

ANTONIO AUGUSTO DE SOUZA BRITO

HARDWARE & SOFTWARE

MANUAL DO PROGRAMADOR



ÉRICA

**Manual do
Programador PC
Hardware & Software**

**Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)**

Brito, Antonio Augusto Souza, 1953-

**Manual do programador PC : hardware & software /
Antonio Augusto Souza Brito. -- São Paulo : Érica,
1990.**

Bibliografia.

**1. IBM Computador Pessoal 2. IBM Computador
Pessoal - Programação I. Título.**

CDD-001.64

-001.642

90-0554

Índices para catálogo sistemático:

- 1. IBM PC : Computador Pessoal : Processamento de
dados 001.64**
- 2. IBM PC : Computador Pessoal : Programação : Pro-
cessamento de dados 001.642**

Autor:

Bacharel (1974) e Doutor em Física (1984) pelo Instituto de Física da USP. Especialista em sistema de aquisição de dados baseados em microprocessador. Pesquisador do Laboratório de Eletrônica e Dispositivos da Faculdade de Engenharia Elétrica da UNICAMP.

ANTONIO AUGUSTO SOUZA BRITO

**Manual
do
Programador PC
Hardware & Software**

Ano: 1994 93 92 91 90

Edição: 10 9 8 7 6 5 4 3 2 1

Livros Érica Editora Ltda.

TODOS OS DIREITOS RESERVADOS. Proibida a reprodução total ou parcial, por qualquer meio ou processo, especialmente por sistemas gráficos, microfilmicos, fotográficos, reprográficos, fonográficos, videográficos. Vedada a memorização e/ou a recuperação total ou parcial em qualquer sistema de processamento de dados e a inclusão de qualquer parte da obra em qualquer programa juscibernético. Essas proibições aplicam-se também às características gráficas da obra e à sua editoração. A violação dos direitos autorais é punível como crime (art. 184 e parágrafos, do Código Penal, cf. Lei n.º 6.895, de 17.12.80) com pena de prisão e multa, conjuntamente com busca e apreensão e indenizações diversas (artigos 122, 123, 124, 126, da Lei n.º 5.988, de 14.12.73, Lei dos Direitos Autorais).

LIVROS ÉRICA EDITORA LTDA

Rua Jarinu, 594 - Tatuapé - São Paulo

Fone. 294-8686 - C.G.C. 50.268 838/0001-39

Caixa Postal 15 617

Dedicatória:

À Alba, pelo incentivo e compreensão nas ausências.

Ao Bruno e à Maíra, pelos sorrisos e alegria de viver

A Dona Ruth, emérita professora e educadora.

AGRADECIMENTOS

À Coordenação do Laboratório de Eletrônica e Dispositivos da UNICAMP por oferecer as facilidades e os meios.
Ao Eng. Luiz Otávio Ferreira pelo estímulo.
Ao Daniel Roseno, programador dos bons, a quem respeito e admiro.

PREFÁCIO

Este livro foi escrito para o técnico, o engenheiro, o profissional de informática e o hobbista interessado em explorar a eletrônica e os recursos do IBM Personal Computer, aqui denominado simplesmente de PC. Se você acredita que o microcomputador não é uma caixa preta que executa programas, porém um poderoso instrumento para ser interfaceado com o mundo real, então estamos no mesmo caminho.

O PC desde seu aparecimento em 1981, definiu um padrão para microcomputadores pessoais. Sua arquitetura, baseada num microprocessador de 16 bits, é um projeto aberto que possibilita a conexão de interfaces e dispositivos de modo simples. Nosso objetivo é fornecer as informações suficientes de modo a possibilitar a qualquer leitor com conhecimentos básicos de eletrônica e computação, projetar e conectar seus próprios dispositivos ao PC.

Ao desenvolvermos um curso a respeito do Assembly do PC para instrumentação e controle, tivemos que garimpar em diversos livros e revistas especializadas. A maior parte das publicações são especializadas em hardware ou em software. Porém para compreender as características eletrônicas do PC e as potencialidades de seu interfaceamento com o mundo real, é necessário o domínio em conjunto de ambas as partes. Fruto desta necessidade, procuramos reunir, num só volume, um material capaz de ser lido tanto pelos especialistas de eletrônica bem como os de computação. Discutimos, sempre que possível, os aspectos de hardware e software simultaneamente.

A arquitetura do PC e a eletrônica envolvida em sua operação é descrita na Parte I, Caps. 1 a 5. Esse material pode ser utilizado como texto em cursos avançados de circuitos lógicos ou arquitetura de microcomputador. Os recursos existentes no PC para interfaceamento, em particular as características eletrônicas e mecânicas dos slots de expansão, são descritos nesta parte.

O PC possui um poderoso conjunto de programas, localizados no **ROM-BIOS** e no sistema operacional **DOS**, que possibilita a interface entre a placa principal e os periféricos. Na Parte II, Caps. 6 a 11, utilizamos o programa utilitário do **DOS-DEBUG** para montar pequenos programas em linguagem Assembly. Através desses programas pode-se perceber como um conjunto de circuitos integrados está interligado dentro do próprio PC, tais como o relógio e contadores internos, o controlador de interrupção e a porta paralela. Uma série de programas exemplos, em linguagens Assembly e C são

comentados, e alguns exercícios são propostos. Parte desse material pode ser utilizado num curso de introdução ao Assembly 8086/8088. O desenvolvimento desses programas pode ser considerado exercícios de aulas práticas, onde o computador é o próprio laboratório. Todos os programas apresentados foram desenvolvidos pelo autor e testados em microcomputador padrão PC-XT nacional. Os programas fontes e os exercícios resolvidos estão num disco 5.1/4" que o leitor poderá solicitar diretamente ao autor.

Tratar as áreas de software e hardware em conjunto é um desafio. Apesar disto espero que esse material possa ser utilizado pelo leitor como uma base para futuros saltos, tanto na área de computação como de eletrônica em microcomputadores PC.

DISQUETE OPCIONAL

Existem vários programas exemplos neste livro. Alguns em linguagem Assembly 8088 que podem ser montados com o utilitário DOS DEBUG. Dois outros programas em C com um programa cabeçalho padrão para uso das Interrupções BIOS e DOS, de modo natural, podem ser compilados em Turbo C 2.0. Todas as rotinas apresentadas podem ser digitadas e testadas num IBM-PC com um mínimo de 256K de RAM e um disk-drive na versão DOS 2.0 ou posterior. Porém, se você quiser poupar tempo de teste e digitação, é possível obter diretamente com o autor um disco contendo todos os programas fontes e executáveis. Em adição, este disquete contém o código fonte comentado de um programa residente, escrito em Assembly, mostrando um relógio em tempo real no canto superior do display.

Para obter o disquete escreva para:

Antonio A.S.Brito
R.Maria Tereza Dias da Silva, 484
Cidade Universitária
13083 Campinas SP

SUMÁRIO

PARTE I - HARDWARE

A FAMÍLIA PC

1.1 Um Padrão em Microcomputador Pessoal	1
1.2 A Configuração do Sistema	4
1.3 Periféricos e Acessórios	11

ARQUITETURA DO PC

2.1 Introdução	12
2.2 O Microprocessador 8088	14
2.3 O Circuito Temporizador	15
2.4 O "bus" do Sistema	16
2.5 O Mapeamento de Memória	17
2.6 O Contador/Temporizador do Sistema	22
2.7 Acesso Direto à Memória - DMA	24
2.8 As Interrupções do Sistema	24
2.9 Interface com o Mundo Exterior	28

MICROPROCESSADOR 8088

3.1 Introdução	31
3.2 Arquitetura do 8088.	32
3.3 Endereçamento através da CPU	37
3.4 Descrição funcional do 8088	40

BUS DO SISTEMA E SLOT

4.1 Caracterização do "slot"	43
4.2 Sinais de I/O e Memória	47
4.3 Mapeamento da área de I/O	51

INTERRUPÇÃO POR HARDWARE

5.1 Ciclo de interrupção.	53
5.2 Controlador de Interrupção por hardware (8259)	56

PARTE II - SOFTWARE

LINGUAGENS DE PROGRAMAÇÃO

6.1 Introdução	61
6.2 Linguagem de Máquina	61
6.3 Linguagem ASSEMBLY 8088	67
6.4 Arquivos tipo *.COM e *.EXE	68
6.5 Linguagens de Alto Nível	69

REVELANDO O INTERIOR DO PC

7.1 Introdução	71
7.2 Usando o DEBUG	71
7.3 Registradores e Aritmética	74
7.4 Qual é a marca registrada do seu PC?	77
7.5 Reescrevendo a marca registrada	79
7.6 Escrevendo diretamente na memória de vídeo	80
7.7 Criando um programa executável tipo *.COM	84

O CÓDIGO DE INSTRUÇÃO ASSEMBLY

8.1 O Conjunto de Instruções	87
8.2 Instruções de Transferência de Dados	88
8.3 Instruções Aritméticas	93
8.4 Instruções de Manipulação de bits	94
8.5 Instruções de transferência de programa	95
8.6 Instruções de Controle	98

INTERRUPÇÕES DO BIOS

9.1 Os serviços do ROM-BIOS	100
9.2 Os serviços de vídeo da ROM-BIOS	105
9.3 Qual é a configuração do seu PC?	107
9.4 Manipulando o Teclado	111
9.5 Programando em C os serviços do ROM-BIOS	115

INTERRUPÇÕES E FUNÇÕES DOS

10.1 Interrupções DOS	120
10.2 As funções DOS	122
10.3 Obtendo informações através do DOS	126
10.4 Os serviços DOS na linguagem C	130

PROGRAMAS APLICATIVOS AVANÇADOS

11.1 Introdução	134
11.2 Fazendo do Teclado um órgão	134
11.3 Alterando uma interrupção de hardware	138
11.4 Programa Termina e Fica Residente	145

APÊNDICES:

A: NOTAÇÃO HEXADECIMAL E CÓDIGO ASCII NO PC	151
B: MANUAL DO DEBUG	160
C: CÓDIGO DE INSTRUÇÕES 8088/8086	165
D: INTERRUPÇÕES BIOS	185
E: AS FUNÇÕES DOS	206
F: PROTÓTIPO DOS PROGRAMAS EM C	231

A FAMÍLIA PC

1.1 Um Padrão em Microcomputador Pessoal

A partir de 1981, uma nova palavra entrou para o vocabulário de milhares de pessoas: IBM-PC ou simplesmente PC. Provavelmente este ano é o marco de uma nova revolução industrial, tal como foi a construção da primeira rede elétrica, em 1881 por Thomas Edison. Nesta década, o computador pessoal passou a fazer parte dos escritórios, fábricas, escolas e residências, afetando o sistema produtivo e os modos de comunicação. Antes, o conceito de computador era uma abstração, uma máquina poderosa controlada por um grupo fechado nas salas de processamento de dados (CPD).

A popularização do computador como instrumento pessoal de trabalho e lazer começou em 1977, quando Steve Wozniak e Steve Jobs projetaram e construíram um microcomputador para uso individual denominado APPLE. O sucesso comercial desta empresa, fundada por universitários, foi tamanho que em 1980 a gigantesca IBM, até então líder na área de grandes e médios computadores, decidiu entrar na faixa de microcomputadores de uso pessoal.

Até então os microcomputadores usavam microprocessadores de 8 bits, tais como o 6502, 8085 e Z-80. O projeto PC foi desenvolvido em torno do 8088 da INTEL, um microprocessador de 16 bits, com um "bus" de dados de 8 bits, diferente do microprocessador 8086. Esta empresa era fabricante de vários componentes suportes para uso em microcomputadores, tais como: 8259 (controlador programável de interrupção), 8255 (interface periférica ou paralela programável), 8253 (relógio de intervalo programável) e outros, projetados originalmente para serem conectados ao microprocessador de 8 bits 8085. O projeto original da IBM utilizou os dispositivos periféricos de 8 bits da INTEL, de modo a desenvolver rapidamente um microcomputador de baixo custo.

O PC incorporou vários dos excelentes recursos de hardware do APPLE: gráficos de alta resolução, slots de expansão, ROM residente documentada e aberta. Além disto possui "bus" de endereçamento de 20 bits, permitindo decodificar 2^{20} endereços

(1.048.576 ou 1 Mb), ao invés do usual limite de 16 bits (64 Kb) dos microcomputadores de 8 bits.

O PC foi projetado e montado num tempo curto. No início de 1982 já era um sucesso comercial. Mais do que a marca da empresa fabricante, seu sucesso reside na qualidade do projeto original e no suporte de software que oferece.

São os programas que dão vida a um computador. As pessoas não necessitam saber um enorme conjunto de instruções para poder usar um computador. Isto deve ser feito por um programa de controle, o sistema operacional. Armazenar e transferir milhares de bits é uma tarefa complexa e exigente.

Nos microcomputadores de 8 bits, o sistema operacional CP/M (Control Program/Microcomputer), desenvolvido por Gary Kildall, parecia definir um padrão no início dos anos 80. A IBM encomendou, à empresa de software Microsoft, o desenvolvimento de um sistema operacional de fácil manuseio, tal como o CP/M, além de outros requisitos, tais como um utilitário capaz de fazer pequenas rotinas em Assembly do 8088, como o programa DEBUG.

O resultado foi o sistema operacional DOS ("Disk Operational System"), conhecido como MS-DOS ("MicroSoft") ou PC-DOS. O DOS fez do PC uma máquina simples de se usar e de alta confiabilidade. Após algumas horas de contacto com o PC, qualquer usuário é capaz de operá-lo facilmente. Descrevemos na Seção 2.5 como o DOS é carregado no sistema, bem como seus recursos podem ser utilizados nos projetos de interfaceamento, ver Capítulo 10. Utilizamos intensamente o DEBUG para o desenvolvimento de pequenos programas em Assembly, ver Capítulo 7 e Apêndice B.

O PC é uma arquitetura aberta, o que possibilita modificar a ROM sem afetar as características do clone original. Isto permitiu o aparecimento de diversos outros fabricantes de equipamentos PC equivalente. A concorrência, a produção em larga escala nas diversas partes do mundo, permite a construção de um equipamento de custo acessível a um grande número de usuários.

A IBM e seus parceiros INTEL e Microsoft não ficaram parados. Ainda nesta década, lançaram diversos modelos de microcomputadores, com circuitos integrados mais rápidos e com maior escala de integração, além de diferentes versões melhoradas do DOS. Porém, como é grande o esforço no desenvolvimento de programas e treinamento de pessoal, os fabricantes têm tido o cuidado em preservar a capacidade de executar um programa da versão antiga nos modelos aperfeiçoados, ver Seção 10.1. O quadro da Tabela 1.1 mostra um resumo das diversas versões da família PC e o sistema operacional indicado na época de lançamento:

sistema	PC	PC-XT	PC/XT286	PC-AT	PS/2-50	PS/2-80
CPU	8088	8088	80286	80286	80286	80386
CLOCK	5 Mhz	5 Mhz	6 Mhz	8 Mhz	10 Mhz	16 Mhz
#RAM	64 K	256 K	256 K	512 K	1 Mb	1 Mb
ROM	40 K	40 K	64 K	64 K	128 K	128 K
slot 8 bits	5	8	2	2	0	0
16 bits	0	0	0	6	4	5
driver						
floppy	1	2	1	1/1.2M	1/1.4M	1/1.4M
hard	-	10Mb	20Mb	20Mb	20Mb	44Mb
&DOS	1.0	2.0	3.2	2.1	3.3	3.3
@ano	1981	1983	1986	1984	1987	1987

Tabela 1.1 Família PC e versões DOS

Legenda

memória RAM na placa principal no ano da introdução.

& versão DOS recomendada no ano da introdução

@ ano do início de produção.

Na Tabela 1.1, omitimos alguns tipos fora de fabricação ou raramente disponíveis no país, tais como: PCJr, PC portátil, modelos PS/2 30 e 60.

Neste livro, vamos nos limitar a discutir o PC-XT ("eXtend Technology") em detalhes. Este modelo é o mais vendido atualmente no país, possui o maior número de unidades instaladas, é o de mais baixo custo e o mais bem documentado. A velocidade de operação nem sempre é um fator decisivo nos processos de interfaceamento, dada a necessidade de sincronismo com dispositivos periféricos mais lentos. Contudo, a capacidade de operar no modo de multitarefas não deve ser desprezada em processos de monitoramento e o uso do microprocessador 80286, utilizado no AT ("Advanced Technology"), pode ser útil em alguns ambientes.

Tendo em vista a grande quantidade de unidades instaladas, o volume de programas existentes, o PC-XT será, por um bom tempo ainda, um padrão de microcomputador pessoal para uso em processo de instrumentação e controle.

1.2 A Configuração do Sistema

O PC possui um módulo básico, a placa do sistema e diversos periféricos a ela conectados. A placa do sistema é vista em detalhes no Cap.2, aqui descreveremos as características dos periféricos. A Figura 1.1 descreve o diagrama do conjunto.

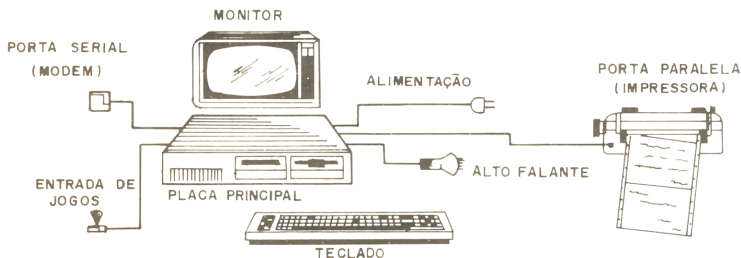


Figura 1.1 Diagrama da configuração básica

1.2.1 Placa Principal

Nesta unidade reside a eletrônica principal. Contém o microprocessador 8088, os 640 Kbytes de memória "RAM" (Random - Access Memory, ou memória volátil), os 40 Kbytes de "ROM" (Read Only Memory, memória de leitura), os conectores de teclado e de disk-driver, a saída de vídeo e do alto-falante. Para interfaceamento a parte mais interessante é o conjunto de 8 slots, dos quais somente 5 estão livres para uso, ver Seção 4.1. No próximo Capítulo descrevemos em detalhes as funções da placa principal em conjunto com a arquitetura do microcomputador

1.2.2 Fonte de Alimentação

A fonte de alimentação fornece 4 níveis de diferença de potencial DC: + 5 V, -5 V, + 12 V, - 12 V. Como a maioria dos circuitos integrados do PC necessitam de alimentação padrão TTL (+5V e 0), o nível + 5 V possui a maior capacidade para fornecimento de corrente. A Tabela 2.1 mostra as características de uma fonte padrão para o PC.

DC(nominal) volts	valor máximo volts	valor mínimo volts	corrente máxima (A)
+ 5	5.25	4.80	7.0
- 5	- 4.60	- 5.50	0.3
+ 12	12.60	11.52	2.0
-12	- 10.92	- 13.20	0.2

Tabela 1.2 Fonte de alimentação padrão do PC

A fonte de alimentação é isolada do conjunto. Esta é conectada à placa principal através de 2 conjuntos de soquetes na parte traseira direita da placa principal com 6 pinos cada. Ela deve ser capaz de alimentar os circuitos da placa principal e dos periféricos, exceto o vídeo. Esses níveis de alimentação estão disponíveis nos slots e podem ser utilizados em projetos de interfaceamento, desde que a corrente drenada pelo slot seja inferior a 500 mA, ver Tabela 4.2, Seção 4.1.

1.2.3 O Conector de Disk Drive

A configuração básica do PC-XT é de 2 drivers de disco flexível ("floppy") de 5.1/4" polegadas e com capacidade de armazenamento de aproximadamente 320 K bytes. Atualmente, a configuração mais comum consiste em acrescentar 1 unidade de disco rígido 20 Mb (mega bytes) ou mais. Os discos de 3.1/2", com alta capacidade de armazenamento (1.44 Mb) vêm sendo utilizados cada vez mais nos computadores mais recentes. Um conjunto de chaves na placa principal identifica os periféricos que estão conectados à placa principal. Após qualquer alteração, é necessário reposicionar as chaves para que os serviços de identificação do BIOS ("Basic Input/ Output System") reconheça os periféricos instalados. Apresentamos na Seção 9.5, o programa ACHAR_B.C capaz de reconhecer os dispositivos instalados. A maioria das linguagens de alto nível possui instruções para identificar os dispositivos de "drivers" instalados. Recomenda-se a utilização destes recursos, como faremos neste livro, em vez de atuar diretamente no sistema de controle dos drivers.

1.2.4 O Adaptador de Vídeo

A conexão entre o monitor e a placa principal é feita através de um adaptador de vídeo conectado a 1 dos 8 slots. A princípio, o adaptador mais utilizado era o adaptador monocromático padrão MDA ("Monochrome Display Adapter) da IBM. Outros fabricantes utilizavam a placa Hércules, capaz de reproduzir bem textos e gráficos. O Adaptador Gráfico Colorido - CGA (Color Graphic Adapter) é o mais comum nos microcomputadores nacionais mais novos. Esta placa pode ser conectada ao monitor

colorido RGB (Red Green Blue), apresentando boa resolução gráfica (320x200 pixels em 4 cores). Porém para programas onde os recursos gráficos coloridos são importantes, as placas EGA (Enhanced Graphics Adapter), com resolução de 640x200 pixels em 16 cores e a VGA (Video Gate Array) permite ao programador usar 256 cores diferentes simultaneamente. Infelizmente, o uso dessas placas no país é bastante limitado. Na Seção 7.6, mostramos como escrever caracteres diretamente na página texto de vídeo.

A maioria dos programas suportes de alto nível, como as versões mais recentes de C ou Pascal, permitem utilizar qualquer uma das placas gráficas além de identificar qual placa está instalada no seu sistema. O programa ACHAR_.B.C na Seção 9.5 procura identificar o adaptador de vídeo instalado num PC genérico.

1.2.5 Teclado

O teclado original do PC é de 83 teclas. Com a introdução do AT o teclado passou a 84 teclas (59 tipos de letras padrões, 15 teclas numéricas e 10 teclas de funções). Atualmente, o modelo mais usado possui 101 teclas, algumas com funções repetidas. O teclado é conectado a placa principal através de um conector tipo 5-DIN.

A cada tecla acionada, um microprocessador de 8 bits 8048 envia uma interrupção de "hardware" (INT 09h, ver Seção 5.2 a respeito das interrupções) ao 8088. Isto permite que o computador possa desempenhar outras operações e só atender ao teclado quando for solicitada uma interrupção de hardware. Neste caso, o vetor de interrupção aponta para uma rotina BIOS que vai identificar qual tecla foi acionada. Porém para compreender os sinais de teclado é necessário converter os símbolos, a letra A por exemplo, em termos de um número. Isto é feito através de um código de 256 caracteres, o código ASCII ("American Standard Code for Information Interchange"). Veja Apêndice A a respeito dos caracteres ASCII e seu uso no PC.

Parte da área de dados do BIOS, (RAM 417h--43Dh) ver Tabela 9.5 Seção 9.3, é usada para monitorar e armazenar ("buffer") provisoriamente as informações de teclado. Existem várias teclas especiais no teclado: Alt, Ctrl, Shift, Esc, e funções F1..F12, que permitem inúmeras combinações, ver Exemplo 9.5 na Seção 9.4. Para resolver a limitação do código ASCII (o código original tinha somente 128 caracteres, estendidos posteriormente para 256), a informação recebida do teclado é armazenada em 2 bytes: o byte principal, de ordem mais baixa, e o auxiliar. Quando o byte principal for nulo uma das teclas especiais foi acionada e o código de rastreamento correspondente deve estar no byte auxiliar.

Em processos de monitoração de fenômenos externos pelo PC pode ser interessante a programação de teclas especiais ("hot keys"), de modo a possibilitar ao operador uma rápida descrição de uma informação, ver Seção 11.2.

1.2.6 Porta Paralela

A expressão paralela significa que 8 bits (1 byte) são enviados simultaneamente. No sistema padrão uma porta paralela é destinada para conexão da impressora. O conector é do tipo fêmea, padrão DB-25, conforme figura:

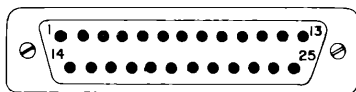


Figura 1.2 Conector DB-25

A tabela 2.1 mostra uma descrição destes pinos, na configuração padrão de escrita na impressora.

pino	direção	tipo de sinal	ativo no estado	função
1	saída	strobe	baixo	dados prontos para envio bit menos significativo
2	saída	D0	alto	
3	saída	D1	alto	bit mais significativo acknowledge; periférico avisa que dados foram recebidos e aguarda novos dados impressora ocupada, não pode receber dados impressora sem papel. impressora selecionada solicita avanço de uma linha após final de linha automaticamente.
4	saída	D2	alto	
5	saída	D3	alto	
6	saída	D4	alto	
7	saída	D5	alto	
8	saída	D6	alto	
9	saída	D7	alto	
10	entrada	ack	baixo	
11	entrada	busy	alto	
12	entrada	Pend	alto	
13	entrada	slct	alto	
14	saída	a.feed	baixo	

Tabela 1.3 Descrição dos pinos da porta paralela

pino	direção	tipo de sinal	ativo no estado	função
15	entrada	error	baixo	dispositivo incapaz de executar tarefa
16	saída	init	baixo	sinal de reset
17	entrada	slct in	out	selecionar impressora
18	nc	terra	-	pinos sem função alguma conectados ao terra
19	nc	terra	-	
20	nc	terra	-	
21	nc	terra	-	
22	nc	terra	-	
23	nc	terra	-	
24	nc	terra	-	
25	nc	terra	-	

Tabela 1.3 Descrição dos pinos da porta paralela (continuação)

A porta paralela pode ser programada como porta de entrada e saída digital e ser utilizada como interface. Isto requer algum cuidado no uso de instruções IN e OUT (instruções em Assembly, ver Secção 8.2) nos endereços 0378h, 0379h, 037Ah, ver Exemplo 9.3 na Secção 9.3. Para dados e endereços usamos a notação hexadecimal com sufixo h. Estes endereços podem ser modificados, entretanto seu valor será sempre armazenado na área de dados do BIOS (408h). Na Secção 9.5, descrevemos o programa ACHAR_B.C capaz de localizar este endereço.

A maioria dos programadores já teve problemas em solicitar um serviço de impressora e o resultado é um branco total ("crash"), o microcomputador fica em pane geral. O programa deve ter solicitado um serviço de interrupção da BIOS INT 17h, (mostramos os serviços de interrupção BIOS no Cap.9) solicitando permissão para imprimir, por exemplo através do strobe, pino 1 no estado baixo, e ficou aguardando uma resposta da impressora, "acknowledge", pino 10 baixo. Devido a um problema qualquer, por exemplo mau contacto ou impressora desligada, a interrupção não é concluída, e todas as outras interrupções, por exemplo o teclado (INT 09h), estão desabilitadas, restando ao operador desligar o microcomputador e perder o texto editado. Felizmente, alguns editores aguardam um pouco e avisam que não é possível imprimir.

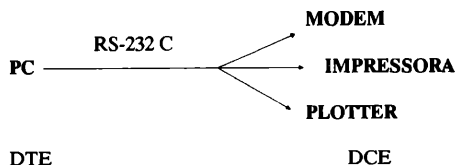
1.2.7 Porta Serial

Diferente da porta paralela, a serial envia 1 bit de cada vez em uma velocidade programável que pode variar de 300 a 9600 bps ("bits per second"). Isto é particularmente útil na transmissão de informações através de um único cabo, como é o caso do telefone. A porta serial do PC opera no padrão RS-232 C, como um dispositivo terminal, DTE, ("Data Terminal Equipment"), conectado em geral a um modem, DCE ("Data Communication Equipment"). Deste modo o PC pode operar em redes, quer com um computador ou outro dispositivo ao lado, quer com qualquer outro computador a milhares de quilômetros através da rede de telefonia.

O conector utilizado é o padrão DB-25P, tipo macho, conforme descrição da Figura 1.2. Somente alguns pinos são utilizados, conforme descrição na Tabela 1.4:

Direção	DTE		DCE
		TD ("Transmit data")	
Saída	2.	→	2.
		RD ("Receive data")	
Entrada	3.	→	3.
		RTS ("Request to send")	
Saída	4.	→	4.
		CTS ("Clear to send")	
Entrada	5.	→	5.
		DSR ("Data set read")	
Entrada	6.	→	6.
		GND ("Ground", terra)	
Comum	7.	→	7.
		DCD ("Carrier detect")	
Entrada	8.	→	8.
		DTR ("Data Terminal Ready")	
Saída	20.	→	20.
		RI ("Ring indicator")	
Entrada	22.	→	22.

Tabela 1.4 Descrição da Porta Serial RS-232



Em geral o endereço da primeira porta serial ou COM1, está indicado na base da área de dados do BIOS (400h) e vale 3F8h. Na Seção 9.5, o programa ACHAR_B.C sugere como você pode reconhecer este endereço no seu PC. Ver na Tabela 4.1, Seção 4.3 o endereço das demais portas seriais.

Existe um programa residente no BIOS, BIOS INT 14h, capaz de configurar, controlar, enviar e receber informações através da porta serial, ver Seção 9.1. É sempre preferível usar esta rotina de interrupção do que controlar diretamente a saída serial.

1.2.8 Entrada de Jogos

São 4 entradas digitais (2 chaves liga/desliga e 4 entradas analógicas definindo 2 coordenadas X,Y para cada jogador.

O endereço desta porta é 0201h, ver Tabela 4.1, Seção 4.3. Através de instrução IN neste endereço pode-se ler o estado das 4 chaves digitais. A localização das 4 chaves no conector tipo D15, é:

Bit	Função	Pino
4	chave 1/entrada digital	2
5	chave 2/entrada digital	7
6	chave 3/entrada digital	10
7	chave 4/entrada digital	14
-	terra (GND)	4
-	terra (GND)	5
-	terra (GND)	12
-	+ 5 V	1
-	+ 5 V	8
-	+ 5 V	12
-	+ 5 V	15

Tabela 1.5 Descrição dos pinos da entrada de jogos

A entrada analógica está nos pinos 3,6,11 e 13, que podem estar conectados ao ponto médio de um potenciômetro.

1.2.9 Alto-Falante

Localizado próximo ao canto frontal esquerdo este pequeno alto-falante fornece o bip usual de aviso. Vamos utilizá-lo na Seção 11.2 para descrever a função de 2 circuitos importantes dentro do PC, o temporizador programável ("timer") e a porta paralela 8255. Mostraremos um pequeno programa capaz de tornar o teclado do PC um órgão, de qualidade sonora um tanto duvidosa devido à limitação do sistema de áudio do alto-falante.

1.3 Periféricos e Acessórios

Os slots livres na arquitetura do PC permitem uma atualização e aperfeiçoamento constante. Da configuração inicial de 2 disk-drivers flexíveis de 5.1/4", utilizando-se os "slots" livres, ela foi expandida para incluir discos de alta densidade, de 3.1/2" ou discos rígidos Winchester de 30 Mb ou mais. Ao longo do tempo periféricos tais como entrada de gravador tornaram-se obsoletos. Porém foram incluídos ou-tros como o "mouse" (dispositivo externo de controle do cursor), e "scanner" (dispositivo de digitalização de imagens gráficas) que serão comuns nos futuros PCs.

O uso versátil do slot do PC permite a conexão de placas com microprocessadores mais poderosos, o que permite a transformação do PC ou PC-XT num microcomputador AT equivalente, a baixo custo.

ARQUITETURA DO PC

2.1 Introdução

Este Capítulo é um sumário das funções principais e dos integrados existentes na placa principal, onde reside o núcleo das principais funções eletrônicas do PC. Nosso objetivo resume-se em analisar as partes do circuito necessárias para o projeto e implementação de interfaces do PC.

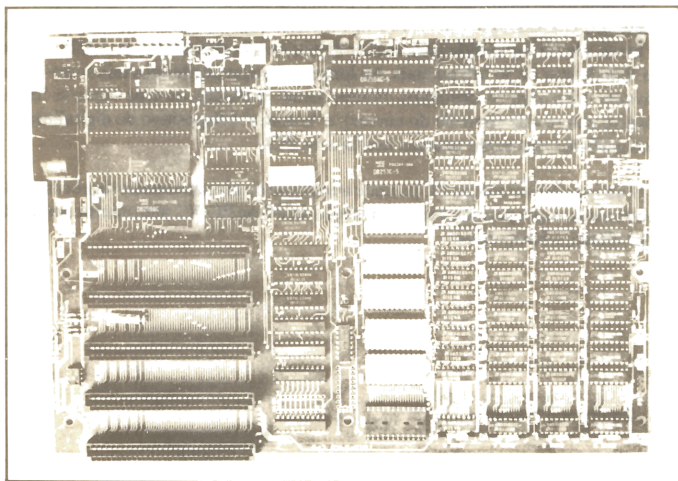


Figura 2.1 Placa Principal do PC

A Figura 2.1 é uma fotografia da placa principal do PC. Esta é a imagem de um sistema sem a fonte e os "disk drivers".

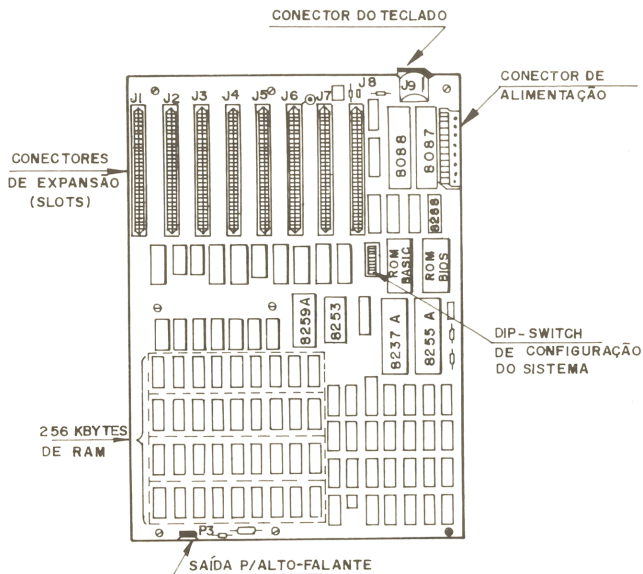


Figura 2.2 Dispositivos Principais

A Figura 2.2 é uma descrição dos dispositivos principais. As Figuras 2.1 e 2.2 estão relacionadas entre si conforme esboço do diagrama elétrico da Figura 2.3.

disponíveis 20 pinos, o que possibilita o acesso direto a $2^{20} = 1$ Mbyte de endereços de memória, ver na Seção 2.6 o mapeamento da memória.

A capacidade de receber e transmitir instruções, a anatomia dos seus registradores internos, estão discutidos no próximo Capítulo.

2.3 O Circuito Temporizador

O microprocessador gasta um tempo mínimo para executar qualquer tarefa, o período de "clock". No PC padrão, a frequência de trabalho é $f = 4.77$ MHz, ou um período de clock, $T = 1/f = 210$ nano-segundos. Esta frequência de trabalho é determinada pela capacidade do microprocessador. Por exemplo a versão 8088-2 trabalha com uma frequência máxima de 8 MHz, o que permite microcomputadores mais rápidos, tal como o PC-XT Turbo, que pode trabalhar tanto com 4.77 MHz como com 8 MHz.

A geração da base de tempo é feita através de um cristal de frequência $f = 14.31818$ MHz, conforme diagrama da Figura 2.4, conectado ao circuito integrado temporizador ("timer"). O 8284 divide esta frequência por 3, fornecendo o valor de 4.77 MHz para o microprocessador 8088. Ambos os sinais, de 4.77 MHz e 14.31818 MHz, estão presentes nos slots de expansão como discutimos no Capítulo 4.

O PC possui outros contadores, tal como o contador programável 8253, que utilizam a base de tempo do clock. Por exemplo, o 8253 usa a base de tempo do clock dividido por 3, ou seja $f = 4.77/3 = 1.19$ MHz. Na Seção 2.6, discutimos mais detalhadamente este contador.

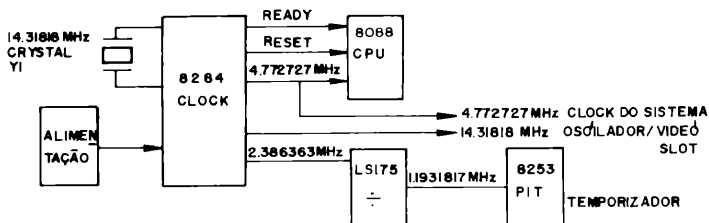


Figura 2.4 Base de tempo do PC.

2.4 O "bus" do Sistema

A conexão entre o microprocessador e os demais componentes é feita através do "bus" do sistema. Análogo ao cabo telefônico, o bus é um conjunto de linhas de transmissão de sinais onde cada fio pode representar uma via. A Figura 2.5 representa um diagrama do bus do sistema:

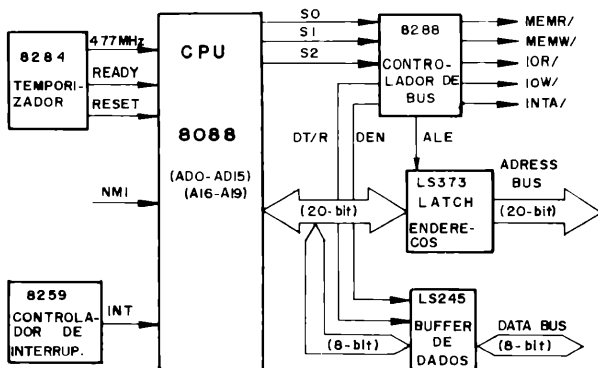


Figura 2.5 O "bus" do sistema.

O "bus" de controle envolve uma série de sinais: controle dos "latches", reset, temporização, interrupção, escrita e leitura na memória e nos periféricos, etc.. Para estabelecer prioridades e ampliar as informações de controle geradas na CPU, existe um circuito integrado para manejar o "bus" de controle - 8288. Por exemplo, é o 8288 que estabelece a distinção entre a solicitação de escrita numa memória interna, MR ("Memory Read") ou num dispositivo externo IOR ("Input/Output Read"), no bus de controle a partir da decodificação dos sinais de status S0, S1 e S2 enviados pelo 8088, ver Tabela 3.2 na Seção 3.4.

A maioria dos sinais do "bus" do sistema estão presentes nos slots de interfaceamento. O projetista deve ser capaz de caracterizar estes sinais para aproveitar ao máximo os recursos do PC. No Cap. 4, analisamos os sinais do bus existentes nos slots.

2.5 O Mapeamento de Memória

Como o "bus" de endereço é de 20 bits, o sistema pode decodificar de 00000h a FFFFFh. Fazendo uma analogia com uma coleção de livros, podemos usar os 4 bits mais significativos (A16-A19) para representar um volume de memória. Neste caso temos $2^4 = 16$ volumes com $2^{16} = 64$ Kbytes de páginas cada um. Cada volume contém um tipo de informação. Os dez primeiros volumes são páginas em branco, onde o leitor pode escrever ou apagar os seus próprios programas, os $10 \times 64K = 640K$ de memória RAM. Os outros 6 são volumes onde só é possível a leitura e, dentre eles, alguns estão até faltando em algumas versões do PC. Porém o mais importante de todos é o último, onde esta localizado o ROM - BIOS. A Figura 2.6 representa uma pilha destes volumes.

F	FFFF	ROM do sistema
F	0000	uso do sistema ?
E	0000	uso do sistema ?
D	0000	controlador de disco
C	0000	vídeo RAM
B	0000	expansão vídeo RAM
A	0000	livre para uso
9	0000	livre para uso
8	0000	livre para uso
7	0000	livre para uso
6	0000	livre para uso
5	0000	livre para uso
4	0000	livre para uso
3	0000	livre para uso
2	0000	livre para uso
1	0000	uso do sistema
0	0000	uso do sistema

$10 \times 64K = 640 K RAM$

Figura 2.6 Descrição em volumes da memória

O programa residente mais importante, o BIOS, ver Seção 2.5.1, localiza-se no topo da memória. Alguns volumes nem sempre são utilizados, (volumes D e E), pois foram projetados para uso do Basic residente do PC com fita cassete, ou seja sem disk-drive. Em todo caso estão reservados para futuras expansões, por exemplo como controlador do disco rígido.

O volume inferior da memória é destinado a armazenar os vetores de interrupção e a área de dados do BIOS e do DOS. Escrever um pixel na tela de vídeo significa visualizar uma certa área de memória, ver Seção 7.6. No caso, o volume B é destinado a armazenar as imagens de vídeo. Para placas gráficas de alta resolução existe a possibilidade de expansão no volume A, tal como é utilizado pela placa EGA.

2.5.1. A ROM do sistema

Os 4 últimos volumes no topo da pilha de memória, com páginas entre : C0000h e FFFFFh , são destinados à memória não volátil do sistema, a ROM ("Read Only Memory") ou EPROM ("Erasable Programmable ROM").

Pode parecer um desperdício, porém dos 256 K de ROM, somente uma pequena parcela de 8 K entre os endereços: FE000h - FFFFFh é utilizada para o controle básico de entrada e saída, a BIOS ("Basic Input/Output System"). Os volumes C,D,E (192 k) são destinados a possíveis futuras expansões. Mesmo no último volume existem espaços livres que podem ser utilizados, é o caso da região de memória entre F4000h - F6000h, destinada originalmente para extensões do Basic residente, sem uso do disk-drive, e atualmente é um soquete vazio. Nas últimas versões de PC, nem sempre o Basic residente interpretado está presente. As 4 EPROMS de 8 k (ao todo 32 K), nos endereços F6000h e FF000h é uma área vazia na placa principal na região das EPROMS nestes microcomputadores.

Nos 8 K bytes do topo da memória ROM reside o BIOS, um conjunto de programas capaz de realizar várias tarefas tais como:

- inicialização do sistema
- diagnóstico da fonte de alimentação
- teste da memória RAM do sistema
- determinação da configuração dos periféricos.
- controle dos dispositivos externos (vídeo, teclado, drivers, etc..)
- armazenamento da fonte padronizada dos 128 primeiros caracteres do sistema

As tarefas do BIOS são fundamentais e devem ser utilizadas preferencialmente. Usar as interrupções do BIOS, ver Capítulo 9, é o modo mais fácil de manipular os periféricos além de facilitar o uso do programa em diferentes versões de microcomputador. A caracterização da versão do microcomputador e os dados do fabricante estão na base dos endereços do BIOS e, na prática, cada fabricante possui uma versão própria do BIOS. No Exemplo 7.2, Seção 7.4., você pode verificar como o fabricante reescreveu a marca original da IBM no BIOS do seu PC.

Os periféricos estão em constante evolução, assim como o tamanho da memória RAM livre para o usuário. Isto poderia sugerir o problema de fazer nova versão do BIOS para cada modificação feita. Felizmente isto pode ser evitado, pois junto com o sistema operacional existe o IBMBIO.COM (alguns fabricantes alteram-no para XXXBIO.COM). Este programa é invisível através do DIR do DOS, porém você pode verificá-lo através do PCTOOLS. Nele estão as extensões do ROM BIOS que são carregadas junto com o "boot" do sistema operacional. Deste modo não é necessário alterar o ROM BIOS, basta reescrever o programa IBMBIO.COM redirecionando os vetores de interrupção para rotinas capazes de manipular novos dispositivos.

2.5.2 A área de RAM do sistema

O preço de um microcomputador e a quantidade de memória livre para uso estão relacionados, diretamente, com a capacidade de fabricação de memórias dinâmicas RAM com maior densidade e mais baixo custo. O PC original foi projetado para usar memórias do tipo 4116 (16 K bit x 1, ou 2 Kbytes). No desenho original existem 4 bancos com 9 integrados de memórias cada. Porém, uma memória em cada banco é usada para teste de paridade, o que significa que $4 \times 8 \times 2 = 64K$ de memória RAM era o máximo permitido na placa principal do PC original. Atualmente utiliza-se memórias de maior capacidade tipo 4164 (8K) ou 41256 (32 K), o que permite alocar na placa principal o 640K, o limite previsto de memória não volátil pela arquitetura do PC. É interessante notar que este avanço foi realizado sem acrescentar pinos extras aos 16 pinos da memória 4116, o que permite usar o mesmo espaço original do PC destinado ao bloco de memória RAM.

Podemos considerar os 640K de memória RAM livre como virtual, pois parte dela já está comprometida com o sistema. A Figura 2.7 representa um diagrama mais detalhado do Bloco de memória RAM.

ENDEREÇO	USO
A 0000	topo da memória RAM livre
	COMMAND.COM
	programas transientes
	programas residentes (TSR)
	COMMAND.COM
	controle do driver
	núcleo do DOS
0 0500	área de dados do DOS
0 0400	área de dados do BIOS
	BASIC INT vetor (80/FFh)
0 0200	INT vetor livre (40/7Fh)
0 0100	DOS INT vetor (20-3Fh)
0 0080	BIOS INT vetor (10-1Fh)
0 0020	INT vetor sistema
0 0000	

Figura 2.7 Uso da memória RAM pelo DOS.

A Figura 2.7 não está em escala, porém indica a ordem em que os programas são montados pelo DOS na área de RAM. De fato, quanto de memória livre o usuário dispõe? A resposta depende de uma série de fatores, dentre os quais as versões DOS e como o DOS aloca memória para diferentes tipos de programas. Porém pode-se considerar a área livre para programas do usuário acima de 1 500h. Em todo caso é sempre preferível deixar para o DOS e o LINKER do compilador alocar a área física para programas e dados, logicamente delimitada no programa, em ASSEMBLY ou em uma linguagem de alto nível.

A área de vetores de interrupção é vital para funcionamento do microcomputador e para projetos de interfaceamento, ver Seção 2.7 para uma discussão mais detalhada desta região de memória.

Ao carregar o sistema operacional ("boot"), os dois programas "ocultos" IBMBIO.COM E IBMDOS.COM preenchem a memória RAM da área de dados do BIOS e do DOS, além de alocar os vetores de interrupção e ficam disponíveis para uso do sistema.

O programa COMMAND.COM é carregado junto com o sistema operacional. Pode aparecer estranho o uso de 2 áreas distintas para alocar este programa na Figura 2.7. O COMMAND.COM é dividido em 2 partes, a inferior, porção residente, e a superior transiente. A parte residente é responsável pela manipulação dos erros de programa e permanece mesmo quando o programa do usuário estiver sendo executado. A parte superior transiente contém os comandos intrínsecos do DOS, tais como DIR, COPY, SET e só são úteis no início ou na depuração dos programas do usuário. Por isto é comum ao se finalizar uma operação com o editor de texto, o WORD (Microsoft) por exemplo, ele solicitar a introdução de um disco com o programa COMMAND.COM, pois é necessário reestabelecer a parte transiente para retomar o comando integral do DOS.

Programas residentes referem-se aos programas que terminam e permanecem residentes ou TSR ("Terminate and Stay Resident"), tal como o SIDEKICK da Borland. Eles são os primeiros alocados ao final do DOS e só são executados através de um comando externo, ao acionamento de uma tecla especial ("hot key") ou periodicamente quando solicitado por uma interrupção, ver Seção 11.4. Esse tipo de programação é bastante interessante para rotinas em tempo real, pois permite a realização de várias tarefas simultaneamente. Porém, é necessário um conhecimento profundo da capacidade da versão DOS, das características de hardware e das interrupções BIOS e DOS. Na Seção 11.4, discutimos os requisitos para a implementação desses programas.

Na memória destinada ao controlador de dispositivos, estão armazenadas as informações a respeito dos periféricos instalados, tal como detectado pelo BIOS ou redefinido pelo CONFIG.SYS, ver Seção 9.3.

O núcleo do DOS é onde estão armazenados os programas apontados pelos vetores de interrupção DOS, tal como as funções da INT 21h, como está discutido no Capítulo 10 e Apêndice E.

A área dos vetores de interrupção contém os endereços das sub-rotinas apontadas pelos 256 vetores de interrupção, permitidos pelo 8088. Porém, como cada endereço ocupa 4 bytes (2 para designar o segmento e 2 para offset, ver Cap.3 para definição dos endereços pelo 8088), o espaço total ocupado é 400h (1.024 bytes). Ver na Seção 2.8 os vetores de interrupção do PC.

2.6 O Contador/Temporizador do Sistema

Na Seção 2.2 descrevemos o circuito de "clock" até o ponto em que ele envia um clock de frequência de 1.19 Hz, do temporizador programável PIT ("Programmable Interval Timer") A Figura 2.8 é um diagrama do circuito de "clock" e seus 3 canais de saída.

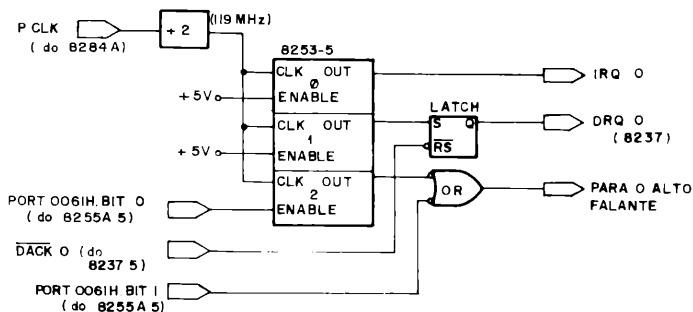


Figura 2.8 O circuito externo do temporizador.

A entrada de "clock" nos 3 canais é um sinal de onda quadrada de frequência 1.19 MHz. Os 3 canais podem ser programados de modo independentes através das intruções IN e OUT (Ver Seção 11.2), nos endereços 0040h a 0043h, conforme Tabela:

Endereço	Escrita(out)	Leitura(in)
0040h	carrega contador 0	lê valor do contador 0
0041h	carrega contador 1	lê valor do contador 1
0042h	carrega contador 2	lê valor do contador 2
0043h	define modo de controle	operação inválida

Tabela 2.1 Endereços do temporizador 8253.

Embora seja possível mudar a programação do 8253, quer na frequência ou na forma do sinal de saída, tal procedimento não é aconselhável, particularmente para os canais 0 e 1, pois estes sinais são utilizados para controle do sistema.

O canal 0 é carregado com o valor $N = \text{FFFFh}$, e programado para operar como gerador de uma onda qua-drada de frequência igual à frequência do clock dividido por N, ou:

$$f = 1.19\text{MHz}/\text{FFFFh} = 18.206\text{Hz}.$$

Ou seja a cada período de $T = 1/f = 54.936$ mili-segundos o canal 0 gera uma interrupção (INT 08h) de hardware. Deste modo, a cada aproximadamente 55 mili-segundos, o canal 0 solicita uma rotina de interrupção com as seguintes funções: atualizar o relógio interno do microcomputador e a data, verificar o estado do disk-drive e, o que é mais interessante para programação em tempo real, procurar através da INT 1Ch, alguma rotina para ser executada. A BIOS INT 1Ah fornece o status do relógio em tempo real do PC, Ver Capítulo 9.

As memórias RAM do PC são do tipo dinâmicas e necessitam de um pulso de "Refresh" periodicamente. Isto é realizado através do DMA ("Direct Memory Access"), através do sinal enviado pelo canal 1 do timer a cada 15,12 micro-segundos. Como este processo é crítico para operação do sistema, não se pode nem pensar em alterar ou modificar a programação deste canal.

O canal 2 pode ser programado livremente e é utilizado para controle do alto-falante. A porta paralela 8255, liga e desliga o alto-falante, ao passo que o canal 2 da 8253 envia uma onda com frequência programável para o alto-falante. Na Seção 11.2, Exemplo 11.1, mostramos um programa que codifica as teclas do PC para enviar sons de frequências variáveis para o alto falante através da programação deste canal.

2.7 Acesso Direto à Memória - DMA

Alguns dispositivos de entrada e saída podem transferir dados mais rapidamente que a capacidade do 8088, haja vista o clock de trabalho, em controlar esta operação. A solução reside no integrado 8237 - DMA ("Direct Memory Access") que é capaz de controlar a transmissão de dados diretamente de um dispositivo externo para o banco de memória RAM, sem passar pelo microprocessador. A velocidade de transmissão de dados através das funções usuais de interrupção é de aproximadamente 30KHz por byte. Porém com o auxílio do DMA esta faixa pode ser estendida a 500 KHz. O usuário pode implementar seu próprio DMA, pois os pinos de acesso e controle estão nos slots, além disto é possível reprogramar o 8237.

2.8 As Interrupções do Sistema

Uma característica da família 8088 é a grande capacidade de manipular até 256 tipos diferentes de interrupção. O diagrama da Figura 2.9 representa o fluxograma de uma interrupção genérica.

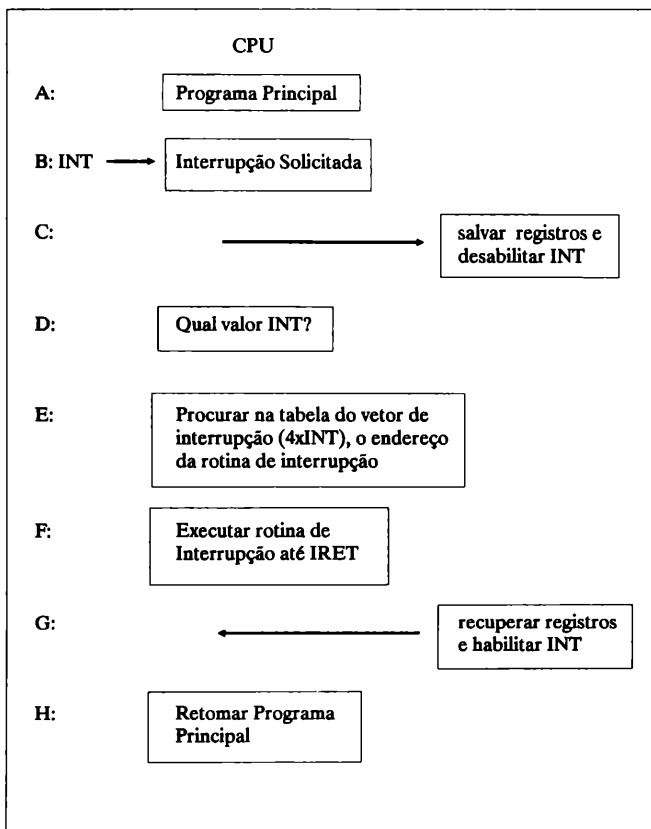


Figura 2.9 Diagrama do ciclo de uma interrupção

As interrupções podem ocorrer em 3 níveis:

- internas ao microprocessador
- interrupções não mascaráveis, NMI (Non-Maskable Interrupt)
- interrupções mascaráveis (software ou hardware)

As interrupções internas são definidas pelo fabricante do microprocessador, no caso INTEL, e servem para sinalizar alguma operação interna do 8088, por exemplo erro de divisão por zero.

As interrupções não mascaráveis não podem ser desabilitadas por software. Na arquitetura do PC ela é destinada à comunicação entre o co-processador matemático 8087 e a CPU.

A grande maioria das interrupções são do tipo mascaráveis, ou desabilitadas por software. Neste caso, quando as interrupções estiverem desabilitadas, a CPU mascara, ou ignora, novas solicitações de interrupção. A CPU só executa a rotina de interrupção que foi solicitada e ignora as demais interrupções mascaráveis, entre as etapas C e G na Figura 2.9.

Essas interrupções podem ser tanto através de software como por hardware. Na interrupção por hardware o circuito eletrônico deve ser capaz de indicar ao microprocessador o vetor de interrupção. O 8088 só possui um pino para entrada de interrupção por hardware e isto é muito pouco. A solução foi recorrer a outro circuito integrado, o 8259, um controlador de interrupção capaz de ampliar para 8 o número de possíveis interrupções. Ver Cap. 5 para detalhes a respeito das interrupções por hardware controladas pelo 8259 no PC.

A Tabela 2.2 é um resumo dos vetores de interrupção, com os respectivos endereços e funções:

número INT	endereço vetor	definido por	função
0	00-03h	INTEL	erro divisão por zero
1	04-07h	INTEL	execução passo a passo(trace)
2	08-0Bh	INTEL	NMI("Non-Maskable Interrupt")
3	C-0Fh	INTEL	break-point
4	10-13	INTEL	overflow
5	14-17h	BIOS	Print-Screen
6	18-1Bh	INTEL	reservado
7	1C-1Fh	INTEL	reservado
8/IRQ0	20-23h	IBM	8259 timer
9/IRQ1	24-27h	IBM	8259 teclado
A/IRQ2	28-2Bh	IBM	8259 livre para uso
B/IRQ3	2C-2Fh	IBM	8259 livre p/ uso(ou serial 2)
C/IRQ4	30-33h	IBM	8259 serial 1 (COM1)
D/IRQ5	34-37h	IBM	8259 livre (ou impressora 2)
E/IRQ6	38-3Bh	IBM	8259 disk-driver
F/IRQ7	3C-3Fh	IBM	8259 paralela impressora
10/1F	40/7Fh	BIOS	BIOS INT
20/3F	80/FFh	DOS	DOS INT
40/5F	100/17Fh	BIOS	BIOS avançado
60/66	180/19Bh	-	livre para uso
68/F0	1A0/3C3 h	-	miscelânea (BIOS, Basic, etc..)
F1/FF	3C4/3FF h	-	reservado.

Tabela 2.2 Vetores e endereços de interrupção

Os vetores de interrupção ocupam os primeiros 400h bytes de memória. No PC, cada endereço efetivo (Ver Cap 3.) ocupa 4 bytes, (2 para o segmento, mais 2 para o offset). Por isto o endereço dos vetores de interrupção é multiplicado por 4. Por exemplo, a interrupção do temporizador BIOS INT 1Ch, tem o seu vetor em: $4 \times 1Ch = 70h$. Com o auxílio do DEBUG (Ver Cap 7) você pode verificar, a partir de 70h, o endereço efetivo da sub-rotina de interrupção de manipulação do vídeo:

73h	—————>	F0	
72h	—————>	00	Segmento : F000 (word)
71h	—————>	FF	
70h	—————>	53	Deslocamento(off-set) : FF53 (word)

No PC os endereços são lidos a partir do byte menos significativo. Ou seja, subindo na pilha de endereços a partir da base, encontramos os 2 primeiros bytes menos significativos de deslocamento (às vezes o denominaremos como off-set, tal como é usual) e depois os 2 últimos bytes mais significativos, apontando para a parte do segmento onde está a rotina. Neste exemplo, o endereço da sub-rotina de interrupção do relógio é: (CS:IP) F000:FF53 (Ver Cap 3). É este endereço que a CPU vai utilizar para executar a sub-rotina de interrupção no passo F da Figura 2.9 .

As interrupções são a base de toda programação do PC. Felizmente deixaram uma interrupção por hardware, INT Ah/ IRQ2, para projetos do usuário. Esta interrupção pode ser ampliada caso necessário. No Cap.5 discutimos a operação de interrupção por hardware, ver também Seção 11.3.

As interrupções BIOS são fundamentais para controle de dispositivos externos, tais como vídeo e teclado. No Cap.9 tratamos exclusivamente deste tipo de interrupção.

Embora exista um grande conjunto de interrupções destinado ao DOS, na prática, somente uma DOS INT é utilizada intensamente, a INT 21h. Em contrapartida a ela está agregada um grande número de funções. No Cap 10 mostramos vários programas envolvendo as funções do DOS INT 21h.

2.9 Interface com o Mundo Exterior

Alguns dispositivos externos, tais como o teclado e o alto-falante estão conectados ao bus do sistema através da porta paralela 8255-PPI ("Programmable Peripheral Interface") Esta porta é importante pois trata-se de 24 bits para uso como controle digital de entrada e saída, e utiliza os endereços de I/O (Ver Cap 4) de 0060h/0063h, conforme Tabela 2.3.

endereço	porta	função principal no PC-XT
60h	A	leitura do teclado
61h	B	sinais de controle, som.
62h	C	estado de configuração do sistema, chaves.
63h	controle	usado para programar 8255

Tabela 2.3 Endereços da porta paralela 8255

O 8255 possui 3 portas A,B,C que podem ser programadas como entrada e saída de modo independente. Na placa principal, ela está configurada conforme Figura 2.10.

8255 PPI	p o r t A	PA0 — entrada de teclado PA1 — entrada de teclado PA2 — entrada de teclado PA3 — entrada de teclado PA4 — entrada de teclado PA5 — entrada de teclado PA6 — entrada de teclado PA7 — entrada de teclado
	p o r t B	PB0 — habilita canal 2 do 8253 PB1 — controle alto-falante PB2 — controle do DMA PB3 — controle leitura chaves PB4 — habilita teste paridade RAM PB5 — testa canal I/O PB6 — controle de teclado PB7 — controle de teclado
	p o r t C	PC0 — ler chaves internas da placa PC1 — ler chaves internas da placa PC2 — ler chaves internas da placa PC3 — ler chaves internas da placa PC4 — monitorar alto-falante PC5 — monitorar canal 2 do 8253 PC6 — testa canal I/O PC7 — testa paridade da RAM

Figura 2.10: Configuração da porta paralela 8255

Esta configuração deve ser vista com cautela, pois houve muitas modificações a partir do PC original. As chaves internas que caracterizam os periféricos instalados no PC podem ser observadas através de uma leitura IN na port C, tal como é feito através da BIOS INT 11h , ver Exemplo 9.2 na Seção 9.3 .

O port B está configurado como saída e pode ser programado, como fazemos no Exemplo 11.1, Seção 11.2, quando o utilizamos para acionar o alto-falante. Como os outros bits têm funções específicas esta programação deve ser feita com cuidado para não alterar a operação normal do PC. O port C está configurado como entrada e é utilizado para funções de teste internos do PC, particularmente do banco de memórias RAM.

O modo mais simples e seguro de se construir uma interface do PC com o mundo exterior é através da conexão de uma porta paralela 8255 no slot de interface. Deste modo possuímos uma porta digital de entrada e saída versátil, capaz de controlar conversor analógico/digital e digital/analógico, motor de passo, relés, etc...

CAPÍTULO 3

MICROPROCESSADOR 8088

3.1 Introdução

O microprocessador 8088, também denominado CPU ("Central Processing Unit") e o 8086 são membros da terceira geração de microprocessadores. Neste Capítulo, descrevemos as principais características de hardware do 8088 tal como é utilizado no PC. Ver Cap.8 e Apêndice B para uma descrição do código de instruções. O 8088 é projetado com um bus de dados de 8 bits enquanto o 8086 pode transferir 16 bits simultaneamente. Porém em quase todos os outros aspectos, ambos processadores são idênticos. O chip vem no encapsulamento padrão de 40 pinos e é alimentado unicamente com +5V, conforme Figura 3.1

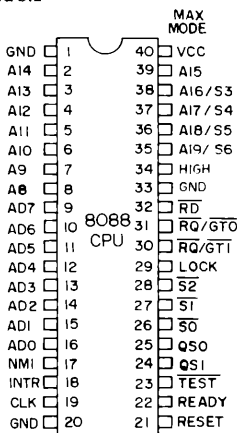
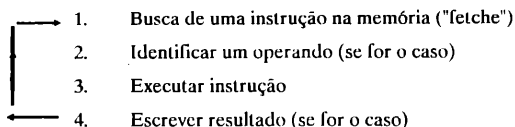


Figura 3.1 Descrição dos pinos do 8088

Existe uma incompatibilidade física entre o número de pinos (20) da CPU destinados aos endereçamentos de memória e I/O (Input/Output) e a capacidade de armazenamento dos registradores internos do 8088 (16 bits). A falta destes 4 bits obriga a CPU usar 2 registradores, o de segmento ("segment") e o de deslocamento ("off-set") para apontar um único endereço de memória. Esta aparente dificuldade é uma fonte de confusão para o iniciante e um incômodo para o programador experiente. Neste Capítulo, na Seção 3.3, abordamos o modo de endereçamento do 8088.

O microprocessador executa, sob o compasso do clock, quatro operações básicas de modo cíclico :



Como encontrar a instrução? O registrador IP (Instruction Pointer) aponta sempre para a próxima instrução a ser executada pela CPU. As instruções podem ser desde não fazer nada, NOP ("No Operation") até executar instruções lógicas tipo AND ou OR ou aritméticas ADD ("Addition") ou divisão DIV ("Division"). As operações lógicas e aritméticas são realizadas na unidade ALU, ("Arithmetic Logic Unit") da CPU, e são monitoradas por um registrador de sinalização, ou FLAG.

Ver Cap.8 e Apêndice B. o código instruções e Assembly menmônicos do 8088. Os operandos, para serem manipulados pela ALU, devem estar num acumulador ou registrador de uso geral, bem como numa região de memória de dados, apontada por ou mais registradores do tipo apontadores e índice.

3.2 Arquitetura do 8088.

A Figura 3.2 é um diagrama da arquitetura do 8088.

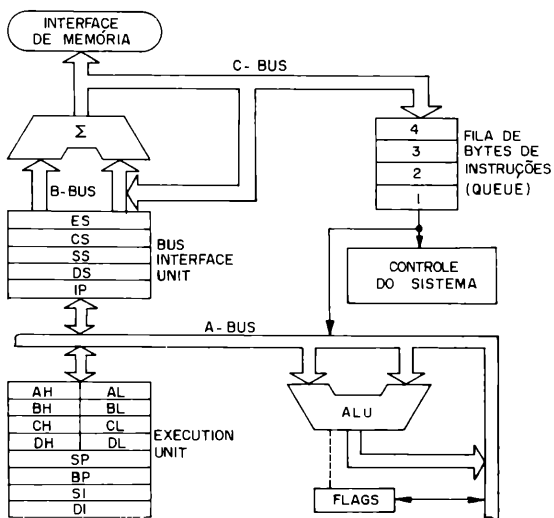


Figura 3.2 Arquitetura do 8088

O microprocessador é dividido em 2 unidades lógicas: a unidade de execução - EU ("Execution Unit") e a unidade de interface de bus - BIU ("Bus Interface Unit"). A comunicação da CPU com o exterior é controlada pela BIU. As 4 etapas de uma instrução são condensadas devido à capacidade da BIU acomodar mais de uma instrução ("queue") simultaneamente a execução de uma instrução na EU.

A Figura 3.3 caracteriza os registradores internos do 8088.

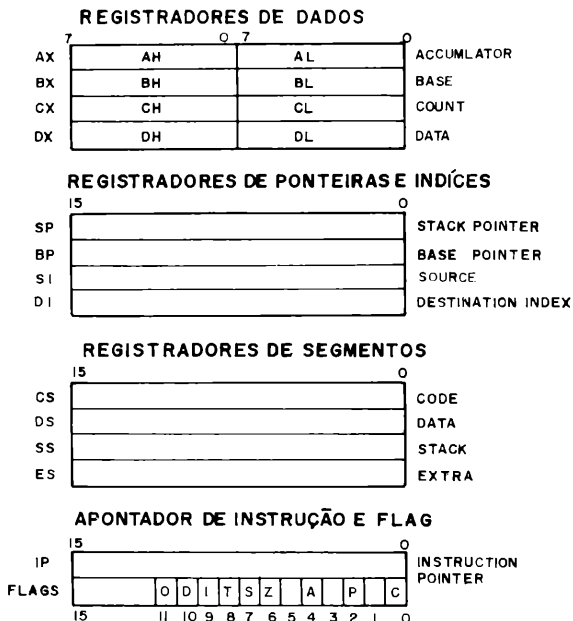


Figura 3.3 Registradores internos do 8088

Fazendo uso do utilitário DEBUG (Ver Cap 6), é possível não só verificar como modificar o conteúdo destes registradores.

Embora sejam registradores de uso geral, suas iniciais indicam um tipo de função que executam prioritariamente:

AX (AH + AL): Acumulador de 16 bits (AX) ou 2 de 8 bits (AH e AL).
Usado preferencialmente para operações com a ALU.
Utilizado para identificar as funções DOS, ver Cap 11.

BX (BH + BL): Base. usado preferencialmente como apontador da base de uma tabela de dados.

CX (CH + CL): Contador. Usado prioritariamente para contar o número de interações no interior de um loop. Usado na manipulação de cadeias ("strings").

DX (DH + DL): Dados. usado para guardar dados de 16 bits nas operações com ALU e controle indireto de I/O.

Através dos 4 registradores de segmentos a CPU pode acessar até 4 segmentos de 64K simultaneamente. Ver Seção 3.3 a respeito de endereçamento no 8088.

CS: ("Code Segment") aponta para o segmento de código de instruções em uso, em conjunto com o IP define o endereço efetivo da próxima instrução a ser executada pela CPU.

SS: ("Stack Segment") Aponta para o segmento da pilha em uso. Ver no Cap.8 as instruções POP ("Pop word off stack") e PUSH ("Push word onto stack").

DS: ("Data Segment") aponta para os segmentos de dados, onde geralmente são armazenadas as variáveis de programa.

ES: ("Extra Segment") uma opção extra para apontar a base de segmentos de dados.

Os registradores de ponteiros: SP ("Stack Pointer") e BP ("Base Pointer") são registradores auxiliares para operações na pilha e na definição de um endereço efetivo. Os registradores de índices SI ("Source Index") e DI ("Destination Index") são utilizados para operações com cadeias ("strings").

O registrador de Flag pode ser dividido em 2 blocos, um de 6 bits, que reflete o status de operações realizadas na ALU, outro de 3 bits de controle da própria CPU, conforme Figura 3.4.

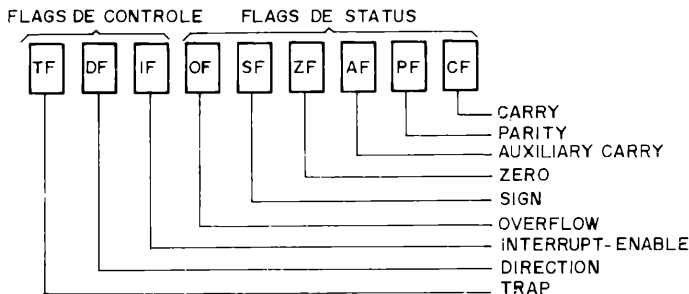


Figura 3.4 Bits do registrador de Flag do 8088

Bits de Status

1. C(arry): bit 0.

Se C = 1 então ocorreu um "vai um" ou um "empréstimo" no bit de mais alta ordem do resultado durante uma operação de soma ou subtração em binário.

2. P(aridade): bit 2.

Se P = 1, o resultado de uma operação tem um número par de bits. Pode ser usado para teste de paridade durante comunicação serial.

3. A(uxiliary): bit 4.

Se A = 1, então ocorreu um "vai um" ou um "empréstimo" entre os 4 bits menos significativos, ("nibble"). Usado em operações aritméticas decimais. Ver instrução DAA ("Decimal Adjust for Addition") no Cap.7.

4. Z(ero): bit 6.

Se o resultado final é nulo, então Z = 1. Usado em operações de salto, JZ (Jump if Zero)

5. S(ign): bit 7.

Se o bit de mais alta ordem do resultado for 1, S = 1. Caso os operandos sejam números negativos, S indica se o resultado final é positivo (S = 0) ou negativo (S = 1), conforme notação usual de complemento 2.

6. O(overflow): bit 11.

O = 1 se ocorreu um estouro ("overflow") na operação aritmética. Neste caso, um bit significativo perdeu-se, pois o resultado excedeu a capacidade de armazenamento do registrador ou da memória a que se destina.

Bits de Controle

1. I(nterrupt): bit 9.

Caso a CPU esteja habilitada para receber interrupções mascaráveis, (ver Secão 2.7), então I = 1. Caso contrário, I = 0 desabilita interrupções mascaráveis.

2. D(irection): bit 10.

Determina se as operações de string devem ser auto-decrementadas (D = 1) ou incrementadas (D = 0)

3. T(rap): bit 8.

Se T = 1 a CPU executa instruções passo-a-passo, usado em operações de "debug"

3.3 Endereçamento através da CPU

É preciso distinguir 2 tipos de endereçamento no PC:

endereço físico: de 0 0000h a F FFFFh, 1 MByte de endereço definido pelo bus de endereço (A0-A19).

endereço lógico: definido pelos registradores de segmento e de deslocamento, tais como são utilizados programas.

As instruções e dados, de 1 byte (8 bits) ou 1 word (16 bits), podem ser armazenados livremente em qualquer endereço físico. Na Secção 2.8, observamos como 1 word é armazenada e esta operação pode ser visualizada com o uso do DEBUG, Ver Cap 7.

Segmento é uma unidade lógica de 64 Kbytes capaz de ser identificada por um único registrador de segmento (16 bits). O registrador identifica o endereço da base de memória, ou ponto de partida. Diferentes da memória física, os segmentos lógicos não obedecem a nenhuma restrição de localização e podem ser: adjacentes, separados, sobrepostos parcial ou totalmente, ver Figura 3.5

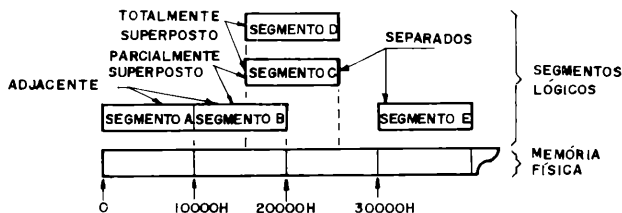


Figura 3.5 Endereços físicos e lógicos.

Os compiladores procuram compactar ao máximo a memória dos programas. É comum alocar os mesmos segmentos para dados e para programas de modo total ou parcialmente superpostos.

Porém, esta versatilidade é fonte de confusão, pois a um mesmo endereço físico pode estar associado mais de um segmento. Além disto existem diversas maneiras de usar o registrador de segmento e de deslocamento para compor um único endereço físico.

Considere uma situação semelhante. Pode-se dizer que a distância entre Porto Alegre e Rio de Janeiro é a mesma que a soma dos trechos :Porto Alegre/Curitiba (segmento) e Curitiba/Rio de Janeiro (deslocamento). Porém esta distância pode ser representada por um outro segmento (Porto Alegre / São Paulo) mais um deslocamento (São Paulo/Rio de Janeiro). Usamos como exemplo estes dois segmentos, porém poderíamos ter escolhido qualquer cidade intermediária e uma única distância física pode ser descrita de várias maneiras.

No PC a um único endereço pode-se associar vários segmentos e deslocamentos. Por exemplo o endereço físico: 0EDA0h pode ser representado por:

segmento	+	deslocamento	= endereço físico
0E000h	+	DA0h	0EDA0h
0ED00h	+	0A0h	0EDA0h
0DF00h	+	EA0h	0EDA0h

Como a BIU identifica um único endereço efetivo a partir do registrador de segmento e o registrador de deslocamento ?

Existem 16 módulos possíveis de 64 Kbytes na memória física, ver Seção 2.5. A BIU faz 2 operações distintas, conforme Figura 3.6:

- A: multiplicar por 16 o registrador de segmento, deslocando 4 bits para esquerda, e acrescentando zeros nos 4 bits menos significativos do registrador de segmento
- B: somar o resultado com o registrador de deslocamento. O resultado é um número hexadecimal de 20 bits, representando o endereço físico efetivo.

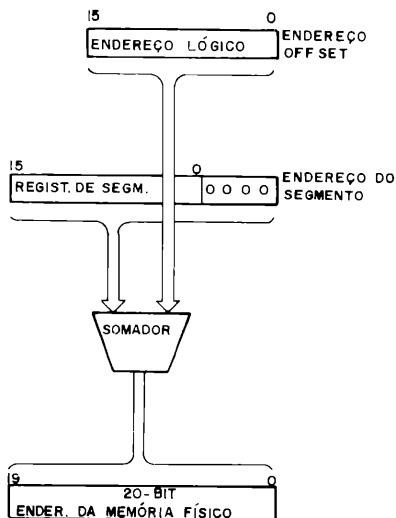


Figura 3.6 Geração do endereço de memória pelo 8088

O exemplo acima pode ser reescrito em termos do registrador de segmentos e registrador de deslocamento, CS:IP da forma:

CS = {0E00}	IP = {0DA0}	→ 0E00:0DA0	→ 0EDA0h
CS = {0ED0}	IP = {00A0}	→ 0ED0:00A0	→ 0EDA0h
CS = {0DF0}	IP = {0EA0}	→ 0DF0:0EA0	→ 0EDA0h

O DEBUG só indica os endereços na forma CS:IP, ver Cap. 7. Nem sempre a identificação do endereço efetivo é direto como nas duas primeiras representações do exemplo acima. A última representação envolve uma soma de números hexadecimais, nem sempre é familiar.

Após o sinal de RESET (+ 5V no pino 21 do 8088, ver Secção 3.4), a CPU assume a seguinte configuração: CS = FFFFh e IP = 0000h. Qual é a primeira instrução a ser executada pelo PC após um RESET? A resposta depende da instrução localizada no endereço:FFFF:0000 ou no endereço efetivo:

A: FFFF → FFFF0 e B: FFFF0 + 0000 h → FFFF0h.

Ou seja a CPU irá buscar no endereço FFFF0h no ROM-BIOS a primeira instrução a ser executada após um reset.

A BIU utiliza os registradores dependendo da operação a ser executada. A Tabela 3.1 associa ao tipo de atividade o registrador de segmento e de deslocamento que são utilizados com prioridade, ou exclusividade.

tipo de operação	segmento usual	segmento alternativo	deslocamento
busca de instrução	CS	exclusivo	IP
uso da pilha	SS	exclusivo	SP
operação com variáveis	DS	CS,ES,SS	endereço efetivo
strings/ origem	DS	CS,ES,SS	SI
strings/ destino	ES	exclusivo	DI

Tabela 3.1 Uso dos registradores em endereçamentos

Ver no Cap 7 exemplos com as instruções MOV, usando estes registradores.

A maior parte dos utilitários como o Macro-Assembler da Microsoft e os Compiladores C, tais como Turbo C, Quick C e Microsoft C (ver Cap. 6) possui instruções internas para alocação de segmentos e memória, deixando ao usuário transparente o modelo de memória e alocação de segmentos em uso.

3.4 Descrição funcional do 8088

O PC utiliza o 8088 no modo máximo, permitindo o uso do coprocessador 8087. Nesta Seccção vamos nos limitar as funções da CPU mais relevantes em projetos de interfaces do PC e nem todos os pinos da CPU da Fig. 3.1 são discutidos.

A Figura 3.7 resume os sinais de entrada e saída da CPU de interesse.

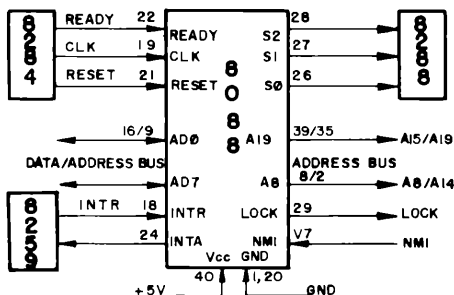


Figura 3.7 Descrição funcional da CPU 8088

O integrado temporizador programável, 8284 (ver Figura 2.4) não só fornece o clock de 4.77 Mhz para a CPU, como controla os sinais de RESET e READY.

- RESET:** Controla o início de operação do sistema através do nível de tensão de +5V. Quando o sinal de RESET é alto, a CPU (HALT) para qualquer execução e espera o sinal de RESET baixo. Assim o sistema é reinicializado com CS ← FFFFh e IP ← 0000h; (FFFF:0000). Este endereço FFFF0h está na área ROM BIOS, daí porque a arquitetura do PC aloca o topo da memória para ROM-BIOS.
- READY:** Gera os estados de "wait" no 8088 CPU, estendendo comprimento de um ciclo de "clock". Por exemplo, no interfaceamento de I/O, cujos dispositivos podem ser mais lentos que a CPU, o sinal de I/O CHRDY ("Input/Output Channel Ready") é traduzido pelo 8084 como uma solicitação de esperar ("wait") alguns ciclos de clock.
- NMI :** ("Non-Maskable Interrupt") No projeto do PC este sinal é destinado a receber interrupções de alta prioridade tais como as solicitadas por: coprocessador 8087, teste de paridade de memória e outros integrados da placa principal.
- INTR :** ("Interrupt") Trata-se da interrupção mascarável controlada pelo integrado 8259. Ver Cap 5.

AD0/

AD7 :

Transmitem e recebem dados da memória em cada ciclo de bus, conectados ao data bus. Esses pinos compartilham dados, no final do ciclo de clock, com endereços A0-A7, no início do ciclo de clock. Esta arquitetura multiplexada permitiu a INTEL projetar um microprocessador de 16 bits em um integrado padrão (40 pinos).

LOCK :

A CPU utiliza-se deste sinal para comunicação com o controlador de interrupção 8259. Processa o INTA ("Interrupt Acknowledge"), alocando o data bus para uso exclusivo do controlador de interrupção de modo a receber o vetor de interrupção, após receber uma solicitação de interrupção. Ver Cap 5.

S0,S1

e S2:

Sinais de "Status". Indicam o tipo de ciclo de bus em execução. No projeto do PC são conectados à entrada do 8288, onde são decodificados e definem o controle de bus. Como existem 2 estados possíveis para cada pino, o 8288 decodifica 8 tipos de ciclo de bus conforme Tabela 3.2

S2	S1	S0	tipo de ciclo de bus
0	0	0	solicitação de interrupção.
0	0	1	leitura port I/O
0	1	0	escrita port I/O
0	1	1	parado (Halt)
1	0	0	busca de instrução ("fetcbe")
1	0	1	leitura de memória
1	1	0	escrita de memória
1	1	1	passivo

Tabela 3.2 Sinais de Status do 8088

Como os primeiros 64Kbytes de endereços são comuns tanto a memória como os I/O port, são os bits de status através do 8288 que identificam qual é o tipo de dispositivo em uso. Ver Cap. 4 a descrição destes sinais no slot do PC.

BUS DO SISTEMA E SLOT

4.1 Caracterização do "slot"

A facilidade de adaptar novos periféricos ao PC é devido à existência dos 5 slots livres de expansão. Neles pode-se acrescentar desde uma expansão de memória até uma placa de transmissão e recepção de FAX ("Fax Board"). Esta facilidade permite projetar placas com tarefas específicas de I/O e contar com todo o potencial de processamento do PC.

A idéia de deixar slots de expansão foi usada originalmente nos microcomputadores APPLE. Diferente deste, onde a cada slot está associada uma região de memória, no PC os 5 slots são idênticos entre si. Esta versatilidade, em contrapartida, obriga o projetista a conhecer os endereços de I/O utilizados por outras placas, tais como impressora, jogos, serial e vídeo, ver Seção 4.3.

A Figura 4.1 representa os 62 pinos do slot de interfaceamento.

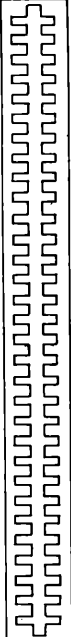
SINAL	PINO		PINO	SINAL
GND	B1		A1	I/O CHCH
RESET DRV	B2		A2	D7
+5V DC	B3		A3	D6
IRQ 2	B4		A4	D5
-5V DC	B5		A5	D4
DRQ 2	B6		A6	D3
-12V DC	B7		A7	D2
CARD SLCTD	B8		A8	D1
+12V DC	B9		A9	D0
GND	B10		A10	I/O CHRDY
MEMW	B11		A11	AEN
MEMR	B12		A12	A19
IOW	B13		A13	A18
IOR	B14		A14	A17
DACK 3	B15		A15	A16
DRQ 3	B16		A16	A15
DACK 1	B17		A17	A14
DRQ 1	B18		A18	A13
DACK 0	B19		A19	A12
CLK	B20		A20	A11
IRQ 7	B21		A21	A10
IRQ 6	B22		A22	A9
IRQ 5	B23		A23	A8
IRQ 4	B24		A24	A7
IRQ 3	B25		A25	A6
DACK 2	B26		A26	A5
T/C	B27		A27	A4
ALE	B28		A28	A3
+5V DC	B29		A29	A2
OSC	B30		A30	A1
GND	B31		A31	A0

Figura 4.1 Definição dos pinos e sinais do slot.

A maioria dos sinais do bus interno do PC, (Ver Cap.3), estão presentes no bus do slot. Porém existe uma diferença. Entre o bus local e o bus externo dos slots. Um conjunto de integrados desempenham as funções de latches, decoders e buffers, conectando o bus interno e externo. Por exemplo, o data bus (D0/D7), diferente do bus local, não compartilha os mesmos pinos com os 8 sinais de endereços menos significativos (A0-A7), ver Seção 3.4. Isto facilita o projeto de interfaces.

Neste Capítulo, discutimos os sinais de interesse, para projetos de interfaceamento com o PC, sem considerar os sinais de acesso direto à memória.

Os pinos dos slots estão classificados, na Tabela 4.1, de acordo com a função e a direção de entrada (do exterior para o PC) ou saída.

Função		Direção	Pinos
alimentação	GND(terra = 0V)	-	B1,B10,B31
	+ 5 V	-	B3,B29
	- 5 V	-	B5
	+ 12 V	-	B9
	-12 V	-	B7
memória	escrita (-MEMW)	saída	B11
	leitura (-MEMR)	saída	B12
I/O	escrita (-IOW)	saída	B13
	leitura (-IOR)	saída	B14
	canal I/O pronto, sinal de entrada baixo(+ I/O CH RDY)	entrada	A10
	checar I/O (-I/O CH CK)	entrada	A1
	Reset drive(+ RESET DRV)	saída	B2
DMA	DMA Request 1(+ DRQ1)	entrada	B18
	DMA Request 2(+ DRQ2)	entrada	B6
	DMA Request 3(+ DRQ3)	entrada	B16
	DMA Acknowledge 0(-DACK0)	saída	B19
	DMA Acknowledge 1(-DACK1)	saída	B17
	DMA Acknowledge 2(-DACK2)	saída	B26
	DMA Acknowledge 3(-DACK3)	saída	B15
	habilitação de endereço, alto quando DMA controla o bus (+ AEN)	saída	A11
clock	oscilador, f = 14.18MHz(+ OSC)	saída	B30
	clock, f = 4.77 MHz. (CLOCK)	saída	B20
Interrupção	Request, livre (+ IRQ2)	entrada	B4
	Request, livre (+ IRQ3)	entrada	B25
	Request, porta serial(+ IRQ4)	entrada	B24
	Request, disco rígido(+ IRQ5)	entrada	B23
	Request, disk-driver (+ IRQ6)	entrada	B22
	Request, paralela (+ IRQ7)	entrada	B21
Data bus	+ D0/ + D7, data bit	entrada/saída	A9/A2
Address bus	+ A0/ + A19, address bus	saída	A31/A12
	Address Latch Enable (+ ALE)	saída	B28

Tabela 4.1 Descrição funcional do slot

Legenda: sinal "+" significa ativado no estado alto
 sinal "-" significa ativado no estado baixo (terra).

A disponibilidade da alimentação ($\pm 5V$ e $\pm 12V$) no slot permite projetar interface sem usar fonte externa auxiliar, Ver Seção 1.2. Porém, existe um limite máximo de corrente que pode ser drenado pelo slot, conforme Tabela 4.2

ddp	corrente máxima em(mA)	pinos do slot
+ 5 V	700	B3, B29
-5 V	30	B5
+ 12 V	100	B9
-12 V	50	B7

Tabela 4.2 Corrente máxima disponível no slot

A corrente disponível é suficiente para a maioria dos projetos de interface para alimentar integrados CMOS (Complementary Metal Oxide Semiconductor), TTL (Transistor-Transistor Logic) e OP-AMP (Operational Amplifier) e relés. No caso deste último, é necessário verificar a corrente de chaveamento, o ideal é o uso dos relés encapsulados com consumo inferior a 100 mA.

Os sinais de escrita e leitura na memória e de I/O estão descritos na próxima Seção e os de interrupção no próximo Capítulo.

O PC tem uma série de recursos para implementação de sistemas DMA (Direct Memory Access) capazes de transferir dados de periféricos diretamente para a memória, sem interferência do microprocessador. Utilizando os processos de I/O usuais do PC, a taxa máxima de transferência de dados é de 30 Kbytes/s. Com o uso do DMA é possível ampliar esta taxa para até 476 Kbytes/s. O projetista deve levar em consideração não só a rapidez, como a capacidade de armazenamento na memória. Por exemplo, usando a taxa máxima de transferência através do DMA em menos de 1 segundo a memória livre do PC estaria esgotada.

As placas de interface a serem conectadas ao slot estão limitadas pelas dimensões do soquete e do gabinete do PC. A Figura 4.2 descreve as dimensões máxima de uma placa para o slot do PC e PC-XT (o AT é ligeiramente diferente).

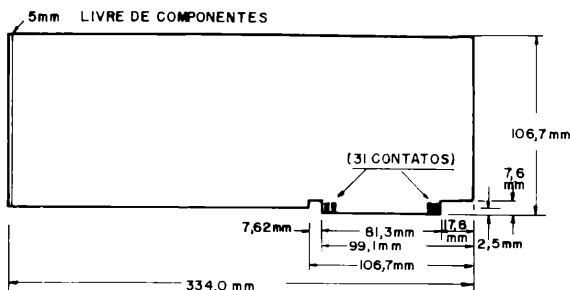


Figura 4.2 Dimensão máxima de uma placa interface

A nível de protótipo, é recomendável usar uma placa padrão com furos metalizados para fixação de soquete "wire-wrap", (por exemplo usamos a placa PCI 202, de Global Eletrônica Ltda, R. da Glória 465, Recife PE), deste modo o teste de viabilidade do projeto é feito rapidamente e a baixo custo.

4.2 Sinais de I/O e Memória

O princípio operacional de qualquer microprocessador reside na eletrônica digital. Embora nos referimos sempre a níveis de tensão alto e baixo na transmissão de informações de um circuito para outro, o que importa de fato é a capacidade de receber (baixa impedância ou nível baixo) ou fornecer corrente (nível alto). Num integrado TTL deixar um bit de entrada desconectado é o mesmo que colocá-lo em alta impedância, ou nível alto, impedindo a entrada de corrente. Disto decorre a necessidade de não deixar pinos de integrados TTL desconectados, conectando-os a V_{cc} (+5V) ou terra através de um resistor.

Os níveis alto (bit = 1) e baixo (bit = 0) definem uma margem de potencial, alto (2.5 V → 5 V), baixo (0 → 0.8 V) e indeterminado (0.8 V → 2.5 V). Estas margens permitem flutuações dentro desta faixa, devido a ruídos e outras indefinições, sem perda de informação associada ao status do bit. É interessante observar o sinal de clock (pino 19 do 8088 ou B20 do slot) e verificar quanto sua imagem real num osciloscópio difere dos diagramas de uma onda quadrada. Longe de uma transição súbita de alto/baixo ou vice-versa, o sinal real apresenta uma rampa em declive. Como a mudança de estado representa o transporte de uma informação, sempre existirá um tempo de atraso nesta operação. O tempo de atraso entre uma transição deve-se à capacitância externa e

interna, associado a cada porta lógica. Numa porta inversora TTL o atraso típico, entre a informação de entrada e o sinal de saída, é da ordem de 10 nano-segundos (10^{-8} s).

No circuito do PC, existem vários tipos de integrados, em geral padrão TTL LS, com velocidades de transição das mais diversas. É fácil imaginar a dificuldade do projetista em sincronizar todas estas atividades com os ciclos de clock do PC. A maior parte do tempo a CPU está enviando (escrevendo) ou recebendo (lendo) informação da memória. Para realizar a operação de escrita na memória a CPU deve controlar o bus de dados, de controle e de endereço para realizar as seguintes operações:

1. Selecionar um chip de memória de um dos bancos de memória.
2. Selecionar, estabilizar ou travar (latche) um endereço dentro do chip de memória.
3. Habilitar a memória para receber um dado.
4. Estabilizar o dado no data bus o tempo suficiente para ser captado pela memória.
5. Restabelecer o estado inicial.

O tempo mínimo do ciclo de escrita na memória é da ordem de 100 nano-segundos. Além deste tempo é necessário sincronizar um grande conjunto de decoders (74 LS 158), latches (74 LS 74, 74 LS 373) e buffers (74 LS 244, 74 LS 245) dispostos entre o barramento da CPU e do banco de memória, de modo que o tempo gasto numa instrução de escrita é da ordem de 840 nano-segundos. (Ver Figura 4.3).

A CPU 8088 demora quatro ciclos de clock para executar uma instrução de escrita na memória. Como cada ciclo de clock tem a duração de: $T = 1/f = 1/4.77 \text{ MHz} = 210$ nano-segundos, o tempo previsto pela CPU para realizar esta operação é 10×210 nano-segundos = 2.1 micro-segundos. Ora, como o controlador de bus 8288 gasta somente 840 nano-segundos, (ver Figura 4.3), para controlar todas as operações de escrita, existe uma folga entre o tempo gasto pelo CPU e o previsto no projeto do PC. Disto decorre a possibilidade de usar um clock mais rápido para o PC, sem necessidade de alterar a configuração original. Tal é o caso do PC-XT Turbo, cuja frequência de clock é mais alta, $f = 8 \text{ MHz}$.

Diferente da memória que tem um tempo bem definido de acesso, é impossível prever o tempo gasto com operações de escrita e leitura em periféricos de I/O, por exemplo o tempo de chaveamento de um relé mecânico é muito superior ao de um transistor. A solução encontrada pelo projetista foi permitir ao PC atrasar alguns ciclos

de clock (no máximo 10) através de controle externo, sinal I/O CH RDY pino A10 do slot.

Isto acrescenta estados de espera ("wait") extras no ciclo de escrita e leitura de I/O. Em qualquer projeto de I/O, deve-se levar em conta o sincronismo das operações do bus do PC e do periférico externo projetado. Para evitar esta dificuldade é possível conectar uma porta paralela programável tipo latch (PPI- 8255), capaz de controlar os sinais através de programas. Este projeto, útil em alguns casos, diminui a velocidade do processo como um todo.

A Figura 4.3 descreve os sinais de bus durante o processo de escrita/leitura na memória.

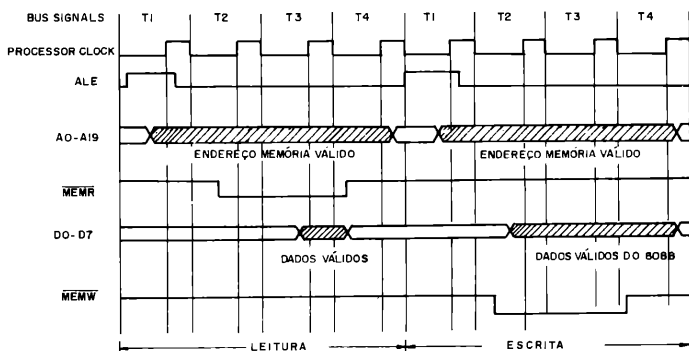


Figura 4.3 Ciclo de bus de leitura/escrita na memória

As 5 operações descritas acima estão presentes durante o ciclo de escrita nos sinais de bus. Estes sinais são controlados pelo 8288 a partir dos sinais de status de operação enviados pela CPU, ver Seção 3.4. Embora os tempos de clock ($T = 210$ nano-segundos) estejam representados como uma onda quadrada, de fato as transições estão em active, ou declive, e os tempos devem ser tomados a partir do ponto médio da transição alto/baixo ou vice-versa. As etapas de escrita podem ser resumidas como:

1. Durante T1 (primeiro ciclo de clock), 8288 envia a mensagem de ALE (Address Latch Enable), indicando que a CPU pretende usar o bus de endereço para alguma operação.
2. Antes do término do sinal de strobe ALE, o bus de endereço é estabilizado através dos latches entre a CPU e o bus de sinal, e deverá permanecer neste estado até o final.
3. No início do período T2, o 8288 já decodificou o status de instrução, (neste caso: S0 = 0, S1 = 1, S2 = 1, escrita na memória) e envia o sinal de baixo para o MEMR ("Memory Read"), desabilitando simultaneamente qualquer processo de escrita na memória através de MEMW ("Memory Write") alto. Evitando ambiguidade com uma porta I/O de mesmo endereço, qualquer processo de escrita ou leitura de I/O estão desativados, colocando IOR ("I/O Read") e IOW ("I/O Write") em nível alto. Neste ponto, um endereço específico de memória, decodificado por (A0-A15), seleciona uma memória, de um dos bancos (A16-A19), habilitando-a para receber os dados a serem escritos. (pino WR da memória baixo, pino RD alto, e chip select CS baixo)
4. Enquanto os períodos T1 e T2 foram necessários para preparar a escrita na memória, o próximo será destinado a transferir os dados do data bus para o barramento de dados da memória. A memória gasta um tempo mínimo de 300 nano-segundos para gravar a informação (o tempo de leitura é bem menor, da ordem de 50 nano-segundos), ou seja T3 mais a metade de T4.
5. Como T3 foi tempo suficiente para o processo de escrita, no início de T4 MEMR é desativado, estado alto, e o bus de dados e de endereços estão liberados para a execução da próxima instrução.

O ciclo de leitura é semelhante ao de escrita, exceto quanto a ativação do sinal de leitura (MEMR baixo) e desabilitação do sinal de escrita (MEMW alto).

A Figura 4.4 representa o ciclo de bus de leitura e escrita de um periférico I/O.

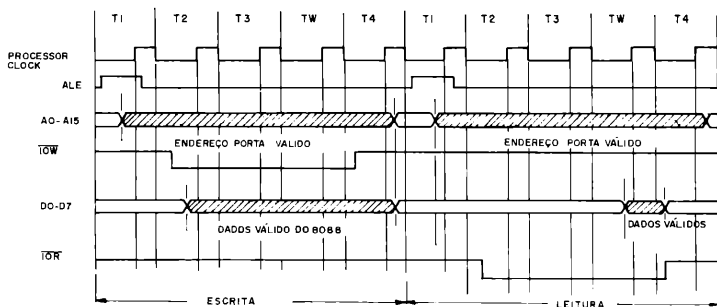


Figura 4.4 O ciclo de bus leitura/escrita de I/O.

Um ciclo de leitura (ou escrita) de I/O é iniciado quando a CPU identifica uma intrusão IN (OUT), ver Cap.7. Esta intrusão tem por objetivo ler (ou escrever) dados numa porta de I/O no espaço destinado pelo PC para esta função, ver no Cap.8 mapeamento de I/O do PC.

O ciclo de bus de leitura(ou escrita) de I/O gasta no mínimo 5 intervalos de clock, ou 1.05 micro-segundos para $f = 4.77$ MHz. A duração deste ciclo pode ser controlada externamente, ativando o sinal I/O CH RDY ("Ready bus") em estado baixo. Esta informação é traduzida pela CPU como um sinal de espera ("wait"), ou seja acrescenta ciclos de clock extras na execução da instrução. Porém, existe um limite máximo da elongação do estado de espera de 10 ciclos de clock, ou 2.10 micro-segundos.

Exceto quanto ao estado de espera ("wait"), o raciocínio de escrita/leitura de I/O é semelhante ao da memória.

4.3 Mapeamento da área de I/O

Existe uma diferença fundamental entre os endereços de I/O e memória. O 8088 só aceita endereços de I/O para os 16 primeiros bits (A0-A15), limitando a 64 Kbytes o número máximo de endereços I/O acessíveis. Entretanto o projeto do PC restringe ainda mais este número, ao conectar somente os 9 bits de endereços para área de I/O (A0-A8), portanto somente 512 diferentes "ports" são acessíveis no PC.

A disputa deste espaço é maior a cada introdução de um novo periférico, tal como a placa gráfica EGA e o "mouse".

Uma atitude prudente consiste em projetar interfaces que usam um espaço de I/O livre, ou que raramente são utilizados, tal como a entrada de jogos (200...20Fh) ou da segunda impressora (270...27Fh). Do contrário é melhor limitar-se à área reservada à instalação de protótipos para o usuário (300...31Fh).

A Tabela 4.1 resume os principais endereços de I/O usados na versão PC/XT

endereço de I/O	uso no PC
00-0Fh	controlador de DMA (8237)
10-1Fh	utilizado pela placa principal
20-2Fh	controlador de interrupção 8259
30-3Fh	não documentada
40-5Fh	temporizador ("timer") 8253
60-6Fh	porta paralela PPI 8255
70-7Fh	não documentada
80-9Fh	registrador de página do DMA
A0-AFh	registrador de máscara da NMI
B0-1FFh	não documentada ou reservada
200-20Fh	adaptador de jogos
210-21Fh	unidades de expansão
220-26Fh	reservada ou não documentada
270-27Fh	porta paralela de impressora 2
280-2EFh	reservada ou não documentada
2F0-2FFh	porta serial 2
300-31Fh	placa de protótipo, livre para uso
320-32Fh	adaptador de disco rígido
330-36Fh	não documentada
370-37Fh	impressora paralela principal
380-3AFh	não documentado ou reservado
3B0-3BFh	placa de monitor monocromático MDA
3C0-3Cfh	placa de vídeo EGA
3D0-3DFh	placa de vídeo CGA
3E0-3EFh	reservado
3F0-3FFh	disco flexível e porta serial principal(3F8h)

Tabela 4.3 Mapeamento da área de I/O

Por reservado ou não documentado, entende-se uma área em que o fabricante usa, ou pretende usar, em futuras implementações. Os endereços usados pelos dispositivos instalados no PC estão, em geral, na área de dados da BIOS, ver Seção 9.3

INTERRUPÇÃO POR HARDWARE

5.1 Ciclo de interrupção.

Se existe algo de revolucionário na família 8086, é sua capacidade de manipular interrupções. Longe de seu parente mais próximo, o 8085 com somente 4 tipos de interrupções, o 8086/8088 pode manipular até 256 diferentes tipos de interrupções. Esta característica de hardware possibilitou o uso intensivo em programação das rotinas de interrupções. Uma classificação destas interrupções está na Tabela 2.2 e o ciclo de uma interrupção genérica está esboçado na Figura 2.8.

Na arquitetura do PC, as interrupções por software e hardware são tratadas de modo equivalente, ressaltando a ordem de prioridade definida pelo controlador de interrupção por hardware, PIC- 8259 ("Programmable Interrupt Controller")

Quando uma interrupção é solicitada, e aceita, ocorre a seguinte sequência de eventos:

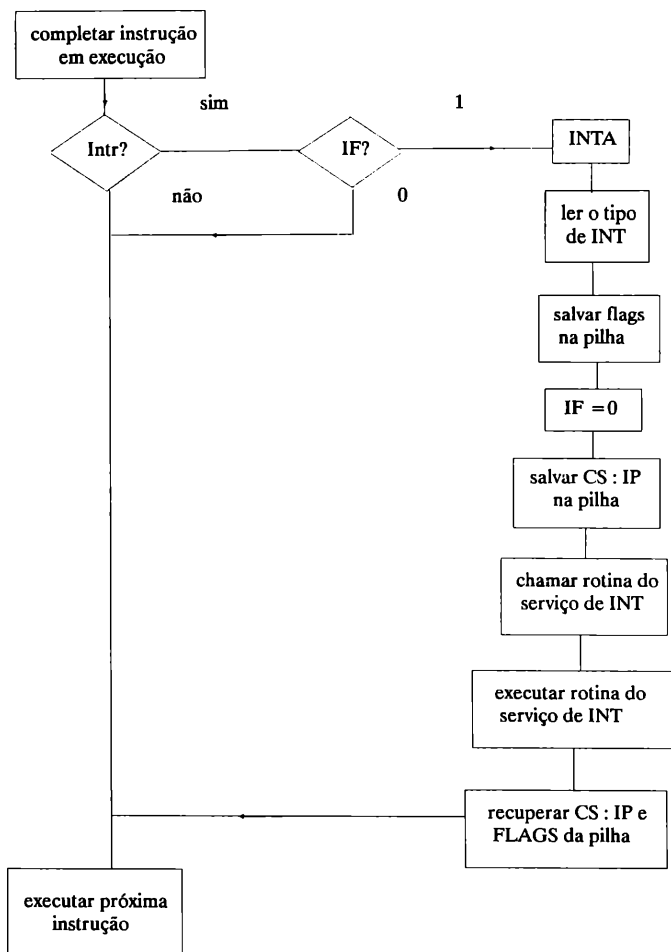


Figura 5.1 Ciclo de uma interrupção mascarável

Neste diagrama, não constam as interrupções não-mascaráveis (NMI) nem a execução de intrusão passo a passo ("Trace").

Ao finalizar uma instrução, a CPU verifica se alguma interrupção foi solicitada. Caso positivo testa se o flag de interrupção IF (ver Seção 3.2), programável (através de CLI e STI, ver Cap.7) está habilitado (IF=1). Caso contrário (IF=0) ela ignora a interrupção solicitada. Ao iniciar a instrução de interrupção, a CPU ativa o protocolo de interrupção, INTA ("Interrupt Acknowledge") para conhecer o tipo de interrupção (ver Figuras 5.2 e 3.7). Quando se trata de uma interrupção por software, esta etapa é dispensável. O tipo de interrupção será capturado pela CPU no data bus (00h-FFh), daí restrição de 256 tipos de interrupções. Conhecendo o tipo de interrupção, a CPU lê na tabela dos vetores de interrupção, (memória RAM 000h -- 100h, ver Seção 2.8), o endereço da rotina do serviço de interrupção. Como é necessário preservar a integridade do programa principal, a CPU automaticamente salva na pilha ("STACK") o registrador de flag de status (ver Seção 3.2) e o endereço da próxima instrução, (definido pelo apontador CS:IP, ver Seção 3.3).

Quando a CPU estiver executando uma interrupção, ela desabilita todas as outras instruções mascaráveis, colocando zero no flag de interrupção (IF=0). Este fator de proteção possibilita a execução integral de um serviço de interrupção. Porém, obriga o programador a ser cauteloso na implementação de sua rotina de interrupção. Um defeito de programação pode deixar o microcomputador em pane ("crash"). A leitura do teclado também é uma rotina de interrupção mascarável (ver próxima Seção). Este problema só pode ser resolvido desligando e ligando o PC, quando após um Reset todos os vetores são reinicializados. Ou seja, durante a execução de uma rotina de interrupção, as teclas que desviam o fluxo de um programa (CTRL + C, CTRL + ALT + DEL, CTRL BREAK) ficam desativadas.

Após esta inicialização, a CPU passa a executar o programa cujo endereço inicial encontra-se definido pelo vetor de interrupção. Por exemplo na Seção 2.8, verificamos que o vetor de interrupção de relógio (INT 1Ch, portanto no endereço 70h da tabela), aponta para o endereço: (CS=F000:IP=FF53). Na Seção 7.4 mostramos como este programa pode ser desassemblado através do DEBUG. A rotina de interrupção deve ter como última instrução um retorno de interrupção, IRET (Interrupt Return), senão a CPU continuará executando outras instruções e pode perder a posição do endereço de retorno na pilha colocando o microcomputador em pane. Encontrando o término da rotina de interrupção, a CPU retira da pilha (POP "Pop Word of Stack") o registrador de status de flag e o endereço da próxima execução a ser executada (CS:IP) que foi armazenada na pilha provisoriamente. Com o novo endereço a CPU salta de volta ao programa principal.

5.2 Controlador de Interrupção por hardware (8259)

O diagrama de pinos do 8088 (ver Figura 3.1) mostra que só existe um pino destinado à interrupção por hardware (INTR, pino 18). Evidentemente isto é pouco, a solução de projeto do PC foi acrescentar um circuito integrado PIC-8259 capaz de ampliar para 8 o número de possíveis interrupções, além de estabelecer uma prioridade entre elas, conforme Figura 5.2.

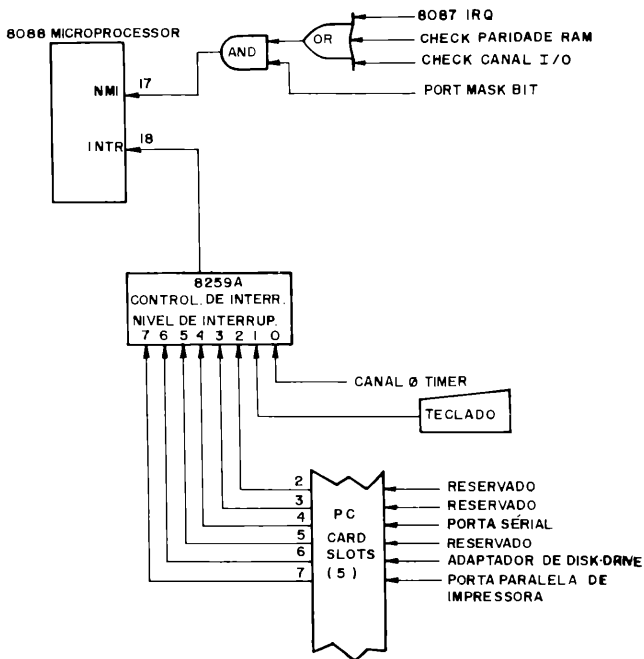


Figura 5.2 Diagrama das Interrupções por hardware.

O 8259 é programável e sua tarefa é traduzir uma das 8 solicitações possíveis de interrupção dos periféricos externos a ele conectados, através dos pinos de slot (ver Seção 4.1), de modo a ativar a única linha de interrupção do 8088. A maior parte das interrupções previstas no projeto já estão destinadas a dispositivos específicos conectados ao slot no sistema básico (porta paralela de impressora, porta serial, disk-drive, disco rígido, etc..), restando somente uma (IRQ 2) ou duas (IRQ 3) interrupções livres para uso. Contudo é possível seguir na linha do projetista original e expandir uma destas linhas reservadas ao usuário em outras 8, conectando mais um 8259 externo.

O PIC-8259 é programado durante o "boot" através do programa "invisível" IBMBIO (ver Seção 2.4.1) juntamente com a tabela dos vetores de interrupção. Os endereços destas rotinas dependem das configurações de hardware do fabricante e da versão DOS em uso e podem ser visualizados através do DEBUG, ver Seção 9.3. Os registros de controle do 8259 estão no endereço de I/O 0020h e 0021h, (ver Tabela 4.3, Seção 4.3). Aconselha-se a não reprogramar o 8259, pois isto pode alterar a tabela dos vetores de interrupção e levar o PC ao estado de pane.

Como o PIC-8259 consegue traduzir 8 diferentes interrupções de hardware em um ciclo de interrupção? A Figura 5.3 é um diagrama dos sinais de comunicação entre a interrupção solicitada por um periférico e a CPU através do 8259.

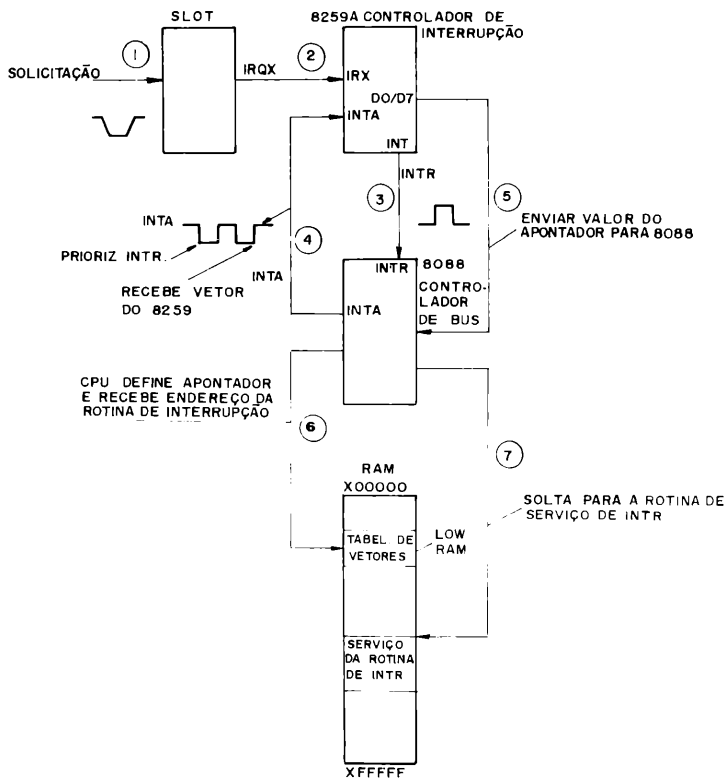


Figura 5.3 Diagrama de interrupção externa através do 8259

O ciclo de interrupção compreende os passos:

- (1) Dispositivo externo solicita uma interrupção (sinal de strobe baixo em uma das 8 entradas do 8259)
- (2) Definido por hardware (ver Tabela 5.1), esta interrupção está associada a um dos níveis de prioridade estabelecidos pelo 8259. A informação de que esta interrupção foi solicitada é preservada pelo 8259 ("latch") caso a CPU esteja ocupada.
- (3) O controlador 8259 solicita uma interrupção a CPU (INTR)
- (4) A CPU define o status de execução de uma interrupção ($S2 = S1 = S0 = 0$, ver Tabela 3.2). O controlador de bus 8288 decodifica esta informação e envia 2 pulsos de protocolo INTA (interrupt acknowledge), solicitando definição a respeito de qual tipo de interrupção foi solicitada.
- (5) O controlador coloca no bus de dados (D0-D8) o endereço do vetor de interrupção indicado pelo tipo de interrupção.
- (6) A CPU, reconhecendo o tipo de interrupção procura na tabela do vetor de interrupção (20h--3Ch) o endereço da rotina do serviço de interrupção.
- (7) Capturando o endereço da rotina de interrupção a CPU executa um ciclo de interrupção tal como descrito na Figura 5.1

No PC, as 2 interrupções mais prioritárias são para as funções internas, o relógio (IRQ 0, INT 8) e o teclado (IRQ 1, INT 9h), ao passo que as restantes estão disponíveis no slot. A Tabela 5.1 identifica os vetores de interrupção controlados pelo 8259 e os periféricos a eles associados.

INT (TIPO)	Tabela Vetor	IRQ	8259 D3	DATA BUS D2 D1 D0	Função
08h	20h	0	1	0 0 0	relógio, canal 0 / interno
09h	24h	1	1	0 0 1	teclado / interno
0Ah	28h	2	1	0 1 0	livre para uso / slot
0Bh	2Ch	3	1	0 1 1	livre p/uso(serial2)/slot
0Ch	30h	4	1	1 0 0	porta serial-RS 232 / slot
0Dh	34h	5	1	1 0 1	disco rígido / slot
0Eh	38h	6	1	1 1 0	disk-drive / slot
0Fh	3C	7	1	1 1 1	porta paralela(impr)/slot

Tabela 5.1 Interrupções controladas pelo 8259

Os 4 bits mais significativos ($D7=D6=D5=D4=0$) são nulos. Utilizando o DEBUG e a TABELA VETOR de interrupção, é possível rastrear as rotinas de interrupção do BIOS e do DOS para cada interrupção. (Ver Cap. 7).

O usuário pode redefinir suas próprias rotinas de serviço interrupção, usando por exemplo IRQ 2 ou IRQ 3, sem necessidade de reprogramar o PIC 8259 ou alterar as rotinas já existentes. Basta interceptar o vetor de endereço de rotina de interrupção da TABELA VETOR e orientá-lo para o início da rotina do usuário. Na Seção 11.3 mostramos um exemplo de como isto pode ser feito, ver Exemplo 11.3.

Além das interrupções livres para uso, as de relógio e teclado são de interesse para programação em tempo real. Não temos acesso direto à interrupção do relógio, IRQ 0 (ou INT 8). A cada 55 milisegundos aproximadamente o canal zero do temporizador aciona o IRQ 0 para a realização de uma série de rotinas BIOS, tais como monitorar o drive e atualizar o relógio interno do PC, ver Seção 2.6. Além destas funções o projetista previu uma nova interrupção BIOS, INT 1Ch (não documentada) em conjunto com IRQ 0. Em geral, esta interrupção está inativa, contendo uma simples instrução de retorno da interrupção (IRET), tal como rastreamos no Cap. 7 através do DEBUG.

Como usar a base do tempo do PC para programação em tempo real? Para tempos superiores a 55 ms é possível usar a INT 1Ch como marcador de tempo, de modo idêntico à atualização do relógio interno do PC. Basta interceptar vetor de interrupção localizado em 70h ($4 \times 1Ch$ ver Seção 2.8) orientando-o para o endereço de uma rotina por nós definida.

Uma interrupção de teclado (IRQ 1 ou INT 9) é executada cada vez que uma tecla é pressionada ou liberada. Na parte de dados da BIOS, em 00400h (ou 0040:0000), localiza-se o "buffer" de dados do teclado. Esta rotina de interrupção é bastante complexa e executa vários procedimentos, tais como: rastrear ("scan") o teclado e identificar qual tecla foi movimentada, verificar o status das teclas especiais (Ctrl, Shift, Caps Lock, Num Lock, Delete, Esc, etc...), atualizar os dados no "buffer" de teclado.

Programando em tempo real, ou em programas que ficam residentes, é útil a introdução de teclas definidas pelo usuário ("hot key") com a finalidade de executar uma tarefa programada sempre que determinada tecla for acionada. Neste caso, é melhor manipular o teclado através de linguagens de alto nível (tal como INKEY\$ em BASIC, KeyPressed em PASCAL ou kbhit() em C). Em Assembly é mais conveniente usar as funções de interrupções BIOS (INT 16h ou 1Bh, ver Cap.9) ou funções DOS (INT 21, funções 0Ah, 0Bh e 0Ch ver Cap. 10).

LINGUAGENS DE PROGRAMAÇÃO

**"El número de símbolos ortográficos
es veinticinco... Esa comprobación
permitió decifrar la naturaleza informe
y caótica de casi todos los libros"
(La Biblioteca de Babel, J.L.Borges)**

6.1 Introdução

Não existe uma separação nítida entre hardware e software em projetos de interface com microcomputador. O projetista deve dominar os aspectos básicos de ambos.

Este livro não é sobre programação no PC. Limitamo-nos a explorar as funções e rotinas de maior utilidade em projetos de interface. Não é necessário conhecer as linguagens de programação ASSEMBLY ou C para acompanhar os programas apresentados. O leitor deve conhecer o sistema operacional, DOS versão 2.0 ou posterior, e possuir um PC (XT ou AT), na configuração mínima. Os programas exemplos apresentados não são do tipo dicas e truques. Procuramos apresentar os recursos mais interessantes existentes no PC, de modo a facilitar ao leitor construir suas próprias rotinas. Esta parte será mais proveitosa, caso o leitor pratique no PC os exercícios apresentados. Para tanto só é necessário um disco com o utilitário DEBUG.COM. Os programas são apresentados em módulos curtos comentados e documentados. Porém as instruções completas do 8088, as interrupções BIOS e DOS estão apresentadas nos Apêndices.

6.2 Linguagem de Máquina

Qualquer linguagem de programação só pode ser executada através de um código único associado a cada tipo de microprocessador, a linguagem de máquina. Afinal, o microprocessador não passa de um conjunto de diodos, transistores, resitores e capacitores alimentados com energia elétrica. O que é fantástico neste processo é sua

1. Álgebra de Boole

$$C = A \cdot B$$

A	B	C (A AND B)
0	0	0
1	0	0
0	1	0
1	1	1

Tabela verdade AND

2. Teoria dos conjuntos

$$C = A \cap B$$

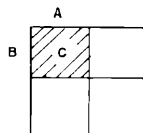
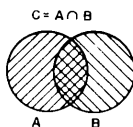


Figura 6.1 Diagrama de Venn e Mapa de Karnaugh AND.

3. Operação analógica

Circuito elétrico associação em série.

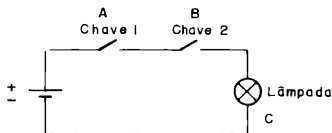


Figura 6.2 Circuito elétrico analógico AND.

CHAVE 1	CHAVE2	LÂMPADA
desligada	desligada	desligada
ligada	desligada	desligada
desligada	ligada	desligada
ligada	ligada	ligada

Operação AND analógica

A lâmpada C só estará acesa se ambas as chaves A e B estiverem ligadas. Antes da eletrônica digital, os computadores analógicos eram um conjunto de chaves analógicas. Imagine a complexidade da operação desses computadores.

4. Eletrônica Digital, nível TTL.

A configuração de 2 diodos e resistores equivale à porta lógica AND, representada e simbolizada como

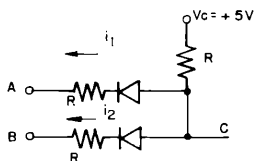


Figura 6.3 Configuração de diodos e símbolo para AND.

Em A e B, podemos aplicar dois estados de tensão possíveis alto (2.5 → 5 V) ou baixo (0 → 0.8V), ou mais precisamente +5V ou terra. A diferença de potencial na saída C ("output") é função dos sinais de entrada ("input") A e B.

O diodo só permite corrente elétrica num sentido, tal como indicado por i_1 e i_2 , para a esquerda. A tabela verdade pode ser representada por:

estados de A	estados de B	estados de C
0 V, $i_1 > 0$;baixo 0	0 V, $i_2 > 0$;baixo 0	0 V, baixo, 0
+5 V, $i_1 = 0$;alto 1	0 V, $i_2 > 0$;baixo 0	0 V, baixo, 0
0 V, $i_1 > 0$;baixo 0	+5 V, $i_2 = 0$; alto 1	0 V, baixo, 0
+5 V, $i_1 = 0$;alto 1	+5 V, $i_2 = 0$;alto 1	+5 V, alto, 1

Tabela 6.2 Operação lógica AND digital.

O Diagrama da Figura 6.3 representa um circuito físico de uma operação lógica AND, difere de uma montagem TTL (Transistor-Transistor-Lógic) porque as correntes estão chaveadas através de diodos (DDL - Diode-Diode Logic), porém representam uma situação física que ocorre a nível de microprocessador. É possível montar a configuração da Figura 6.3 usando 2 Leds ("Light-Emitting Diode"), resistores (qualquer valor abaixo de 1 K ohm e acima de 100 ohm), uma fonte de + 5V (pode ser 5 pilhas de 1,5 V em série) fios de ligação e um voltímetro (a tensão em C pode ser visualizada através de um outro diodo, ligando C a terra através de um resistor).

No microprocessador, existem milhares de transistores e diodos como os representados na Figura 6.3. cuja função é realizar operações lógicas e aritméticas entre os bits. Quando a CPU reconhece um código ela solicita a ALU executar uma função que fornece uma saída C a partir dos estado das entradas A e B.

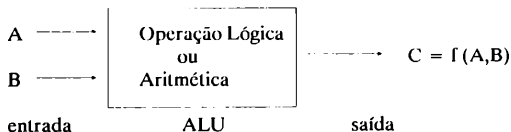


Figura 6.4 Função lógica

As entradas são bytes(8 bits) ou words(16 bits) localizadas ou nos registradores ou na memória, a saída também será através dos registradores ou armazenando os resultados na memória. A ALU contém o circuito lógico capaz de realizar a função lógica e aritmética $f(A,B)$. Esta função está codificada na CPU em notação binária.

Afinal como traduzir o namoro de João e Maria numa instrução em linguagem de máquina no PC.? Um programador poderá descrever este problema num código de alto nível (BASIC, Pascal ou C), do tipo :

IF A AND B then C = "namoram" (BASIC)

IF (A AND B) Then C: = True; (Pascal)

IF (A & B) C; (C)

Entre o problema geral e estas expressões, o programador define uma interface relacionando as variáveis abstratas A, B e C e a situação concreta.

O código de alto nível está ainda longe de ser compreendido pela CPU. Um programa especial, o compilador, específico para cada linguagem de programação, deverá traduzir esta expressão em termos de uma operação com os registradores e a ALU, no código de máquina. O compilador poderá, por exemplo, reescrever estas expressões na operação:

entrada :	Conteúdo do registrador AX Conteúdo do registrador BX
operação:	AND AX,BX (AX ← AX AND BX)
saída :	Conteúdo do registrador AX.

Para execução desta operação, em algum momento o apontador de programa (CS:IP) deverá se defrontar com a instrução:

CS:IP 89D8 (2 bytes em hexadecimal)

Pode-se programar diretamente em código de máquina, colocando num endereço de memória o número hexadecimal, que nada mais é que notação compacta de números binários, correspondente a cada instrução. Este método é bastante primitivo e pouco eficiente no PC. Algumas revistas ainda procuram escrever programas de modo compacto simplesmente introduzindo dados hexadecimais na memória (do tipo DATA " " em BASIC). Em microcomputadores de 8 bits, esta prática já era um método precário, no PC é um contra-senso.

Usando o DEBUG, é possível introduzir esta operação em linguagem ASSEMBLY (ver Cap.7), através de :

A > DEBUG

- a

CS:0100 AND AX,BX; entrada A = AX, B = BX, C = A ^ B
CS:0102

- u 100

CS:0100 89D8 AND AX,BX

CS é um valor qualquer alocado para o segmento de código pelo DEBUG. Na primeira etapa o DEBUG monta o programa a partir da linguagem Assembly "A" depois desassembla "U" (unassembly) montando nos dois bytes de endereço (CS:0100 e CS:0101) o código de máquina 89D8.

O escritor transmite todas as emoções e raciocínios, usando somente 25 símbolos (22 letras do alfabeto, o ponto, a vírgula e o espaço em branco). Ao programador compete reproduzir a lógica do raciocínio, usando somente 2 símbolos, o zero e o 1.

6.3 Linguagem ASSEMBLY 8088

É muito mais fácil decorar uma expressão do tipo:

AND AX,BC do que um número de código hexadecimal 89D8. Por isto cada instrução do 8088 está associada a um código mnemônico (do grego, *mnemonikós*, relativo à memória) de mais fácil memorização. Esta linguagem é a de mais baixo nível, ou seja a mais próxima do código de máquina e é específica a cada microprocessador.

Esta simplificação tem um preço. É necessário usar um montador (ASSEMBLER) capaz de traduzir os símbolos mnemônicos em códigos de máquina. No exemplo acima, usamos o comando "A" (assembly) do DEBUG para montar a instrução :A AND B.

De fato, os montadores comerciais realizam muito mais do que uma simples montagem, pois são capazes de processar rótulos e definir variáveis. Os montadores mais utilizados no PC são: Turbo Assembler 1.0 (Borland, preço estimado em U\$ 100) e MacroAssembler Microsoft , MASM 5.1 (Microsoft, preço estimado U\$ 100). Estes montadores não necessitam de disco rígido e podem fazer parte de um único disco flexível. Porém, é praticamente impossível usá-los sem o manual do usuário e a documentação fornecida pelo fabricante.

Neste livro não usamos montadores Assembler profissionais. Para tanto, seria necessária uma apresentação detalhada muito além do espaço disponível. A solução foi usar um poderoso utilitário desenvolvido pela Microsoft, que acompanha o sistema operacional DOS desde a versão original DOS 1.0, o DEBUG.COM. ("debug" é uma gíria inglesa de eletrônica que significa eliminar defeitos de aparelhos, literalmente

retirar insetos, "bug"). Mais do que um simples montador, este utilitário é um monitor capaz de analisar qualquer programa, com o poderoso recurso de desassemblar o código de máquina ("unassemble"), ou seja converter código de máquina em linguagem ASSEMBLY. Com ele podemos introduzir programas em código ASSEMBLY e criar arquivos executáveis tipo *.COM.

6.4 Arquivos tipo *.COM e *.EXE

O programa montado em código de máquina, denominado programa objeto, é executável diretamente a partir do DOS e pode ser de dois tipos: *.COM e *.EXE. O DEBUG produz arquivos objeto tipo *.COM, ao passo que os montadores ASSEMBLER e compiladores das linguagens de alto nível produzem arquivos tipo *.EXE. O programa mais importante do DOS é do tipo *.COM, COMMAND.COM, cuja tarefa é inicializar o sistema e fica residente à espera de comandos solicitados via teclado (DIR, DEL, etc...).

Atualmente, *.EXE é o tipo natural para programas executáveis. O tipo *.COM foi projetado originalmente para ser compatível com o sistema CP/M embora isto não seja mais necessário, ele continua existindo. Ele é mais simples que o tipo *.EXE e pode ser transferido mais rapidamente do disco para memória. Talvez por isto os utilitários DOS sejam deste tipo, embora para o usuário esta diferença de tempo não seja perceptível. O MASM usa o utilitário DOS "EXE2BIN" para conversão de arquivos tipo *.EXE em tipo *.COM. As principais diferenças entre estes tipos estão sumarizadas na Tabela 6.3

Atributos	*.COM	*.EXE
número de segmentos	somente 1	múltiplos segmentos
segmentos de referência	nenhum	permite definição
segmento de pilha	(pré definido em 256 bytes)	deve ser definido
origem do código	ORG em 100h	não necessita ORG
tamanho do programa	menor que 64K	arbitrário
bloco de memória alocado	toda a memória	a ser dimensionado

*Tabela 6.3 Diferenças entre *.COM e *.EXE*

A maior limitação dos programas tipo *.COM reside no tamanho da pilha e na impossibilidade de definir um segmento para dados. A delimitação do programa em 64K não é séria em pequenos programas.

Nossos programas em Assembly serão montados com o DEBUG que produz programas executáveis *.COM.

6.5 Linguagens de Alto Nível

A linguagem Assembly é a mais próxima ao código de máquina e executada com maior rapidez. Porém tem o inconveniente de obrigar o programador a escrever extensos programas para rotinas simples.

As linguagens de alto nível possuem uma interface mais simples com o programador, porém quando compiladas produzem códigos objetos mais longos e de execução mais lenta. O ideal é mesclar estas duas opções, desenvolvendo o programa principal em linguagem de alto nível e enxertando programas em Assembly em trechos onde a rapidez for um fator crucial.

O modo de execução de um programa fonte em BASIC pode ser interpretado ou compilado. No primeiro, o interpretador lê um código fonte e executa suas instruções linha após linha. O compilador transforma o texto em código ASCII do programa fonte em um programa objeto em linguagem de máquina do tipo *.EXE executável. O programa fonte pode ser editorado com editor de textos, com todos os comentários necessários, pois eles não constarão do programa executável, tal como no modo interpretado. O desenvolvimento de um programa compilado compreende os seguintes passos:

- (1) Editorar programa fonte através do editor texto.
- (2) Compilar o programa fonte em programa objeto
- (3) Aglutinar os programas da biblioteca ("link")

Nos utilitários mais recentes estas 3 etapas podem ser desenvolvidas em único ambiente, como por exemplo no Turbo PASCAL e Turbo C 2.0. A maior parte dos erros de sintaxe são detectados durante a compilação ("compile-time errors"). Mesmo quando o programa objeto esta livre de erros de compilação, podem aparecer erros no momento de unificá-lo com os programas de biblioteca ("linker errors"). Finalmente, ao executar o programa podem surgir erros de execução ("run time errors"). Infelizmente, estes erros ocorrem com maior frequência que gostaríamos.

As linguagens de alto nível mais comum no PC são: BASIC, PASCAL e C. O BASIC é a mais popular porém a menos estruturada. Existem diversas implementações de BASIC para o PC. No PC original, parte da ROM (ver Seção 2.5.1) era destinada ao BASIC interpretado (complementado posteriormente com extensões em disco BASICA.COM e GWBASIC.COM), semelhante ao Applesoft Basic em micros APPLE.

Nos modelos de PC nacionais esta parte da ROM, em geral, está vazia e os fabricantes incluem um **BASICA.COM** em disco.

O BASIC compilado pode ser o primeiro passo para a programação profissional. As versões mais recentes do tipo QuickBasic 4.5 (Microsoft, custo aproximado de U\$70,00) e o TurboBasic (Borland, custo aproximado de U\$70,00) são ferramentas avançadas de programação em Basic. Além de introduzir recursos de linguagens mais estruturadas (do tipo "while", rótulos, etc..) são capazes de manipular os recursos gráficos do PC de modo direto e simples. Estas versões são recomendáveis ao engenheiro que domina a linguagem BASIC e necessita implementar seus programas rapidamente.

Dentre as linguagens estruturadas profissionais PASCAL é a mais popular. Ela permite produzir programas estruturados legíveis e sem erros. O Turbo Pascal na sua versão mais recente 5.5 (Borland, preço aproximado de U\$120) é bastante atraente pelo ambiente integrado de trabalho e o acoplamento com o Debugger.

A linguagem C pode ser colocada como uma linguagem de nível intermediária. Ela oferece as ferramentas para uma programação estruturada, tal como o PASCAL, porém permite a manipulação de bits individualmente, tal como o Assembly. Outro fator do C é a sua portabilidade para uso em sistemas operacionais diferente do DOS, tal como o UNIX. Em compensação o C apresenta uma linguagem compacta, com uma sintaxe rígida, exigindo um certo esforço de aprendizado inicial.

Apresentamos, sempre que possível, uma versão em C dos programas em Assembly implementados com o DEBUG. As rotinas de vídeo e teclado mais complexas são escritas diretamente em C, por comodidade e espaço disponível para listagem, usamos o Turbo C 2.0 (Borland, preço aproximado U\$ 120,00, não necessita de disco rígido). Porém com pequenas adaptações estes programas podem ser montados com o Microsoft C 5.1 (Microsoft, preço aproximado U\$ 300,00 necessita de disco rígido) ou Quick-C (Microsoft, preço aproximado de U\$ 70,00, não necessita de disco rígido).

Procuramos escrever os programas em C de modo mais próximo ao código Assembly. Para isto redefinimos os registradores através de um programa cabeçalho ("header") denominado CABEÇA.H (ver Apêndice C). As pequenas rotinas e módulos em C incluem este programa, o que elimina listagem de linhas repetitivamente e possibilita tratar cada módulo de programa separadamente.

Todos os programas fonte e objeto discutidos ou sugeridos podem ser obtidos diretamente com o autor.

REVELANDO O INTERIOR DO PC

7.1 Introdução

O hardware descrito na Parte I, será parcialmente desvendado com o auxílio do DEBUG neste Capítulo. Caso o leitor tenha acesso a um PC com um disco contendo este utilitário DOS, será útil a prática dos exemplos propostos. Deste modo, não só aprenderá usar o DEBUG como compreender parte importante do hardware do PC. O DEBUG é um utilitário do sistema operacional que acompanha o sistema desde a versão DOS 1.0. Antes, é necessário verificar se este programa consta do disco original de sistema operacional, através de:

```
A > DIR DEBUG.COM
```

Se a resposta for positiva, copie este utilitário num disco de trabalho, pois jamais se deve usar o disco original do sistema operacional e sim uma cópia.

O DEBUG é mais do que um simples montador ("assembler" - A). Ele é capaz de desassemblar ("unassembler" - U) qualquer programa, traduzindo seu código de máquina em símbolos mnemônicos, verificar os registradores ("registers" - R) além de possibilitar a gravação ("write" - W), dos programas tipo *.COM montados, em disco. Infelizmente, é difícil encontrar uma boa documentação deste utilitário. Neste Capítulo e no Apêndice B, procuramos preencher esta lacuna.

7.2 Usando o DEBUG

Uma vez pronto um disco com o DEBUG e espaço livre é possível executá-lo com o comando:

```
A > DEBUG          ; chamar a partir do DOS programa executável  
-                  ; DEBUG.COM ( " - " prompt do DEBUG)
```

Deste modo o DEBUG está instalado e pronto para ser usado. A Tabela 7.1 resume os comandos do DEBUG, por tipo de operação, descritos em detalhes no Apêndice B.

Comando	Exemplo	Comentário
aritmética		
H <n1> <n2> (hexadecimal)	- H 0F F0	soma e subtrai em hexadecimal resultado na próxima linha
registradores		
R ("Register")	- R - R AX	mostra todos os registradores mostra conteúdo de AX e aguarda novo valor
programas		
A <endereço> ("Assembly")	- A 100	montar código Assembly no endereço indicado. Se nenhum endereço for especificado, o programa é montado em CS:IP.
U <endereço> <tamanho> ("Unassembly")	- U11 011A	desassemblar código de máquina entre os endereços apontados. Se nenhum endereço for especificado, o programa é desassemblado a partir de CS:IP
G <endereço> <ponto final> ("Go")	-G = 100 10B	inicia a execução de um programa no endereço até encontrar um ponto final "break point"). Se nenhum endereço for especificado o programa inicia execução em CS:IP
T <=endereço> <tamanho> ("Trace")	- T 100 1A	inicia a execução de um programa passo a passo a partir do endereço indicado (CS:100) executando um certo número de bytes. Se nenhum endereço for especificado, inicia execução em CS:IP, uma instrução por vez

Tabela 7.1 Funções e comandos do DEBUG

Comando	Exemplo	Comentário
programas		
P < = endereço > < n. de vezes > ("Proceed")	-P = 100 10	executa bloco de instruções a partir do endereço, um certo número de vezes. Se nenhum endereço for especificado, inicia execução em CS:IP até o final de um loop, ou interrupção.
memória		
S < endereço > < tamanho > < cadeia > ("Search")	-S 100 L A0 "BOM"	procura por uma cadeia ("string"), no caso " BOM" a partir de CS:100 num bloco de tamanho definido por L.
F < endereço > < cadeia > ("Fill")	-F DS:00 "OK!"	preenche um bloco de memória com a cadeia indicada a partir do endereço especificado
E < endereço > ("Enter")	-E CS:FF	preenche um byte de memória no endereço indicado.
D < endereço > ("Dump")	-D 100 L F	mostra um bloco de memória com valor hexadecimal e caractere ASCII no endereço indicado e tamanho definido por L. Se nenhum endereço for especificado, mostra um certo número de bytes a partir de CS:IP
entrada e saída(I/O)		
I (endereço) ("Input")	-I 3F8	recebe byte da porta especificada, obrigatoriamente.
O (endereço)	-O 2A5	envia byte à porta especificada, obrigatoriamente

Tabela 7.1 Funções e comandos do DEBUG (continuação)

Comando	Exemplo	Comentário
interface DOS		
L < endereço > ("Load")	- L 120	carrega na memória no endereço apontado, programa arquivado no disco, cujo nome está especificado no endereço CS:0080
N < nome > ("Name")	- N OK.COM	carrega cadeia no endereço CS:80, para ser salvo (-W) ou lido (-L)
W < endereço > ("Write")	- W 12A	grava arquivo em disco, bloco de bytes de tamanho definido por BX: CX com nome especificado em CS:0080. Não aceita programas tipo *.EXE. Se nenhum endereço for especificado, CS:IP corrente será utilizado.

Tabela 7.1 Funções e comandos do DEBUG (continuação)

7.3 Registradores e Aritmética

Nesta Seção, mostramos o seguinte:

Exemplo 7.1 Somar 2 números hexadecimais usando o comando -H do DEBUG. Realizar a mesma operação através dos registradores, e montando o código de máquina

Para verificar os registradores basta usar o comando -R a partir do prompt do DEBUG "--"

```
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=CF DEBP=0000 SI=0000
DI=0000 DS=0BF1 SS=0BF1 ES=0BF1 CS=0BF1 IP=0100
0BF1:0100 B80F00 MOV AX,000F NV UP DI PL NZ NA PO NC
```

Os registradores de uso geral A,B,C,D, são mostrados na forma de 16 bits, inicializados em zero pelo DEBUG, assim como os registradores de índice SI e DI (ver Seção 3.2) e o apontador de base BP ("base pointer"). Porém, o apontador de pilha SP

("stack pointer") é diferente de zero, indicando que a pilha armazena informações de interesse do sistema. Os registradores de segmento, DS, ES, SS, CS apontam para um mesmo segmento no endereço de memória RAM alocado pelo DEBUG, para o início de programas transientes, ver Figura 2.7, Seção 2.5.2. O apontador de instruções IP tem o valor 0100h usual para início de programas tipo *.COM tal como são produzidos pelo DEBUG. O registrador de Flag (ver Seção 3.2) necessita de uma explicação extra pois o DEBUG indica-os na forma de código conforme Tabela 7.2, ver Exemplo 7.1

FLAG	bit = 0	bit = 1
estouro	No Overflow (NV)	overflow (OV)
direção	incremento (UP)	decremento (DN)
interrupção	desabilitado (DI)	habilitado (EI)
sinal	positivo (PL)	negativo (NG)
zero	não zero (NZ)	zero (ZR)
transporte	Não carry	carry
auxiliar	auxiliar (NA)	auxiliar (CA)
paridade	ímpar(odd) (PO)	par("even") (PE)
transporte	não carry (NC)	carry (CY)

Tabela 7.2 Registradores de Flag no DEBUG.

O comando -R mostra, além dos registradores, o código da primeira instrução a ser executada, no caso:

0BF1:0100	B80F000	MOV AX,000F
(endereço)	código de máquina	mnemônico Assembly

ou seja, o DEBUG procura no endereço apontado por CS:IP (no caso 0BF1:0100) a próxima instrução, no caso o código hexadecimal B8. A CPU decodifica esta instrução e procura os dois próximos bytes definindo-os como operandos, considerando o primeiro como o menos significativo (0F) e o DEBUG monta a instrução, usando os 3 primeiros bytes:

0100	B8	→	B80F00
0101	0F		
0102	00		
endereço	código		código completo:instrução + operando

Um problema incômodo é a soma ou subtração de números na notação hexadecimal. O comando -H do Debug pode ser usado para esta tarefa. Por exemplo para somar ou subtrair 2 números N1=F0h e N2=0Fh a partir do Debug, basta:

-H	F0 0F	< enter >
	00FF	00E1
soma		subtração

Este exemplo pode ser resolvido, montando uma instrução e executando-a. Para isto é necessário:

- (i) carregar A com uma parcela da soma(A ← F0h)
- (ii) carregar B com outra parcela da soma(B ← 0Fh)
- (iii) carregar código de instrução 01D8 nos bytes de endereço 0100h e 0101h (AX ← AX + BX).
- (iv) executar a instrução apontada por IP = 0100h e verificar o resultado em A.

Este processo pode ser assim operacionalizado:

-R mostrar os registros do DEBUG

-R AX mostrar e alterar registrador AX

AX 0000

: F0 novo valor em AX

-R BX mostrar e alterar registrador BX

BX 0000

: 0F novo valor em BX

-R verificar se os registradores foram alterados

AX=00F0 BX=000F CX=0000 DX=0000 SP=CFDE BP=0000 SI=0000
 DI=0000 DS=0BF1 ES=0BF1 SS=0BF1 CS=0BF1 IP=0100
 0BF1:0100 B80F00 MOV AX,000F NV UP DI PL NZ NA PO NC

-E 100 mostrar código hexadecimal no endereço 0BF1:0100
 B8.01 CS:0100 , alterar valor para 01

-E 101 alterar valor hexadecimal de CS:0101

0BF1:0101 00.D8

-R verificar a instrução montada em código.

AX=00F0	BX=000F	CX=0000	DX=0000	SP=CFDE	BP=0000	SI=0000
DI=0000	DS=0BF1	ES=0BF1	SS=0BF1	CS=0BF1	IP=0100	
0BF1:0100 01D8	ADD AX,BX				NV UP DI PL	

-T executar a instrução apontada por CS:IP,
verificar antes se IP=0100h.
verificar se valor de AX foi alterado.

AX=00FF	BX=000F	CX=0000	DX=0000	SP=CFDE	BP=0000	SI=0000
DI=0000	DS=0BF1	ES=0BF1	SS=0BF1	CS=0BF1	IP=0102	
0BF1:0102 33C0	XOR AX,AX				NV UP DI PL NZ NA PO N	

Observe o novo valor de AX: $AX \leftarrow AX + BX$ e de IP (0102)

Pode-se realizar a operação de subtração, basta usar o código de instrução 29D8 ($AX \leftarrow AX - BX$). É interessante fazer outras operações com resultados zero, nulo ou com estouro e verificar a alteração dos flags correspondentes.

7.4 Qual é a marca registrada do seu PC?

O PC é um projeto aberto. Sua área de ROM é documentada e pode ser alterada. É possível usar uma ROM diferente da original produzida pela IBM e fazer um microcomputador PC compatível. Para isto basta trocar uma EPROM de 8K bytes (6264) localizada entre os endereços F000:E000 e F000:FFFF, ver Seção 2.5.1.

A IBM reservou uma região da ROM para escrever sua marca registrada ("trade mark"). Este espaço começa na base da ROM F000:E000. Os fabricantes são obrigados a alterar a mensagem de marca registrada, colocando em geral sua própria marca. O comando D ("display" ou "dump") permite visualizar a marca registrada do seu PC, que aparece no vídeo logo após ligar o PC.

Exemplo 7.2 Qual é a marca registrada da BIOS?

-D FE00:0000 L 50

FE00:0000	53636F7075732054-65636E6F6C6769	Scopus Tecnologi
FE00:0010	61FFFFFFFFCBFFFF-FFFFFFFFFFFFFFF	A.....K.....
FE00:0020	FFFFFFFFFFFFFFFF-FFFFFFFFFFFFFFF	
FE00:0030	FFFFFFFFFFFFFFFF-FFFFFFFFFFFFFFF	
FE00:0000	FFFFFFFFFFFFFFFF-FFFFFFFFFFFFFFF	

endereço bloco de memória 16 bytes decodificação

Além do conteúdo do bloco de memória, ao final da linha o correspondente caractere ASCII é mostrado. Ver Apêndice A.

Exemplo 7.3 Mostrar o diretório apontado pelo DOS

Como vimos na Seção 2.5.2 a área de dados do DOS está na região 0000:0500 . Para conhecer o diretório do disco em uso apontado pelo DOS basta:

-D 0000:0500

Na falta de uma definição do tamanho do bloco de memória a ser mostrado L, o DEBUG mostra os primeiros 80h bytes a partir do endereço indicado.

Exemplo 7.4 Detectar o endereço de uma rotina apontada pelo vetor de interrupção.

O DEBUG permite analisar qualquer rotina de interrupção. Basta conhecer o endereço da rotina apontada pelo vetor de interrupção que está na tabela do vetor de interrupção (Ver Seção 2.7). Por exemplo o vetor de interrupção do temporizador (BIOS INT 1Ch, não documentada) esta localizado na tabela dos vetor de interrupção no endereço 70h (4 x 1Ch). Para conhecê-lo, basta:

-D 0000:0070 L 4

0000:0070 53 FF 00 F0

ou seja o endereço desta rotina de interrupção está em:

F000:FF53. É um endereço da ROM BIOS, ver este programa em código assembly

- U F000:FF53 L 4

F000:FF53 CF	IRET	desassemblar rotina
F000:FF54 1E	PUSH DS	fim da rotina de interrupção
F000:FF56 B850000	MOV AX,0050	código provável de outra rotina

No caso, esta rotina de interrupção resume-se a um simples retorno. Isto permite a definição de uma rotina própria para marcação do tempo.

7.5 Reescrevendo a marca registada

O comando -E permite alterar um ou mais bytes na memória RAM.

Exemplo 7.5 Escrever 2 caracteres num endereço

Considere a tarefa de escrever os caracteres "PC" na região de memória RAM definida por DS:0200. Antes é necessário verificar no Apêndice A, o valor do código ASCII correspondente aos caracteres:

P	→	50h
C	→	43h
caracteres		código ASCII

O exemplo 7.8 na Seção 7.7 mostra uma tabela de código de caracteres ASCII. Eles também podem ser verificados através do utilitário SideKick (Borland, preço aproximado U\$ 59,00) ou diretamente a partir do DOS (teclar ALT + número hexadecimal e o caractere correspondente é mostrado no vídeo).

O comando E e D podem fazer uma escrita e leitura na RAM de modo a operacionalizar o Exemplo 7.5

-E	DS:0200	02. 50	Introduzir novo valor em DS:0200
-E	Ds:0201	3A. 43	introduzir novo valor em DS:0201

O resultado pode ser verificado através de:

-D	DS:0200	L 2	
DS:0200		50 43	PC
endereço		código	caractere

Poderíamos escrever numa área de ROM? O leitor pode tentar ver o que acontece, refazendo o exemplo anterior porém redefinindo o segmentos de dados DS:

-R DS
 0BF1
 : FE00 DS ← FE00 (novo valor para segmento de dados)

O endereço 0BF1 é um endereço de RAM alocado pelo DEBUG ao passo que FE00 é uma região de ROM.

Escrever uma mensagem byte a byte é muito trabalhoso. O DEBUG permite escrever diretamente uma cadeia de caracteres.

Exemplo 7.6 Escrevendo uma mensagem

Podemos usar a região DS:0200 para escrever a mensagem
 " Erica Editora", usando o comando -F ("fill").

-D DS:0200 L 10 verificar bloco de memória
 0BF1:0200 2020202020202020-20202020202020

-F DS:0200 L 10 " Erica - Editora"
 preenche bloco de memória de tamanho 10h com a mensagem indicada.

-D DS:0200 L 10 verificar se a nova mensagem está presente
 0BF1:0200 4552494341202D20-454449544F524120 ERICA - EDITORA

Escrevendo uma cadeia de caracteres não é difícil rebitar a EPROM com outra marca registrada diferente do fabricante, basta possuir um copiador de EPROM e alterar a cadeia de caracteres na base da EPROM em FE00:0000

7.6 Escrevendo diretamente na memória de vídeo

Como vimos na Seção 2.5 existe no PC uma região de memória RAM para armazenamento das imagens de vídeo. No adaptador de vídeo monocromático ("MDA") este endereço começa em B000:0000 e nos adaptadores coloridos ("CGA") em B800:000. Nesta Seção, montaremos um pequeno programa capaz de limpar a página texto de vídeo e mostrar um conjunto de caracteres.

O controle direto do modo de vídeo é um assunto vasto. As linguagens de alto nível são capazes de identificar o adaptador de vídeo instalado e definem o modo mais indicado para manejo das operações de vídeo. A BIOS INT 10h (Ver Cap. 10) possui um vasto conjunto de funções para manipulação do controlador de vídeo. O Exemplo

7.7 é um pequeno programa, montado com o DEBUG, capaz de limpar a página de vídeo texto para um sistema com adaptador CGA ou EGA (ver Seção 1.2.4).

Exemplo 7.7 Limpar a página texto de vídeo

Este exemplo é idêntico à sub-rotina DOS CLS ("Clear Screen") e pode ser implementado através de BIOS INT 10h, função 06h, (ver Cap. 10).

Quando observamos um caractere no vídeo, significa que este caractere está numa área de memória que a CPU lê na frequência de 60 Hz e os envia ao vídeo através do controlador de vídeo. Um único ponto na tela denomina-se pixel e esta resolução é utilizada no modo gráfico. No modo texto, cada caractere é um conjunto de pixels, e a tela padrão tem 80 colunas e 25 linhas. Para cada caractere no vídeo são necessários 2 bytes; um para definir o atributo (cor, modo inverso, etc..) outro para indicar o próprio caracter. A Figura 7.1 mostra a organização do bloco de memória da página de texto para o adaptador CGA.

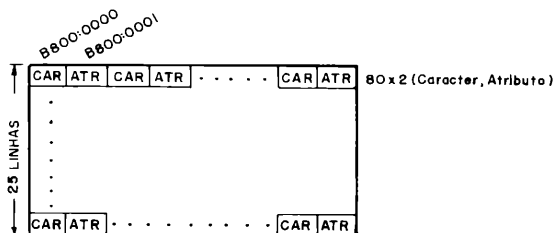


Figura 7.1 Mostrando o caracter no vídeo.

O atributo usual do caractere de vídeo para monitor monocromático é 07, (componente de frente branco e fundo preto, intenso e sem piscar). Ver nas Referências Bibliográficas indicações para maiores detalhes a respeito do manejo de vídeo.

O programa pode ser assim descrito:

```
; programa LimpTEL.com
; limpar de texto para adaptador CGA
; colocando caractere branco (ASCII 20h) e atributo 07
; na região de memória com início em B800:000 e
; tamanho 80 colunas e 25 linhas = 4000 ou FA0h
```

(usar Shift Scr)

-A 100		; montar (assembler) programa em CS:0100
CS:0100	MOV AX,B800	;1 definir segmento de dados DS na página
CS:0103	MOV DS,AX	;2 texto(vídeo CGA),através de A
CS:0105	MOV CX,0FA0	;3 tamanho da página (80col)x(25li)x2
CS:0108	MOV BX,0000	;4 = 4000h = FA0h; apontar para início
CS:010B	MOV AX,0720	;5 atributo = 7 character branco = 20h
CS:010E	MOV [BX],AX	;6 escrever página apontada por BX
CS:0110	INC BX	;7 avançar 2 bytes, byte par = char
CS:0111	INC BX	;8 byte ímpar = atributo
CS:0112	DEC CX	;9 Chegou ao final da página? CX=0?
CS:0113	JNZ 10E	;10 se CX0 continuar; do contrário
CS:0115	INT 3	;11 fim da operação, DEBUG INT.

O montador -A não aceita erros de sintaxe na editoração. Para verificar a existência de erros lógicos, o programa deve ser desassemblado através de -U ("unassembler").

-U 100 L 16		
CS:0100	B800B8	MOV AX,B800
CS:0103	8ED8	MOV DS,AX
CS:0105	B9A00F	MOV CX,0FA0
CS:0108	BB0000	MOV BX,0000
CS:010B	B82007	MOV AX,0720
CS:010E	B907	MOV [BX],AX
CS:0110	43	INC BX
CS:0111	43	INC BX
CS:0112	49	DEC CX
CS:0113	75F9	JNZ 010E
CS:0115	CC	INT 3

Para executar o programa basta :

-G = 100 (go)

Comentários a respeito do programa Exemplo 7.7

- linha 1,2 : Os registradores de segmentos não podem ser carregados diretamente com constantes imediatas, daí o uso temporário de AX
- linha 3 : CX é usado como contador e é decrementado na linha 9. Quando CX = 0, toda a tela foi varrida (25 linhas e 80 colunas)
- linha 4 : BX, usado como apontador de endereço de deslocamento no segmento de vídeo DS.
- linha 5 : AH = 07. atributo de caractere, byte mais significativo (ímpar), e AL = 20 caractere branco.
- linha 6 : Escrever no endereço indicado por DS:BX o caractere branco.
- linha 7,8 : Avançar 2 bytes para escrever próximo caractere.
- linha 9,10 : Controla fluxo do programa. Se C < 0X, então IP retorna para 10E (JNZ "Jump if Not Zero")
- linha 11 : É o modo de interromper a execução do programa e voltar ao DEBUG. Só deve ser executado dentro do DEBUG.

Caso não exista uma interrupção o programa entra em pane pois IP continua incrementando e executando instruções não previstas pelo programador. Uma alternativa é usar o comando -G com ponto de parada ("break point"),

-G = 100 1F0

e o programa será executado até IP encontrar o ponto de parada.

7.7 Criando um programa executável tipo *.COM

Os exemplos apresentados estão limitados aos ambientes do DEBUG, não podem ser salvos em disco e executados a partir do DOS. Nesta Seção, usamos o DEBUG para criar, executar e salvar um programa executável tipo *.COM

Exemplo 7.8 Criar e arquivar um programa *.COM capaz de mostrar os 256 caracteres ASCII de vídeo.

O microprocessador não entende símbolos, tal como os caracteres gráficos, e sim números. O código ASCII é um modo de representar símbolos na comunicação entre o microcomputador e o meio exterior através do teclado, monitor, porta paralela (impressora), porta serial (modem). Embora seja um padrão, existem algumas diferenças quanto ao seu uso

- código ASCII de controle 32 caracteres (00 h a 1Fh), usado para controle da impressora.
- conjunto de caracteres ASCII IBM: 255 caracteres utilizado para vídeo
- conjunto de caracteres ASCII IBM estendido, 97 caracteres associados às teclas especiais do PC.

Ver Apêndice A para uma descrição detalhada a respeito do código ASCII .

Neste exemplo, será confeccionado um programa capaz de exibir os 256 (00h - FFh) caracteres ASCII de vídeo IBM que será salvo com o nome:

ASC_VID.COM (caracteres ASCII de vídeo)

Programa 1. ASC_VID.COM

;mostrar os 256 caracteres ASCII de vídeo, com 32 caracteres

; por linha (20h). Começa na linha 4 e coluna 10

;escreve diretamente na página de vídeo modo texto para CGA

;retornar ao DOS através da Função 4Ch de DOS INT 21.

-A 100

CS:0100	MOV	AX,B800	;1 apontar segmento de dados para modo
CS:0103	MOV	DS,AX	;2 vídeo CGA B800, apontado por DS
CS:0105	MOV	AX,0700	;3 atributo(AH) = 7 caractere ASCII(AL) = 0
CS:0108	MOV	DX,0280	;4 (4lin)x(80 col)x (2 bytes) = 640 = 280h
CS:010B	MOV	CX,0014	;5 (10 col)x(2 bytes) = 20 = 14h
CS:010E	MOV	BX,0000	;6 zerar apontador
CS:0111	ADD	BX,CX	;7 apontar para coluna definida por CX
CS:0113	ADD	BX,DX	;8 apontar para linha definida por DX
CS:0115	MOV	[BX+01],AH	;9 escrever atributo no endereço ímpar
CS:0118	MOV	[BX],AL	;10 escrever caractere definido em AL
CS:011A	INC	AL	;11 carregar próximo caractere ASCII
CS:011C	CMP	AL,FF	;12 se for superior ou igual ao último
CS:011E	JNB	012F	;13 parar, do contrário continuar loop
CS:0120	ADD	CX,4	;14 avançar 4 bytes,deixando um branco
CS:0123	CMP	CX,0094	;15 fim de linha? 94h = 14h + (4x32 = 80h)
CS:0127	JNZ	010E	;16 se não próxima coluna
CS:0129	ADD	DX,0140	;17 mais 2 linhas (2 lin)x(80 col)x 2 =
CS:012D	JMP	010B	;18 320 = 140h. Iniciar nova linha.
CS:012E	MOV	AH,4C	;19 fim da impressão, voltar ao DOS
CS:0131	INT	21	;20 através da função 4C de DOS INT 21

Salvando o programa

Calcular o tamanho

-H	133 100	
	233 0033	33 bytes a serem salvos.
-R	CX	
	0033	colocar em CX número de bytes a serem salvos.
-R	BX :	
	00	definidos por BX:CX (0033)

-N	ASC_VID.COM	definir nome do programa a ser salvo
-D	80 L F	verificar se nome está em CS:80
-W		("write") arquivar 33 bytes em disco
-Q		("quit") retornar ao DOS
A >	cls	limpar tela
A >	ASC_VID	executar programa gravado.

O algoritmo principal deste programa reside em apontar corretamente o byte onde se localiza o caractere. O apontador BX é uma combinação dos registradores CX (coluna) mais DX (linha). As linhas de vídeo são escritas de cima para baixo, no loop entre as linhas de programa 7 e 16. Além do salto diferente de zero (JNZ) usado no Exemplo 7.7, usamos mais 2 tipos de saltos neste Exemplo.

linha 14	JNB ("Jump Not Below" ou "Jump if Above or Equal") somente quando AL for acima de 255 ("not below") o programa salta para o fim, garantindo que os 255 caracteres foram escritos
linha 19	JMP ("Jump") salto incondicional. Sem nenhuma condição, IP é bruscamente alterado para o valor indicado, neste Exemplo para iniciar na primeira coluna em nova linha
linha 20,21	Finalmente usamos pela primeira vez uma interrupção DOS função DOS 4C. (Neste livro sempre por função DOS entenda-se uma das funções de DOS INT 21h. (Ver Cap. 11). Esta função é a mais indicada para terminar um programa executável.

Ver o Capítulo 8 e Apêndice C para uma discussão detalhada do código de instrução Assembly 8088.

Com estes exemplos executamos a maior parte dos comandos do DEBUG. Recomendando a execução de 2 exercícios adicionais. O mais simples seria unir os Exemplos 7.7 e 7.8 num único programa, de modo a limpar a tela antes de escrever os caracteres. Um exercício mais avançado seria numerar os caracteres ASCII em notação hexadecimal, com uma linha superior com os 32 caracteres (00-1F) e uma coluna com o valor de inicial de cada linha (00, 20, 40, etc...) Deste modo fica simples identificar o código de um símbolo. Por exemplo o caractere "A" está na linha 3 (40h) e coluna 1, portanto tem o valor :

$$\text{"A"} \longrightarrow 40\text{h} + 01\text{h} = 41\text{h (código ASCII)}$$

O CÓDIGO DE INSTRUÇÃO ASSEMBLY

8.1 O Conjunto de Instruções

É impossível detalhar 300 instruções, aproximadamente, do 8088 num único livro. Em vez disto, preferimos utilizá-las livremente através de exemplos. A melhor maneira de conhecê-las é verificar seu uso em programas. No Apêndice C apresentamos o conjunto completo de instruções em ordem alfabética. Indicamos nas referências bibliográficas, os livros específicos de linguagem Assembly.

Neste Capítulo, as instruções serão apresentadas através de blocos de funções. Deste modo, é possível ter uma visão global das principais operações sem adentrar em detalhes, em geral confusos para o iniciante. Nosso objetivo é apontar as principais direções nesta selva de mnemônicos e siglas sem nos preocupar com os caminhos específicos. Além de resumir as principais instruções em Assembly do 8088 definimos, neste Capítulo, os termos mais usados em programação Assembly, de modo a capacitar o leitor a ler e interpretar outros programas além dos exemplos discutidos.

Uma instrução genérica em Assembly consiste de 4 partes:

rótulo instrução operando ; comentário

O rótulo é opcional e define uma posição de memória onde está localizada uma instrução ou uma variável. A instrução é um item obrigatório, mesmo que seja para não executar nada, como NOP ("No Operation", ver Exemplo 9.4). Nem todas as instruções requisitam um operando. Este pode ser um registrador, uma posição de memória ou uma constante imediata.

O DEBUG (Ver Cap. 7) não possibilita o uso de rótulos nem a definição de variáveis, tal como os montadores ASSEMBLER (Ver Cap. 6). Pode-se classificar os operandos em tipos, tal como os descritos na Tabela 8.1

Tipo	Uso	Comentário
destino	transferência de dados, manipulação de bits.	registrador ou memória onde um resultado será armazenado após operação
origem	transferência de dados, manipulação de bits.	registrador, memória ou valor imediato usado na operação que permanecerá inalterado
rótulo curto	transferência condicional e controle de loop	rótulo (endereço) de acesso controlado por uma transferência, a uma distância de -128 a +127 bytes da instrução
acumulador	AX, AH, AL	registrador usado na transferência de word (AX) ou bytes.
port	IN, OUT	número de port de I/O, imediato se < 255, ou indireto apontado por DX.
contador	deslocamento, rotação	especifica o número de bits a ser deslocado ou rodado. De modo imediato, se = 1 ou indireto, se apontado por CL.
interrupção	INT	valor imediato (0...255) que identifica o número da interrupção.

Tabela 8.1 Tipos de operandos

É praticamente impossível acompanhar o tempo gasto pela CPU para execução de um programa, dada a diversidade dos ciclos de clock gastos para execução dos diferentes tipos de código. É mais simples executar o programa um certo número de vezes e avaliar o tempo gasto através, por exemplo, da função DOS 2Ch (Ver Cap. 10). Por isto nunca nos preocupamos com o número de ciclos que a CPU gasta para a execução de uma instrução específica.

O uso do código de máquina para execução de uma instrução só será feito no Exemplo 7.1. É muito mais simples montarmos programas com o DEBUG, ou outro montador, sem nos preocuparmos com códigos de máquina.

8.2 Instruções de Transferência de Dados

A Tabela 8.2 descreve as 14 instruções de transferência de bytes e words entre memória e registradores, bem como entre os registradores AL ou AX e os port de

entrada e saída. Neste conjunto, incluem-se as operações com a pilha ("stack") e com o registrador de Flag.

Instrução		Comentário
Uso Geral		
MOV	destino, origem	transfere byte ou word
PUSH	origem	salva uma word na pilha
POP	destino	recupera uma word da pilha
XCHG	destino, origem	troca byte ou word
XLAT	tabela-translação	translação de tabelas
Entrada e Saída		
IN	acumulador, port	entrada (na CPU) de byte ou word
OUT	port, acumulador	saída (da CPU) de byte ou word
Transferência de Endereço		
LEA	destino, origem	carregar endereço efetivo
LDS	destino, origem	carregar apontador através de DS
LES	destino, origem	carregar apontador através de ES
Transferência de Flags		
LAHF		carregar AH com flags
SAHF		transferir AH para os flags
PUSHF		salvar flags na pilha
POPF		recuperar flags da pilha

Tabela 8.2 Instruções de Transferência de Dados

MOV destino,origem

Transfere um byte ou word do operando fonte para o operando destino, que podem ser registradores ou endereço de memória. É uma das instruções mais utilizadas em qualquer programa. Ver Exemplos 7.7 e 7.8

PUSH fonte

Decrementa a pilha SP ("Stack Pointer") por 2 ($SP - 2$) e transfere uma word do operando fonte para o topo da pilha apontado por SP. É usado geralmente para armazenar parâmetros na pilha antes de chamar uma rotina. É o modo mais rápido de armazenar dados, provisoriamente, na pilha. Ver Exemplo 8.1 abaixo.

POP destino

Recupera uma palavra localizada na posição corrente da pilha para o operando de destino, incrementando SP por 2 ($SP \rightarrow SP + 2$), passando a apontar para uma posição 2 bytes acima na pilha. Usado para mover variáveis armazenadas na pilha para os registradores ou memória. Ver exemplo 8.1 abaixo. No 8088, a pilha funciona do modo "último a entrar, primeiro a sair", LIFO ("Last In First Out").

IN acumulador, port

Transfere um byte ou uma word e um port de entrada para o registrador AL (8 bits) ou AX (16 bits) respectivamente. O endereço do port pode ser especificado diretamente de modo imediato se o endereço for de um byte (0...255), ou indiretamente através do registrador DX, o que permite acessar qualquer um dos 512 ports do PC (A0-A8), Ver Seção 4.3. Intensamente utilizada em projetos de interfaceamento, Ver Cap. 11

OUT port, acumulador

Transfere um byte ou uma word de AL ou AX, respectivamente, para um port de saída. O endereço do port é especificado de modo idêntico a IN. Ver Cap. 11.

PUSHF

Decrementa SP de 2, ($SP \rightarrow SP - 2$), transferindo todos os registros de flags para a pilha, sem modificá-los. Ver Exemplo 8.1 abaixo.

POPF

Transfere os bits da word apontada por SP para o registrador de flag, incrementando SP por 2 ($SP \rightarrow SP + 2$). Permite recuperar o estado de flag depois da execução de uma rotina, retomando o fluxo normal do programa.

Exemplo 8.1 Salvar Flags na pilha. Resetar Flags. Realizar uma operação com resultado nulo. Recuperar Flags

Através do Debug monte o seguinte programa:

```

- A 100
CS:0100  PUSHF                , salvar registro de Flag na pilha
CS:0101  MOV  AX,0ED5         , máscara de flag, setar bits
CS:0104  PUSH AX              , salvar máscara na pilha
CS:0105  POPF                 , recuperar flags modificados
CS:0106  XOR  AX,AX           , zerar acumulador
CS:0108  DEC  AX              ; produzir um número negativo
CS:0109  INT  3               ; fim do programa através DEBUG
CS:010A

```

Executar ,dentro do DEBUG, o programa acima analisando o que ocorre com o apontador de pilha SP e com os registros de Flag, do seguinte modo

```

-R
AX=0000 BX=0000  CX=0000  DX=0000  SP=CFDE  BP=0000  SI=0000
DI=0000  DS=0BF1  ES=0BF1  SS=0BF1  CS=0BF1  IP=0100
0BF1:0100 9C          PUSHF                NV UP DI PL NZ NA PO NC

```

Estado inicial configurado pelo DEBUG. O apontador de pilha já foi usado pelo DEBUG e está alterado. O registrador de flag está zerado, conforme Tabela 7.2. Verificar se IP aponta para o início da instrução e CS no segmento em uso.

```

-T
AX=0000 BX=0000  CX=0000  DX=0000  SP=CFDC  BP=0000  SI=0000
DI=0000  DS=0BF1  ES=0BF1  SS=0BF1  CS=0BF1  IP=0101
0BF1:0101 B8D50E      MOV  AX,0ED5        NV UP DI PL NZ NA PO NC

```

Salvando o registrador de flag na pilha. Neste caso, SP ← SP - 2, sem modificar demais registradores.

```

-T
AX=0ED5 BX=0000  CX=0000  DX=0000  SP=CFDC  BP=0000  SI=0000
DI=0000  DS=0BF1  ES=0BF1  SS=0BF1  CS=0BF1  IP=0104
0BF1:0104 50          PUSH AX             NV UP DI PL NZ NA PO NC

```


Colocar a máscara capaz de setar os bits de flag (= 1) no registrador AX, ver Figuras 3.3 e 3.4, além da tabela 7.2 para compreender porque a escolha deste valor.

-T

```
AX=0ED5 BX=0000 CX=0000 DX=0000 SP=CFDA BP=0000 SI=0000
DI=0000 DS=0BF1 ES=0BF1 SS=0BF1 CS=0BF1 IP=0A05
0BF1:0105 9D POPF NV UP DI PL NZ NA PO NC
```

Transferindo AX para pilha. Além de SP ← SP - 2 nenhum outro registrador é alterado.

-T

```
AX=0ED5 BX=0000 CX=0000 DX=0000 SP=CFDC BP=0000 SI=0000
DI=0000 DS=0BF1 ES=0BF1 SS=0BF1 CS=0BF1 IP=0106
0BF1:0106 31C0 XOR AX,AX OV DN EI NG ZR AC PE CY
```

Retirar da pilha os bits de flag mascarados. Verificar resultado com auxílio da Tabela 7.2. A pilha sobe 2 bytes o que pode ser verificado através de: SP ← SP + 2

-T

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=CFDC BP=0000 SI=0000
DI=0000 DS=0BF1 ES=0BF1 SS=0BF1 CS=0BF1 IP=0108
0BF1:0108 48 DEC AX NV DN EI PL ZR NA PE NC
```

A função OR EXCLUSIVE XOR ($Y = (\text{not } A) * B + A * (\text{not } B)$) é nula caso $A = B$. Deste modo é possível zerar o acumulador usando somente 2 bytes. O resultado de valor positivo desta operação é apontado pelo flag de sinal (PL ← NG). Verifique como uma operação lógica zera os flags de overflow (NV) e de carry (NC).

-T

```
AX=FFFF BX=0000 CX=0000 DX=0000 SP=CFDC BP=0000 SI=0000
DI=0000 DS=0BF1 ES=0BF1 SS=0BF1 CS=0BF1 IP=0109
0BF1:0109 CC INT 3 NV DN EI NG NZ AC PE NC
```

Decrementar de operando nulo significa obter um número negativo. O que é indicado pelo flag de sinal (NG ← PL).

8.3 Instruções Aritméticas

O 8088 possui um grande conjunto de instruções aritméticas, tais como divisão e multiplicação, operação com decimais, etc... O programador deve ter o cuidado em especificar os operandos corretamente, pois o uso de dados inválidos pode fornecer resultados estranhos. A Tabela 8.3 mostra o conjunto de operações aritméticas do 8088.

Instrução		Comentário
Adição		
ADD	destino, origem	soma de byte ou word. afeta flags
ADC	destino, origem	soma de byte ou word com carry.
INC	destino	incrementa byte ou word com + 1
AAA		ASCII ajuste para adição
DAA		ajuste decimal para adição
Subtração		
SUB	destino, origem	subtrair byte ou word destino-origem
SBB	destino, origem	subtrair byte ou word com empréstimo
DEC	destino	decrementa byte ou word por 1 (- 1)
NEG	destino	reverte o sinal de um inteiro
CMP	destino, origem	destino-origem, sem alterar operandos
AAS		ajuste ASCII para subtração
DAS		ajuste decimal para subtração
Multiplicação		
MUL	origem	multiplicação do acumulador com origem
IMUL	origem	multiplicação com sinal, origem e AX
AAM		ajuste ASCII para multiplicação
Divisão		
DIV	origem	divide acumulador pela origem, sem sinal
IDIV	origem	divisão inteira com sinal, AX pela origem
AAD		ajuste ASCII para divisão
CBW		converte byte para word
CWD		converte word para duplo ("double) word.

Tabela 8.3 Operações Aritméticas

Algumas destas instruções já foram utilizadas nos exemplos anteriores. No Exemplo 7.7 usamos as instruções de incrementar por 1 (INC) e decrementar por 1 (DEC). As instruções de adição sem carry (ADD) e de comparação (CMP) foram utilizadas no Exemplo 7.8. A instrução de comparação é particularmente útil no controle do fluxo de programa antes de uma instrução de salto condicional, ver Seção 8.5. Estas instruções, em geral, afetam os flags de operações. Ver, no Apêndice C, quais flags afetados em cada uma das instruções aritméticas.

8.4 Instruções de Manipulação de bits

Um conjunto de instruções do 8088, Tabela 8.4, permite conhecer o estado de um bit individual. Isto pode ser feito através de uma operação lógica, deslocamento rotação de bits no interior de bytes ou words.

Instrução		Comentário
Lógico		
NOT	destino	inverter (complemento 1) byte ou word
AND	destino, origem	AND lógico entre destino e origem
OR	destino, origem	OR lógico entre destino e origem.
XOR	destino, origem	OR exclusivo de byte ou word
TEST	destino, origem	AND sem alterar operando, somente flags
Deslocamento		
SHL	destino, contador	deslocamento à esquerda lógico
SAL	destino, contador	deslocamento aritmético à esquerda
SHR	destino, origem	deslocamento à direita de byte ou word
SAR	destino, contador	deslocamento aritmético à direita
Rotação		
ROL	destino, contador	rotação para esquerda de byte ou word
ROR	destino, contador	rotação para direita de byte ou word
RCL	destino, contador	rotação para esquerda incluindo carry
RCR	destino, contador	rotação para direita incluindo carry

Tabela 8.4 Instruções de manipulações de bits.

As operações lógicas booleanas afetam os flags do seguinte modo: Overflow (O) e Carry (C) são sempre zerados (NV) e (NC). O valor de carry auxiliar (A) é indefinido. Os Flags de sinal (S), zero (Z) e paridade (P) dependem do resultado da operação, e são geralmente usados antes de saltos condicionais. A instrução inversora NOT não afeta os flags e sim o operando de destino. A instrução XOR foi usada no Exemplo 8.1 para zerar o acumulador.

As instruções de deslocamento "shift" para a esquerda "left" ou direita "right" podem ser feitas até 255 vezes, dependendo do valor indicado pelo contador (a constante 1 ou o registrador CL). São úteis para a identificação do estado de um bit pois o flag de carry (C) contém o valor do último bit deslocado para fora do operando de destino. As instruções de rotação são idênticas às de deslocamento, exceto quanto ao fato que o byte ou word não perde sua integridade e seu valor pode ser recuperado após o mesmo número de deslocamento em sentidos opostos.

8.5 Instruções de transferência de programa

O conteúdo do par de registradores CS:IP aponta sempre para a próxima instrução a ser executada pela CPU (ou melhor para entrar na fila da interface de bus BIU, Ver Seção 3.2). A transferência de uma sequência de instruções para outra é feita alterando o conteúdo destes registradores.

No 8088, quatro tipos de transferências são permitidas:

- | | |
|---|------------|
| ● transferência condicional, | Tabela 8.5 |
| ● transferência incondicional, | Tabela 8.6 |
| ● instruções de controle de interações, | Tabela 8.6 |
| ● instruções de Interrupção. | Tabela 8.6 |

Exceto as instruções de interrupções, as demais não afetam os flags da CPU. Na maioria das vezes estas transferências dependem do estado de flag.

Instrução	Condição de flag	Saltar se...
JA/JNBE	(C OR F = 0)	superior ("Jump if Above" ou "Not Below or Equal")
JAE/JNB	C = 0	superior ou igual ("Jump if Above or Equal" ou "Not Below")
JB/JNAE	C = 1	inferior ("Jump if Below" ou "Not Above or Equal")
JBE/JNA	(C OR F = 1)	inferior ou igual ("Jump if Below or Equal" ou "Not Above")
JC	(C = 1)	carry (Carry) (DEBUG CY)
JE/JZ	(Z = 1)	igual ou zero ("Jump if Equal" ou "Zero") (DEBUG ZR)
JG/JNLE	((S XOR O) OR Z = 0)	maior ("Jump if Greater" ou "Not Less or Equal")
JGE/JNL	(S XOR O) = 0	maior ou igual ("Jump if Greater or Equal" ou "Not Less")
JL/JNGE	(S XOR O) = 1	menor ("Jump if Less" ou "Not Greater or Equal")
JLE/JNG	((S XOR O) OR Z = 1)	menor ou igual ("Jump if Less or Equal" ou "Not Greater")
JNC	C = 0	não carry ("Jump if Not Carry")
JNE/JNZ	Z = 0	diferente ("Jump if Not Equal" ou "Not Zero" DEBUG ZR)
JNO	O = 0	não overflow ("Jump if Not Overflow" DEBUG NV)
JNP/JPO	P = 0	ímpar ("Jump if Not parity" ou "parity odd" DEBUG PE)
JNS	S = 0	positivo ("Jump if Not Signal")
JO	O = 1	overflow ("Jump if Overflow")
JP/JPE	P = 1	par ("Jump if Parity" ou "Parity Even" DEBUG PE)
JS	S = 1	negativo ("Jump if Signal")

Tabela 8.5 Transferências de Programa condicional

Instrução		Comentário
Transferência Incondicional		
CALL	nome da sub-rotina	chamar uma sub-rotina
RET		retornar de uma sub-rotina
JMP	alvo	saltar de modo incondicional
Controle de Interações		
LOOP	rótulo-curto	loop
LOOPE/ LOOPZ	rótulo-curto	loop se igual/ ou zero
LOOPNE/ LOOPNZ	rótulo-curto	loop se diferente / não zero
JCXZ	rótulo-curto	salta se contador de loop CX = 0
Interrupções		
INT	tipo-número	vetor de interrupção em 4 x <i>tipo</i>
IRET		retornar do serviço de interrupção
INTO		interromper se estouro (tipo = 4)

Tabela 8.6 Transferência de programas

CALL nome da rotina e RET

Interrompe o fluxo de instruções para executar uma rotina ("procedure") especificada. Após encontrar a instrução de retorno RET volta ao programa principal. O endereço pode estar tanto dentro do mesmo segmento (intra-segmento, ou "NEAR") como em outro segmento (intersegmento, ou "FAR"). Em ambos os modos, o endereço da próxima instrução, definido por CS:IP, fica armazenado na pilha para retornar ao programa principal ao fim da rotina, ou seja quando a instrução RET for encontrada.

JMP alvo

Esta instrução de salto transfere o controle de programa para o endereço alvo de modo incondicional. Ver seu uso no Exemplo 7.8. Ao contrário de CALL nenhuma informação é armazenada na pilha. Se a distância a ser saltada for inferior a 128 bytes, trata-se de um salto curto ("Short Jump"). Se o salto for a uma distância maior, porém dentro do mesmo segmento, trata-se de um salto próximo ("Near Jump"). Através do

apontador indireto na memória, é possível realizar saltos longos para fora do segmento corrente.

As instruções de transferência condicional checam o registrador de flag que sinalizam (talvez daí o termo Flag, bandeira, usado em operações navais). Se a condição testada for verdadeira, conforme Tabela 8.5, o desvio de programa é realizado, do contrário, a CPU desconhece a instrução de transferência. Todas as transferências condicionais são do tipo curto ("SHORT"). Ver uso de JNB e JNZ no Exemplo 7.8.

As interações de controle necessitam de um rótulo curto ("short label"). São usados para regular a execução de um bloco de instruções o número de vezes indicado pelo contador CX.

As instruções de INT necessitam de um tipo (0...255). Ver no Cap.5 o ciclo de uma instrução de interrupção. Tanto as interrupções por hardware como por software retornam ao programa principal quando encontrarem a instrução IRET (Interrupt Return), restabelecendo os estados de flag e apontador de instrução originais.

8.6 Instruções de Controle

Essas instruções, Tabela 8.7, permitem ao programa controlar várias funções da CPU e não necessitam de operandos. Um grupo destina-se ao controle do registrador de flag e outro para sincronismo com eventos externos.

Instrução	Comentário
Controle de Flags	
STC	C = 1 (DEBUG CY)
CLC	C = 0 (DEBUG NC)
CMC	inverte ("toggle") flag de carry
STD	D = 1 autodecrementar apontador de índices em cadeias
CLD	D = 0 autoincrementar apontador de índices em cadeias
STI	I = 1 habilita interrupção permitindo à CPU reconhecer a solicitação de uma interrupção.
CLI	I = 0 desabilita interrupção mascarável.

Tabela 8.7 Instruções de Controle

Instrução	Comentário
Sincronismo Externo	
HLT	Alto ("Halt") interrompe instruções, até reconhecer uma interrupção ou Reset
WAIT	Espera até pino (-TEST) ativo, ver Seção 4.2
LOCK	controle do sinal de bus
LOCK	controle do sinal de bus (-LOCK)
ESC	Escape, acesso a um processador externo
NOP	Nenhuma operação. A CPU desconhece a instrução.

Tabela 8.7 Instruções de Controle (continuação)

As instruções apresentadas, em conjunto com os Exemplos discutidos e o Apêndice C, permitem compreender a maioria dos programas Assembly do 8088 usados no PC. As operações com cadeias ("strings") são bem mais simples quando manipuladas em linguagem de alto nível. Por isto limitamos o comentário destas instruções ao apresentado no Apêndice C.

INTERRUPÇÕES DO BIOS

9.1 Os serviços do ROM-BIOS

Para que reinventar a roda? A maioria dos programas básicos de controle de entrada e saída do PC foram desenvolvidos e aperfeiçoados pela IBM e estão gravados na ROM-BIOS (Ver Seção 2.5.1). Conhecer e usar este recurso é a maneira mais rápida e eficiente de produzir programas confiáveis e enxutos.

As diferentes versões do PC estenderam em muito a capacidade dos programas originais da ROM. Parte deste aperfeiçoamento está armazenado como extensão junto com o sistema operacional, o programa "oculto" IBMBIO (ver Seção 2.5.1). As últimas versões de hardware do PC, como do AT e PS/2 implementaram novas interrupções BIOS, além de manter as existentes no PC. Neste Capítulo, somente os serviços BIOS do PC-XT e para placa de vídeo CGA estão documentados. No Apêndice D, está o conjunto completo de serviços da ROM BIOS.

No projeto original do PC, existem 16 possíveis interrupções BIOS (10h —> 1Fh mais hardware int 5h). Porém somente 11 dessas funções são utilizadas. As demais não estão documentadas, como a interrupção 1Ch que veremos no Capítulo 11, ou tornaram-se obsoletas, como por exemplo os serviços de BIOS INT 15h de controle de fita cassete. Estas interrupções dividem-se num conjunto de rotinas, denominadas serviço, cujo acesso é controlado pelo acumulador (AH e AL). Por exemplo a interrupção de vídeo BIOS INT 10h (só usamos notação hexadecimal, com sufixo "h" para identificar as interrupções) possui mais de 50 diferentes serviços. As funções e serviços podem ser classificados de acordo com o dispositivo de entrada e saída que controlam, conforme Tabela 9.1

serviços (AH)	registradores de entrada	comentários	registradores de saída
	INT 10h	Serviços de vídeo	
00	AL (modo)	define modo de vídeo (00-0A)	nenhum
01	CH, CL	tamanho do cursor	nenhum
02	BH (pag) DH (lin) DL (col)	define posição do cursor, BH página DH coordenada y (max 24), DL coordenada x (max 79)	nenhum
03	BH (pag)	ler posição do cursor	CX: forma cursor DH: (lin, y coord) DL: (col, x coord)
04	-	ler posição da caneta ótica	AH: estado ligado BX: coordenada x CX: coordenada y DX: caractere
05	AL (pag)	definir página vídeo	nenhum
06	AL (num lin) BH (atributo) CX (canto sup.esq.) DX (canto inf.dir.)	inicializa uma janela com atributo ou sobe o conteúdo de uma janela um certo número de linhas	nenhum
07	AL (num lin) BH (atributo) CX (canto sup.esq.) DX (canto inf.dir.)	inicializa uma janela com atributo e branco desce número de linhas de uma janela	nenhum nenhum
08	BH (pag)	ler caractere e atributo na posição atual do cursor	AH = atributo AL = caractere

Tabela 9.1 Serviços de vídeo BIOS INT 10h

serviços (AH)	registradores de entrada	comentários	registradores de saída
	INT 10h	Serviços de vídeo	
09	AL (caractere) BH (página) BL (atributo) CX (contador)	escrever caractere ASCII e atributo na posição do cursor um certo número de vezes	nenhum
0A	AL (caractere) BH (página) CX (contador)	escrever caractere ASCII na posição do cursor sem atributo CX vezes	nenhum
0B	BH (paleta) BL (cor)	definir paleta, cor de fundo ou borda, gráfico	nenhum
0C	AL (cor pixel) BH (página) CX (coluna) DX (linha)	escrever pixel gráfico na coluna CX (coordenada x) e linha DX (coordenada y) na cor definida em AL	nenhum
0D	BH (página) CX (coluna) DX (linha)	ler pixel gráfico na coluna CX (coordenada x) linha DX (coordenada y)	AL (cor)
0E	AL (caractere) BH (página)	escrever caractere no modo teletipo (TTY)	nenhum
0F	-	obter modo corrente de vídeo, para todas as placas	AH (num de col.) AL (modo display) BH (página ativa)

Tabela 9.1 Serviços de vídeo BIOS INT 10h (continuação)

Na Seção 9.2 e nos Exemplos 9.1 e 9.4, comentamos e utilizamos algumas destes serviços de vídeo.

As demais interrupções BIOS e respectivos serviços estão resumidos na Tabela 9.2

serviços (AH)	registradores de entrada	comentários	registradores de saída
	INT 11h -	Configuração do sistema	AX
	INT 12h -	Memória RAM disponível (em KB)	AX
	INT 13h	Serviços de DISCO	
00	DL = drive	reinicializar sistema de disco	CF = 0, se ok
01	DL = drive	ler estado do disco do sistema	AH = 0, AL ?
02	AL, CX, DX ES: BX	transferir um ou mais setores do disco para memória	CF = 0, se ok AH = 0, se ok
03	AL, CX, Dx ES: BX	escrever um ou mais setores da memória no disco	CF = 0, se ok AH = 0, se ok
04	AL, CX, DX ES: BX	verificar endereço dos campos de um ou mais setores	CF = 0, se ok AH = 0, se ok
05	AL, CX, DX ES: BX	formatar trilha inicializando setor do disco	CF = 0, se ok AH = 0, se ok

Tabela 9.2 Funções e serviços gerais da ROM-BIOS

serviços (AH)	registradores de entrada	comentários	registradores de saída
	INT 14h	Controle da porta Serial	
00	AL,DX	inicializar porta serial	AX
01	AL,DX	enviar caractere para porta serial	AX
02	DX	receber caractere da porta serial	AX
03	DX	obter estado da porta serial	AX
	INT 15h	(funções com cassete,obsoletas)	
	INT 16h	Serviços do Teclado	
00	-	ler caractere do teclado	AH,AL
01	-	obter estado do teclado	ZF,AX
02	-	obter estado teclas especiais	AL
	INT 17h	Controle da Impressora	
00	AL,DX	enviar caractere(AL) para impressora(DX)	AH
01	DX	inicializar impressora,retornando estado	AH
02	DX	obter estado da impressora	AH

Tabela 9.2 Funções e serviços gerais da ROM-BIOS (continuação)

serviços (AH)	registradores de entrada	comentários	registradores de saída
	INT 18h	(passar controle ao ROM-BASIC, se existir)	
	INT 19h	reinicializar o sistema ("boot")	
	INT 1Ah	Relógio	
00	-	obter numero de pulsos do contador	AL, CX:DX
01	CX:DX	inicializar contador de pulsos	-
02	-	obter hora em decimal	CX, DX
03	CX, DX	inicializar hora em decimal	-
04	-	obter data em decimal	CX, DX
05	CX, DX	inicializar data em decimal	-

Tabela 9.2 Funções e serviços gerais da ROM-BIOS (continuação)

Algumas funções tais como a configuração do sistema e teclado são mostradas através de exemplos nas Seções 9.3 e 9.4 respectivamente. Os pulsos do contador indicados nas interrupções de relógio (INT 1A), têm como base de tempo a frequência de $f = 18.2 \text{ Hz}$ do canal 0 do temporizador, ver Seção 2.3.

9.2 Os serviços de vídeo da ROM-BIOS

Se existe uma tarefa onde velocidade é um fator crucial, esta é a manipulação de vídeo pela CPU. A tela de vídeo é recomposta na frequência de 60 Hz, e o tempo de retrocesso vertical é da ordem de 1,25 ms. As linguagens de alto nível são capazes de manipular a maioria dos adaptadores de vídeo (ver Seção 1.2.4). Porém nem sempre o código objeto produzido por estes compiladores é rápido o suficiente para rotinas de vídeo. As vezes é necessário programar diretamente o controlador de vídeo 6845. Uma

opção intermediária reside na programação em Assembly, usando os recursos da interrupção do ROM-BIOS.

No Exemplo 7.7, mostramos um código capaz de limpar a tela texto de vídeo, escrevendo diretamente na memória RAM. O Exemplo 9.1 retoma essa tarefa, utilizando os recursos da BIOS INT 10h, função 06h. Na verdade, esta função é de carácter mais geral pois permite manipular uma janela qualquer de vídeo, delimitado pelo número de linhas e colunas especificadas nos registradores CX e DX (ver Apêndice D).

Exemplo 9.1 Limpar tela texto e reposicionar cursor no canto superior esquerdo.

Usando o DEBUG, montar e executar o seguinte programa:

MOV AH,06	, preparar função 06 deslocar janela
MOV BH,07	, usando atributo normal
MOV CH,00	, canto superior, linha 0
MOV CL,00	, esquerdo, coluna 0
MOV DH,18	, canto inferior, linha 24 (18h = 24)
MOV DL,4F	, direito, coluna 79 (4Fh = 79)
INT 10	, executar BIOS INT 10h, vídeo
MOV AH,02	, preparar função de posicionar cursor
MOV BX,0000	, definindo página zero de vídeo
MOV DH,00	, no canto superior, linha 0
MOV DL,00	, esquerdo, coluna 0
INT 10	, executar BIOS INT 10h, vídeo
MOV AH,4C	, retornar ao DOS através da função
INT 21	, DOS 4Ch.

Exemplo 9.1 Listagem programa EX_9#1.COM

Nomear o programa como:

-N EX_9#1.COM

Gravar e executar diretamente a partir do DOS. Como exercício pode-se variar o tamanho da janela, ou redefinir a posição do cursor.

Na Seção 9.5, apresentamos uma versão em C dessa rotina.

9.3 Qual é a configuração do seu PC?

Uma dúvida comum é a respeito dos dados fornecidos pelo fabricante do PC. Mesmo os usuários mais experientes às vezes se confundem com as informações apresentadas pelos diferentes fabricantes, versões e extensões do equipamento IBM-PC compatível. Nesta Seção, mostramos como utilizar rotinas de interrupção da ROM-BIOS, Exemplo 9.2, capazes de identificar os dispositivos e a capacidade de um PC genérico.

Na área de dados da BIOS, ver Tabela 9.5 abaixo, localizam-se as informações a respeito dos dispositivos instalados, tal como identificados pelo programa IBMBIO associado ao sistema operacional.

Exemplo 9.2 Quais são os dispositivos disponíveis instalados num PC genérico?

Nos 2 bytes de endereço 0000:0410h e 0000:0411h, localiza-se a lista de periféricos detectados pelo BIOS, conforme Tabelas 9.3 e 9.4 respectivamente.

bits		dispositivos instalados
(em 0000:0410)		
0 = 1		se existir disco-flexível
1 = 1		se existir co-processador matemático
2	reservado	usados na versão original do PC para indicar
3	reservado	quantidade de memória na placa principal.
4 e 5	00	existência provável de placa EGA (modo texto)
	01	40 col x 25 lin modo texto colorido (CGA)
	10	80 col x 25 lin modo texto colorido (CGA)
	11	80 col x 25 lin modo texto monocromático (MDA)
6 e 7		número de discos flexíveis instalados.

Tabela 9.3 Dispositivos de vídeo e drivers instalados

Por exemplo no meu PC, usando o DEBUG obtive o seguinte:

-D 0000:0410 L 1
6D

Ou seja o valor 6D = 10101101

bit	valor	periférico
0	1	tem ao menos um disco flexível instalado
1	0	não tem co-processador matemático
2	1	indiferente
3	1	indiferente
4	0	
5	1	vídeo modo 80 col. placa CGA
6	0	
7	1	2 discos flexíveis instalados.

Qual é a relação entre estes valores e o conjunto de chaves de configuração existentes na placa principal? Na verdade, estas informações de hardware estão conectadas aos 4 bits menos significativos da Port C da porta paralela 8255 (ver Figura 2.10). O IBMBIO ao ser instalado lê os valores desta porta e armazena os valores na região de dados da BIOS, no caso no endereço 0000:0410h

bits (em 0000:0411)	dispositivos instalados
0	reservado
1, 2 e 3	número de portas seriais RS-232
4 = 1	se existir adaptador de jogos
5 = 1	se existir modem interno
6 e 7	número de portas paralelas

Tabela 9.4 Portas seriais e paralelas instaladas.

Você pode verificar se o seu PC possui um adaptador de jogos instalados lendo o estado do segundo bit do endereço 0000:0411h com o DEBUG.

Vejamos em detalhes como a BIOS executa uma tarefa. Vamos analisar a BIOS INT 11h. O resultado final desta interrupção é colocar no acumulador AX uma word capaz de identificar os dispositivos instalados, ver Apêndice D. Como já sabemos através das Tabelas 9.1 e 9.2, estas informações estão codificadas na área de dados da ROM-

BIOS. Para rastrear e desassemblar uma rotina de interrupção qualquer com o DEBUG é necessário:

1. Identificar na Tabela dos vetores de interrupção o endereço da rotina de serviço de interrupção.(ver Seção 2.8).
2. Desassemblar a rotina de serviço de interrupção.

Usando o DEBUG, é simples executar o primeiro passo, basta verificar o valor apontado em INT X 4, no caso, em $11h \times 4 = 44h$, verificando os 4 bytes existentes neste endereço, temos:

-D 0000:0044 L 4 ler na tabela de vetor de interrupção
4DF800F0 endereço da rotina , no caso F000:F84D

O endereço apontado é uma região de EPROM, ver Seção 2.5.1), com o DEBUG é possível desassemblar (Unassembler)

-U F000:F84D		, região da ROM BIOS
F000:F84D FB	ST	, desabilitar interrupção
F000:F84E 1E	PUSH DS	, armazenar DS e apontar
F000:F84F B84000	MOV AX,0040	, DS para a área de dados
F000:F852 8ED8	MOV DS,AX	, da BIOS 0040:0000
F000:F854 A11000	MOV AX,[0010]	, ler configuração dos equipamentos
F000:F857 1F	POP DS	, recuperar DS
F000:F859 CF	IRET	, fim da rotina de interrupção.

Usar o BIOS INT , neste exemplo, é equivalente a ler uma região de memória, porém é uma maneira mais compacta e confiável de realizar tal operação.

Uma série de outras informações importantes estão armazenadas na área de dados da BIOS. A Tabela 9.3 é um resumo de alguns valores de interesse. Estes dados estão no programa ACHAR_B.EXE que desenvolvemos em C cujo código fonte comentado está na Seção 9.5.

endereço (0000:)	tamanho	descrição
0400h	word	endereço porta serial 1
0402h	word	endereço porta serial 2
0408h	word	endereço porta paralela impressora 1
040Ah	word	endereço porta paralela impressora 2
0410h	byte	dispositivos de vídeo e drive
0411h	byte	dispositivos portas serial e paralela
0413h	word	dimensão da memória em Kbytes (0..640)
0417h	byte	controle de teclado-1, teclas especiais
0418h	byte	controle de teclado-2, teclas especiais
041Eh	32 bytes	buffer de teclado
0441h	byte	estado do disco flexível.
0449h	byte	modo de vídeo em uso
044Ah	word	largura tela de texto (40 ou 80 col)
0450h	8 words	indica posição do cursor em 8 páginas
0460h	word	tamanho do cursor (linha inicial e final)
0462h	byte	página de vídeo em uso
0463h	word	endereço inicial do contador de vídeo
046Ch	2 words	contador do timer, BIOS INT 1Ah.
0470h	byte	flag do relógio, maior que 0 após 24 hs.
0471h	byte	estado da tecla de break
0496h	byte	monitor do estado de teclado
0497h	byte	aviso de leds do teclado
0500h	byte	estado da tecla Print Screen

Tabela 9.5 *Resumo da área de dados do ROM BIOS*

Exemplo 9.3 Qual é o endereço da porta de impressora paralela?

Com o DEBUG, basta:

-D 0000:0408 L 2 , mostrar 2 bytes (word) neste endereço
0040:0000 7803 , endereço de I/O da Impressora 0378h

Exemplo 9.4 Qual é memória RAM livre para uso no PC?

A memória RAM é um dos componentes mais caros do PC. A quantidade de memória livre para uso está no endereço :

- D 0000:0413 L 2

8002 —————→ 640 Kbytes

No meu PC, a resposta indica 0280h Kbytes de RAM livres, ou em decimal, 640 Kbytes. A BIOS INT 12h também fornece esta informação. Seguindo o Exemplo 9.2, verifique com o DEBUG como o ROM-BIOS implementa esta rotina de serviço de interrupção.

9.4 Manipulando o Teclado

As rotinas de BIOS INT de vídeo (ver Seção 9.2) e teclado são as mais utilizadas em projetos de interfaceamento do PC, devido à rapidez no acesso aos dispositivos de hardware externo. Na manipulação com o Disk-Drive, é preferível o uso das rotinas DOS, ver Cap.10. Porém é aconselhável e mais seguro usar os recursos da linguagem de alto nível quando o fator tempo não for crucial nas operações com arquivos. Do mesmo modo o controle da porta serial e da impressora pode ser feito através da BIOS INT 14h e 17h respectivamente, porém as rotinas de linguagem de alto nível podem economizar tempo de programação, pois a maioria das configurações possíveis são facilmente programáveis. A maior dificuldade não reside tanto na capacidade de controlar esses periféricos, porém na elaboração de um programa para definir e testar um protocolo de comunicação seguro. A BIOS INT 5h, é uma interrupção por hardware, ver Cap.5, utilizada para enviar uma tela de vídeo para a impressora. No Cap.11, vamos utilizá-la para exemplificar como uma interrupção por hardware pode ser controlada por software.

A BIOS INT 1Ah não existia na versão original do PC, introduzida nas versões XT e posteriores. Esta rotina é um contador de pulsos do clock de 18.2 Hz do canal 0 do temporizador, ver Seção 2.3, e pode ser utilizada como base de tempo para uso em programação em tempo real. Parte destas tarefas com o relógio existem como funções DOS, ver Cap. 10. As funções DOS devem ser usadas prioritariamente aos BIOS INT, pois o DOS acompanha as evoluções dos recursos de hardware, prolongando a vida útil do programa.

A IBM estendeu o teclado original estabelecendo um conjunto especial de código ASCII estendido (ou ASCIIZ) para incorporar as teclas especiais, (F1,...F10, ALT,CTRL ,etc..) ver Seção 1.2.5 e Apêndice A. O estado destas teclas especiais é monitorado na memória da área de dados da BIOS e pode ser verificado através da BIOS INT 16h, função 02h

Exemplo 9.5 Determinar o estado das teclas de controle

usando INT 16H, função 02 (ver Apêndice D) e DEBUG

-A 100

MOV AH,02 , preparar função para leitura do

INT 16 , estado das teclas de controle.

INT 3 , fim de programa no DEBUG

R IP

- 100

G = 100 (configurar teclado antes de enter!)

verificar com o estado do acumulador AL.

bit	setados	configuração do teclado exibido por AL
0	= 1	chave shift direita ativa (pressionada)
1	= 1	chave shift esquerda ativa
2	= 1	chave CTRL ativa
3	= 1	chave ALT ativa
4	= 1	chave Scrool Lock ativa
5	= 1	chave Num Lock ativa
6	= 1	chave Caps Lock ativa
7	= 1	chave Insert ativa

Tabela 9.6 Estado das chaves de controle do teclado

Exemplos:

AL = 00 —————> nenhuma tecla especial acionada

AL = F2 —————> chaves Caps lock, Num lock, Scrool lock, Insert
on e Shift direita ativas.

É interessante verificar outras configurações. O estado destas teclas, armazenado em 0000:0417h , ver Tabela 9.5, é utilizado pelo PC para identificar qual é o caractere ASCII (por exemplo, maiúscula ou minúscula ? depende se Shift ou CAPS LOCK está ou não ativada).

No Exemplo 9.6, procuramos preencher uma lacuna. É muito limitada a difusão da documentação do código de teclado IBM estendido. Enquanto no Exemplo 7.8 limitamo-nos a mostrar os 255 caracteres ASCII convencionais de vídeo, no próximo vamos reproduzir no vídeo o caractere associado a uma tecla. Montamos o programa,

utilizando somente o DEBUG. Embora o tamanho do código seja um pouco longo, e os recursos disponíveis limitados, insistimos neste exemplo para completar a discussão sobre programação ASSEMBLY e a biblioteca do BIOS INT existentes no PC.

Exemplo 9.6 Desenhar no canto superior direito do vídeo, o código ASCII correspondente à tecla acionada. Imprimir 2 vezes quando se tratar de tecla especial correspondente ao código ASCII estendido.

Os passos a serem executados neste programa são:

1. Zerar o buffer de teclado
2. Posicionar o cursor
3. Se CTRL + ALT for acionada, então fim
4. Se nenhuma tecla for acionada, voltar ao passo 3
5. Se for tecla especial, preparar 2 impressões
6. Imprimir caractere ASCII
7. Voltar ao passo 1.

A listagem abaixo, é o programa fonte a ser montado pelo Assembler do DEBUG comentado. Salvar o programa como EX_9#6.COM tal como se encontra no disco de Exemplos que acompanha o livro.

-A 0100

CS:0100	CALL 0130	, ZERAR BUFFER DO TECLADO
CS:0103	CALL 0126	, POSICIONAR CURSOR
CS:0106	MOV AH,02	, verificar se CTRL + ALT estão acionadas
CS:0108	INT 16	, checando bits 2 e 3
CS:010A	CMP AL,0C	, se verdade, vá para o final
CS:010C	JZ 0148	, do contrário, continuar
CS:010E	MOV AH,01	, Se alguma tecla foi acionada,
CS:0110	INT 16	, executar rotina de impressão
CS:0112	JZ 0106	, Do contrário, esperar acionar teclado
CS:0114	MOV CX,001	, preparar rotina de impressão, 1 vez
CS:0117	CMP AL,00	, É código de teclado estendido?
CS:0119	JNZ 011E	, se não for, imprimir uma única vez
CS:011B	MOV AL,AH	, do contrário, captar código especial
CS:011D	INC CX	, preparando-se para imprimir 2 vezes
CS:011E	MOV AH,0A	, imprimir caractere na posição
CS:0120	INT 10	, apontada pelo cursor
CS:0122	NOP	,
CS:0123	JMP 100	, REINICIALIZAR PROCESSO
CS:0125	NOP	,
CS:0126	MOV BH,00	, sub-rotina POSICIONAR CURSOR
DS:0128	MOV DX,004E	, na linha 0 (DH = 00h)
DS:012B	MOV AH,02	, e coluna 78 (DL = 4Eh = '78)
DS:012D	INT 10	,
CS:012F	RET	,
CS:0130	PUSH BP	, sub-rotina ZERAR BUFFER DO TECLADO
CS:0131	MOV BP,SP	, preservando o apontador de base
CS:0133	MOV AH,01	, Se buffer do teclado vazio?
CS:0135	INT 16	, então fim da rotina
CS:0137	JZ 0143	, Do contrário,
CS:0139	MOV AH,00	, esvaziar caractere do buffer teclado
CS:013B	INT 16	,
CS:013D	MOV AH,01	, até buffer do teclado vazio
CS:013F	INT 16	,

Exemplo 9.6 Listagem do Programa EX_9#6.COM

CS:0141	JMP 137	, em loop contínuo
CS:0143	POP BP	, fim da rotina
CS:0144	XOR CX,CX	, zerando antes o contador
CS:0146	RET	,
CS:0147	NOP	,
CS:0148	MOV AH,4C	, sub-rotina FIM DO PROGRAMA
CS:014A	INT 21	, retornando ao DOS
CS:014C	NOP	,

Exemplo 9.6 Listagem do programa EX_9#6.COM (continuação)

Este exemplo mostra como é possível chamar rotinas tipo CALL diretamente dentro do Assembly. A incapacidade de implementar rótulos inviabiliza a confecção de programas mais complexos do que o mostrado no Exemplo 9.6. Uma tarefa tediosa é a leitura e a verificação de bits para obter informações a partir dos valores armazenados nos registradores. A partir do exemplo acima, é possível avaliar o volume de código necessário para mostrar uma tela texto de vídeo, caso utilizássemos somente linguagem ASSEMBLY. Escrever programas capazes de interfacear com o usuário em linguagem ASSEMBLY é um enorme desperdício de tempo. As linguagens de alto nível, como C ou Pascal, possuem um enorme repertório de rotinas e comandos desenvolvidos para este fim. O ponto ótimo pode ser alcançado quando o programador conseguir mesclar linguagem de alto nível, nas tarefas onde o tempo não for fator limitante (interação "off-line" com o usuário) e integrar pequenas rotinas em Assembly quando se fizer necessário. A capacidade de manipular bits individualmente da linguagem C oferece um excelente instrumento para este fim, tal como mostramos na próxima Seção.

9.5 Programando em C os serviços do ROM-BIOS

Uma das grandes vantagens da linguagem C é sua capacidade de interagir diretamente com os registradores. Nesta Seção, integraremos algumas rotinas em Assembly dos Exemplos anteriores em linguagem C, particularmente as rotinas de interrupção da ROM-BIOS. Procuramos escrever o programa fonte e comentá-lo, de modo a ser compreendido mesmo por um programador pouco familiar em C. Com um pouco de atenção será possível reproduzir os resultados finais desse programa utilizando somente o DEBUG.

A linguagem C é conhecida por ser uma linguagem de programação orientada para o objeto "OOP" ("Object-Oriented -Programming"). Os programas são voltados para a construção de pequenos módulos capazes de se conectarem e usados em diferentes partes do sistema ou em outras aplicações. O tempo é o recurso mais escasso

do programador. O leitor poderá economizar um valioso tempo, integrando os programas desenvolvidos em suas aplicações futuras. Para isto tivemos o cuidado de desenvolver pequenos módulos, onde as funções são definidas como protótipos de fácil integração para uso em diferentes programas.

Nesta Seção, apresentamos um programa fonte denominado ACHAR_B.C. Este programa é capaz de caracterizar os principais recursos disponíveis no PC tal como são detectados pelo BIOS. Trata-se de uma extensão dos Exemplos anteriores. O resultado é um programa capaz de identificar os principais recursos existentes num PC genérico.

Evitando a duplicação de código, desenvolvemos um módulo cabeçalho : CABEÇA.H, que deve ser incluído com outras bibliotecas do C. Através deste cabeçalho redefinimos os registradores, as constantes utilizadas e o protótipo das funções utilizadas num único bloco. O código em formato de caractere ASCII deste cabeçalho está no Apêndice F.

O código fonte foi editado e compilado com o utilitário Turbo C 2.0 (ver Cap. 6).

```

/*  Módulo principal para obter informações da ROM e BIOS
*
*  salvar este programa como ACHAR_B
*
*  OBJETIVO: caracterizar a marca registrada e data da versão ROM
*  chama outras rotinas para obter mais informações do BIOS
*/
#  define MAIN
#  include "CABEÇA.H"/* reúne todos os includes necessários */

void main (void)
{
    Info_rom();          /* leitura da ROM */
    info_bios();         /* funções BIOS de caracterização */
}

void Info_rom(void)
{
    char far *ROM;        /* define um apontador de caracter far */
    char marca [60];      /* reserva de 60 caracteres, marca do PC */
    char data [9];        /* reserva de 9 caracteres para data ROM */
    int i,j;              /* variáveis locais de loop */

    clrscr();             /* função padrão Turbo C de limpar video */

    MAKE_FP(ROM,0xf000,0); /* apontar para a base do segmento ROM */
    for (i = 0; i<+ +)
        data[i] = *(ROM + 0xfff5 + i); /* ler data a partir de F000:FFF5 */
        data[8] = '\0';               /* término da cadeia em C */
    for (j = 0; j<+ +)
        marca[j] = *(ROM + 0xe000 + j); /* ler marca a partir de F000:E000 */
        marca[60] = '\0';
    if (data[2] == 0x2f) /* checar se caracter '/' existe */
        printf (" o BIOS de seu computador tem a data : %s\n",data);
    else /* do contrário fabricante deixou ROM sem data*/
        printf (" versão ROM sem data\n");
}

```

```

printf (" A marca registrada do fabricante do seu PC :\n %s\n",marca);
}

/*          INFORMAÇÕES DA ÁREA DE DADOS DA BIOS
*
* esta função lê os dispositivos apontados na área de dados da BIOS
* ver Tabela 9.4 e 9.5 da Seção 9.3
*/

void info_bios(void)
{
    char far *data_bios;          /* apontar para área de dados da BIOS */
    char *mensagem;              /* mensagem de video, cadeia de char */
    int l,                        /* variável local índice de loop */
    np,                           /* número de portas paralelas */
    ns,                           /* número de portas seriais */
    nd;                           /* número de disk-drivers */
    MAKE_FP(data_bios,0x40,0);    /* apontar para área de dados BIOS 0:0400 */

    CALLINT(0x11);                /* BIOS INT 11h, retorna em AX lista */
/*          Porta Paralela da impressora */

    np = (AX & 0xC000) > 14;
    printf ("\t %d porta paralela instalada",np );
    for (l=0;l < np) {
        unsigned char Hpaddr = *(data_bios + 8 + l + 1);
        unsigned char Lpaddr = *(data_bios + 8 + l);
        printf("\t\t LPT%d %02x%02x h\n",l+1,Hpaddr,Lpaddr);}

/*          Porta Serial de comunicação */

    ns = (AX & 0x0e00) > 9;
    printf ("\t %d porta serial instalada",ns );
    for (l=0;l < ns) {
        unsigned char Hpaddr = *(data_bios + l + 1);
        unsigned char Lpaddr = *(data_bios + l);
        printf("\t\t COM%d %02x%02x h\n",l+1,Hpaddr,Lpaddr);}

```

```

/*                                     Disk - drivers                                     */
{
    nd = (AX & 0x00c0) > 6 ;
    printf ("\t %d unidades de discos flexíveis\n",nd + 1);
}

/*                                     Modo de vídeo                                     */
{
    switch ((AX & 0x0030) > 4) {
        case 0 : mensagem = " Provavelmente modo texto de placa EGA ";break;
        case 1 : mensagem = " Color/Graphics Card - CGA 40 colunas ";break;
        case 2 : mensagem = " Color/Graphics Card - CGA 80 colunas ";break;
        case 3 : mensagem = " Adaptador Monocromático - MDA 80 colunas";break;
    }
    printf ("\t modo de vídeo usual  %s\n",mensagem);
}

/*                                     miscelânea                                     */
printf ("\t adaptador de jogos %s instalado\n",
        (AX & 0x1000) ? "está": "não está");
printf ("\t modem interno do PC %s instalado\n",
        (AX & 0x2000) ? "está": "não está");
printf ("\t coprocessador matemático 8087 %s instalado\n\n",
        (AX & 0x0002) ? "está": "não está");

CALLINT (0x12);    /* BIOS INT 12H, obter tamanho memória                                     */
printf ("\t Você dispõe de %dK bytes de memória RAM Instalada\n",AX);
printf ("\n\n\n\n\n\t\t Qualquer tecla para continuar");
getch();
clrscr();
}

```

INTERRUPÇÕES E FUNÇÕES DOS

10.1 Interrupções DOS

A constante evolução do hardware do PC, em particular o aumento na capacidade do armazenamento em disco, ver Tabela 1.1, Cap. 1, obrigou um constante aperfeiçoamento do sistema operacional. Este fato é apontado pelas diversas versões DOS existentes. A rapidez no uso de diferentes versões apresenta um problema. Como compatibilizar um programa para as diferentes versões em uso? Embora a maioria dos recursos existentes em uma versão sejam incorporados numa versão posterior, algumas rotinas tornam-se obsoletas, tendo em vista o grau de aperfeiçoamento introduzido nas novas versões. Em todo caso, o uso das rotinas DOS é mais indicado que das rotinas BIOS, devido sua menor dependência com uma determinada configuração de hardware.

A partir da versão DOS 2.0, existe um alto grau de compatibilidade entre as diversas versões para a execução das rotinas básicas. Por outro lado, nem todo o potencial disponível nas versões mais recentes, tal como a manipulação de discos com alta capacidade, ou o gerenciamento de redes de microcomputadores, pode ser utilizado na maioria dos equipamentos instalados. A Tabela 10.1 resume as características das diferentes versões DOS.

versão	ano	característica de hardware
1.00	81	PC original, disco de face simples
1.10	82	disco de dupla face
2.00	83	PC-XT, introdução do disco rígido
3.00	84	PC-AT, discos de alta densidade
3.10	85	uso em rede multiusuário
3.30	87	modelo PS/2, discos de 3.5"
4.01	88	disco rígido de tamanho ilimitado, expansão de memória

Tabela 10.1 Diversas versões DOS.

O Exemplo 10.1, na Seção 10.3, mostra um pequeno programa capaz de identificar a versão DOS em uso.

As rotinas de interrupções e funções DOS executam outras tarefas além do controle das operações de disco, tais como: Controle do início e término dos programas, reservar espaço na memória para instalar programas, enviar caracteres ao vídeo, ler caracteres do teclado e outros serviços gerais de entrada e saída.

O sistema operacional é objeto de uma forte competição comercial como software de suporte do equipamento. Talvez por proteção ao código fonte a IBM e Microsoft deixem de divulgar a documentação completa a respeito de uma série de funções, rotulando-as simplesmente de reservadas. Estas funções, em geral, são utilizadas pelo COMMAND.COM e nem sempre guardam algum interesse especial para o usuário. Porém em programação avançada, particularmente na confecção e proteção de programas residentes TSR ("Terminante-and-Stay Resident"), algumas dessas interrupções podem ser importantes.

Ao contrário do BIOS, a filosofia do DOS consiste em aglutinar as rotinas em uma única interrupção, a DOS INT 21h (ou 33 em decimal). A Tabela 10.2 resume as interrupções DOS. Na próxima Seção, e no Apêndice E, as funções DOS são detalhadas.

Interrupção	comentário	versão DOS		
		2.0	3.0	3.3
20h	termina programa	?	?	?
21h	chama funções	s	s	s
22h	endereço de término de programa	s	s	s
23h	CTRL-Break endereço	s	s	s
24h	endereço de erro crítico	s	s	s
25h	leitura absoluta do disco	s	s	s
26h	gravação absoluta do disco	s	s	s
27h	termina e fica residente	?	?	?
28h	reservada	r	r	r
29h	reservada	r	r	r
2Ah	acesso a MS-rede	r	r	s
2Bh	reservada	r	r	r
2Ch	reservada	r	r	r
2Dh	reservada	r	r	r
2Eh	reservada	r	r	r
2Fh	impressora multiplex	-	s	s
33h	funções suporte de mouse	-	s	s
67h	gerenciador de expansão memória	-	s	s

Tabela 10.2: Interrupções DOS

Legenda: ? recurso disponível porém considerado obsoleto
 s recurso disponível na versão indicada
 - recurso não disponível na versão indicada
 r reservada, para futuro uso ou não documentada.

10.2 As funções DOS

Convencionamos chamar de funções DOS aquelas pertencentes a DOS INT 21h, cujo acesso é controlado pelo acumulador AH.

A Tabela 10.3 resume as principais funções DOS da versão 2.0, e posteriores, de controle de programa e entrada-saída. As funções de operação e controle de arquivos estão no Apêndice E.

número da função (AH =)	registradores de entrada	descrição	registradores de saída
Entrada e Saída de Caractere e cadeias			
01h	-	entrada de caractere sem eco para o vídeo	AL
02h	DL	saída de uma caractere para o vídeo	-
03h	-	entrada caractere porta serial/ou	AL
04h	DL	saída de um caractere para porta serial/ou	-
05h	DL	saída de um caractere para impressora/ou	-
06h	DL	ler caractere de teclado/ou	DL
07h	-	entrada de caractere sem filtro e eco	AL
08h	-	entrada de caractere sem eco para vídeo	AL
09h	DS:DX	enviar cadeia de caracteres vídeo/ou	-
0Ah	DS:DX	entrada do teclado com buffer	-
0Bh	-	verificar estado de entrada do teclado	AL
0Ch	AL DS:DX	limpar buffer do teclado e ler caractere disponível, retornando valor em AL	AL

Tabela 10.3 Resumo das Funções de DOS de controle

número da função (AH =)	registradores de entrada	descrição	registradores de saída
Controle de Programa			
00h	-	termina programa sem mensagem	-
26h	DX	cria novo segmento de programa (PSP)	-
31h	AL,DX	termina e fica residente (TSR)	-
33h	AL,DL	obtem ou define estado de CTRL + Break	DL
4Bh	AL ES:BX DS:DX	carrega e executa programa tipo *.EXE	se Ok! CF = 0
4Ch	AL	termina programa com mensagem de retorno	-
4Dh	-	obter código de retorno	AX
62h	-	obter endereço do prefixo de segmento de programa (PSP)	BX
Controle de memória			
48h	BX	reserva parágrafos (BX) de memória	CF,AX
49h	ES	libera segmento(ES)do bloco de memória	CF,AX
4Ah	BX,ES	modifica bloco de memória	CF,AX

Tabela 10.3 Resumo das Funções de DOS de controle (continuação)

número da função (AH =)	registradores de entrada	descrição	registradores de saída
Hora e data			
2Ah	-	obtem ano(CX) mês(DH) e dia(DL) e dia da semana(AL, 0 = domingo)	CX DX,AL
2Bh	CX,DX	define ano(CX) mês(DH) dia (DL)	AL
2Ch	-	obtem horas(CH) minutos(CL) segundos(DH) centésimos de segundos(DL), formato BCD	CX DX
2Dh	CX,DX	definir horas e minutos(CX) segundos(DH)	AL
57h	AL,BX	obter/definir hora e data de arquivo	CF,AX
Funções Diversas			
25h	AL, DS:DX	define vetor de interrupção (AL) no endereço (DS:DX) da rotina de serviço	-
30h	AL = 0	obter versão DOS principal(AL). secundária(AH)	AX
35h	AL	obter endereço (ES:BX) da interrupção(AL)	ES:BX
38h	AL, DS:DX	obter/definir informação do país, formato de data e símbolo ASCII moeda corrente	CF BX,DX

Tabela 10.3 Resumo das Funções de DOS de controle (continuação)

Destas funções a mais utilizada em nossos programas Exemplos é a função de término de programa 4Ch. Embora a DOS INT 20h realize função semelhante, ver Tabela 10.2, é um modo considerado obsoleto de terminar um programa.

A maior parte das funções DOS referem-se à manipulação de arquivos, excluídos da Tabela 10.3 e mostrados no Apêndice E. Manipular arquivos diretamente através do DOS não é uma atitude prudente. É muito mais seguro, e produtivo, utilizar as rotinas existentes nas linguagens de alto nível.

Uma estratégia sofisticada em programação DOS consiste em desenvolver programas residentes. Estes programas ficam ocultos durante a execução do programa principal e são chamados a operar através de uma interrupção de hardware ou uma combinação de teclas especiais ("hot key"), tal como o popular utilitário Borland "Sidekick". No próximo Capítulo, apresentamos um exemplo deste tipo, cujo controle ao DOS é passado através da função 31h.

Interceptar através da função 35h e redefinir o vetor de interrupção, função 25h, é um recurso muito útil de programação de interfaces do PC. Por exemplo, a interrupção do temporizador , INT 1Ch (ver Seção 2.7 e Exemplo 7.4), pode ser desviada para executar uma rotina de interrupção numa frequência de 18.2 Hz para programação em tempo real.

10.3 Obtendo informações através do DOS

As informações fornecidas pelas funções DOS são mais resumidas que as da ROM-BIOS (ver Seção 9.3 e 9.5). Os Exemplos montados nesta Seção, em Assembly, com o DEBUG, exploram algumas informações DOS de interesse. Uma extensão desses programas em linguagem C é o programa fonte ACHAR_.D.C do próximo Capítulo.

Exemplo 10.1 Qual é a versão DOS em uso?

Um pequeno programa executa esta tarefa dentro do ambiente do DEBUG. Por exemplo no meu sistema:

```
-A 100
CS:0100  MOV AH,30          ; função DOS 30h capaz de identificar
CS:0102  INT 21             ; versão DOS em uso
CS:0104  INT 3              ; fim do programa no DEBUG
```

G = 100

-R

AX = 0003

Observe retorno de AX: AH = 00 e AL = 03. A versão DOS em uso é 3.00 (AL.AH).

No Exemplo 7.6, utilizamos o DEBUG para armazenar uma cadeia de caracteres. O DOS possui um recurso capaz de enviar uma cadeia ("string") de caracteres para o vídeo.

Exemplo 10.2 Escrever uma cadeia de caracteres no vídeo

Aproveitando o Exemplo 7.6, vamos escrever a mensagem "Erica Editora" num buffer de dados com ponteiro de base em DS:0200 e enviar esta mensagem ao vídeo através da função 09h. Observe o detalhe de incluir o caractere \$ no final da mensagem para identificar seu término.

escrever a mensagem em DS:0200

F DS:0200 L 10 "ERICA - EDITORA\$"

montar e executar o programa:

-A 100

CS:0100	MOV DX,0200	; apontar para base do buffer
CS:0103	MOV AH,09	; da mensagem, chamando a função
CS:0105	INT 21	; 09h para imprimir cadeia
CS:0109	MOV AH,4C	; retornando ao DOS
CS:010B	INT 21	

Exemplo 10.2 Listagem do programa EX_10#2.COM

executar o programa a partir do DOS

A > EX_10#2.COM
ERICA - EDITORA

O Exemplo 10.4 vai criar uma rotina de interrupção especial. Vamos definir uma rotina de serviço de interrupção, 60h, capaz de limpar a página texto de vídeo. Para isto vamos usar o programa apresentado no Exemplo 7.7, Seção 7.6, para limpar tela.

Exemplo 10.4 Instalar e definir uma rotina de interrupção especial, capaz de limpar a tela de vídeo.

```
-A 100
CS:0100 CALL 010D      , INSTALAR NOVA ROTINA
CS:0103 NOP            ,
CS:0104 NOP            ,
CS:0105 INT 60          , Testar nova rotina
CS:0107 MOV AH,4C       , retornar ao DOS
CS:0109 INT 21          ,
CS:010B NOP            ,
CS:010C NOP            , sub-rotina de instalação
CS:010D MOV DX,0119     , apontar CS:DX para início da nova
CS:0110 MOV AL,60       , rotina de interrupção 60h
CS:0112 MOV AH,25       , função DOS 25h, escrever endereço
CS:0114 INT 21          , nova rotina na Tabela Vetor INT
CS:0116 RET             , FIM DA ROTINA DE INSTALAÇÃO
CS:0117 NOP            ,
CS:0118 NOP            , NOVA ROTINA DE INTERRUPÇÃO
CS:0119 MOV AH,0F       , obter modo de vídeo corrente, função
CS:011B INT 10          , vídeo 0Fh, BIOS INT 10h, em AL
CS:011D MOV AH,00       , redefinir o mesmo modo de vídeo,
CS:011F INT 10          , limpando o buffer de vídeo
CS:0121 NOP            ,
CS:0122 MOV DL,07       , soando um apito, através do código
CS:0124 MOV AH,02       , ASCII de controle 07, Bell
CS:0126 INT 21          ,
CS:0128 IRET           , RETORNAR À ROTINA DE INT.
```

Exemplo 10.4 Listagem do programa EX_10#4.COM

O programa pode ser chamado diretamente a partir do DOS. Este Exemplo ilustra muito bem o modo operacional do DOS. Ao carregar o sistema operacional ("boot") os programas "ocultos" IBMBIO.COM e IBMDOS.COM carregam a tabela do vetor de interrupção, tal como nosso primeiro módulo no Exemplo acima. Deste modo, cada vez que um programa for executar uma interrupção, o comando passa para a rotina definida na tabela do vetor de interrupção. Em nosso caso, isto pode ser verificado após uma execução do programa, chamar o DEBUG EX_10#4.COM e verificar o vetor no endereço"

-D 0000:0180 L 4 (60h x 4 = 180h)

19 01 08 0C —————> 0C08:0119

e o endereço apontado será o da nossa rotina o que pode ser verificado com:

```
-U 0C08:0119 I 11
0C08:0119 B40F      MOV AH,0F
0C08:011B CD10      INT 10
0C08:011D B400      MOV AH,00
0C08:011F CD10      INT 10
0C08:0121 90        NOP
0C08:0122 B207      MOV DL,07
0C08:0124 B402      MOV AH,02
0C08:0126 CD21      INT 21
0C08:0128 CF        IRET
```

Nova rotina de interrupção desassemblada

Este endereço não é absoluto, pois o DOS reserva uma região de memória para colocar o programa de acordo com a memória utilizada, em particular por outros programas residentes.

Percebe-se aqui o caminho mais fácil para criar confusão nos programas alheios. Basta redefinir uma nova rotina de interrupção, associada à uma data (sexta-feira 13), ou ao acesso ao disco ou à chamada de um programa tipo *.EXE através de um programa "oculto". Deste modo, cada vez que uma destas interrupções for solicitada, o apontador de instrução passará a executar uma rotina adulterada, diferente da usual. Se este programa "oculto" conseguir ficar agregado ao sistema operacional, ele terá capacidade de se propagar e infectar outros sistemas. A maneira mais simples de evitar este dano é prevenir-se. Ao trabalhar com um computador, que pode estar contaminado, carregue o sistema operacional não adulterado, a partir de um disco confiável. Ao usar um disco de origem duvidosa, não utilizar seu sistema operacional, e desligar o computador após uso. Deste modo, mantendo a tabela do vetor de interrupção intacta.

10.4 Os serviços DOS na linguagem C

O programa cabeçalho header.h do Apêndice F redefine a DOS INT 21 por: CALLDOS. O programa fonte ACHAR_D.C descreve módulos de programação C, capazes de obter, e apresentar, as informações que podem ser obtidas com as funções DOS.

Como resultado final o programa exibe hora e data e os recursos de "disk-drivers" instalados tanto físico como lógico.

```

/*  Módulo principal para obter informações do DOS
*
*  Salvar este programa como Achar_D

*  OBJETIVO: caracterizar a versão DOS em uso e uso de outras funções
*  tais como: data, caracterização de arquivos
*/
#define MAIN
#include "CABEÇA.H"                /* reúne todos os includes necessários */
                                   /* prótipos das funções de ACHAR_D */

static void data_hora (void);      /* obter data e hora,
função DOS 2Ah                      */
static void versão (void);         /* obter versão dos,
função DOS 30h                      */
static void Info_disco(void);      /* informações de arquivos em disco */

/*                                variáveis global
*/
int versão_dos;                   /* versão dos, obtida
em versão()                       */

void main (void)
{
    clrscr();
    data_hora();
    versão();
    esperar();
    info_disco();
}
void esperar (void)                /* função de controle de tela */
{
    printf ("\n\n\n\n\n\t\t Qualquer tecla para continuar");
    getch();
    clrscr();
}

void data_hora (void)              /* obtém data e hora através do DOS */
{
    char *dia [ ] =
{"Domingo", "Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado"};

```



```

AH = 0x2a;
CALLDOS;
printf ("\t hoje e %s, %02d/%02d/%02d\n", dia[AL], DH, DL, CX - 1900);

AH = 0x2c;
CALLDOS;
printf ("\n\t a hora
%02d:%02d:%02d.%02d\n", CH, CL, DH, DL);
}

void versão (void)
{
AH = 0x30;
CALLDOS;
versão_dos = AL*100 + AH;      /* para uso em info_disco()          */
printf ("\n\t Você está usando a versão DOS %d.%02d\n", AL,AH);
}

void info_disco(void)
{
int ndrive,i;                  /* variáveis locais auxiliares */
*/
char diretório [64];          /* diretório ativo              */
/

AH = 0x19;                     /* função DOS 19h
CALLDOS;
printf ("Drive corrente em uso %c\n", AL + 'A');

DL = AL;                        /* selecionar drive em uso      */
/
Ah = 0x0e;                      /* através da função DOS 0Eh    */
CALLDOS;
if (versão_dos < 300)           /* para versões anterior a 3.0  */
printf ("Existe%d drivers físicos\n", AL);
else                            /* do contrário drivers lógicos\n", AL;
printf ("Existe%d drivers lógicos\n", AL);
ndrive = AL;
for (i=0; i < ndrive; i++)
{
AH = 0x36;                     /* obter capacidade do disco    */
/

```

```

DL = (char) i + 1;          /* para o drive i (A,B,C...)          */
CALLDOS;                   /* através função DOS36h          */
    if (AX == 0xffff)
    { printf ("\n Drive%c não está conectado\n", i + 'A');
      esperar();
    }
    else
    {
      printf ("\n\t Drive \t\t\t %c\n", i + 'A');
      printf ("\n\t bytes por setor \t:%u\n", CX);
      printf ("\n\t Setores por clusters\t:%u\n", AX);
      printf ("\n\t Total de clusters \t:%u\n", DX);
      printf ("\n\t espaço disponível \t:%1u bytes livres\n", (long) BX*AX*CX);
      SI = (unsigned) diretório;
      DL = (char) i + 1:          /* obter diretório corrente          *
/
      AH = 0x47;                 /* através da função DOS47h          */
      CALLDOS;
      printf ("\n\t O diretório corrente em uso e %c:\\%s\n", i + 'A', diretório);
      esperar();
    }
  }
}

```

PROGRAMAS APLICATIVOS AVANÇADOS

11.1 Introdução

Conhecer e explorar os recursos íntimos do PC é uma tarefa atraente e tentadora embora nem sempre bem sucedida. Neste Capítulo, mostramos 4 programas de interesse em projetos de interface do PC com o mundo real.

Programar a porta paralela interna PPI- 8255, ver Seção 2.9, é o primeiro passo para programação de uma interface de entrada e saída (I/O) qualquer. A programação do próprio temporizador do PC - 8253 como base de tempo, ver Seção 2.3, é o princípio da programação em tempo real. Neste caso, a criatividade do programador é o único limite para a produção de programas de utilidade, tal como a aquisição de dados em tempo real, ou de diversão, como música controlada por microcomputador. A capacidade de manipular interrupção é a base da programação distribuída em multi-tarefas. As peculiaridades de programas residentes, em particular os de acesso instantâneo ("pop up"), são tratadas de modo superficial na maioria dos textos elementares. Como se sabe, esta é a chave para confecção de programas comerciais de sucesso.

Evitando uma discussão longa sobre estes temas, mostramos alguns pequenos programas exemplos capazes de ilustrar a potencialidade do PC em interface com o mundo real. Utilizamos somente o DEBUG, ver Seção 7.2, para montar estes códigos. Como se trata de um utilitário de limitados recursos, os programas devem ser compreendidos como um embrião de outros mais completos, tal é o caso do relógio de vídeo em tempo real. Esperamos com isto estar fornecendo uma base para o leitor construir e projetar seus próprios programas, com os mesmos efeitos dos programas comerciais.

11.2 Fazendo do Teclado um órgão.

A interface da CPU e o exterior é feita através da porta paralela PPI 8255, ver Seção 2.9. Dois bits desta porta têm a curiosa tarefa de controlar o alto-falante existente no PC. Vamos utilizar a capacidade de áudio do PC para ilustrar como esta porta

paralela, localizada na placa principal, pode ser programada. Numa outra oportunidade, ou melhor num próximo volume, veremos como uma porta semelhante pode ser conectada a um slot de expansão do PC, abrindo uma importante porta de interfaceamento do PC com o mundo exterior.

A Figura 2.8, Seção 2.6, mostra como a saída do canal 2 do temporizador 8253 e a interface paralela 8255 controlam o alto-falante. Na configuração do PC, as portas A,B,C e o modo de controle já estão programados e não podem ser alterados sob pena de desabilitar o teclado e consequente pane geral do microcomputador. Ver na Tabela 2.3, Seção 2.9, os endereços desses registradores.

Os 2 bits menos significativos da porta paralela do 8255 controlam o alto-falante de 2 modos diferentes:

PB0 : habilita, ou não, a saída do canal 2 do temporizador 8253.

PB1 : liga ou desliga o alto-falante.

Ligar e desligar o alto-falante periodicamente produz um som de uma determinada frequência. Porém deve-se ter cautelas. A porta B do 8255 tem outras funções para executar além de controlar o alto-falante, ver Figura 2.9, Seção 2.9. Portanto é necessário preservar o estado inicial dos bits da porta B.

Exemplo 11.1 Programar a porta paralela para ligar e desligar o alto-falante, um certo número de vezes, de modo a produzir um som.

O ouvido humano é capaz de distinguir sons na faixa de frequência de 20 Hz (grave) a 20.000 Hz (agudo). Para ser perceptível é necessário que o som tenha uma certa duração. Deste modo, o programa deve ter as seguintes etapas:

- (a) Inicialização, definição da duração do som
- (b) Desligar o alto-falante por um determinado tempo
- (c) Ligar o alto-falante por um certo período
- (d) Continuar a operação até o fim do intervalo

CS:0100	IN AL,61	; manter estado inicial da porta B.
CS:0102	MOV BX,0100	; definir o número de ciclos lga/
CS:0105	NOP	; deslga = 100h
CS:0106	NOP	; DESLIGAR
CS:0107	AND AL,FC	; mascarar bits da porta B
CS:0109	OUT 61, AL	; desligar alto-falante, PB0 = PB1 = 0
CS:010B	CALL 0124	; chamar rotina de espera
CS:010E	NOP	;
CS:010F	NOP	; LIGAR
CS:0110	OR AL,02	; mascarar bits da porta B, ativando
CS:0112	OUT 61,AL	; alto-falante (PB1 = 1)
CS:0114	CALL 0124	; chamar rotina de espera
CS:0117	NOP	;
CS:0118	NOP	; FIM ?
CS:0119	DEC BX	; Se não for fim do ciclo de
CS:011A	JNZ 0107	; operações, então continuar loop
CS:011C	AND AL,FC	; Do contrário terminar, recuperando
CS:011E	OUT 61,AL	; o estado inicial da porta B
CS:0120	MOV AH,4C	; retornar ao DOS através da
CS:0122	INT 21	; função 4Ch.
CS:0124	MOV CX,0400	; ROTINA DE ESPERA
CS:0127	LOOP 0127	; decrementar CX 400h vezes antes
CS:0129	RET	; retornar
CS:012A	NOP	;

Exemplo 11.1 Listagem EX_11#1.COM

Produzir sons ligando e desligando o alto-falante é uma maneira um tanto primitiva. Um modo mais elegante seria deixar o alto-falante habilitado e permitir que o canal 2 do temporizador envie uma onda na frequência de áudio. Deste modo, para produzir sons de uma determinada frequência basta programar o canal 2 do temporizador, carregando no seu registrador o divisor de frequência N adequado e:

$$\text{frequência do som} = (1.193.180 \text{ Hz})/N$$

Como o registrador do temporizador 8253 é de 16 bits, o valor máximo de N = 65536. Este valor permite produzir o som mais grave no PC:

$$f = 1.193.180 \text{ Hz}/65536 = 18.2 \text{ Hz.}$$

Esta frequência é a base de tempo do relógio interno do PC, ver Exemplo 11.3 Seção 11.2 .

Tomando os devidos cuidados, o temporizador 8253 interno do PC pode ser programado, em particular o canal 2 cuja única função é enviar um trem de pulsos para o alto-falante. Ver Seção 2.6 e na Tabela 2.1, os endereços do temporizador.

Em vez de simplesmente enviar um trem de pulsos propomos um exemplo mais interessante. Para cada valor de código ASCII, ver Apêndice A, enviado pelo teclado, pode-se associar uma determinada frequência. Deste modo, a cada tecla está associada uma nota específica. A duração do som deve ser controlada de modo a ser perceptível. O único modo de terminar o programa é através da interrupção forçada de CTRL + C.

Exemplo 11.2 Fazer do teclado do PC um mini órgão, programando o temporizador interno.

CS:0100	MOV AL,B6	; programar o canal 2 do 8253 no
CS:0102	OUT 43,AL	; modo de onda quadrada contínua
CS:0104	NOP	;
CS:0105	XOR AL,AL	; carregar parte menos significativa
CS:0107	OUT 42,AL	; primeiro com qualquer valor,
CS:0109	NOP	; no caso zero
CS:010A	MOV AH,01	; esperar acionamento do teclado na
CS:010C	INT 21	; função que lê caractere e ecoa
CS:010E	NOP	; no vídeo, função DOS 01h
CS:010F	NOP	;
CS:0110	OUT 42,AL	; programar parte alta do 8253 com
CS:0112	NOP	; valor ASCII recebido do teclado
CS:0113	IN AX,61	; mascarar bits da 8255, Porta B
CS:0115	OR AX,0003	; ligando alto-falante (PB0 = PB1 = 1)
CS:0118	OUT 61,AL	;
CS:011A	CALL 0127	; loop de espera, mantendo ativado
CS:011D	NOP	; o alto-falante
CS:011E	AND AX,FFFC	; desligar o alto-falante
CS:0121	OUT 61,AL	;
CS:0123	JMP 0104	; continuar loop até interrupção
CS:0125	NOP	; forçada CTRL + C ou CTRL + BREAK
CS:0126	NOP	;
CS:0127	MOV CX,FFFF	; ROTINA DE ESPERAR LIGADO
CS:012A	LOOP 012A	; decrementar CX 64 K passos
CS:012C	RET	; antes de retornar

Exemplo 11.2 Listagem programa EX_11#2.COM

A palavra do controle do temporizador:

B6h = 10 11 011 0 em binário, deve ser decodificada como:

- contador 2 (bit 7 = 1 e bit 6 = 0, 10)
- carregar primeiro o byte menos significativo no contador, (bit 5 = bit 4 = 1)
- selecionar modo 3 de operação, onda quadrada contínua (bit 3 = 0, bit 2 = 1 e bit 1 = 1; 011)
- decrementar no modo binário e não decimal (bit 0 = 0)

O programa em si é um loop infinito só interrompido através de CTRL + C ou CTRL + BREAK.

O alto-falante do PC não reproduz igualmente todos os níveis de frequência. O resultado é um órgão de qualidade sonora duvidosa, que varia para cada microcomputador, dependendo do barulho produzindo pelo ventilador da fonte de alimentação e do alto-falante instalado.

11.3 Alterando uma interrupção de hardware

O núcleo central da programação do PC reside nas interrupções. Nos exemplos anteriores analisados, é o PC que comanda a comunicação com o meio exterior. Nos processos de monitoramento e controle, ao contrário, é um evento externo que solicita a atenção do PC para a execução de uma rotina de interrupção.

A interação entre o PC e dispositivos externos de controle pode ser feita através do processo de "pooling" ou de interrupção externa priorizada. No primeiro caso, é o PC que comanda um conjunto "pool" de dispositivos externos, verificando periodicamente se existe ou não alguma tarefa a ser executada, Figura 11.1

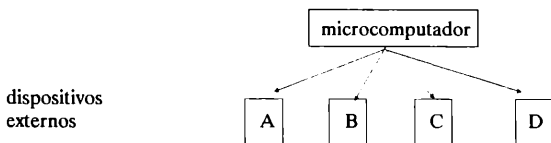


Figura 11.1 Sistema "pooling" de interfaceamento

Em algumas situações, nem sempre esta solução é a mais indicada. O PC precisa ficar dedicado exclusivamente a esta tarefa. No monitoramento de eventos externos aleatórios, ou eventuais, tal como um sistema de alarme contra incêndios, a maior parte do tempo o microcomputador não realiza tarefa alguma. Neste tempo morto o microcomputador poderia estar realizando uma outra tarefa, ou seja ele poderia trabalhar no modo de multitarefa. Nestes casos, o processo de interrupção com prioridades é o mais indicado, Figura 11.2

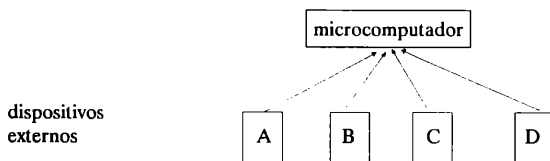


Figura 11.2 Interfaceamento por interrupção

As entradas dos sinais de interrupção externos já estão disponíveis nos slots do PC, ver Figura 5.2 Seção 5.2 e Cap. 4. Por exemplo, a entrada do sinal de interrupção por hardware, controlada pelo 8259 IRQ 02, ou INT 0Ah, ver Tabela 5.1 Seção 5.2, está no pino B4 do slot, ver Tabela 4.1 e Figura 4.1 na Seção 4.1. Para utilizá-la basta instalar uma rotina de interrupção apontada na Tabela dos Vetores de Interrupção ($4 \times 0Ah = 28h$). Criar uma rotina para manipular esta interrupção é simples, tal como descreve o Exemplo 10.4, Seção 10.3

Um projeto audacioso seria interceptar e redefinir uma das interrupções usuais já existentes no PC, por exemplo a interrupção de teclado, IRQ 1 ou INT 09h, ou do temporizador IRQ 0 ou INT 08h. Neste caso, toda vez que uma destas interrupções for solicitada, o PC passa executar uma rotina definida pelo usuário, podendo retornar, ao final, à tarefa usual ("default"). Uma das grandes vantagens do PC é a possibilidade de alterar, por software, os endereços apontados na Tabela do Vetor de Interrupção. Esta Tabela está na parte baixa da memória e é preenchida quando o sistema está sendo carregado através do programa IBMBIO.COM, ver Seção 2.5.2. Esta facilidade inicial tornou-se um problema posteriormente, pois possibilitou a introdução de programas residentes tipo "virus". Estes programas são carregados junto com o sistema operacional, alteram as rotinas usuais de carregamento e às vezes sob algum critério (conforme data, número de gravações de um determinado programa, etc...) executam tarefas desagradáveis, tal como adulterar a tabela de alocação de arquivos FAT ("File Allocation Table") destruindo os arquivos em disco.

Interceptar uma rotina de interrupção, Figura 11.3, é o modo de ativar programas residentes de efeitos imediatos ("pop up").

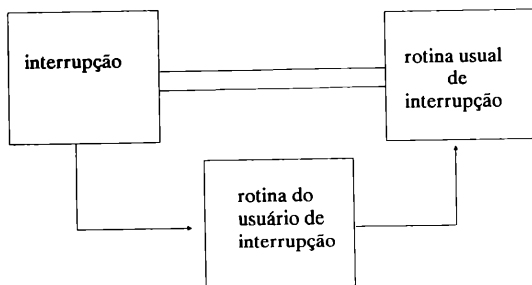


Figura 11.3 Intercepção de uma rotina de interrupção

Em processo de monitoramento e simulação é necessário à execução de um programa em tempo real. Isto pode ser feito através de uma interrupção por hardware através de um temporizador externo. Por exemplo, a cada segundo o PC pode ser solicitado a ler o valor de um sensor de temperatura conectado em um dos slots. Neste caso, periodicamente, um temporizador produz uma interrupção de hardware, solicitando a execução de uma determinada rotina.

O PC possui um temporizador interno que produz uma interrupção por hardware, IRQ 0 ou INT 08h, 18 vezes por segundo, ou aproximadamente a cada 55 milissegundos, ver Seção 2.3. Esta base de tempo do PC pode ser usada na programação em tempo real quando o intervalo entre as ações for um tempo superior a 0,1 segundos (este intervalo pode ser reprogramado para intervalos mais curtos, porém isto deve ser feito com cautela pois o canal 0 do temporizador tem outras funções, ver Seção 2.6). No projeto do PC, esta interrupção é contínua e serve para incrementar o relógio interno do sistema. Além disto, esta rotina de interrupção solicita uma outra interrupção por hardware, a BIOS INT 1Ch. Esta interrupção BIOS é um gancho que o projetista deixou para futuras implementações, pois só contém uma única instrução, a de retorno, ver Exemplo 7.4 na Seção 7.4. Deste modo é possível usar indiretamente esta interrupção por hardware, em tempo real, para a implementação de uma nova rotina sem alterar a configuração do sistema, tal como propõe o Exemplo 11.3

Exemplo 11.3 Alterar a interrupção de hardware do temporizador (1Ch) de modo a mostrar as unidades de segundos.

Antes de chamar a interrupção 1Ch, a interrupção 08h do canal 0 do temporizador atualiza o contador do relógio na área de dados da BIOS no endereço 0000:046Ch, ver Tabela 9.5 na Seção 9.3.

O programa deste exemplo foi montado exclusivamente com o DEBUG e está dividido em 3 módulos:

1. Rotina de limpar tela, posicionar cursor e enviar mensagem.
2. Manipulação da rotina em tempo real
3. Intercepção e instalação de nova rotina de interrupção.

Este programa tem a finalidade de ilustrar o modo de alterar uma interrupção por hardware e realçar a importância de programas residentes. Deve ser considerado um programa experimental embrião de um programa de display do relógio no canto superior da tela em tempo real.

CS:0100	CALL 015B	; chamar rotina de MENSAGEM
CS:0103	NOP	;
CS:0104	JMP 0143	; interceptar e redefinir nova
CS:0106	NOP	; rotina de interrupção
CS:0107	NOP	;
CS:0108	PUSHF	;
CS:0109	PUSH AX	PROGRAMA PRINCIPAL
CS:010A	PUSH BX	;
CS:010B	PUSH CX	; a ser executado 18 vezes por
CS:010C	PUSH DX	; segundo, através da redefinição
CS:010D	PUSH DS	; da BIOS INT 1Ch.
CS:010E	PUSH BP	; salvar na pilha os registradores
CS:010F	PUSH ES	; principais.
CS:0110	MOV AX,0000	;
CS:0113	MOV DS,AX	; apontar para área de dados BIOS
CS:0115	MOV DX,[046C]	; para obter valor do contador de
CS:0119	MOV AX,B800	; relógio no endereço 0000:046C
CS:011C	MOV DS,AX	;
CS:011E	MOV BX,0098	; apontar para área de vídeo
CS:0121	MOV CX,0004	; na posição de segundos
CS:0124	XHG AX,DX	;
CS:0125	SHR AX,CL	; dividir por 16 o valor do
CS:0127	AND AL,0F	; contador para mostrar 1 segundo
CS:0129	CMP AL,0A	; aproximadamente, considerando
CS:012B	JL 0130	; somente o dígito da unidade,
CS:012D	DAA	; fazendo o ajuste decimal se
CS:0130	ADD AL,30	; necessário, além do ajuste do
CS:0132	MOV AH,07	; código ASCII correspondente
CS:0134	MOV [BX],AX	; antes de enviar o caractere
CS:0136	POP ES	; para o vídeo
		; fim da operação

CS:0137	POP BP	; recuperar os registros na ordem
CS:0138	POP DS	; último entrar, primeiro a sair
CS:0139	POP DX	; LIFO ("Last In First Out")
CS:013A	POP CX	;
CS:013B	POP BX	;
CS:013C	POP AX	; antes de retornar à rotina usual
CS:013D	POPF	;
CS:013E	NOP	; no caso trata-se de um simples
CS:013F	NOP	; retorno de interrupção
CS:0140	IRET	;
CS:0141	NOP	; INTERCEPTAR VETOR INTERRUPÇÃO
CS:0142	NOP	;
CS:0143	CLI	; desabilitar interrupções por um
CS:0144	PUSH DS	; momento, salvando registrador DS
CS:0145	XOR AX,AX	; que será apontado para área da
CS:0147	MOV DS,AX	; Tabela do Vetor de Interrupção
CS:0148	MOV AX,0108	; carregando primeiro o "off-set"
CS:014C	MOV [0070],AX	; na tabela (70h = 4 x 1Ch)
CS:014F	MOV [0072],CS	; e o código de segmento para o
CS:0153	POP DS	; endereço CS:0108
CS:0154	STI	; reabilitar interrupção
CS:0155	MOV AH,4C	; terminar o programando retornando
CS:0157	INT 21	; ao DOS através da função 4Ch.
CS:0159	NOP	;
CS:015A	NOP	; ROTINA MENSAGEM DE TELA
CS:015B	MOV AH,0F	; limpando a tela, usando o BIOS
CS:015D	INT 10	; primeiro, obtendo o modo de vídeo
CS:015F	MOV AH,00	; corrente em AL e depois redefinir
CS:0161	INT 10	; o mesmo modo de vídeo, zerando o
CS:0163	XOR BX,BX	; buffer da memória de vídeo,
CS:0165	MOV DX,003E	; posicionando o cursor na primeira
CS:0168	MOV AH,02	; linha na coluna início da
CS:016A	INT 10	; mensagem
CS:016C	MOV DX,0176	; apontada no endereço 0176,
CS:016F	MOV AH,09	; enviando cadeia de caracteres ao
CS:0171	INT 21	; vídeo através da função DOS 09h.
CS:0173	RET	; fim da rotina mensagem de vídeo
CS:0174	NOP	

-D 176 L40

CS:0176	484F5241203D20203A	HORA =
CS:0180	20203A20202E2020-2020202020435549	: . CUI
CS:0190	4441444F2C204441-52204F205245424F	DADO,DAR O REBO
CS:01A0	4F5420434F4D2043-54524C2B414C542B	OT COM CTRL + ALT +
CS:01B0	44454C202448	DEL \$

Exemplo 11.3 Listagem programa EX_11#3.COM

Cuidado !!! Este programa é experimental e o usuário deve usar um disco de trabalho específico. Deve ser carregado a partir do DOS prompt e interrompido através de reinicialização do sistema CTRL + ALT + DEL

Programas de manipulação de interrupção são um pouco confusos pois eles não obedecem ao fluxograma usual em programação. Neste Exemplo, após executar a rotina de mensagem de tela, o apontador de instrução dá um salto incondicional (CS:0104 JMP) e deixa de executar o bloco de programa principal (entre CS:0108 e 0140), terminando com uma instrução de retorno de interrupção (CS:0140 IRET) que, ademais, não é chamada dentro do programa.

A rotina de mensagem é um exemplo das limitações da manipulação de cadeias no vídeo em linguagem ASSEMBLY. Para escrever uma simples mensagem é necessário limpar a tela, posicionar o cursor e apontar para uma cadeia de caracteres. Usar as rotinas do BIOS, através da redefinição do modo corrente de vídeo, é um processo mais geral que o apresentado no Exemplo 7.7, Seção 7.6. A mensagem enviada é simplesmente um aviso de como o operador deve terminar o programa, que neste caso só pode ser através da reinicialização do sistema, ("boot"), desligando o microcomputador e religando ou através de CTRL + ALT + DEL.

Neste Exemplo, antes de executar o programa principal o apontador de instruções salta para uma rotina de instalação e retorna ao DOS. Quando este programa principal será então executado? Simples, trata-se de uma interrupção por hardware. A cada 55 ms a CPU acusa a solicitação de executar uma rotina de interrupção 1Ch. A CPU procura, na Tabela do vetor de interrupção, o endereço da rotina, ver Figura 2.8 na Seção 2.8. Neste caso a rotina usual foi interceptada e desviada para a porção de código com início em CS:0108

A rotina do programa principal(entre CS:0108 e CS:0140) é somente um embrião de um programa maior, capaz de mostrar, no vídeo, um relógio em tempo real residente, que o leitor pode desenvolver como exercício, ou estudar o programa fonte em Assembly no disco que acompanha o livro. O Debug não permite a introdução de rótulos "labels" nem de macros o que estende muito o código da listagem, excedendo o espaço aqui disponível. Por isto mostramos um aperitivo, usando o próprio contador do relógio do PC localizado na área de dados do BIOS (0000:0406, Ver Tabela 9.5, Seção 9.3), tal como faz a BIOS INT 1Ah e a função DOS 2Ch. Contudo, as técnicas de apontar o segmento de dados para a tabela do vetor de interrupção, aproveitar informações já existentes na área de dados da BIOS, ficam evidenciadas neste pedaço de código.

Para mostrar os dígitos de unidades de segundos em valores aproximados foi necessário:

- (a) obter o valor do contador do relógio
- (b) dividir este valor por 16 (o mais correto seria 18)
- (c) fazer o ajuste decimal caso necessário
- (d) adaptar o caractere ASCII correspondente
- (e) enviar valor para posição de unidades de segundos

Um modo alternativo seria a criação de 3 contadores em uma região de memória para registro dos valores de segundos, minutos e horas. Esta opção mais correta implica num código mais extenso além dos limites aqui disponíveis. Veja programa fonte do relógio em tempo real residente do disco.

Como instalar e interceptar uma interrupção por hardware? O DOS possui funções específicas para tal, funções 25h e 35h, ver Tabela 10.3 da Seção 10.2 . Neste exemplo, preferimos alterar manualmente o apontador da Tabela do vetor de interrupção INT 1Ch. (endereço base em 0000:0070), e o código do programa de instalação fica bastante reduzido. O deslocamento da nova rotina de interrupção aponta para o início do programa principal, no caso 0108h e o código de segmento é o código em uso, onde está instalado o programa apontado pelo Debug na fase de montagem,e será realocado posteriormente pelo DOS quando o programa *.COM a ele associado for carregado na memória.

Observe que evitamos o uso de funções e interrupções DOS dentro do programa principal. Isto se deve ao fato de que as interrupções DOS são não reentrantes, ver Seção 11.4. Este programa não é residente. Deste modo, não é possível de carregar outro programa tipo *.COM e ambos funcionarem normalmente em conjunto, ver programa fonte do disco. Uma vez alterado o vetor de interrupção a cada 55 ms a CPU irá procurar no endereço CS:0108 uma rotina para ser executada. Carregar outro programa, sem

antes proteger esta área de memória, significa apontar o vetor de interrupção para um programa sem sentido, o que fatalmente levará o microcomputador a uma pane geral ("crash"). Daí o aviso, no final da listagem, a respeito de não carregar outro programa quando este estiver em funcionamento. Antes é necessário restabelecer a tabela do vetor de interrupção através da reinicialização do sistema ("boot") através de CTRL + ALT + DEL ou ligando e religando o microcomputador.

Na próxima Seção, mostramos como terminar e deixar uma parte do programa residente, evitando que o DOS utilize uma área de memória específica para carregar outro programa.

11.4 Programa Termina e Fica Residente

O DOS é um sistema operacional projetado, a princípio, para comandar a execução de um único programa para um único usuário. A Microsoft adicionou alguns ganchos para manipular programas que terminam e ficam residentes TSR ("Terminate and Stay Resident"). Entretanto, estes recursos nem sempre são bem documentados.

Embora um único programa possa ser executado em um determinado instante, nada impede que outros programas estejam presentes na memória. Como acessar um programa residente ? O modo mais utilizado é através da captura de uma interrupção por hardware, ver Exemplo 11.3, ou através de uma interrupção externa, usualmente pelo teclado.

Os programas residentes na memória nela permanecem quando o sistema retorna ao DOS, Figura 11.4, ficando na área de proteção ao DOS como um anexo, a parte residente de COMMAND.COM.

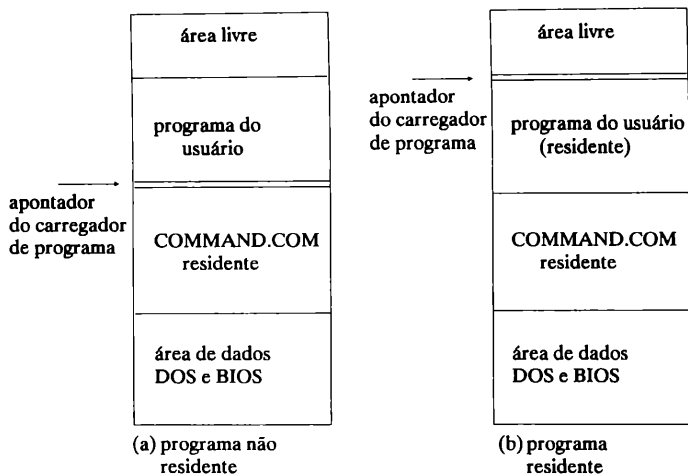


Figura 11.4 Alocação de programa residente

Ao terminar a execução de um programa e retornando ao DOS, através da função DOS 4Ch por exemplo, o bloco de memória alocado inicialmente para o programa do usuário é liberado. Porém caso o programa termine, usando a função DOS 31h, ver Seção 10.2, uma área de memória fica residente. O apontador do carregador do programa do prefixo do segmento de programa PSP ("Program Segment Prefix"). Deste modo o programa fica protegido num nicho de memória e é invisível ao sistema operacional ou para outros programas. Quando solicitado a operar é ativado instantaneamente, produzindo o efeito de brotar rapidamente na tela de vídeo ("pop up"). Por exemplo o programa residente da Borland SIDEKICK (agenda de mesa, custo aproximado US\$ 65,00) é um belo exemplo de programa residente ativado pelo teclado.

Manter a integridade de um programa residente é relativamente simples através da função DOS 31h. A maior dificuldade reside na convivência do programa residente com os outros programas. Para coexistir em harmonia com outros programas em execução e o DOS, o programa residente deve ser capaz de restabelecer o estado original após término da execução de sua tarefa, retornando o registrador de flag e os demais registradores intactos. Porém existem rotinas, como as funções DOS que são não reentrantes, isto é devem ser executadas completamente antes de serem interrompidas.

O DOS usa sua própria pilha ("stack") de modo a não interferir na pilha do usuário. Ao receber a solicitação de uma interrupção no meio da execução de uma função DOS apontador de programa é capaz de executar a rotina de interrupção. Porém não será possível um retorno normal ao programa principal devido ao desarranjo que a interrupção causa na pilha do DOS.

Existem dois modos de contornar o problema das rotinas não reentrantes em programas residentes. O mais simples é evitar o seu uso. Isto é possível em pequenos programas com tarefas bem limitadas, tal como o Exemplo 11.4. Uma outra alternativa é evitar a ativação do programa residente durante a execução de uma rotina DOS. Isto implica num trabalho bem mais complexo de manipulação do vetor de interrupção, em particular da interrupção DOS principal, 21h.

No desenvolvimento de um programa residente existem 2 módulos: instalação e parte residente. A função do módulo de instalação limita-se a interceptar o vetor de interrupção, quando necessário, e terminar o programa com volta ao DOS, mantendo uma bloco de código de programa localizado na memória residente.

O módulo residente pode ter duas partes. Uma para verificar se a ativação do programa residente é possível, de modo a preservar a integridade do sistema. A outra é o núcleo de uma tarefa a ser executada com retorno ao programa principal.

No disco que acompanha este livro, apresentamos 2 programas fontes comentados residentes. Um deles é o relógio em tempo real nas versões de Assembly e C, o que completa o Exemplo 11.3. Este programa só pode ser montado com o auxílio de um montador (no caso usamos o Microsoft Macro Assembler, MASM 5.0), ou de um compilador C (no caso o Turbo C 2.0), ver Seção 6.5. A extensão da listagem deste programa fonte excede o espaço aqui disponível.

O outro programa residente apresentado pode ser considerado um pequeno utilitário. Procuramos, através deste Exemplo, mostrar como desviar uma rotina de interrupção e deixar uma rotina de hardware residente. Todas as teclas estão associadas à interrupção de teclado, INT 09h ou IRQ 01 ver Tabela 5.1, Seção 5.2. Porém uma tecla especial define por si só uma interrupção de hardware, trata-se da tecla Print-Screen, que quando associado com Shift, (Shift-PrintScreen), produz a interrupção de hardware BIOS INT 05h, ver Tabela 2.2, Seção 2.8. O Exemplo 11.4 altera esta interrupção por hardware, de modo que em vez de enviar os caracteres do vídeo para a impressora, passa a executar uma rotina de limpar a tela de vídeo em uso.

Exemplo 11.4 Redefinir a rotina de interrupção Shift- PrintScreen como um programa residente com a função de limpar a tela de vídeo.

CS:0100	JMP 011A	; saltar para INSTALAÇÃO
CS:0102	NOP	;
CS:0103	NOP	;
CS:0104	NOP	; PROGRAMA RESIDENTE
CS:0105	PUSH DS	; rotina principal com início em
CS:0106	PUSH AX	; CS:0105
CS:0107	PUSH BX	; salvando primeiro os registradores
CS:0108	PUSH CX	;
CS:0109	PUSH DX	;
CS:010A	MOV AH,0F	; limpar tela através das funções BIOS
CS:010C	INT 10	; vídeo. Obter modo corrente, função 0F
CS:010E	MOV AH,00	; em AL e redefinindo o modo de vídeo,
CS:0110	INT 10	; função 00, de modo a zerar buffer
CS:0112	POP DX	; da página de vídeo corrente.
CS:0113	POP CX	; recuperando os registradores da pilha
CS:0114	POP BX	; na ordem do último a entrar, primeiro
CS:0115	POP AX	; a sair "LIFO"
CS:0116	POP DS	; antes de terminar rotina de residente
CS:0117	IRET	; e retornando da interrupção
CS:0118	NOP	;
CS:0119	NOP	; PROGRAMA INSTALAÇÃO
CS:011A	MOV DX,0105	; redefinir nova rotina de interrupção
CS:011D	MOV AL,05	; de INT 05h para CS:0105, através da
CS:011F	MOV AH,25	; função DOS 25
CS:0121	INT 21	;
CS:0123	MOV DX,0134	; enviar mensagem de vídeo apontada em
CS:0126	MOV AH,09	; CS:0134, através da função DOS 09
CS:0128	INT 21	;
CS:012A	MOV AX,3100	; função de Terminar e Ficar Residente
CS:012D	MOV DX,0C19	; o bloco definido de CS:0000 a CS:0118
CS:0130	INT 21	; através da função DOS 31
CS:0131	NOP	

D 134 L 50

CS:0134	524F5449-4E41205348494654	ROTINA SHIFT
CS:0140	2D50525343524545-4E203D20434C4541	-PRSCREEN = CLEA
CS:0150	522053435245454E-20494E5354414C41	R SCREEN INSTALA
CS:0160	44412024	DA \$

Exemplo 11.4 Listagem EX_11#4.COM

O módulo de instalação com início em CS:011Ah executa as tarefas:

- (a) desviar o vetor de interrupção INT 05h
- (b) enviar mensagem de instalação
- (c) terminar o programa deixando parte residente

Esta parte do programa é usada uma única vez, por isto pode-se usar as funções DOS sem problemas, no caso para alterar o vetor na Tabela de Interrupção (função DOS 25h) e enviar mensagem para o vídeo (função DOS 09h).

Pode parecer estranho a definição da quantidade de memória, em parágrafos, que deve ficar residente, no caso 0C19, tal como apontado por DX antes de chamar a função DOS 31h. O trecho de programa que deve ficar residente está entre os endereços 0C07:0105 e 0C07:0118 (no caso o DEBUG iniciou a montagem do programa no segmento de código 0C07). A função DOS 31h toma o início do programa a ser protegido a base do segmento corrente, no caso: 0C07:0000. O final deve ser calculado, reescrevendo o endereço definitivo em termos de parágrafos:

0C07:0118	—————→	0C188
endereço		endereço
segmento:deslocamento		efetivo

O parágrafo associado a este endereço é o menor segmento que quando escrito na forma Segmento:0000, aponta para um endereço efetivo maior do que 0C188. No caso é :

0C19:0000	—————→	0C190	>	0C188
endereço		endereço		
segmento:deslocamento		efetivo		

Observe que estamos preservando mais bytes do que o necessário, pois enquanto a área protegida começa em CS:0000, a rotina residente de interesse está 0105 bytes acima, com início em CS:0105. Isto não apresenta inconveniente algum, pois somente uma parte do bloco total residente será utilizado, o restante é um código sem nenhum sentido. Esta é uma limitação do DEBUG, que só monta programas tipo *.COM que se inicia sempre em ORG = 100h, ver Tabela 6.4 na Seção 6.4.

O módulo residente é bastante simples. Ele será ativado cada vez que a interrupção de hardware BIOS INT 05h for solicitada, ou seja quando as teclas SHIFT + PrintScreen estiverem ativadas. Diferente das funções DOS, as funções BIOS são reentrantes e podem ser utilizadas livremente em programas residentes, tal como fizemos neste Exemplo.

Este programa tem funcionado bem em conjunto com vários utilitários, exceto com programas que checam o status da impressora, tal como os editores de texto. Para retornar ao prompt, ou ao estado inicial, é necessário teclar CR ("Carriage Return" ou Enter) pois o prompt do DOS atual fica invisível após a execução deste programa.

NOTAÇÃO HEXADECIMAL E CÓDIGOS ASCII

A.1 Notação hexadecimal

Enquanto o microprocessador só compreende a linguagem eletrônica dos estados lógicos e (Ver Capítulo 6), nós estamos acostumados a raciocinar usando algarismos arábicos em base decimal.

Anotar qualquer instrução ou dado na forma binária (base 2) é um desperdício de tempo e espaço. Por outro lado, a base 10, que nos é familiar, não é uma estrutura natural para os dados utilizados pelo microprocessador. Por isto, adotamos como padrão, a representação hexadecimal, para descrever endereços e dados. Para diferenciar da notação decimal usamos o sufixo "h" (minúsculo) para indicar a representação na base 16, ou hexadecimal.

Uma mesma quantidade pode ser descrita em diferentes bases. Por exemplo:

valor	representação
$15 = 1 \times 10^1 + 5 \times 10^0$	→ 15 base decimal
$15 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	→ 1111 base binária
$15 = 0 \times 16^1 + 15 \times 16^0$	→ 0Fh base hexadecimal

Se por um lado a base binária utiliza somente 2 símbolos (0 e 1), a base hexadecimal necessita adicionar aos 10 algarismos usuais da base decimal (0-9), outros símbolos para representar os valores entre 10 e 15. Os símbolos adotados foram as 6 primeiras letras do alfabeto (A...F), conforme Tabela A.1

decimal	binário	hexadecimal
00 $0x10^1 + 0x10^0$	0000 $0x2^3 + 0x2^2 + 0x2^1 + 0x2^0$	00h $0x16^1 + 0x16^0$
01 $0x10^1 + 1x10^0$	0001 $0x2^3 + 0x2^2 + 0x2^1 + 1x2^0$	01h $0x16^1 + 1x16^0$
02 $0x10^1 + 2x10^0$	0010 $0x2^3 + 0x2^2 + 1x2^1 + 0x2^0$	02h $0x16^1 + 2x16^0$
03 $0x10^1 + 3x10^0$	0011 $0x2^3 + 0x2^2 + 1x2^1 + 1x2^0$	03h $0x16^1 + 3x16^0$
04 $0x10^1 + 4x10^0$	0100 $0x2^3 + 1x2^2 + 0x2^1 + 0x2^0$	04h $0x16^1 + 4x16^0$
05 $0x10^1 + 5x10^0$	0101 $0x2^3 + 1x2^2 + 0x2^1 + 1x2^0$	05h $0x16^1 + 5x16^0$
06 $0x10^1 + 6x10^0$	0110 $0x2^3 + 1x2^2 + 1x2^1 + 0x2^0$	06h $0x16^1 + 6x16^0$
07 $0x10^1 + 7x10^0$	0111 $0x2^3 + 1x2^2 + 1x2^1 + 1x2^0$	07h $0x16^1 + 7x16^0$
08 $0x10^1 + 8x10^0$	1000 $1x2^3 + 0x2^2 + 0x2^1 + 0x2^0$	08h $0x16^1 + 8x16^0$
09 $0x10^1 + 9x10^0$	1001 $1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$	09h $0x16^1 + 9x16^0$
10 $1x10^1 + 0x10^0$	1010 $1x2^3 + 0x2^2 + 1x2^1 + 0x2^0$	0Ah $0x16^1 + Ax16^0$
11 $1x10^1 + 1x10^0$	1011 $1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$	0Bh $0x16^1 + Bx16^0$
12 $1x10^1 + 2x10^0$	1100 $1x2^3 + 1x2^2 + 0x2^1 + 0x2^0$	0Ch $0x16^1 + Cx16^0$
13 $1x10^1 + 3x10^0$	1101 $1x2^3 + 1x2^2 + 0x2^1 + 1x2^0$	0Dh $0x16^1 + Dx16^0$
14 $1x10^1 + 4x10^0$	1110 $1x2^3 + 1x2^2 + 1x2^1 + 0x2^0$	0Eh $0x16^1 + Ex16^0$
15 $1x10^1 + 5x10^0$	1111 $1x2^3 + 1x2^2 + 1x2^1 + 1x2^0$	0Fh $0x16^1 + Fx16^0$
16 $1x10^1 + 6x10^0$	1 0000 $1x2^4 + 0x(2^3 + 2^2 + 2^1 + 2^0)$	10h $1x16^1 + 0x16^0$

Tabela A.1 Representação nas diferentes bases

Como fazer a conversão de uma base para outra? Embora alguns livros mostrem longas tabelas de conversão, a maneira mais fácil é usar o utilitário Sidekick (Borland) para fazer esta conversão diretamente entre hexadecimal e decimal.

Enquanto a conversão de decimal para hexadecimal é feita através da operação de multiplicação, ver Tabela A.1, a operação inversa envolve um conjunto de divisões. Por exemplo, qual é a representação em hexadecimal do valor:

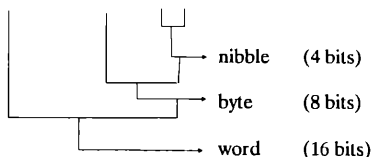
$$\begin{aligned}
 1000 &= ?x16^2 + ?x16^1 + ?x16^0 \\
 1000/16 &= 62 + \text{resto } 8 \\
 62/16 &= 3 + \text{resto } 14 \\
 1000 &= 3x16^2 + 14x16^1 + 8x16^0 \longrightarrow 03E8h
 \end{aligned}$$

Operações aritméticas com números hexadecimais são mais complexas. A solução mais fácil consiste em usar o Debug (Ver Seção 7.3 e Apêndice B) e para isto basta um PC com o sistema operacional completo.

Um grupo de bit possui uma denominação especial quando dividido em conjunto de 4, 8 ou 16 bits. Por exemplo: A18Eh pode ser descrito como:

$$A \quad 1 \quad 8 \quad E \quad h = 10 \times 16^3 + 1 \times 16^2 + 8 \times 16^1 + 14 \times 16^0 = 41358$$

1010 0001 1000 1110



A.2 O Código ASCII

O usuário e o microcomputador falam uma linguagem diferente. Enquanto utilizamos símbolos, textos, gráficos, sons e imagens para nos comunicar, o microprocessador utiliza-se exclusivamente de estados lógicos na representação binária. Portanto, é necessário um código para estabelecer um canal de comunicação entre as atividades humanas e o microcomputador. Esta necessidade técnica acabou criando uma linguagem universal. Embora os computadores estejam espalhados em todas as partes do mundo, eles são capazes de se comunicar eletronicamente e transmitir imagens e textos de um terminal a outro. Isto só é possível porque existe um código padrão. A princípio não existia um consenso e cada fabricante de computador procurava definir seu próprio código de comunicação. O Instituto Americano de Normas (ANSI) estabeleceu um código denominado ASCII "American Standard Code for Information Interchange" que se tornou um padrão aos microcomputadores pessoais.

Embora exista um núcleo padrão, pequenas alterações ocorrem de um país para outro. O fato do microcomputador editar textos nas mais diversas línguas, obriga o uso de símbolos especiais para receber do teclado, enviar para o vídeo ou impressora, um caractere diferente do utilizado na língua inglesa, tal como é a acentuação. Isto obriga o usuário a configurar o editor de textos ou a impressora quando for editar textos em línguas diferentes do inglês.

A um mesmo valor do código ASCII podem estar associados símbolos diferentes, dependendo da função a ser executada. Um mesmo código pode produzir resultados diferentes quando for enviado para uma impressora ou para o vídeo.

O código ASCII original tem 128 caracteres. Os primeiros 32 (20h) caracteres não têm uma representação gráfica e são usados para controle nos protocolos de comunicação. No projeto do PC, a IBM estendeu o código original de modo a:

- criar símbolos gráficos para os 32 primeiros caracteres
- estender para 256 o número de caracteres possíveis
- criar um conjunto de caracteres especiais para aceitar as diversas associações do teclado.

Os caracteres de controle, Tabela A.2, são usados pelo PC para se comunicar com os periféricos, tais como: impressoras, plotters, modem etc..

código	mnemônico nome	comentário
0	NUL Null	preencher um caractere em branco
1	SOH Start of Header	início do cabeçalho de mensagem
2	STX Start of TeXt	início de um bloco de texto
3	ETX End of TeXt	fim de um bloco de text
4	EOT End Of Transmission	Final de uma sessão de transmissão
5	ENQ iNquiry	solicita uma identificação
6	ACK ACKnowledgment	protocolo de comunicação assíncro- na. Resposta "Sim" do dispositivo indica "pronto para transmitir"
7	BEL BELl	aciona alarme do terminal
8	BS Back Space	mover cursor para trás
9	HT Horizontal Tab	mover cursor até próxima tabulação
10	LF Line Feed	mover cursor para próxima linha
11	VT Vertical Tab	mover cursor para baixo até a próxi- ma linha de tabulação
12	FF Form Feed	mover cursor para próxima página
13	CR Carriage Return	mover cursor para margem esquerda
14	SO Shift Out	avisa que os próximos caracteres
15	SI Shift In	próximos caracteres revertidos padrão ASCII

Tabela A.2 Caracteres ASCII de controle

código	mnemônico nome	comentário
16	DLE Data Link Escape	controla transmissão com tecla ESC
17	DC1 Device Control 1	a ser definido pelo usuário
18	DC2 Device Control 2	a ser definido pelo usuário
19	DC3 Device Control 3	a ser definido pelo usuário
20	DC4 Device Control 4	a ser definido pelo usuário
21	NAK No ACKnowledgment	protocolo para transmissão assíncrona. Resposta "Não" do dispositivo indica erro na transmissão.
22	SYN SYNchronous idle	indica que comunicação assíncrona é possível.
23	ETB End Trans. Block	marca limite final de um bloco de transmissão
24	CAN CANCEL	cancelar transmissão solicitada
25	EM End of Medium	final do meio físico, do papel.
26	SUB SUBstitute	usado para modificar um caractere
27	ESC ESCape	iniciar uma sequência de controle com ESC
28	FS File Separator	início de transmissão de arquivo
29	GS Group Separator	início de transmissão de registro
30	RS Record Separator	início de transmissão de um campo
31	US Unit Separator	início de transmissão de uma unidade

Tabela A.2 Caracteres ASCII de controle (continuação)

Como visualizar os caracteres ASCII? A partir do DOS é possível obter a imagem gráfica de um caractere, acionando a tecla ALT mais o código do caractere. Na ROM-BIOS, existe uma biblioteca dos caracteres ASCII de vídeo, ver Exemplo 9.1. Na Tabela A.3, eles estão representados na notação hexadecimal e decimal na seguinte convenção: Considerar o valor da coluna (entre 0 e Fh) como o nibble mais significativo e uma determinada linha (entre 0 e Fh) como o nibble menos significativo para compor o byte. Exemplo o caractere A possui o código 41h (coluna 4h e linha 1h) ou em decimal 65 (coluna 65 + linha 1).

Conjunto de Caracteres (00-7F) Referência Rápida

VALOR DECIMAL		0	16	32	48	64	80	96	112
	VALOR HEXA DECIMAL	0	1	2	3	4	5	6	7
0	0	BRANCO (NULO)	▶	BRANCO (NULO)	0	@	P	'	p
1	1	☺	◀	!	1	A	Q	a	q
2	2	☹	↕	"	2	B	R	b	r
3	3	♥	!!	#	3	C	S	c	s
4	4	♦	¶	\$	4	D	T	d	t
5	5	♣	§	%	5	E	U	e	u
6	6	♠	■	&	6	F	V	f	v
7	7	•	↓	'	7	G	W	g	w
8	8	•	↑	(	8	H	X	h	x
9	9	○	↓	)	9	I	Y	i	y
10	A	○	→	*	:	J	Z	j	z
11	B	♂	←	+	;	K	I	k	{
12	C	♀	└	,	<	L	\	l	!
13	D	♪	↔	—	=	M	J	m	}
14	E	♪	▲	.	>	N	^	n	~
15	F	☼	▼	/	?	O	_	o	△

Tabela A.3 Caracteres ASCII-IBM de vídeo

Conjunto de Caracteres (88-FF)

VALOR DECIMAL	▼	128	144	160	176	192	208	224	240
▼	VALOR HEXA DECIMAL	8	9	A	B	C	D	E	F
0	0	Ç	É	á	▤	▥	▦	∞	≡
1	1	ü	æ	í	▧	▨	▩	β	±
2	2	é	Æ	ó	▪	▫	▬	Γ	≥
3	3	â	ô	ú	▭	▮	▯	π	≤
4	4	ä	ö	ñ	▰	▱	▲	Σ	∫
5	5	à	ò	Ñ	△	▴	▵	σ	∫
6	6	å	û	ä	▴	▵	▶	μ	÷
7	7	ç	ù	ø	▶	▷	▸	τ	≈
8	8	ê	ÿ	ï	▷	▸	▹	ø	°
9	9	ë	Ö	Γ	▸	▹	►	θ	•
10	A	è	Ü	Γ	▹	►	▻	Ω	•
11	B	ï	¢	½	▻	▼	▽	δ	√
12	C	î	£	¼	▼	▽	▾	∞	n
13	D	ì	¥	ì	▽	▾	▿	φ	²
14	E	Ä	℞	«	▾	▿	▹	∈	■
15	F	Å	ƒ	»	▿	▹	►	∩	BRANCO "

Tabela A.3 Caracteres ASCII-IBM de vídeo (continuação)

O projeto do PC usou um modo engenhoso para decodificar o grande conjunto de combinações possíveis com o teclado. Como vemos nas Tabelas A.2 e A.3, o código 0 não tem uma função definida. Por isto foi usado pelos projetistas como uma chave para incorporar outros 256 caracteres. A entrada de um caractere nulo, via teclado, indica ao programa tratar-se de uma tecla especial cujo identificação só se completa com a leitura do próximo byte. Nem todas as 256 possibilidades são utilizadas, conforme Tabela A.4

código	tecla acionada	código	tecla acionada
003	caractere nulo	085	Shift + F2
015	Shift Tab	086	Shift + F3
016	Alt + Q	087	Shift + F4
017	Alt + W	088	Shift + F5
018	Alt + E	089	Shift + F6
019	Alt + R	090	Shift + F7
020	Alt + T	091	Shift + F8
021	Alt + Y	092	Shift + F9
022	Alt + U	093	Shift + F10
023	Alt + I	094	Control + F1
024	Alt + O	095	Control + F2
025	Alt + P	096	Control + F3
030	Alt + A	097	Control + F4
031	Alt + S	098	Control + F5
032	Alt + D	099	Control + F6
033	Alt + F	100	Control + F7
034	Alt + G	101	Control + F8
035	Alt + H	102	Control + F9
036	Alt + J	103	Control + F10
037	Alt + K	104	Alt + F1
038	Alt + L	105	Alt + F2
044	Alt + Z	106	Alt + F3
045	Alt + X	107	Alt + F4
046	Alt + C	108	Alt + F5
047	Alt + V	109	Alt + F6
048	Alt + B	110	Alt + F7
049	Alt + N	111	Alt + F8

Tabela A.4 Código expandido do teclado

código	tecla acionada	código	tecla acionada
050	Alt + M	112	Alt + F9
059	F1	113	Alt + F10
060	F2	114	Control + PrtSc
061	F3	115	Control +
062	F4	116	Control + --
063	F5	117	Control + End
064	F6	118	Control + PgDn
065	F7	119	Control + Home
066	F8	120	Alt + 1
067	F9	121	Alt + 2
068	F10	122	Alt + 3
071	Home	123	Alt + 4
072	seta para cima	124	Alt + 5
073	Page up	125	Alt + 6
075	←	126	Alt + 7
077	→	127	Alt + 8
079	End	128	Alt + 9
080	seta para baixo	129	Alt + 0
081	Page Down	130	Alt + -
082	Insert	131	Alt + =
083	Delete	132	Control + PgUp
084	Shift + F1		

Tabela A.4 Código expandido do teclado (continuação)

MANUAL DO DEBUG

O Debug é um poderoso utilitário que acompanha o sistema operacional DOS desde sua primeira versão. É o modo mais rápido e simples para montar pequenas rotinas em Assembly. Não deve ser confundido com um "Assembler", um montador profissional, pois ele não é capaz de manipular rótulos ("labels") nem funções macro. A Seção 7.1 descreve as principais funções do Debug. Neste Apêndice elas são descritas em ordem alfabética.

Para indicar os parâmetros de cada comando usamos a seguinte convenção:

<.. > aceita um parâmetro opcionalmente

(..) solicita um parâmetro obrigatoriamente.

A < Endereço >	<i>Assembler</i>
Monta instruções na memória. Cada linha de instrução só termina após teclar ENTER. A sintaxe é verificada pela ROM-BIOS logo após o final de uma linha. A linha começa no endereço apontado. Caso não existir na última linha montada ou no valor apontado por CS:IP	

C (endereço1 Lcomprimento endereço2)	<i>Compare</i>
Comparar o número de bytes definido pelo comprimento de 2 blocos de memória apontados pelos endereços 1 e 2. Exemplo: -C 200 L 30 300 mostrar os 48 bytes (30h) armazenados a partir do endereço CS:0200 (endereço1) com os armazenados a partir do endereço CS:0300	

D < endereço > < Lcomprimento >	Dump ou Display
Mostra um bloco de memória a partir do endereço apontado (caso não existir a partir de CS:IP) conforme comprimento indicado (na ausência deste, 64 bytes). No canto direito do vídeo, são mostrados os caracteres em código ASCII, ver Exemplo 7.2 na Seção 7.4 Exemplo: -D 100 L 20 mostra o conteúdo de memória de 32 bytes a partir de CS:0100	

E (endereço) < lista de bytes > < "cadeia de caracteres" >	Enter
Preenche uma região de memória com os bytes especificados. Diretamente, através de uma lista de bytes em código ASCII, ou uma cadeia entre aspas. Ver Exemplo 7.1 Seção 7.3 Exemplo: -E DS:100 entrada de bytes a partir de DS:0100. Espaço em branco separa bytes e tecla enter indica fim do processo -E DS:00 20 20 20 Preenche com 3 espaços em branco (código ASCII 20h) a partir de DS:0000	

F (endereço Lcomprimento "cadeia de caracteres)	Fill
Preenche uma região de memória a partir do endereço apontado de comprimento definido com a cadeia de caracteres indicada. Ver Exemplo 7.6 na Seção 7.5 Exemplo: -F DS:0200 L 05 "TESTE" Preenche os 5 bytes da memória apontado por DS:0200 com a cadeia de caracteres "TESTE"	

G < endereço > < endereço > de final	Go
Executa uma instrução apontada por CS:IP se nenhum endereço for indicado. Existindo um endereço final, o comando deixa de incrementar IP a partir deste ponto, do contrário continua a incrementar indefinidamente. Ver Exemplo 7.7 na Seção 7.6. Exemplo: -G 100 120 Inicia a execução de uma instrução em CS:0100 e termina em CS:0120.	

H (valor1 valor2)	<i>Hexadecimal</i>
Executa operação de adição (valor1 + valor2) e subtração (valor1-valor2) na base hexadecimal. Ver Exemplo 7.1, Seção 7.3. Exemplo: -H 100 200 Adicionar 100h + 200h e subtrair 100h-200h	

I (endereço da porta)	<i>Input</i>
Entrada de um byte da porta de I/O especificada Exemplo: -I 03F8	

L < endereço > < drive setor1 setor 2 >	<i>Load</i>
Carregar na memória arquivo de nome encontrado em CS:0080, no endereço apontado (ou em CS:0100) caso nenhum endereço foi indicado. Caso algum drive for especificado (em valores numéricos, A = 0, B = 1, C = 2) o bloco entre os setores é carregado no endereço indicado. Exemplo: -L ES:0100 1 3 5 Carregar em ES:0100 os setores 3,4 e 5 do drive B (1)	

M (endereço1 Lcomprimento endereço2)	<i>Move</i>
Movimentar um bloco de memória de comprimento L do endereço inicial endereço1 para o endereço2. Exemplo: -M 100 L 20 200 Movimentar 32 bytes (20h) com início em CS:0100 para CS:0200	

N (nome do arquivo)	<i>Name</i>
Preencher o endereço CS:0081h com o nome de um arquivo para gravação em disco. Arquivos tipo *.EXE e *.Hex não são aceitos. Ver Exemplo 7.8 na Seção 7.7 Exemplo: -N TESTE.COM Preparar para arquivar o programa TESTE.COM	

O (endereço da porta de saída)(byte)	Output
Envia um byte para a porta de I/O especificada Exemplo: -O 03F8 00 Enviar o byte 00 para o endereço de I/O 03F8	

Q	Quit
Abandonar o ambiente do Debug e retornar ao DOS	

R < nome do registro >	Register
R sem o nome do registrador mostra o conteúdo de todos os registradores., ver Exemplo 7.1 Seção 7.3. Indicando um registrador além de mostrar o seu conteúdo, o Debug aguarda um byte para modificá-lo. Neste caso, teclar enter simplesmente significa deixar o registrador inalterado.	

S (endereço Lcomprimento "cadeia")	Search
Procurar uma "cadeia" de caracteres no bloco de memória de comprimento L com início no endereço apontado. Exemplo: -S 200 L 20 "teste" Procurar nos 32 (20h) primeiros bytes a partir de CS:0200 um bloco com os caracteres "teste"	

T (= endereço)(passos)	Trace
Executa uma instrução (ou um grupo caso o número de passos for especificado) a partir do endereço apontado. Caso nenhum endereço for indicado o endereço assumido será o conteúdo de CS:IP. Ver Exemplo 7.4 na Seção 7.3. Exemplo: -T = CS:0100 20 Executar as 32 (20h) primeiras instruções a partir do endereço CS:0100	

U < endereço inicial > < endereço final >	<i>Unassemble</i>
Decodificar um código hexadecimal em instruções ASSEMBLY (desassemblar), existentes entre os endereços apontados. Na indefinição de um bloco de memória, o Debug começa a desassemblar a partir de CS:IP. Ver Exemplo 7.7 na Seção 7.6 Exemplo: -U 100 120 Desassemblar as instruções existentes entre CS:0100 e CS:0120	

W < endereço > < drive setor1 setor2 >	<i>Write</i>
Gravar um arquivo com o nome armazenado em CS:0080 (ver comando -N). O número de bytes a ser gravado depende do conteúdo de BX:CX. Caso nenhum endereço for especificado CS:IP é assumido como o valor inicial. A exemplo de Load um conjunto de setores pode ser gravado num drive específico. Ver Exemplo 7.8 na Seção 7.7. Exemplo: -W 120 1 0 2 Gravar os dados da memória a partir de CS:0120, no Drive B (1) nos setores 0,1 e 2	

CÓDIGO DE INSTRUÇÕES 8088/8086

No Capítulo 8, apresentamos as principais funções do código ASSEMBLY 8086/8088. Neste Apêndice, eles serão apresentados em ordem alfabética. Com os comandos -A e -U do Debug é muito simples montar o código referente a uma instrução em ASSEMBLY e vice versa. Tendo em vista esta facilidade, optamos por não apresentar o Código Operacional de cada instrução ("OpCode"). Para descrever os diversos formatos dos operandos usamos uma tabela de resumo, ver a Tabela 8.1 na Seção 8.1. O tipo dos operandos usados de modo abreviado estão na Tabela C.1. Observe que preferimos manter o termo original em inglês, por considerar o modo mais fácil de familiarizar o leitor com as expressões utilizadas nas revistas especializadas.

menmônimo	comentário
(no operands)	instrução não necessita de operandos
register	um registrador geral de 8 ou 16 bits
reg 16	registrador geral de 16 bits
seg-reg	registrador de segmento
accumulator	registradores AX,AL ou AH
immediate	uma constante na faixa de 0-FFFFh
immed8	uma constante na faixa de 0-FFh
memory	uma região de memória de 8 ou 16 bits
mem8	uma região de memória de 8 bits
mem16	uma região de memória de 16 bits
source-table	nome de tabela translação de 256 bytes
source-string	cadeia endereçada pelo registrador SI
dest-string	cadeia endereçada pelo registrador DI
short-label	um rótulo a uma distância inferior de -128 a + 127 bytes do final da instrução
near-label	um rótulo no código de segmento em uso

Tabela C.1 Caracterização dos operandos

mnemônico	comentário
far-label	um rótulo em outro código de segmento
near-proc	uma procedure (rotina) no código de segmento em uso
far-proc	uma procedure (rotina) em outro código de segmento
memptr16	uma word (16 bits) contendo o deslocamento (offset) da localização no código de segmento corrente, para onde o controle será transferido.
memptr32	uma dupla word (32 bits) contendo o deslocamento e o endereço da base do segmento da localização em outro código de segmento, para onde o controle será transferido.
regptr16	um registrador geral de 16 bits contendo o deslocamento (offset) da localização, no código de segmento em uso, para o qual o controle será transferido

Tabela C.1 Caracterização dos operandos (continuação)

Uma instrução pode alterar o registrador de flag, deixando alguns bits alterados ou simplesmente tomando um valor qualquer após a execução de uma instrução, ver Seção 3.1. Para anotar de modo resumido a influência de uma instrução sobre os flags, adotamos a seguinte conversão:

mnemônico	comentário
(vazio)	nenhuma alteração
0	resetado a zero
1	set a um
?	0 ou 1, dependendo do resultado
*	valor indefinido e não confiável
>	restabelece o valor original salvo

Tabela C.2 Caracterização das alterações nos flags

A Tabela a seguir, apresenta o conjunto de instruções ASSEMBLY 8086/8088, indicando as alterações produzidas no registrador de flag, ver Seção 3.1, e que tipo de operando elas solicitam.

AAA	(no operands) ASCII adjust for addition	Flags	O	D	I	T	S	Z	A	P	C
			*				*	*	?	*	?
	OPERANDO (no operands)	exemplo: AAA									

AAD (no operands) ASCII adjust for division	Flags O D I T S Z A P C * ? ? * ? *
OPERANDO (no operands)	exemplo: AAD

AAM (no operands) ASCII adjust for multiply	Flags O D I T S Z A P C * ? ? * ? *
OPERANDO (no operands)	exemplo: AAM

AAS (no operands) ASCII adjust for subtraction	Flags O D I T S Z A P C * * * ? * ?
OPERANDO (no operands)	exemplo: AAS

ADC	destination, source Add with carry	Flags O D I T S Z A P C ? ? ? ? ? ? ? ?
	OPERANDO register,register register,memory memory,register register,immediate memory,immediate accumulator,immediate	exemplo ADC AX,BX ADC DX, valor[SI] ADC soma[BX][SI],BX ADC CX,20 ADC valor SI],20 ADC AL,0F

ADD	destination, source Addition	Flags O D I T S Z A P C ? ? ? ? ? ? ? ?
	OPERANDO (ver ADC)	exemplo (trocar ADC por ADD)

AND	destination, source Logical and	Flags O D I T S Z A P C 0 ? ? * ? 0
	OPERANDO (ver ADC)	exemplo (trocar ADC por AND)

CALL	target Call a procedure	Flags O D I T S Z A P C
	OPERANDO near-proc far-proc memptr16 regptr16 memptr32	exemplo CALL próxima CALL longe CALL tabela [SI] CALL DX CALL [CX],double[SI]

CBW (no operands) Convert byte to word	Flags O D I T S Z A P C
OPERANDO (no operands)	exemplo CBW

CLC (no operands) Clear carry flag	Flags O D I T S Z A P C 0
OPERANDO (no operands)	exemplo CLC

CLD (no operands) Clear direction flag	Flags O D I T S Z A P C 0
OPERANDO (no operands)	exemplo CLD

CLI (no operands) Clear interrupt flag	Flags O D I T S Z A P C 0
OPERANDO (no operands)	exemplo CLI

CMC (no operands) Complement carry flag	Flags O D I T S Z A P C ?
OPERANDO (no operands)	exemplo CMC

CMP destination,source Compare destination to source ?	Flags O D I T S Z A P C ? ? ? ? ? ?
OPERANDO (ver ADC)	exemplo (trocar ADC por CMP)

CMPS	dest-string,source-string Compare String	Flag	O	I	D	T	S	Z	A	P	C
			?				?	?	?	?	?
	OPERANDO dest-string,source-string	exemplo CMPS tabela1,tabela2									

CWD	(no operands) Convert word to doubleword	Flags	O	D	I	T	S	Z	A	P	C
	OPERANDO (no operands)	exemplo CWD									

DAA	(no operands) Decimal adjust for addition	Flag	O	I	D	T	S	Z	A	P	C
			?				?	?	?	?	?
	OPERANDO (no operands)	exemplo DAA									

DAS	(no operands) Decimal adjust for subtraction	Flags	O	D	I	T	S	Z	A	P	C
			*				?	?	?	?	
	OPERANDO (no operands)	exemplo DAS									

DEC	destination Decrement by 1	Flags	O	D	I	T	S	Z	A	P	C
			?				?	?	?	?	?
	OPERANDO reg16 reg8 memory	exemplo DEC CX DEC CL DEC tabela[SI]									

DIV	source Division, unsigned	Flags O D I T S Z A P C * * * * * *
	OPERANDO reg16 reg8 mem8 mem16	exemplo DIV CX DIV CL DIV memória DIV tabela

ESC	external-opcode, source Escape	Flags O D I T S Z A P C
	OPERANDO immediate, memory immediate, register	exemplo ESC A < tabela [SI] ESC 10, BX

HLT	(no operands) Halt	Flags O D I T S Z A P C
	OPERANDO (no operands)	exemplo HLT

IDIV	source Integer division	Flags O D I T S Z A P C * * * * * *
	OPERANDO (ver DIV)	exemplo (trocar DIV por IDIV)

IMUL	source Integer multiplication	Flags O D I T S Z A P C ? * * * * ?
	OPERANDO (ver DIV)	exemplo (trocar DIV por IMUL)

IN	accumulator,port Input byte or word	Flags O D I T S Z A P C
	OPERANDO accumulator,immed8 accumulator,DX	exemplo IN AL,03F8h IN AX,DX

INC	destination Increment by 1	Flags O D I T S Z A P C ? ? ? ?
	OPERANDO (ver DEC)	exemplo (trocar DEC por INC)

INT	interrupt-type Interrupt	Flags O D I T S Z A P C 0 0
	OPERANDO immed8	exemplo INT 21

INTO	(no operands) Interrupt if overflow	Flags O D I T S Z A P C 0 0
	OPERANDO (no operands)	exemplo INTO

IRET	(no operands) Interrupt Return	Flags O D I T S Z A P C > > > > > > > >
	OPERANDO (no operands)	exemplo IRET

JA/JNBE	short-label jump if above jump if not below nor equal	Flags O D I T S Z A P C
	OPERANDO short-label	exemplo JA superior

JAE/JNB short-label jump if above or equal jump if not below	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JAE sup_igual

JB/JNAE short-label jump if below jump if not above nor equal	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JAE inferior

JBE/JNA short-label jump if below or equal jump if not above	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JAE inf_igual

JC short-label jump if carry	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JC Se_carry

JCXZ short-label jump if CX = 0	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JCXZ cont_fim

JE/JZ short-label jump if equal jump if zero	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JZ igual

JG/JNLE short-label jump if greater jump if not less nor equal	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JG maior

JGE/JNL short-label jump if greater or equal jump if not less	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JGE maior_igual

JL/JNGE short-label jump if less jump if not great nor equal	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JL menor

JLE/JNG short-label jump if less or equal jump if not greater	Flags O D I T S Z A P C
OPERANDO short-label	exemplo JNG menor_igual

JMP	target Jump	Flags O D I T S Z A P C
	OPERANDO short-label near-label far-label memptr16 regptr16 memptr32	exemplo JMP próximo JMP in_segmento JMP out_segmento JMP BX].alo JMP CX JMP out_segmento SI]

JNC	short-label jump if not carry	Flags O D I T S Z A P C
	OPERANDO short-label	exemplo JNC no_carry

JNE/JNZ	short-label jump if not equal jump if not zero	Flags O D I T S Z A P C
	OPERANDO short-label	exemplo JNE diferente

JNO	short-label jump if not overflow	Flags O D I T S Z A P C
	OPERANDO short-label	exemplo JNO no_estouro

JNP/JPO	short-label jump if not parity jump if parity odd	Flags O D I T S Z A P C
	OPERANDO short-label	exemplo JNP ímpar

JNS	short-label jump if not sign	Flags O D I T S Z A P C
	OPERANDO short-label	exemplo JNS positivo

JO	short-label jump if overflow	Flags O D I T S Z A P C
	OPERANDO short-label	exemplo JO estouro

JP/JPE	short-label jump if parity jump if parity even	Flags O D I T S Z A P C
	OPERANDO short-label	exemplo JP par

JS	short-label jump if sign	Flags O D I T S Z A P C
	OPERANDO short-label	exemplo JS negativo

LAHF	(no operands) Load AH from flags	Flags O D I T S Z A P C
	OPERANDO (no operands)	exemplo AHF

LDS	destination,source Load pointer using DS	Flags O D I T S Z A P C
	OPERANDO reg16,mem32	exemplo LDS ES,dados DIJ

LEA	destination,source Load effective address	Flags O D I T S Z A P C
OPERANDO	reg16,mem16	exemplo LEA DX, <BP> <SI>

LES	destination,source Load pointer using ES	Flags O D I T S Z A P C
OPERANDO	reg16,mem32	exemplo LES DX,[BX].buffer

LOCK	(no operands) Lock bus	Flags O D I T S Z A P C
OPERANDO	(no operands)	exemplo LOCK

LODS	source-string Load string	Flags O D I T S Z A P C
OPERANDO	source-string	exemplo LODS tabela

LOOP	short-label Loop	Flags O D I T S Z A P C
OPERANDO	short-label	exemplo LOOP continuar

LOOPE/LOOPZ	short-label Loop if equal/loop if zero	Flags O D I T S Z A P C
OPERANDO	short-label	exemplo LOOPE continuar

LOOPNE/LOOPNZ short-label Loop if not equal Loop if not zero	Flags O D I T S Z A P C
OPERANDO short-label	exemplo LOOPNE continuar

MOV destination,source Move	Flags O D I T S Z A P C
OPERANDO memory,accumulator accumulator,memory register,register register,memory memory,register register,immediate memory,immediate seg-reg,reg16 seg-reg,mem16 reg16,seg-reg memory,seg-reg	exemplo MOV memória,AL MOV AH,memória MOV AX,DX MOV BX,memória MOV memória,BX MOV CX,00FF MOV memória,0Fh MOV ES,DX MOV ES,base MOV BP,ES MOV [BX].segmento,CS

MOVS dest-string,source string Move string	Flags O D I T S Z A P C
OPERANDO dest-string,source-string	exemplo MOVS tabela1,tabela2

MOVSB/MOVS (no operands) Move string(byte/word)	Flags O D I T S Z A P C
OPERANDO (no operands)	exemplo MOVSB

MUL	source multiplication,unsigned	Flags	O	D	I	T	S	Z	A	P	C
			?				*	*	*	*	?
	OPERANDO (ver DIV)	exemplo (trocar DIV por MUL)									

NEG	destination negate	Flags	O	D	I	T	S	Z	A	P	C
			?				?	?	?	?	1
	OPERANDO register memory	exemplo NEG AX NEG memória									

NOP	(no operand) No operation	Flags	O	D	I	T	S	Z	A	P	C
	OPERANDO (no operands)	exemplo NOP									

NOT	destination Logical Not	Flags	O	D	I	T	S	Z	A	P	C
	OPERANDO (ver NEG)	exemplo (trocar NEG por NOT)									

OR	destination,source Logical OR	Flags	O	D	I	T	S	Z	A	P	C
			0				?	?	*	?	0
	OPERANDO (ver ADC)	exemplo (trocar ADC por OR)									

OUT	port,accumulator output byte or word	Flags	O	D	I	T	S	Z	A	P	C
	OPERANDO immed8,accumulator DX,accumulator	exemplo OUT 03F8,AL OUT DX,AL									

POP	destination Pop word off stack	Flags O D I T S Z A P C
	OPERANDO register seg-reg (no CS) memory	exemplo POP CX POP ES POP valor

POPF	(no operands) Pop flags off stack	Flags O D I T S Z A P C
	OPERANDO (no operands)	exemplo POPF

PUSH	source Push word onto stack	Flags O D I T S Z A P C
	OPERANDO register seg-reg memory	exemplo PUSH BX PUSH CS PUSH memória

PUSHF	(no operands) Push flags onto stack	Flags O D I T S Z A P C
	OPERANDO (no operands)	exemplo PUSHF

RCL	destination,count Rotate left through carry	Flags O D I T S Z A P C ? ?
	OPERANDO register,1 register,CL memory,1 memory,CL	exemplo RCL BX,1 RCL AX,CL RCL memória,1 RCL memória,CL

RCR	destination,count Rotate right through carry	Flags O D I T S Z A P C ? ?
	OPERANDO (ver RCL)	exemplo (trocar RCL por RCR)

REP	(no operands) Repeat string operation	Flags O D I T S Z A P C
	OPERANDO (no operands)	exemplo REP MOVSB

REPE/REPZ	(no operands) Repeat string operation while equal	Flags O D I T S Z A P C
	OPERANDO (no operands)	exemplo REP MOVS destino,fonte

REPNE/REPNZ	(no operands) Repeat string operation while not equal	Flags O D I T S Z A P C
	OPERANDO (no operands)	exemplo REPNE MOVS destino,fonte

RET	optional-pop-value Return from procedure	Flags O D I T S Z A P C
	OPERANDO	exemplo RET

ROL	destination,count Rotate left	Flags O D I T S Z A P C ? ?
	OPERANDO (ver RCL)	exemplo (trocar RCL por ROL)

ROR	destination,count Rotate right	Flags O D I T S Z A P C ?
	OPERANDO (ver RCL)	exemplo (trocar RCL por ROR)

SAHF	(no operands) Store AH into flags	Flags O D I T S Z A P C
	OPERANDO (no operands)	exemplo SAHF

SAL/SHL	destination,count Shift arithmetic left Shift logical left	Flags O D I T S Z A P C ?
	OPERANDO (ver RCL)	exemplo (trocar RCL por SAL)

SAR	destination,count Shift arithmetic left Shift logical left	Flags O D I T S Z A P C ?
	OPERANDO (ver RCL)	exemplo (trocar RCL por SAR)

SBB	destination,source Subtract with borrow	Flags O D I T S Z A P C ? ? ? ? ?
	OPERANDO (ver ADC)	exemplo (substituir ADC por SBB)

SCAS	dest-string Scan string	Flags O D I T S Z A P C ? ? ? ? ?
	OPERANDO dest-string	exemplo SCAS tabela

SHR	destination,count Shift logical right	Flags O D I T S Z A P C ? ? ? ? ? ? ?
	OPERANDO (ver RCL)	exemplo (trocar RCL por SHR)

STC	(no operands) Set carry flag	Flags O D I T S Z A P C 1
	OPERANDO (no operands)	exemplo STC

STD	(no operands) Set direction flag	Flags O D I T S Z A P C 1
	OPERANDO (no operands)	exemplo STD

STI	(no operands) Set interrupt enable flag	Flags O D I T S Z A P C 1
	OPERANDO (no operands)	exemplo STI

STOS	dest-string Store byte or word string	Flags O D I T S Z A P C
	OPERANDO dest-string	exemplo STOS tabela

SUB	destination,source Subtraction	Flags O D I T S Z A P C ? ? ? ? ? ? ?
	OPERANDO (ver ADC)	exemplo (trocar ADC por SUB)

TEST	destination,source Test Non destructive logical AND	Flags	O	D	I	T	S	Z	A	P	C
			0					?	?	*	?
	OPERANDO (ver ADC,exceto memory, register)	exemplo (trocar ADC por SUB)									

WAIT	(no operands) Wait while not-TEST (hardware)	Flags	O	D	I	T	S	Z	A	P	C
	OPERANDO (no operands)	exemplo WAIT									

XCHG	destination,source Exchange	Flags	O	D	I	T	S	Z	A	P	C
	OPERANDO accumulator,reg16 memory,register register,register	exemplo XCHG AX,CX XCHG memória, BX XCHG CL,DL									

XLAT	source-table Translate	Flags	O	D	I	T	S	Z	A	P	C
	OPERANDO source-table	exemplo XLAT tabela									

XOR	destination,source logical exclusive OR	Flags	O	D	I	T	S	Z	A	P	C
			0					?	?	*	?
	OPERANDO (ver ADC)	exemplo (trocar ADC por XOR)									

INTERRUPÇÕES BIOS

Neste Apêndice, detalhamos os Interrupções BIOS mostradas no Capítulo 9. Enquanto naquele Capítulo descrevemos, resumidamente, as Interrupções quanto à função, aqui elas são descritas detalhadamente, em ordem numérica. Adotamos como usual o padrão de vídeo CGA para a versão PC-XT. Quando outro modelo for descrito estará indicado explicitamente conforme a seguinte abreviação:

<MDA> Monochrome Display Adapter (adaptador monocromático)

<CGA> Color/Graphics Adapter (adaptador gráfico colorido)

<EGA> Enhanced Graphics Adapter (adaptador gráfico melhorado)

<PC> IBM PC original e PC-XT

<AT> PC/AT e PC/XT-286

INT 10h	Função 00h definir modo de vídeo	< MDA > < CGA > < EGA >				
entrada:	AH = 00h	AL = (00-1F)				
saída :	nada					
comentário:	a definição dos diversos modos depende dos adaptadores em uso, conforme tabela.					
modo (AL)	resolução (linha/pixel)	cores número	tipo gráfico/texto	MDA	CGA	EGA
00h	40x25(s/fundo)	16(EGA-64)	texto		X	X
01h	40x25(c/fundo)	16(EGA-64)	texto		X	X
02h	80x25(s/fundo)	16(EGA-64)	texto		X	X
03h	0x25(c/fundo)	16(EGA-64)	texto		X	X
04h	20x200	4	gráfico		X	X
05h	320x200	4(cinza)	gráfico		X	X
06h	640x200	2	gráfico		X	X
07h	80x25	2	texto	X		X
08h	(sem uso ou PCJr)					
09h	(sem uso ou PCJr)					
0Ah	(sem uso ou PCJr)					
0Bh	reservado					
0Ch	reservado					
0Dh	320x200	16	gráfico		X	
0Eh	640x200	16	gráfico		X	
0Fh	640x350	2(b/p)	gráfico		X	
10h	640X350	16	gráfico		X	

INT 10h	Função 01h Define o formato do cursor	< MDA > < CGA > < EGA >
entrada:	AH = 01h CH(linha superior do cursor) bits 0-4 CL (linha inferior do cursor) bits 0-4	
saída:	nada	
comentário:	Só aparece no modo texto. Valor usual CH = 6 CL = 7	

INT 10h	Função 02h Define posição do cursor	<MDA> <CGA> <EGA>
entrada:	AH = 02h BH (página) DH (linha, coordenada y) DL (coluna, coordenada x)	
saída:	nada	
comentário:	A numeração das linhas e colunas começa no canto superior esquerdo (0,0) e seu valor máximo depende do modo de vídeo em uso : canto inferior direito (24,39) para 25 linhas e 40 colunas (modos 0 e 1) ou (24,79) para 25 linhas e 80 colunas (modos 2 e 3). No modo gráfico, existe um cursor virtual acessível por esta função. Ver Exemplo 9.1 na Seção 9.2.	

INT 10h	Função 03h Obtém posição do cursor	<MDA> <CGA> <EGA>
entrada:	AH = 03h BH (página)	
saída :	CH (linha superior do cursor) CL (linha inferior do cursor) DH linha (coordenada y) DL coluna (coordenada x)	
comentário:	Obtém a cordenada do cursor e formato para a página apontada.	

INT 10h	Função 04h Obtém posição da caneta ótica	<CGA> <EGA>
entrada:	AH = 04h	
saída :	AH = 00 (se caneta ativada) 01 (se caneta ativada) BX coordenada gráfica x CH coordenada gráfica y DH coordenada de texto y e DL coordenada de texto x	
comentário:	O tamanho máximo das coordenadas depende do modo de vídeo utilizado.	

INT 10h	Função 05h Define página de vídeo ativa	< CGA > < EGA >
entrada:	AH = 05h AL = página (0-7) para modos 00 e 01h (0-3) para modos 02 e 03h	
saída :	nada	
comentário:	A mudança entre as páginas não altera seu conteúdo. Cada página possui 2 Kbytes (40 colunas) ou 4 Kbytes de memória (80 col.)	

INT 10h	Função 06h Inicializa ou move uma janela para cima	< MDA > < CGA > < EGA >
entrada:	AH = 06h AL = número de linhas a serem deslocadas BH = atributos para as linhas em branco (CH,CL) = (y,x) canto superior esquerdo (DH,DL) = (y,x) canto inferior direito	
saída :	nada	
comentário:	O texto movido acima da linha superior da janela é perdido. Ver Exemplo 9.1 na Seção 9.2	

INT 10h	Função 07h Inicializa ou move uma janela para baixo	< MDA > < CGA > < EGA >
entrada:	AH = 07h demais parâmetros iguais à função 06h.	
comentário:	Ver INT 10h Função 06h	

INT 10h	Função 08h Ler caractere e atributo na posição do cursor	< MDA > < CGA > < EGA >
entrada:	AH = 08h BH = página	
saída :	AH = atributo AL = caractere	
comentário:	Obtém o caractere ASCII e seu atributo na atual posição do cursor na página especificada.	

INT 10h	Função 09h Escrever caractere e atributo na posição do cursor	< MDA > < CGA > < EGA >
entrada:	AH = 09h AL = caractere BH = página BL = atributo (se modo texto) ou cor (modo gráfico) CX = número de vezes que o caractere será escrito	
saída :	nada	
comentário:	Escreve em ASCII o caractere e seu atributo. Para escrever o caractere sem alterar o atributo, usar INT 10h, Função 0Ah	

INT 10h	Função 0Ah Escrever caractere e na posição do cursor	< MDA > < CGA > < EGA >
entrada:	AH = 0Ah AL = caractere BH = página CX = número de vezes que o caractere será escrito	
saída :	nada	
comentário:	Ao escrever um caractere na posição do cursor, esta função não altera o atributo do caractere original. Ver Exemplo 9.6 na Seção 9.4	

INT 10h	Função 0Bh Define paleta de cor, fundo e borda	< CGA > < EGA >
entrada:	AH = 0Bh BH = 00h para definir cor de fundo e borda no modo BL = cor gráfico ou cor da borda no modo texto BH = 01h selecionar paleta no modo gráfico 320x200 BL = paleta	
saída :	nada	
comentário:	No modo texto, esta função seleciona somente a cor da borda. No modo gráfico, esta função é válida para seleção da paleta somente para modo gráfico de 4 cores na resolu- ção 320x200 pixel, conforme a seguintetabela:	
	paleta	valor do pixel cor
	0	0 igual a cor de fundo
		1 verde
		2 vermelho
		3 amarelo
1	0	igual à cor de fundo
	1	ciano
	2	magenta
	3	branco

INT 10h	Função 0Ch Escreve pixel no modo gráfico	< CGA > < EGA >
entrada:	AH = 0Ch AL = valor do pixel BH = página CX = coluna (coordenada x) DX = linha (coordenada y)	
saída :	nada	
comentário:	O valor máximo das coordenadas do pixel depende do modo gráfico em uso. A coordenada inicial é o canto su- perior esquerdo (0,0)	

INT 10h	Função 0Dh Ler posição de pixel no modo gráfico	< CGA > < EGA >
entrada:	AH = 0Dh BH = página CX = coluna (coordenada gráfica x) DX = linha (coordenada gráfica y)	
saída :	AL = valor do pixel	
comentário:	Obtém o valor do pixel (ver INT 10h função 0Bh) na posição de coordenada indicada.	

INT 10h	Função 0Eh Escrever caractere no modo teletipo	< MDA > < CGA > < EGA >
entrada:	AH = 0Eh AL = caractere BH = página	
saída :	nada	
comentário:	Diferente da Função 09h, esta função não altera a cor do caractere, embora avance uma posição após execução da tarefa. Reconhece alguns caracteres de controle, ver Tabela A.2 no Apêndice A, tais como chr\$(7), retorno de um caractere chr\$(8) e outros.	

INT 10h	Função 0Fh Obter modo de vídeo corrente	< MDA > < CGA > < EGA >
entrada:	AH = 0F	
saída :	AH = número de caracteres por linha AL = modo de vídeo (ver INT 10h, função 00h) BH = página de vídeo ativa (modo gráfico = 0)	
comentário:	Obtém as características do vídeo, tal como o número de colunas (40 ou 80). Ver Exemplo 11.3 na Seção 11.3	

INT 11h	Obter configuração do sistema	
entrada:	nada	
saída :	AX	lista dos equipamentos do sistema
	bits	configuração
	14 e 15	número de interfaces de impressoras
	13	= 1 se modem interno do PC instalado
	12	= 1 se adaptador de jogos instalado
	9,10,11	número de seriais RS-232 instaladas
	8	reservado
	6 e 7	número de discos flexíveis instalados
	00	→ 1
	01	→ 2
	10	→ 3
	11	→ 4
	4 e 5	modo de vídeo usual
	00	→ reservado (provável adaptador EGA)
	01	→ 40 x 25 CGA modo texto
	10	→ 80 x 25 CGA modo texto
	11	→ 80 x 25 MDA
	2 e 3	Quantidade de RAM (obsoleto)
	1	= 1 se existir co-processador matemático
	0	= 1 se existir ao menos um disk-drive
comentário: Ver Seção 9.3 e programa ACHAR_B.C na Seção 9.5		

INT 12h	Obter tamanho da memória convencional	
entrada:	nada	
saída :	AX =	tamanho da memória em KB (em hexadecimal)
comentário: Ver Exemplo 9.4 na Seção 9.3 e programa ACHAR_B.C na Seção 9.5		

INT 13H	Função 00H Inicializar disco do sistema
entrada:	AH = 00h DL = drive (00h..7Fh) disco flexível (80h..FFh) disco rígido
saída :	Se a operação for bem sucedida flag de carry = 0 e AH = 0 do contrário, flag de carry = 1 e AH = status
comentário:	inicializa o controlador de disco, preparando a operação de I/O no disco apontado.

INT 13H	Função 01h Obter status do sistema de disco																																																				
entrada:	AH = 01h DL = drive (00h..7Fh) disco flexível (80h..FFh) disco rígido																																																				
saída :	AH = 00 AL = status da operação prévia do disco <table> <tr><td>00h</td><td>nenhum erro</td></tr> <tr><td>01h</td><td>comando inválido</td></tr> <tr><td>02h</td><td>marca de endereço não encontrada</td></tr> <tr><td>03h</td><td>disco(flexível) protegido contra escrita</td></tr> <tr><td>04h</td><td>setor não encontrado</td></tr> <tr><td>05h</td><td>falha na inicialização (disco rígido)</td></tr> <tr><td>06h</td><td>disco flexível ausente</td></tr> <tr><td>07h</td><td>erro tabela de parâmetros (disco rígido)</td></tr> <tr><td>08h</td><td>decurso do DMA (disco flexível)</td></tr> <tr><td>09h</td><td>DMA atravessou limite de 64K</td></tr> <tr><td>0Ah</td><td>flag de setor defeituoso (rígido)</td></tr> <tr><td>0Bh</td><td>flag de cilindro defeituoso (rígido)</td></tr> <tr><td>0Ch</td><td>média não encontrada</td></tr> <tr><td>0Dh</td><td>número inválido de setores no formato</td></tr> <tr><td>0Eh</td><td>marca do endereço de controle de dados</td></tr> <tr><td>0Fh</td><td>nível apontado pelo DMA fora da faixa</td></tr> <tr><td>10h</td><td>erro de avaliação de redundância cíclica</td></tr> <tr><td>11h</td><td>checagem de erro com erro de dados</td></tr> <tr><td>20h</td><td>falha no controlador</td></tr> <tr><td>40h</td><td>falha no processo de busca</td></tr> <tr><td>80h</td><td>estouro no tempo, falha na resposta</td></tr> <tr><td>AAh</td><td>drive não está pronto (disco rígido)</td></tr> <tr><td>BBh</td><td>erro indefinido (disco rígido)</td></tr> <tr><td>CCh</td><td>erro na escrita (disco rígido)</td></tr> <tr><td>E0h</td><td>erro no registrador de status (rígido)</td></tr> <tr><td>FFh</td><td>falha na operação (disco rígido)</td></tr> </table>	00h	nenhum erro	01h	comando inválido	02h	marca de endereço não encontrada	03h	disco(flexível) protegido contra escrita	04h	setor não encontrado	05h	falha na inicialização (disco rígido)	06h	disco flexível ausente	07h	erro tabela de parâmetros (disco rígido)	08h	decurso do DMA (disco flexível)	09h	DMA atravessou limite de 64K	0Ah	flag de setor defeituoso (rígido)	0Bh	flag de cilindro defeituoso (rígido)	0Ch	média não encontrada	0Dh	número inválido de setores no formato	0Eh	marca do endereço de controle de dados	0Fh	nível apontado pelo DMA fora da faixa	10h	erro de avaliação de redundância cíclica	11h	checagem de erro com erro de dados	20h	falha no controlador	40h	falha no processo de busca	80h	estouro no tempo, falha na resposta	AAh	drive não está pronto (disco rígido)	BBh	erro indefinido (disco rígido)	CCh	erro na escrita (disco rígido)	E0h	erro no registrador de status (rígido)	FFh	falha na operação (disco rígido)
00h	nenhum erro																																																				
01h	comando inválido																																																				
02h	marca de endereço não encontrada																																																				
03h	disco(flexível) protegido contra escrita																																																				
04h	setor não encontrado																																																				
05h	falha na inicialização (disco rígido)																																																				
06h	disco flexível ausente																																																				
07h	erro tabela de parâmetros (disco rígido)																																																				
08h	decurso do DMA (disco flexível)																																																				
09h	DMA atravessou limite de 64K																																																				
0Ah	flag de setor defeituoso (rígido)																																																				
0Bh	flag de cilindro defeituoso (rígido)																																																				
0Ch	média não encontrada																																																				
0Dh	número inválido de setores no formato																																																				
0Eh	marca do endereço de controle de dados																																																				
0Fh	nível apontado pelo DMA fora da faixa																																																				
10h	erro de avaliação de redundância cíclica																																																				
11h	checagem de erro com erro de dados																																																				
20h	falha no controlador																																																				
40h	falha no processo de busca																																																				
80h	estouro no tempo, falha na resposta																																																				
AAh	drive não está pronto (disco rígido)																																																				
BBh	erro indefinido (disco rígido)																																																				
CCh	erro na escrita (disco rígido)																																																				
E0h	erro no registrador de status (rígido)																																																				
FFh	falha na operação (disco rígido)																																																				

INT 13H	Função 02h Leitura de setores
entrada:	AH = 02h AL = número de setores CH = cilindro CL = setor DH = cabeça DL = drive ES:BX = segmento:offset do buffer
saída :	se operação for bem sucedida flag de carry = 0 AH = 00h AL = número de setores se operação não for bem sucedida flag de carry = 1 AH = status (INT 13 função 01h)
comentário:	em discos flexíveis pode ocorrer um erro, caso o motor do driver estiver desligado no momento de acesso.

INT 13H	Função 03h Escrever num setor
entrada:	AH = 03h, demais idênticos a INT 13h, Função 02h
saída :	idêntico a INT 13h, Função 02h
comentário:	escreve um ou mais setores da memória no disco

INT 13H	Função 04h Verificar um setor
entrada:	AH = 04h, demais idênticos a INT 13h, Função 02h
saída :	idêntico a INT 13h, Função 02h
comentário:	verifica o endereço dos campos de um ou mais setores. Não existe transferência de dados da ou para a memória nesta operação.

INT 13H	Função 05h Formatar uma trilha
entrada:	AH = 05h AL = setor CH = trilha DH = cabeça DL = drive ES:BX segmento:offset da lista do campo de endereço
saída :	idêntico a INT 13h, Função 00h
comentário:	em discos flexíveis a lista do campo de endereços consiste em uma série de entradas de 4 bytes, uma entrada por setor, no formato: byte conteúdo 0 trilha 1 cabeça 2 setor 3 código do tamanho do setor 00h 128 bytes/setor 01h 256 bytes/setor 02h 512 bytes/setor (usual) 03h 1024 bytes/setor

INT 13H	Função 06h Formatar uma trilha defeituosa
entrada:	AH = 06h AL = setores CH = trilha DH = cabeça DL = drive
saída :	idêntico a INT 13h Função 00h
comentário:	exclusivo para disco rígido do PC/XT. Inicializa uma trilha, escrevendo nos endereços do disco de campos e dados, apontando os flags de setores defeituosos.

INT 13H	Função 07h Formatar drive
entrada:	AH = 07h AL = setor CH = cilindro DL = drive
saída :	idêntico a INT 13h Função 00h
comentário:	Exclusivo para disco rígido do PC-XT

INT 13h	Função 08h Obter parâmetros de drive
entrada: AH = 08h DL = drive (00-7Fh flexível, 90-FFh rígido) saída : se operação bem sucedida flag de carry = 0 BL = tipo de drive (para disco flexível AT) 01h 360Kb, 40 trilhas, 5.25" 02h 1.2Mb, 80 trilhas, 5.25" 03h 720Kb, 80 trilhas, 3.5" 04h 1.44Mb, 80 trilhas, 3.5" CH = número máximo de trilhas CL = número máximo de setores DH = número máximo de lados DL = número de drivers ES:DI segmento:offset da tabela dos parâmetros de disk-drive do contrário idêntico a INT 13h Função 00h	
comentário:	exclusivo para disco rígido no PC-XT

INT 13h	Função 09h Inicializa características disco rígido
entrada: AH = 09h DL = drive (80-FFh) disco rígido saída : idêntico a INT 13h Função 02h	
comentário:	inicializa o disco rígido para operações subsequentes de I/O, usando os valores encontrados no bloco de parâmetro de disco do ROM BIOS, cujo endereço no PC-XT é apontado por INT 41h.

INT 13h	Função 0Ah Ler setor longo
entrada: AH = 0Ah outros idênticos a INT 13h Função 02h saída : idêntico a INT 13h Função 02h	
comentário:	Transfere setor ou setores do disco para a memória, com 4 bytes de código de correção e teste de erro (ECC) para cada setor. Somente para discos rígidos

INT 13H	Função 0Bh Escrever setor longo
entrada:	AH = 0B outros idênticos a INT 13h Função 02h
saída :	idêntico a INT 13h Função 02h
comentário:	válido somente para discos rígidos

INT 13H	Função 0Ch Busca de cilindro
entrada:	AH = 0Ch CH = 8 bits inferiores do cilindro CL = 2 bits superiores do cilindro nos bits 6 e 7 DH = cabeça DL = drive (80-FH disco rígido)
saída :	idêntico a INT 13h Função 02h
comentário:	válido somente para disco rígido

INT 13H	Função 0Dh Inicializa disco rígido do sistema
entrada:	AH = 0Dh DL = drive (80-FH disco rígido)
saída :	idêntico a INT 13h Função 02h
comentário:	inicializa o controlador de disco rígido e prepara o sistema para subseqüente operação de I/O

INT 13H	Função 10h Obter status de drive
entrada:	AH = 10h DL = drive
saída :	idêntico a INT 13h Função 02h
comentário:	testa se um disco rígido está operando e retorna o status do drive.

INT 13H	Função 11h Recalibrar drive
entrada:	AH = 11h DL = drive
saída :	idêntico a INT 13h Função 02h
comentário:	solicita ao adaptador do disco rígido uma autocalibração para o drive apontado, posicionando o a leitura para o cilindro 0 e retorna o drive de status.

INT 13H	Função 14h Diagnóstico do controlador interno
entrada:	AH = 14h DL = drive
saída :	idêntico a INT 13h Função 02h
comentário:	solicita ao adaptador do disco rígido a realizar um autodiagnóstico, indicando o status final do disco.

INT 13H	Função 15h Obtém tipo de disco
entrada:	AH = 15h DL = drive (flexível ou rígido)
saída :	se operação for bem sucedida AH = código do tipo de drive 00h → nenhum drive foi detectado 01h → disco flexível, nenhuma troca detectada 02h → disco flexível, com troca detectada 03h → disco rígido. Neste caso CX:DX número de 512 byte/setor se operação for mal sucedida flag de carry = 1 e AH = status
comentário:	retorna um código, indicando o tipo de disco apontado

INT 14h	Função 00h Inicializa porta de comunicação serial		
entrada: AH = 00h AL = parâmetros DX = número da porta serial saída: AH = status da porta	bit 7 6 5 4 3 2 1 0 caso = 1, significado erro de espera muito longa registrador de transferência vazio registrador de manutenção vazio erro detectando um break erro de enquadramento detectado erro de paridade detectado erro de transposição (overrun) pronto para recepção de dados		
AL = status do modem	bit 7 6 5 4 3 2 1 0 caso = 1, significado recebido sinal de detecção de linha indicador de chamada conjunto de dados prontos limpo para enviar alteração no sinal de detecção de linha indicação de subida de borda alteração no status pronto para enviar alteração no status de limpo para enviar		
comentário:	parâmetros de inicialização da porta serial		
bits 7 6 5	4 3	2	1 0
taxa de transmissão (BAUD)	paridade	bit de parada	tamanho da palavra
000 = 110	00 = nenhuma	0 = 1 bit	10 = 7 bits
001 = 150	01 = ímpar	1 = 2 bits	11 = 8 bits
010 = 300	10 = nenhuma		
011 = 600	11 = par		
100 = 1200			
101 = 2400			
110 = 4800			
111 = 9600			

INT 14h	Função 01h Escrever um caractere na porta serial
entrada: saída :	AH = 01h AL = caractere DX = número da porta serial se a operação for bem sucedida AH bit 7 = 0 e status da porta INT 14h Função 00h do contrário AH bit 7 = 1
comentário:	Escreve um caractere numa porta de comunicação serial especificada, retornando o status da porta, conforme INT14h Função 00h

INT 14h	Função 02h Ler um caractere da porta serial
entrada: saída :	AH = 01h DX = número da porta serial se a operação for bem sucedida AH bit 7 = 0 AH bits 0-6 status bit significado (se = 1) 4 detectada solicitação de parada (break) 3 erro de dimensionamento 2 detectado erro de paridade 1 detectado erro de transposição AL = caractere lido se a operação não for bem sucedida AH bit 7 = 1
comentário:	Ler um caractere de uma porta de comunicação serial especificada, retornando o status da porta

INT 14h	Função 03h Obter status da porta serial
entrada: saída :	AH = 01h DX = número da porta serial AH = status da porta (idêntico a INT 14h Função 00h) AL = status do modem (idêntico a INT 14h Função 00h)
comentário:	retorna o status de uma porta serial definida

INT 15h	Interrupção usada para controle de fita cassete na primeira versão do PC e atualmente obsoleta. Utilizado no controlador de disco rígido nos novos modelos PS/2, porém sem uso especial no PC XT
----------------	--

INT 16h	Função 00h Ler um caractere do teclado
entrada:	AH = 00h
saída :	AH = código de varredura do teclado AL = caractere
comentário:	ler o caractere do teclado, retornando o código de varredura.

INT 16h	Função 01h Obter status do teclado
entrada:	AH = 01h
saída:	se uma tecla foi acionada e aguarda ser assimilada flag de zero = 0 AL = caractere AH = código de varredura
comentário:	Determina se uma tecla foi ou não acionada. Ver Exemplo 9.6 na Seção 9.4

INT 16h	Função 02h Obter status das teclas especiais
entrada:	AH = 02h
saída :	AL = flag
	bits tecla ativada (se = 1)
	7 Insert on
	6 Caps Lock on
	5 Num Lock on
	4 Scroll Lock on
	3 tecla Alt pressionada
	2 tecla Ctrl pressionada
	1 tecla Shift esquerda pressionada
	0 tecla Shift direita pressionada
comentário:	retorna os flags da ROM BIOS que monitora as teclas especiais. Ver Exemplo 9.6 na Seção 9.4

INT 17h	Função 00h Escrever um caractere na impressora
entrada:	AH = 00h AL = caractere DX = número da impressora 0 = LPT1, 1 = LPT2
saída :	AH = status da impressora bit status (se bit = 1) 7 impressora está livre 6 protocolo da impressora (acknowledge) 5 fim do papel 4 impressora selecionada 3 erro de I/O 2 não usada 1 não usada 0 demora excessiva na resposta

INT 17h	Função 01h Inicializa porta paralela da impressora
entrada:	AH = 01h DX = número da impressora
saída :	status (idêntico a INT 17h Função 00h)
comentário:	Inicializa uma porta especificada da impressora e retorna seu status.

INT 17h	Função 02h Obter status da impressora
entrada:	AH = 02h DX = número da impressora
saída :	status (idêntico a INT 17h Função 00h)
comentário:	Obtém o status de uma porta especificada da impressora

INT 18h	ROM BASIC transfere o controle para o BASIC residente quando estiver instalado.
entrada:	nada
saída :	nada

INT 19h	Reinicializa o sistema
entrada :	nada
saída :	nada
comentário:	reinicializa (reboot) o sistema a partir do disco flexível, ou do rígido na sua falta.

INT 1Ah	Função 00h Obter número de pulsos do relógio
entrada:	AH = 00h
saída :	AL = 00 se não passou por meia-noite desde a última leitura. Do contrário, AL 00h
CX:DX	número de pulsos após meia-noite
comentário:	O valor retornado no contador é o número de pulsos contados a partir da meia-noite. Como a frequência do clock é de 18,2 Hz, ver Seção 2.3, quando o contador atinge 1.573.040, ele volta a zero acionando o flag.

INT 1Ah	Função 01h Definir valor do contador
entrada:	AH = 01h CX:DX = valor a ser carregado (32 bits)
saída :	nada
comentário:	Armazena um valor no contador que pode ser lido através de INT 1Ah Função 00h.

INT 1Ah	Função 02h Ler hora
entrada:	AH = 02h
saída :	CH = horas em decimal codificado em binário (BCD) CL = minutos em BCD DH = segundos em BCD flag de carry = 0 se relógio está operando

INT 1Ah	Função 03h Definir hora
entrada:	AH = 03h CH = horas CL = minutos DH = segundos
saída :	nada
comentário:	Os valores devem ser carregados no formato BCD

INT 1Ah	Função 04h Obter data
entrada:	AH = 04h
saída :	CH = século (19 ou 20) CL = ano DH = mês DL = dia
comentário:	valores em BCD. Lê o relógio interno do PC se este existir.

INT 1Ah	Função 05h Definir data
entrada:	AH = 05h CH = século (19 ou 20) CL = ano DH = mês
saída :	DL = dia em valores BCD
comentário:	Define a data do relógio interno

AS FUNÇÕES DOS

Ao contrário do ROM-BIOS, o DOS utiliza praticamente uma única interrupção, a INT 21h (ou 33) e uma centena de funções a ela associadas. No Capítulo 10, mostramos o conjunto das interrupções DOS, ver Tabela 10.2 na Seção 10.1. Neste Apêndice, vamos nos concentrar nas funções documentadas dispostas em ordem numérica. No Capítulo 10, resumimos estas funções de acordo com os resultados finais.

Elas devem ser usadas prioritariamente a outras interrupções, tanto do DOS como do BIOS. Algumas interrupções DOS são consideradas um tanto obsoletas, pois existem funções que desempenham a mesma tarefa com maior eficiência. As funções DOS são baseadas no sistema operacional que evolui constantemente, ver Tabela 10.1 na Seção 10.1, e são menos dependentes das características de hardware, tal como o BIOS INT. Isto garante maior portabilidade aos programas desenvolvidos baseados nas funções DOS.

Todas as funções abaixo relacionadas, referem-se a uma única interrupção, a DOS INT 21h, comum às versões DOS 2.0 e posteriores.

Função 00h	Termina programa
entrada: saída :	AH = 00h CS: endereço do segmento do PSP nada
comentário:	O PSP "Prefix Segment Program" é um bloco de memória que controla a instalação e o término de um programa no ambiente DOS. Esta função é considerada obsoleta e pode ser substituída por outras funções tais como: Função 31h e 4Ch.

Função 01h	Entrada do teclado com eco
entrada: AH = 01h saída: AL caractere de entrada	
comentário:	Aguarda um caractere enviado do teclado e envia para o dispositivo padrão de saída, em geral o vídeo. No caso de caractere especial de teclado estendido, ver Tabela A.4 no Apêndice A, é necessário usar esta função duas vezes em seguida se o primeiro caractere lido for 0.

Função 02h	Saída de um caractere
entrada: AH = 02h DL = caractere saída: nada	
comentário:	Envia um caractere para o vídeo ou outro dispositivos saída pré-definido. A redefinição do dispositivo de saída pode ser feita através do comando DOS MODE. Ver função 09Ch para envio de cadeia de caracteres.

Função 03h	Entrada de um caractere porta serial
entrada: AH = 03h saída: AL = caractere lido	
comentário:	Lê um caractere da porta serial padrão COM1 ou de outro dispositivo previamente definido. Esta função não permite a detecção de erro de comunicação serial, tal como a BIOS INT 14, ver Apêndice D.

Função 04h	Saída de um caractere porta serial
entrada: AH = 04h DL = caractere a ser enviado saída: nada	
comentário:	Envia um caractere para a porta serial COM1 ou outro dispositivo padrão pré-definido. Se o dispositivo de saída estiver ocupado o DOS espera o momento em que o caractere pode ser enviado. Esta função não detecta erros na comunicação serial tal como o BIOS INT 14h, ver Apêndice D.

Função 05h	Saída de um caractere para impressora
entrada: AH = 05h DL = caractere a ser enviado saída: nada	
comentário:	Envia um caractere para a porta paralela da impressora ou outro dispositivo pré-configurado. A redefinição do dispositivo de saída é feita através do comando DOS MODE. Por Exemplo, MODE LPT1 = COM1.

Função 06h	Entrada de Teclado saída para o Vídeo
entrada: AH = 0h DL = 00h FEh se saída saída: se saída, nada FFh se entrada se entrada, caso caractere de teclado disponível: Flag zero = 0 e AL = 8 bits de entrada se nenhum caractere estiver disponível, Flag zero = 1	
comentário:	Caso tratar-se de tecla especial, a função deve ser usada 2 vezes. A primeira retornando 00 e a segunda o código ASCII estendido para o PC. Ver Tabela A.4, Apêndice A. Os dispositivos de entrada e saída podem ser redefinidos.

Função 07h	Entrada de caractere sem filtro e eco
entrada: AH = 07h saída: AL = caractere de entrada	
comentário:	Recebe um caractere do teclado, ou outro dispositivo pré-definido e encaminha para o vídeo. A checagem dos comandos de parada, como CTRL + C, está desativada

Função 08h	Entrada de caractere sem eco
entrada: AH = 08h saída: AL = caractere de entrada	
comentário:	Ao contrário da Função 01h, este comando não envia o caractere lido ao vídeo.

Função 09h	Enviar cadeia de caractere ao vídeo	
entrada:	AH = 0h	DS:DX apontador do endereço da cadeia
saída:	nada	
comentário:	Enviar uma cadeia ao vídeo ou outro dispositivo pré-definido. Término da cadeia deve ser marcado como caractere \$ (24h). Ver Exemplo 10.2 na Seção 10.3	

Função 0Ah	Entrada do Teclado com buffer	
entrada:	AH = 0A h	DS:DX apontador do buffer
saída:	nada	
comentário:	Ler uma cadeia de bytes do teclado ou outro dispositivo pré-definido até detectar um caractere de controle de retorno CHR\$(0Dh) ou CHR\$(13) ver Tabela A.2 no Apêndice A.	

Função 0Bh	Verifica estado de entrada do teclado	
entrada:	AH = 0Bh	
saída:	se houver caractere disponível, do contrário	AL = 00h AL = FFh
comentário:	Verifica se existe um caractere disponível no teclado ou de outro dispositivo padrão pré-definido.	

Função 0Ch	Limpar buffer do teclado e ler caractere	
entrada:	AH = 0Ch AL = 01h , 06h, 07h, 08h ou 0Ah(DS:DX)	
saída:	se entrada Função 01h, 06h, 07h ou 08h AL = caractere lido da entrada se AL = 0Ah nada	
comentário:	Após limpar o buffer do teclado ou outra entrada padrão pré-definido, chama outra função DOS. No caso de chamar a Função 0Ah, DS:DX aponta para o buffer de teclado.	

Função 0Dh	Reseta disco
entrada:	AH = 0Dh
saída:	nada
comentário:	Esvazia todos os buffers de arquivo e reseta o disco. Função um tanto obsoleta, pois a função 68h desempenha idêntica tarefa, atualizando o diretório do disco e a tabela de alocação de arquivo (FAT).

Função 0Eh	Seleciona drive corrente
entrada:	AH = 0Eh DL = número do drive (A = 0, B = 1, etc..)
saída:	AL = número de drivers lógicos do sistema
comentário:	adota o drive especificado como drive corrente

Função 0Fh	Abre arquivo
entrada:	AH = 0Fh
saída:	DS:DX apontador do bloco controle do arquivo (FCB) se arquivo não for encontrado, AL = FFh do contrário AL = 00h
comentário:	função considerada obsoleta, pois foi aperfeiçoada pela função 3Dh.

Função 10h	Fecha arquivo
entrada:	AH = 10h
saída:	DS:DX apontador do bloco controle de arquivo (FCB) idêntico à função 0Fh
comentário:	função considerada obsoleta, pois foi aperfeiçoada pela função 3Eh

Função 11h	Procurar primeiro arquivo
entrada:	AH = 11h DS:DX apontador do bloco controle de arquivo (FCB)
saída:	idêntico à função 0Fh
comentário:	função considerada obsoleta, pois foi aperfeiçoada pela função 4Eh

Função 12h	Encontra próximo arquivo
entrada:	AH = 12h DS:DX apontador do bloco controle de arquivo (FCB)
saída:	idêntico à função 0Fh
comentário:	função considerada obsoleta, pois foi aperfeiçoada pela função 4Eh

Função 13h	Deleta arquivo
entrada:	AH = 13h DS:DX apontador do bloco controle de arquivo (FCB)
saída:	idêntico à função 0Fh
comentário:	função considerada obsoleta, pois foi aperfeiçoada pela função 41h

Função 14h	Ler Arquivo Sequencial
entrada:	AH = 14h DS:DX apontador do bloco controle de arquivo (FCB)
saída:	AL = 00h operação bem sucedida 01h encontrou fim de arquivo 02h excedeu segmento 03h registro parcial no final de arquivo
comentário:	função considerada obsoleta, pois foi aperfeiçoada pela função 3Fh

Função 15h	Gravar arquivo sequencial
entrada:	AH = 15h DS:DX apontador do bloco controle de arquivo (FCB)
saída:	AL = 00h se gravação bem sucedida AL = 01h se disco cheio AL = 02h se excedeu segmento
comentário:	função considerada obsoleta, pois foi aperfeiçoada pela função 40h

Função 16h	Cria arquivo
entrada:	AH = 16h DS:DX apontador do bloco controle de arquivo (FCB)
saída:	idêntico à função 0Fh
comentário:	função considerada obsoleta pois foi aperfeiçoada pelas funções 3Ch, 5Ah ou 5Bh

Função 17h	Renomeia arquivo
entrada:	AH = 17h DS:DX apontador do bloco controle de arquivo (FCB)
saída:	idêntico à função 0Fh
comentário:	função considerada obsoleta, pois foi aperfeiçoada pela função 56h

Função 19h	Obter disco corrente
entrada:	AH = 19h
saída:	AL = código de drive (A = 0, B = 1, etc..)
comentário:	Ver programa ACHAR_D.C no Capítulo 10.4

Função 1Ah	Define endereço da área de transferência de disco
entrada:	AH = 1Ah
saída:	DS:DX apontador da área de transferência de disco
comentário:	nada
comentário:	Ao estado usual é atribuído FCB (File Control Block) um buffer de 128 bytes do CS definido pelo PSP (Prefixo de Segmento de Programa), e um offset deslocamento) 0080h, tal como é utilizado no Debug, ver Apêndice B.

Função 1Bh	Obtém informação do driver em uso
entrada:	AH = 1Bh
saída:	AL = FFh se operação mal sucedida.
comentário:	função considerada obsoleta, aperfeiçoada pela Função 36h.

Função 1Ch	Obtém informação de qualquer driver
entrada:	AH = 1Ch
saída:	AL = FFh se operação mal sucedida.
comentário:	função considerada obsoleta, aperfeiçoada pela Função 36h.

Função 21h	Lê arquivo randômico
entrada:	AH = 21h
saída:	DS:DX apontador do FCB do arquivo aberto
saída:	idêntico à Função 14h
comentário:	função considerada obsoleta, aperfeiçoada pela Função 3Fh.

Função 22h	Grava arquivo randômico
entrada:	AH = 22h
saída:	DS:DX apontador do FCB de arquivo não aberto
saída:	idêntico à Função 15h
comentário:	função considerada obsoleta, aperfeiçoada pela Função 40h.

Função 23h	Obtém tamanho do arquivo
entrada:	AH = 23h DS:DX apontador do FCB de arquivo não aberto
saída:	idêntico à Função 0Fh
comentário:	Retorna o dado no registro FCB. Função considerada obsoleta, aperfeiçoada pela Função 42h

Função 24h	Define campo de registro randômico
entrada:	AH = 24h DS:DX apontador do FCB de arquivo aberto
saída:	altera somente bloco de controle de arquivo FCB ("File Control Block")
comentário:	Retorna o dado no registro FCB. Função considerada obsoleta, aperfeiçoada pela Função 42h

Função 25h	Define vetor de interrupção
entrada:	AH = 25h AL = número da interrupção(em hexadecimal) DS:DX apontador da rotina de interrupção
saída:	nada
comentário:	Ver Exemplo 10.4 na Seção 10.3

Função 26h	Cria novo segmento de programa (PSP)
entrada:	AH = 26h DX = endereço de segmento do novo PSP
saída:	nada
comentário:	função considerada obsoleta, aperfeiçoada pela Função 4Bh.

Função 27h	Lê registros de arquivo randômico
entrada:	AH = 27h CX = número de registros a serem lidos DS:DX apontador do FCB de arquivo aberto
saída:	AL = idêntico à Função 14h CX = número de registros lidos
comentário:	função considerada obsoleta, aperfeiçoada pela Função 3Fh.

Função 28h	Grava arquivo randômico
entrada:	AH = 28h CX = número de registros a serem gravados DS:DX apontador do FCB de arquivo aberto
saída:	AL = idêntico à Função 15h CX = número de registros gravados
comentário:	função considerada obsoleta, aperfeiçoada pela Função 40h.

Função 29h	Analisa nome de arquivo
entrada:	AH = 29h AL = vetor de controle da análise DS:SI = apontador da cadeia de texto ES:DI = apontador do FCB
saída:	AL = indica o tipo de arquivo procurado DS:SI apontador do final do nome do arquivo ES:DI apontador do FCB formatado e não aberto
comentário:	Analisa uma linha de comando, procurando identificar o formato de uma cadeia a um arquivo, no formato drive: arquivo.extensão.

Função 2Ah	Obtém data
entrada:	AH = 2Ah
saída:	CX = ano (1980..2099) DH = mês DL = dia AL = dia da semana
comentário:	Ver programa ACHAR_D.C Seção 10.4

Função 2Bh	Define data
entrada:	AH = 2Bh CX = ano (1980...2099) DH = mês DL = dia
saída:	se data definida com sucesso AL = 00h do contrário AL = FFh
comentário:	Inicializa a data do sistema sem afetar a hora

Função 2Ch	Obtém hora
entrada:	AH = 2Ch
saída:	CH = horas CL = minutos DH = segundos DL = centésimos
comentário:	Obtém hora do dia do relógio de tempo real do sistema. O valor de centésimos de segundos (0.01s) é um aproximado.

Função 2Dh	Define hora
entrada:	AH = 2Dh
saída:	CH = horas CL = minutos DH = segundos DL = centésimos idêntico à Função 2B

Função 2Eh	ativa/desativa flag de verificar gravação
entrada:	AH = 2Eh
saída:	AL = 00h para desligar flag e 01h para ligar nada
comentário:	Controla, manualmente, a verificação de leitura após gravação de dados.

Função 2Fh	Obtém endereço da transferência de disco
entrada:	AH = 2Fh
saída:	ES:BX apontador da área de transferência de disco
comentário:	obtém endereço para manipulação de arquivo com o FCB. A posição usual é o offset 0080h no PSP

Função 36h	Obtém informação do espaço livre no disco
entrada: AH = 36h DL = código de drive(usual = 0, A = 1,etc..) saída: AX = número de setores por cluster BX = número de clusters disponíveis CX = bytes por setor DX = número total de clusters	
comentário:	Se função não tiver sucesso, AX = FFFFh. Ver programa ACHAR_D.C na Seção 10.4

Função 38h	Obtém/define Informação do País
entrada: AH = 38h DS:DX apontador do buffer para receber informações AL = 00 para informação padrão ou do país corrente saída: DX = FFFFh para definir código de país. se função bem sucedida, flag de carry = 0 BX = código do país. do contrário flag de carry = 1 e AX código de erro	
comentário:	Esta função é diferente para versões DOS 2.0 e DOS 3.0. Além disto existem diferenças entre a implementação PC-DOS (IBM) e MS-DOS (Microsoft). Os principais grupos de países referem-se ao formato de datas e moedas para EUA, Europa e Japão.

Função 39h	Cria diretório: MKDIR
entrada: AH = 39h DS:DX apontador endereço da via em caractere ASCIIZ (ASCII estendido) saída: se sucesso flag de carry = 0 do contrário, flag de carry = 1 e AX = código de erro	
comentário:	Cria um subdiretório. O término da cadeia, indicando a via é o byte nulo. Códigos de erro possíveis, AL = 3 caso via não encontrada e AL = 5 acesso negado.

Função 3Ah	Remove diretório - RMDIR
entrada:	AH = 3Ah DS:DX idêntico à Função 39h
saída:	idêntico à Função 39h
comentário:	remove ("delete") um subdiretório, usando a via(pathname) especificada.

Função 3Bh	Altera diretório corrente - CHDIR
entrada:	AH = 3Bh DS:DX idêntico à Função 39h
saída:	idêntico à Função 39h
comentário:	altera ("change") um subdiretório, usando a via(pathname) especificada.

Função 3Ch	Cria arquivo - CREAT
entrada:	AH = 3Ch DS:DX idêntico à Função 39h CX = atributo de arquivo
	00h normal 01h somente leitura 02h escondido 03h sistema
saída:	se bem sucedido, flag de carry = 0 AX = manipulador do contrário, flag de carry = 1 AX = código de erro
comentário:	Cria um arquivo com o nome apontado. O arquivo é aberto e retorna um manipulador para uso pelo programa.

Função 3Dh	Abre arquivo
entrada:	AH = 3Dh DS:DX idêntico à Função 39h
saída:	AL = modo de acesso idêntico à Função 3Ch
comentário:	Abre um arquivo com o nome apontado. O arquivo é aberto e retorna um manipulador para uso pelo programa.

Função 3Eh	Fecha arquivo
entrada:	AH = 3Eh BX = manipulador
saída:	idêntico à Função 39h
comentário:	Através de um manipulador de arquivo, o buffer interno é esvaziado, o arquivo fechado e o manipulador liberado. O manipulador deve ser um valor válido sob pena de pane geral.

Função 3Fh	Lê arquivo ou dispositivo
entrada:	AH = 3Fh BX = manipulador CX = número de bytes a serem lidos DS:DX apontador da área de buffer
saída:	idêntico à Função 39h
comentário:	Conhecidos um manipulador de arquivo, um endereço de buffer e o número de bytes, esta função transfere dados do apontador do arquivo de disco para o buffer.

Função 40h	Grava arquivo ou dispositivo
entrada:	AH = 40h BX = manipulador CX = número de bytes a gravar DS:DX apontador da área de buffer
saída:	idêntico à Função 39h
comentário:	Conhecidos um manipulador de arquivo, um endereço do buffer e o número de bytes os dados do buffer são transferidos para o arquivo ou dispositivo.

Função 41h	Deleta arquivo
entrada:	AH = 41h DS:DX apontador da via do arquivo em ASCIIZ
saída:	idêntico à Função 39h
comentário:	Deleta um arquivo de disco no diretório apontado. Substitui o primeiro caractere do nome do arquivo pelo caractere e (CHR\$(E5h)) e marca os clusters do arquivo correspondente como livre da tabela de alocação de arquivos- FAT ("File Allocation Table"). Os dados armazenados nestes clusters não são utilizados, a menos que a gravação de um outro arquivo necessite deste espaço. Isto permite a recuperação de arquivos recém-deletados ("Undelete")

Função 42h	Altera ponteiro de arquivo
entrada:	AH = 42h AL = código de método 00h deslocamento absoluto do início do arquivo 01h deslocamento a partir da posição atual 02h deslocamento a partir do fim de arquivo
saída:	BX = manipulador CX:DX valor do deslocamento se sucesso flag de carry = 0 DX:AX ponteiro de arquivo do contrário flag de carry = 1 e AX = código de erro
comentário:	define o ponteiro de arquivo, indicando a posição para início de escrita/leitura de novo arquivo.

Função 43h	Obtém/define atributos de arquivo - CHMOD
entrada:	AH = 43h AL = 00h (01h) para obter (definir) o atributo CX = atributo de arquivo (se definir) bit 5 = arquivo bit 2 = sistema bit 1 = escondido bit 0 = somente leitura DS:DX apontador do nome da via
saída:	se bem sucedido flag de carry = 0, CX = atributo do contrário, flag de carry = 1 e AX = código de erro
comentário:	obtém ou modifica o atributo de um arquivo ("Change Mode")

Função 44h	Controle de E/S para dispositivos-IOCTL
entrada:	AH = 44h AL = subfunção BL = driver BX = manipulador de arquivo CX = número de bytes a serem lidos ou gravados
saída:	se bem sucedido flag de carry = 0 AX = número de bytes do contrário flag de carry = 1 AX = código de erro
comentário:	Esta função está associada a um conjunto de sub-funções dependentes da versão DOS em uso. Permite a comunicação direta entre um programa aplicativo e um driver.

Função 45h	Duplica manipulador de arquivo
entrada:	AH = 45h BX: manipulador a ser duplicado
saída:	se bem sucedido flag de carry = 0 AX = manipulador do contrário flag de carry = 1 e AX = código de erro
comentário:	possibilita o uso de 2 manipuladores para um mesmo arquivo.

Função 46h	Redireciona manipulador duplo
entrada: AH = 46h BX = manipulador para arquivo CX = manipulador a ser redirecionado	
saída:	idêntico à Função 45h
comentário:	Com dois manipuladores redireciona o segundo a referir-se ao mesmo dispositivo do primeiro.

Função 47h	Obtém diretório corrente
entrada: AH = 47h DL = (driver) DS:SI apontador buffer 64 bytes	
saída:	se bem sucedido buffer com nome do arquivo do contrário flag de carry = 1 e AX = código de erro
comentário:	O nome obtido não retorna drive nem separação "\". Apontador de término é o byte nulo.

Função 48h	Aloca bloco de memória
entrada: AH = 48h BX = número de parágrafos de memória	
saída:	se função bem sucedida flag de carry = 0 AX = segmento inicial do bloco alocado do contrário flag de carry = 1 AX = código de erro e BX = parágrafos do maior bloco disponível
comentário:	O DOS procura no endereço mais baixo o espaço suficiente para alocar a quantidade de memória solicitada.

Função 49h	Libera bloco de memória alocada
entrada: AH = 49h ES = segmento do bloco a ser liberado	
saída:	idêntico à Função 39h
comentário:	Libera um bloco de memória alocada por 48h para uso por outros programas.

Função 4Ah	Modifica bloco de memória alocada
entrada:	AH = 4Ah BX = quantidade de parágrafos do novo bloco ES = segmento do bloco a ser liberado
saída:	idêntico à Função 39h
comentário:	diminui ou aumenta dinamicamente um bloco de memória.

Função 4Bh	Carrega/Executa um programa - EXEC
entrada:	AH = 4Bh AL = subfunção ES:BX apontador do bloco de parâmetros DS:Dx apontador do nome do arquivo a ser carregado
saída:	idêntico à Função 39h
comentário:	Permite um programa carregar um subprograma, executando-o opcionalmente, dependendo de AL (AL = 0 carrega e executa, AL = 3 carrega simplesmente).

Função 4Ch	Termina Processo
entrada:	AH = 4Ch
saída:	AL = código de retorno nada
comentário:	É o modo mais indicado para finalizar um processo e retornar ao ambiente de origem. Ao final, toda a memória alocada pelo processo é liberada, os buffers de arquivos esvaziados e os manipuladores de arquivos abertos são fechados.

Função 4Dh	Obtém código de retorno
entrada: AH = 4Dh saída: AH = código de saída 00h término normal 01h término por CTRL + C manual 02h término por manipulador de erro crítico 03h término por programa residente (31h) AL = código de retorno (00h se com Função 00h)	
comentário:	usado pelo programa de origem para obter informação a respeito de um programa tipo EXE por ele solicitado e terminou de modo bem sucedido.

Função 4Eh	Encontrar o primeiro arquivo
entrada: AH = 4Eh CX = atributo chave de pesquisa DS:DX apontador do nome de arquivo saída: se sucesso flag de carry = 0 e área de transferência de disco preenchido - DTA (Data Transfer Area) do contrário, flag de carry = 1 e AX = código de erro	
comentário:	Conhecido o nome do arquivo procura no drive especificado, ou em uso, o arquivo solicitado.

Função 4Fh	Encontrar próximo arquivo
entrada: AH = 4Fh CX e DS:DX idêntico à Função 4Eh saída: idêntico à Função 4Eh	
comentário:	Após a Função 4Eh ser bem sucedida, a Função 4Fh continua procurando próximo arquivo.

Função 54h	Obtém flag de estado de verificação
entrada:	AH = 54h
saída:	AL = 01h (00h) se (des)ligada
comentário:	verifica se a operação de leitura após gravação (o que aumenta o tempo de acesso ao disco) está ou não ativada. Este estado pode ser alterado pelo comando DOS VERIFY (on/off)

Função 56h	Renomear arquivo - RENAME
entrada:	AH = 56h DS:DX apontador do nome do arquivo fonte
saída:	ES:DI apontador do nome do arquivo destino idêntico à Função 39h
comentário:	renomeia um arquivo ou move de um diretório para outro no mesmo disco. Se já existir um arquivo com o novo nome, a operação não será concluída.

Função 57h	Obtém/define data e hora do arquivo
entrada:	AH = 57h para obter hora/data AL = 00h BX = manipulador para definir AL = 01h BX = manipulador CX = hora DX = data
saída:	obtendo hora/data com sucesso CX = hora DX = data flag de carry = 0 do contrário, flag de carry = 1 e AX = código de erro
comentário:	permite obter ou definir a hora e data associado a um arquivo aberto ou criado previamente.

Função 59h	Obtém código de erro estendido VERSÃO DOS 3.0 OU POSTERIOR
entrada:	AH = 59h BX = 00h
saída:	AX = código de erro MS-DOS estendido. Ver Tabela E.2 ao final.
	BH = classe de erro, conforme código
01h	falta de recurso, não há mais o que pedimos
02h	situação provisória, tentar novamente?
03h	acesso não habilitado
04h	falha no DOS, não é erro do usuário
05h	falha no hardware
06h	falha no software do sistema (problemas DOS?)
07h	falha no software do programa aplicativo
08h	arquivo não encontrado
09h	arquivo ou formato inválido
0Ah	arquivo travado
0Bh	defeito no disco ou driver
0Ch	outros erros
	BL = ação sugerida ao usuário, conforme código
01h	tentar mais algumas vezes antes de abortar
02h	tentar mais vezes com tempo de espera
03h	indagar ao usuário valor correto (drive? nome?)
04h	terminar programa fechando arquivos
05h	sair imediatamente sem fechar arquivos
06h	ignorar erro
07h	tentar novamente após correção do usuário.
	CH = localização do erro, conforme código
01h	desconhecido
02h	dispositivo de bloco (disco)
03h	rede local
04h	dispositivo serial (modem, impressora)
05h	memória
comentário:	Após uma indicação de erro ao término de uma Função DOS, esta Função pode ser solicitada para identificar o código (AX), o tipo (BH) e a localização (CH) do erro. Ver código de erro MS-DOS estendido no final deste Apêndice.

Tabela E.1 Funções de DOS INT 21h

As demais funções DOS ou podem ser substituídas pelas acima descritas, ou só funcionam no modo de rede quando o programa Networks da Microsoft estiver sendo executado.

O MS-DOS (Microsoft) possui um código de erros que é atualizado a cada nova versão DOS. A Tabela abaixo, mostra o código de retorno em AX após a detectar um erro e solicitar a Função 59h, embora possa ser usado em outros programas e compiladores.

código h = hex(decimal)		significado
01h	(01)	número de função inválido
02h	(02)	arquivo não encontrado
03h	(03)	via ("path") de acesso não encontrada
04h	(04)	muitos arquivos abertos, não há manipulador
05h	(05)	acesso negado
06h	(06)	manipulador inválido
07h	(07)	bloco de controle de memória inválido ou destruído
08h	(08)	memória insuficiente
09h	(09)	endereço do bloco de memória inválido
0Ah	(10)	ambiente inválido
0Bh	(11)	formato inválido
0Ch	(12)	código de acesso a arquivo inválido
0Dh	(13)	dados inválidos
0Eh	(14)	reservado (ou sem uso)
0Fh	(15)	driver especificado inválido
10h	(16)	tentativa de remover diretório em uso
11h	(17)	não é o mesmo dispositivo
12h	(18)	não há mais arquivos para procurar
13h	(19)	disco protegido contra escrita
14h	(20)	unidade de disco inválida
15h	(21)	drive não está pronto ("not ready")
16h	(22)	comando desconhecido ou inválido
17h	(23)	erro de dados no disco (CRC)
18h	(24)	erro de tamanho na estrutura de pedido
19h	(25)	erro de busca no disco

Tabela E.2 Código de Erro MS-DOS estendido

código		significado
h = hex(decimal)		
1Ah	(26)	não é um disco DOS, formato desconhecido
1Bh	(27)	setor de disco não encontrado
1Ch	(28)	falta de papel na impressora
1Dh	(29)	falha no processo de gravação
1Eh	(30)	falha no processo de leitura
1Fh	(31)	falha geral
20h	(32)	violação de arquivo compartilhado
21h	(33)	violação na trava dos arquivos
22h	(34)	troca de disco inválida
23h	(35)	bloco de controle de arquivo (FCB) não disponível
24h	(36)	estouro ("overflow") de buffer compartilhado
25h..41h	(37..49)	reservado
42h	(50)	falha na requisição de rede
43h	(51)	terminal remota não recebe
44h	(52)	nome duplo na rede
45h	(53)	nome de rede não encontrado
46h	(54)	rede ocupada
47h	(55)	dispositivo não conectado à rede
48h	(56)	limite do comando de netBIOS excedido
49h	(57)	erro de hardware no adaptador da rede
4Ah	(58)	resposta da rede incorreta
4Bh	(59)	erro imprevisto na rede
4Ch	(60)	adaptador remoto incompatível
4Dh	(61)	fila de impressão lotada
4Eh	(62)	Espaço insuficiente para imprimir arquivo
4Fh	(63)	impressão de arquivo cancelada
50h	(64)	nome da rede deletado
51h	(65)	acesso negado
52h	(66)	tipo incorreto de dispositivo da rede
53h	(67)	nome da rede não encontrado
54h	(68)	nome da rede excedeu limite
55h	(69)	excedido limite da sessão de Net BIOS
56h	(70)	pausa temporária
57h	(71)	pedido de rede não atendido
58h	(72)	pausa no redirecionamento na impressora ou disco
59h..5Fh	(73..79)	reservado

Tabela E.2 Código de Erro MS-DOS estendido (continuação)

código		significado
h = hex(decimal)		
60h	(80)	arquivo já existe
61h	(81)	não usado
62h	(82)	impossível criar diretório
63h	(83)	falha na DOS INT 24h (erro crítico)
64h	(84)	falha na estrutura
65h	(85)	atributo já utilizado
66h	(86)	senha (password) inválida
67h	(87)	parâmetro inválido
68h	(88)	falha de escrita/gravação na rede

Tabela E.2 Código de Erro MS-DOS estendido (continuação)

PROTÓTIPO DOS PROGRAMAS EM C

/* programa cabeçalho para uso com DOS e BIOS Interrupt

* salvar como A:CABEÇA.H

* deve estar no diretorio de include e lib

* CABEÇA.H Turbo C 2.0

* versão outubro de 1989 Antonio A.S.Brito

*/

/* include headers area */

include < dos.h > /* declarações p/ DOS call & int 8086 */

include < bios.h > /* declarações p/ ROM BIOS rotinas */

include < stdarg.h > /* standard argumentos */

include < conio.h > /* DOS console & I/O rotinas */

include < stdio.h > /* standard I/O Kernighan & Ritchie */

include < stdlib.h > /* rotinas básicas de busca/conversão */

/* macro definições */

define NULL 0

define FALSE 0

define TRUE 1

define ZERO 0x30 /* caractere ASCII de zero */

define CR 0x0D /* caractere ASCII de carriage return */

define LF 0x0A /* caractere ASCII de line feed */

define BEL 0x08 /* caractere ASCII de BELL */

/*

incluir prototipos das funções */

void esperar(void); /* mensagem ao final dos módulos */

void info_rom(void); /* obter informações da ROM */

void info_bios(void); /* obter informações do BIOS */

/* variáveis globais, num único módulo */

```
# ifdef MAIN
# define loc          /*      variável local se MAIN módulo      */
# else
# define loc extern    /*      definido em um único módulo      */
# endif
/*      lista de variáveis globais      */
loc union REGS regs;   /*      redefinição de registradores      */
loc struct SREGS segres; /*      redefinição de reg. de segmentos      */

/*      DESTACAR REGISTRADORES      */
```

```
# define AX regs.x.ax
# define BX regs.x.bx
# define CX regs.x.cx
# define DX regs.x.dx
# define SI regs.x.si
# define DI regs.x.di
# define AH regs.h.ah
# define AL regs.h.al
# define BH regs.h.bh
# define BL regs.h.bl
# define CH regs.h.ch
# define CL regs.h.cl
# define DH regs.h.dh
# define DL regs.h.dl
# define DS segregs.ds
# define ES segregs.es
```

```
/*      MACROS DOS INTERRUPT      */
```

```
# define CALLDOS      intdos(&regs,&regs)
# define CALLDOSX      intdosx(&regs,&regs,&segregs)
# define CALLINT(num)  int86(num,&regs,&regs)
# define CALLINTX(num) int86x(num, &regs,&regs,&segregs)
# define VIDEO 0x10    /*      bios int 10, video function      */
/*
```

```
@Macro de um far pointer em TURBO C      */
```

```
#define MAKE_FP(ptr,seg,off) {ptr = MK_FP(seg,off);}
```

ÍNDICE REMISSIVO

80286

microprocessador INTEL 3

80386

microprocessador INTEL 3

8085

microprocessador INTEL 1, 53

8087

Co-processador matemático 40, 107

8088

Arquitetura 32-36

conjunto de instruções Assembly do 67-68,87-99,165-184

descrição dos pinos 31, 40-42

interrupções do 26-27, 53-55

microprocessador, uso geral 1, 3, 14-15,31-42

modos de endereçamento 37-40

sinais de status 16, 42, 49

8237

Controlador do DMA 24, 46

8253

configuração no PC 22-23, 135

programação 23, 134-138

temporizador programável, uso geral 1, 11, 22-23, 135-136

8255

configuração no PC 28-30, 134-135

Porta paralela, uso geral 1, 11, 28-30, 48-49

programação 134-137

8259

controlador de interrupção

uso geral 1, 26-27, 41, 56-60, 139

A

- Acesso Direto à memória
 - veja 8237 , Controlador do DMA
- ACHAR_B.C 5-6, 109, 116-119
- ACHAR_D.C 128, 130-133
- Acumulador
 - veja registradores
- Adaptador de vídeo monocromático
 - veja MDA
- Adaptador de vídeo
 - uso geral 5-6
- Adaptador gráfico colorido
 - veja CGA
- Alimentação
 - fonte de 4-5
 - slots 45-46
- Alto-falante 28-29, 134-136
- ALU 31, 65
- AND 62-65
- APPLE 1, 43, 69

B

- BASICA.COM 69
- BIOS
 - área de dados 6, 20-21, 52, 60, 107-108, 144
 - configuração dos dispositivos 103,107-109,192
 - controle da impressora 104, 203
 - controle da porta serial 104, 200-201
 - controle do relógio 105, 204-205
 - controle do teclado 104, 111-112,202
 - funções de disco 103, 193-199
 - funções de vídeo 101-102, 105-106, 186-191
 - interrupções 60, 100-119, 147, 185-205
 - memória RAM disponível 103, 192
- Bus
 - de dados 1, 14, 45
 - do sistema, uso geral 16, 43-45
 - sinais nos slots 44-45

C

- CABEÇA.H 70, 116-117,131,231-232
- CGA 5, 80-85, 100, 107, 185-192

- Clock 41, 47-48, 111, 140
 - no slot 44-45
- Código ASCII
 - de controle 84, 153-155
 - de vídeo 84-85, 155-157
 - estendido 84, 153, 158-159
 - uso geral, tabelas 6, 69, 84, 153-159
- COMMAND.COM 20-21, 68, 121, 145-146
- Conversão
 - decimal/binária 151-152
 - decimal/hexadecimal 151-152

D

- Debug
 - comando de controle de programas 72
 - comandos com o DOS 74
 - comandos com os registradores 72
 - comandos aritméticos 72
 - comandos de I/O 73
 - comandos de memória 73
 - uso geral 2, 27, 55, 61, 67-68, 71-74, 81, 87, 160-164
- Disco
 - flexível 5, 11
 - rígido 5, 11
- DMA
 - veja 8237
- DOS
 - área de dados 20-21
 - código de erros 227-230
 - funções 111, 120-133, 206-230

E

- EGA 5, 107, 185-192
- Endereçamento no 8088
 - veja 8088 Modos de Endereçamento
- EPROM
 - veja memória ROM

I

- I/O
 - leitura e escrita 51
 - mapeamento de endereços 51-52

IBM-PC

modelo AT 3, 11, 61, 100, 120

modelo PCJr. 3

modelo PS/2 3, 100, 120

modelo XT 1, 3, 11, 46, 61, 69, 77-78, 100, 120, 134, 185

IBMBIO.COM 19-20, 57, 100, 107, 129, 139

IBMDOS.COM 21, 129

Interrupção

Acknowledge 55

ciclo de uma 25, 53-55, 57

desvio de uma rotina de 126, 139-140, 149-150

Interrupções

do BIOS 8, 21, 28, 60, 100-101, 140, 185-205

do DOS 8, 21, 28, 60, 120-133, 144, 206-230

do sistema, geral 24-28

endereços das rotinas de 27, 55, 60

maskáveis 26, 41, 54-55

por hardware 26, 28, 45, 138-145, 147

por software 26, 53

vetores (Tabela de) 20, 22, 27, 55, 57-58, 109, 128-129, 139, 144, 214, 217

Interrupções

não maskáveis 26, 41

IRQ

veja Interrupções por hardware

M

MASM

Microsoft Macro Assembler 40, 67, 147

MDA 5, 80, 107, 185-192

Memória

de vídeo 80-83

escrita/leitura 49-50

RAM 3-4, 17, 19-21

ROM 3-4, 17-18, 77

Microsoft C 5.1 70

MS-DOS

veja Sistema Operacional DOS

N

- Nibble 153
- NMI
 - veja Interrupções não mascaráveis
- Notação Hexadecimal 151-153

O

- Offset 32
- OOP (Programação Orientada para o Objeto) 115
- Operandos
 - tipos de 88, 165-166

P

- PC-DOS
 - veja Sistema operacional DOS
- PCTOOLS 19
- Porta paralela impressora
 - conector 7
 - endereços 110
 - uso geral 7-8
- Programação
 - arquivos tipo *.COM 68, 71, 84-86, 149
 - arquivos tipo *.EXE 68
 - Assembly 8088
 - conjunto de instruções 61, 87-99, 165-184
 - uso geral 2, 61, 67-68, 115, 234
 - baseado em interrupções 139 código de máquina 61-67, 88
 - em tempo real 21, 60, 126, 134, 140
 - linguagem Basic 66, 69-70
 - linguagem C 6, 40, 61, 66, 70, 109, 115, 126, 130, 235
 - linguagem Pascal 6, 66, 70
 - linguagens de 61-70
 - linguagens de alto nível 69-70, 115
 - Orientada para o objeto
 - veja OOP
 - Programas compilados 69
 - sistema pooling 138

Q

- QuickBasic 70
- Quick C 70

R

Registradores

apontador da pilha 35, 74-75, 89

apontador de instruções 35, 75

de flag 34-36, 75, 166

segmento 34-35, 39-40, 75

de uso geral 34-35, 66, 74

RESET 39, 41

S

SideKick 21, 79, 126, 146

Sistema Operacional

CP/M 2, 68

DOS 2, 61, 71

Slot

pinos do 44

sinais de I/O 47-51

uso geral 3, 5, 43-52

T

Teclado

interrupção do 6, 60, 139

teclas especiais 6, 55, 60, 111-112, 126

uso geral 6, 28-29

TSR

programas que terminam e ficam residentes 20-21, 60, 121, 145-150

TTL 47, 64

Turbo

Assembler 67

C 147

Pascal 70

U

Unidade Lógica Aritmética

veja ALU

V

Versão DOS 3, 21, 57, 61, 120-122, 126, 132, 206, 217

Vídeo

modos de 186

BIBLIOGRAFIA

Os livros e revistas comentados foram utilizados como fonte de referência. O leitor poderá a eles recorrer para aprofundar algum tópico aqui resumido. Além de uma bibliografia básica que descreve o microprocessador e a arquitetura do PC, é necessário acompanhar em revistas as constantes atualizações de hardware e software.

Hardware

- SALVADOR, Rafael M.S. , "IBM PC-XT", Ed. Érica, 1988

Descreve todas as ligações da placa principal do IBM PC\XT, com uma breve introdução a respeito de microprocessador e memória.

- CUOCOLO, Raimondo. "Periféricos magnéticos para computadores", Ed. Érica, 1988

O controle e os mecanismos dos discos flexíveis e rígidos do PC é um assunto amplo e complexo. Este livro é para aqueles que pretendem se especializar neste assunto.

- EGGBRECHT,L.C. "Interfacing to the IBM Personal Computer" Ed. Howard W. Sams, 1983

Descreve em detalhes as características do slot. Sugere o diagrama de alguns projetos de interface. Limita-se aos aspectos de hardware.

- IBM Corp."Technical Reference: Personal Computer XT and Portable Personal Computer". Ed. IBM Corp., 1986.

Contém a versão atualizada (janeiro 1986) da listagem da BIOS e todos os detalhes técnicos das versões anteriores. Embora de leitura relativamente complexa, é a fonte primária de informação.

Um livro compacto com vários programas exemplos. Às vezes bastante técnico.

- MILLER, A. R. "Assembly IBM PC" Ed. Ebras, 1987

Enfatiza o desenvolvimento de programas em Assembly através de uma biblioteca de Macros. Necessita de alguma familiaridade com um montador Assembler.

- INTEL Corp. "The 8086 Family User's Manual", Ed. INTEL Corp. 1979

Uma documentação detalhada do hardware e código de instrução deste microprocessador. No tempo em que o PC só existia como projeto é interessante verificar as sugestões dos projetistas da INTEL para um virtual microcomputador.

SOFTWARE - Linguagem C

- MESQUITA, Thelmo J.M. "Linguagem C, Teoria & Programas", Ed. Érica, 1988.

Introdução à linguagem C usando o compilador AZTEC C86. São abordados os aspectos básicos da linguagem .

- SHILDT, Herbert "Linguagem C - Guia do usuário", Ed. McGraw-Hill, 1987.

Uma introdução à linguagem C em geral usando o compilador AZTEC - C86. Embora não seja específico, os programas exemplos adaptam-se bem ao PC.

- KERNIGHAN, Brian W., RITCHIE Dennis M. "C a linguagem de programação". Ed. Editora Campus, 1986.

Também conhecido como o padrão K & R em C. Escrito pelos pais da linguagem C é útil para os que dominam um compilador e querem aprender na fonte a filosofia da programação em C.

- JAMSA, Kris "Turbo C - Programmer's Library", Ed. Borland.Osborne/McGraw-Hill, 1988

Apresenta um grande conjunto de funções úteis a uma boa biblioteca em C. Escrito em módulos com boas ilustrações

- INTEL Corp. "Microprocessor and Peripheral Handbook. vol 1 Microprocessor, vol 2 Peripheral " Ed. INTEL Corp., 1987

Contém os "data-sheets" e "Application Notes" que a INTEL organiza para estimular o uso de seus componentes É uma fonte primária das características eletrônicas dos integrados que compõem o PC.

SOFTWARE/GERAL

- NORTON, Peter "Guia do Programador para IBM-PC", Ed. Campus, 1989.

Descreve as características dos vários tipos de PC. Contém informações detalhadas a respeito do sistema de disco, vídeo, ROM-BIOS e DOS. Infelizmente, os programas exemplos são resumidos.

- HOGAN, Thom "The Programmer's PC Sourcebook". Ed. Microsoft Press, 1988.

A mais extensa e profunda compilação atualizada de informações a respeito do PC. Muito útil para aqueles que já sabem o que querem procurar

- DUNCAN, Ray "Advanced MS-DOS", Ed. Microsoft Press, 1986

É a referência básica do DOS, com vários exemplos ilustrativos. O mesmo autor possui 2 guias de referência rápida ("IBM ROM-BIOS" e "MS-DOS Funções", em português, Ed. Campus, 1989) descrevendo e comentando todas as funções BIOS e DOS.

- Waite Group's (vários autores) "MS-DOS Developer's Guide" Ed. Howard W.Sams, 1989

É um livro profundo a respeito da filosofia do DOS e seus recursos. Programas exemplos em MacroAssembler e C ilustram a potencialidade do DOS. Destinado aos profissionais que precisam conhecer o DOS por dentro.

SOFTWARE/ASSEMBLY

- CARVALHO, J.A.D. e Rocha, R.F. "8086/8088 Vol. 1 - Software", Ed. Érica, 1989.

Descreve e comenta o código de instrução do 8088. Uma excelente introdução à linguagem Assembly do 8088, caso o leitor disponha de um montador (Assembler).

- QUADROS, D.G.A. "PC Assembler: usando o BIOS", Ed. Campus, 1987

- DAVIS, Stephen R., "Turbo C - The Art of Advanced Program Design, Optimization, and Debugging" Ed. M&T Books, 1988

Finalmente, um livro escrito em cima de um bom compilador (Turbo C) para rodar especificamente em um tipo de máquina (o PC) com excelentes programas documentados. Para os programadores que conhecem Pascal é uma ponte para a linguagem C.

REVISTAS

- PC-Resource (custo aproximado U\$ 3,00). Assinatura P.O.Box 58742, Boulder Co, 80321-0302 USA.

Excelente nas dicas de manutenção do PC e de configuração das placas de hardware. Embora as dicas em torno do DOS sejam para principiantes, às vezes apresenta bons programas em C ou em Quickbasic (ou Turbobasic).

- Micro-Cornucopia (The Micro Technical Journal) , (preço aproximado U\$4,00), bimensal, Assinatura P.O. Box 223, BEND, OR 97709 USA.

Revista escrita por um pessoal bem humorado com excelente dicas para o iniciante em projetos baseados no PC. Embora o C seja a linguagem padrão, programas em C++ e Assembly são frequentes.

- Dr. Dobbs Journal (Software Tools for The Professional Programmer), (preço aproximado U\$3,00), Assinatura P.O. Box 3771, Escondido, CA 92025 USA.

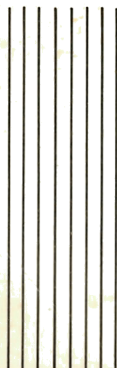
A revista dos programadores profissionais. Acompanha todas as evoluções dos compiladores e das mais recentes inovações de hardware. Os números antigos são uma valiosa fonte para os futuros programadores profissionais em C. Procurar entender os programas nela listados é uma tarefa não trivial, porém o aprendizado vale a pena o sacrifício.

Não é possível separar hardware e software quando se pretende interfacear o PC com o mundo real. Atendendo a este desafio desenvolvemos este livro de modo que o engenheiro, o programador, o técnico em eletrônica e o hobbista possam encontrar em um único volume, as informações básicas do projeto PC, tais como:

- Descrição da arquitetura do IBM-PC
- Código de instruções do microprocessador 8088
- Descrição detalhada do slot de interfaceamento e das interrupções por hardware
- Uso do ROM-BIOS e DOS, nas linguagens Assembly e C
- Exemplos de programas residentes e que manipulam interrupção.

Para atravessar esta selva de informações, mostramos vários programas exemplos que podem ser montados usando somente os recursos do DOS, prescindindo de conhecimentos prévios de eletrônica ou computação.

Este livro poderá ser um meio para transformar um usuário em um programador, capaz de explorar ao máximo os recursos do PC

The logo for ERICA, featuring the word in a stylized, blocky font where each letter is composed of horizontal lines.

LIVROS DE CONTEÚDO, COM QUALIDADE
RUA JARINÚ, 594 - TATUAPÉ - CEP 03306
CX. P. 15617 - TEL.: (011) 294-8686 - S.P.

ISBN 85-7194-035-5