou quatro *words*, ou oito *bytes*, ou ainda dezesseis *nibbles*.

Com o tipo *quad word* é possível representar 18.446.744.073.709.551.616 valores numéricos diferentes (ou seja, 2 elevado a 64) entre 0 (zero) e 18.446.744.073.709.551.615. Pode-se também representar com o dado do tipo *double word* valores numéricos entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807. O tipo de dado *quad word* é normalmente encontrado em computadores digitais dotados de microprocessadores de 64 *bits*. Por esta razão não será detalhado esse tipo de dado, pois foge do propósito do livro.

2.2.6 - Unidades de medidas computacionais

O volume de dados a ser processado numa memória principal ou armazenado numa memória secundária é medido em relação à quantidade de *bytes*, como já exposto. A Tabela 2.5 apresenta um resumo de algumas unidades de medida computacional utilizadas na área da computação.

Tabela 2.5 - Unidade de medida computacional

Unidade	Quantidade de caracteres
Bit (b)	Conjunto de dois *bits* que possibilita a ativação e a desativação de recursos e circuitos eletrônicos. Menor dado que pode ser operacionalizado dentro de um computador.
Nibble (N)	Conjunto de quatro *bits* que possibilita a definição do número mínimo de algarismos binários necessários para representar uma cifra decimal. Base para o sistema BDC (*Binary-Coded Decimal*).
Byte (B)	Conjunto de oito *bits* que possibilita a definição e o uso de duzentos a cinquenta e seis (2^8) símbolos para representação de caracteres numéricos, alfabéticos, de pontuação e gráficos (opcional).
KByte (KB)	Definição da quantidade de caracteres a ser utilizada e armazenada em memórias: principal e secundária. 1 KByte (*kilobyte*) equivale a 1.024 caracteres, sendo obtido a partir de 2^{10}.
Mbyte (MB)	Definição da quantidade de caracteres a ser utilizada e armazenada em memórias: principal e secundária. 1 Mbyte (*megabyte*) equivale a 1.048.576 caracteres (1.024 KBytes), sendo obtido a partir de 2^{20}.
Gbyte (GB)	Definição da quantidade de caracteres a ser utilizada e armazenada em memórias: principal e secundária. 1 Gbyte (*gigabyte*) equivale a 1.073.741.824 caracteres (1.024 Mbytes), sendo obtido a partir de 2^{30}.
Tbyte (TB)	Definição da quantidade de caracteres a ser utilizada e armazenada em memórias: principal e secundária. 1 Tbyte (*terabyte*) equivale a 1.099.511.627.776 caracteres (1.024 Gbytes), sendo obtido a partir de 2^{40}.
Pbyte (PB)	Definição da quantidade de caracteres a ser utilizada e armazenada em memórias: principal e secundária. 1 Pbyte (*petabyte*) equivale a 1.125.899.906.842.624 caracteres (1.024 Tbytes), sendo obtido a partir de 2^{50}.

Unidade	Quantidade de caracteres
Ebyte (EB)	Definição da quantidade de caracteres a ser utilizada e armazenada em memórias: principal e secundária. 1 Ebyte (*exabyte*) equivale a 1.152.921.504.606.846.976 caracteres (1.024 Pbytes), sendo obtido a partir de 2^{60}.
Zbyte (ZB)	Definição da quantidade de caracteres a ser utilizada e armazenada em memórias: principal e secundária. 1 Zbyte (*zettabyte*) equivale a 1.180.591.620.717.411.303.424 caracteres (1.024 Ebytes), sendo obtido a partir de 2^{70}.
Ybyte (YB)	Definição da quantidade de caracteres a ser utilizada e armazenada em memórias: principal e secundária. 1 Ybyte (*yottabyte*) equivale a 1.208.925.819.614.629.174.706.176 caracteres (1.024 Zbytes), sendo obtido a partir de 2^{80}.

2.2.7 - Sistemas de numeração e potências computacionais

A seguir é apresentada uma tabela para facilitar a conversão de valores no sistema de numeração computacional básico. São mostrados valores grafados nos sistemas decimal, binário e hexadecimal, correspondentes às 256 combinações possíveis de serem obtidas a partir de um conjunto de oito *bits* (*byte*).

Para converter um valor de um sistema numérico de 8 *bits* no outro valor também de 8 *bits*, basta localizar o valor na base desejada e olhar para o seu equivalente na mesma linha. A conversão de valores acima de 8 *bits* pela tabela seguinte pode ser facilmente conseguida para valores binários e hexadecimais. Por exemplo, o valor binário de 16 *bits* **0101 1010 1100 0010** equivale em hexadecimal a **5AC2** e para fazer a conversão pela Tabela 2.6, divida os 16 *bits* em dois *bytes* e busque na tabela os valores correspondentes de cada *byte* e agrupe novamente os dois valores.

Tabela 2.6 - Tabela de conversão de bases

Decimal	Binário	Hexadecimal
000	0000 0000	00
001	0000 0001	01
002	0000 0010	02
003	0000 0011	03
004	0000 0100	04
005	0000 0101	05
006	0000 0110	06
007	0000 0111	07
008	0000 1000	08
009	0000 1001	09
010	0000 1010	0A
011	0000 1011	0B
012	0000 1100	0C
013	0000 1101	0D
014	0000 1110	0E
015	0000 1111	0F
016	0001 0000	10
017	0001 0001	11
018	0001 0010	12
019	0001 0011	13

Decimal	Binário	Hexadecimal
026	0001 1010	1A
027	0001 1011	1B
028	0001 1100	1C
029	0001 1101	1D
030	0001 1110	1E
031	0001 1111	1F
032	0010 0000	20
033	0010 0001	21
034	0010 0010	22
035	0010 0011	23
036	0010 0100	24
037	0010 0101	25
038	0010 0110	26
039	0010 0111	27
040	0010 1000	28
041	0010 1001	29
042	0010 1010	2A
043	0010 1011	2B
044	0010 1100	2C
045	0010 1101	2D

Decimal	Binário	Hexadecimal
020	0001 0100	14
021	0001 0101	15
022	0001 0110	16
023	0001 0111	17
024	0001 1000	18
025	0001 1001	19
052	0011 0100	34
053	0011 0101	35
054	0011 0110	36
055	0011 0111	37
056	0011 1000	38
057	0011 1001	39
058	0011 1010	3A
059	0011 1011	3B
060	0011 1100	3C
061	0011 1101	3D
062	0011 1110	3E
063	0011 1111	3F
064	0100 0000	40
065	0100 0001	41
066	0100 0010	42
067	0100 0011	43
068	0100 0100	44
069	0100 0101	45
070	0100 0110	46
071	0100 0111	47
072	0100 1000	48
073	0100 1001	49
074	0100 1010	4A
075	0100 1011	4B
076	0100 1100	4C
077	0100 1101	4D
078	0100 1110	4E
079	0100 1111	4F
080	0101 0000	50
081	0101 0001	51
082	0101 0010	52

Decimal	Binário	Hexadecimal
046	0010 1110	2E
047	0010 1111	2F
048	0011 0000	30
049	0011 0001	31
050	0011 0010	32
051	0011 0011	33
105	0110 1001	69
106	0110 1010	6A
107	0110 1011	6B
108	0110 1100	6C
109	0110 1101	6D
110	0110 1110	6E
111	0110 1111	6F
112	0111 0000	70
113	0111 0001	71
114	0111 0010	72
115	0111 0011	73
116	0111 0100	74
117	0111 0101	75
118	0111 0110	76
119	0111 0111	77
120	0111 1000	78
121	0111 1001	79
122	0111 1010	7A
123	0111 1011	7B
124	0111 1100	7C
125	0111 1101	7D
126	0111 1110	7E
127	0111 1111	7F
128	1000 0000	80
129	1000 0001	81
130	1000 0010	82
131	1000 0011	83
132	1000 0100	84
133	1000 0101	85
134	1000 0110	86
135	1000 0111	87

Decimal	Binário	Hexadecimal
083	0101 0011	53
084	0101 0100	54
085	0101 0101	55
086	0101 0110	56
087	0101 0111	57
088	0101 1000	58
089	0101 1001	59
090	0101 1010	5A
091	0101 1011	5B
092	0101 1100	5C
093	0101 1101	5D
094	0101 1110	5E
095	0101 1111	5F
096	0110 0000	60
097	0110 0001	61
098	0110 0010	62
099	0110 0011	63
100	0110 0100	64
101	0110 0101	65
102	0110 0110	66
103	0110 0111	67
104	0110 1000	68
158	1001 1110	9E
159	1001 1111	9F
160	1010 0000	A0
161	1010 0001	A1
162	1010 0010	A2
163	1010 0011	A3
164	1010 0100	A4
165	1010 0101	A5
166	1010 0110	A6
167	1010 0111	A7
168	1010 1000	A8
169	1010 1001	A9
170	1010 1010	AA
171	1010 1011	AB
172	1010 1100	AC

Decimal	Binário	Hexadecimal
136	1000 1000	88
137	1000 1001	89
138	1000 1010	8A
139	1000 1011	8B
140	1000 1100	8C
141	1000 1101	8D
142	1000 1110	8E
143	1000 1111	8F
144	1001 0000	90
145	1001 0001	91
146	1001 0010	92
147	1001 0011	93
148	1001 0100	94
149	1001 0101	95
150	1001 0110	96
151	1001 0111	97
152	1001 1000	98
153	1001 1001	99
154	1001 1010	9A
155	1001 1011	9B
156	1001 1100	9C
157	1001 1101	9D
207	1100 1111	CF
208	1101 0000	D0
209	1101 0001	D1
210	1101 0010	D2
211	1101 0011	D3
212	1101 0100	D4
213	1101 0101	D5
214	1101 0110	D6
215	1101 0111	D7
216	1101 1000	D8
217	1101 1001	D9
218	1101 1010	DA
219	1101 1011	DB
220	1101 1100	DC
221	1101 1101	DD

Decimal	Binário	Hexadecimal
173	1010 1101	AD
174	1010 1110	AE
175	1010 1111	AF
176	1011 0000	B0
177	1011 0001	B1
178	1011 0010	B2
179	1011 0011	B3
180	1011 0100	B4
181	1011 0101	B5
182	1011 0110	B6
183	1011 0111	B7
184	1011 1000	B8
185	1011 1001	B9
186	1011 1010	BA
187	1011 1011	BB
188	1011 1100	BC
189	1011 1101	BD
190	1011 1110	BE
191	1011 1111	BF
192	1100 0000	C0
193	1100 0001	C1
194	1100 0010	C2
195	1100 0011	C3
196	1100 0100	C4
197	1100 0101	C5
198	1100 0110	C6
199	1100 0111	C7
200	1100 1000	C8
201	1100 1001	C9
202	1100 1010	CA
203	1100 1011	CB
204	1100 1100	CC
205	1100 1101	CD
206	1100 1110	CE

Decimal	Binário	Hexadecimal
222	1101 1110	DE
223	1101 1111	DF
224	1110 0000	E0
225	1110 0001	E1
226	1110 0010	E2
227	1110 0011	E3
228	1110 0100	E4
229	1110 0101	E5
230	1110 0110	E6
231	1110 0111	E7
232	1110 1000	E8
233	1110 1001	E9
234	1110 1010	EA
235	1110 1011	EB
236	1110 1100	EC
237	1110 1101	ED
238	1110 1110	EE
239	1110 1111	EF
240	1111 0000	F0
241	1111 0001	F1
242	1111 0010	F2
243	1111 0011	F3
244	1111 0100	F4
245	1111 0101	F5
246	1111 0110	F6
247	1111 0111	F7
248	1111 1000	F8
249	1111 1001	F9
250	1111 1010	FA
251	1111 1011	FB
252	1111 1100	FC
253	1111 1101	FD
254	1111 1110	FE
255	1111 1111	FF

A Tabela 2.7 apresenta os valores dos resultados das potências de dois de zero a dezesseis.

Tabela 2.7 - Tabela com os valores de potencia dois

Expoente	Valor
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1.024
11	2.048
12	4.096
13	8.192
14	16.384
15	32.768
16	65.536

Expoente	Valor
17	131.072
18	262.144
19	524.288
20	1.048.576
21	2.097.152
22	4.194.304
23	8.388.608
24	16.777.216
25	33.554.432
26	67.108.864
27	134.217.728
28	268.435.456
29	536.870.912
30	1.073.741.824
31	2.147.483.648
32	4.294.967.296

A Tabela 2.8 apresentada os valores dos resultados das potências de dezesseis de zero a oito.

Tabela 2.8 - Tabela com os valores de potencia dezesseis

Expoente	Valor
0	1
1	16
2	256
3	4.096
4	65.536

Expoente	Valor
5	1.048.576
6	16.777.216
7	268.435.456
8	4.294.967.296

2.3 - Arquitetura 8086

O microprocessador 8086 foi apresentado pela empresa Intel no ano de 1978 (MORSE, 1987, p. 10), tendo sido desenvolvido pelo engenheiro e PhD Stephen P. Morse com a principal finalidade de ser usado em calculadoras desenvolvidas pela empresa Intel. Esse processador foi também utilizado para fazer o controle de automatização de semáforos de trânsito, como já ocorria com seus antecessores.

Ninguém imaginou, em 1978, que o microprocessador 8086 poderia vir a ser utilizado como “cérebro” de um microcomputador. Quando a IBM decidiu projetar e desenvolver seu microcomputador, optou por usar o microprocessador da Intel, mas percebeu que não seria possível, pois ele era demais avançado para sua máquina. Na época todos os periféricos fabricados operavam com barramento de dados (*bus*) de 8 *bits*. Assim sendo, o microprocessador 8086 não conseguia conversar com os periféricos existentes no mercado. O microprocessador 8086 operava internamente a 16 *bits* e se comunicava externamente a 16 *bits*. Era preciso que o microcomputador se comunicasse com os periféricos externos a 8 *bits*. Foi quando a Intel desenvolveu, em 1979, uma versão simplificada do 8086 chamado 8088. Essa versão simplificada operava internamente a 16 *bits*, comunicava-se externamente a 8 *bits*. Os microprocessadores 8086 e 8088 são idênticos no nível de processamento interno, tendo como diferença o barramento de dados para comunicação com periféricos externos.

A partir do lançamento do microprocessador 8088, foi possível à IBM lançar seu microcomputador IBM-PC (PC de *Personal Computer*), em 12 de agosto de 1981, e também o modelo IBM-XT (XT de *eXTended*), em 8 de março de 1983. O IBM-XT era idêntico ao IBM-PC, tendo como diferença principal a existência de uma unidade de disco rígido de 10 MB (*Megabyte*). Basicamente o microprocessador 8086 não chegou a ser utilizado em nenhum dos microcomputadores da IBM. O uso da tecnologia 8086, ou seja, um microprocessador de 16 *bits* com barramento 16 *bits* somente aconteceu quando a Intel redesenhou o microprocessador e o chamou de 80286, introduzindo-o no mercado em 1º de fevereiro de 1982. Isso possibilitou à IBM lançar, em agosto de 1984, o microcomputador IBM-AT (AT de *Advanced Tecnology*). Além do novo microprocessador, esse equipamento vinha com um disco rígido de 20 Mb.

A partir de então, a empresa Intel vem lançando microprocessadores pertencentes à família x86, destacando-se os já comentados, além dos microprocessadores 80186 (1982), 80188 (1982), 80286 (1982), 80386 (1985), 80486 (1990) e toda a série Pentium (a partir de 1993). No lançamento do 80286, os periféricos no mercado já se comunicam a 16 *bits* e atualmente a 32 *bits*.

Pelo fato de todos esses microprocessadores pertencerem à família x86, eles operam basicamente com o mesmo conjunto de instruções, e a cada lançamento novas instruções são inseridas, a menos nos casos de simplificação de operação de barramento de dados, sendo este um dos motivos em que se aprende a programação em linguagem de máquina a partir dos microprocessadores 8086.

O microprocessador 8086 e seu irmão 8088 são divididos em duas unidades, sendo BIU (*Bus Interface Unit* - unidade de interface de barramento de dados) e EU (*Execution Unit* - unidade de execução). Para entender melhor o que aconteceu no processo de simplificação do microprocessador 8086 para o microprocessador 8088, observe nas Figuras 2.3 e 2.4 a estrutura funcional simplificada da arquitetura desses microprocessadores.

Ao observar as Figuras 2.3 e 2.4, é possível notar as diferenças existentes nos diagramas entre os números de instruções de fila de 6, para o 8086, a 4, para o 8088, e o barramento de dados a partir do módulo somador de 16 para 8 *bits* no sentido de direcionar o fluxo de comunicação externa com os periféricos por meio do barramento de controle lógico.

A unidade BIU é responsável por efetivar todas as operações de transferências de endereços e de dados por meio do barramento de dados e dos registradores[1] de deslocamento **CS**, **DS**, **SS**, **ES** e **IP** para efetivar operações na memória, além de estabelecer comunicação com o barramento de controle lógico por meio dos barramentos de endereço e de dados no sentido de acessar o barramento externo e se comunicar com os periféricos agregados ao equipamento.

A unidade EU tem por base se comunicar por meio do barramento de dados da ALU (*Arithmetic Logic Unit* - unidade lógica e aritmética) com a unidade BIU e informar o local onde os dados ou instruções serão executados e em quais endereços. Essa unidade opera sobre os registradores gerais e registradores de estados (*flags*). Todas as operações dessa unidade são controladas pela ALU e pela EU Control System (*Execution Unit Control System* - unidade de controle de execução do sistema).

Enquanto a unidade EU executa alguma instrução que não exige o uso do barramento de dados da ALU, a unidade BIU apanha até seis *bytes* de instrução da próxima instrução e armazena-os no barramento de fila. Assim que a EU estiver pronta para executar a próxima instrução, ela coleta os *bytes* no barramento de fila. Esse ciclo de execução busca a próxima instrução enquanto outra instrução é executada pelo ciclo de controle denominado *pipelining* (canalização).

O funcionamento de um microprocessador 8086/8088 é baseado a partir do seguinte procedimento de trabalho: a unidade *BIU* coloca a instrução apontada no registrador **IP** somado ao registrador **CS** no barramento de fila para que esta instrução seja operacionalizada pela unidade EU. Em seguida o registrador **IP** é incrementado e passa a apontar o endereço de memória da próxima instrução, que após ser lida é executada. A unidade EU pega sempre a primeira instrução da fila e faz sua execução. Enquanto a unidade EU executa uma instrução a unidade BIU efetua nova busca de instrução e preenche a fila. Caso a instrução a ser executada pela unidade EU seja demorada, a unidade BIU fará o preenchimento de toda a fila.

As instruções da fila não são usadas quando estiver em execução instruções de acesso a memória ou estiver em uso instruções de desvios, que provocam o descarte e a sobreposição do conteúdo da fila.

---

[1] *Registradores são células especiais de memória usadas para o armazenamento temporário de dados que tenham no máximo o tamanho de uma* word.

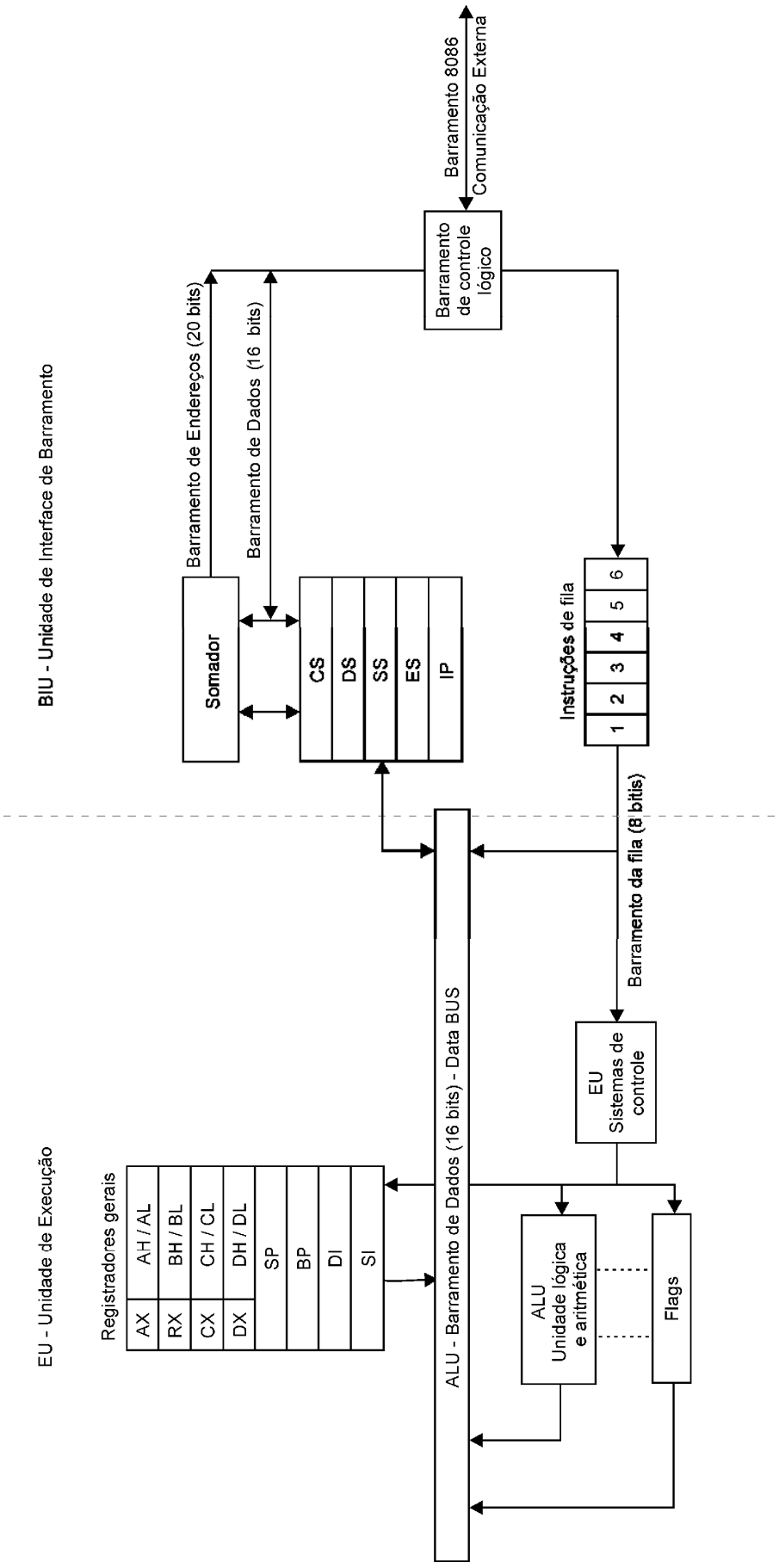

Figura 2.3 - Diagrama de bloco funcional do microprocessador 8086. Adaptado de INTEL, 1985, p. 1-4.

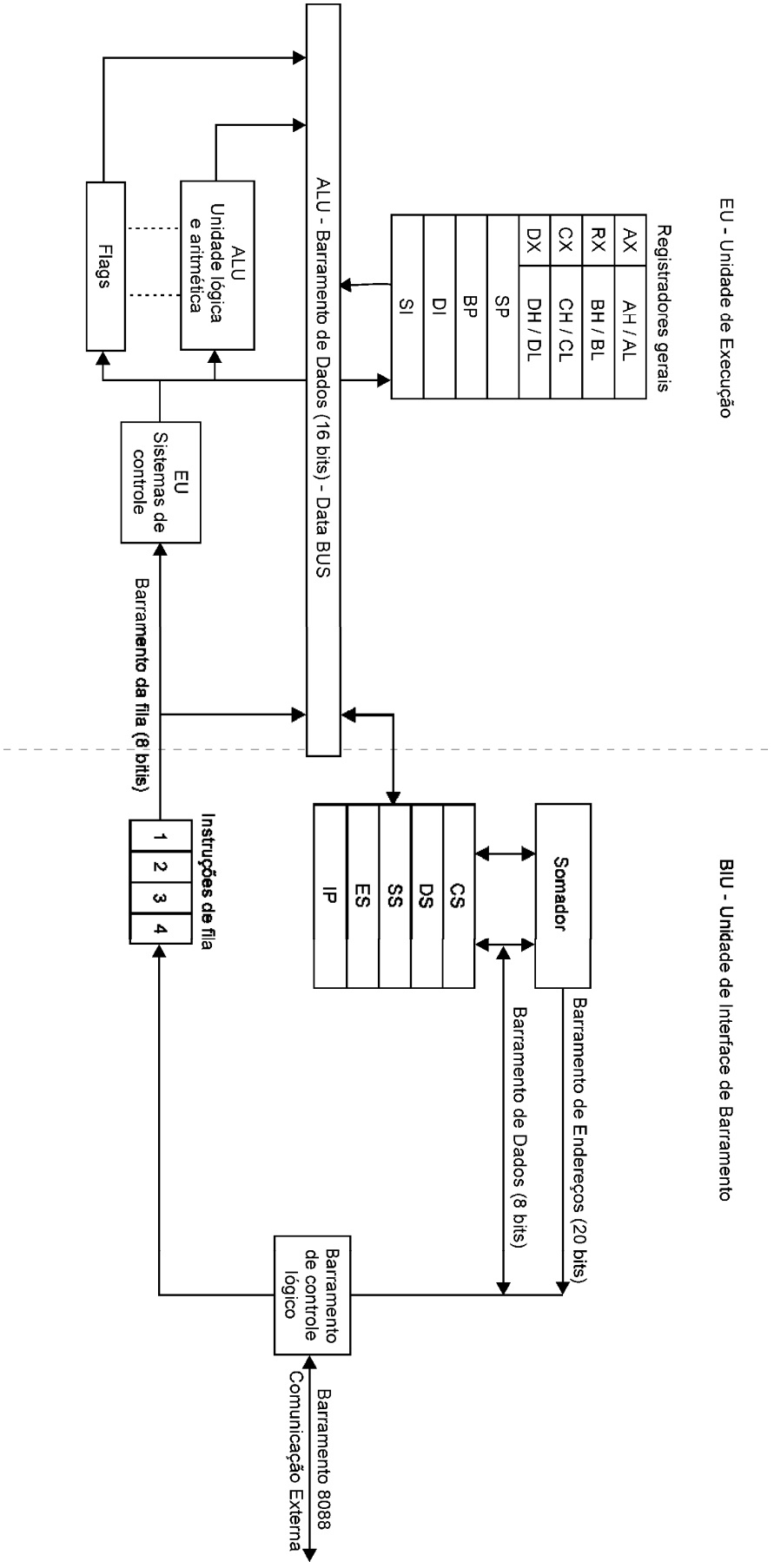

Figura 2.4 - Diagrama de bloco funcional do microprocessador 8088. Adaptado de INTEL, 1985, p. 1-5.

Como pode ser percebido, os microprocessadores 8086 e 8088 utilizam registradores para efetivar o armazenamento temporário de dados em memória, os quais serão manipulados por intermédio de um programa. Grosso modo, os registradores são semelhantes às variáveis encontradas nas linguagens de programação de alto nível para tratar a maior parte dos dados em memória.

O registrador está intimamente relacionado com a estrutura do microprocessador em uso. Para cada tipo de microprocessador existe uma forma peculiar de tratar este conceito. Será apresentada e considerada a estrutura de registradores para computadores digitais baseada na arquitetura do microprocessador 8086 da empresa Intel (podendo considerar também os microprocessadores fornecidos por outros fabricantes, como é o caso do AMD), conforme a Figura 2.6, a qual mostra de forma esquemática a estrutura interna da arquitetura de um microprocessador padrão 8086.

Perceba que a estrutura é formada por 14 registradores (**AX**, **BX**, **CX**, **DX**, **CS**, **DS**, **ES**, **SS**, **SI**, **DI**, **SP**, **BP**, **IP**, **F**), cada um com 16 *bits* (numerados de 0 a 15 da direita para a esquerda) divididos em quatro grupos funcionais denominados: registradores gerais (**AX**, **BX**, **CX**, **DX**), registradores de segmento (**CS**, **DS**, **ES**, **SS**), registradores de ponteiros (**SI**, **DI**, **SP**, **BP**, **IP**) e registradores de estado (**F**). A Figura 2.5 mostra um resumo dos registradores encontrados na arquitetura dos microprocessadores 8086/8088.

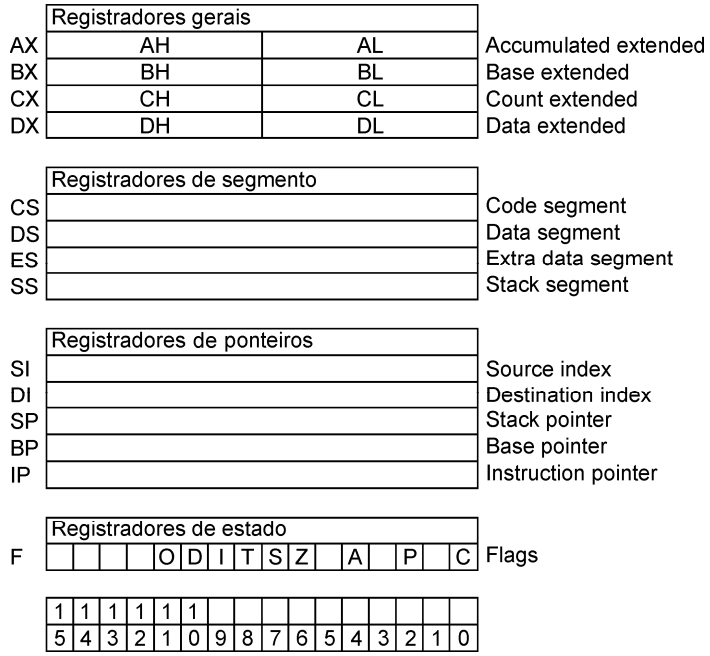

Figura 2.5 - Esquema da arquitetura interna do microprocessador 8086/8088.

Observação

Não se preocupe em entender plenamente tudo o que está exposto neste momento. Os necessários conceitos serão absorvidos durante a aplicação prática de exercícios de aprendizagem ao longo do livro.

2.3.1 - Registradores gerais

Os registradores gerais (*general registers*) **AX** (***A**ccumulator e**X**tended*), **BX** (***B**ase e**X**tended*), **CX** (***C**ounter e**X**tended*) e **DX** (***D**ata e**X**tended*) possuem 16 *bits* de dados, e cada um desses registradores pode ser dividido em duas partes, cada uma com 8 *bits*: parte mais significativa (do *bit* 8 até o *bit* 16), lado esquerdo e parte menos significativa (do *bit* 0 até o *bit* 7), lado direito. Desta forma, obtém-se oito registradores de 8 *bits* cada um, como indicado na Figura 2.6. Os registradores gerais fazem parte do grupo de dados (*data group*) e são utilizados com a finalidade de armazenar dados a serem processados.

Quando divididos, cada registrador geral é tratado com 8 *bits* mais significativos (alto) e 8 *bits* menos significativos (baixo). Desta forma, o registrador de 8 *bits* **AH** (***A**ccumulator **H**igh*) e o registrador de 8 *bits* **AL** (***A**ccumulator **L**ow*) são, respectivamente, divisões do registrador **AX** de 16 *bits*. Para os demais registradores gerais segue a mesma estrutura,

BH-**BL**, **CH**-**CL** e **DH**-**DL** no sentido de determinar registradores de 8 *bits* mais e menos significativos pertencentes, respectivamente, a cada um de seus registradores de 16 *bits* **BX**, **CX** e **DX**. A Figura 2.6 mostra como esta divisão deve ser considerada.

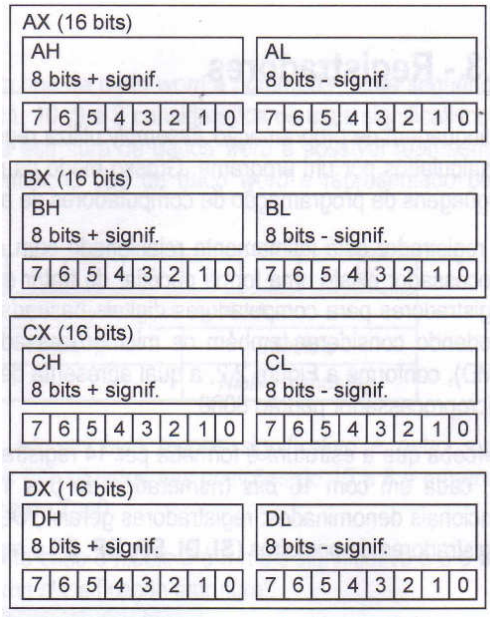

Figura 2.6 - Esquema da divisão interna dos registradores gerais.

O registrador geral **AX** (**AH-AL**) pode ser utilizado em operações aritméticas e lógicas, acessos de portas de entrada e saída, chamadas de interrupções, transferência de dados, entre outras possibilidades. A definição de seu valor pode ser feita sobre todo o registrador **AX** ou sobre os 4 *bits* à direita, ou seja, da sua parte menos significativa, sendo **AL**. Este é o principal registrador usado na manipulação de dados.

O registrador geral **BX** (**BH-BL**) pode ser utilizado como ponteiro para acessar a memória (apontador de endereço) no sentido de obter algum valor de retorno ou mesmo definir valores que serão usados para auxiliar operações aritméticas efetuadas com o registrador **AX**, além de servir de base para instruções que operem com vetores de dados.

O registrador geral **CX** (**CH-CL**) também é usado para receber alguns valores de interrupções, mas sua principal finalidade é servir como contador de laços de repetição e operações de deslocamento (*shift*) registradas normalmente em **CL**. Este registrador é usado em operações de ações repetitivas e iterativas.

O registrador geral **DX** (**DH-DL**) é usado em operações aritméticas (mais precisamente em operações de multiplicação para armazenamento da parte de um produto de 32 *bits* e também em operações de divisão para o armazenamento do resto da divisão), acessos de portas de entrada, acessos a portas de saída, acesso a portas lógicas e em chamadas de interrupções e apontamento de endereços de memória para deslocamentos.

2.3.2 - Registradores de segmento

Os registradores de segmento (*segment registers*) **CS** (***C**ode **S**egment* - segmento de código), **DS** (***D**ata **S**egment* - segmento de dados), **ES** (***E**xtra **S**egment* - segmento extra) e **SS** (***S**tack **S**egment* - segmento de pilha) têm 16 bits e são utilizados para acessar uma determinada área de memória, ou seja, para auxiliar o microprocessador a encontrar o caminho pela memória do computador. Eles não podem ser divididos em registradores de 8 bits.

O registrador de segmento **CS** aponta para uma área de memória que contém o segmento de código de programa que se encontra em execução. A mudança do valor existente nesse registrador de segmento pode resultar em travamento do computador. Quando esse segmento tem seu valor somado ao valor do segmento de deslocamento **IP** (comentado no próximo tópico), gera-se o endereço de 20 *bits* da instrução. O registrador **CS** contém as instruções que são executadas em um programa. Este registrador armazena o endereço inicial do segmento de código.

O registrador de segmento **DS** aponta para a área de memória que geralmente é utilizada para o armazenamento dos dados do programa em execução. A mudança do valor existente nesse registrador de segmento pode ocasionar a obtenção de dados incorretos. Quando utilizado em conjunto com o registrador de deslocamento **IP** (comentado no próximo tópico), permite definir o endereço efetivo de memória onde um dado será lido ou escrito por meio do controle de execução de instruções. Este registrador armazena o endereço inicial do segmento de dados.

O registrador de segmento **ES** é utilizado para determinar um segmento extra de endereço de dados (um novo segmento, ou seja, um segmento *far pointer*) distante da área em que se está operando. Normalmente necessário para acessar, por exemplo, a memória de vídeo. Esse registrador aponta para um segmento de dados do tipo *string* (texto).

Observação

Os registradores de segmento DS e ES são usados em conjunto com os registradores de deslocamento SI e DI, que serão comentados no tópico seguinte.

O registrador de segmento **SS** é utilizado para identificar a área de memória que será usada como **pilha** (*stack*). Desta forma, aponta para o segmento da pilha em uso no sentido de armazenar dados temporários para a execução de um determinado programa. Esse registrador de segmento pode algumas vezes conter o mesmo valor encontrado no registrador de segmento **DS**. A mudança do valor existente nesse registrador de segmento pode trazer resultados imprevisíveis, normalmente relacionados aos dados.

O segmento de pilha caracteriza-se por ser uma área a qual contém dados e endereços de retorno de procedimentos ou sub-rotinas. A pilha é implementada como uma estrutura de dados (**stack**). O registrador de segmento de pilha **SS** armazena o endereço inicial da pilha.

2.3.3 - Registradores de deslocamento

Os registradores de deslocamento (apontamento), também denominados registradores de índice, estão associados ao acesso de uma determinada posição de memória previamente conhecida, com a utilização dos *registradores de segmento*. Os registradores de deslocamento pertencem ao grupo de ponteiros e índices (*pointer and índex group*).

Os registradores de ponteiros são cinco de 16 *bits*. Diferentemente dos registradores gerais, eles não podem ser divididos em registradores de 8 *bits*.

Dos cinco registradores de apontamento, quatro são manipuláveis, sendo **SI** (***S**ource **Índex*** - índice de origem), **DI** (***D**estination **Índex*** - índice de destino), **SP** (***S**tack **P**ointer* - ponteiro de pilha) e **BP** (***B**ase **P**ointer* - ponteiro de base). O registrador de apontamento (deslocamento) **IP** (***I**nstruction **P**ointer* - ponteiro de instrução), conhecido como *apontador da próxima instrução*, possui o valor de deslocamento (*offset*) do código da próxima instrução a ser executada.

O registrador de apontamento **IP** é de uso interno do microprocessador e tem por finalidade guardar o deslocamento (*offset*) da próxima instrução de um programa a ser executada, ou seja, esse registrador possui o valor da distância em *bytes* da próxima instrução a ser executada em relação ao início da instrução que se encontra em execução. O valor existente nesse registrador só pode ser lido. O registrador **IP** associado ao registrador **CS** fornece o endereço completo da instrução atual no segmento de código.

Os registradores de apontamentos **SI** (endereço fonte de operações com cadeias de caracteres) e **DI** (endereço destino de operações com cadeias de caracteres) são utilizados para manipular índices de uma tabela, sendo o registrador de apontamento **SI** usado para a leitura de dados do tipo *string* de uma tabela e o registrador de apontamento **DI** usado para a escrita de dados do tipo *string* em uma tabela. Em especial o registrador de apontamento **DI** também é utilizado na definição de endereçamento distante associado ao registrador de segmento **ES**.

Os registradores de apontamento **SP** (fornece o valor de deslocamento dentro da pilha do programa) e **BP** (usado na referência de parâmetro passado para uma sub-rotina) permitem o acesso à **pilha** de programa (memória para armazenamento de dados). A pilha possibilita armazenar dados em memória, sem utilizar registradores gerais. Uma pilha é um mecanismo que armazena dados de cima para baixo. Imagine uma pilha como uma matriz de uma dimensão. O registrador de apontamento **BP** armazena o endereço da base da pilha, enquanto o registrador de apontamento **SP** armazena o endereço do topo da pilha. O registrador **SP** associado ao registrador **SS** é usado na referência da posição atual dos dados ou endereço dentro da pilha do programa. Já o registrador **BP** combinado com o registrador **SS** fornece a

localização do parâmetro de uma sub-rotina, podendo ainda ser associado aos registradores **DI** e **SI** como registro de base para a definição de um endereçamento especial.

2.3.4 - Registradores de estado (status flags)

Os registradores de estado são considerados os mais importantes meios para a sinalização da efetivação de operações lógicas, aritméticas, manipulação de blocos e interrupções, pois indicam o estado de comportamento do microprocessador quando da execução de alguma instrução da linguagem de programação *Assembly*, pois são responsáveis pela definição das condições a serem utilizadas pelo microprocessador. Os registradores de estado são considerados registradores de controle. A Figura 2.7 apresenta o esquema do registrador de estado (*flag*) para o microprocessador 8086.

O registrador de estado **F** (*flags*) tem 16 *bits* que agrupam um conjunto de *flags* de um 1 *bit* (sendo um por *bit*), e cada *flag* define ou sinaliza um estado de comportamento particular do computador.

Se o valor de cada *bit* estiver sinalizado como 1, indica que o *flag* em questão está setado (acionado), caso esteja sinalizado com o valor 0, significa que o *flag* não está setado (desabilitado).

O *flag* está vinculado ao fato de uma ação ser ou não executada. Se uma determinada ação é executada, o *flag* (a bandeira) é levantado, indicando que a ação ocorreu (valor 1). Quando a bandeira está abaixada, é sinal de que a ação em questão não sofreu nenhum tipo de alteração (valor 0). Os *flags* do registrador de estado são independentes, mas por questões de conveniência são agrupados no mesmo conjunto de registrador de estado.

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

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

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

Figura 2.7 - Esquema do registrador de estado para o microprocessador 8086.

Os *flags* **CF**, **PF**, **AF**, **ZF** e **SF** são de estado; os *flags* **TF**, **IF** e **OF** são de sistema; o *flag* **DF** é de controle.

A Figura 2.7 apresenta o registrador de estado com o conjunto de nove *flags* associados a cada operação executada. Dos nove *flags*, seis são utilizados para indicar os resultados obtidos em operações aritméticas ou lógicas, são estes os registradores de estado (*status flags*) **CF** (***C**arry **F**lag*), **PF** (***P**aity **F**lag*), **AF** (***A**uxiliar **F**lag*), **ZF** (***Z**ero **F**lag*), **SF** (***S**ignal **F**lag*) e **OF** (***O**verflow **F**lag*). E os outros três *flags* são utilizados para as tarefas de controle (*control flags*): **TF** (***T**rap **F**lag*), **IF** (***I**nterrupt **F**lag*) e **DF** (***D**irection **F**lag*). Perceba que os *bits* 1, 3, 5, 12, 13, 14 e 15 do registrador estão indicados em branco (na figura). Internamente não existe como prever seus estados lógicos, se eles estarão com valores "0" ou "1", pois estão reservados propositalmente pela Intel.

O *flag* **CF** (*bit* 0) indica a sinalização da operação de *carry* (operação de vai um) para a próxima posição, após a efetivação de uma operação aritmética de adição ou operação de *carry* (operação de empresta 1) para a próxima posição, após a efetivação de uma operação de subtração que ultrapassar a capacidade de armazenamento daquele valor. Neste caso, será este sinalizado com o valor **1**; se não ocorrer a movimentação de vai um, o valor sinalizado será **0**. Por exemplo, ao se efetuar a soma dos valores 255 e 1, obter-se-á o valor 256. Considerando-se apenas a capacidade de armazenamento de um valor entre 0 e 255, este resultado estará fora da faixa, o que ocasiona a mudança do *flag* **CF** para **1**.

O *flag* **PF** (*bit* 2) indica se a paridade do *byte* inferior do resultado é par ou ímpar após a execução de uma operação aritmética ou lógica resultar ou não um par de *bits* **1s** (uns). Se o valor sinalizado for **1**, será então considerada paridade par; se o valor sinalizado for **0**, será considerada paridade ímpar.

O *flag* **AF** (*bit* 4) indica a sinalização de *carry* (operação de vai-um) do *bit* 3 para o *bit* 4 de uma adição ou com a sinalização de *carry* (operação de empresta-um) do *bit* 4 para o *bit* 3 de uma subtração com valores decimais, representados por sequências numéricas escritas em formato BCD (*Binary Coded Decimal*). Se o valor sinalizado for **1**, indica que ocorreu a operação de vai um; se o valor sinalizado for **0**, indica que ocorreu a operação de empresta um.

O *flag* **ZF** (*bit* 6) indica se o resultado após uma operação aritmética ou lógica. Em uma operação aritmética será igual a zero quando sinalizado pelo valor **1** ou diferente de zero quando sinalizado pelo valor **0**. Pode ser usada para a obtenção de resultado de comparação (ação lógica), sendo sinalizado pelo valor **1** quando os valores comparados forem iguais e **0** quando forem diferentes.

O *flag* **SF** (*bit* 7) mostra a obtenção de um resultado negativo ou positivo após uma operação aritmética de complemento 2, não ocorrendo estouro - *overflow*. Se o resultado for positivo, será sinalizado como valor **0**; caso seja negativo, será sinalizado com valor **1**.

O *flag* **TF** (*bit* 8) possibilita uma interrupção especial após executar uma única instrução, com a finalidade de acompanhar passo a passo a execução individual das instruções de um determinado programa em operações de *debug*. Quando sinalizado com valor **1**, indica a execução da ação passo a passo da própria instrução em foco e após essa ação ocorre uma interrupção e seu valor tornar-se-á **0**. Se o valor estiver em **0**, ocorre o contrário.

O *flag* **IF** (*bit* 9) habilita ou desabilita o reconhecimento à chamada de interrupções por meio da instrução **INT**. Se estiver uma ocorrência de interrupção habilitada, ele será sinalizado com o valor **1**; caso contrário, será sinalizado o valor **0**.

O *flag* **DF** (*bit* 10) aponta a direção das operações de manipulação de blocos da direita para a esquerda ou vice-versa em sequências de caracteres (*strings*). Quando estiver sinalizado com **0**, indica que os registradores de índice serão incrementados; caso esteja sinalizado com valor **1**, os registradores de índice são decrementados.

O *flag* **OF** (*bit* 11) indica a obtenção de um valor muito grande após uma operação aritmética ou lógica, estouro de capacidade quando um número positivo for muito grande ou um número negativo for muito pequeno para ser processado, ou seja, fora da faixa de valores permitida para a operação do microprocessador. Quando o estouro ocorrer, ele será sinalizado com valor **1**; permanecendo sinalizado o valor **0**, não ocorreu estouro na operação.

2.4 - Interrupções

A interrupção está associada à ação executada por um computador na realização de tarefas e a capacidade de interrompê-las a qualquer momento, para então realizar outras tarefas solicitadas e retornar àquilo que estava fazendo.

O processo de interrupção é de fundamental importância para o funcionamento adequado de um computador. É por meio das interrupções que se torna possível controlar vários dispositivos associados ao computador

O mecanismo de funcionamento ocorre pela comunicação entre o dispositivo externo (vídeo, teclado, mouse, unidades de disco, impressora, *scanner*, erros de execução de programa etc.) e o microprocessador.

Imagine o microprocessador trabalhando em uma determinada tarefa, quando é interrompido por um dispositivo externo que solicita autorização de ação. Nesse instante, o microprocessador para o que está fazendo e aceita a comunicação do dispositivo externo. Ao final da ação do dispositivo externo o microprocessador continua sua ação de trabalho a partir do ponto em que houve a interrupção. Esta característica torna os computadores máquinas eficientes e eficazes.

Graças à interrupção torna-se possível, por exemplo, utilizar um teclado para a entrada de dados quando da execução de um programa de cadastro. Para que uma interrupção seja acionada, usa-se a instrução **INT** seguida de um valor hexadecimal situado na faixa de valores entre **00h** e **FFh** para alterar o comportamento de execução do microcomputador, totalizando 256 interrupções possíveis. No entanto, nem toda interrupção pode ser usada, principalmente as interrupções de hardware.

No microprocessador 8086 os primeiros 1024 *bytes* da memória são reservados para a definição da tabela de interrupções (endereço físico de 0 a 1023). Essa tabela é formada por um conjunto de vetores que são indexados nos registradores **CS** e **IP**. Cada vetor de interrupção ocupa um total de 4 *bytes*, sendo 2 *bytes* usados no registrador **CS** e 2 *bytes* usados no registrador **IP**. A tabela de interrupções é conhecida como ISR (*Interrupt Service Routine*) e possui os endereços das funções de cada uma das interrupções existentes.

As interrupções que podem ser executadas em um computador padrão IBM-PC são divididas em três categorias, a saber:

- **Interrupção por hardware** - possibilita acionar periféricos externos, como unidades de disco, teclado, impressora, porta de comunicação etc. Esse tipo de interrupção conecta-se (comunica-se) ao microprocessador pelas linhas de comunicação de **IRQ** (*Interrupt ReQuest* - pedido de interrupção) que são controladas pelas rotinas de sistema operacional ou pelas rotinas da ROM-BIOS (BIOS é o nome do primeiro programa executado no computador após este ser acionado, significando *Basic Input Output System*, ou seja, sistema básico de entrada e saída, estando armazenado na memória ROM, daí chamá-lo de ROM-BIOS). Esse tipo de interrupção está na faixa hexadecimal de valores de **08h** a **0Fh**. É possível usar até oito dispositivos externos[2]. As interrupções de hardware são gerenciadas pelo próprio computador. Assim sendo, não é possível modificar seu estado de comportamento.
- **Interrupção por exceção** - possibilita fazer o controle condicional interno do microprocessador, retornando algum tipo de erro. Um exemplo típico é a tentativa de um programa dividir um determinado valor por zero. Esse tipo de interrupção está na faixa de **00h** a **07h**.
- **Interrupção por software** - efetua o acionamento de determinadas interrupções desejadas por intermédio de um programa em execução, podendo interferir na ação de algum periférico conectado ao microcomputador.

Como exemplos de interrupções por hardware para o microprocessador 8086 pode-se indicar as seguintes: 08h (associada ao IRQ0 opera o *chip* temporizador do sistema), 09h (associada ao IRQ1 opera o teclado), 0Ah (associada ao IRQ2 é uma interrupção reservada), 0Bh (associada ao IRQ3 opera as portas de comunicação COM2), 0Ch (associada ao IRQ4 opera as portas de comunicação COM1), 0Dh (associada ao IRQ5 opera uma unidade de disco rígido), 0Eh (associada ao IRQ6 opera unidade de discos flexíveis) e 0Fh (associada ao IRQ7 opera a porta LPT1 para impressora do tipo paralela).

Como exemplos de interrupções por exceção para o microprocessador 8086 pode-se indicar as seguintes: 00h (falha de divisão quando o divisor é igual a zero), 01h (execução passo a passo num processo de *debug* quando *flag* TF é setado), 02h (interrupção reservada para a condição de operação para NMI - *Non-Maskable-Interrupt*), 03h (ocorre na definição da execução de um ponto de parada - *breakpoint* sobre a execução da instrução **INT 3**) e 04h (ocorre quando há estouro registrado no *flag* OF quando da execução da instrução **INT O**).

Como exemplos de interrupções de software para o microprocessador 8086 mais utilizadas pode-se indicar as interrupções de ROM-BIOS: 10h (escrita de caractere no monitor de vídeo), 13h (operações de leitura e escrita em disco), 14h (comunicação com porta serial), 16h (leitura de caractere no teclado), 17h (operações com impressora), 19h (ação de *reset*), 1Ah (retorna a hora do sistema), 20h (finaliza a operação do sistema), 21h (operações de leitura e escrita nos periféricos padrão conectados ao sistema) e 33h (utilização do mouse).

Durante a apresentação dos exemplos de programas dos próximos capítulos serão automaticamente utilizadas algumas das interrupções de software mais comuns das 256 interrupções existentes, pois são as aceitas para uso pelo programa **emu8086**.

2.5 - Segmentos e deslocamentos

A linguagem de programação *Assembly x86* utiliza as características operacionais do microprocessador 8086 (ou 8088) para controlar as funções de trabalho de um computador. Entre as funções operacionais de trabalho está o uso da memória.

A memória de um microcomputador da família IBM-PC se assemelha a uma tabela de dados, cujas linhas são os **segmentos** e as colunas são os **deslocamentos** (*offset*) com a capacidade de endereçar uma memória de até 1 MB, dividida em 16 blocos (numerados de 0 a F em hexadecimal) com 64 KBytes (65.536 caracteres) cada bloco. Assim sendo, cada bloco de memória é denominado **segmento** de memória com 64KB cada segmento.

Os 16 segmentos de memória são endereçados de **00000h** (**0000 0000h**) até **FFFFFh** (**000F FFFFh**), sendo **00000h** o valor do primeiro segmento e **FFFFFh** o valor do último segmento, que equivale a 1 MB de espaço em memória. Um endereço no formato **0xxxxh** faz referência ao segmento **0h**, assim como o endereço **1xxxxh** faz referência ao seg-

[2] *Levando-se em consideração o uso de um microprocessador Intel 8086.*

mento **1h** e assim por diante até o segmento **Fxxxxh**. A indicação xxxx após o valor do segmento é o complemento da formação do endereço físico (que é formado ao todo por cinco dígitos) do segmento de memória em uso.

Cada segmento é dividido em deslocamentos que variam de **0yyyyh** até **Fyyyyh**, e cada posição de deslocamento tem um total de 16 *bits* (manipulados pelos registradores de segmento). O segmento de memória é de fato dividido em duas partes, sendo a primeira parte formada por um segmento de 16 *bits* e uma segunda parte formada por um deslocamento de 4 *bits*. Isto posto, ocorre o fato de um endereçamento de memória ocupar a quantidade de 20 *bits* na memória, ou seja, a posição de memória a ser especificada é definida pelo valor do segmento e pelo seu deslocamento em relação ao início do segmento.

O deslocamento para a representação do endereço de memória a ser usado com os programas é representado pela notação **xxxxh:yyyyh** (*segmento*:*deslocamento*), sendo **xxxxh** o endereço de segmento e **yyyyh** o endereço da posição de deslocamento desejado, e está situado na parte convencional da memória principal, ou seja, entre a posição de memória 0 Kb até 640 Kb.

Perceba que o uso de segmentos e deslocamento são uma forma de se fazer acesso as posições de memória. Para exemplificar o exposto da forma mais simples possível de um ponto de vista apenas didático, tome a Figura 2.8 como exemplo, a qual representa, em semelhança, como ocorre o armazenamento de dados. Para tanto, considere como dados armazenados os seguintes textos:

APRENDER ASSEMBLY REQUER ATENÇÃO E DEDICAÇÃO.

EDITORA ÉRICA LTDA.

QUEM DESCOBRIU O BRASIL FOI PEDRO ÁLVARES CABRAL.

D. PEDRO I PROCLAMOU A INDEPENDÊNCIA.

- Observe que o armazenamento dos dados na memória ocorre de forma contígua, um dado após o outro. Cada dado em particular ocupa uma posição de **deslocamento** (coluna) em relação a uma posição de **segmento** (linha). Assim sendo, tem-se como nomenclatura de posicionamento de dados na tabela a indicação **segmento:deslocamento**.
- Perceba que os dados indicados **EDITORA ÉRICA LTDA.** estão na memória a partir da posição de segmento **2h** e deslocamento **Eh**, ou seja, endereço **2h:Eh** e se estende até a posição de segmento **4h** e deslocamento **0h**, ou seja, endereço **4h:0h** (segmento:deslocamento). Efetivamente para saber o endereço de posicionamento de um dado na memória, basta multiplicar o valor do segmento desejado por **16d** ou **10h** e somar o valor do deslocamento.
- Tomando por base os dados **EDITORA ÉRICA LTDA.**, o endereço de inicialização dos dados é **2h * 10h + Eh**, sendo assim a posição **2Eh** em que a sequência de dados se inicia.
- Posteriormente no capítulo nove este assunto será tratado com maiores detalhes. Neste momento está sendo apresentado a título de ilustração e de formação da base conceitual para o entendimento da linguagem de programação *Assembly* apenas um mero exemplo.

Mais detalhes sobre segmentos, deslocamentos e outras informações são estendidas e apresentadas no capítulo 10.

2.6 - Endereçamento de memória

A memória de um computador padrão IBM-PC tem como função primária a tarefa de armazenar dados e executar programas. Para que isso ocorra, essa memória é formada por um grande conjunto de células de armazenamento, e cada célula é formada por um conjunto de 8 *bits*, podendo assumir um dos 256 significados existentes. Cada célula que compõe a memória tem um endereço absoluto de sua posição. Cada posição é considerada de modo individual com um valor numérico que representa seu endereço de posicionamento.

	Deslocamento															
Segmento	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	A	P	R	E	N	D	E	R		A	S	S	E	M	B	L
1	Y		R	E	Q	U	E	R		A	T	E	N	Ç	Ã	O
2		E		D	E	D	I	C	A	Ç	Ã	O	.		E	D
3	I	T	O	R	A		É	R	I	C	A		L	T	D	A
4	.		Q	U	E	M		D	E	S	C	O	B	R	I	U
5		O		B	R	A	S	I	L		F	O	I		P	E
6	D	R	O		A	L	V	A	R	E	S		C	A	B	R
7	A	L	.		D	.		P	E	D	R	O		I		P
8	R	O	C	L	A	M	O	U		A		I	N	D	E	P
9	E	N	D	Ê	N	C	I	A	.							
A																
B																
C																
D																
E																
F																

Figura 2.8 - Exemplo simplificado da forma de composição da memória.

Esta obra se baseia no uso da estrutura de um microprocessador Intel 8086 ou 8088. O espaço máximo de memória para uso por programas e dados está na casa de 640 KBytes, sendo permitido o endereçamento máximo de 1 Mb. Do espaço total de 1 Mb de memória, 384 Kb são reservados para uso interno do sistema e não se pode usar esse espaço; sobram 640 KBytes que são divididos para guardar a tabela de interrupções que ocupa 1 Kb, os dados utilizados pelo programa BIOS que ocupa 1,5 Kb, sobrando um total de 637.5 Kb para serem usados pelo DOS (*Disk Operation System* - sistema operacional em disco) e também pelos outros programas que são utilizados, bem como o espaço para trabalhar com os dados desses programas.

O modelo de microprocessador 80286 conseguia endereçar no máximo 1 Gb de memória. A partir dos modelos de microprocessadores com 32 *bits* (acima do modelo 80386) já foi possível fazer uso máximo de 4 Gb de endereçamento de memória e para os microprocessadores de 64 *bits* é possível usar 128 Gb de endereçamento de memória. No entanto, em todos os modelos de microprocessadores padrão Intel existentes a estrutura de endereçamento de memória de 1 Mb do modelo 8086 ou 8088 é mantida por questões de compatibilidade e pelo fato de o mais moderno microprocessador da Intel ou da AMD ser originado do modelo 8086 e baseado nele, sendo este um bom começo para o estudo de programação da linguagem *Assembly*.

A Figura 2.9 representa graficamente a estrutura de endereçamento de memória de 1 MB típica usada pelos microprocessadores Intel 8086/8088 e também por seus sucessores.

A região de memória convencional vai de 0 KByte até 640 KBytes. Essa região é utilizada para armazenar a tabela de interrupções, os dados do programa BIOS e também pelo DOS, além de armazenar os dados de programas de uso geral, bem como os próprios programas. Assim sendo, os endereços de memória são:

- A partir da posição 0 até a posição 1.024 situa-se a faixa de memória reservada para o armazenamento da tabela de interrupções (pode ser controlado o acesso ao vídeo e teclado). Essa faixa equivale a 1 KByte de memória.
- A partir da posição 1.024 até a posição 1.536 situa-se a faixa de memória reservada para o armaze-namento dos dados do programa BIOS. Essa faixa equivale a 0,5 KByte (512 bytes) de memória.
- A partir da posição 1.536 até a posição 655.360 situa-se a faixa de memória reservada para o armazenamento do sistema operacional (DOS) e também de qualquer programa a ser utilizado. Essa faixa equivale a 638,5 KBytes de memória livre.

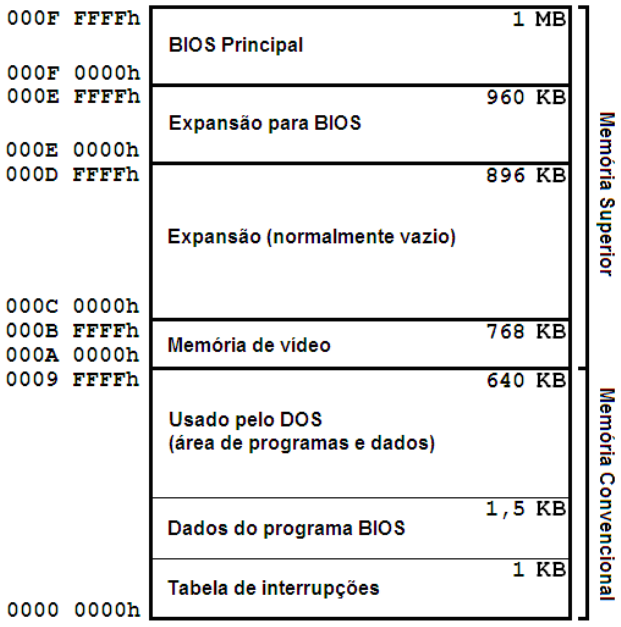

Figura 2.9 - Endereçamento de memória.

A área de memória livre para trabalho é menor que a marca de 640 KBytes, pois 1,5 KByte dos 640 KBytes de memória é reservado para a tabela de interrupções e para os dados do BIOS. Além da perda desse espaço, ocorre ainda a utilização de parte dele para arquivos de controle do sistema operacional como os arquivos IO.SYS, MSDOS.SYS e COMMAND.COM.

Pelo fato de os programas BIOS do sistema e do vídeo estarem gravados em memória ROM, é necessário ter suas informações temporárias armazenadas em uma área de memória RAM. Desta forma, é possível proceder com algum tipo de alteração nessas informações.

Acima da marca dos 640 KBytes encontra-se a área de memória de vídeo que ocupa um total de 128 Kb da parte de memória superior. Essa área é utilizada exclusivamente pelas placas de vídeo para que os programas executados possam "enxergar" a capacidade de memória que a placa de vídeo conectada no computador possui.

Entre 768 Kb e 896 Kb encontra-se a área de expansão com um total de 128 Kb. Normalmente essa parte da memória encontra-se vazia e quando utilizada, é por meio de programas de gerenciamento de memória no sentido de deixar mais espaço para a memória convencional.

No espaço de 64 Kb compreendido na faixa de 896 Kb até 960 kb encontra-se a área de expansão para o BIOS que complementa o espaço da área de dados do programa BIOS na memória convencional.

Por último, os 64 Kb acima de 960 Kb são usados pelo sistema para armazenar o programa BIOS propriamente dito. O programa BIOS encontra-se gravado na memória ROM.

A parte de memória convencional é usada para a execução do DOS e dos programas, gerenciamento dos dados utilizados pelos programas executados, armazenamento dos dados da BIOS e da tabela de interrupções. Já a área de memória superior é utilizada pelo próprio computador e não é diretamente acessível.

2.7 - A linguagem Assembly 8086

A denominação da linguagem de programação de computadores *Assembly* para computadores IBM-PC é feita normalmente com algumas siglas de identificação, tais como *ASM86*, *x86*, *ASM8086*, entre outras. Esta denominação refere-se ao uso da linguagem de programação de computadores *Assembly* para o microprocessador Intel ou equivalentes de mercado, como os microprocessadores fornecidos pela empresa AMD.

Os demais lançamentos de microprocessadores da família Intel, tais como 80186, 80286, 80386, 80486 e também a série Pentium, possuem basicamente o mesmo conjunto de instruções existentes no padrão de microprocessador 8086

e 8088, além de cada um possuir em sua evolução novas instruções e recursos. Os recursos desses outros microprocessadores não serão abordados nesta obra. Este trabalho limita-se somente às instruções básicas e características da linguagem *Assembly* para o microprocessador 8086/8088. A linguagem de programação *Assembly 8086* (ou *ASM86*, *x86)* possui 124 comandos (mnemônicos) diferentes destinadas ao controle do microprocessador. Os demais processadores da família Intel possuem a cada série um número maior de novas instruções. Para quem está começando a estudar a linguagem *Assembly*, é aconselhável primeiro conhecer basicamente as instruções para o microprocessador modelo 8086 ou modelo 8088. O *Assembly* do modelo 8086 é o mesmo do modelo 8088, tanto que se faz normalmente referência aos modelos com a nomenclatura 8086/8088, doravante utilizada nesta obra.

Os comandos da linguagem *Assembly* estão divididas em seis grupos funcionais, a saber:

- **Transferência de dados** - instruções destinadas à movimentação de dados. Os dados podem ser movimentados entre registradores, entre registradores e posições de memória e entre registradores e unidades de entrada e de saída.
- **Aritméticas** - instruções destinadas aos cálculos matemáticos básicos, como adição, subtração, multiplicação e divisão.
- **Manipulação de bits** - instruções que fazem o deslocamento de *bits* em um registrador ou posição de memória. As funções de operações lógicas de conjunção, disjunção e negação são incluídas nesse grupo.
- **Manipulação de strings** - elas fazem o controle (comparação, análise e movimentação) de grupos de sequências de caracteres.
- **Controle de programa** - instruções que controlam a execução do código de programa. O controle pode ser uma execução sequencial, com laços (*loopings*) e com sub-rotinas ou subprogramas. As instruções podem manipular as interrupções de um programa em execução. A ação de interromper um programa durante sua execução pode ocorrer por vários motivos, como, por exemplo, aceitar uma entrada de dados via teclado. Para que o dado possa ser digitado, é necessário fazer uma interrupção no programa que após aceitar o dado continua sua execução.
- **Controle do microprocessador** - instruções que possibilitam o acesso dos registradores de controle do microprocessador, com o objetivo de mudar seu estado de comportamento.

A linguagem *Assembly* como qualquer outra linguagem de programação manipula na memória dados. Os tipos de dados reconhecidos pela linguagem *Assembly 8086* são: constantes, variáveis e rótulos. Segundo Weber (2004, p. 240-241) os tipos de dados da linguagem *Assembly* operam da seguinte forma:

- **Constantes** – refere-se a identificação de um rótulo que será associado a determinado valor numérico sem o uso de algum atributo;
- **Variáveis** – refere-se a objetos manipuláveis usados como operandos para as instruções de manipulação de dados;
- **Rótulos** – refere-se a definição de nomes usados na elaboração de sub-rotinas e definição de pontos de salto a serem executados por instruções de saltos condicionais e incondicionais.

Uma variável ou um rótulo em *Assembly* podem ser associados a um segmento de memória, a um deslocamento dentro de certo segmento da memória, a definição de um tipo de dado (*byte*, *word*, entre outros) associado a uma variável ou a definição de acesso **near** ou **far** quando se usa definição de sub-rotinas.

2.8 - Os modos 32 e 64 bits

Apesar de não ser o foco central deste trabalho a linguagem *Assembly* para processadores que operam em 32 ou 64 *bits*, é oportuno comentar rapidamente as diferenças encontradas nesses microprocessadores no que tange à sua estrutura interna de registradores em relação ao modelo de 16 *bits*. No modo de operação 32 *bits*, os registradores utilizados em memória possuem o dobro da capacidade dos registradores de 16 *bits*, consequentemente os registradores de 64 *bits* possuem como capacidade o quádruplo dessa potência.

Na estrutura 32 *bits*, a maioria dos registradores é acrescida do identificador de prefixo **E** (de *extended*) para sua representação, destacando-se **EAX** [**AX** (**AH-AL**)], **EBX** [**BX** (**BH-BL**)], **ECX** [**CX** (**CH-CL**)], **EDX** [**DX** (**DH-DL**)], **ESI** [**SI**], **EDI** [**DI**], **EBP** [**BP**], **ESP** [**SP**], **EIP** [**IP**] e registradores de estado **EF** [**F**]. Os registradores de segmento **CS**, **DS**, **ES** e **SS** permanecem com a estrutura de 16 *bits*, acrescendo-se a eles **FS** e **GS** como registradores de segmentos extras. Nos registradores de estado **EF** de 32 *bits*, mantêm-se os mesmos registradores existentes no modo de 16 *bits*, mas são acrescidos os registradores de estado **IOPL** (décimo segundo *bit*), **NT** (décimo terceiro *bit*), **R** (décimo sexto *bit*), **VM**

(décimo sétimo *bit*), **AC** (décimo oitavo *bit*), **VI** (décimo nono *bit*), **VIP** (vigésimo *bit*) e **ID** (vigésimo primeiro *bit*). Os *bits* 1, 3, 5, 14, 15 e de 22 até 31 não são usados.

Na estrutura de 64 *bits*, a maioria dos registradores é acrescida do identificador de prefixo **R** (de *rex*) para sua representação, destacando-se os registradores **RAX** {**EAX** [**AX** (**AH-AL**)]} como **R0**, **RBX** {**EBX** [**BX** (**BH-BL**)]} como **R1**, **RCX** {**ECX** [**CX** (**CH-CL**)]} como **R2**, **RDX** {**EDX** [**DX** (**DH-DL**)]} como **R3**, **RBP** {**EBP** [**BP**]} como **R4**, **RSI** {**ESI** [**SI**]} como **R5**, **RDI** {**EDI** [**DI**]} como **R6**, **RSP** {**ESP** [**SP**]} como **R7**, além dos registradores reservados **R8**, **R9**, **R10**, **R11**, **R12**, **R13**, **R14** e **R15**, o registrador de ponteiro **RIP** {**EIP** [**IP**]} e registradores de estado **RF** {**EF** [**F**]}. Os registradores de segmento **CS**, **DS**, **ES**, **SS**, **FS** e **GS** permanecem com a estrutura de 16 *bits*. Nos registradores de estado **RF** de 64 *bits*, mantêm-se os mesmos registradores encontrados no modo de 32 *bits*, acrescentando-se a sequência dos bits de 32 até 63 que estão reservados.

A diferença encontrada nos processadores de 16, 32 e 64 *bits* reside na quantidade de dados e também de instruções que o processador pode executar por vez. Um processador de 16 *bits* consegue manipular um valor numérico inteiro de até 65.536 (2^{16}) numa única operação. Caso necessite trabalhar com um valor maior, será preciso alocar mais uma porção de 16 *bits* da memória. Um processador de 32 *bits* consegue de uma única vez manipular um valor inteiro de até 4.294.967.296 (2^{32}). Em relação a um processador de 64 *bits*, esse valor.