

TELECOMUNICAÇÕES

# MICROPROCESSADORES E MICROCOMPUTADORES



TELECOMUNICAÇÕES

# MICROPROCESSADORES E MICROCOMPUTADORES



# SUMÁRIO

APRESENTAÇÃO .....	04
BOAS VINDAS .....	05
INFORMAÇÕES INTRODUTÓRIAS .....	06
Organização curricular .....	06
Sistema de tutoria .....	06
Sistema de avaliação .....	06
VOCÊ E OS ESTUDOS À DISTÂNCIA .....	07
Organizando os estudos .....	07
Conhecendo o ambiente virtual de aprendizagem .....	08
MICROPROCESSADORES .....	09
ANÁLISE DE UM MICROPROCESSADOR GENÉRICO .....	19
SISTEMA DE MEMÓRIA DE COMPUTADORES .....	29
ARQUITETURA DE SOFTWARE DE UM MICROPROCESSADOR ...	46
PROGRAMAÇÃO EM LINGUAGEM ASSEMBLY .....	66
INTERRUPÇÕES E EXCEÇÕES .....	100
ANEXO A - TABELA ASCII .....	113
ANEXO B - CÓDIGO ESTENDIDO DO TECLADO .....	114
ANEXO C - INTERRUPÇÕES BIOS E DOS .....	116
SISTEMA DE COMPUTAÇÃO E HARDWARE .....	145

# APRESENTAÇÃO

A **Escola Técnica Nossa Senhora Aparecida** com o intuito de se tornar referência em ensino técnico no Brasil, lança cursos técnicos em diversos eixos, de forma a atender uma demanda regional e estadual.

Por meio de um trabalho diferenciado o estudante é instigado ao seu autodesenvolvimento, aliando a pesquisa e a prática.

Essa competência e boa formação são os requisitos necessários para quem deseja estar preparado para enfrentar os desafios do mercado profissional. A escolha de um curso que aproxime teoria e prática e permita a realização de experiências contribui de maneira decisiva para a formação de um profissional comprometido com a qualidade e a inovação.

Ciente dessa importância a escola técnica Nossa Senhora Aparecida reuniu profissionais especialistas das áreas fins dos cursos propostos para fornecer cursos técnicos de qualidade para a comunidade da região.

Como escola de desenvolvimento tecnológico, na área de educação, através de um trabalho sério, realizado nos últimos anos no campo da educação básica, fortalece e amplia o seu programa de cursos, instituindo, em Goiás cursos técnicos de educação profissional.

Os cursos da Escola Técnica Nossa Senhora Aparecida são oferecidos na modalidade semipresencial, utilizando-se da plataforma Moodle ou Material Apostilado, mediado por professores formadores/tutores renomados. Além dos momentos presenciais, serão oferecidos no ambiente virtual: fórum de apresentação, fórum de notícias, slide com conteúdos pertinentes ao curso em questão, links de reportagens direcionadas, sistematização da aprendizagem.

# BOAS VINDAS

Bem vindo à Escola Técnica Nossa Senhora Aparecida!

Prezado (a) Cursista,

Que bom tê-lo (a) conosco!

Ao ter escolhido estudar na modalidade à distância, por meio de um ambiente virtual de aprendizagem, você optou por uma forma de aprender que requer habilidades e competências específicas por parte dos professores e estudantes. Em nossos cursos à distância, é você quem organiza a forma e o tempo de seus estudos, ou seja, é você o agente da sua aprendizagem. Estudar e aprender a distância exigirá disciplina.

Recomendamos que antes de acessar o espaço virtual de aprendizagem, faça uma leitura cuidadosa de todas as orientações para realização das atividades.

É importante que, ao iniciar o curso, você tenha uma compreensão clara de como será estruturada sua aprendizagem.

Uma orientação importante é que você crie uma conta de e-mail específica para receber informações do curso, seus exercícios corrigidos, comunicados e avisos. É de responsabilidade do estudante verificar também sua caixa de spam-lixo para ter acesso a todas as informações enviadas.

Desejamos um ótimo curso.

# INFORMAÇÕES INTRODUTÓRIAS

## **ORGANIZAÇÃO CURRICULAR**

Cada curso possui matriz curricular própria dividida em módulos de ensino semanais. O cronograma e planejamento de cada curso são modulados conforme as disposições dos professores e as atualizações dos conteúdos.

Os cursos têm apostilas de conteúdo para cada componente curricular, elaboradas por profissionais de referência em Goiás.

Os certificados serão emitidos pela Escola Nossa Senhora Aparecida até 90 dias após o término do curso, tendo em vista o trabalho de fechamento das notas e avaliação do curso.

## **SISTEMA DE TUTORIA**

O tutor será o profissional que estará mais próximo de você durante o período do curso, passando todos os comunicados e avisos, cobrando a entrega das atividades.

Conte com o tutor da sua turma para tirar suas dúvidas sejam elas do ambiente virtual, conteúdo do curso ou dúvida e questionamentos sobre os exercícios.

## **SISTEMA DE AVALIAÇÃO**

A avaliação será obtida através da participação e da avaliação do nível de conhecimento que o estudante demonstrar em chats, fóruns e exercícios.

Ao término do curso será informado para os estudantes de forma individualizada sobre sua aprovação e desempenho no curso.

# VOCÊ E OS ESTUDOS À DISTÂNCIA

## ORGANIZANDO OS ESTUDOS

O estudo por meio de um ambiente virtual de aprendizagem não é mais difícil e nem mais fácil do que num ambiente presencial. É apenas diferente. O estudo à distância exige muita disciplina. As orientações a seguir irão auxiliá-lo a criar hábitos de estudo.

- Elabore um horário semanal, considerando a carga horária do curso. Nesse plano, você deve prever o tempo a ser dedicado:
  - à leitura do conteúdo das aulas, incluindo seus links para leituras complementares, sites externos, glossário e referências bibliográficas;
  - à realização das atividades ao final de cada semana;
  - à participação nos chats;
  - à participação nos fóruns de discussão;
  - ao processo de interação com o professor e/ou com o tutor;
  - ao processo de interação com seus colegas de curso, por mensagem ou por chat.

Uma vez iniciados os seus estudos, faça o possível para manter um ritmo constante, procurando seguir o plano previamente elaborado. Na educação à distância, é você, que deve gerenciar o seu processo de aprendizagem.

Procure manter uma comunicação constante com seu tutor, com o intuito de tirar dúvidas sobre o conteúdo e/ou curso e trocar informações, experiências e outras questões pertinentes.

Explore ao máximo as ferramentas de comunicação disponíveis (mensageiro, fórum de discussão, chat).

É imprescindível sua participação nas atividades presenciais obrigatórias (aulas), elas são parte obrigatória para finalização do curso.

## **CONHECENDO O AMBIENTE VIRTUAL DE APRENDIZAGEM**

No ambiente virtual de aprendizagem também necessitamos de uma organização para que ocorram os processos de ensino, de aprendizagem e principalmente a interação entre professor/tutor e estudantes.

O ambiente virtual de aprendizagem da Escola Nossa Senhora Aparecida é o Moodle.

Em sua sala de aula, você encontrará espaços de comunicação e interação: quadro de notícias, atividades recentes, informações sobre o professor e sobre seus colegas de turma, calendário, recurso para o envio da sistematização ao seu tutor e ferramentas de comunicação.

Sucesso no seu Curso!



# MICROPROCESSADORES

## ARQUITETURA DE COMPUTADORES

Embora tenham ocorrido revolucionárias transformações na área de Eletrônica, os microcomputadores de hoje ainda mantém a mesma concepção funcional dos primeiros computadores eletrônicos. Tal concepção, conhecida como Arquitetura de Von Neumann, é definida da seguinte forma:

Uma unidade central de processamento recebe informações através de uma unidade de entrada de dados, processa estas informações segundo as especificações de um programa armazenado em uma unidade de memória, e devolve os resultados através de uma unidade de saída de dados.

A Figura 1 mostra, por diagrama de blocos, a organização de um sistema com estas características. O sistema de computador envolve, como pode ser visto, o computador e os elementos geradores e receptores de informações.

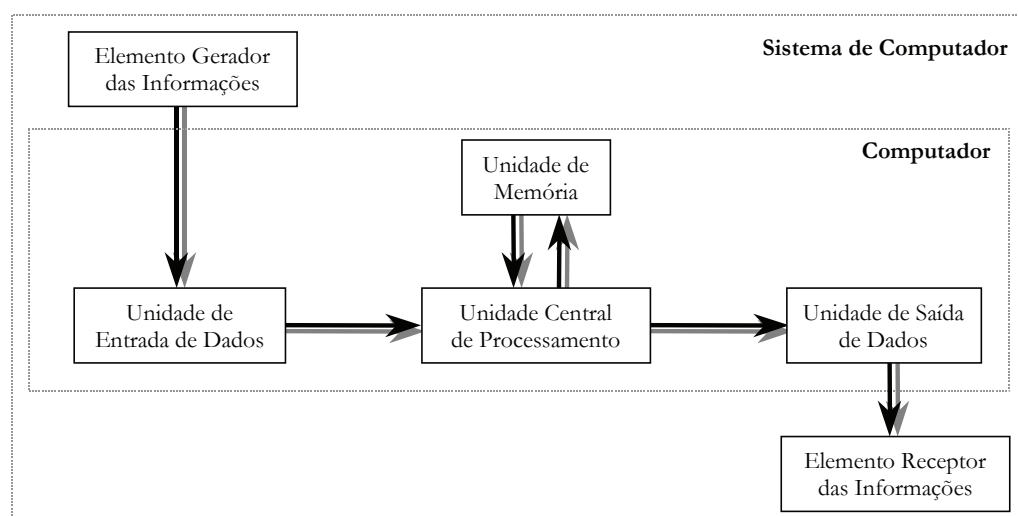


Figura 1 Organização de um sistema de computador

## Unidade Central de Processamento (CPU)

A unidade gestora do computador capaz de administrar todas as operações de leitura/escrita da memória ou de uma unidade de entrada/saída de dados, de executar operações aritméticas ou lógicas e de interpretar todas as instruções recebidas de um programa que está em execução.

## Microprocessador

Dispositivo LSI (large scale integration – alto grau de integração) que condensa em um único chip a maioria das funções associadas a uma unidade central de processamento.

## Microcomputador ou sistema a microprocessador

Computador que se caracteriza por apresentar blocos lógicos de CPU, memória e E/S bem definidos e onde todas as funções de processamento da unidade central são desempenhadas por um processador (Figura 2).

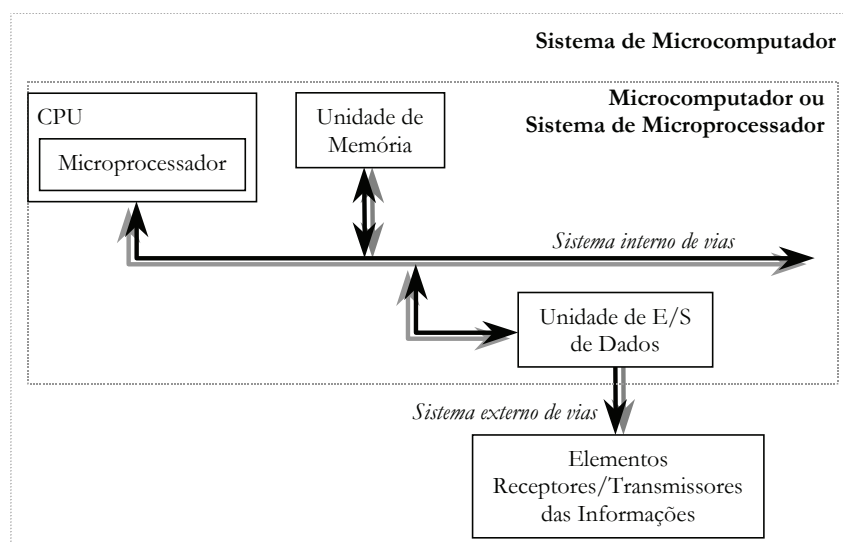


Figura 2 Sistema de microcomputador

## Periférico

Qualquer elemento gerador ou receptor de informação em sistemas

de computadores (Figura 3).

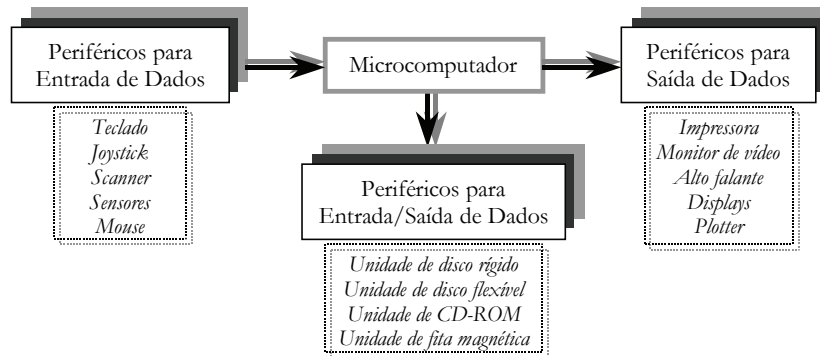


Figura 3 Periféricos de um microcomputador

## Unidades de Entrada/Saída

Blocos internos responsáveis pelas transferências de dados entre o microcomputador e qualquer dispositivo periférico. É através de uma unidade de entrada de dados que as informações de periférico de entrada são levadas à CPU ou à memória. De forma similar, é através de uma porta de saída de dados que as informações são levadas da CPU ou da memória para um periférico de saída.

## HARDWARE E SOFTWARE

Hardware é o conjunto de dispositivos elétricos/eletrônicos que englobam a CPU, a memória e os dispositivos de entrada/saída de um sistema de computador. O hardware é composto de objetos tangíveis (circuitos integrados, placas de circuito impresso, cabos, fontes de alimentação, memórias, impressoras, terminais de vídeo e teclados).

O software, ao contrário, consiste em algoritmos (instruções detalhadas que dizem como fazer algo) e suas representações para o computador ou seja, os programas.

Qualquer instrução efetuada pelo software pode ser implementada diretamente em hardware e qualquer operação executada pelo

hardware pode também ser simulada pelo software.

A decisão de se colocar certas funções em hardware e outras em software se baseia em fatores, tais como: custo, velocidade, confiabilidade e possibilidade/facilidade de modificação.

## ARQUITETURA BÁSICA DE MICROPROCESSADORES

Para compreender bem a arquitetura básica de um microprocessador (Figura 4) basta associar as operações que caracterizam uma unidade central de processamento com os elementos funcionais que permitem a sua realização.

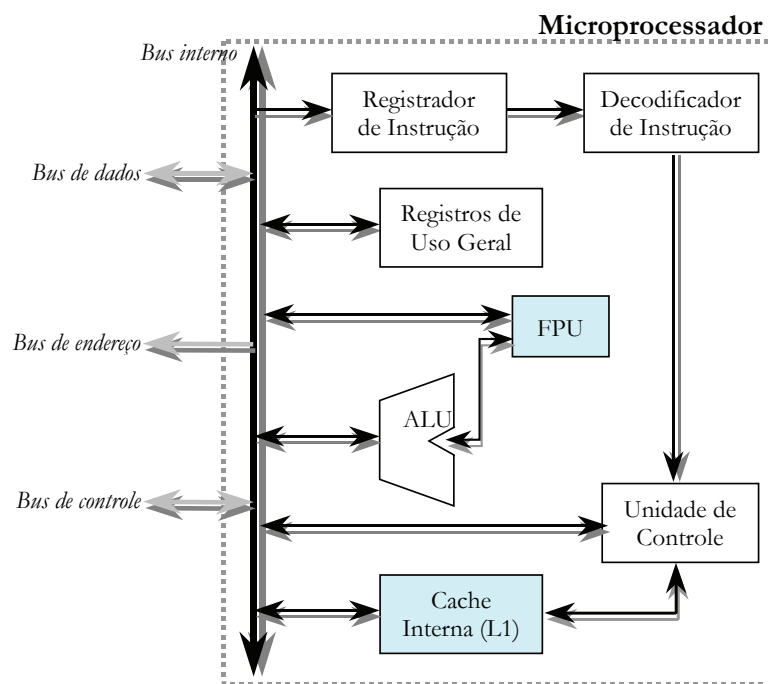


Figura 4 Arquitetura básica de um microprocessador com FPU e cache interna

- Para administrar operações de leitura/escrita da memória ou de uma E/S de dados são necessárias:
  - uma unidade de controle, que oriente a busca ou o envio das informações;
  - uma pequena capacidade de memória onde estas

- informações possam ficar temporariamente armazenadas (registrador de uso geral); e
- alguns barramentos (ou vias) onde possam ser manipulados os dados, os endereços e os sinais de controle.
- Para executar operações aritméticas e lógicas é necessária a presença de uma ALU (arithmetic and logic unit – unidade lógica/aritmética) e, nos processadores mais avançados, uma FPU (float point unit – unidade de ponto flutuante) para operações com números reais. A inclusão da FPU veio permitir a execução de operações antes só possíveis com o auxílio de um coprocessador aritmético (operações em ponto flutuante).
- Para interpretar as instruções estabelecidas por um programa devem existir:
  - um decodificador de instrução (microcódigo) e;
  - um registrador de instrução, no qual a instrução recebida fica temporariamente armazenada.

## **Barramentos**

Um barramento ou via ou bus é um conjunto de pinos do microprocessador por onde trafegam um ou mais sinais de hardware. Um microprocessador possui três tipos de barramentos utilizados para transporte: bus de dados, bus de endereços e bus de controle.

**Barramento de dados:** Barramento bidirecional, utilizado para realizar o intercâmbio de dados e instruções com o exterior. Uma das principais características de um microprocessador é o número de bits que o barramento de dados pode transferir, que determina se o processador é de 8, 16, 32 ou 64 bits. Determina o número de bits da palavra de dados que pode ser transferida de/para o microprocessador e, também (quase sempre) o tamanho da palavra de dados que pode ser operada pela ALU.

**Barramento de endereços:** Barramento unidirecional, constituído de um conjunto de linhas de endereço que indicam a posição de memória onde se encontra o dado requisitado. Uma vez dada a posição, a informação armazenada na memória passará à CPU através do barramento de dados. Define a quantidade de posições de memória e/ou de portas de entrada/saída que podem ser acessadas pelo microprocessador (para  $n$  bits do barramento de endereços,  $2^n$  bytes de memória podem ser endereçados, ou seja,  $2^n$  endereços físicos podem ser acessados – capacidade de endereçamento).

**Barramento de controle:** Barramento bidirecional, formado por um número variável de linhas, através das quais se controlam as unidades complementares (habilitação e desabilitação das memórias para leitura e escrita, permissão para periféricos ou coprocessadores acessarem as vias de dados e endereços). Transfere, para as diversas partes do sistema, sinais que definem e orientam toda a sua operação. Sinais de controle típicos de um microprocessador são:

- leia de uma posição de memória (memory read);
- leia de uma porta de E/S (I/O read);
- escreva em uma posição de memória (memory write);
- escreva em uma porta de E/S (I/O write);
- pedido de interrupção de programa (interruption request);
- pedido de uso de vias (bus request ou hold request);
- pedido de espera (wait ou ready);
- sinal de relógio (clock); e
- sinal de partida/reinício (reset).

## EVOLUÇÃO DOS MICROPROCESSADORES

Microprocessador	Bits Internos	Bits do bus de dados	Bits do bus de endereços	Cache L1	Observações
1a Geração					
Intel 4004	4	4	4	–	Primeiro microprocessador (1971)
Intel 8008	8	8	8	–	Calculadoras ou sistemas de controle dedicados
Intel 8080/8085	8	8	16	–	Arquitetura escalar com estrutura seqüencial; Outros fabricantes: Motorola (6800) e Zilog (Z80)

2a Geração					
Intel 8086/8088	16/8	16/8	20	–	IBM PC/XT; Arquitetura escalar e estrutura pipelined; Outros fabricantes: Zilog (Z8000) e Motorola (68000)
Intel 80286	16	16	24	–	IBM PC/AT 286; Arquitetura escalar e estrutura pipelined; Outro fabricante: Motorola (68010)
3a Geração					
Intel 386DX	32	32	32	–	Primeira CPU de 32 bits a incluir gerenciamento de memória; Ambiente multi-usuário; Arquitetura escalar e estrutura pipelined melhorada; Possibilidade de cache L2; Outro fabricante: Motorola (68020/68030)
Intel 386SX	32	16	24	–	Idêntico ao 386DX, exceto pelos barramentos
4a Geração					
Intel 486DX	32	32	32	8KB	Arquitetura escalar e estrutura pipelined otimizada; Possui FPU
Intel 486DX2	32	32	32	8KB	Possui FPU; Utiliza duplicação do clock
Intel 486DX4	32	32	32	16KB	Possui FPU; Utiliza triplicação do clock
Intel 486SX	32	32	32	8KB	Não possui FPU (coprocessador interno)
Cyrix 486DLC	32	32	32	1KB	Semelhante ao Cyrix 486SLC, com 32 bits
AMD 5x86	32	32	32	16KB	Semelhante a um 486 de 133MHz, com desempenho de um Pentium-75
Cyrix 5x86 (M I)	32	32	32	16KB	Possui FPU; Características do Pentium e pinagem do 486DX4
5a Geração					
Intel Pentium	32	64	32	16KB	Cache L1 = 8KB instruções + 8KB dados; Projeto híbrido CISC/RISC
AMD K5	32	64	32	24KB	Semelhante ao Pentium; Projeto híbrido CISC/RISC; Cache L1 = 16KB instruções + 8KB dados
Cyrix 6x86 (M II)	32	64	32	64KB	Características do Pentium Pro e pinagem do Pentium; FPU (64 bits); Tecnologia MMX; Arquitetura superescalar; Execução dinâmica
6a Geração					
Intel Pentium Pro	32	64	32	16KB	Execução dinâmica; Tecnologia DIB; Cache L2 de 256 e 512KB
Intel Pentium MMX	32	64	32	16KB	Pentium com tecnologia MMX; Esquema duplo de tensão nos circuitos interno (core) e externo
AMD K6	32	64	32	64KB	Tecnologia MMX; Arquitetura superescalar; FPU (64 bits); Cache L1 = 32KB instruções + 32KB dados
AMD K6-2 (K6 3D)	32	64	32	64KB	Semelhante ao K6. Mais velocidade (barramento externo de 100MHz)

AMD K6-3 (K6+)	32	64	32	64KB	Semelhante ao K6-2. Cache L2 (256KB) integrado ao processador. Cache L3 na placa-mãe.
Intel Pentium Celeron	32	64	32	32KB	Cache L1 = 16KB instruções + 16KB dados; Cache L2 de 128KB integrada; Tecnologia MMX; FPU (32 e 64 bits); Execução dinâmica; Arquitetura superescalar
Intel Pentium II	32	64	32	32KB	Cache L1 = 16KB instruções + 16KB dados; Tecnologia DIB; Tecnologia MMX; Execução dinâmica; Cache L2 de 512KB; Suporte à memória expandida de 36 bits (endereçamento de memória > 4GB)
Intel Pentium II Xeon	32	64	32	32KB	Cache L2 de 1MB ou 2MB; Pentium II projetado para servidores e estações de trabalho
Intel Pentium III	32	64	32	32KB	Cache L2 de 512KB; FPU (32, 64 e 80 bits); Tecnologia MMX; Tecnologia DIB; Execução dinâmica; Suporte à memória expandida de 36 bits; Internet Streaming SIMD Extensions; Intel Processor Serial Number
Intel Pentium III Xeon	32	64	32	32KB	Cache L2 de 512KB, 1MB ou 2MB; Pentium III projetado para servidores e estações de trabalho

Durante muito tempo uma disputa desleal vinha sendo travada entre empresas dedicadas à produção e comercialização de microprocessadores: de um lado a poderosa Intel, líder de mercado, dominadora de avançadas tecnologias e manipuladora de técnicas e estratégias que não deixavam folga para concorrência; de outro lado, todas as outras empresas que sempre tiveram de se contentar em receber migalhas da fatia do mercado restante, sempre com produtos de tecnologia licenciada e atrasados em relação aos produtos da Intel. Entretanto, esta situação começa a mudar. Competidores como AMD, Cyrix, NexGen, Sun Microsystems e algumas outras empresas associadas começam a introduzir no mercado novos processadores com significativas diferenças dos integrados Intel. Estes projetos originais e altamente otimizados prometem melhor desempenho do que os Pentiums da Intel, ao mesmo tempo mantendo total compatibilidade com os softwares DOS e Windows. O Pentium marca uma bifurcação entre a era exclusiva da Intel e uma nova era, quando diferentes tecnologias estão surgindo, com o desenvolvimento de novas microarquiteturas, capazes, dizem os fabricantes, de superar



o desempenho do Pentium.

Percebe-se, com a evolução tecnológica e a redução nos custos de desenvolvimento dos componentes eletrônicos, que:

- as máquinas CISC (complex instruction set computer) aos poucos absorvem características típicas de RISC (reduced instruction set computer), como estrutura superescalar e cache otimizados de memória interna; e
- as máquinas RISC, em contrapartida, aos poucos vão barateando seus custos, mantendo excelente rapidez na execução de programas, e passando a competir de igual para igual com as máquinas CISC de última geração.

Dentre as novas tecnologias adotadas, podemos comentar:

- MMX (multimedia extensions): Tecnologia projetada para acelerar aplicações de multimídia e comunicação, adicionando novos conjuntos de instruções e diferentes tipos de dados, além de explorar o paralelismo (SIMD – single instruction multiple data).
- DIB (dual independent bus): Arquitetura composta pelo barramento de cache L2 e o barramento de sistema (entre o processador e a memória principal), ambos podendo ser usados simultaneamente.
- Cache L2 acoplada (integrada) ao processador dentro de um único encapsulamento.
- Execução dinâmica (Microarquitetura P6): Previsão de múltiplos desvios nos programas em execução, acelerando o fluxo de trabalho do processador; Análise de fluxo de dados, reordenando o escalonamento das instruções; e Execução especulativa, antecipando as instruções do programa e executando as que provavelmente serão necessárias.
- Internet Streaming SIMD Extensions: Conjunto de novas instruções, incluindo SIMD para ponto flutuante e instruções SIMD adicionais e de controle de cache.
- Intel Processor Serial Number: número de série eletrônico do processador que permite sua identificação por redes e aplicações.

- Tensão dupla de operação Intel: 2,0 ou 2,8V para os circuitos internos (core) e 3,3V para os circuitos que fazem ligação externa.
- Encapsulamento S.E.C. (single edge contact): integra todas as tecnologias de alto desempenho do processador.

# ANÁLISE DE UM MICROPROCESSADOR GENÉRICO

Por questões didáticas, será feita, nesse capítulo, a análise de um processador genérico com muitas características comuns aos microprocessadores da família 80x86 da Intel.

Para trabalhar com um novo microprocessador, o programador ou projetista deve conhecer detalhes sobre: a arquitetura interna; a função dos pinos e o timing (temporização) durante os ciclos de barramento; e a estrutura de software.

Pela análise da arquitetura interna será possível saber quais são e como estão organizados os seus registradores, qual o número de bits nos barramentos de dados, de controle e de endereços, quais as características da ALU, e quais as suas lógicas de decodificação e de controle.

Conhecendo as funções dos pinos e o timing durante os ciclos de barramento, será possível ao usuário definir, com exatidão, dentre outras coisas, qual a relação pino/tarefa, como e quando uma dada ação do processador se realizará ou, até mesmo, se determinado pedido de serviço será (e quando) ou não atendido pelo processador.

Conhecendo a estrutura de software de será possível desenvolver programas sem ter que entrar em detalhes de implementação do chip.

Características do processador genérico proposto para análise:

- alimentação por fonte única, baixa potência de dissipação, saídas bufferizadas;
- vias independentes para dados, endereço e controle;
- capacidade de endereçamento de 1 Mbyte de memória e 64 kbytes de portas de E/S;
- grupo de aproximadamente 150 instruções;
- instruções diferenciadas para acesso a memória ou a E/S;

- lógica interna para controle de pedidos de interrupção de programa e de DMA;
- lógica de controle para operação com coprocessador aritmético; e
- arquitetura escalar pipelined implementada por 4 unidades funcionais básicas: unidades de barramentos, decodificação, execução e geração de endereços (UB, UI, UE e UA).

## **ARQUITETURA INTERNA**

### **Unidades Funcionais**

A arquitetura interna do microprocessador é caracterizada pela existência de quatro unidades funcionais básicas: UB, UI, UE e UA.

A unidade de interfaceamento de barramentos (UB), através das interfaces de controle de vias, da interface do bus de endereços e da interface do bus de dados, manipula todos os sinais de controle, endereços e dados necessários para que ocorram os acessos à memória e E/S requeridos pela CPU. Ela também é responsável pelo tratamento dos sinais necessários à interface de unidades coprocessadoras e outros barramentos mestres. Além disso, incorpora uma pequena memória de 8 bytes: a fila de pré-busca (prefetch), onde são armazenados os últimos dados buscados da memória ou de uma E/S. A UB opera tipicamente por looking ahead, buscando instruções na memória e colocando-as na fila de pré-busca.

A unidade de decodificação de instruções (UI) recebe possíveis instruções da fila de pré-busca, decodifica-as (através do decodificador de instruções), e gera uma nova fila de instruções pré-decodificadas para a unidade de execução.

A unidade de execução (UE) executa as instruções pré-decodificadas pela UI. Caso a instrução exija acessos a memória ou a alguma E/S, as pré-buscas são suspensas e a instrução em curso na UE terá acesso aos barramentos, através da UB. À UE estão relacionados

todos os registradores de uso geral e de controle da ALU.

A unidade de geração de endereços (UA) é responsável pela geração dos endereços físicos necessários à operação da CPU.

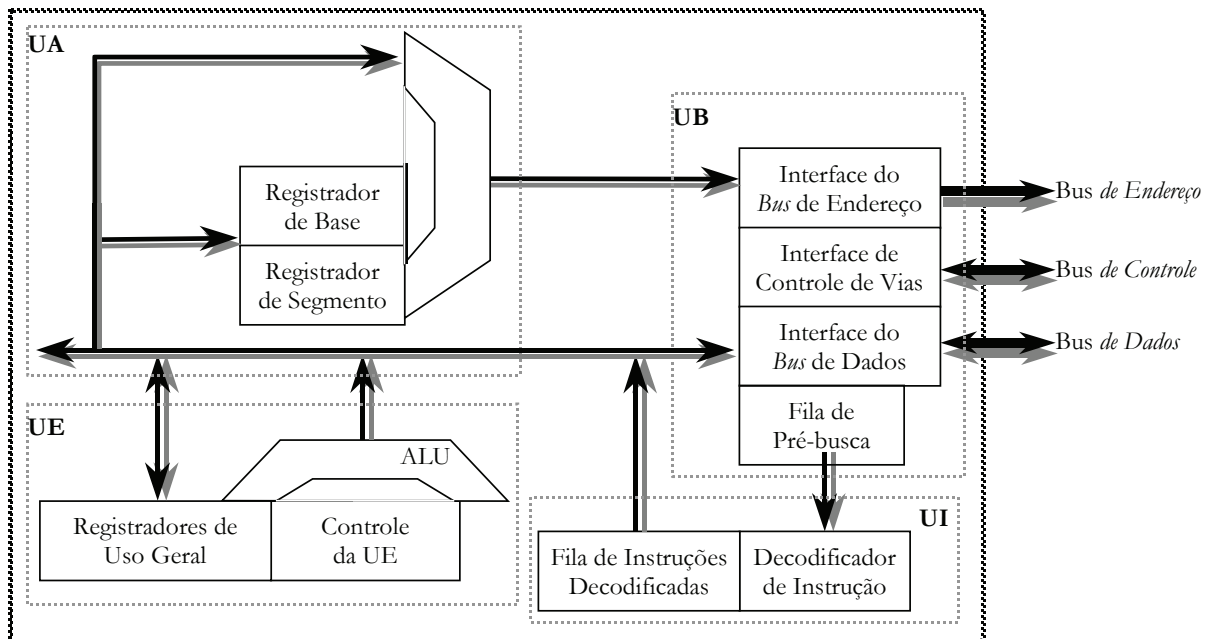


Figura 5 Arquitetura interna do processador

## Estrutura Pipelined

Os processadores da 1ª geração possuíam uma arquitetura escalar com estrutura seqüencial. Dessa forma, a quantidade de ciclos necessários para executar um conjunto de instruções era sempre igual à quantidade total de ciclos alocados para as unidades, uma vez que apenas uma das unidades poderia estar sendo utilizada a cada ciclo.

A partir da 2ª geração, uma estrutura pipelined começou a ser adotada pelos processadores. Nesse caso, em um determinado ciclo, pode haver mais de uma unidade sendo utilizada, dependendo da alocação de recursos (unidades) das instruções a serem executadas.

A execução de uma instrução pode ser distribuída nas seguintes fases lógicas para um pipeline de instruções (Figura 6):

- Determinação do endereço da instrução e busca na memória

[IF = instruction fetch]

- Decodificação da instrução a ser executada [ID = instruction decode]
- Geração dos endereços e busca dos operandos [OF = operand fetch]
- Execução da instrução [EX = execution]
- Armazenamento do resultado [WB = write back]

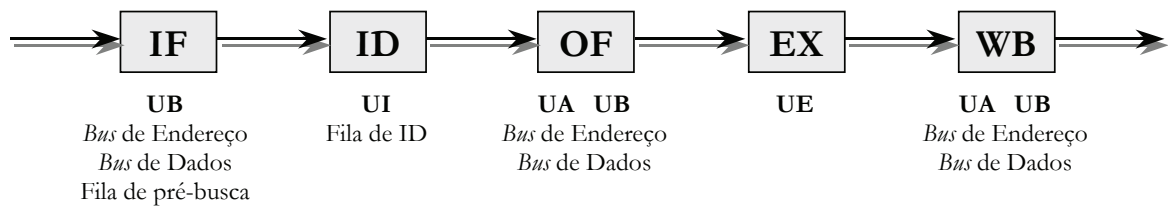


Figura 6 - Fases lógicas e recursos físicos necessários à execução de instruções em um pipeline

Após a execução das instruções é possível determinar a quantidade total de ciclos gastos e calcular a taxa média de ciclos por instrução (CPI), dada por:  $CPI = \frac{\text{Total de Ciclos}}{\text{N}^\circ \text{ de Instruções}}$ . O CPI significa a quantidade média de ciclos necessários para executar uma instrução.

Uma outra medida que pode ser determinada é o ganho da estrutura pipelined sobre a estrutura seqüencial. O ganho, também conhecido como Speedup, é dado por:  $Speedup = \frac{\text{CPI ou ciclos seqüencial}}{\text{CPI ou ciclos pipelined}}$  ou  $S(n) = \frac{T(1)}{T(n)}$ .  $S(n) \geq 1$ , significa que o desempenho da estrutura pipelined é melhor que ou igual ao da seqüencial; caso contrário, teremos  $0 < S(n) < 1$ .

Exemplo: Dado o seguinte conjunto de instruções e os recursos necessários para sua execução, calcule o CPI, considerando que as filas presentes nas unidades comportam apenas uma instrução.

- I1 = {UB, UI, UE, UA, UB}
- I2 = {UB, UI, UE, UE}
- I3 = {UB, UI, UE}
- I4 = {UB, UI, UA, UB, UE, UA, UB}
- I5 = {UB, UI, UE, UA, UB}
- I6 = {UB, UI, UE}

A Figura 7 ilustra como instruções são executadas na arquitetura escalar com estrutura pipelined do processador. Observe que, no exemplo, as execuções das instruções 1, 4 e 5 exigem acesso aos barramentos externos, enquanto as execuções das demais instruções são feitas internamente. Um dado importante para a execução no pipeline é a capacidade das filas presentes na UI e na UB (de instruções decodificadas e de pré-busca), que determina a quantidade máxima de instruções que poderão estar presentes nessas unidades. Uma observação deve ser feita quanto à UB: a fila de pré-busca armazena instruções a serem decodificadas, não impedindo que outra instrução esteja acessando barramentos para leitura ou escrita de dados na memória. Portanto, a exigência na UB é que apenas uma instrução pode acessar os barramentos em um determinado ciclo. A existência de uma unidade ociosa configura uma bolha (x) no pipeline.

UB	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>1</sub> I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>4</sub>	x	I <sub>5</sub>	I <sub>4</sub>
UI	x	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>6</sub>	x	x
UE	x	x	I <sub>1</sub>	I <sub>2</sub>	I <sub>2</sub>	I <sub>3</sub>	x	I <sub>5</sub>	I <sub>4</sub>	I <sub>6</sub>	x
UA	x	x	x	I <sub>1</sub>	x	x	I <sub>4</sub>	x	I <sub>5</sub>	I <sub>4</sub>	x
Ciclos	1	2	3	4	5	6	7	8	9	10	11

Figura 7 - Execução das instruções do exemplo numa estrutura pipeline

Cálculos para o Exemplo:

Estrutura pipelined:

Total de Ciclos = 11

Nº de Instruções = 6

$CPI = \frac{11}{6} = 1,83 \text{ ciclos/instrução}$

Estrutura seqüencial:

Total de Ciclos = 5+4+3+7+5+3 = 27

Nº de Instruções = 6

$CPI = \frac{27}{6} = 4,5 \text{ ciclos/instrução}$

Ganho da estrutura pipelined sobre a seqüencial:  $S(n) = \frac{27}{11} \text{ ou } \frac{4,5}{1,83} = 2,45$

## FUNÇÃO DOS PINOS

\Pela Figura 8, o processador apresenta: um barramento de dados de 16 bits; um barramento de endereços de 20 bits; e um barramento de controle. O barramento de controle, por sua vez, encontra-se dividido nos grupos de:

- controle de memória e I/O;
- controle de interrupção;
- controle de DMA (direct memory access – acesso direto à memória); e
- controle de operação com coprocessador (somente em alguns processadores).

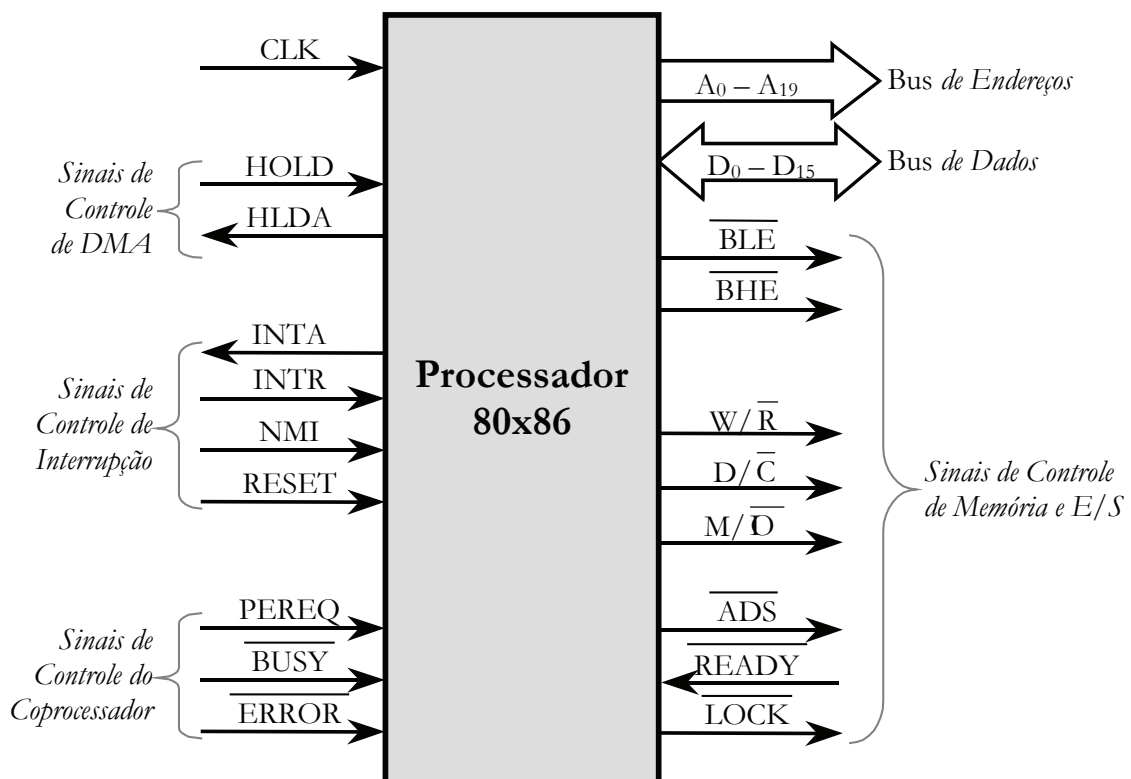


Figura 8 Função dos pinos do processador genérico

A Tabela 1 lista cada um destes sinais, especificando o nome do pino,



sua função, tipo e nível ativo.

Nome	Função	Tipo	Nível Lógico
CLK	Clock do sistema	E	–
A0-A19	Bus de endereço (20 bits)	S	1
D0-D15	Bus de dados (16 bits)	E/S	1
BHE, BLE	Byte enables - Selecionam parte alta (BHE) e baixa (BLE) do bus de dados	S	0
HOLD	Pedido de uso do bus de dados (pedido de DMA)	E	1
HLDA	Reconhecimento HOLD (reconhecimento de DMA)	S	1
INTR	Pedido de interrupção mascarável	E	1
NMI	Pedido de interrupção não-mascarável	E	1
RESET	Reinicialização do sistema	E	1
PEREQ	Requisição de dados pelo coprocessador	E	1
BUSY	Coprocessador ocupado	E	0
ERROR	Erro no coprocessador	E	0
W/R	Referente a escrita (W) ou leitura (R)	S	1/0
D/C	Referente a dados (D) ou controle/código (C)	S	1/0
M/IO	Referente a memória (M) ou E/S (IO)	S	1/0
ADS	Status de endereço (indicador de ciclo de barramento válido)	S	0
READY	Pedido de espera (por um periférico)	E	0
LOCK	Indicador de barramento ocupado	S	0

Tabela 1 Tabela descritiva dos pinos do processador genérico

Através do barramento de dados de 16 bits do processador, é possível se transferir informações da largura de um byte (8 bits) ou de uma word (16 bits). Os sinais  $\overline{BHE}$  e  $\overline{BLE}$  são usados para indicar se a transferência de um byte está ocorrendo pela parte menos significativa do barramento de dados (parte baixa:  $\overline{BHE}=1$  e  $\overline{BLE}=0$ ), pela parte mais significativa (parte alta:  $\overline{BHE}=0$  e  $\overline{BLE}=1$ ), ou se está ocorrendo a transferência de uma word (parte alta + parte baixa:  $\overline{BHE}=0$  e  $\overline{BLE}=0$ ). Enquanto todos os 20 bits do bus de endereços podem ser usados para acessar uma posição de memória, apenas os 16 menos significativos podem ser usados para acessar uma E/S. Desta forma, embora o espaço endereçável de memória seja de 1 Mbyte ( $2^{20}$ ), o de E/S é de apenas 64 kbytes ( $2^{16}$ ).

Dos três sinais disponíveis para controle de interrupção, o INTR pode ser mascarado por software, o NMI é sempre atendido, independentemente da programação, e o de RESET reinicializa o sistema.

Para controle de DMA são disponíveis os sinais HOLD e HLDA, os quais permitem que o processador tome conhecimento e sinalize o atendimento de um pedido de uso de vias feito por um outro dispositivo mestre (controlador de DMA). Enquanto durar a concessão do uso de vias (sinal HOLD permanece ativo após o HLDA), o processador mantém seus pinos de saída em estado de alta impedância (estado indefinido entre 0 e 1). Durante o DMA, transferências diretas de dados podem ser feitas entre a memória e o dispositivo de E/S.

Seguindo o padrão das máquinas anteriores ao 486DX, o processador só executa operações aritméticas com números inteiros. Operações com números reais só são possíveis através de bibliotecas matemáticas em ponto flutuante ou através do uso de um coprocessador aritmético, o qual partilha com o processador a execução do programa, tomando para si a execução das tarefas com números reais. Para viabilizar o uso de um coprocessador aritmético, são disponibilizados os sinais PEREQ, BUSY e ERROR, os quais permitem que o processador tome conhecimento de um pedido de dados pelo coprocessador, do seu tempo de ocupação e da ocorrência de erros no tratamento matemático.

## SISTEMA DE CLOCK E CICLOS DE BARRAMENTO

Toda sincronização de barramento é feita a partir do sinal de clock denominado de clock do processador (PCLK). Este sinal, gerado internamente, tem como base de tempo o sinal fornecido na entrada CLK (clock do sistema/externo). Para o processador 8086, o PCLK compreende um período igual a duas vezes o do CLK (frequência de  $PCLK = \frac{1}{2} \times \text{frequência de CLK}$ ), com ciclo de trabalho (duty cycle) de 50% (Figura 9).

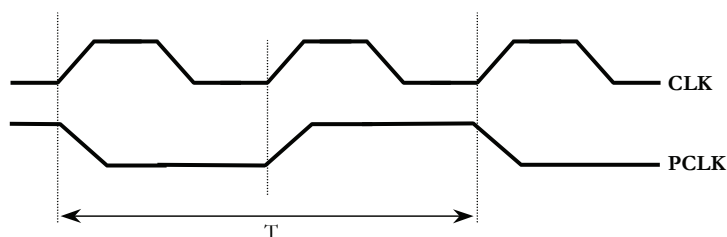


Figura 9 Temporização genérica

Qualquer operação externa de leitura ou escrita feita pelo processador ocorrerá em um ciclo de barramento e terá uma duração mínima de dois períodos de PCLK ( $T_1$  e  $T_2$ ). Os ciclos de barramento possíveis são mostrados na Tabela 3 e o seu reconhecimento é feito através da decodificação dos sinais de saída  $M/\overline{IO}$  (memória/IO),  $D/\overline{C}$  (dados/código) e  $W/\overline{R}$  (escrita/leitura).

$M/\overline{IO}$	$D/\overline{C}$	$W/\overline{R}$	Tipo de ciclo de barramento
0	0	0	Reconhecimento de interrupção
0	0	1	Processador ocioso (nenhum ciclo de barramento está sendo processado)
0	1	0	Leitura de dados numa interface de E/S
0	1	1	Escrita de dados numa interface de E/S
1	0	0	Leitura de código na memória
1	0	1	Parado (halt/shutdown)
1	1	0	Leitura de dados na memória
1	1	1	Escrita de dados na memória

Tabela 2 Sinais M/IO, D/C, W/R

A saída ADS indica que os sinais de definição de ciclo de barramento ( $M/IO$ ,  $R/W$ ,  $D/C$ ), código de byte enable ( $BHE$ ,  $BLE$ ) e os sinais de endereço ( $A_0$  a  $A_{15}$ ) estão estáveis. Esse sinal é geralmente aplicado a um circuito de lógica de controle de barramento externo para indicar que uma definição de ciclo de barramento válido e um endereço estão disponíveis. Através da entrada READY, é possível estender o ciclo de barramento corrente pela inclusão de estados de espera, permitindo que uma memória ou um dispositivo de E/S lento possa ser atendido pelo processador.

Em sistemas multiprocessados, é comum o compartilhamento de vias de dados, endereços e controle. Para suprir o uso exclusivo de vias durante a execução de tarefas prioritárias, o processador dispõe do sinal de saída LOCK (ativado por software), durante o qual qualquer outro sistema ou dispositivo ficará devidamente informado que nenhuma concessão de vias poderá ocorrer naquele momento.

A Figura 10 mostra um timing genérico no qual os seguintes ciclos de barramento possíveis estão representados:

- C1: leitura de código na memória ( $W/ = 0$ ,  $D/ = 0$  e  $M/ = 1$ );
- C2: leitura de dados na memória ( $W/ = 0$ ,  $D/ = 1$  e  $M/ = 1$ );
- C3: ocioso ( $W/ = 1$ ,  $D/ = 0$  e  $M/ = 0$ ); e
- C4: escrita de dados numa interface de E/S ( $W/ = 1$ ,  $D/ = 1$  e  $M/ = 0$ ) com pedido de espera (feito através da linha READY).

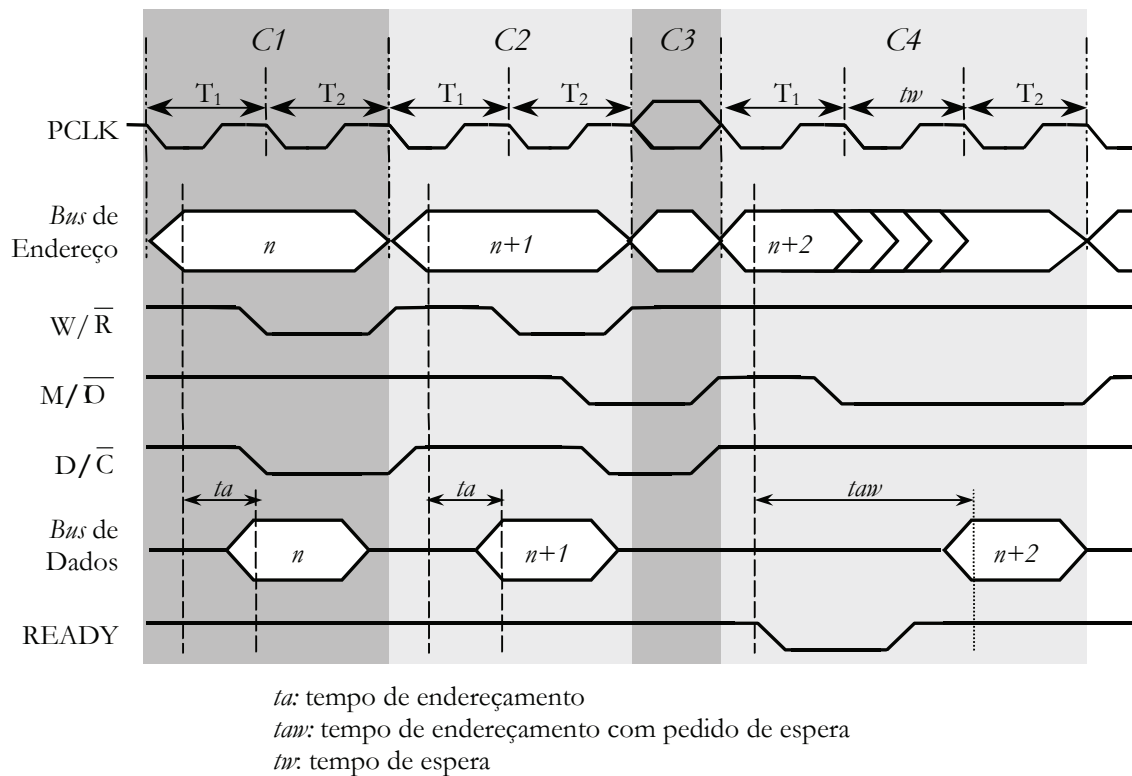


Figura 10 Temporização genérica

# SISTEMA DE MEMÓRIA DE COMPUTADORES

## ESTRUTURA DE MEMÓRIA

Segundo a Arquitetura de Von Neumann, uma unidade central recebe dados dos dispositivos de entrada, processa-os segundo as especificações de um programa, e devolve-os através de um dispositivo de saída. As instruções do programa e os dados processados residem na memória do computador. Esta memória é dividida em uma série de localidades, cada qual com um endereço associado. Cada localização é denominada de byte, o qual é formado por 8 bits (unidade binária).

## Terminologia

**Endereço e localização de memória:** O endereço é um número que identifica a posição (localização) de uma palavra na memória. Cada palavra armazenada em qualquer dispositivo ou sistema de memória possui um único endereço, expresso como números binários ou, por conveniência, hexadecimais. Cada localização de memória possui um endereço associado, onde estão presentes os dados a serem acessados.

**Operação de leitura:** Ao ler o conteúdo de um endereço, o computador faz uma cópia do conteúdo. Dessa forma, a operação de leitura é chamada de não destrutiva (operação de busca).

**Operação de escrita:** Quando o computador acessa uma posição de memória e escreve um dado, o conteúdo anterior é completamente perdido. Assim, toda operação de escrita pode ser chamada de destrutiva (operação de armazenamento).

**Tempo de acesso:** Medida da velocidade do dispositivo de memória. Quantidade de tempo necessária à efetivação de uma operação de leitura (tempo decorrido entre o momento da recepção pela memória

de um novo endereço e o instante em que a informação daquele endereço fica disponível).

**Memória volátil:** Memória que necessita de energia elétrica para reter a informação armazenada. Se a energia for retirada, toda a informação armazenada será perdida.

**Memória não-volátil:** Memória que não necessita de energia elétrica para reter a informação armazenada.

**ROM (read only memory)** – memória apenas para leitura: São memórias a semicondutor usadas para armazenar dados e instruções permanentes, que o computador deve executar freqüentemente ou durante a inicialização do sistema. Normalmente o conteúdo de uma ROM é gravado no circuito integrado, não podendo ser alterado (não voláteis). Aplicações: firmware, memória de partida fria (bootstrap), tabelas de dados, conversores de dados, geradores de caracteres e de funções.

A principal característica da memória ROM é o fato de que suas informações vêm geralmente gravadas de fábrica e são, portanto utilizadas durante toda sua vida útil para as mesmas atividades.

Os diversos tipos de ROMs existentes no mercado diferem no modo de programação e na possibilidade de apagamento e de reprogramação. Algumas variações da ROM são:

- **EPROM (erasable programmable read-only memory)** – ROM apagável programável: Basicamente uma memória ROM na qual informações podem ser apagadas através de exposição a luz ultravioleta de alta intensidade e reprogramadas eletricamente.
- **EEPROM (electrically erasable programmable read-only memory)** – ROM apagável programável eletricamente: versão mais barata e prática da EPROM, a qual utiliza sinais elétricos tanto para sua microprogramação quanto para que suas informações sejam apagadas. As chamadas memórias flash ou EEPROM flash possibilitam a atualização

do BIOS, por exemplo, sem a necessidade de substituição do chip. Uma vantagem das EEPROMs sobre as EPROMs é a possibilidade de apagamento e reprogramação de palavras individuais, em vez da memória inteira.

RAM (random access memory) – memória de acesso aleatório: São memórias de leitura e escrita, usadas para o armazenamento temporário de dados. As memórias RAM são voláteis. Nas memórias RAM, a localização física real de uma palavra não tem efeito sobre o tempo de leitura ou escrita (o tempo de acesso é constante para qualquer endereço). Aplicação: memória principal e caches.

Entre os principais tipos de RAM temos:

- DRAM (dynamic random access memory) – RAM dinâmica: Memórias RAM a semicondutor nas quais a informação armazenada não permanecerá armazenada, mesmo em presença de alimentação do circuito, a não ser que a informação seja reescrita na memória com determinada frequência (operação de recarga ou refresh da memória). Características: alta capacidade de armazenamento, baixo consumo de potência, velocidade de operação moderada e custo/byte relativamente baixo. Principal tecnologia de implementação de memórias RAM, constituindo-se na representação de números binários 0 e 1 a partir do carregamento de milhares ou milhões de microcapacitores reenergizados pela passagem de correntes pelas células a cada dezena de milissegundos.
- SRAM (static random access memory) - RAM estática: Memórias RAM nas quais a informação permanecerá armazenada enquanto houver energia elétrica aplicada à memória, sem que haja necessidade da informação ser permanentemente reescrita na memória. Características: baixa capacidade de armazenamento, alta velocidade de operação (baixo tempo de acesso) e alto custo/byte. A principal limitação da DRAM, a velocidade, é a principal

vantagem da SRAM. Estima-se que o tempo de acesso aos dados na SRAM sejam cerca 25% do tempo de acesso a DRAMs. Uma outra vantagem diz respeito a desnecessidade de reenergização. No entanto, uma grande limitação é decorrente de seu alto custo, o que faz com que esta tecnologia seja usada mais comumente em estruturas cache. Utilizam flip-flops (bipolares ou MOS) como células de armazenamento.

**Memórias de acesso seqüencial (SAM):** Utilizam registradores de deslocamento para armazenar os dados que podem ser acessados de forma seqüencial, não podendo ser utilizadas na memória principal (baixa velocidade de acesso). Aplicações: armazenamento e transmissão seqüencial de dados codificados em ASCII, osciloscópios digitais e analisadores lógicos, memórias FIFO.

**Memórias FIFO (first in first out) – primeiro a entrar, primeiro a sair:** Memória seqüencial que utiliza registradores de deslocamento, na qual as palavras são descarregadas na saída de dados na mesma ordem em que entraram pela entrada de dados. Aplicação: operação de transferência de dados entre sistemas operando em velocidades muito diferentes (buffers de impressora ou teclado).

**Hierarquia de memória:** a hierarquia de memória de um computador é caracterizada por cinco parâmetros: tempo de acesso, tamanho da memória, custo por byte, largura de banda (bandwidth) da transferência e unidade de transferência (bytes). A hierarquia de memória em microcomputadores, representada na Figura 11, possui quatro níveis, compostos de: memória secundária, memória principal, caches e registradores.

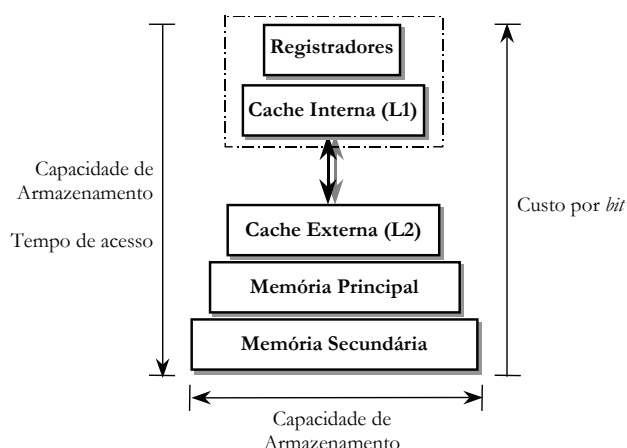


Figura 11 Hierarquia de memória



**Memória secundária ou de massa ou auxiliar:** Armazena uma grande quantidade de informação, sendo bem mais lenta que a memória principal, e sempre é não-volátil. As informações armazenadas nos dispositivos de memória de massa são transferidas para a memória principal quando forem necessárias ao computador. Exemplos deste tipo de elemento são as fitas magnéticas, os disquetes e os HDs (hard disks).

**Memória principal ou primária:** Serve para armazenar as instruções e os dados que estão sendo usados pelo processador. Implementada por chips de memórias DRAM. Controlada por uma MMU (memory management unit – unidade de gerência de memória) em cooperação com o sistema operacional.

**Memória cache:** Formada por uma pequena quantidade de memória SRAM, com alta velocidade de acesso. Armazena dados com grande probabilidade de reutilização, evitando outros acessos à memória principal, diminuindo o tempo de acesso ao dado. Existem dois tipos de memória cache, dependendo da localização em relação ao microprocessador: cache interna ou primária ou L1 (com pequena capacidade, geralmente de 32 kbytes), e cache “externa” (integrada) ou secundária ou L2 (com capacidade média de 256 a 512 kbytes).

### **Armazenamento de Informações na Memória**

A menor quantidade de informação disponível em qualquer computador é o bit. A principal unidade de informação é um grupo de bits, denominado palavra. O número de bits que formam uma palavra é denominado tamanho da palavra do computador. Muitas vezes os computadores são descritos em termos do tamanho da sua palavra, que também indica a largura do barramento de dados.

Uma palavra armazenada numa posição de memória pode conter dois tipos de informação: instruções ou dados. Para as instruções, armazenadas em código binário, o conteúdo armazenado na memória são comandos que levam à execução de alguma tarefa. Os dados podem ser informações numéricas ou alfanuméricas, que podem

estar em vários formatos: números binários com e sem sinal, números em BCD compactado, números em ponto flutuante, caracteres em BCD ou ASCII.

Apesar de existirem diversos formatos lógicos de armazenamento para dados, não se deve esquecer que, fisicamente, informações são armazenadas como seqüências de 0 e 1 e que, mais importante ainda, o computador não tem como saber a diferença entre dois conteúdos. É responsabilidade do programador conhecer os tipos de dados que estão sendo armazenados, para assegurar que o programa possa interpretá-los e processá-los corretamente.

O processador armazena dados na memória de forma linear, em grupos de 8 bits. Primeiro é armazenado o byte menos significativo (LSB – least significant byte) de uma word e em seguida o byte mais significativo (MSB – most significant byte). Por exemplo, o número A3C1h é armazenado na forma indicada na Figura 12.

Endereço	Conteúdo	
<i>Segmento de Dados</i>		
'	<b>C1</b>	<i>Parte baixa</i>
' +1	<b>A3</b>	<i>Parte alta</i>

Figura 12 Armazenamento de dados na memória

Números binários não sinalizados e números hexadecimais: Uma forma mais clara de representação dos valores é o sistema hexadecimal, que utiliza um conjunto de 16 símbolos: os algarismos de 0 a 9 e as letras de A a F. A cada conjunto de 4 bits em um byte atribui-se um dígito hexadecimal. Atenção! Números em hexadecimal são apenas uma representação de números binários!!!

Caracteres: A cada uma das 256 combinações possíveis em um byte, pode-se atribuir um caractere do alfabeto, minúsculos e maiúsculos, algarismos e símbolos especiais do teclado, caracteres de controle, caracteres semigráficos, símbolos matemáticos e letras do alfabeto grego (código ASCII – american standard code for information

interchange). Não se pode esquecer que a cada código ASCII (ou a cada caractere), está associado um número de um byte, e portanto, podemos dizer que, por exemplo, 'A' = 65d = 41h = 01000001b.

Valores em BCD compactado: O código BCD (binary decimal code – código binário decimal) é utilizado para armazenar dois dígitos decimais em um byte na memória, especialmente para realizar cálculos aritméticos. Para isto, pode-se usar a tabela dos números hexadecimais com as combinações para os algarismos de 0 a 9.

Exemplo:  $153_{10} = 0001\ 0101\ 0011_{\text{BCD}} = 1001\ 1001_2 = 99_{16}$   
 Para cada dígito decimal existe um nibble (quarteto) correspondente em BCD.

Operações BCD são realizadas como operações com números binários. Entretanto, é necessário fazer um ajuste no resultado para que o valor seja um BCD válido. Portanto, uma soma BCD nada mais é que uma soma binária com um ajuste. O ajuste é feito adicionando-se

$6_{10} = 0110_2$  ao nibble cuja representação binária seja superior a  $9_{10}$ .

Exemplo:  $49_{10} + 21_{10} = 0100\ 1001_{\text{BCD}} + 0010\ 0001_{\text{BCD}} = 0110\ 1010$

Ajuste:

```

+ 0110 10102
  0000 01102
  —————
  0111 0000BCD
  
```

Após o ajuste, o resultado da soma é:  $0111\ 0000_{\text{BCD}} = 70_{10}$

Valor Binário	Dígito Hexadecimal	Valor Decimal	Dígito BCD
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5

0110	6	6	6
0111	7	7	7
1000	8	8	8
1001	9	9	9
1010	A	10	-
1011	B	11	-
1100	C	12	-
1101	D	13	-
1110	E	14	-
1111	F	15	-

Tabela 3 Equivalência entre os sistemas numéricos

Valores sinalizados: Um byte pode armazenar 256 valores diferentes (00h a FFh), atribuindo a cada combinação um valor decimal positivo ( $0_{10}$  a  $+255_{10}$ ). Para armazenar na memória valores positivos e negativos ( $-128_{10}$  a  $+127_{10}$ ), o bit mais significativo de um byte é utilizado para indicar o sinal: 0 para positivo e 1 para negativo em complemento de 2 (ou seja,  $-B = \text{Complemento de } 2(B)$ ). O complemento de 2 consiste em duas operações: realização do complemento de 1 (negação bit a bit) e adição com 1.

Exemplo:  $A = 12_{10} = 0000\ 1100_2$   
 $-A = C_2(A) = C_1(A) + 1 = 1111\ 0100_2$

Internamente, os computadores não realizam operações de subtração. A subtração ( $A-B$ ) é feita através da soma ( $A+(-B) = A+\text{Complemento de } 2(B)$ ).

Exemplo:  $A = 7_{10}$  e  $B = 8_{10}$ . Calcular, por complemento de 2, o valor  $C = A - B$ .

$A = 0000\ 0111_2$  e  $B = 0000\ 1000_2$

Operação 1  $\rightarrow C_1(B)$ :  $1111\ 0111_2$

Operação 2  $\rightarrow$  adição com 1 ( $1+0=1$ ,  $1+1=0$  vai 1):

+ 1111 0111  
0000 0001

$$1111\ 1000 = C2(B)$$

Adição ( $A + C2(B)$ ):

$$\begin{array}{r} +\ 0000\ 0111 \\ \underline{1111\ 1000} \\ 1111\ 1111 \end{array}$$

Logo,  $A - B = 1111\ 1111_2$ .

Note que o dado sinalizado é armazenado na memória da mesma forma que um não sinalizado. Para saber quanto vale o número, é só refazer o complemento de dois, uma vez que  $[-(-B)=B]$ . Dessa forma, teríamos:

$$\begin{array}{r} +\ 0000\ 0000 \\ \underline{0000\ 0001} \\ 0000\ 0001 \end{array}$$

Portanto,  $1111\ 1111_2 = -1$ .

## ORGANIZAÇÃO DA MEMÓRIA PRINCIPAL

### Organização Modular da Memória

A memória é freqüentemente separada em módulos com funcionalidade independente como forma de aumentar a velocidade e a confiabilidade, e flexibilizar possíveis mudanças de tamanho. Existem duas formas de organização modular: high-order interleave e low-order interleave.

High-order interleave: sucessivos endereços de memória presentes no mesmo módulo (Figura 13). O aumento do desempenho resulta da probabilidade de instruções, tabelas de referências e dados residirem em módulos diferentes. Esta separação pode ser forçada pelo sistema operacional. Caso isto venha a ocorrer, existe pouca

probabilidade de uma referência à memória por uma instrução ter de esperar pelo término de uma referência à uma tabela ou a um dado.

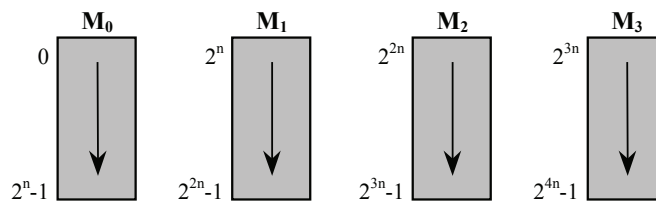


Figura 13 Organização high-order interleave

Low-order interleave: endereços sucessivos alocados em módulos diferentes (Figura 14). Oferece vantagem quanto à localidade de programas e dados. Neste método, um dado ou uma instrução tende a estar na localidade de memória mais próxima do último item de dado ou instrução buscado. Existe também a possibilidade de um fluxo de palavra poder ser transferido ao processador a uma velocidade maior que aquela organizada em endereços sucessivos.

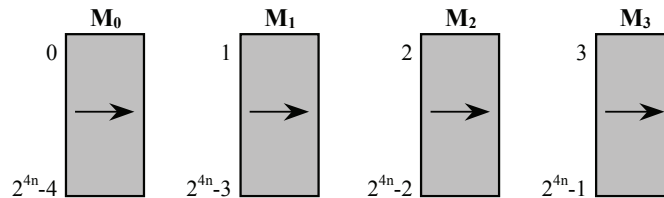


Figura 14 Organização low-order interleave

É difícil fazer um julgamento sobre qual forma de organizações é melhor. A organização high-order interleave certamente produz alta confiabilidade ao sistema, uma vez que o processamento pode continuar quando um módulo não está operando, principalmente nos casos em que um conteúdo pode ser recuperado da memória secundária e realocado em outro módulo.

Considerando as duas formas, é mais importante que se incorpore uma organização de memória compatível com a estrutura do restante do sistema do computador e com a organização do processador. Com isso, a memória passa a acomodar as características do resto do sistema.

Uma unidade de gerência de memória (MMU) é utilizada para fazer o endereçamento ou acesso à memória. A MMU é um processador especial usado particularmente para coordenar transferências de bytes, words e doublewords. Isto também facilita o projeto de diferentes computadores usando microprocessador e memórias padrões. Funções típicas de uma MMU incluem: controle de endereçamento por segmento e página, separação do espaço do usuário e do sistema e suporte de hardware para proteção de memória.

### **Organização Lógica**

Vamos considerar um processador com barramento de endereço de 32 bits, o que possibilita o acesso a 4 Gbytes ( $2^{32}$ ) posições de memória. Essa capacidade de endereçamento, entretanto, só é possível quando o microprocessador está trabalhando no modo protegido. No modo real, apenas 20 bits de endereçamento são utilizados e, conseqüentemente, a capacidade de endereçamento diminui para 1 Mbyte ( $2^{20}$ ), semelhante ao processador proposto para análise.

**Memória Convencional:** até 640 kbytes.

**Memória Superior:** região de memória entre 640 kbytes e 1 Mbyte, que abriga a memória de vídeo, a UMB e a EMS Page Frame.

**Blocos de Memória Superior (UMB):** armazenamento de programas residentes e drivers que estariam na memória convencional. **Localização:** de 800 a 896 kbytes da memória superior.

**EMS Page Frame:** bloco de 64 kbytes da memória superior, através do qual é feito o chaveamento de bancos da memória EMS. **Configuração:** 4 páginas de 16 kbytes = 64 kbytes. **Localização:** 896 a 960 kbytes da memória superior.

**Memória Alta (HMA):** utilizada para armazenar partes do núcleo do sistema operacional DOS, que poderiam ocupar espaço na memória

convencional ou na memória superior.

**Memória Estendida (XMS):** memória acessível apenas em modo protegido, localizada a partir de 1 Mbyte, abrigando a memória HMA e a memória expandida.

**Memória Expandida (EMS):** fornece acesso superior a 1 Mbyte de memória em modo real, independente da capacidade de endereçamento do processador. Localização: placas de expansão EMS (XT e 286), ou final da memória estendida (386 ou superior). Utilização de recursos (drivers) de gerência de memória que permitem a função de chaveamento de bancos sem necessidade de hardware adicional (EMM386.EXE e HIMEM.SYS). Total de memória: 64 blocos (frames) de 16 kbytes = 1 Mbyte. Apenas 4 blocos de 16 kbytes podem ser acessados a cada instante. Apesar de não poder armazenar programas a serem executados, possibilita o armazenamento dos dados utilizados pelos programas.

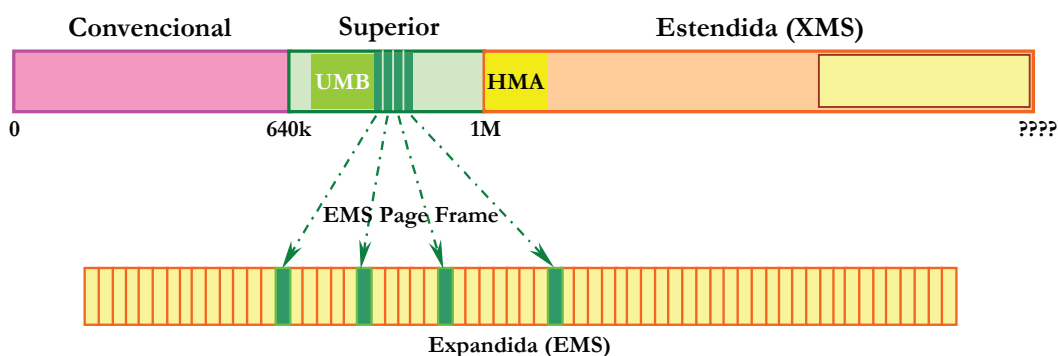


Figura 15 Organização da memória principal

## Organização Física

O espaço de endereçamento de memória é visto como um conjunto de quatro bancos independentes onde cada banco apresenta 1Gbyte de memória (modo protegido) ou de 256 kbytes (modo real).

Os bancos estão relacionados diretamente com os 4 sinais byte enables (BE0, BE1, BE2 e BE3) do processador. A ativação de cada byte enable coloca o banco de memória correspondente em operação.



### Características da organização física:

- a memória está organizada em seqüências de doublewords;
- os bits de endereço (A2 a A31) são aplicados aos quatro bancos em paralelo;
- os byte enables selecionam quais bytes da doubleword serão manipulados;
- cada doubleword alinhada possui endereço inicial múltiplo de 4; e
- cada banco de memória utiliza 8 das 32 linhas do barramento de dados.

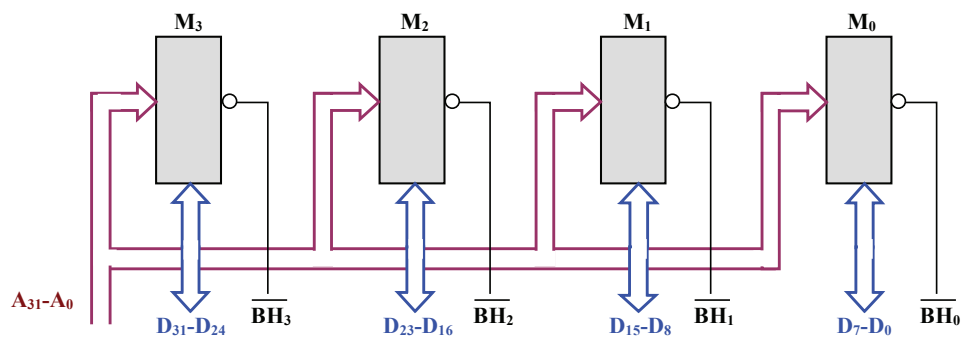


Figura 16 Organização física da memória principal

### Acesso à Memória

As Figuras a seguir mostram a forma de acesso a dados de 1, 2 e 4 bytes, respectivamente, a partir do endereço inicial X. É importante notar que apenas os bancos selecionados através dos byte enables poderão realizar operação de escrita ou leitura e a transferência dos dados é feita através das linhas correspondentes do barramento de dados.

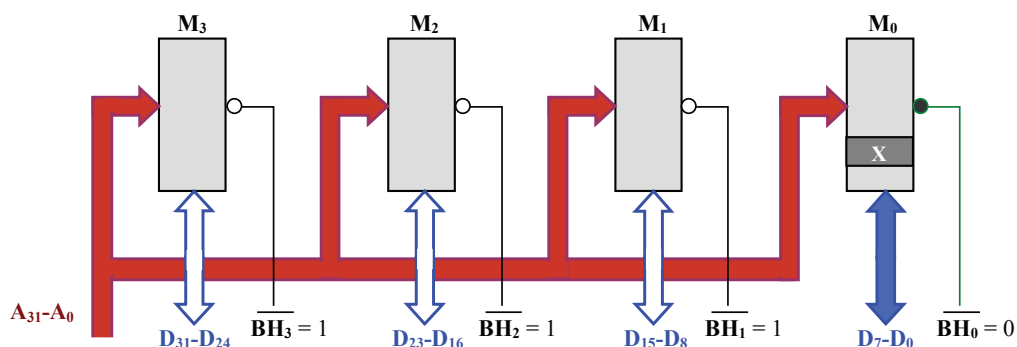


Figura 17 Acesso a um dado de 1 byte

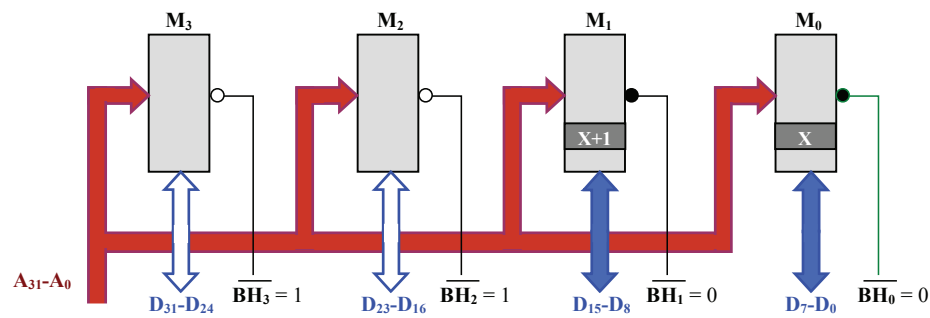


Figura 18 Acesso a um dado de 2 bytes

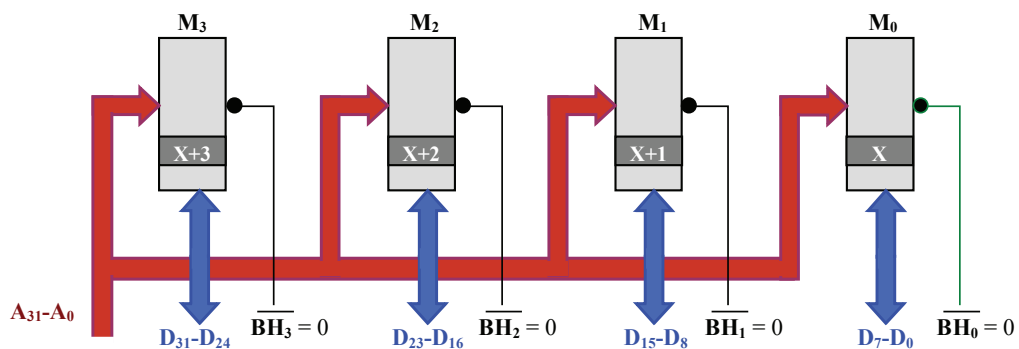


Figura 19 Acesso a um dado de 4 bytes

Nem sempre é possível, entretanto, ter todas as words e doublewords alinhadas. Nesse caso, a transferência será feita em dois ciclos de barramento. No primeiro, a parte alta do dado será acessada; no segundo, a parte baixa.

## MEMÓRIA CACHE

Quando um sistema de computador explora um subsistema de memória principal muito grande, esta memória é normalmente implementada com DRAMs e EPROMs de alta capacidade de armazenamento, mas de muito baixa velocidade. As DRAMs disponíveis apresentam tempos de acesso elevado em relação à velocidade do processador, inviabilizando a operação síncrona de um sistema a microprocessador com estes dispositivos de memória. Como não existe disponibilidade de dispositivos com tempo de resposta compatível, estados de espera são introduzidos em todos

os ciclos de acesso a memória de programa ou de dados. Estes estados de espera degradam a performance global do sistema a microprocessador.

Para suprir esta deficiência, um pequeno subsistema de memória, conhecido por memória cache, é inserido entre o processador e a memória principal. O sistema de memória cache armazena os dados e códigos mais recentemente utilizados, permitindo que, ao invés de realizar novos acessos à memória principal, estes dados e códigos sejam acessados diretamente da cache, com a possibilidade de zero estados de espera. Um outro dispositivo, o controlador de memória cache determina, de acordo com as necessidades, quais os blocos de memória a serem movimentados de/para o bloco cache, ou de/para a memória principal.

A observação de que referências à memória feitas em qualquer intervalo de tempo curto de tempo tendem a usar apenas uma pequena fração da memória total é chamada princípio da localidade e forma a base de todos os sistemas cache.

Para aumentar ainda a mais o desempenho, os processadores de última geração incorporam caches de dados e de código internamente ao chip (cache L1).

### Arquitetura de um Sistema Cache

A Figura 20 apresenta a arquitetura de um sistema cache. É interessante notar que um dos lados da memória cache está ligado ao barramento local do microprocessador e o outro lado está ligado ao barramento da memória principal.

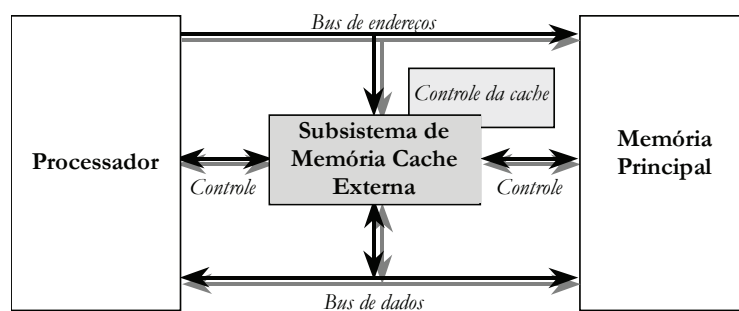


Figura 20 Microcomputador com memória cache L2

A primeira vez que um microprocessador executa um segmento de programa, uma instrução após a outra é lida da memória principal e executada. O grupo de instruções mais recentemente lido é então copiado na memória cache. Como normalmente os softwares implementam seqüências de instruções que são executadas repetidas vezes, o acesso a essas informações poderá ser feito diretamente da cache.

Considere, por exemplo, uma seqüência de instruções em loop. Durante a primeira iteração, o código é lido da memória principal pelo microprocessador e automaticamente copiado para a memória cache (Figura 21). As demais iterações do loop não mais requisitarão acessos à memória principal. Durante a execução do loop, tanto os dados como o código podem ser copiados para a memória cache. Quanto mais acessos à memória cache e menos à memória principal, melhor será o desempenho do sistema.

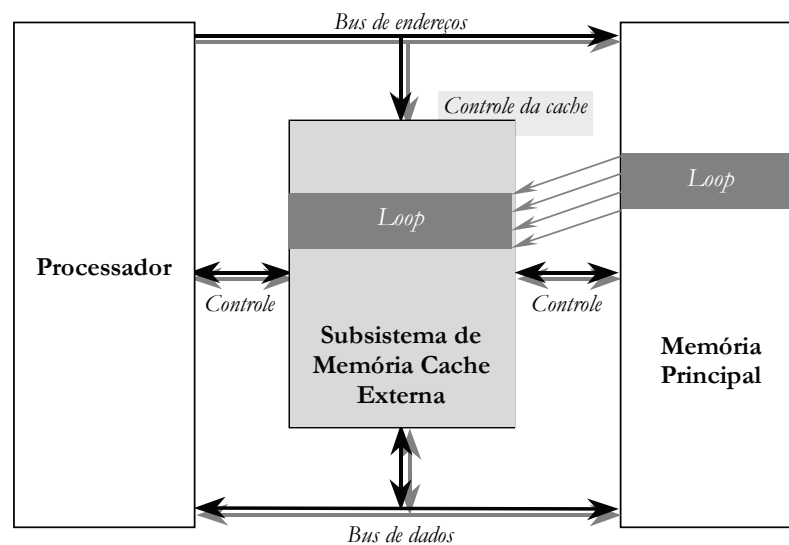


Figura 21 Exemplo de funcionamento da memória cache

### Taxa de Acerto

A cache pode reduzir significativamente o tempo de acesso se organizada de forma a conter os dados e o código requeridos pelo processador. Obviamente, quanto maior a cache, maior a chance de que ele contenha as informações desejadas. Quando o processador necessita de uma informação, o subsistema de memória cache

verifica seu conteúdo. Se a informação estiver presente na cache, o ciclo de memória é denominado cache hit (acerto de cache) e a cópia da informação é acessada. Caso contrário, diz-se que ocorreu um cache miss e a informação deve ser buscada na memória principal.

A taxa de acerto (hit rate) é a razão entre o número de acessos à cache e o número total de acessos à memória.

$$\text{Hit Rate} = \frac{\text{Acessos à cache}}{\text{Acessos à memória principal}} = \frac{\text{Total de cache hits}}{\text{Total de cache miss}} \quad \text{ou} \quad h = \frac{k-1}{k}$$

Se o tempo de acesso de um dado na cache é  $c$  e da memória principal é  $m$ , o tempo médio de acesso de um dado, considerando o sistema de memória cache + memória principal, pode ser calculado por:  $t_{\text{med}} = c + (1 - h) \times m$ . À medida que  $h \rightarrow 1$ , todas as referências podem ser satisfeitas pela cache, e  $t_{\text{med}} \rightarrow c$ . Por outro lado, à medida que  $h \rightarrow 0$ , uma referência à memória é necessária toda vez, de forma que  $t_{\text{med}} \rightarrow (c + m)$ , tempo para verificar o cache (sem sucesso) e para fazer a referência à memória.

Quanto maior a taxa de acerto, maior a eficiência do subsistema de memória cache. A taxa de acerto não é um valor fixo e depende da dimensão e da organização da cache, do algoritmo de controle usado pela cache e do código em execução, de tal forma que pode ser completamente distinta para códigos diferentes.

# ARQUITETURA DE SOFTWARE DE UM MICROPROCESSADOR

O conhecimento da arquitetura de software de um microprocessador permite o desenvolvimento de programas sem a necessidade de detalhes de implementação do chip. A arquitetura de software de um processador compreende:

- a forma de organização da memória e da E/S;
- que tipos de registradores estão disponíveis internamente e quais as suas funções;
- que tipos de dados servem de operando;
- quais os modos de endereçamento existentes para se acessar um operando na memória, numa E/S ou num dos registradores internos; e
- quais os comandos que constituem o seu conjunto de instruções.

## MODELO DE SOFTWARE

O modelo de software inclui (Figura 22):

- um espaço contínuo e endereçável de 1 Mbyte de memória (20 bits do barramento de endereços);
- um espaço para endereçamento de portas de E/S de 64 kbytes (16 bits do barramento de endereços); e
- uma estrutura com 14 registradores internos disponíveis.

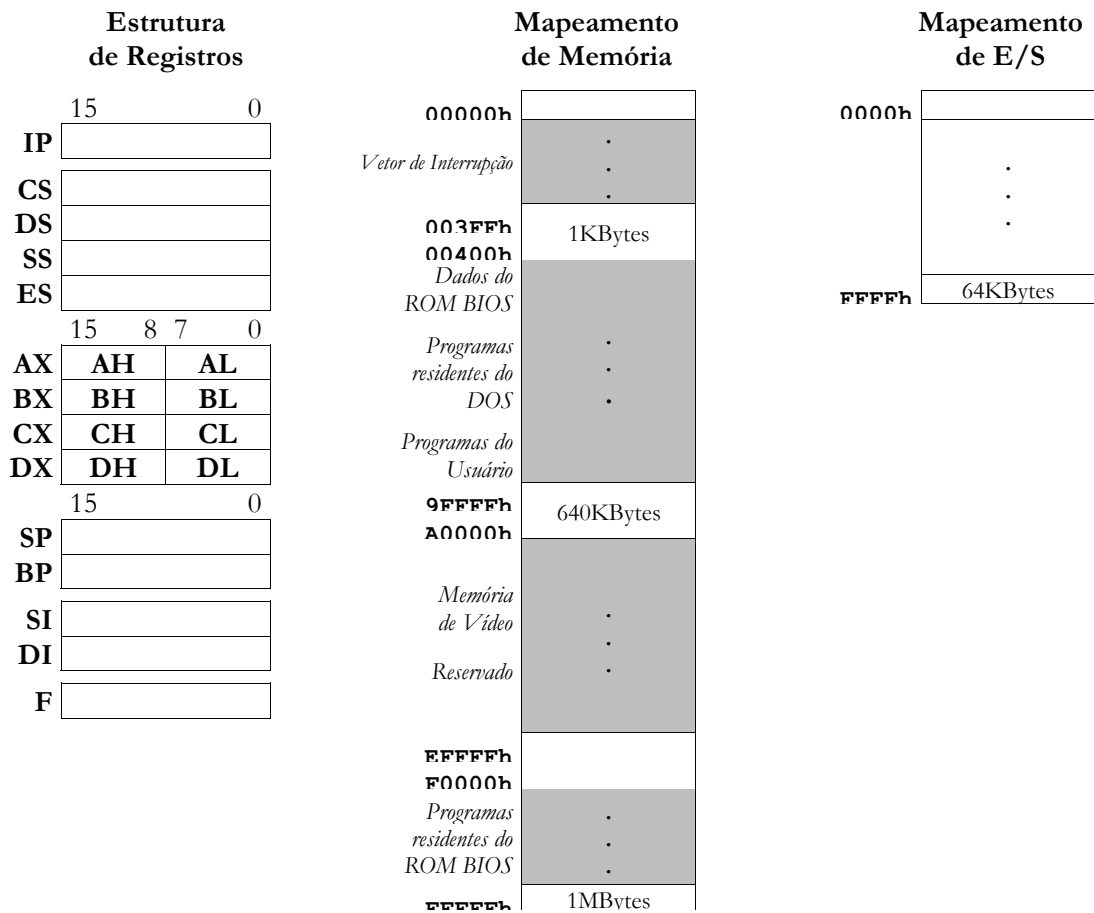


Figura 22 Modelo de software de um processador

## REGISTRADORES

### Registradores de Dados (Registradores de Uso Geral)

Os registradores de dados podem ser usados como:

- 4 registradores de 16 bits (AX, BX, CX e DX), para manipulação de words (16 bits); ou
- 8 registradores de 8 bits (AL, AH, BL, BH, CL, CH, DL e DH), para manipulação de bytes (8 bits).

A terminação L (low) ou H (high) define onde será armazenado o byte de mais baixa ordem ou o de mais alta ordem de uma palavra de 16 bits.

Embora considerados de uso igualitário pela maioria das instruções

aritméticas, lógicas ou de transferência de informação, os registradores de dados apresentam algumas características próprias que os diferenciam:

- as instruções de multiplicação, divisão ou de transferência de dados de uma E/S exigem o registrador AX como acumulador e, o registrador DX como registrador de dados auxiliar;
- as instruções que usam forma de endereçamento de memória mais complexa, exigem o registrador BX como registrador de base; e
- as instruções que manipulam strings ou loops de contagem exigem o registrador CX como registrador de contagem.

### **Registradores de Segmento**

Um programa executável deve ser constituído por módulos de código e de dados. Para suportar esta estrutura de programa modular, o processador aloca cada unidade lógica de um programa em regiões específicas da memória denominadas de segmentos. Por razões de implementação física, cada um destes segmentos não poderá exceder o limite de 64 kbytes.

Como apenas um pequeno número de módulos de programa e dados são necessários em um dado instante, este mecanismo, chamado de segmentação de memória, permite que os programas sejam executados rapidamente e tomem pouco espaço na memória principal, além de facilitar o desenvolvimento de programas e a sua manutenção.

Em qualquer instante, um programa em execução só poderá ter acesso a 4 segmentos:

- um segmento de código, onde será alocado o módulo executável (instruções);
- um segmento de dados, onde serão alocadas tabelas, mensagens, variáveis ou constantes (dados) necessárias à execução do programa;
- um segmento de pilha, onde serão manipulados



principalmente os endereços de retorno de subrotinas e as variáveis locais de módulos de programa em execução; e

- um segmento extra de dados, onde serão alocadas tabelas, mensagens, variáveis ou constantes não suportadas pelo segmento de dados.

Para identificar e apontar cada um destes segmentos na memória, o processador utiliza 4 registradores de 16 bits:

- CS – como registrador de segmento de código;
- DS – como registrador de segmento de dados;
- SS – como registrador de segmento de pilha; e
- ES – como registrador de segmento extra de dados.

A alocação física de um segmento pode se apresentar totalmente dissociada, parcialmente sobreposta ou totalmente sobreposta. Na Figura 23, por exemplo, o segmento de pilha está parcialmente sobreposto ao segmento de código, o segmento de código está totalmente dissociado do segmento de dados e o segmento extra de dados está totalmente sobreposto ao segmento de dados.

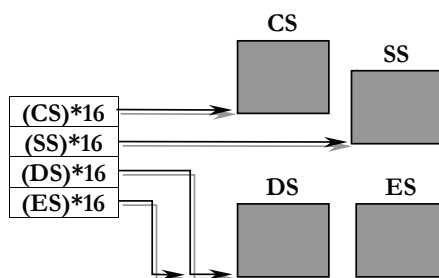


Figura 23 - Segmentos parcialmente sobrepostos, totalmente dissociados e totalmente sobrepostos

## Geração de Endereço Físico em Segmentos

Para poder gerar o endereço físico de memória correspondente ao início de um segmento de código, de dados, de pilha ou de dados extra, o processador busca o conteúdo do registrador de segmento CS, DS, SS ou ES (de 16 bits), e multiplica-o por 16 (quatro deslocamentos com zeros à esquerda). Como resultado, tem-se uma quantidade da

largura do barramento de endereços, ou seja, 20 bits. Isso significa que um segmento poderá se localizar, no espaço de 1 Mbyte, a partir de qualquer endereço múltiplo de 16.

Como o conteúdo de um registrador de segmento permite a definição apenas do endereço da base do segmento, a manipulação de um dado dentro do espaço físico ocupado por todo o segmento só será completamente determinada (Figura 24) se for fornecido um deslocamento, que caracterize a distância relativa do dado a esta base. A fonte para este deslocamento (offset) depende do tipo de referência feita à memória: poderá ser o conteúdo do registrador de instrução (busca do código de uma operação), ou do registrador de índice (acesso a uma área de memória de dados), ou de um registrador de base de pilha (acesso a uma área de memória de pilha), ou ainda uma associação de um índice e uma base. Esta definição é possível através da análise dos modos de endereçamento de memória do processador.

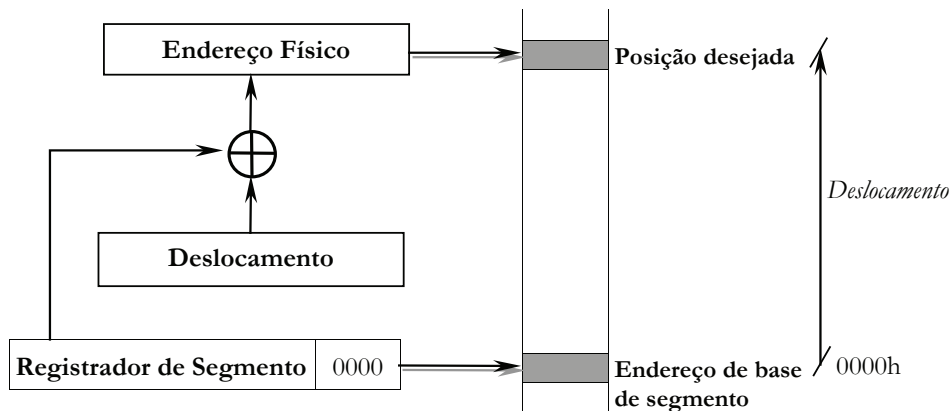


Figura 24 Endereço físico

Formalmente, o endereço físico (EF) de um determinado dado presente na memória estará perfeitamente definido se referenciado pelo conteúdo de um registrador de segmento e um deslocamento dentro do segmento, ou seja:

EF = registrador de segmento : deslocamento (representação do endereço físico)

EF = registrador de segmento  $\times$  16 + deslocamento (valor do endereço físico)

Exemplo:

Se o conteúdo em

CS é 0200h e o conteúdo em IP é 0450h,

a próxima instrução a ser buscada estará no endereço

$$EF = CS \times 16 + IP = 0200 \times 16 + 0450 = 02000 + 0450 = 02450h$$

+ 02000

0450 Base Deslocamento

02450 Endereço Físico

representado por 0200h : 0450h (CS : IP).

Logo, 0200h : 0450h é a representação para o endereço físico 02450h, dentro do segmento de código.

### **Registradores Ponteiros e de Índice (Registradores de Deslocamento)**

Os registradores ponteiros e de índice são usados para armazenar valores de deslocamento no acesso a instruções (IP) ou no acesso a certas posições da memória pilha (SP e BP), ou ainda, na manipulação de dados ou de blocos de dados tipo matrizes ou tabelas nos segmentos de dados (SI e DI). Especificamente têm-se:

- IP como apontador de instrução no segmento de código CS.
- SP como ponteiro do topo do segmento de pilha corrente (SS);
- BP como ponteiro de base do segmento de pilha (deslocamento);
- SI como índice fonte de estruturas de dados presentes em DS ou ES;
- DI como índice destino de estruturas de dados presentes em DS ou ES; e

No tratamento de transferências de dados controladas pelos ponteiros de índice, o SI normalmente especifica um índice fonte no segmento DS, enquanto DI especifica um índice destino no segmento ES.

## Registrador de Flags

O registrador de flags (Figura 25) é composto de 16 bits independentes, sendo que apenas 9 bits são utilizados como flags:

- 6 flags de estado (status flags): descrevem os resultados gerados por uma instrução;
- 3 flags de controle: controlam a operação do processador; e

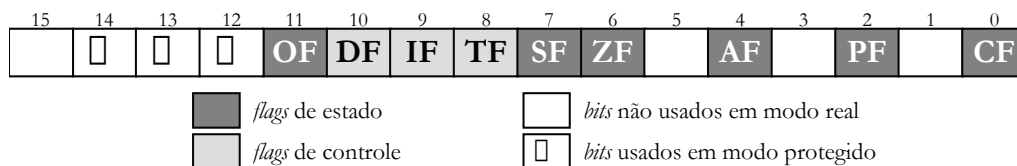


Figura 25 Registrador de flags

## Flags de Estado:

Os flags de estado, usados normalmente pelo programador como auxiliares na hora de uma tomada de decisão, são setados (1) ou resetados (0) após a execução de instruções aritméticas e lógicas, refletindo propriedades características do resultado.

Todos os flags de estado vão para o nível lógico 1 para indicar uma ação correspondente ao seu nome.

Os flags de estado presentes no processador são:

- ZF – zero flag (flag zero): indica que o resultado de uma operação é zero.
- SF – sign flag (flag de sinal): indica que o resultado de uma

operação é negativo.

- OF – overflow flag (flag de estouro): indica que o resultado de uma operação com números sinalizados excede o limite de possível de representação do operando destino.
- CF – carry flag (flag de carregamento): indica, em operações com números não sinalizados, que o resultado de uma operação não cabe no operando destino (transporte ou empréstimo em operações aritméticas, ou bit expulso em deslocamento ou rotação);
- AF – auxiliar carry flag (flag de carregamento auxiliar): indica que um ajuste, após uma operação com números BCD, é necessário.
- PF – parity flag (flag de paridade): usado para indicar se o resultado de uma operação possui um número par de bits 1 (comunicações de dados).

### **Flags de Controle:**

O flags de controle controlam operações do processador, modificando o comportamento de instruções que serão executadas.

Os flags de controle presentes no processador são:

- DF – direction flag (flag de direção): controla, nas instruções com string, se os registradores SI e DI terão seus conteúdos automaticamente incrementados (DF=0) ou decrementados (DF=1) → acesso direto ou inverso a strings.
- IF – interrupt flag (flag de interrupção): mascara, com IF=0, pedidos de interrupção feitos através da entrada INTR do microprocessador.
- TF – trap flag: coloca, com TF=1, o processador no modo de

operação passo a passo.

No modo de operação passo a passo, após a execução de cada instrução do programa, ocorre uma interrupção de passo único a qual, aciona uma rotina de depuração. Quando o microprocessador executa a interrupção de passo único, o TF=1 é salvo na pilha e resetado para permitir a execução normal da rotina de depuração. No retorno desta rotina, a condição TF=1 será novamente reativada (Figura 26).

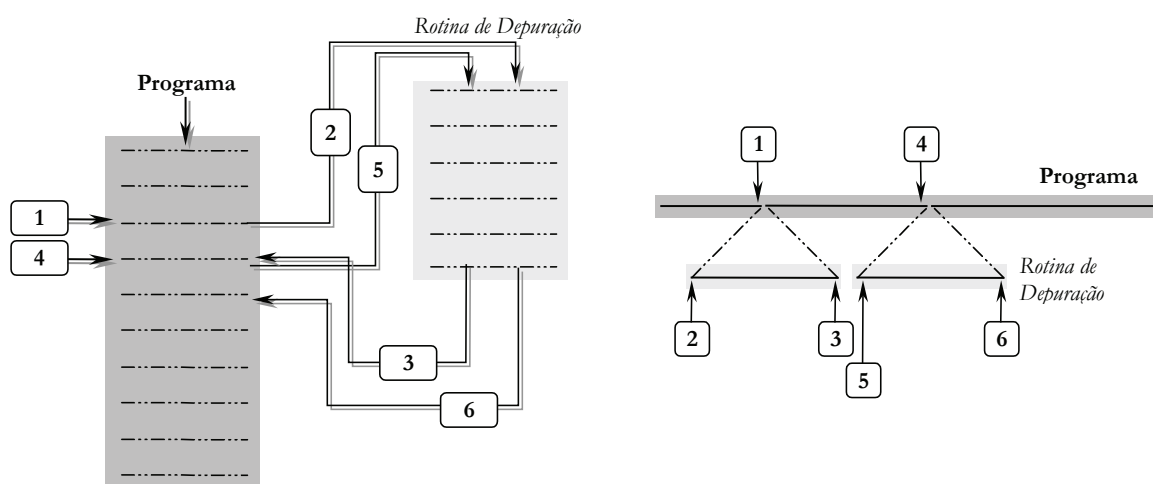


Figura 26 Modo de operação passo a passo

## PILHA

A pilha de um processador é uma área de memória usada para guardar dados, normalmente valores presentes nos registradores, que, devido à complexidade do programa, precisam ser alterados, mas que devem depois recuperar seu antigo valor. As instruções que fazem isso são, basicamente PUSH e POP.

Os usos mais comuns da pilha são:

- para salvar endereços de retorno para instruções na chamada e nos retornos de subrotinas e ocorrências de interrupções;
- para salvar o conteúdo de registradores quando ocorre uma chamada a subrotina;
- para passar informações de uma rotina para outra; e

- para armazenar, temporariamente, resultados durante operações complexas.

No processador temos o registrador SS que contém o valor do segmento reservado para a pilha e o ponteiro SP que indica, dentro desse segmento, qual é o offset do topo da pilha.

Após a execução de instruções que acessam dados na pilha, o registrador SP é automaticamente alterado (decrementado na inserção e incrementado na retirada), indicando o novo topo da pilha. SP assume, inicialmente o valor 0000h, quando nenhum dado está presente na pilha

O registrador BP indica um deslocamento qualquer dentro da pilha, entre os endereços da base e do topo da pilha. BP assume, inicialmente, o valor 0000h (offset 0000), apontando para a base do segmento, podendo receber atribuições durante a execução do programa.

Todo acesso à pilha é feito em words, isto é, não se pode acessá-la para guardar apenas o conteúdo dos registradores AL ou CL, sendo necessário guardar todo o AX ou o CX.

Uma característica interessante e que deve ser notada é que a pilha do processador tem sua base num endereço de memória alto dentro do segmento de pilha, crescendo em direção à memória de endereço baixo (Figura 27).

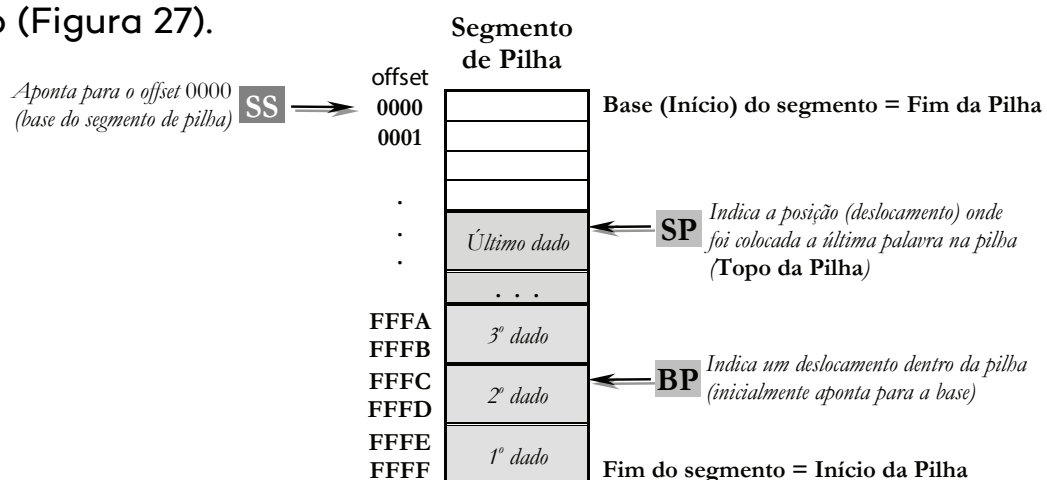


Figura 27 Segmento de pilha

## MODOS DE ENDEREÇAMENTO DE MEMÓRIA

Quando um processador executa uma instrução, ele realiza uma determinada função sobre um ou dois dados. Estes dados, chamados de operandos, podem ser parte da instrução, podem residir em um registrador interno do processador, podem estar armazenados em um endereço de memória, ou podem estar presente em uma porta de E/S. Para acessar estas diferentes localizações dos operandos, o processador apresenta sete modos de endereçamento de dados: endereçamento por registro, endereçamento imediato, endereçamento direto, endereçamento indireto por registro, endereçamento por base, endereçamento indexado, e endereçamento por base indexada. Em geral, as instruções Assembly utilizam dois operandos: um operando deve ser um registrador e o outro operando (dado imediato, offset ou registrador) identifica o modo de endereçamento utilizado.

Devemos entender os modos de endereçamento como as formas possíveis de se manipular dados de/para a memória. Portanto, os casos não incluídos nos endereçamentos apresentados devem ser entendidos como não possíveis, uma vez que provocam erros de compilação.

Para entender as figuras que explicam os modos de endereçamento, deve-se imaginar o comportamento adotado pelo processador para executar uma instrução. Os seguintes passos devem ser seguidos:

- Busca da instrução na memória (segmento de código): Geração do endereço físico correspondente à instrução que será executada (composição CS : IP).
- Decodificação da instrução: Transformação de Assembly para linguagem de máquina.
- Busca dos operandos para execução da instrução: Quando o operando estiver localizado na memória (dado por um deslocamento), geração do endereço físico correspondente ao deslocamento do dado no segmento correspondente (DS, ES ou SS), através da composição registrador de segmento : deslocamento.
- Execução da instrução: Verificação das modificações em



registradores e segmentos de memória.

Algumas observações quanto à modificação em registradores devem ser feitas:

- Os registradores de segmento (CS, DS, ES, SS) não modificam seu valor durante a execução de um programa a menos que seja feito algum redirecionamento.
- O ponteiro de instrução (IP), aponta sempre para a instrução corrente, sendo incrementado automaticamente após a execução de uma instrução. O IP, portanto, não deve receber atribuições num programa.
- O registrador SP aponta sempre para o topo da pilha e seu valor é alterado (incrementado ou decrementado) automaticamente após a execução de instruções para manipulação de pilha.
- Os registradores AX, BX, CX, DX, SI, DI e BP podem receber atribuições e, dessa forma, ter seus valores alterados.
- O registrador de flags (F) pode ser alterado a partir de atribuições (através das instruções para manipulação de flags) ou depois da execução de alguma instrução (flags de estado).

Uma observação importante deve ser feita quanto à utilização de variáveis. As variáveis, largamente utilizadas por linguagens de alto nível, nada mais são que simples representação de um deslocamento de memória. Dessa forma, todas as observações feitas para os modos de endereçamento que manipulam deslocamentos são válidas para utilização com variáveis.

### **Modo de Endereçamento por Registro**

No modo de endereçamento por registro, os dois operandos são registradores. Outro operando: registrador.

Exemplo (Figura 28): MOV AX, BX  
AX ← BX

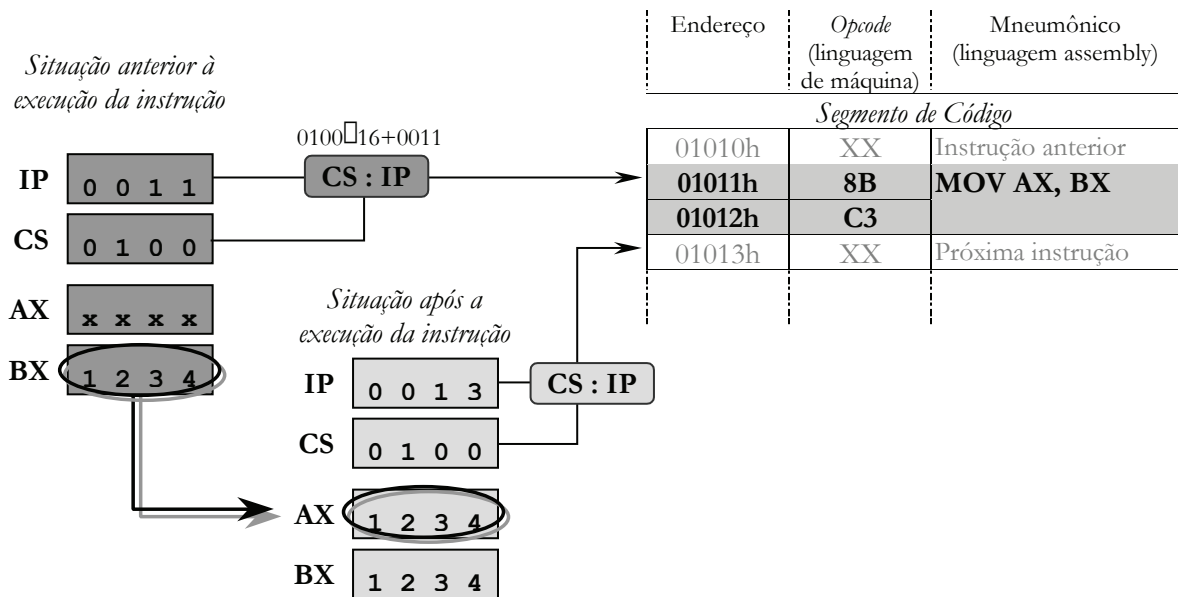


Figura 28 Modo de Endereçamento por Registro

## Modo de Endereçamento Imediato

No modo de endereçamento imediato, um dos operandos está presente no byte seguinte ao código da instrução (opcode). Se bytes de endereçamento seguem o opcode, então o dado a ser transferido de maneira imediata virá logo após os bytes de endereçamento.

Outro operando: número (dado imediato).

Exemplo (Figura 29): MOV AL, 17h  
AL ← 17h

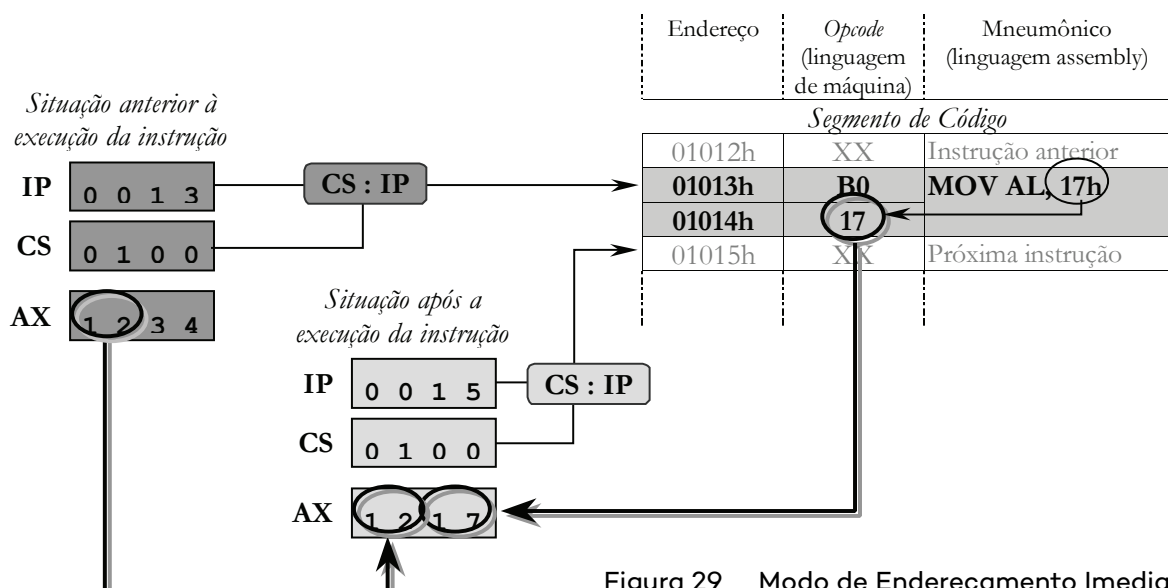


Figura 29 Modo de Endereçamento Imediato

## Modo de Endereçamento Direto

O modo de endereçamento direto é feito somando-se os dois bytes seguintes ao opcode ao DS, para compor um novo endereço absoluto.

Outro operando: deslocamento dado por um número.

O deslocamento pode ser o nome de uma variável, uma vez que uma variável é um rótulo de uma posição de memória de dados.

Exemplo (Figura 30): `MOV BX, [1102h]`  
 $BX \leftarrow (DS : 1102h)$

[ ] indica um deslocamento no segmento de dados (DS).

Para fazer o endereçamento através de ES ou SS, o segmento deve ser especificado explicitamente. Por exemplo, `MOV BX, (ES:1102h)`

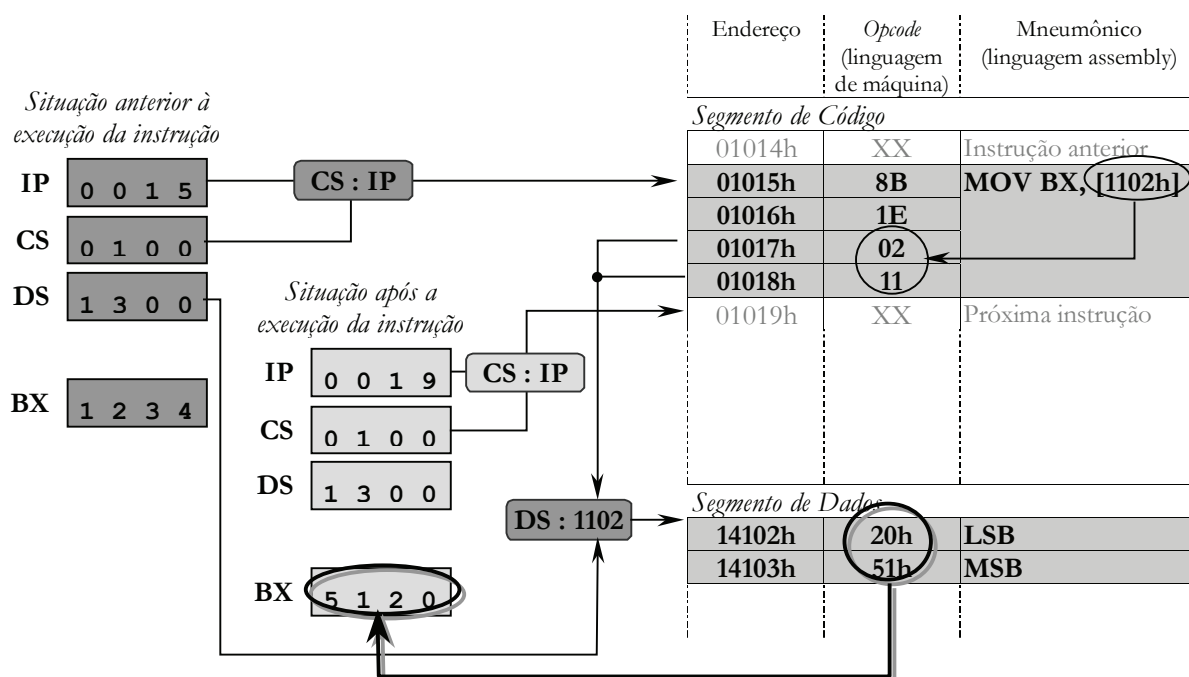


Figura 30 Modo de Endereçamento Direto

## Modo de Endereçamento Indireto por Registro

Nesse modo de endereçamento, qualquer registrador pode ser utilizado.

Outro operando: deslocamento dado por um registrador.

Exemplo (Figura 31): `MOV AX, [BX]`  
 $AX \leftarrow (DS : BX)$

BX aponta (ponteiro) para um dado no segmento de dados (DS).

[BX] – conteúdo do endereço apontado por BX no segmento de dados.

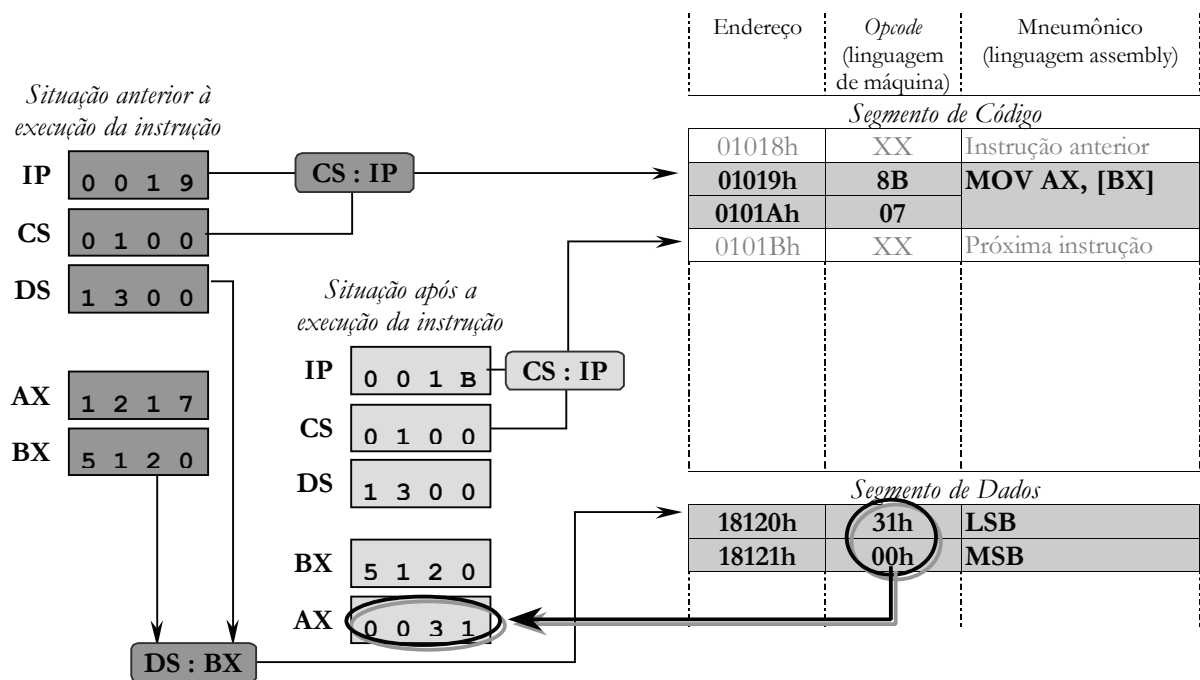


Figura 31 Modo de Endereçamento Indireto por Registro

## Modo de Endereçamento por Base

Utilizado no tratamento de vetores. Apenas os registradores BX ou BP (quando estiver usando a pilha) podem ser utilizados como base.

Os vetores são armazenados no segmento de dados de forma linear, de acordo com a ordem imposta pelos índices correspondentes. Para referenciar um elemento  $v_i = v[i]$  de um vetor  $v$  formado por  $n$  elementos

$$v = \{v_0, v_1, v_2, \dots, v_{n-1}\} = \{v[0], v[1], v[2], \dots, v[n-1]\}$$

o processador utiliza a noção de deslocamento, que depende não somente do índice, mas também da quantidade de bytes de cada elemento. Dessa forma, cada elemento do vetor é representado a partir

do elemento inicial  $v[0]$   $v+0 = v$ , adicionado de um deslocamento. A Figura 32 ilustra a o armazenamento e a representação de vetores com elementos de 1 byte na memória. De forma geral, para um vetor cujos elementos são de  $b$  bytes, o elemento  $i$  será dado por:  $v[i] = v + i \times b$ .

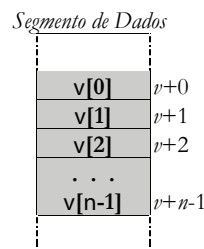


Figura 32 Forma de armazenamento e representação de vetores na memória

Outro operando: deslocamento dado por um registrador (base) + número (deslocamento).

Exemplo (Figura 33):  $MOV\ CX, [BX+0102h]$   $CX \leftarrow (DS : (BX+0102h))$   
 $[BX+0102h] = \text{base (v)} + \text{deslocamento adicional}$   
 deslocamento adicional = índice  $\times$  quantidade de bytes

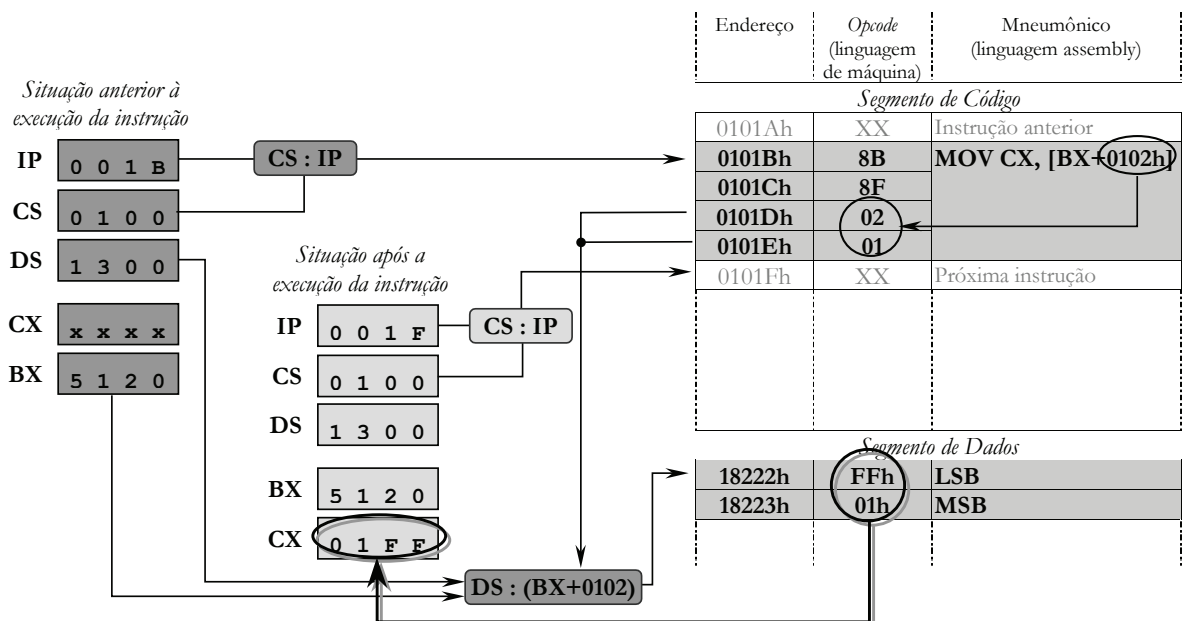


Figura 33 Modo de Endereçamento por Base

## Regras para Especificação de Índices:

As seguintes regras devem ser observadas para endereçamento de dados:

- os índices podem estar em qualquer ordem;
- nomes de registradores devem estar sempre entre colchetes [ ];
- podem ser combinados nomes de registradores e números (endereços constantes) em um único par de colchetes [ ], desde que separados pelo sinal +; e
- se o número vier antes do registrador, não é necessário usar o sinal +.

Notação mais recomendada: variável [registrador de base] [registrador de índice] + constante

Exemplos: A[BX][SI]+8    A[BX][SI+8]    A[8+SI+BX]    [A+SI+BX+8]

Matriz: A[BX][SI]    [A+BX+SI]

Vetor: A[BX] ou A[SI]    [BX+A] ou [A+SI]    [BX+SI] (BX contém o endereço de A)

## Modo de Endereçamento Direto Indexado

Esse modo de endereçamento também é usado para manipulação de vetores e é feito utilizando-se os registradores SI ou DI como indexadores. Um deslocamento é somado a um desses registradores.

Outro operando: deslocamento dado por um número (base) + um registrador de ponteiro de índice (deslocamento adicional).

Exemplo (Figura 34): MOV DX, [0100h+SI]

DX ← (DS : 0100h+SI)

[0100h+SI] = base (v) + deslocamento adicional (índice do elemento × número de bytes)

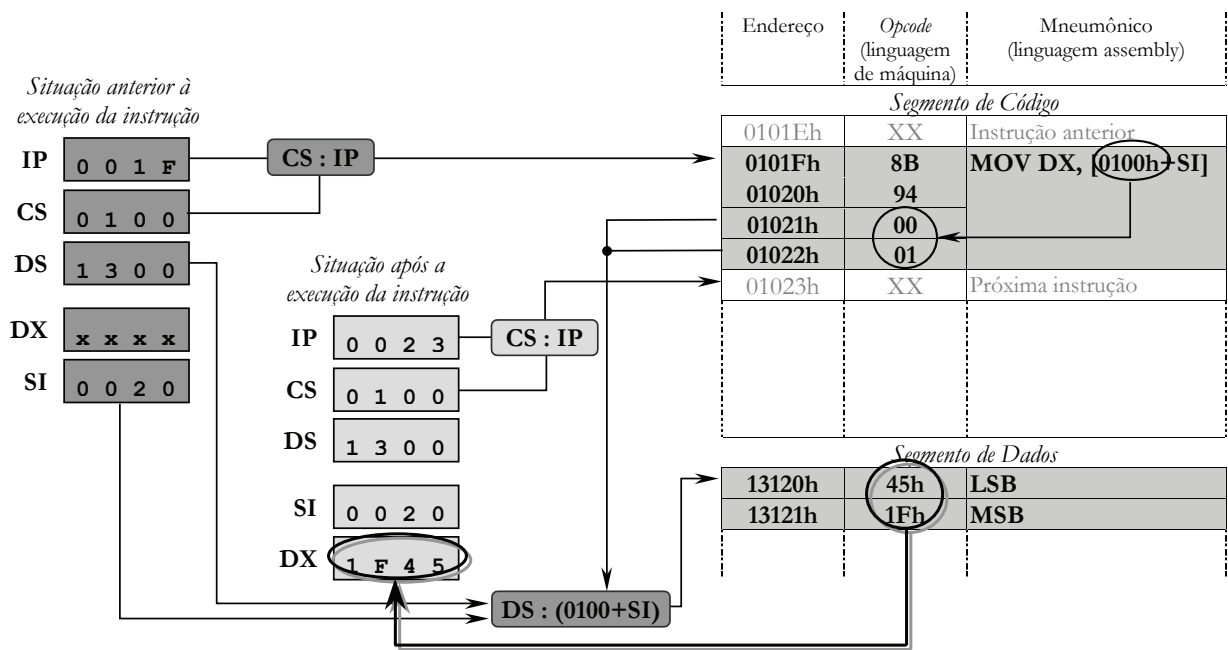


Figura 34 Modo de Endereçamento Indexado

### Modo de Endereçamento por Base Indexada

Utilizado para a manipulação de matrizes na forma  $A_{m \times n}$ . Usa BX ou BP (quando estiver usando a pilha) para indicar o número da linha, e SI ou DI para o número da coluna.

Outro operando: deslocamento dado por um número (base) + um registrador (número da linha) + um registrador de ponteiro de índice (número da coluna), ou, alternativamente, um registrador (base) + um número (número da linha) + um registrador de ponteiro de índice (número da coluna).

As matrizes, apesar de serem entendidas como dados bidimensionais, são armazenadas no segmento de dados de forma linear, linha a linha. Uma matriz  $A$  formada por  $m \times n$  elementos, possui representação matemática dada por

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & & \cdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}_{m \times n} \equiv \begin{bmatrix} a[1][1] & a[1][2] & a[1][3] & \cdots & a[1][n] \\ \cdots & \cdots & \cdots & & \cdots \\ a[m][1] & a[m][2] & a[m][3] & \cdots & a[m][n] \end{bmatrix}_{m \times n}$$

e representação computacional equivalente dada por

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,(n-1)} \\ \cdots & \cdots & \cdots & & \cdots \\ a_{(m-1),0} & a_{(m-1),1} & a_{(m-1),2} & \cdots & a_{(m-1),(n-1)} \end{bmatrix}_{m \times n}$$

ou, alternativamente,

$$A = \begin{bmatrix} a[0][0] & a[0][1] & a[0][2] & \cdots & a[0][n-1] \\ \cdots & \cdots & \cdots & & \cdots \\ a[m-1][0] & a[m-1][1] & a[m-1][2] & \cdots & a[m-1][n-1] \end{bmatrix}_{m \times n}$$

podendo ser representada por um vetor correspondente, composto por m sub-vetores na forma

$$A = \begin{bmatrix} A[0] \\ A[1] \\ \cdots \\ A[m-1] \end{bmatrix}$$

no qual cada sub-vetor  $A[i]$  possui n elementos.

Para referenciar um elemento  $A_{ij} \equiv A[i][j]$  de uma matriz A formada por  $m \times n$  elementos o processador utiliza, da mesma forma que para vetores, a noção de deslocamento. No caso de vetores, apenas um deslocamento adicional é suficiente para caracterizar um elemento. Para matrizes, no entanto, dois deslocamentos são necessários: um para indicar o número da linha e outro para o número da coluna às quais o elemento pertence.

Dessa forma, cada elemento da matriz é representado a partir do elemento inicial  $a[0][0] \equiv A+0+0 = A$ , adicionado de dois deslocamentos (linha+coluna). A Figura 35 ilustra o armazenamento e a representação de matrizes com elementos de 1 byte na memória, bem como o vetor linear correspondente a esta representação.

De forma geral, para uma matriz cujos elementos são de b bytes, o elemento (i, j) será dado por:  $a[i][j] = A + i \times n \times b + j \times b$ .



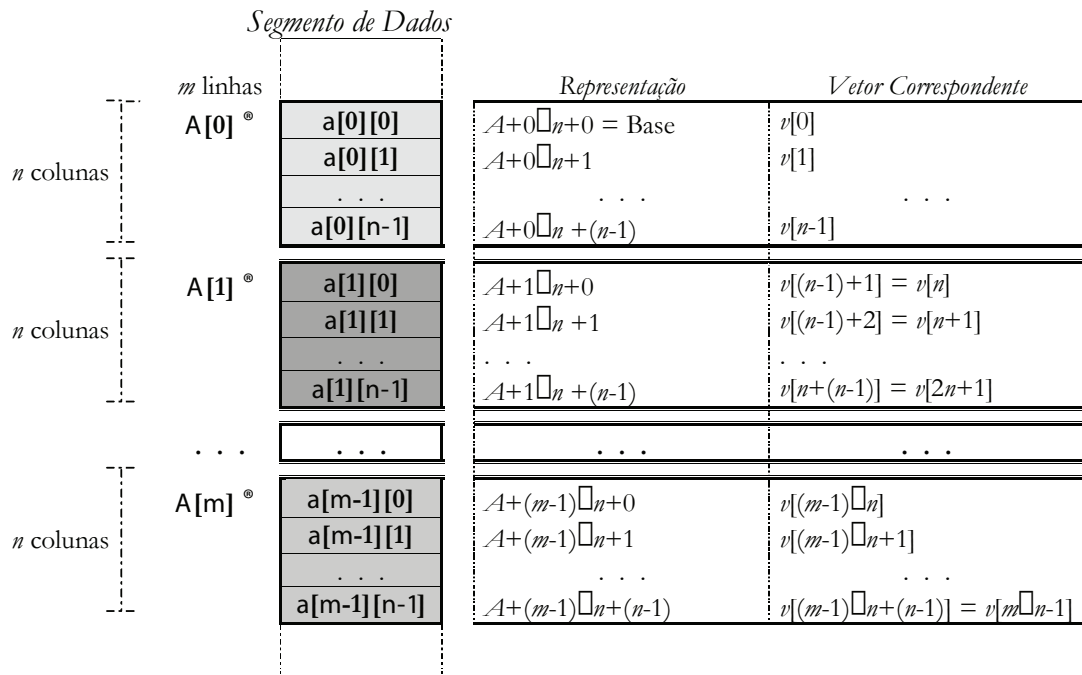


Figura 35 Forma de armazenamento e representação de matrizes na memória

Exemplo (Figura 36): **MOV AH, [BX+0100h+SI]**

**AH ← (DS : (BX+0100h+SI))**

**[BX+0100h+SI] = linha × bytes + base (A) + coluna × bytes = base (A) + linha × bytes + coluna × bytes**

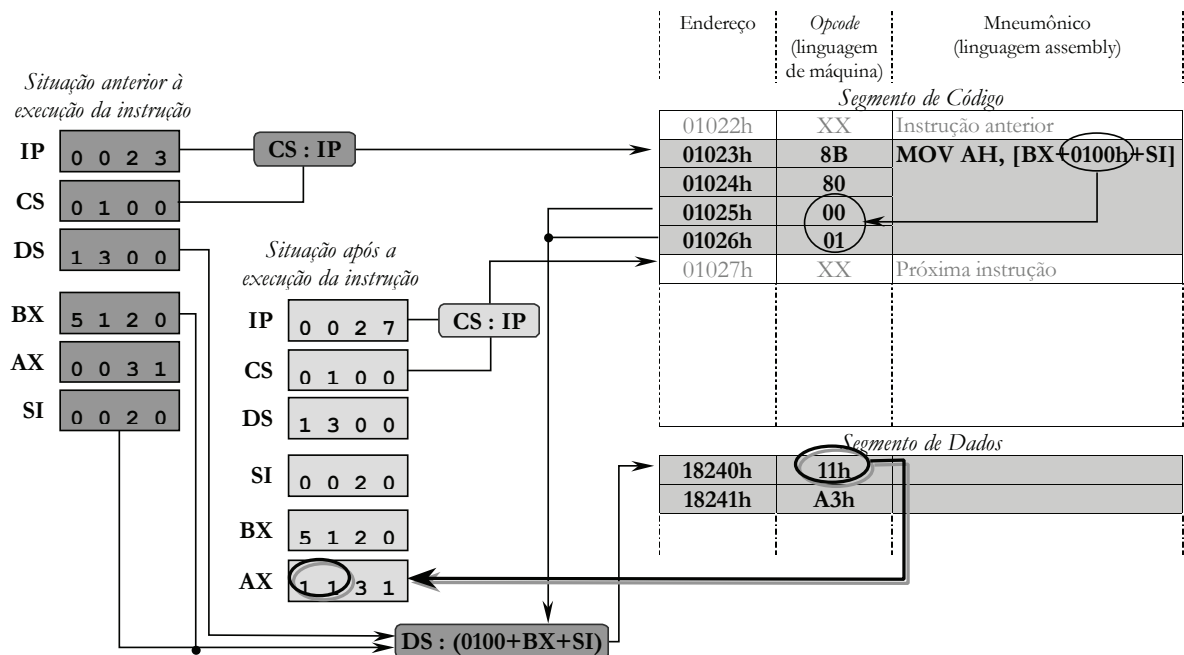


Figura 36 Modo de Endereçamento por Base Indexada

# PROGRAMAÇÃO EM LINGUAGEM ASSEMBLY

## SEGMENTAÇÃO E ESTRUTURA DE PROGRAMAÇÃO (PROGRAMA BÁSICO)

As declarações de um programa fonte escrito em assembly (o que corresponde a cada linha de entrada) podem ser:

- Comentários;
- Instruções assembly (linguagem); ou
- Diretivas do assembler (montador).

Os comentários permitem que explicações sobre determinadas linhas de programa sejam realizadas sem ocasionar erro de compilação. As instruções assembly indicam as ordens que devem ser executadas pela CPU e são transcrições (ou notações) simplificadas, que correspondem aos códigos binários das instruções de máquina. As diretivas do assembler (pseudo-operações) são comandos especiais com o objetivo de facilitar a escrita de um programa sob forma simbólica. Não são incorporadas ao programa objeto (extensão .obj) e servem simplesmente como orientação para o montador. Tanto as instruções assembly quanto as diretivas do assembler podem incluir operadores. Os operadores dão ao assembler informações adicionais acerca dos operandos, nos locais onde possam existir ambigüidades.

Apesar das declarações poderem ser escritas começando em qualquer posição na linha, a Figura 37 ilustra a convenção de alinhamento mais utilizada.

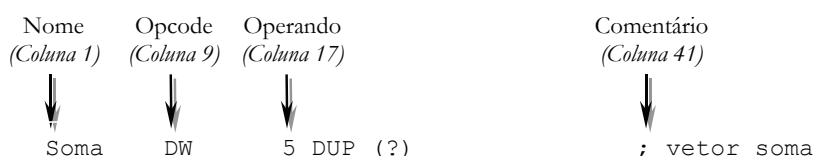


Figura 37 Convenção para alinhamento de declarações

As instruções e diretivas de um programa Assembly podem ser escritas em maiúsculas ou minúsculas (Assembly é uma linguagem não case-sensitive); entretanto, as seguintes sugestões são dadas de forma a tornar o programa mais legível:

- palavras reservadas (instruções e diretivas) devem ser escritas em letras maiúsculas; e
- nomes em geral (comentários e variáveis) podem ser utilizadas letras minúsculas ou maiúsculas, dando-se preferência às minúsculas.

### **Sintaxe dos Comentários**

Os comentários podem ser utilizados de três formas diferentes:

- como uma linha em branco;
- como uma linha iniciada com o caractere ponto e vírgula (;) e seguida de texto; ou
- depois de uma instrução, bastando adicionar o caractere (;) para delimitar o início do comentário.

Exemplos de comentários:

; comentario como uma linha iniciada com ponto e virgula

MOV AX, 1234H ; comentario depois de uma instrucao

### **Sintaxe das Instruções e Diretivas do Assembly**

Para os montadores MASM (Macro ASseMbler) e TASM (Turbo ASseMbler), as instruções devem ser escrita, obrigatoriamente, uma por linha, podendo ter até quatro campos, delimitados de acordo com a seguinte ordem:

[label:] mneumônico [operando(s)] ; comentário]

onde label é o rótulo dado ao endereço da instrução (no segmento de código), mneumônico representa a instrução, operandos são os dados operados pela instrução e comentário é qualquer texto escrito para elucidar ao leitor do programa o procedimento ou objetivo da instrução. Destes, apenas o campo mneumônico é sempre obrigatório.

O campo operandos depende da instrução incluída na linha e os campos label e comentário são sempre opcionais. Todos os valores numéricos estão, por default, em base decimal, a menos que outra base seja especificada.

### Modelo de Programa Assembler Simplificado (.EXE)

```

;                                     definição do modelo desejado
. *****
;
    DOSSEG
    .MODEL modelo

;                                     definição da base numérica desejada
. *****
;
    [.RADIX    base]

;                                     definição do segmento de pilha
. *****
;
    .STACK [tamanho]

;                                     área de definição de equivalências
. *****
;
                                     ; equivalências

;                                     criação do segmento de dados
. *****
;
    .DATA
                                     ; variáveis

;                                     início do segmento de código
. *****
;
    .CODE

;                                     procedimento principal
. *****
;
Principal PROC NEAR                ; início do procedimento principal

```

```

MOV  AX, @DATA      ; instruções para que DS e ES
MOV  DS, AX          ; apontem para a área de
MOV  ES, AX          ; dados criada

```

; corpo do programa principal

```

MOV  AH, 4Ch         ; função para término de programa
INT  21h             ; através da INT 21h

```

```

Principal ENDP        ; final do procedimento principal
END  Principal        ; final do programa

```

Observação: Quando um programa é carregado, os registradores de dados DS e ES não vêm apontando para os segmentos de dados, devendo ser implementado através das instruções (primeira linha do código do programa principal):

```

MOV  AX, @DATA        ; copia para AX o endereço da área
                        de dados
MOV  DS, AX            ; faz DS apontar para a área de dados
MOV  ES, AX            ; faz ES apontar para a área de dados

```

### **Diretivas Simplificadas de Definição de Segmentos**

O compilador implementa uma forma simplificada de definição de segmentos, a qual pode ser usada na maioria dos programas .exe. Muitos defaults assumidos são os mesmos usados pelos compiladores das linguagens de alto nível, facilitando a criação de módulos que serão ligados a programas criados em outras linguagens. Para que o montador assuma uma dada estrutura, é suficiente a definição de um modelo. As principais diretivas utilizadas no modelo simplificado são: DOSSEG, .MODEL, .STACK, .DATA, .CODE.

DOSSEG, .MODEL:

A diretiva `.MODEL` é usada em conjunto com a diretiva `DOSSEG` para definir um modelo e ao mesmo tempo especificar ao ligador (ou linkador, ou linkeditor) que os segmentos devem ser agrupados na ordem convencional adotada pelo sistema operacional para programas escritos em linguagens de alto nível. Sintaxe:

```
DOSSEG
.MODEL [modelo]
```

Os modelos possíveis são:

- **TINY**, no qual um único segmento de 64 kbytes será definido para acomodar o código, a pilha e os dados do programa. Programas com características de `.com`;
- **SMALL**, modelo default, no qual todo o código será agrupado num mesmo segmento de 64KB, enquanto os dados e a pilha estarão num outro segmento de 64KB;
- **MEDIUM**, no qual o código poderá ser maior que 64KB, mas os dados e a pilha estarão num mesmo segmento;
- **COMPACT**, no qual o código é menor que 64KB e dados e pilha podem ser maiores que 64KB;
- **LARGE**, no qual tanto o código quanto os dados podem ser maiores que 64KB.
- **HUGE**, semelhante ao **LARGE**, porém permite a criação de tabelas (matrizes) de dados maiores que 64KB; e
- **FLAT**, onde todos os dados e o código estão num único segmento de 4 Gbytes (disponível apenas para operações em modo protegido).

Modelo de Memória	Atributo para o Código	Atributo para Dados	Sistema Operacional	Segmento Único
TINY	NEAR	NEAR	DOS	Sim
SMALL	NEAR	NEAR	DOS e Windows	Não
MEDIUM	FAR	NEAR	DOS e Windows	Não
COMPACT	NEAR	FAR	DOS e Windows	Não
LARGE	FAR	FAR	DOS e Windows	Não
HUGE	FAR	FAR	DOS e Windows	Não
FLAT	NEAR	NEAR	Windows NT	Sim

Tabela 4 Atribuições default para os modelos de programa

**.STACK:** Permite a criação de um segmento de pilha. O tamanho default é 1 kbyte, podendo ser alterado na própria diretiva.

Sintaxe: `.STACK [tamanho_da_pilha]`

**.DATA:** Marca o início do segmento de dados, no qual todas as variáveis, tabelas e mensagens devem ser colocadas.

Sintaxe: `.DATA`

**.CODE:** Marca o início do segmento de código do programa. No caso de um programa possuir mais de um segmento de código, um nome deve ser especificado para cada segmento.

Sintaxe: `.CODE [segmento]`

**Observação:** A abertura de um novo segmento implica o fechamento do anterior.

### **Operadores de Referência a Segmentos no Modo Simplificado**

Os principais operadores de referência a segmentos utilizados no modelo simplificado são: `@DATA`, `@CODE` e `@CURSEG`.

O operador `@DATA` é usado para referenciar o grupo compartilhado por todos os segmentos de dados, o operador `@CODE` é usado para referenciar o segmento de código e operador `@CURSEG` permite referenciar o segmento corrente.

## **FERRAMENTAS PARA MONTAGEM, LIGAÇÃO E DEPURAÇÃO DE PROGRAMAS**

### **Montador Assembler (TASM)**

**TASM.EXE**      Montador para a linguagem Assembly.

Terminação dos arquivos fonte: `.ASM`

Sintaxe:

`TASM [opções] fonte [,objeto] [,listagem] [,referência_cruzada] [;]`

Linha de Comando mais Comumente Utilizada:

TASM /z/zi programa ↵

Opção      Significado

/z      Display source line with error message

/zi      Debug info: zi=full

## Ligador (TLINK)

TLINK.EXE Ligador – cria um programa executável a partir de um objeto.

Sintaxe:

TLINK [opções] objetos [, executável] [, mapa] [, bibliotecas] [;]

Linha de Comando mais Comumente Utilizada:

TLINK /x/v programa ↵

Opção	Significado
/x	No map file at all
/v	Full symbolic debug information

## Depurador Turbo Debugger (TD)

TD.EXE Depurador – permite a depuração de programas.

Sintaxe:

TD [opções] [programa [argumentos]] -x- (desabilita a opção x)

Linha de Comando mais Comumente Utilizada: TD programa ↵

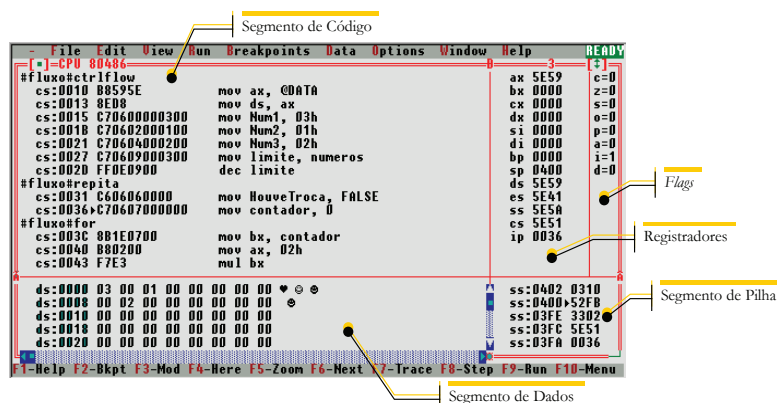


Figura 38 Tela de depuração do Turbo Debugger



Funções:

F2: brkpt (insere um breakpoint)

F4: here (executa até a posição do cursor)

F7: trace (executa linha por linha, sem entrar nos desvios e subrotinas)

F8: step (executa linha por linha, entrando nos desvios e subrotinas)

F9: run (executa até o final do programa ou até um breakpoint)

CTRL-F2: reset (reinicia o programa, sem precisar sair do depurador)

Tela com os Segmentos e Registradores: View, CPU.

Ver o Segmento de dados: Clica com o botão esquerdo do mouse em cima da janela do segmento de dados (para selecionar a janela), clica com o botão direito (aparece uma janela) e seleciona Goto.... O início do segmento de dados é em DS:0000.

## **DIRETIVAS DO ASSEMBLER**

### **Diretivas de Equivalência para o Programa (Definição de Constantes)**

São diretivas que ajudam a documentar melhor o programa, tornando-o claro para quem o estudar.

EQU: Usada para atribuir uma expressão (numérica ou não) a um símbolo. Sempre que este símbolo aparecer no programa, o montador o substituirá pela expressão à qual ele está associado. Um valor fixado por EQU não poderá ser redefinido. A expressão pode ser um número, um caractere, uma string.

Sintaxe: nome EQU expressão

### **Diretiva de Definição de Base Numérica**

.RADIX:

A base default é a decimal, podendo ser modificada através da diretiva .RADIX.

Quando a base fixada como atual for hexadecimal, deve-se ter o cuidado com valores terminados por D ou B. Nestes casos, deve-se necessariamente usar o H após o valor pois a ausência levaria o montador a reconhecer o valor como na base decimal ou binária, o que fatalmente implicaria um erro de sintaxe ou mesmo de lógica.)

Sintaxe: `.RADIX base`

O operando base pode assumir os valores 2 (binária), 8 (octal), 10 (decimal) ou 16 (hexadecimal).

A forma de representar números nas bases possíveis é colocando as letras B (binária),

Q (octal), H (hexadecimal) e D (decimal) depois do número. Exemplo: 0A2h, 34d, 1101b.

### **Diretivas de Definição de Área de Armazenamento de Dados (Variáveis)**

O acesso a uma variável sempre deve ser feito por um registrador de tipo compatível com a definição da variável. As variáveis podem ou não ser inicializadas na definição (utilização do operador ?). Para reservar memória para dados do tipo variáveis, tabelas (matrizes) e mensagens (strings), podem ser utilizadas as diretivas DB, DW, DD, DQ, DT.

**DB:** Para definição de variáveis, tabelas e mensagens, alocando espaço de 1 byte (8 bits) para cada um dos elementos. Os valores podem variar de -128 a 127 (-27 a 27-1) ou de 0 a 255 (0 a 28-1). Principal uso: números pequenos ou caracteres.

**DW:** Para definição de variáveis, tabelas e vetores, alocando 2 bytes (16 bits) para cada elemento. Principal uso: números maiores e endereços (offset). Variação: -215 a 215-1 ou de 0 a 216-1)

**DD:** Para definição de variáveis ou palavras duplas de 4 bytes (32

bits), ou real curto (1 bit mais significativo para o sinal, 8 bits para o expoente e 23 bits para mantissa). Os valores inteiros podem variar de  $-2^{31}$  a  $2^{31}-1$  ou de 0 a  $2^{32}-1$ . Principal uso: endereços completos (segmento : offset).

DQ: Para definição de palavras quádruplas de 8 bytes (64 bits), ou real longo (1 bit mais significativo para o sinal, 11 bits para o expoente e 52 bits para mantissa). Os valores inteiros podem variar de  $-2^{63}$  a  $2^{63}-1$  ou de 0 a  $2^{64}-1$ . Não existe uma forma para acessar os elementos diretamente, sendo necessário um método com uso de endereçamento indireto com registrador. Principal uso: pelo coprocessador matemático.

DT: Para definição de BCD compactado de 10 bytes (9 bytes para valor + 1 byte para sinal). Uso principal: pelo coprocessador matemático.

Sintaxe: [variável] diretiva [tipo PTR] expressão1 [, expressão 2, expressão 3, ...]

O tipo pode ser: BYTE, WORD, DWORD, QWORD, TWORD ou o nome de uma estrutura.

Regras para Especificação de Nomes:

- Nomes podem conter letras, números (dígitos de 0 a 9) e os caracteres ? \_ \$ @
- O primeiro caractere não pode ser um número. Portanto, números devem, obrigatoriamente, começar com um dígito. Exemplo: AAh (nome)  $\neq$  OAAh (número).
- Nomes podem possuir qualquer quantidade de caracteres; entretanto, apenas os 31 primeiros serão utilizados pelo compilador. Logo, nomes devem ser pequenos e significativos.

Atenção! Números começam sempre com dígitos e variáveis começam sempre com letras!

Declaração de Vetores:

Como todas as variáveis, os vetores são declarados através da forma

geral. A diretiva define o tipo dos elementos do vetor e pode ser DB, DW, DD ou DQ, de acordo com a quantidade de bytes desejados. A expressão representa a inicialização dos elementos. A inicialização de vetores pode ser feita de duas formas: a partir da enumeração dos valores, ou de forma genérica, através do operador DUP.

Exemplos:

```
X db 'A', 0Ah, 3d, 0110b
```

Inicializa 5 elementos do vetor X, cada elemento do tipo byte, com valores iniciais 'A' (ASCII), A (hexadecimal), 3 (decimal) e 0110 (binário).

```
Y dw NMAX dup (?)
```

Inicializa NMAX elementos do vetor Y, todos do tipo word e com valor inicial qualquer. NMAX deve ser uma constante.

Declaração de Strings:

As strings são vetores especiais formados por caracteres. Como os caracteres são elementos de 1 byte, a diretiva de declaração de strings deve ser obrigatoriamente DB.

Exemplos:

```
X db 'V','I','N','I','C','I','U','S'
```

```
Y DB "VINICIUS"
```

As duas strings são inicializados com a mesma seqüência de caracteres "Vinicius".

Declaração de Matrizes:

As matrizes, como já foi visto, são conjuntos de vetores, associados em 2 ou mais dimensões. A forma de declarar matrizes é semelhante a de vetores, apenas com uma modificação lógica.

```
X DB 1,5,2,8,3,5
```

Inicializa tanto um vetor de 6 elementos como uma matriz 2×3 ou 3×2 ou 1×6 ou 6×1.

Y DW M DUP (N DUP (?))

Inicializa M×N elementos de uma matriz m×n, todos do tipo word e valor inicial qualquer.

Lembrando que N e M devem ser constantes!

### **Diretivas de Definição de Procedimentos**

No programa assembly, todo o código deve estar dentro de um ou mais procedimentos. Basicamente, um procedimento é uma subrotina, com a particularidade que o próprio módulo principal é tratado como um procedimento pelo sistema operacional.

PROC, ENDP:

O par de diretivas PROC, ENDP é usado para limitar um procedimento e determinar o seu atributo NEAR ou FAR.

Sintaxe:

```
nome_do_procedimento PROC [NEAR ou FAR]
                                ;
                                ;corpo do procedimento
                                ;
nome_do_procedimento ENDP
```

Se um procedimento tem atributo FAR, significa que será chamado com uma instrução CALL inter-segmento; se for NEAR, por um CALL intra-segmento, afetando apenas o IP.

### **Diretivas de Controle do Assembly**

END: Marca o fim de um programa fonte.

Sintaxe:

END [nome\_de\_entrada\_do\_programa]

ORG: Define um novo valor para o contador de locação ou para o apontador de instrução dentro do segmento corrente. Instrui o compilador a começar de um offset maior que zero.

Principal Uso: Em programas de extensão .com, que não possuem áreas de dados, para forçar a primeira instrução a ficar no endereço 100h (256 bytes livres).

Sintaxe:

ORG endereço

## OPERADORES DO ASSEMBLER

### Operadores para Dados

Operadores para Criação de Dados:

? (sinal de interrogação): Usado em conjunto com as diretivas de reserva de área de armazenamento BD, DW, DD e DT, para indicar que, naquela posição, não será definido um valor inicial. Inicialização de variáveis com valores arbitrários (lixo).

DUP: Possibilita a alocação de um dados tantas vezes quanto for o seu prefixo n. Usado na inicialização coletiva de vetore, matrizes e strings. A expressão deve aparecer entre parênteses.

Sintaxe:

n DUP (expressão1, expressão2, ...)

+ - \* / MOD: Operadores aritméticos que realizam, respectivamente,

soma, subtração, multiplicação, divisão inteira e resto da divisão inteira.

Operadores para Referência a Dados: utilizados, geralmente, como operandos de instruções.

TYPE: Retorna um número indicando o tamanho (bytes) de um dado ou do tipo de um símbolo. O número pode ser: 0 (constante), 1 (BYTE), 2 (WORD), 4 (DWORD), 8 (QWORD), 10 (TBYTE), 0FFFFh (NEAR), 0FFFEh (FAR) ou o número de bytes de uma estrutura.

LENGTH: Retorna o número de elementos alocados com uma das diretivas de dados (número de elementos do dado).

Sintaxe:

LENGTH variável

SIZE: Devolve o tamanho do dado alocado com uma das diretivas de dados (número total de bytes alocados = type × length).

Sintaxe:

SIZE variável

SEG: Devolve o endereço do segmento onde o dado está alocado (16 bits).

Sintaxe:

SEG variável

OFFSET: Devolve o deslocamento de uma variável (ou rótulo) dentro do segmento onde ela está definida (distância desde o primeiro byte dentro do segmento). Fornece o deslocamento do dado na memória (offset). Endereço físico do dado = SEG : OFFSET.

É interessante notar que a instrução

MOV BX, OFFSET tabela

é semelhante à instrução

LEA BX, tabela

\$ (dólar): Contador de locação. Representa o endereço de uma instrução ou de um dado. Contém o offset da próxima locação disponível. Utilizado para calcular o comprimento de um dado ou de instruções.

Exemplo:

msg	DB	"Vinicius"	; define uma string msg
tammsg	EQU	\$-msg	; define o tamanho da string msg (tammsg=8)

## **Operadores de Especificação de Tamanho**

Especificação de tamanho para Dados:

BYTE PTR: Força o montador a assumir o dado referenciado como do tamanho de um byte. A expressão precisa ser um endereço.

WORD PTR: Força o montador a assumir o dado referenciado como do tamanho de uma palavra. A expressão precisa ser um endereço.

DWORD PTR: Força o montador a assumir o dado referenciado como do tamanho de uma dupla palavra, respectivamente. A expressão precisa ser um endereço ou nome no programa.

Especificação de tamanho para Código:

FAR PTR: Força uma expressão a gerar um código do tipo inter-segmento. Usado para chamar uma sub-rotina ou provocar um desvio para uma sub-rotina em outros segmento de memória.



NEAR PTR: Usado com instruções JMP e CALL para especificar que a instrução deve gerar um código do tipo intra-segmento.

## CONJUNTO DE INSTRUÇÕES ASSEMBLY

### Instruções para Transferência

MOV: Transfere um byte ou uma palavra (word) de dados de um operando fonte (opf) para um operando destino (opd).

Mneumônico	Formato	Operação
MOV	MOV opd, opf	(opd) $\leftarrow$ (opf)

Variações possíveis de operandos:

Operando destino	Operando fonte
Memória	Registrador
Registrador	Memória
Registrador	Registrador
Registrador	Dado imediato
Memória	Dado Imediato
Registrador de segmento	Registrador
Registrador	Registrador de segmento

XCHG: Permite a troca de um dado presente num operando fonte por um presente num operando destino.

Mneumônico	Formato	Operação
XCHG	XCHG opd, opf	(opd) $\leftrightarrow$ (opf)

Variações possíveis de operandos:

Operando destino	Operando fonte
Memória	Registrador
Registrador	Registrador

XLAT, XLATB: A execução de uma instrução XLATB permite que o conteúdo na posição de memória endereçada por DS : (BX + AL) seja trazido para AL. Na instrução XLAT, pode ser referenciado um outro segmento de dados associado a BX.

Mneumônico	Formato	Operação
XLATB	XLATB	$(AL) \leftarrow ((AL) + (BX) + (DS)0)$
XLAT	XLAT fonte_da_tabela	$(AL) \leftarrow ((AL) + (BX) + (\text{segmento})0)$

LEA: Usada para carregar um registrador específico (reg16) com um offset de 16 bits.

Mneumônico	Formato	Operação
LEA	LEA reg16, offset	$(\text{reg16}) \leftarrow (\text{offset})$

LDS, LES ou LSS: Usadas para carregar, respectivamente, DS, ES ou SS, e um registrador específico com um endereço completo de memória (segmento : offset).

Mneumônico	Formato	Operação
LxS	LxS reg16, offset	$(\text{reg16}) \leftarrow (\text{offset})$ $(xS) \leftarrow (\text{segmento})$

## Instruções Aritméticas

ADD: Permite adicionar operandos de 8 ou de 16 bits.

ADC: Permite adicionar operandos de 8 ou de 16 bits e mais o bit CARRY.

SUB: Permite subtrair operandos de 8 ou de 16 bits.

SBB: Permite subtrair operandos de 8 ou de 16 bits, com empréstimo (bit CARRY).

Mneumônico	Formato	Operação
ADD	ADD opd, opf	$(\text{opd}) \leftarrow (\text{opf}) + (\text{opd})$ $(CF) \leftarrow \text{transporte}$
ADC	ADC opd, opf	$(\text{opd}) \leftarrow (\text{opf}) + (\text{opd}) + (CF)$ $(CF) \leftarrow \text{transporte}$

SUB	SUB opd, opf	$(\text{opd}) \leftarrow (\text{opd}) - (\text{opf})$ $(\text{CF}) \leftarrow \text{empréstimo}$
SBB	SBB opd, opf	$(\text{opd}) \leftarrow (\text{opd}) - (\text{opf}) - (\text{CF})$ $(\text{CF}) \leftarrow \text{empréstimo}$

Variações possíveis de operandos:

Operando destino	Operando fonte
Memória	Registrador
Registrador	Memória
Registrador	Registrador
Registrador	Dado imediato
Memória	Dado Imediato

INC: Permite incrementar operandos de 8 ou de 16 bits.

DEC: Permite decrementar operandos de 8 ou de 16 bits.

NEG: Faz o complemento de 2 do operando destino.

Mneumônico	Formato	Operação
INC	INC opd	$(\text{opd}) \leftarrow (\text{opd}) + 1$
DEC	DEC opd	$(\text{opd}) \leftarrow (\text{opd}) - 1$
NEG	NEG opd	$(\text{opd}) \leftarrow -(\text{opd})$

Variações possíveis de operando destino: memória, registrador de 8 bits ou de 16 bits.

AAA: Faz um ajuste no acumulador AL após uma operação de soma ou de incremento com números BCD não compactados (ASCII).

DAA: Faz um ajuste no acumulador AL após uma operação de soma ou de incremento com números BCD compactados.

AAS: Faz um ajuste no acumulador AL após uma operação de subtração ou de decremento com números BCD não compactados (ASCII).

DAS: Faz um ajuste no acumulador AL após uma operação de subtração ou de decremento com números BCD compactados.

As instruções AAA, DAA, AAS e DAS devem ser incluídas

imediatamente após a operações de adição, subtração, incremento ou decremento de números ASCII ou BCD.

Mneumônico	Formato	Operação
AAA	AAA	$(AL)BCD \leftarrow (AL)2$ $(AH) \leftarrow 0$ , se $(AL)BCD \leq 9$ $(AH) \leftarrow 1$ , se $(AL)BCD > 9$
DAA	DAA	$(AL)BCD \leftarrow (AL)2$
AAS	AAS	$(AL)BCD \leftarrow (AL)2$ $(AH) \leftarrow 0$ , se $(CF) = 0$ $(AH) \leftarrow FFh$ , se $(CF) = 1$
DAS	DAS	$(AL)BCD \leftarrow (AL)2$

MUL, DIV, IMUL, IDIV:

O processador suporta instruções de multiplicação e divisão de números binários (sinalizados ou não) e números BCD. Para operações com números não sinalizados, existem as instruções MUL e DIV, e para números sinalizados, as instruções IMUL e IDIV. O modo de operação das instruções, para números sinalizados ou não, é o mesmo.

Resultado da Multiplicação:

Operando de 8 bits:  $(AX) \leftarrow (AL) \times (\text{operando de 8 bits})$

Operando de 16 bits:  $(DX, AX) \leftarrow (AX) \times (\text{operando de 16 bits})$

Observação: É importante notar que a multiplicação é sempre feita entre operadores de mesmo tamanho (8 ou 16 bits).

Resultado da Divisão:

Numa operação de divisão, apenas o operando fonte é especificado. O outro operando é o conteúdo de AX, para operações com operandos de 8 bits, ou o conteúdo de (DX, AX), para operações com operandos de 16 bits. Se, na divisão, o dividendo for zero ou o quociente não couber no registrador de resultado, é gerada uma interrupção INT 0.

Operando de 8 bits:

$$(AL) \leftarrow \text{Quociente} \left( \frac{AX}{\text{operando de 8 bits}} \right) \quad (AH) \leftarrow \text{Resto} \left( \frac{AX}{\text{operando de 8 bits}} \right)$$

Operando de 16 bits:

$$(AX) \leftarrow \text{Quociente} \left( \frac{DX, AX}{\text{operando de 16 bits}} \right) \quad (DX) \leftarrow \text{Resto} \left( \frac{DX, AX}{\text{operando de 16 bits}} \right)$$

Observação: As instruções CBW e CWD ajustam valores de AL (byte) para AX (word) e de AX (word) para DX : AX (dword), respectivamente.

AAM: Ajusta o resultado do acumulador AL após uma operação de multiplicação com números BCD não compactados, devolvendo-o a AX (AH contém as dezenas e AL as unidades).

Formato de implementação: AAM

AAD: Ajusta o dividendo contido em AX antes da operação de divisão de números BCD não compactados.

Formato de implementação: AAD

Observação: Não existe multiplicação ou divisão para números BCD compactados. Portanto, é necessário descompactá-los antes de realizar essas operações.

## Instruções Lógicas

AND, OR, XOR, NOT: Permitem implementar as operação lógicas AND (e), OR (ou), XOR (ou exclusivo) e NOT (negação). Todas as operações são executadas bit a bit entre os operandos fonte e destino.

Mneumônico	Formato	Operação
AND	AND opd, opf	$(\text{opd}) \leftarrow (\text{opf}) .e. (\text{opd})$
OR	OR opd, opf	$(\text{opd}) \leftarrow (\text{opf}) .ou. (\text{opd})$
XOR	XOR opd, opf	$(\text{opd}) \leftarrow (\text{opf}) .xor. (\text{opd})$
NOT	NOT opd	$(\text{opd}) \leftarrow \overline{(\text{opd})}$

Variações possíveis para o operando destino na instrução NOT: memória e registrador.

Variações possíveis de operandos para as instruções AND, OR e XOR:

<b>Operando destino</b>	<b>Operando fonte</b>
Memória	Registrador
Registrador	Memória
Registrador	Registrador
Registrador	Dado imediato
Memória	Dado Imediato

TEST: Testa o primeiro operando para verificar se um ou mais bits são 1. Utiliza uma máscara para o teste: bits 1 indicam necessidade de teste, ao contrário de bits 0. A instrução não modifica o operando destino. O resultado é atribuído ao flag ZF: se pelo menos um dos bits testados for 1, ZF=1; caso contrário, se todos os bits testados forem zero, ZF=0.

Formato de implementação: TEST destino, máscara

Exemplos:

TEST 10001110b, 10000011b → Serão testados os bits 0, 1 e 7 do operando destino. O resultado da operação será ZF = 1.

TEST 10001110b, 00110001b → Serão testados os bits 1, 4 e 5 do operando destino. O resultado da operação será ZF = 0.

SHL: Retorna o valor da expressão deslocada n bits para a esquerda. Um bit 0 é inserido na posição do bit menos significativo e o bit mais significativo é perdido.

Formato de implementação: SHL destino, n

SHR: Retorna o valor da expressão deslocada n bits para a direita. Um bit 0 é inserido na posição do bit mais significativo e o bit menos

significativo é perdido.

Formato de implementação: SHR destino, n

Observação: Deslocamentos representam multiplicações por potências de 2:

$\text{SHL AX, } n \equiv \text{AX} \leftarrow \text{AX} \times 2^n$        $\text{SHR AX, } n \equiv \text{AX} \leftarrow \text{AX} \times 2^{-n}$

ROL, ROR: Retorna o valor da expressão rotacionada n bits para a esquerda ou direita.

Formato de implementação: ROR destino, n      ROL destino, n

RCL, RCR: Retorna o valor da expressão, acoplado do carry flag à direita ou à esquerda, rotacionada n bits para a esquerda ou direita, respectivamente.

Formato de implementação: RCR destino, n      RCL destino, n

### Instruções que Modificam Flags

STC, CLC, CMC: Modificam os flags de estado.

STC: set carry flag, CLC: clear carry flag, CMC: complement carry flag.

Mneumônico	Formato	Operação
STC	STC	$\text{CF} \leftarrow 1$
CLC	CLC	$\text{CF} \leftarrow 0$
CMC	CMC	$\text{CF} \leftarrow \overline{\text{CF}}$

STD, CLD, STI, CLI: Modificam os flags de controle.

STD: set direction flag, CLD: clear direction flag, STI: set interrupt flag, CLI: clear interrupt flag.

<b>Mneumônico</b>	<b>Formato</b>	<b>Operação</b>
STD	STD	DF $\leftarrow$ 1
CLD	CLD	DF $\leftarrow$ 0
STI	STI	IF $\leftarrow$ 1
CLI	CLI	IF $\leftarrow$ 0

Modificando Flags sem Instruções Específicas:

Modificação de flags através da pilha e de um registrador:

- Coloque todo o registrador de flags na pilha, através do comando PUSHF.
- Salve a pilha para um registrador (16 bits), através do comando POP xx, onde xx representa o nome do registrador.
- Faça modificações através de instruções que utilizem máscaras: AND, OR ou XOR.
- Retorne o conteúdo do registrador para pilha, através do comando PUSH xx.
- Restaure o registrador de flags, utilizando a instrução POPF.

Modificação de flags através da pilha (modo 386 avançado):

- Coloque todo o registrador de flags na pilha, através do comando PUSHF.
- Faça alterações nos flags utilizando comandos BTR (bit test reset), para levá-lo ao estado lógico 0, e BTS (bit test set), para levá-lo ao estado lógico 1.
- Feitas as modificações, restaure o registrador de flags, utilizando a instrução POPF.

### **Instruções de Chamada e Retorno de Subrotinas**

**CALL:** Realiza a chamada à subrotina, salvando IP na pilha e redirecionando para a subrotina.

Formato de implementação:      CALL rotina

Variações possíveis de operandos: subrotina próxima (NEAR) ou distante (FAR), registrador (16 bits) ou memória (16 ou 32 bits).



RET: Retorna da subrotina à rotina original. Quando um operando imediato for utilizado, este valor é adicionado ao conteúdo do SP e permite o descarte de empilhamentos feitos no programa principal, antes da chamada à subrotina.

Formato de implementação: RET

### **Instruções para Manipulação de Pilha**

PUSH: Coloca, na área de memória usada como pilha, o conteúdo de um registrador ou posição de memória. Esta área é endereçada por SS e SP.

Formato de implementação: PUSH fonte

POP: Retira a palavra armazenada no topo da pilha, colocando-a no registrador ou posição de memória especificada (16 bits). O registrador CS não pode ser usado como operando.

Formato de implementação: POP destino

PUSHA: Coloca os valores de todos os registradores, seguindo a ordem AX, CX, DX, BX, SP, BP, SI e DI.

Formato de implementação: PUSHA

POPA: restaura os valores dos registradores salvos com PUSHA, seguindo a ordem DI, SI, BP, SP, BX, DX, CX e AX (ordem contrária à salva por PUSHA).

Formato de implementação: POPA

PUSHF: Coloca, na área de memória usada como pilha, o conteúdo do registrador de flags. Esta instrução é usada para salvar na pilha os estados dos flags, alterá-los e depois recuperá-los.

Formato de implementação: PUSHF

POPF: Retira a palavra no topo da pilha e move-a para o registrador de flags.

Formato de implementação: POPF

Instrução NOP

NOP: Nenhuma operação é executada. Normalmente, usa-se NOP para preencher um bloco de memória com código nulo, isto é, com algo que não afetará os estados do processador.

Formato de implementação: NOP

### **Instruções de Entrada e Saída**

O endereço da porta de entrada ou saída pode ser especificado por um byte da própria instrução, permitindo acesso a 256 portas diferentes, ou através de DX, que permite o acesso a 64K endereços diferentes. Para executar as instruções de E/S é necessário certificar-se dos endereços de hardware, consultando uma tabela de endereços de E/S. A vantagem no uso de DX, além do acesso a um número maior de portas, é que, incrementando-o, pode-se acessar sucessivas portas dentro de um laço de repetição.

IN: Transfere o byte de dados presente no endereço de porta especificado em DX para AL.

Formato de implementação: IN AL, DX

OUT: Transfere o byte presente em AL para um endereço de porta especificado em DX.

Formato de implementação: OUT DX, AL

### **5.5.9 Instrução de Comparação**

CMP: Compara dois valores

<b>Mneumônico</b>	<b>Formato</b>	<b>Operação</b>
CMP	CMP op1, op2	op1 – op2

A instrução CMP trabalha subtraindo o segundo operando do primeiro, alterando os flags de estado envolvidos, e sem modificar qualquer um dos operandos. A instrução CMP é seguida, normalmente, de instruções de jumps condicionais para testar os resultados da comparação.

Modo de Operação:

CMP op1, op2     $\equiv$     op1 – op2     $\neq$     SUB op1, op2

Variações possíveis dos operandos:

<b>op1</b>	<b>op2</b>
Memória	Registrador
Registrador	Memória
Registrador	Registrador
Registrador	Dado imediato
Memória	Dado Imediato

Restrições ao uso de CMP:

- A comparação deve ser feita sempre entre dois números sinalizados ou entre dois números não sinalizados; nunca misture-os.
- Ambos os operandos devem ter o mesmo comprimento, BYTE ou WORD.
- Lembre-se que comparações entre duas memórias não podem ser feitas. Caso isso seja necessário, deve-se primeiro copiar um dos operandos para um registrador, e só então executar a comparação entre o conteúdo do registrador e o outro operando.

## Instruções de Desvio

Existem dois tipos de instruções de desvio: incondicional e condicional.

No desvio incondicional, o programa é automaticamente desviado para a nova posição de execução do programa. No desvio condicional, o desvio só ocorrerá se uma dada condição testada nos flags internos for verdadeira. Se a condição testada não for verdadeira, o programa executará a instrução localizada na posição seguinte da memória.

Um jump (desvio) é uma instrução que diz ao processador para continuar a execução do programa em um endereço dado por um label. Os labels podem conter um endereço que está num mesmo segmento de código ou em outro segmento de código. Em programas estruturados, o uso de jumps deve se restringir a um mesmo segmento de código, para tornar o programa mais legível e menos susceptível a erros.

Existem duas situações importantes em que os jumps devem ser usados:

- para implementar laços ou loops; ou
- em caso de ocorrerem condições excepcionais na execução do programa.

**JMP – Instrução de Desvio Incondicional:**

Um jump incondicional é um jump que é sempre executado, independente de uma condição específica ser verdadeira ou falsa. A instrução JMP deve ser usada em dois casos:

- para pular instruções que não serão executadas; e
- para auxiliar a execução de loops.

Formato de implementação:      JMP destino

Variações possíveis para o operando destino: label curto (próximo ou distante), registrador (16 bits) ou memória (32 bits).

**Instruções de Desvio Condicional:**

Um desvio condicional é um jump executado somente se uma condição específica é verdadeira. Existem dois tipos básicos de

jumps:

- jumps que testam flags e registradores; e
- jumps usados após comparações (para números sinalizados e não sinalizados).

Formato de implementação para jumps que testam flags e registradores:

Jxxx label\_destino

onde xxx é uma abreviação para uma condição particular. Se a condição é verdadeira, o processador continua a execução a partir da primeira linha de comando após o label; caso contrário, a execução continua na próxima instrução da seqüência. Observação: o label destino deve estar a, no máximo, 128 bytes acima ou abaixo do local do jump.

O processador tem dois conjuntos de jumps condicionais que podem ser usados após uma comparação. Os conjuntos são similares; a única diferença é que um é usado com números sinalizados e o outro com números não sinalizados.

Formato de implementação do conjunto comparação/instrução de jumps condicionais:

CMP    op1, op2  
Jxxx   label\_destino  
         ...            ; instruções da seqüência  
         label\_destino:

Mneumônico	Formato	Operação
------------	---------	----------

Jumps condicionais utilizados após comparação (CMP) para números não sinalizados

JA/JNBE	JA destino	Jump para destino se acima (CF=0, ZF=0)
JAE/JNB	JAE destino	Jump para destino se acima ou igual (CF=0)
JB/JNAE	JB destino	Jump para destino se abaixo (CF=1)

JBE/JNA	JBE destino	Jump para destino se abaixo ou igual (CF=1 ou ZF=1)
JE	JE destino	Jump para destino se igual (ZF=1)
JNE	JNE destino	Jump para destino se diferente (ZF=0)

Jumps condicionais utilizados após comparação (CMP) para números sinalizados

JG/JNLE	JG destino	Jump para destino se maior (ZF=0 e SF=OF)
JGE/JNL	JGE destino	Jump para destino se maior ou igual (SF=OF)
JL/JNGE	JL destino	Jump para destino se menor (SF≠OF)
JLE/JNG	JLE destino	Jump para destino se menor ou igual (ZF=1 ou SF≠OF)
JE	JE destino	Jump para destino se igual (ZF=1)
JNE	JNE destino	Jump para destino se diferente (ZF=0)

Mneumônico	Formato	Operação
------------	---------	----------

Jumps condicionais que testam flags e registradores

JZ	JZ destino	Jump para destino se ZF=1
JNZ	JNZ destino	Jump para destino se ZF=0
JS	JS destino	Jump para destino se SF=1
JNS	JNS destino	Jump para destino se SF=0
JO	JO destino	Jump para destino se OF=1
JNO	JNO destino	Jump para destino se OF=0
JC	JC destino	Jump para destino se CF=1
JNC	JNC destino	Jump para destino se CF=0
JP/JPE	JP destino	Jump para destino se paridade par (PF=1)
JNP/JPO	JNP destino	Jump para destino se paridade ímpar (PF=0)
JCXZ	JCXZ destino	Jump para destino se (CX)=0

Variações possíveis para o operando destino: label curto (próximo ou distante).

### Instruções de Repetição

LOOP: Cria um laço que é executado um número específico de vezes, usando o registrador CX como contador. A primeira instrução a ser executada para a obtenção de um loop é o carregamento de CX com

o número de execuções do loop.

Formato de implementação do conjunto inicialização de CX/instrução de loops:

```

                MOV CX, número_de_vezes
nome_do_label:
                ...                ; instruções a serem executadas
                LOOP nome_do_label

```

O valor de CX é decrementado a cada passo e o loop será executado enquanto o conteúdo de CX for não nulo.

LOOPE (LOOPZ) e LOOPNE (LOOPNZ):

O princípio de utilização é o mesmo da instrução LOOP. A diferença principal é que a condição para que o laço seja executado inclui não somente o valor do conteúdo de CX, mas o resultado de uma comparação, executada pela instrução imediatamente anterior.

Formato de loops gerados a partir de LOOPE (LOOPZ) e LOOPNE (LOOPNZ):

```

                MOV      CX, número_de_vezes
nome_do_label:
                ...      ; instruções a serem executadas
                CMP      op1, op2
                LOOPxx   nome_do_label

```

## PROGRAMAÇÃO ESTRUTURADA EM ASSEMBLY

A programação estruturada em assembly pode ser feita a partir da utilização de estruturas de controle de fluxo.

### Ferramentas Utilizadas em Controle de Fluxo

Para a utilização de controle de fluxo, as seguintes ferramentas de software são necessárias:

- registrador flags: flags de estado e de controle e as instruções associadas;
- labels (rótulos);
- jumps (saltos): instruções de desvios condicionais (Jxxx) e incondicionais (JMP);
- instrução de comparação: CMP; e
- instruções de repetição: LOOP, LOOPE (LOOPZ), LOOPNE (LOOPNZ).

Sintaxe para Definição de Labels:

Para implementar controle de fluxo, geralmente utilizamos labels, que são marcadores de endereço dentro de um segmento de código. É possível controlar a execução de um programa, apenas promovendo jumps entre labels. As regras para especificação dos nomes dos labels são as mesmas das variáveis.

O assembly reconhece intrinsecamente definições de labels do tipo NEAR, apenas através da seguinte linha de comando, dentro do código do programa:

nome\_do\_label:

### **Estrutura Se-Então-Senão**

A estrutura se-então-senão testa uma condição. Se esta condição for satisfeita, a seqüência de instruções referentes ao então é executada; caso contrario, o corpo do senão é executado. O comando senão não é essencial, podendo ser suprimido. A diferença é que se nenhuma condição for satisfeita, nenhuma seqüência de instruções será executada.

Formato de Implementação:



*Linguagem de Alto Nível*

```

Se (cond), então
~~~~~
~~~~~

Senão
=====

Fim se

```

*Linguagem Assembly*

```

CMP cond
Jcond entao
JMP senao
entao:
~~~~~
~~~~~
JMP fim_se
senao:
=====
=====
fim se:

```

```

CMP cond
JNcond senao
entao:
~~~~~
~~~~~
JMP fim_se
senao:
=====
=====
fim_se:

```

**Estrutura Repita-Até que**

A estrutura repita-até que executa a mesma seqüência de instruções até que uma condição se torne verdadeira.

Formato de Implementação:

*Linguagem de Alto Nível*

```

Repita
=====
Até que (cond)

```

*Linguagem Assembly*

```

repita:
=====
CMP cond
Jcond fim_repita
JMP repita
fim_repita:

```

```

repita:
=====
CMP cond
JNcond repita
fim_repita:

```

**Estrutura Repita-Enquanto**

A estrutura repita-enquanto é semelhante ao repita-até que. Entretanto, uma seqüência de instruções é realizada enquanto uma condição for satisfeita.

Formato de Implementação:

*Linguagem de Alto Nível*

```

Repita
=====
Enquanto (cond)

```

*Linguagem Assembly*

```

repita:
=====
CMP cond
JNcond fim_repita
JMP repita
fim_repita:

```

```

repita:
=====
CMP cond
Jcond repita
fim_repita:

```

## Estrutura Enquanto-Repita

A estrutura enquanto-repita é semelhante à estrutura repita-enquanto, diferindo apenas no teste da condição e na quantidade de vezes que será executado. A condição é testada antes que qualquer seqüência de instruções seja executada. Isto significa dizer que um enquanto-repita pode nunca ser executado, enquanto um repita-enquanto será executado pelo menos uma vez.

Formato de Implementação:

### *Linguagem de Alto Nível*

```
Enquanto (cond), repita
=====
Fim enquanto
```

### *Linguagem Assembly*

```
enquanto:
    CMP cond
    Jcond repita
    JMP fim_enquanto
repita:
    =====
    =====
    JMP enquanto
fim_enquanto:
```

```
enquanto:
    CMP cond
    JNcond fim_enquanto
repita:
    =====
    =====
    JMP enquanto
fim_enquanto:
```

## Estrutura For (Para)

A estrutura para executa a mesma seqüência de instruções um número de vezes predefinido.

Formato de Implementação:

### *Linguagem de Alto Nível*

```
para contador de 1 a n, faça
=====
Fim para
```

### *Linguagem Assembly*

```
MOV contador, 1
para:
    =====
    =====
    CMP contador, n
    JE fim_para
    JMP para
fim_para:
```

```
MOV contador, 1
para:
    =====
    =====
    CMP contador, n
    JNE para
fim_para:
```

### *Linguagem de Alto Nível*

```
para contador de n a 1, faça
=====
Fim para
```

### *Linguagem Assembly*

```
MOV CX, n
para:
    =====
    =====
    LOOP para
fim_para:
```

## Estrutura Case

A estrutura de seleção múltipla case testa valores sucessivos de condições, até que uma delas seja verdadeira. Caso nenhuma condição seja satisfeita, a seqüência de comandos equivalente ao senão será executada. O comando senão não é essencial, podendo ser suprimido. A diferença é que se nenhuma condição for satisfeita, nenhuma instrução será executada.

### Formato de Implementação:

#### *Linguagem de Alto Nível*

```

Case
  cond_1:
    ~~~~~
    ~~~~~
    ...
  cond_n:
    _____
    _____
  Senão:
    .....
    .....
Fim case
  
```

#### *Linguagem Assembly*

```

case:
    CMP (cond_1)
    Jcond_1 caso_1
    ...
    CMP (cond_n)
    Jcond_n caso_n
    JMP senao
caso_1:
    ~~~~~
    ~~~~~
    JMP fim_case
    ...
caso_n:
    _____
    _____
    JMP fim_case
senão:
    .....
    .....
fim_case:
  
```

# INTERRUPÇÕES E EXCEÇÕES

As interrupções são sinais enviados ao microprocessador, através dos quais tarefas sendo executadas são suspensas temporariamente e é atendida uma outra tarefa que necessita de uma atenção imediata. Muitas interrupções são destinadas a tarefas comuns que praticamente todos os softwares necessitam, como por exemplo a obtenção de digitações feitas no teclado ou a abertura de arquivos.

Os tipos de interrupções possíveis no processador são:

- as interrupções por hardware: a INTR (interrupção mascarável) e a NMI (interrupção não mascarável);
- a TRAP, quando ocorre um pedido de execução de programa em passo único (trap flag ativado: TF = 1);
- a INTO, quando verificado um erro de transbordamento (overflow);
- a de erro de divisão por zero; e
- as interrupções por software do tipo INT n.

A interrupção por hardware é iniciada pelos circuitos existentes na placa do sistema, por uma placa expansão ou através de uma porta conectada a um dispositivo externo. As interrupções por hardware podem ser iniciadas por eventos tão diversos como um pulso do chip do timer do computador, um sinal vindo de um modem ou o pressionar de um botão do mouse. A interrupção do teclado é um exemplo típico de interrupção por hardware. Existe um circuito controlador do teclado na placa de sistema do computador que monitora o teclado para receber entrada de dados. O controlador do teclado gera a interrupção 09H todas as vezes que receber um byte de dados (código correspondente a tecla pressionada). O BIOS possui uma rotina de tratamento para a interrupção 09H, cuja finalidade é ler o byte de dados a partir do controlador de teclado, processando-o em seguida. Se a interrupção do teclado não processar o código de tecla recebido no momento da chegada, o código pode ser perdido quando outra tecla for pressionada.

Por outro lado, as interrupções por software serão iniciadas através de programas do usuário, e não pelo hardware do computador. Um programa chama as interrupções por software para poder realizar suas tarefas, como por exemplo apresentar um caractere na tela.

Algumas interrupções executam mais de uma tarefa, e, ao serem chamadas, deve ser especificado um número de função. Por exemplo, a função 00h da interrupção 01h retorna a contagem da hora do dia existente no BIOS do computador, e a função 01h da mesma interrupção ajusta esta contagem. Os números das funções são quase sempre colocados no registrador AH do processador antes da chamada da interrupção. Na Linguagem Assembly pode-se chamar a função 0Ah (apresentar caractere) da interrupção 10h do BIOS, escrevendo:

```
MOV  AH,0Ah
INT  10h           ;chama a função 0Ah
```

Para organizar ainda mais as interrupções, podem existir números de sub-funções para especificar tarefas no interior de uma função. Estes números são colocados no registrador AL do processador. Podemos dizer que as interrupções estão organizadas obedecendo à seguinte hierarquia:

- número da interrupção;
- funções (ou serviços), geralmente especificados em AH; e
- sub-funções, geralmente especificados em AL.

## VETORES E DESCRITORES DE INTERRUPÇÃO

As interrupções são numeradas e relacionadas em uma tabela, onde, a cada interrupção está associado um único endereço segmentado de 4 bytes, chamado de vetor de interrupção, que aponta para uma rotina de tratamento da interrupção (Figura 39). O programa principal, ao se deparar com uma interrupção, suspende temporariamente a sua execução, para processar a rotina de tratamento da interrupção em foco e, logo após, retornar o processamento para o programa de

origem da interrupção.

Dessa forma, quando chega um pedido de interrupção a seguinte seqüência de operações é executada:

- a instrução que está sendo executada é terminada;
- os registradores de flags (F), segmento de código (CS) e de offset (endereço) da próxima instrução (IP) são colocados na pilha;
- outros registradores alterados pela rotina manipuladora de interrupção são colocados na pilha (operação realizada pela rotina manipuladora, quando necessário);
- o endereço da rotina de manipulação é determinado; e
- a rotina de manipulação é executada.

As informações de CS:IP e o registrador de flags são automaticamente inseridos na pilha ao ocorrer uma interrupção. Os outros registradores são salvos pela rotina da interrupção, que é comumente chamada de rotina manipuladora da interrupção (interrupt handler) ou de rotina de serviço da interrupção (interrupt service routine – ISR). Somente serão salvos os registradores que forem alterados pelas rotinas manipuladoras de interrupção. Quando uma rotina manipuladora de interrupção terminar o seu serviço, a mesma restaurará os registradores que tiver alterado, executando no final da rotina a instrução IRET (interrupt return) que possui a função de retirar o registrador de flags e o CS:IP da pilha, retornando-os aos seus lugares no processador. Com o CS:IP restaurado, o programa principal volta a funcionar.

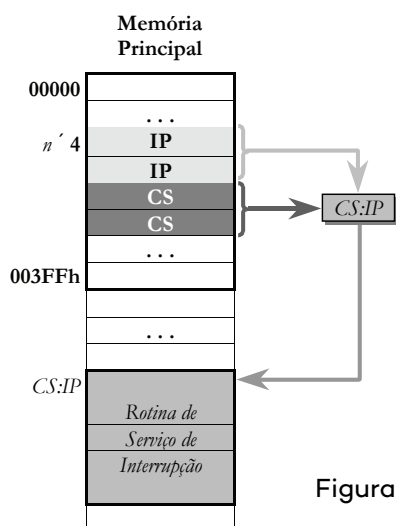


Figura 39 Endereçamento da Rotina de Serviço de Interrupção

Uma tabela de ponteiros é usada para ligar o número-tipo de uma interrupção com as localidades na memória das suas rotinas de serviço. Em um sistema baseado no modo real, esta tabela é chamada de tabela de vetores de interrupção. Em sistemas no modo protegido, esta tabela é chamada de tabela de descritores de interrupção.

No modo real, a tabela de vetores está localizada no limite inferior dos endereços de memória. Ela começa no endereço 0000016 e termina no endereço 003FF16. Isto representa o primeiro kbyte de memória. Na realidade, a tabela de vetores de interrupção pode estar localizada em qualquer lugar na memória. Sua localização inicial e tamanho são definidos pelo conteúdo do registrador da tabela de descritores de interrupção, o IDTR. Quando o processador é inicializado ele entra no modo real com os bits do endereço de base no IDTR todos iguais a zero e o limite fixado em 03FF16. Na tabela de vetores de interrupção, cada um dos 256 vetores requer duas palavras. A palavra de endereço mais alto representa o endereço de base de segmento e identifica na memória, o início do segmento de programa no qual a rotina de serviço reside. Este valor será armazenado no registrador de segmento de código CS no momento do atendimento de uma interrupção. A palavra de endereço mais baixo do vetor representa o deslocamento da primeira instrução da rotina de serviço a partir do início do seu segmento de código e, será armazenado no registrador apontador de instrução – IP.

<b>Endereço físico</b>	<b>Ponteiro de interrupção presente</b>
0000h – 0003h	INT 0 – erro de interrupção
0004h – 0007h	INT 01h – passo único
0008h – 000Bh	INT 02h – NMI
000Ch – 000Fh	INT 03h – breakpoint
0010h – 0013h	INT 04h – overflow
0014h – 0017h	INT 05h – print screen (imprime tela)
0018h – 001Bh	INT 06h – reservada
001Ch – 001Fh	INT 07h – reservada
0020h – 0023h	INT 08h – IRQ0
0024h – 0027h	INT 09h – IRQ1
0028h – 002Bh	INT 0Ah – IRQ2
002Ch – 002Fh	INT 0Bh – IRQ3

0030h – 0033h	INT 0Ch – IRQ4
0034h – 0037h	INT 0Dh – IRQ5
0038h – 003Bh	INT 0Eh – IRQ6
003Ch – 003Fh	INT 0Fh – IRQ7
0040h – 0043h	INT 10h – funções da BIOS (interrupções por software)
...	
0084h – 0087h	INT 21h – funções do DOS (interrupções por software)
...	
01BCh – 01BFh	INT 6Fh – disponível ao usuário (interrupções por software)
01C0h – 01C3h	INT 70h – IRQ8
01C4h – 01C7h	INT 71h – IRQ9
01C8h – 01CBh	INT 72h – IRQ10
01CCh – 01CFh	INT 73h – IRQ11
01D0h – 01D3h	INT 74h – IRQ12
01D4h – 01D7h	INT 75h – IRQ13
01D8h – 01DBh	INT 76h – IRQ14
01DCh – 01DFh	INT 77h – IRQ15
01E0h – 01E3h	INT 78h – disponível ao usuário (interrupções por software)
...	
00FCh – 03FFh	INT FFh – disponível ao usuário (interrupções por software)

Tabela 5 Vetores de interrupção

A tabela de descritores do modo protegido pode residir em qualquer localização do endereço físico do processador. A localização e o tamanho desta tabela são novamente definidos pelo conteúdo do IDTR. O endereço de base no IDTR identifica o ponto inicial da tabela na memória e o limite determina o número de bytes da tabela. A tabela de descritores de interrupção contém descritores de porta (gates), não vetores. São possíveis 256 descritores de interrupção identificados do gate 0 ao gate 255. Cada descritor de gate pode ser do tipo: gate de trap, gate de interrupção, ou gate de tarefa. Os dois primeiros tipos permitem que o controle seja transferido para uma rotina de serviço que está localizada dentro da tarefa atual. Por outro lado, o gate de tarefa permite que o controle de programa seja passado para uma outra tarefa. Como nos vetores de modo real, um gate no modo protegido age como um ponteiro que é usado



para redirecionar a execução do programa para o ponto inicial da rotina de serviço. Entretanto, um descritor de porta ocupa 8 bytes de memória. Se todas as 256 portas não forem necessárias a uma aplicação o limite pode ser fixado em um valor menor que 07FF16.

A Tabela 5 que contém os vetores de interrupção, aloca os vetores de 0 a 256. Cada vetor é composto de uma quantidade de 4 bytes e indica uma posição de memória (CS:IP), onde se inicia a rotina de tratamento.

### **INTERRUPÇÃO POR SOFTWARE: COMANDOS INT E IRET**

As interrupções do software fazem parte dos programas que compõem o ROM BIOS e o sistema operacional. Os serviços do BIOS e do DOS podem ser chamados pelos programas através de uma série de interrupções, cujos vetores são colocados na tabela de vetores de interrupções. As funções do DOS possibilitam um controle mais sofisticado sobre as operações de I/O do que seria possível com as rotinas do BIOS, em especial quando se tratar de operações de arquivos em discos. Convém lembrar que os números mais baixos da tabela de vetores são reservados para as exceções.

A utilização de uma interrupção dentro de um programa não oferece grandes dificuldades, desde que sejam fornecidos todos os dados de entrada nos respectivos registradores; após o processamento da interrupção, obtêm-se os resultados desejados através dos registradores de saída ou através da execução de uma tarefa desejada.

Por outro lado, pode-se desejar a criação de uma rotina de serviço de interrupção para tratar uma necessidade específica do programa. Muitos motivos justificam a criação de uma rotina de serviço de interrupção. Determinados vetores de interrupções são planejados para que sejam redirecionados para rotinas de nossa criação. Pode também ser interessante criar uma interrupção para substituir uma rotina manipuladora já existente, ou podemos até mesmo acrescentar

novos recursos a um manipulador.

Uma rotina manipuladora de interrupções é uma subrotina comum que realiza sua tarefa e encerra a operação com a instrução IRET, além de salvar os registradores alterados por ela, que não sejam aqueles que foram salvos automaticamente. Uma vez criada uma rotina, seu endereço deve ser colocado no local apropriado de vetor; se no local escolhido houver um valor diferente de zero, o valor encontrado deve ser salvo, para possibilitar a sua restauração após o término do programa.

INT: Altera o fluxo normal de execução do programa, desviando-se para uma rotina de interrupção, cujo vetor (endereço formado por CS:IP) está presente numa tabela nos primeiros 1 kbyte da memória. A tabela de vetores de interrupção tem 256 entradas, cada qual com 4bytes (os dois primeiros contendo o valor de IP e os dois seguintes, o valor de CS) que indicam o endereço de uma rotina na memória.

Formato de implementação: INT tipo

Uma interrupção de software é implementada através da instrução INT n (n variando de 0 a 255, em decimal; ou de 0 a FF, em hexadecimal). Através deste mecanismo é possível ao usuário fazer chamadas de funções típicas do sistema operacional (DOS) ou do sistema de entrada/saída (BIOS).

Sempre que um pedido de interrupção válido ocorre (por hardware ou por software), o processador aponta para um vetor presente no espaço de memória que vai de 0000h a 03FFh (primeiro 1 kbyte da memória principal).

As interrupções e exceções do sistema genérico são servidas com um sistema de prioridade conseguida de duas maneiras:

- a sequência de processamento da interrupção implementada no processador testa a ocorrência dos vários grupos baseados na hierarquia de apresentação (reset sendo a de

maior prioridade e as interrupções de hardware sendo as de menor prioridade);

- às várias interrupções dentro de um grupo, são dados diferentes níveis de prioridade, de acordo com seu número-tipo onde, a interrupção de número-tipo 0 (zero) identifica a de maior prioridade, e a de número-tipo 255 a de menor prioridade.

A importância dos níveis de prioridade reside no fato de que, se uma rotina de serviço de interrupção foi iniciada, somente um dispositivo com nível de prioridade mais alto terá poder de interromper sua execução. Para as interrupções de hardware, as decisões de prioridade são complementadas através do circuito responsável pelas requisições. Ao teclado deve ser também associada uma interrupção de alta prioridade. Isto irá assegurar que seu buffer de entrada não ficará cheio, bloqueando entradas adicionais. Por outro lado, dispositivos como unidades de disquete e HD são tipicamente associados com níveis de prioridade mais baixos.

IRET: Retorna de uma rotina de tratamento de interrupção, recuperando, da pilha, o conteúdo dos registradores que foram automaticamente salvos na ocorrência da interrupção. Ao final de uma rotina de tratamento de interrupções, deve ser inserida a instrução IRET.

Formato de implementação: IRET

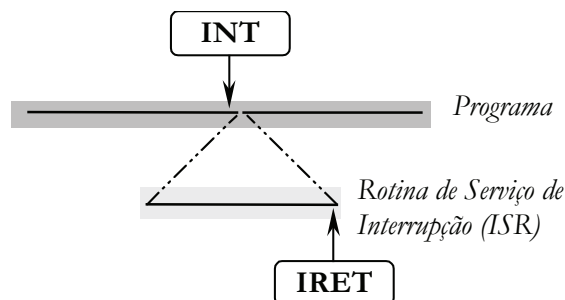


Figura 40 Interrupção por Software (INT e IRET)

## **INTERRUPÇÃO POR HARDWARE: CONTROLADOR DE INTERRUPÇÕES 8259**

Nos sistemas 8086 e 8088, os números das interrupções entre 08h e 0Fh são utilizadas para as interrupções do hardware; nos sistemas 80286 e superiores, os números das interrupções entre 08h e 0Fh (8 e 15 em decimal) e também 70h (112 em decimal) e 77h (119 em decimal) estão reservados para as interrupções de hardware.

Os chips INTEL 8259, que atuam como controladores programáveis de interrupções (programmable interrupt controller – PIC), são utilizados em todos os equipamentos da linha IBM PC na gerência das interrupções por hardware. Um chip possui 8 canais de interrupções. Os equipamentos da linha PC/XT suportam apenas um chip 8259, isto é, apenas 8 canais de interrupções, enquanto que os demais equipamentos utilizam 2 chips em cascata, ou o equivalente a 2 combinados num único chip. No caso mais simples, cada canal estará conectado a um único dispositivo. Quando um canal for ativado, será emitida uma solicitação de interrupção (interrupt request – IRQ). Estas solicitações são numeradas de IRQ0 até IRQ15.

Durante a execução de uma interrupção, podem chegar novas solicitações de para outras interrupções. Um chip controlador de interrupções mantém um controle sobre as solicitações de interrupções, e decide qual será executada em seguida, com base num esquema de prioridade. Os canais de numeração mais baixa terão as prioridades mais altas. Assim, a IRQ0 terá precedência sobre a IRQ1. No caso de 2 chips em cascata (Figura 41), um atuará como mestre e o outro como escravo, sendo que os 8 canais do chip escravo operarão através do canal 2 do chip mestre, portanto os 8 canais do escravo terão maiores prioridades após o IRQ0 e IRQ1. Veja a Tabela 6 que indica as interrupções listadas de acordo com a prioridade decrescente.

Chip	IRQ	Função
Mestre	IRQ0	Atualização da hora do dia no BIOS (timer do sistema)
	IRQ1	Teclado
	IRQ2	Conexão com o PIC escravo
Escravo	IRQ8	Relógio de tempo real
	IRQ9	Conexão com o PIC mestre (IRQ2 redirecionada)
	IRQ10	Uso geral (reservada)
	IRQ11	Uso geral (reservada)
	IRQ12	Mouse do PS/2
	IRQ13	Coprocessador matemático
	IRQ14	Controlador do disco rígido
	IRQ15	Uso geral (reservada)
Mestre	IRQ3	COM2 (porta serial)
	IRQ4	COM1 (porta serial)
	IRQ5	LPT2 (porta paralela)
	IRQ6	Controlador de discos flexíveis
	IRQ7	LPT1 (porta paralela)

Tabela 6 Tabela de IRQs e prioridades

Um chip 8259 possui basicamente 3 registradores de 1 byte cada, que controlam e monitoram as 8 linhas de interrupções por hardware. O registrador de solicitação de interrupção (interrupt request register – IRR), que passa um de seus bits para 1 para sinalizar que está ocorrendo uma solicitação de interrupção de hardware; em seguida o chip irá consultar o registrador em serviço (in service register – ISR) para verificar se uma outra interrupção se encontra em execução. Circuitos adicionais garantem que o critério de prioridades seja obedecido. Finalmente, antes de chamar a interrupção, o registrador de máscara da interrupção (interrupt mask register – IMR) será verificado para ver se uma interrupção deste nível é permitida nesse momento.

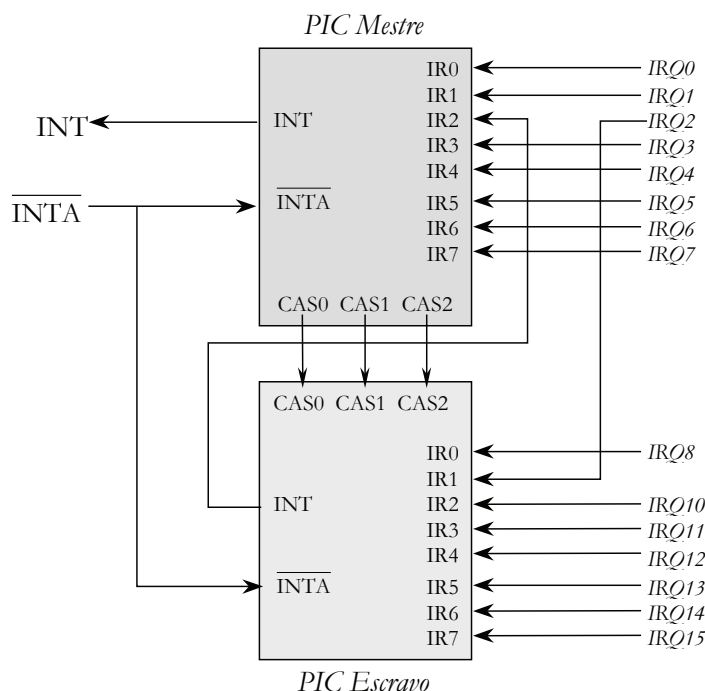


Figura 41 Cascateamento de PICs 8259

Normalmente, os programadores acessam apenas os registradores de máscara de interrupção, que podem ser lidos e gravados. O registrador IMR para o PIC mestre será acessado através do endereço de porta 21h, e através do endereço de porta 1Ah para o PIC escravo. Os registradores do PIC são acessados através das instruções IN e OUT na Linguagem Assembly.

Existe um outro registrador que deve ser acessado pelos programadores que estejam criando rotinas manipuladoras de interrupções por hardware. Trata-se do registrador de interrupção de comandos, que é utilizado para informar ao PIC que uma interrupção por hardware terminou sua tarefa e está chegando ao final. Esse registrador é acessado através das portas 20h para o PIC mestre e A0h no caso do PIC escravo.

## HABILITAÇÃO, DESATIVAÇÃO OU MASCARAMENTO DE INTERRUPÇÕES

Um flag de habilitação de interrupção está disponível no processador

identificado por IF. A possibilidade de iniciar uma interrupção de hardware na entrada INTR é habilitada com  $IF=1$  ou mascarada com  $IF=0$ . Quando  $IF=1$ , o processador cumprirá qualquer solicitação de interrupção que o registrador de máscara de interrupções permitir; quando  $IF=0$ , nenhuma interrupção por hardware poderá ocorrer. Por software, isto pode ser feito através da instrução STI (set IF) ou CLI (clear IF), respectivamente. IF afeta somente a interface de interrupção de hardware não tendo qualquer influência sobre os outros grupos de interrupções. A instrução CLI sempre deverá ser seguida por uma instrução STI, sob risco de travar o equipamento.

Durante a seqüência de inicialização de uma rotina de serviço para uma interrupção de hardware, o processador automaticamente coloca nível lógico 0 em IF ( $IF \leftarrow 0$ ). Isto mascara a ocorrência de qualquer interrupção de hardware adicional. Em algumas aplicações pode ser necessário permitir que outras interrupções de hardware com prioridade mais alta interrompam a rotina de serviço ativa. Se este é o caso, o bit IF pode ser colocado em nível lógico 1 ( $IF \leftarrow 1$ ) com uma instrução STI localizada no início da rotina de serviço.

Os programas podem desativar uma ou todas as interrupções por hardware. Essa ação é normalmente necessária apenas quando for redigido um código de baixo nível que acesse o hardware diretamente. Por exemplo, se for criada uma rotina que insira dados no buffer do teclado, a interrupção do teclado deve ser desativada durante a execução da rotina para que não exista o risco do teclado interferir no processo. As interrupções por hardware são também mascaradas de modo a impedir os atrasos durante a execução de operações sensíveis ao relógio. Uma rotina de I/O precisamente sincronizada não poderia permitir ser prejudicada por uma operação demorada no disco. Em última instância, a execução de todas as interrupções depende do ajuste existente no bit IF do registrador de flags.

Para mascarar interrupções de hardware em particular, basta enviar o padrão apropriado de bits para o IMR. Este registrador de 8 bits está localizado no endereço de porta 21h do PIC mestre, e em 1Ah para o PIC escravo. Para isso, é necessário definir os bits que correspondem

aos números das interrupções que se pretendem mascarar.

## **INTERRUPÇÕES INTERNAS E EXCEÇÕES**

Interrupções internas e exceções diferem das interrupções de hardware externo porque elas ocorrem devido ao resultado da execução de uma instrução, não de um evento que ocorre no hardware externo. Uma interrupção interna ou exceção é iniciada porque uma condição interna de erro foi detectada antes, durante ou depois da execução de uma instrução. Neste caso, uma rotina deve ser iniciada para atender a condição interna antes de prosseguir na execução da mesma ou da próxima instrução do programa. As localizações de maior prioridade foram reservadas para tratamento deste tipo de interrupção ou exceção.

As interrupções internas e exceções são categorizadas como sendo de falha, trap ou aborto. No caso de uma falha, os valores de CS e IP salvos na pilha apontam para a instrução que resultou na interrupção. Desta forma, depois de servir à exceção, pode ser re-executada. No caso de um trap, os valores de CS e IP que foram levados para a pilha apontam para a instrução seguinte à que causou a interrupção. Desta forma, depois da conclusão da rotina de serviço, a execução do programa prossegue normalmente. No caso de um aborto, nenhuma informação é reservada e, o sistema pode precisar ser reinicializado. Algumas interrupções internas e exceções no sistema genérico são:

- exceção por erro de divisão;
- exceção de depuração;
- interrupção de breakpoint;
- exceção de estouro de capacidade (overflow);
- exceção de limite (bound) excedido;
- exceção de código de operação inválido;
- exceção de falta de pilha (stack);
- exceção de overrun de segmento;
- exceção de erro do coprocessador; e
- exceção de coprocessador não disponível.



# ANEXO A - TABELA ASCII

código	caracter	código	caracter	código	caracter	código	caracter	código	caracter	código	caracter	código	caracter
000	CTRL-@	032	(BRANCO)	064	@	096	,	128	C	160	í	192	+
001	CTRL-A	033	!	065	A	097	a	129	u	161	î	193	-
002	CTRL-B	034	"	066	B	098	b	130	f	162	ô	194	-
003	CTRL-C	035	#	067	C	099	c	131	j	163	ù	195	+
004	CTRL-D	036	\$	068	D	100	d	132	k	164	ä	196	-
005	CTRL-E	037	%	069	E	101	e	133	h	165	ñ	197	+
006	CTRL-F	038	&	070	F	102	f	134	l	166	z	198	z
007	CTRL-G	039	'	071	G	103	g	135	c	167	o	199	ä
008	CTRL-H	040	(	072	H	104	h	136	e	168	í	200	+
009	CTRL-I	041	)	073	I	105	i	137	g	169	ö	201	+
010	CTRL-J	042	*	074	J	106	j	138	ä	170	~	202	-
011	CTRL-K	043	+	075	K	107	k	139	l	171	%	203	-
012	CTRL-L	044	,	076	L	108	l	140	í	172	%	204	:
013	CTRL-M	045	-	077	M	109	m	141	í	173	í	205	-
014	CTRL-N	046	.	078	N	110	n	142	Ä	174	+	206	+
015	CTRL-O	047	/	079	O	111	o	143	Ä	175	+	207	+
016	CTRL-P	048	0	080	P	112	p	144	é	176	-	208	z
017	CTRL-Q	049	1	081	Q	113	q	145	W	177	-	209	ö
018	CTRL-R	050	2	082	R	114	r	146	Æ	178	-	210	é
019	CTRL-S	051	3	083	S	115	s	147	ä	179	:	211	é
020	CTRL-T	052	4	084	T	116	t	148	ö	180	:	212	é
021	CTRL-U	053	5	085	U	117	u	149	ö	181	Ä	213	í
022	CTRL-V	054	6	086	V	118	v	150	ü	182	Ä	214	í
023	CTRL-W	055	7	087	W	119	w	151	ü	183	Ä	215	í
024	CTRL-X	056	8	088	X	120	x	152	y	184	ö	216	í
025	CTRL-Y	057	9	089	Y	121	y	153	ö	185	:	217	+
026	CTRL-Z	058	:	090	Z	122	z	154	ü	186	:	218	+
027	CTRL-[	059	;	091	[	123	{	155	g	187	+	219	-
028	CTRL-\	060	<	092	\	124		156	f	188	+	220	-
029	CTRL-]	061	=	093	]	125	}	157	ö	189	z	221	:
030	CTRL-^	062	>	094	^	126	~	158	x	190	z	222	í
031	CTRL_	063	?	095	_	127	DEL	159	f	191	+	223	-

código	caracter	código	caracter	código	caracter	código	caracter
224	+	256	.	288	+	320	-
225	-	257	.	289	z	321	:
226	ö	258	.	290	-	322	í
227	ö	259	-	291	+	254	-
228	ö	260	.	292	+	255	-
229	ö	261	.	293	+		
230	ü	262	.	294	+		
231	ä	263	.	295	+		
232	z	264	.	296	+		
233	ü	265	.	297	+		
234	ü	266	.	298	+		
235	ü	267	.	299	+		
236	y	268	.	300	+		
237	y	269	.	301	+		
238	-	270	.	302	+		
239	-	271	.	303	+		
240	z	272	.	304	+		
241	+	273	.	305	+		
242	-	274	.	306	+		
243	z	275	.	307	+		
244	z	276	.	308	+		
245	z	277	.	309	+		
246	+	278	.	310	+		
247	.	279	.	311	+		
248	z	280	.	312	+		
249	-	281	.	313	+		
250	.	282	.	314	+		
251	+	283	.	315	+		
252	z	284	.	316	+		
253	z	285	.	317	+		
254	-	286	.	318	+		
255	-	287	.	319	+		
		288	.	320	+		
		289	z	321	:		
		290	-	322	í		
		291	+	323	-		
		292	+				
		293	+				
		294	+				
		295	+				
		296	+				
		297	+				
		298	+				
		299	+				
		300	+				
		301	+				
		302	+				
		303	+				
		304	+				
		305	+				
		306	+				
		307	+				
		308	+				
		309	+				
		310	+				
		311	+				
		312	+				
		313	+				
		314	+				
		315	+				
		316	+				
		317	+				
		318	+				
		319	+				
		320	+				
		321	:				
		322	í				
		323	-				

# ANEXO B - CÓDIGO ESTENDIDO DO TECLADO

Tecla	Decimal	Hexa
Alt + A	30	1E
Alt + B	48	30
Alt + C	46	2E
Alt + D	32	20
Alt + E	18	12
Alt + F	33	21
Alt + G	34	22
Alt + H	35	23
Alt + I	23	17
Alt + J	36	24
Alt + K	37	25
Alt + L	38	26
Alt + M	50	32
Alt + N	49	31
Alt + O	24	18
Alt + P	25	19
Alt + Q	16	10
Alt + R	19	13
Alt + S	31	1F
Alt + T	20	14
Alt + U	22	16
Alt + V	47	2F
Alt + W	17	11
Alt + X	45	2D
Alt + Y	21	15
Alt + Z	44	2C
Alt + 0	129	81
Alt + 1	120	78
Alt + 2	121	79
Alt + 3	121	7A
Alt + 4	123	7B
Alt + 5	124	7C
Alt + 6	125	7D
Alt + 7	126	7E
Alt + 8	127	7F
Alt + 9	128	80

Tecla	Decimal	Hexa
F1	59	3B
F2	60	3C
F3	61	3D
F4	62	3E
F5	63	3F
F6	64	40
F7	65	41
F8	66	42
F9	67	43
F10	68	44
F11	133	85
F12	134	86
Alt + F1	104	68
Alt + F2	105	69
Alt + F3	106	6A
Alt + F4	107	6B
Alt + F5	108	6C
Alt + F6	109	6D
Alt + F7	110	6E
Alt + F8	111	6F
Alt + F9	112	70
Alt + F10	113	71
Alt + F11	139	8B
Alt + F12	140	8C
Ctrl + F1	94	5E
Ctrl + F2	95	5F
Ctrl + F3	96	60
Ctrl + F4	97	61
Ctrl + F5	98	62
Ctrl + F6	99	63
Ctrl + F7	100	64
Ctrl + F8	101	65
Ctrl + F9	102	66
Ctrl + F10	103	67
Ctrl + F11	137	89
Ctrl + F12	138	8A

<b>Tecla</b>	<b>Decimal</b>	<b>Hexa</b>
Shift + F1	84	54
Shift + F2	85	55
Shift + F3	86	56
Shift + F4	87	57
Shift + F5	88	58
Shift + F6	89	59
Shift + F7	90	5A
Shift + F8	91	5B
Shift + F9	92	5C
Shift + F10	93	5D
Shift + F11	135	87
Shift + F12	136	88
Alt + -	130	82
Alt + =	131	83
Del	83	53
End	79	4F
Home	71	47
Ins	82	52
PgDn	81	51
PgUp	73	49
↑	72	48
←	75	4B
→	77	4D
↓	80	50
Crtl + ←	115	73
Crtl + →	116	74
Crtl + End	117	75
Crtl + Home	119	77
Crtl + PgDn	118	76
Crtl + PgUp	132	84
Crtl + PrtSc	114	72
Shift+Tab	15	0F

# ANEXO C-INTERRUPÇÕES BIOS E DOS

## INTERRUPÇÕES DO BIOS

### INT 10h – Funções de Vídeo

Função 00h – Seleciona Modo de Tela

Entrada:

AL = Modo

AH = 0

#### Modos de Tela

Modo	Resolução	Máximo Páginas	Cores	Tipo	Adaptadores	Caractere	Buffer
00h	40×25	8	16	Texto	CGA, EGA, VGA	9×16	B800
01h	40×25	8	16	Texto	CGA, EGA, VGA	9×16	B800
02h	80×25	4/8	16	Texto	CGA, EGA, VGA	9×16	B800
03h	80×25	4/8	16	Texto	CGA, EGA, VGA	9×16	B800
04h	320×200	1	4	Gráfico	CGA, EGA, VGA	8×8	B800
05h	320×200	1	4	Gráfico	CGA, EGA, VGA	8×8	B800
06h	640×200	1	2	Gráfico	CGA, EGA, VGA	8×8	B800
07h	80×25	1/8	2	Texto	MDA, EGA, VGA	9×16	B800
0Dh	320×200	-	16	Gráfico	EGA, VGA	8×8	A000
0Eh	640×200	-	16	Gráfico	EGA, VGA	8×14	A000
0Fh	640×350	2	2	Gráfico	EGA, VGA	8×14	A000
10h	640×350	2	4/16	Gráfico	EGA, VGA	8×14	A000
11h	640×480	1	2	Gráfico	VGA	8×16	A000
12h	640×480	1	16	Gráfico	VGA	8×16	A000
13h	320×200	1	256	Gráfico	VGA	8×8	A000

Função 01h – Seleciona Tipo de Cursor

Entrada:

CH = Linha inicial do cursor

CL = Linha final do cursor

AH = 1

**Função 02h – Posiciona cursor****Entrada:**

DH, DL = Linha, coluna  
 BH = Número da página  
 AH = 2

**Observação:**

DH=0 e DL=0 indica o canto superior esquerdo.

**Função 03h – Encontra posição do cursor****Entrada:**

BH = Número da página  
 AH = 3

**Saída:**

DH, DL = Linha, coluna do cursor  
 CH, CL = Modo do cursor ajustado atualmente

**Função 05h – Seleciona página de exibição ativa****Entrada:**

AL = 0 – 7 (modos 0 e 1) ou 0 – 3 (modos 2 e 3)  
 AH = 5

**Função 06h – Rola página ativa para cima****Entrada:**

AL = Número de linhas em branco na parte inferior (zero limpa toda a área)  
 CH, CL = Linha, coluna superior esquerda da área a ser rolada  
 DH, DL = Linha, coluna inferior direita da área a ser rolada  
 BH = Atributo usado na linha em branco  
 AH = 6

**Função 07h – Rola página ativa para baixo****Entrada:**

AL = Número de linhas em branco na parte inferior (zero limpa toda a área)  
 CH, CL = Linha, coluna superior esquerda da área a ser rolada  
 DH, DL = Linha, coluna inferior direita da área a ser rolada

BH = Atributo usado na linha em branco  
 AH = 7

Função 08h – Lê caractere e atributo na posição do cursor

Entrada:

BH = Número da página  
 AH = 8

Saída:

AL = Caractere lido (ASCII)  
 AH = Atributo do caractere (somente texto)

Função 09h – Escreve atributo e caractere na posição do cursor

Entrada:

BH = Número da página  
 BL = Atributo (modo texto) ou Cor (modo gráfico)  
 CX = Número de caracteres a serem escritos  
 AL = Código ASCII do caractere  
 AH = 9

Função 0Ah – Escreve caractere sem atributo na posição do cursor

Entrada:

BH = Número da página  
 CX = Número de caracteres a serem escritos  
 AL = Código ASCII do caractere  
 AH = 10h ou 0Ah

### **INT 13h – Funções de Disco**

Função 00h – Reinicializa disco

Entrada:

AH = 0  
 DL = 81h (disco rígido) ou 80h (disco flexível)

Saída:

CF = 0 (sucesso; AH = 0) ou 1 (erro; AH = código de erro)

Função 01h – Lê status da operação anterior no disco

Entrada:

AH = 1

Saída:

AL = Código de erro

### Códigos de Erro de Disco

Valor	Erro
00h	Nenhum erro
01h	Comando ruim passado ao controlador
02h	Marca de endereço não encontrada
03h	Disco protegido
04h	Setor não encontrado
05h	Falha na reinicialização
07h	Parâmetros para o disco errados
09h	DMA ultrapassou fim do segmento
0Bh	Flag de trilha ruim não encontrada
10h	Verificação de setor ruim encontrada
11h	Dado é erro corrigido
20h	Falha no controlador
40h	Falha em operação de busca
80h	Nenhuma resposta do disco
BBh	Erro indefinido
FFh	Falha no sentido da operação

Função 02h – Lê setores de disco

Entrada:

AH = 2

DL = Número da unidade de disco (discos rígidos, 80h – 87h)

DH = Número do cabeçote

CH = Número do cilindro ou da trilha (discos flexíveis)

CL = Bits 7 e 6 (bits mais altos dos 10 bits para o cilindro); bits 0 a 5 (número do setor)

AL = Número de setores (discos flexíveis, 1 – 8; discos rígidos, 1 – 80h)

ES : BX = Endereço do buffer para leitura/escrita

Saída:

CF = 0 (sucesso; AL = Número de setores lidos) ou 1 (erro; AH = código de erro)

Função 03h – Escreve em setores de disco

Entrada:

AH = 3

DL = Número da unidade de disco (discos rígidos, 80h – 87h)

DH = Número do cabeçote

CH = Número do cilindro ou da trilha (discos flexíveis)

CL = Bits 7 e 6 (bits mais altos dos 10 bits para o cilindro); bits 0 a 5 (número do setor)

AL = Número de setores (discos flexíveis, 1 – 8; discos rígidos, 1 – 80h)

ES : BX = Endereço do buffer para leitura/escrita

Saída:

CF = 0 (sucesso; AL = Número de setores escritos) ou 1 (erro; AH = código de erro)

Função 04h – Verifica setores

Entrada:

AH = 4

DL = Número da unidade de disco (discos rígidos, 80h – 87h)

DH = Número do cabeçote

CH = Número do cilindro ou da trilha (discos flexíveis)

CL = Bits 7 e 6 (bits mais altos dos 10 bits para o cilindro); bits 0 a 5 (número do setor)

AL = Número de setores (discos flexíveis, 1 – 8; discos rígidos, 1 – 80h)

ES : BX = Endereço do buffer para leitura/escrita

Saída:

CF = 0 (sucesso; AH = 0) ou 1 (erro; AH = código de erro)

## **INT 14h – Funções da Porta Serial**

Função 00h – Inicializa porta RS232

Entrada:



AH = 0

Bits de AL:

0, 1 = Comprimento da palavra (01 = 7 bits, 11 = 8 bits)

2 = Bits de parada (0 = 1 stop bits, 1 = 2 stop bits)

3, 4 = Paridade (00 = nenhuma, 01 = ímpar, 11 = par)

5, 6, 7 = Taxa de transmissão (100 = 1200, 101 = 2400, 111 = 9600)

Função 01h – Envia caractere pela porta serial

Entrada:

AH = 1

AL = Caractere a enviar

DX = Número da porta (0 = COM1, 1 = COM2)

Saída:

Bit 7 de AH = 1 (erro) ou 0 (bits 0 a 6 contêm o status da porta serial)

Função 02h – Recebe caractere pela porta serial

Entrada:

AH = 2

Saída:

AL = Caractere recebido

AH = 0 (sucesso) ou Código de erro (status da porta serial)

Função 03h – Retorna status da porta serial

Entrada:

AH = 3

Saída:

Status da porta serial em AH e AL

#### Status da Porta Serial

Bit de AH ativo	Status	Bit de AL ativo	Status
7	Intervalo	7	Detecta sinal de linha recebido
6	Registrador shift vazio	6	Indicador de anel
5	Registrador holding vazio	5	Conjunto de dados pronto
4	Break detectado	4	Limpar para enviar

3	Erro de estrutura	3	Delta detecta Sinal de linha recebido
2	Erro de paridade	2	Saída do detector de anel
1	Erro de excesso	1	Delta conjunto de dados pronto
0	Dados preparados	0	Delta limpar para enviar

## INT 16h – Funções de Teclado

Função 00h – Lê tecla (código estendido para teclados de 83 e 84 teclas)

Entrada:

AH = 0

Saída:

AH = Código de varredura da tecla lida

AL = Código ASCII da tecla lida

Observação:

Aguarda a entrada de um caractere e o retorna em AL. Caso AL=0, o código estendido será encontrado em AH. A condição Ctrl-Break não é detectada.

Função 01h – Verifica se a tecla está pronta para ser lida

Entrada:

AH = 1

Saída:

ZF = 1 (buffer vazio) ou 0 (AH = código de varredura; AL = código ASCII)

Função 02h – Encontra status do teclado (teclas de 2 estados)

Entrada:

AH = 2

Saída:

AL = Byte de status do teclado

Observação:

Os equipamentos que utilizam teclados com 101 teclas podem utilizar também a função 12h da INT 16h, na qual o byte de status é retornado em AL, e um segundo byte é reportado em AH, informando

o status das teclas individuais <ALT> e <CTRL>.

Função 05h – Insere códigos no buffer do teclado

Entrada:

AH = 5

Caracteres ASCII:

CL = código ASCII de caractere

CH = código de varredura da tecla associada ao caractere

Teclas de código estendido:

CL = 0

CH = código estendido

Saída:

AL = 0 (sucesso) ou 1 (buffer do teclado cheio)

Observação:

Função utilizada nos equipamentos que suportam um teclado estendido de 101 teclas. Útil quando programas residentes precisam enviar códigos de controle ou dados ao software de aplicação.

Função 10h – Código estendido para teclados de 101 teclas

Entrada:

AH = 10h

Saída:

AL = código ASCII ou 0 (AH = código estendido)

Observação:

Aguarda a entrada de um caractere. Os códigos estendidos necessitam de apenas uma chamada à interrupção. A condição Ctrl-Break não será detectada.

Função 11h – Verifica o status de entrada para o teclado de 101 teclas

Entrada:

AH = 11h

Saída:

AL = FFh (buffer não vazio) ou 0 (buffer vazio)

## INT 17h – Funções de Impressora

### Função 00h – Imprime caractere

Entrada:

AH = 0

AL = Caractere a ser impresso

DX = Número da impressora (0, 1 ou 2)

Saída:

AH = 1 suspende a impressão

### Função 01h – Inicializa porta de impressora

Entrada:

AH = 1

DX = Número da impressora (0, 1 ou 2)

Saída:

AH = Status da impressora

Status da Impressora

Bit de AH ativo	Status
7	Impressora não ativa
6	Acusa recepção
5	Fim do papel
4	Selecionada
3	Erro de I/O
2	Não usado
1	Não usado
0	Tempo esgotado

### Função 02h – Lê status de impressora

Entrada:

AH = 2

DX = Número da impressora (0, 1 ou 2)

Saída:

AH = Status da impressora

## Interrupções do DOS

### **INT 21h – Funções de Teclado, Vídeo, Disco, Relógio e Memória**

Função 00h – Término de programa

Entrada:

AH = 0

Função 01h – Entrada de caractere do teclado com eco

Entrada:

AH = 1

Saída:

AL = Código ASCII da tecla ou 0

Observação:

Aguarda que um caractere seja digitado, caso nenhum seja encontrado. Gera um eco do caractere na tela, na posição atual do cursor. AL=0 quando um código estendido for interceptado. A interrupção deve ser repetida para que seja retornado o segundo byte do código em AL. Detecta a condição de Ctrl-Break. Ignora a tecla <Esc>, e interpreta normalmente uma digitação de <Tab>. <Backspace> faz com que o cursor retroceda um espaço, mas o caractere existente nessa posição não é apagado, sendo coberto pelo caractere recebido a seguir. <Enter> move o cursor para o início (CR), sem avanço de linha (LF).

Função 02h – Saída de caractere na tela

Entrada:

AH = 2

DL = Código ASCII do caractere

Função 05h – Saída de caractere na impressora

Entrada:

AH = 5

DL = Código ASCII do caractere

Função 06h – I/O na console sem eco

Entrada:

AH = 6

DL = FFh: ZF = 1 (nenhum caractere digitado) ou AL = código ASCII do caractere

DL < FFh: código ASCII presente em DL é enviado para a tela

Observação:

Retorna digitações sem espera (caso nenhuma disponível). Não verifica Ctrl-C ou Ctrl-Break.

Função 07h – Entrada não filtrada de caractere sem eco

Função 08h – Entrada de caractere sem eco

Entrada:

AH = 7 ou 8, respectivamente

Saída:

AL = Código ASCII da tecla

Observação:

Quando AL = 0, um código estendido terá sido recebido e será necessário repetir a interrupção para que o segundo byte seja retornado em AL. A função 08h detecta a condição de Ctrl-Break. Não ecoa na tela.

Função 09h – Escreve string na tela

Entrada:

DS:DX aponta para uma string terminada com o caractere \$

AH = 9

Função 0Ah – Entrada de string pelo teclado

Entrada:

DS:DX aponta para o local onde a string será armazenada (máximo 254 caracteres)

[DS:DX] = quantidade de bytes alocados para a string

AH = 0Ah

Saída:

DS:DX = string digitada

[DS:(DX+1)] = número de caracteres lidos

**Observação:**

Faz eco da entrada na tela. Primeiro byte da string deve ser inicializado com o tamanho. Segundo byte da string receberá o número de caracteres efetivamente lidos. Último caractere da string = <ENTER>. A string começará a partir do terceiro byte. Para ler uma string de n caracteres, deve-se alocar (n+3) bytes de memória e colocar o valor (n+1) no primeiro byte. Verifica Crtl-C e Crtl-Break.

**Função 0Bh – Verifica o status de entrada do teclado**

Entrada: AH = 0Bh

Saída: AL = FFh (buffer não vazio) ou 0 (buffer vazio)

**Função 0Ch – Esvazia o buffer de entrada do teclado e executa uma função**

Entrada: AH = 0Ch

AL = Número da função de teclado

Saída: Saída padrão da função selecionada

**Observação:**

Executa qualquer função de teclado do DOS, limpando primeiramente o buffer. Verifica Crtl-Break.

**Função 0Dh – Reinicializa disco**

Entrada: AH = 0Dh

**Função 0Eh – Seleciona disco**

Entrada:

AH = 0Eh

DL = Número da unidade de disco (0 = A; 1 = B; ...)

**Função 0Fh – Abre arquivo preexistente**

Entrada:

DS:DX aponta para um FCB

AH = 0Fh

Saída: AL = 0 (sucesso) ou FFh (falha)

**Função 10h – Fecha arquivo**

Entrada:

DS:DX aponta para um FCB

AH = 10h

Saída: AL = 0 (sucesso) ou FFh (falha)

**Função 11h – Procura primeiro arquivo coincidente**

Entrada:

DS:DX aponta para um FCB não aberto

AH = 11h

Saída:

AL = 0 (sucesso; DTA contém FCB para comparação) ou FFh (falha)

**Função 12h – Procura próximo arquivo coincidente**

Entrada:

DS:DX aponta para um FCB não aberto

AH = 12h

Saída:

AL = 0 (sucesso; DTA contém FCB para comparação) ou FFh (falha)

Observação:

A função 12h deve ser usada após a função 11h.

**Função 13h – Elimina arquivo**

Entrada:

DS:DX aponta para um FCB não aberto

AH = 13h

Saída:

AL = 0 (sucesso) ou FFh (falha)

**Função 14h – Leitura seqüencial**

Entrada:

DS:DX aponta para um FCB aberto

AH = 14h

Bloco atual e registro ativo em FCB

Saída:



Registro requerido colocado no DTA

Endereço do registro incrementado

AL = 0 (sucesso), 1 (fim de arquivo; nenhum dado no registro), 2 (segmento do DTA muito pequeno para registro) ou 3 (fim de arquivo; registro completado com 0)

#### Função 15h – Escrita seqüencial

Entrada:

DS:DX aponta para um FCB aberto

AH = 15h

Bloco atual e registro ativo em FCB

Saída:

Registro lido no DTA e escrito

Endereço do registro incrementado

AL = 0 (sucesso), 1 (disco cheio), 2 (segmento do DTA muito pequeno para registro)

#### Função 16h – Cria arquivo

Entrada:

DS:DX aponta para um FCB não aberto

AH = 16h

Saída:

AL = 0 (sucesso) ou FFh (falha; diretório cheio)

#### Função 17h – Renomeia arquivo

Entrada:

DS:DX aponta para um FCB modificado

AH = 17h

Saída:

AL = 0 (sucesso) ou FFh (falha)

#### Função 19h – Encontra unidade de disco ativa

Entrada:

AH = 19h

Saída:

AL = unidade de disco ativa (0 = A; 1 = B; ...)

Função 1Ah – Determina a posição do DTA (endereço de transferência de disco)

Entrada:

DS:DX aponta para o novo endereço do DTA

AH = 1Ah

Observação:

O DTA padrão tem 128 bytes e começa em CS:0080 no PSP.

Função 1Bh – Informação da FAT para a unidade de disco padrão

Entrada:

AH = 1Bh

Saída:

DS:BX aponta para o byte da FAT

DX = número de clusters

AL = Número de setores/cluster

CX = Tamanho de um setor (512 bytes)

Função 1Ch – Informação da FAT para a unidade de disco especificada

Entrada:

AH = 1Ch

DL = Número da unidade (0 = padrão, 1 = A, 2 = B, ...)

Saída:

DS:BX aponta para o byte da FAT

DX = número de clusters

AL = Número de setores/cluster

CX = Tamanho de um setor (512 bytes)

Função 21h – Leitura direta

Entrada:

DS:BX aponta para um FCB aberto

Campo de registro direto do FCB = DS:(DX+33) e DS:(DX+35)

AH = 21h

Saída:

AL = 0 (sucesso), 1 (fim de arquivo; nenhum dado no registro), 2 (segmento do DTA muito pequeno para registro) ou 3 (fim de arquivo; registro completado com 0)

**Função 22h – Escrita direta****Entrada:**

DS:BX aponta para um FCB aberto

Campo de registro direto do FCB = DS:(DX+33) e DS:(DX+35)

AH = 22h

**Saída:**

AL = 0 (sucesso), 1 (disco cheio), 2 (segmento do DTA muito pequeno para registro)

**Função 23h – Tamanho de arquivo****Entrada:**

DS:BX aponta para um FCB não aberto

AH = 23h

**Saída:**

AL = 0 (sucesso) ou FFh (nenhum arquivo encontrado no FCB correspondente)

Campo de registro direto = comprimento do arquivo em registro (arredondado para mais)

**Função 24h – Determina campo de registro direto****Entrada:**

DS:BX aponta para um FCB aberto

AH = 24h

**Saída:**

Campo de registro direto = registro atual e bloco ativo

**Função 27h – Lê bloco de arquivo direto****Entrada:**

DS:BX aponta para um FCB aberto

Campo de registro direto do FCB = DS:(DX+33) e DS:(DX+35)

AH = 27h

**Saída:**

AL = 0 (sucesso), 1 (fim de arquivo; nenhum dado no registro), 2 (segmento do DTA muito pequeno para registro) ou 3 (fim de arquivo; registro completado com 0)

CX = Número de registros lidos

Arquivos de registro direto fixados para acesso ao próximo registro

**Observação:**

O buffer de dados usado nos serviços do FCB é o DTA ou a área de transferência do disco.

**Função 28h – Escreve em bloco de arquivo direto****Entrada:**

DS:BX aponta para um FCB aberto

Campo de registro direto do FCB = DS:(DX+33) e DS:(DX+35)

AH = 27h

**Saída:**

AL = 0 (sucesso), 1 (disco cheio), 2 (segmento do DTA muito pequeno para registro)

CX = 0 (fixa o arquivo no tamanho indicado pelo campo de registro direto)

Arquivos de registro direto fixados para acesso ao próximo registro

**Observação:**

O buffer de dados usado nos serviços do FCB é o DTA.

**Função 2Ah – Obtém data****Entrada:** AH = 2Ah**Saída:**

CX = (Ano - 1980)

DH = Mês (1 = janeiro, ...)

DL = Dia do mês

**Função 2Bh – Ajusta data****Entrada:**

CX = (Ano - 1980)

DH = Mês (1 = janeiro, ...)

DL = Dia do mês

AH = 2Bh

**Saída:**

AL = 0 (sucesso) ou FFh (falha; data não válida)

**Função 2Ch – Obtém hora**

Entrada:

AH = 2Ch

Saída:

CH = Horas (0 – 23)

CL = Minutos (0 – 59)

DH = Segundos (0 – 59)

DL = Centésimos de segundo (0 – 99)

Função 2Dh – Ajusta hora

Entrada:

AH = 2Dh

CH = Horas (0 – 23)

CL = Minutos (0 – 59)

DH = Segundos (0 – 59)

DL = Centésimos de segundo (0 – 99)

Saída:

AL = 0 (sucesso) ou FFh (falha; hora inválida)

Função 2Fh – Obtém DTA corrente

Entrada:

AH = 2Fh

Saída:

ES:BX = Endereço do DTA corrente

Observação:

O buffer de dados usado nos serviços do FCB é o DTA ou a área de transferência de disco.

Função 31h – Termina processo e permanece residente

Entrada:

AH = 31h

AL = Código de saída binário

DX = Tamanho da memória requerida em parágrafos

Observação:

O código de saída pode ser lido por um programa principal com a função 4Dh.

**Função 36h – Obtém espaço de disco livre****Entrada:**

AH = 36h

DL = Número da unidade de disco (0 = padrão, 1 = A, 2 = B, ...)

**Saída:**AX = FFFFh (número de unidade inválido) ou Número de setores/  
cluster

BX = Número de clusters disponíveis

CX = Tamanho de um setor (512 bytes)

DX = Número de clusters

**Função 39h – Cria um subdiretório em disco****Entrada:**

AH = 39h

DS:DX aponta para string ASCIIZ (terminada com zero) com o nome  
do diretório**Saída:**CF = 0 (sucesso) ou 1 (erro; AH = 3 – caminho não encontrado,  
5 – acesso negado)**Função 3Ah – Apaga um subdiretório em disco****Entrada:**

AH = 3Ah

DS:DX aponta para uma string ASCIIZ com o nome do diretório

**Saída:**CF = 0 (sucesso) ou 1 (erro; AH = 3 – caminho não encontrado,  
5 – acesso negado ou subdiretório não vazio)**Função 3Bh – Muda o diretório corrente****Entrada:**

AH = 3Bh

DS:DX aponta para uma string ASCIIZ com o nome do diretório

**Saída:**

CF = 0 (sucesso) ou 1 (erro; AH = 3 – caminho não encontrado)

Função 3Ch – Cria um arquivo no disco ou abre um existente

Entrada:

AH = 3Ch

DS:DX aponta para uma string ASCIIZ com o nome do diretório

CX = Atributos de arquivo (bits podem ser combinados)

Saída:

CF = 0 (sucesso; AX = descritor de arquivo) ou 1 (erro; AL = 3 – caminho não encontrado, 4 – muitos arquivos abertos, 5 – diretório cheio ou existe arquivo anterior somente para leitura)

Observação:

Ao tentar criar um arquivo já existente, o conteúdo anterior do arquivo perdido.

#### Atributos de Arquivo

Bitativado	Significado
0	Somente para leitura
1	Oculto
2	Sistema
3	Rótulo de volume
4	Reservado (0)
5	Arquivo
6 – 15	Reservados (0)

Função 3Dh – Abre um arquivo

Entrada:

AH = 3Dh

DS:DX aponta para uma string ASCIIZ com o nome do diretório

AL define modo de operação = 0 (somente leitura), 1 (somente escrita), 2 (leitura e escrita)

Saída:

CF = 0 (sucesso; AX = handler) ou 1 (erro; AL = código do erro)

Observação:

Função mais adequada para abrir arquivos já existentes. O conteúdo anterior não é perdido.

**Função 3Eh – Fecha um arquivo****Entrada:**

AH = 3Eh

BX = Número do descritor do arquivo (handler)

**Saída:**

CF = 0 (sucesso) ou 1 (erro; AL = 6 – descritor inválido)

**Função 3Fh – Lê de arquivo ou dispositivo****Entrada:**

AH = 3Fh

BX = Número do handler associado (descritor de arquivo)

CX = Quantidade de bytes a ler

DS:DX = Endereço do buffer que vai receber o dado

**Saída:**

CF = 0 (sucesso; AX= no bytes lidos), 1 (erro; AX=5–acesso negado, 6–descritor inválido)

**Função 40h – Escreve em arquivo ou dispositivo****Entrada:**

AH = 40h

BX = Número do handler associado (descritor de arquivo)

CX = Quantidade de bytes a escrever

DS:DX = Endereço do buffer de dados

**Saída:**

CF = 0 (sucesso; AX = número de bytes lidos), 1 (erro; AX = 5 ou 6)

**Observação:**

Disco cheio não é considerado erro. Deve-se comparar o número de bytes a serem escritos (CX) com o número de bytes realmente escritos (AX).

**Função 41h – Apaga um arquivo em disco****Entrada:**

AH = 41h

DS:DX aponta para uma string ASCIIZ com o nome do diretório



Saída:

CF = 0 (sucesso) ou 1 (erro; AX = 2 – arquivo não encontrado, 5 – acesso negado)

Observação:

O arquivo a ser apagado não poderá estar em uso ou deverá estar fechado. Curingas (\* ou ?) não são permitidos no nome do arquivo.

Função 42h – Move ponteiro de leitura/escrita

Entrada:

AH = 42h

BX = Número do handler (descriptor) associado ao arquivo

CX:DX = Deslocamento desejado

AL define o código de referência = 0 (a partir do início do arquivo), 1 (a partir da posição atual), 2 (a partir do fim do arquivo).

Saída:

CF = 0 (sucesso; DX, AX = nova posição do ponteiro) ou 1 (erro; AL = 1 – número de função inválido, 6 – descriptor inválido)

Observação: Altera a posição lógica de escrita e leitura, permitindo o acesso aleatório dentro do arquivo.

Função 43h – Obtém ou altera os atributos de um arquivo

Entrada:

AH = 43h

DS:DX aponta para uma string ASCIIZ com o nome do diretório

AL = 0 (lê os atributos e coloca em CX) ou 1 (redefine os atributos especificados em CX)

CX = Atributos de arquivo (se AL = 1)

Saída:

CF = 0 (sucesso) ou 1 (erro; AL = 2 – arquivo não encontrado, 3 – caminho não encontrado, 5 – acesso negado)

CX = Atributos de arquivo (se AL=0)

Função 45h – Duplica um descriptor de arquivo

Entrada:

AH = 45h

BX = Descritor a duplicar

Saída:

CF = 0 (sucesso; AX = novo descritor, duplicado) ou 1 (erro; AL = 4 – muitos arquivos abertos, 6 – descritor inválido)

Função 46h – Força duplicação de um descritor de arquivo

Entrada:

AH = 46h

BX = Descritor de arquivo a duplicar

CX = Segundo descritor de arquivo

Saída:

CF = 0 (sucesso; descritores referem-se ao mesmo fluxo), 1 (erro; AL=6 – descritor inválido)

Função 47h (get current directory):

Entrada:

AH = 47h

DS:SI aponta para a área de 64 bytes (buffer).

DL = Número da unidade (0 = padrão, 1 = A, 2 = B, ...)

Saída:

CF = 0 (sucesso; ASCIIZ em DS:SI), 1 (erro; AH = 15 – unidade especificada inválida)

Observação:

Devolve uma string ASCIIZ de, no máximo, 64 caracteres, contendo o nome do diretório corrente, desde a raiz. Não são incluídos a letra que identifica a unidade, os pontos (:) e a barra (\).

Função 48h – Alocação de memória

Função 49h – Desalocação de memória

Função 4Ah – Realocação de memória

Entrada:

AH = 48h, 49h ou 4Ah, respectivamente

BX = Número de seções de 16 bytes de memória (parágrafos) que devem ser alocados, liberados ou realocados

Saída:

CF = 0 (sucesso; AX:0000 = endereço de bloco de memória) ou 1 (erro; AL = 7 – blocos de controle da memória destruídos, 8 – memória insuficiente com BX = solicitação máxima permissível, 9 – endereço inválido;)

Observação:

As três funções utilizam um endereço inicial de 16 bits para o bloco de memória sobre o qual irão atuar. Esse endereço representa o segmento onde o bloco começa (o bloco sempre iniciará com deslocamento 0 no segmento).

Função 4Bh – Carrega ou executa um programa

Entrada:

AH = 4Bh

DS:DX = String ASCIIZ com unidade, nome de caminho e nome de arquivo

ES:BX = Endereço do bloco de parâmetros

AL = 0 (carrega e executa o programa) ou 3 (carrega mas cria PSP, sem executar)

Bloco de parâmetros para AL = 0: Endereço de segmento do ambiente (word); Endereço do comando a colocar em PSP+80h (dword); Endereço do FCB padrão a colocar em PSP+5Ch (dword); Endereço do 2º FCB padrão a colocar em PSP+6Ch (dword)

Bloco de parâmetros para AL = 3: Endereço de segmento para carregar arquivo (word); Fator de reposição para imagem (word)

Saída:

CF = 0 (sucesso) ou 1 (erro; AL = 1 – número de função inválido, 2 – arquivo não encontrado no disco, 5 – acesso negado, 8 – memória insuficiente para operação solicitada, 10 – ambiente inválido, 11 – formato inválido)

Função 4Ch – Saída

Entrada:

AH = 4Ch

AL = código de retorno binário

Observação:

Função utilizada para encerrar programas.

Função 4Dh – Obtém código de retorno de subprocessos

Entrada:

AH = 4Dh

Saída:

AL = Código de retorno binário do subprocesso

AH = 0 (terminou normalmente), 1 (terminou com Ctrl-Break), 2 (terminou com erro crítico de dispositivo), 3 (terminou com função 31h)

Função 56h – Renomeia arquivo

Entrada:

AH = 56h

DS:DX aponta para uma string ASCIIZ contendo o nome antigo

ES:DI aponta para uma string ASCIIZ contendo o nome novo

Saída:

CF = 0 (sucesso) ou 1 (erro; AL = 3 – caminho não encontrado, 5 – acesso negado, 17 – não é o mesmo dispositivo)

Observação:

O arquivo a ser renomeado não pode estar em uso ou deve estar fechado.

Função 57h – Obtém ou determina data e hora de um arquivo

Entrada:

AH = 57h

BX = Descritor de arquivo

AL = 0 (obtém data e hora) ou 1 (fixa hora e data; CH = hora, DX = data)

Saída:

Na obtenção de data e hora, CX = Hora e DX = Data

CF = 0 (sucesso) ou 1 (erro; AL = 1 – número de função inválido, 6 – descritor inválido)

Observação:

Hora =  $2048 \times \text{horas} + 32 \times \text{minutos} + \text{segundos} / 2$  e

Data =  $512 \times (\text{Ano} - 1980) + 32 \times \text{mês} + \text{dia}$

INT 33h – Funções de Mouse

Função 0h – Inicializa o driver do mouse

Entrada: AX = 0

Saída:

BX = número de botões (FFFFh = 2, 0000h = + de 2, 0003h = mouse Logitech)

AX = FFFFh (mouse instalado) ou 0 (mouse não instalado)

Observação:

Inicializa o driver e posiciona o cursor do mouse, ainda que oculto, no meio da tela.

Função 01h – Apresenta/ativa o cursor do mouse

Entrada:

AX = 1

Função 02h – Oculta o cursor do mouse

Entrada:

AX = 2

Função 03h – Obtém a posição do cursor

Entrada:

AH = 3

Saída:

CX = coordenada x do cursor (coluna), em mickeys

DX = coordenada y do cursor (linha), em mickeys

BX = botão pressionado (1 = esquerdo, 2 = direito, 3 = central)

Observação:

Para transformar de mickey para um valor de vídeo 80×25 válido, é necessário dividir o valor de retorno de CX ou DX por 8.

**Função 04h – Define a posição do cursor****Entrada:**

AX = 4

CX = coordenada x do cursor (coluna), em mickeys

DX = coordenada y do cursor (linha), em mickeys

**Funções 05h e 06h – Monitoram os botões do mouse****Entrada:**

AX = 5 ou 6, respectivamente

BX = botão a ser monitorado (0 = esquerda, 1 = direita, 2 = central)

**Saída:**

AX = botão pressionado (0 = esquerda, 1 = direita, 2 = central)

BX = número de vezes que o botão foi pressionado desde a última chamada

CX = coordenada x do cursor (coluna) no último clique, em mickeys

DX = coordenada y do cursor (linha) no último clique, em mickeys

**Função 07h – Confina o cursor horizontalmente****Entrada:**

AX = 7

CX = coluna esquerda (inicial)

DX = coluna direita (final)

**Função 08h – Confina o cursor verticalmente****Entrada:**

AX = 8

CX = linha superior (inicial)

DX = linha inferior (final)

**Função 0Ah – Modifica o formato do cursor para modo texto****Entrada:**

AX = 0Ah

Software: BX = 0, CX = máscara de tela, DX = máscara de cursor

Hardware: BX = 1, CX = linha inicial, DX = linha final a ser

preenchida da célula texto

Observação:

No formato por software, a máscara de tela realizará um AND com o par caractere-atributo da posição atual da tela. O resultado dessa operação realizará um XOR com a máscara do cursor.

Função 0Bh – Rastreia os movimentos do mouse

Entrada:

AX = 0Bh

Saída:

CX = deslocamento horizontal (mickeys) desde a última chamada

DX = deslocamento vertical (mickeys) desde a última chamada

Observação:

Valores positivos de deslocamento indicam sentido de baixo para cima e da esquerda para a direita. A contagem é inicializada em zero a cada nova chamada.

Função 10h – Define região obscura para o cursor

Entrada:

AX = 10h

CX, DX = coordenadas (x, y) do canto superior esquerdo

SI, DI = coordenadas (x, y) do canto inferior direito

Observação:

Uma vez que o cursor tenha entrado na área obscura, permanecerá desativado, requerendo uma reativação através da função 01h.

Função 15h – Capacidade necessária para salvar o estado do driver

Entrada:

AX = 15h

Saída

BX = tamanho do buffer necessário para armazenar o estado do driver

Função 16h – Salva o estado do driver

Entrada:

AX = 16h

ES:DX aponta para o buffer onde será armazenado o estado do driver

Observação:

Antes de utilizar a função 16h, é necessário executar a função 15h.

Função 17h – Restaura o estado do driver

Entrada:

AX = 17h

ES:DX aponta para o buffer que contém o estado do driver armazenado

Função 1Dh – Define página de vídeo para apresentação do cursor

Entrada:

AX = 1Dh

BX = número da página onde será apresentado o curso



# SISTEMA DE COMPUTAÇÃO E HARDWARE

Conceito de Sistema : Um conjunto identificável e coerente de elementos que interagem coesivamente, onde cada elemento pode ser um sistema”.

Quando chamamos alguma coisa de sistema, traçamos uma fronteira virtual (conceitual) separando esse contexto do resto do mundo.

A palavra sistema envolve, de fato, amplo espectro de idéias. Pode-se pensar em sistema solar ou no corpo humano como um sistema. Diariamente, nos deparamos com sistemas de transporte, de comunicação, biológicos, econômicos, de processamento de dados, etc.

Considera-se sistema um conjunto de elementos interdependentes, ou partes que interagem formando um todo unitário e complexo, que possui um objetivo. No entanto, é preciso distinguir sistemas fechados, como máquinas, relógio, etc., dos sistemas abertos, como os sociais e biológicos: o homem, a sociedade, a empresa e os sistemas de informação.

Os sistemas abertos envolvem a idéia de que determinadas entradas são introduzidas no sistema e que, depois de processadas, geram certas saídas.

## **SISTEMA DE INFORMAÇÃO**

Há muitas formas de se conceituar informação, dependendo do ângulo de observação e do campo de conhecimento em estudo. Do ponto de vista mais específico do Sistema de Informação, podemos partir do entendimento de que informação é o resultado do tratamento de dados. Neste caso, entende-se um dado como um item elementar de informação (um conjunto de idéias ou fatos expressos através

de letras, dígitos ou outros símbolos), que tomado isoladamente não transmite nenhum conhecimento, ou seja, não possui nenhum significado intrínseco.

Partindo-se do conceito acima, pode-se definir informação como o resultado de fatos ou idéias relevantes, ou seja, dados que foram transformados (processados) numa forma inteligível para quem os recebe com valor (utilidade) real ou aparente.

Um sistema de informações é um conjunto de componentes inter-relacionados que coleta, processa, armazena e dissemina informações (processa dados), para apoiar a tomada de decisões, a coordenação e o controle.

No contexto atual, torna-se cada vez mais imprescindível automatizar o sistema de informações. Uma empresa não informatizada, por exemplo, corre o risco de se tornar menos competitiva diante o mercado em que atua.

Em um sistema de informação automatizado, isto é, que utiliza recursos da tecnologia de informática, existem quatro componentes essenciais:

- Hardware - composto pelos equipamentos de informática utilizados pelo sistema: computadores e demais equipamentos auxiliares, como impressoras, scanner, etc.
  - Poder ser entendido como um sistema de computação (um subsistema do sistema de informação)
- Software - formado pelos diversos programas de computador que fornecem instruções específicas sobre tarefas que o hardware deve executar para gerar a informação desejada (um subsistema do sistema de informação).
- Dados - fatos ou idéias relevantes.
- Peopleware - Constituído por pessoas que realizam as tarefas necessárias para o funcionamento dos outros componentes do sistema de forma a atingirem seu objetivo (profissionais de informática) e pelas pessoas que solicitam

e utilizam as informações por ele geradas (Usuários).

## SISTEMAS DE COMPUTAÇÃO

É toda a estrutura de operação dos sistemas de computadores, abrangendo o processamento, a memória, a comunicação de dados, os dispositivos de entrada e saída, os sistemas operacionais e compiladores, enfim abrange tanto o Hardware quanto Software em um Sistema de Informação.

Funcionamento de um Sistema Genérico (sistema aberto)

Entrada, Processamento e Saída: Independente da complexidade de um sistema aberto, os elementos básicos são funcionalmente e operacionalmente os mesmos.

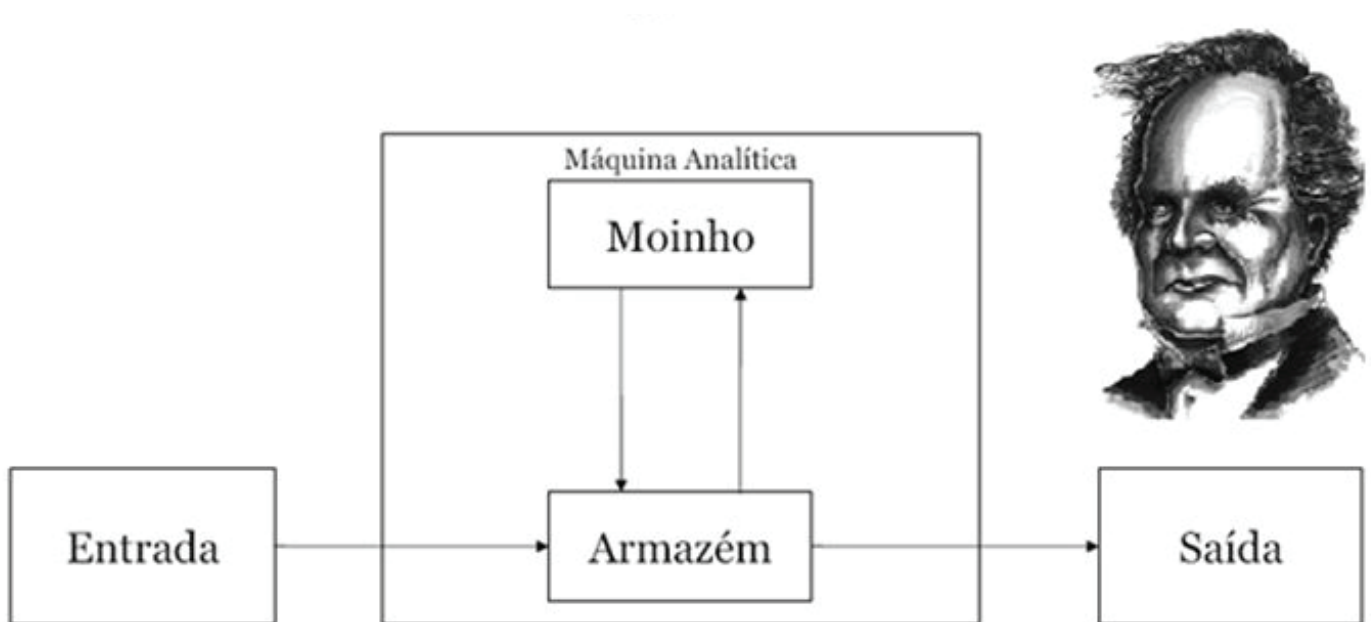
- Entrada: é o componente inicial, sobre o qual o sistema opera (matéria prima). A entrada de um sistema pode ser a saída de outro sistema.
- Processamento: atividade que transforma as entradas em saídas.
- Saída: o resultado de um processamento, o propósito para o qual o sistema foi projetado.

Qualquer sistema aberto pode ser definido em termos destes elementos, formando a Tríade de Processamento.

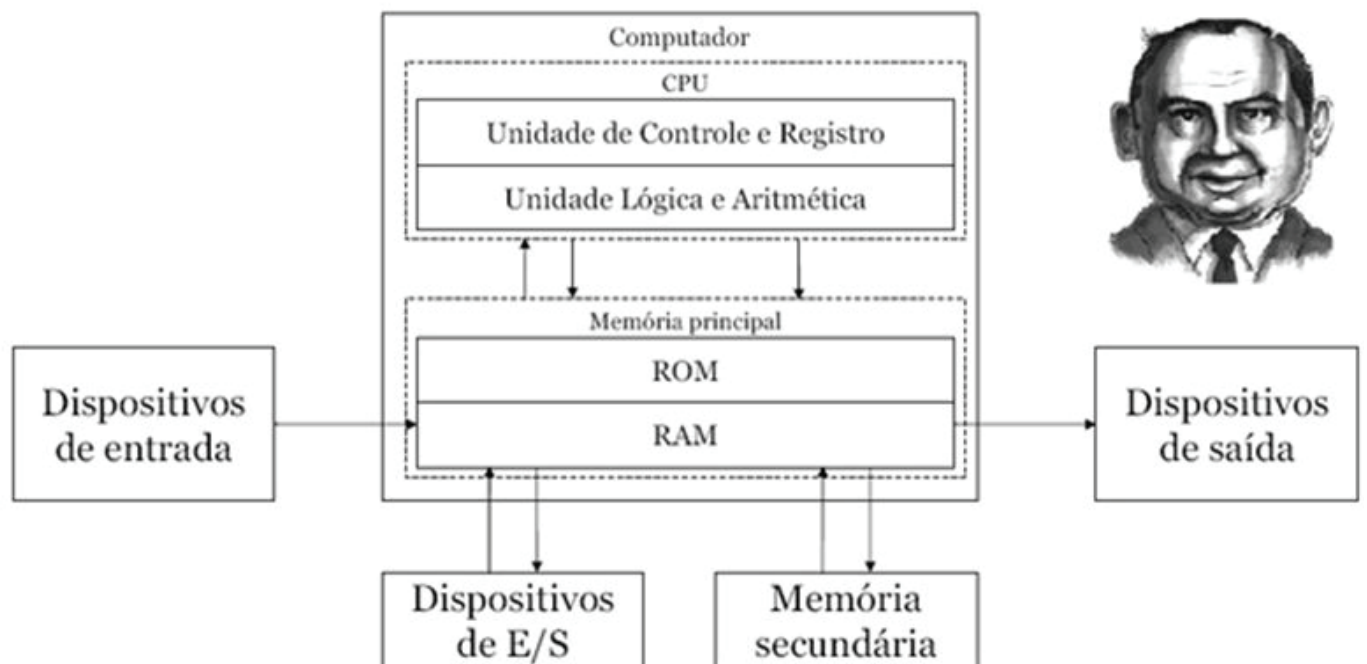


## ESQUEMA SIMPLIFICADO DE UM COMPUTADOR DIGITAL

Partindo do Modelo de Babbage...



...e chegando ao Modelo de Von Neumann



## **A FUNÇÃO DO COMPUTADOR**

A função de um computador é processar dados. Para processá-los é preciso:

- Movê-los até a unidade central de processamento,
- Armazenar resultados intermediários e finais em locais onde eles possam ser encontrados mais tarde e
- Controlar estas funções de transporte, armazenamento e
- Processamento.

Portanto, tudo que um computador faz pode ser classificado como uma destas quatro ações elementares: processar, armazenar e mover dados ou controlar estas atividades. Por mais complexas que pareçam as ações executadas por um computador, elas nada mais são que combinações destas quatro funções básicas.

A função de mover dados é executada através do fluxo da corrente elétrica ao longo de condutores que ligam os pontos de origem e destino e não depende de elementos ativos.

As funções de controle são igualmente executadas através de pulsos de corrente, ou “sinais”, propagados em condutores elétricos (estes pulsos são interpretados pelos componentes ativos, fazendo-os atuar ou não dependendo da presença ou ausência dos sinais).

Portanto estas duas funções, transporte e controle, para serem executadas só dependem da existência de condutores elétricos (fios, cabos, filetes metálicos nas placas de circuito impresso, etc.) e não exigem o concurso de componentes ativos.

## **AS FUNÇÕES DE ARMAZENAR E PROCESSAR DADOS**

Processar dados consiste basicamente em tomar decisões lógicas do tipo “faça isso em função daquilo”. Por exemplo: “compare dois valores e tome um curso de ação se o primeiro for maior, um curso diferente se ambos forem iguais ou ainda um terceiro curso se o

primeiro for menor”. Todo e qualquer processamento de dados, por mais complexo que seja, nada mais é que uma combinação de ações elementares baseadas neste tipo de tomada de decisões simples. O circuito eletrônico elementar capaz de tomar decisões é denominado “porta lógica”, ou simplesmente “porta”.

Armazenar dados consiste em manter um dado em um certo local enquanto ele for necessário, de tal forma que ele possa ser recuperado quando o sistema precisar dele. O circuito lógico elementar capaz de armazenar um dado (expresso sob a forma do elemento mínimo de informação, o “bit”, que pode exprimir apenas os valores numéricos “um” ou “zero” ou ainda os valores lógicos equivalentes, “verdadeiro” ou “falso”) é a célula de memória – um dispositivo capaz de assumir um dentre dois estados possíveis e manter-se nesse estado até que alguma ação externa venha a alterá-lo (dispositivo “bi-estável”).

Tendo isto em vista, pode-se concluir que todo computador digital, por mais complexo que seja, pode ser concebido como uma combinação de um número finito de apenas dois dispositivos básicos, portas lógicas e células de memória, interligados por condutores elétricos.

## **O HARDWARE - COMPONENTES BÁSICOS DE UM COMPUTADOR**

Numa visão externa de grande parte dos usuários, um computador é composto de CPU, teclado, vídeo e mouse. É lógico que esta é uma visão extremamente superficial até para alguns usuários.

Para um melhor detalhamento, observe:

### **A CPU - UNIDADE CENTRAL DE PROCESSAMENTO...**

Um primeiro equívoco aparece nesta divisão, ao chamar-se o gabinete do microcomputador, seja ele em formato torre (o formato vertical) ou desktop (o horizontal), de CPU. O mais correto é chamá-lo mesmo de torre ou gabinete. O nome, ou melhor, a sigla CPU, vem do inglês Central Processing Unit, para designar o componente ou o conjunto

de componentes do computador responsável pela execução das instruções dos programas. O nome CPU sugere, então, um elemento essencial no funcionamento do computador, o Microprocessador, sem ele o computador não poderia funcionar e está montado na placa principal chamada de placa-mãe (motherboard).

## **OS MICROPROCESSADORES**

Desde o início dos anos 70, os processadores ou microprocessadores, elementos de base dos computadores, são produzidos em circuitos integrados. Na maior parte dos estudos dos microprocessadores, encontra-se uma divisão clássica de sua estrutura interna considerando três grandes unidades:

- a Unidade de Controle, responsável de tarefas nobres como a interpretação das instruções de máquina a serem executadas pelo computador, a sincronização destas instruções, o atendimento a eventos de hardware, etc. Esta unidade assume toda a tarefa de controle das ações a serem realizadas pelo computador, comandando todos os demais componentes de sua arquitetura, garantindo a correta execução dos programas e a utilização dos dados corretos nas operações que as manipulam.
- a Seção de Registros, uma memória de alta velocidade, que, como o nome indica, abriga o conjunto de registros dos microprocessadores, essenciais para a realização das instruções dos programas do computador;
- a Unidade Lógica e Aritmética ou ALU (Aritmethic and Logic Unit), que assume todas as tarefas relacionadas às operações booleanas (ou, e, negação, etc...) e aritméticas (adições, subtrações, etc.) a serem realizadas no contexto de uma tarefa realizada através dos computadores;

Considerada o “cérebro” do computador, a CPU é encarregada da execução dos programas armazenados na memória central, através

do carregamento das instruções, sua decodificação e execução seqüencial.

De forma simplificada, pode-se descrever os passos que envolvem a execução de um programa, como realizada internamente pela CPU:

- Carregamento da instrução a executar para o registro de instrução;
- Modificação do valor do contador de programa para que ele aponte a palavra de memória seguinte;
- Decodificação da instrução armazenada no registro de instrução;
- Localização, em memória, dos eventuais valores utilizados pela instrução;
- Carregamento dos dados, se necessário, nos registros internos da CPU;
- Execução da instrução;
- Armazenamento dos resultados em seus destinos respectivos;
- Retorna para o Carregamento da instrução seguinte.

A esta seqüência de etapas, chamamos ciclo de Carregamento-Decodificação-Execução, que é o ciclo de base no funcionamento de todo processador.

## **MEMÓRIA**

O conceito de memória está associado a sua finalidade. Basta pensarmos na memória do ser humano. O seu objetivo fundamental é o de guardar dados para serem utilizados oportunamente.

A capacidade do microprocessador é muito limitada. Não podemos armazenar um programa inteiro dentro dele. Por essa razão os programas são transferidos de uma memória secundária, para a memória principal.



Nos computadores de uso geral, é possível encontrar diferentes denominações para as diferentes categorias de memória que neles são encontradas:

- A memória principal, ou memória de trabalho, onde normalmente ficam armazenados temporariamente os programas e dados a serem manipulados pelo processador;
- A memória secundária que permite armazenar fisicamente uma maior quantidade de dados e instruções por um período de tempo mais longo; os discos rígidos são exemplos mais imediatos de memória secundária de um computador;
- A memória cache, conceito introduzido mais recentemente, que se constitui de uma pequena porção de memória com curto tempo de resposta, normalmente integrada aos processadores (cache interno), que incrementa o desempenho durante a realização de um programa.

## **MEMÓRIA PRINCIPAL**

Ou RAM (Random Access Memory), está fisicamente localizada na placa-mãe, atualmente possui os tamanhos típicos de 16, 32 ou 64 Mb.

É na memória principal que o microprocessador acessa programas e armazena dados. Quando digitamos um texto por exemplo, é na memória principal que ele será inicialmente armazenado e depois oportunamente transferido para a memória secundária, pois, como uma simples calculadora, ela é volátil e perde tudo o que está armazenado assim que desligamos o computador.

Antigamente a memória era encaixada diretamente na placa-mãe, o que dificultava a sua manutenção. Hoje em dia os fabricantes desenvolveram módulos de memória, pequenas placas de circuito impresso que podem ser facilmente encaixadas no que chamamos Banco de Memória, facilitando tanto a substituição quanto a expansão por parte do usuário.

Há ainda um tipo de memória ligada à placa-mãe, a ROM Read Only Memory (ou Memória Somente de Leitura), que armazena dados vitais para o microcomputador. Na ROM encontram-se armazenados três programas:

- a BIOS - Basic Input Output System (Sistema Básico de Entrada e Saída) - Executa tarefas básicas tais como acessar a unidade de disco, colocar um caractere no vídeo ou ainda programar os circuitos de apoio existentes na placa-mãe.
- POST - Power On Self Test (Autoteste) - Responsável pelo autoteste (em todos os periféricos ligados ao micro) executado toda a vez que ligamos o microcomputador.
- SETUP (Configuração) - Responsável por atualizar a configuração do microcomputador. A configuração é armazenada em um chip de memória CMOS, que é alimentado diretamente por uma pequena bateria recarregável de Níquel-Cádmio, de forma que as informações nela gravadas, não se apaguem quando desligamos o microcomputador.

## MEMÓRIA SECUNDÁRIA

Ou de Massa, são unidades de armazenamento físico dos dados, podem ser internos ao gabinete ou externos, podem ainda serem magnéticos (disquete, Hard Disk, Fita, ZIP) ou ópticos (CD-ROM, CD-W).

Atualmente os disquetes possuem largura de 3 1/2 pol e capacidade de 1,44 Mb, a camada magnética dura aproximadamente 5 anos, dependendo das condições de uso e local em que é guardado, é a mídia mais barata do mercado.

O Hard Disk (ou simplesmente HD) é o principal meio de armazenamento de dados, pela velocidade de acesso e capacidade de armazenamento. Atualmente os HDs possuem capacidade de armazenamento em torno de 10 Gb, mas podendo chegar a ordem

dos Tb (Tera Bytes).

As fitas magnéticas são semelhantes às fitas de audio comuns, contudo os dados armazenados são armazenados seqüencialmente. A grande desvantagem é que caso desejarmos ter acesso a um determinado trecho da fita, teremos que avançar toda a fita até o ponto desejado. O que demanda tempo.

O CD-Rom ou Compact Disk Read Only Memory, utiliza a mesma tecnologia optico-digital dos CDs de áudio, possuem capacidade em torno de 650 Mb, e ainda podem ser regraváveis (CD-R ou CD-W).

## **MEMÓRIAS CACHE**

Muitos estudos realizados revelaram que os programas não realizam acesso à memória de maneira completamente aleatória. Se um programa contém uma instrução que efetua um acesso à memória num endereço X, é muito provável que o próximo acesso fará referência a um endereço próximo de X.

O fato de que as referências à memória são realizadas, durante um curto intervalo de tempo, sobre uma fração bem definida da memória total do sistema é o princípio (princípio da localidade) sobre o qual é baseada a existência das memórias cache. A idéia consiste em fazer com que uma palavra de memória (ou um conjunto de palavras de memória) seja(m) copiada(s) da memória principal para a cache de forma que o acesso a ela seja feito de forma muito mais rápida no momento em que ela tiver de ser acessada numa próxima vez.

## **O ACESSO DIRETO À MEMÓRIA**

O mecanismo de Acesso Direto à Memória (ou DMA, de Direct Memory Access) corresponde a uma técnica que permite o acesso a dados que estão presentes na memória do computador sem a necessidade de

intervenção da unidade central de processamento (CPU), sendo que a gestão da comunicação é assumida por um dispositivo controlador, que é composto de pelo menos quatro registros acessíveis à CPU.

O mecanismo de DMA segue os seguintes procedimentos:

- Inicialmente, o processador programa o controlador de DMA, escrevendo nos seus registros os parâmetros da comunicação, particularmente: o endereço inicial de/para onde a informação vai ser transferida; a quantidade de informação a ser transferida; o endereço do dispositivo com o qual a comunicação será realizada e, finalmente, o sentido do fluxo de informação. Esta é a fase de inicialização do controlador de DMA;
- Cabe agora ao controlador de DMA iniciar a etapa de transferência de dados para o dispositivo periférico.
- Uma vez obtida a palavra de memória, o controlador requisita ao controlador de terminal a escritura do dado considerado, concretizando assim a transferência de uma palavra dado;
- Finalmente, a continuidade da transferência da informação é assegurada pela seqüência dos seguintes procedimentos, antecipando a retomada dos dois procedimentos descritos acima: decremento do registro contador de palavras de memória; incremento do registro apontador de endereço;
- Quando o contador de palavras de memória atinge o valor zero, o controlador de DMA considera terminada a fase de transferência de dados, informando o processador através de um sinal gerando um pedido de interrupção.

Os circuitos de DMA são geralmente circuitos relativamente rápidos e possuem normalmente vários grupos de registros o que lhes permite efetuar a realização de várias transferências ao mesmo tempo.

## **MEMÓRIA VIRTUAL**

A idéia da paginação parte da distinção entre dois importantes conceitos: o espaço de endereçamento e do posicionamento em memória. Antes do aparecimento da noção de memória virtual, uma

distinção deveria então ser feita entre os endereços inferiores a 4096 (endereços úteis) e os endereços iguais ou superiores a 4096 (os endereços inúteis, uma vez que eles não correspondem a endereços reais). Existia então uma forte correspondência entre espaço de endereçamento e endereços físicos.

Porém, se uma distinção é feita entre estes dois conceitos, a noção de memória passa a ser diferente, considerando que poderemos endereçar a cada instante 4096 palavras de memória sem corresponder necessariamente aos endereços reais de 0 a 4095.

Em outras palavras, pode-se informar ao processador que estamos acessando a palavra de memória 0 quando endereçamos a posição 4096, a palavra de memória 1 quando endereçamos a posição 4097 e a palavra 4095 quando acessamos a posição 8191. A esta técnica nós chamamos de mapeamento do espaço de endereçamento sobre a memória física.

Quando um programa faz referência a um endereço de memória, seja para carregar ou armazenar um dado, carregar uma instrução, ou mesmo dar um salto, ele gera pelas técnicas clássicas de endereçamento indireto ou indexado um endereço virtual compreendido entre 0 e 65535. A determinação do endereço é feita através de consulta à tabela de páginas que permite indicar o endereço em memória principal que deve corresponder ao endereço virtual indicado.

## **COMO OS DADOS ENTRAM E SAEM? ... OS DISPOSITIVOS DE ENTRADA E SAÍDA**

Há vários dispositivos de Entrada e Saída. Alguns somente de entrada, que apenas levam os dados para o microprocessador, como teclado, mouse e escaner. Outros somente de saída, que unicamente transportam dados para o mundo exterior, como monitor e alto-falante.

Existem outros ainda que funcionam tanto como dispositivos de entrada quanto de saída, como o modem (e, dentre estes, alguns dispositivos especiais, os de armazenamento, ( fitas magnéticas, discos rígidos ou flexíveis - os disquetes, lembra ?) que além de trocar dados com a CPU e memória, armazena-os para futura utilização.

Na verdade, todo o universo dos computadores consiste apenas destes poucos elementos: microprocessador, memória, dispositivos de entrada, de saída e de armazenamento. E sua única função é processar dados trocados com o mundo exterior, através de “portas”. Portas são um dos meios usados pelos dispositivos de entrada e saída para intercambiar dados com o microprocessador. Como qualquer outra, estas portas servem para um único fim: entrar e sair.

Os circuitos que possibilitam a troca de dados com o ambiente externo ao computador (usuário, periféricos ou outros equipamentos) são conhecidas como portas de E/S (ou I/O de Input/Output).

As portas de E/S (Entrada/Saída da CPU) é o elo de ligação entre a CPU e o restante do circuito do computador. Essencialmente ela comunica a CPU com a memória principal e com os dispositivos de Entrada e Saída (do computador).

A CPU é ligada ao mundo exterior por três barramentos distintos: o de dados, o de endereços e o de controle. Os dois primeiros transportam valores correspondentes, respectivamente, a dados e endereços e o terceiro os sinais de controle.

## **BARRAMENTO DE CONTROLE**

Para a CPU executar uma operação (leitura ou escrita), são necessários três parâmetros: o endereço da posição de memória, o dado a ser lido ou escrito e um sinal de controle indicando o tipo da operação.

## **BARRAMENTO DE DADOS**

Corresponde às vias de comunicação entre a CPU e os demais elementos compondo a arquitetura de um computador. Como o nome indica, este barramento é utilizado para permitir a troca de dados bidirecionais (leitura e escrita) entre o microprocessador e os demais elementos da arquitetura do computador como memória e dispositivos de E/S.

## **BARRAMENTO DE ENDEREÇOS**

O barramento de endereços de um microprocessador é o conjunto de pinos presentes em sua interface através dos quais informa aos demais componentes da arquitetura do computador os endereços nos quais uma operação de comunicação será realizada. Por exemplo, durante um ciclo de leitura em memória, um dos passos é a colocação, neste barramento, do endereço de memória onde está armazenado o dado a ser lido.

## **A UNIDADE DE ENTRADA/SAÍDA - CANAIS**

A unidade de Entrada/Saída (E/S ou I/O, de “Input/Output”), também chamadas de Canais, tem duas funções básicas:

- transferir valores (endereços ou dados) entre os componentes externos e os internos da CPU.
- adequar os sinais de controle para transitar de dentro para fora da CPU e vice-versa, pois os componentes internos da CPU são delicados e não suportariam as tensões e correntes elétricas usadas para transportar os sinais através do barramento.

Enquanto os circuitos internos da CPU se comunicam diretamente com o barramento de controle através da unidade de E/S, a comunicação com os barramentos de dados e endereços é feita através de dois registradores específicos que fazem parte da unidade

de E/S, o Registrador de Dados da Memória (RDM) e Registrador de Endereços da Memória (REM), respectivamente.

A unidade de E/S pode ser simples, composta apenas pelos registradores REM, RDM e de um amortecedor de sinais, até unidades muito complexas, que incluem cache de memória interno.

## **O PROCESSO DE LEITURA/ESCRITA NA MEMÓRIA PRINCIPAL**

Para ilustrar o funcionamento de uma CPU, vamos examinar o procedimento adotado para efetuar a leitura ou a escrita de um dado em uma posição de memória.

Vamos examinar, inicialmente, a operação de escrita na memória.

Como a única forma da CPU se comunicar com os barramentos de dados e endereços é através dos registradores RDM e REM, os valores do dado a ser escrito e do endereço da posição da memória que irá armazenar o dado deverão ter sido previamente escritos no RDM e REM, respectivamente. Por sua vez, os circuitos internos da Unidade de Controle aplicam um sinal na linha do barramento de controle correspondente à escrita na memória (Memory Write), já que se trata de uma operação de escrita.

A operação, tipicamente, é executada em duas etapas que duram pelo menos um ciclo de máquina cada (podendo durar mais dependendo do tipo de memória e dos tempos de acesso à memória; ver adiante). Na primeira etapa, duas ações ocorrem simultaneamente. Numa delas, o circuito externo que controla a memória lê o valor contido no REM e “aponta” os circuitos que efetuarão a escrita para o endereço correspondente. Na outra, o circuito de controle da memória consulta o barramento de controle e verifica que há um sinal presente na linha de escrita de memória, preparando-se então para uma operação de escrita.

Na segunda etapa, os mesmos circuitos lerão o dado contido no RDM e o copiarão na posição de memória para a qual estão “apontados”.



No caso da operação de leitura, as coisas se passam de forma semelhante. Preliminarmente, os circuitos de controle escrevem no REM o endereço de onde o dado será lido e aplicam um sinal na linha do barramento de controle correspondente à leitura de memória (Memory Read).

Também nesse caso a operação transcorre em duas etapas. Na primeira, o circuito externo que controla a memória lê o valor contido no REM e “aponta” os circuitos que efetuarão a leitura para o endereço correspondente, enquanto ao mesmo tempo consulta o barramento de controle e encontra um sinal na linha de leitura de memória, preparando-se para uma operação de leitura.

Na segunda etapa o dado contido na posição de memória para a qual estão “apontados” os circuitos de leitura é lido e copiado no RDM.

Este mecanismo permite entender porque a CPU não consegue executar instruções que impliquem operações binárias (com dois operandos) nas quais ambos os operandos estão armazenados na memória principal (por exemplo: somar dois números, cada um armazenado em uma posição de memória principal).

A razão é que a CPU somente consegue “enxergar” o conteúdo de uma posição de memória de cada vez (aquele cujo endereço está no RDM), e para realizar uma operação binária, a CPU deve alimentar a ULA com ambos os operandos simultaneamente. Assim, é possível somar com uma única instrução, por exemplo, o valor contido no Acumulador com o de uma determinada posição de memória (cujo valor é lido e copiado no RDM), já que a ULA pode ser alimentada simultaneamente com ambos os valores, posto que estão contidos em dois registradores (Acumulador e RDM). Mas não é possível fazer o mesmo com dois valores armazenados na memória principal (é claro que esta operação pode ser viabilizada em dois passos, cada um correspondendo a uma instrução: no primeiro, um dado é lido da memória e copiado no acumulador, no segundo este valor é somado com o conteúdo da segunda posição de memória, copiado no RDM; o

que é impossível é realizar esta operação com uma única instrução em linguagem de máquina).

Podem haver diversos projetos de CPU, como consequência, conjuntos diferenciados de instruções. Mesmo as CPUs fabricadas pela mesma empresa podem ter conjuntos diferentes de instruções, porém de maneira geral os fabricantes criam “famílias” de processadores, assim, quando um novo projeto de CPU é criado, procura-se preservar a compatibilidade de suas instruções com a do projeto anterior.

Existem diversos outros termos relacionados que devem ser compreendidos:

### **PILHA (STACK)**

É a área da memória principal do computador alocada para fins de armazenamento temporário de dados. Como exemplos de dados que utilizam a Pilha temos parâmetros de subrotinas e endereços de retorno para a finalização de subrotinas.

### **INTERRUPÇÃO**

Mecanismo implementado em praticamente todos os microprocessadores para o tratamento de eventos relacionados à operação de um computador. O mecanismo implica na possibilidade de interromper a execução de uma seqüência de instruções de programa para executar uma rotina de tratamento, a partir da ativação por hardware (modificação do estado de um sinal digital num pino do microprocessador) ou por software (instrução específica de chamada).

### **PIPELING**

Mecanismo implementado em alguns microprocessadores mais

recentes, onde a execução de um conjunto de instruções de um programa, é antecipado de modo a se obter um melhor desempenho do sistema.

## DEFINIÇÕES GERAIS

**Sistemas** Qualquer coleção de elementos que trabalham juntos para executar uma tarefa.

Sistema de Informações	Lida com geração, coleção, organização, armazenagem, recordação e disseminação de conhecimento gravado.
Sistema de Computação	Subsistema de um sistema de informações. Conjunto formado por Hardware e Software.
Hardware	Conjunto de partes (físicas) de um computador
Computador	Aparelho eletrônico que pode receber uma série de instruções, ou programa, e então realizar este programa através de cálculos com dados numéricos ou compilando e correlacionando outras formas de informação.
Software	Conjunto de programas de um computador
Programador	Um indivíduo que escreve programas de computador.
Analista	Aquele que examina cuidadosamente algo, dividindo-o em partes.
Dados	Informações sobre determinado assunto.
Informação	Dado sobre alguém ou algo; notícia trazida ao conhecimento de alguém.
Analista de Sistemas	Aquele que examina e estuda detalhadamente os sistemas, dividindo-o em partes.



Rua Leonice, Qd. 160, Lt. 12, Parque Estrela Dalva II, Luziânia-GO.